BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

MULTI-USER COOPERATION IN AUGMENTED REALITY ON IOS

SPOLUPRÁCE VÍCE UŽIVATELŮ V ROZŠÍŘENÉ REALITĚ NA IOS

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **ADAM JURCZYK**
AUTOR PRÁCE

SUPERVISOR **Ing. VÍTĚZSLAV BERAN, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Jurczyk Adam**
Program: Informační technologie
Název: **Spolupráce více uživatelů v rozšířené realitě na iOS**
Multi-User Cooperation in Augmented Reality on iOS
Kategorie: Uživatelská rozhraní

Zadání:

1. Prostudujte možnosti technologie ARKit iOS. Seznamte se s vhodnými nástroji pro sdílení scény a spoupráci více uživatelů na iOS.
2. Navrhněte systém umožňující spolupráci více uživatelů v jedné scéně. Zaměřte se na interakci s virutálními objekty.
3. Implementujte navržený systém s využitím relevantních dostupných technologií. Zaměřte se na efektivitu uživatelské interakce.
4. Vyhodnoťte vlastnosti výsledného systému na základě experimentů v reálném prostředí.
5. Prezentujte klíčové vlastnosti řešení formou plakátu a krátkého videa.

Literatura:

- Dieter Schmalstieg, Tobias Hollerer. *Augmented Reality: Principles and Practice.* Addison-Wesley, 2016. ISBN: 978-0321883575.
- Russ Unger, Carolyn Chandler. *A Project Guide to UX Design: For user experience designers in the field or in the making.* New Riders, 2012. ISBN: 0132931729.
- Dále dle pokynu vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2 a částečně bod 3.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Beran Vítězslav, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 6. listopadu 2018

Abstract

The aim of this work was to create a solution that would allow multiple users to cooperate in a shared Augmented Reality environment. The proposed solution enables users to interact and edit a 3D scene in real time. Thesis touches on the subject of how a user interface should look like to assist with synchronization of devices in the shared scene. It also describes a system for user permission and action conflict resolution when dealing with virtual object interaction.

A hybrid peer-to-peer architecture is designed to facilitate communication between individual peers. The solution is based on the iOS platform and uses the new functions available in the ARKit 2 framework.

The resulting application is used to demonstrate and evaluate how the designed systems perform during multi-user cooperation. The designed solution can be used as a basis for an application that presents work from a designer to a client with the ability to get expressive user feedback in real time and in the intended physical environment.

Abstrakt

Cílem této práce bylo vytvoření řešení, které by umožňovalo spolupráci více uživatelů ve sdílené rozšířené realitě. Navržené řešení umožňuje více uživatelům úpravu virtuálních objektů a interakci s nimi ve 3D sdílené scéně v reálném čase. Práce řeší problematiku asistence při synchronizaci zařízení do sdílené scény pomocí uživatelského rozhraní. Navrhuje také systém pro řešení kolizí a povolení při interakci uživatelů s virtuálními objekty.

Součástí řešení je také návrh hybridní P2P síťové architektury pro komunikaci mezi jednotlivými zařízeními. Řešení je navrhnuto pro platformu iOS a využívá nejnovější funkce dostupné ve frameworku ARKit 2.

Výsledná aplikace demonstruje a hodnotí výkon navržených prvků při spolupráci více uživatelů. Navrhnuté řešení se dá použít jako základ pro tvorbu aplikací umožňujících návrhářům prezentování navržených řešení uživatelům v rozšířené realitě, což jim umožňuje získávání uživatelského hodnocení jejich práce v reálném čase a cílovém fyzickém prostředí.

Keywords

Augmented reality, ARKit, 3D virtual object, multi-user cooperation, hybrid P2P network architecture, Swift, user interaction, user interface design, GUI, iOS, Apple

Klíčová slova

Rozšířená realita, ARKit, 3D virtuální objekt, spolupráce více uživatelů, hybridní P2P architektura, Swift, uživatelská interakce, návrh uživatelského rozhraní, GUI, iOS, Apple

Reference

JURCZYK, Adam. *Multi-User Cooperation in Augmented Reality on iOS*. Brno, 2019. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Vítězslav Beran, Ph.D.

Rozšířený abstrakt

Cílem této práce bylo vytvoření řešení, které by umožňovalo spolupráci více uživatelů ve sdílené rozšířené realitě. Navržené řešení umožňuje více uživatelům úpravu virtuálních objektů a interakci s nimi ve 3D sdílené scéně v reálném čase. Práce řeší problematiku asistence při synchronizaci zařízení do sdílené scény pomocí uživatelského rozhraní. Navrhuje také systém pro řešení kolizí a povolení při interakci uživatelů s virtuálními objekty. Součástí řešení je také návrh hybridní P2P síťové architektury pro komunikaci mezi jednotlivými zařízeními. Řešení je navrhnuto pro platformu iOS a využívá nejnovější funkce dostupné ve frameworku ARKit 2.

Úspěšný návrh řešení, které by umožňovalo více uživatelům spolupracovat v jedné sdílené scéně, musí vyřešit určité problémy. Tyto problémy spadají do třech hlavních kategorií. První z kategorií je zajištění synchronizace a umístění všech zařízení do společné scény v rozšířené realitě. Jednotlivá zařízení musí sdílet svoje chápání fyzického prostředí. Aby toho zařízení bylo schopno, musí všechna zařízení sdílet stejný systém souřadnic ve SLAM mapě. Musí být také schopna se v této mapě zorientovat a vědět, kde přesně se v daný moment nachází. Tento proces se nazývá relokalizací.

Pro dosažení relokalizace jednotlivá zařízení porovnávají údaje získané z kamery zařízení a dalších senzorů s daty obsaženými v mapě. Tyto data jsou většinou v mapě uložena ve formě mračna bodů a klíčových snímků prostředí. Aby zařízení uspělo s relokalizací, musí být splněny dvě podmínky. První podmínkou je vytvoření dostatečně komplexní mapy prostředí, která je poskytnuta ostaním zařízením. Je tedy důležité poskytnout uživateli informace o stavu kvality mapování, aby takovou mapu byl schopen vytvořit. Druhou podmínkou pro úspěšnou relokalizaci je to, aby připojující zařízení sledovalo fyzický prostor ze stejného úhlu pohledu, z jakého byla mapa nasdílena. Pro tento účel je třeba vytvořit kvalitní uživatelské rozhraní, které uživateli pomůže s dosažením tohoto cíle. Toto je jeden z hlavních bodů řešených v této práci.

Druhou kategorií problémů je zajištění synchronizace a správy virtuálních objektů umístěných ve scéně. 3D objekty umístěné ve scéně se můžou skládat z různých částí, které můžou mít rozdílné textury či barvu. Uživatelé můžou za běhu aplikace s objekty pohybovat a měnit jejich vlastnosti. Pro zachování jednotného stavu objektů napříč zařízeními je nutné navrhnout a implementovat objekt pro správu, který tyto údaje bude uchovávat. Jelikož s jedním objektem může najednou manipulovat více uživatelů, tento systém musí umět vyřešit tuto kolizi. V této práci je to řešeno uzamknutím možnosti interakce pro ostatní uživatele v době, kdy je zrovna manipulován jiným uživatelem. Pro každý virtuální objekt musí být uchována i jeho aktuální transformace v prostoru.

Poslední kategorií řešených problémů je vytvoření vhodné síťové architektury pro sdílení informací o stavu virtuálních objektů a uživatelských akcích mezi jednotlivým zařízeními. Pro tento účel je v práci navržena hybridní peer-to-peer síť, kde jeden z klientů v síti, v tomto případě zakladatel sdílené scény, zastává také roli serveru. Tento server má za úkol sloužit jako centrální jednotka pro synchronizaci stavu objektů. V rozsahu řešení práce se jedná o správu povolení manipulace s jednotlivými objekty, a sdílení počátečního stavu scény s nově příchozími klienty. Průběžné informace o změně pozice objektů ve scéně nebo změnách si klienti posílají přímo mezi sebou. Tedy klient v sítí vždy nejprve požádá server o povolení k manipulaci s daným objektem, a po obdržení souhlasu již informace o aktualizaci transformace posílá ostatním klientům v síti na přímo.

Pro výměnu informaci byl navržen vlastní komunikační protokol, jednotlivé zprávy si mezi sebou klienti posílají ve formátu JSON. Pro přenos dat a nalezení ostatních zařízení v sítí je využíván framework MultipeerConnectivity, který je nativní po platformu iOS.

Tento framework funguje na principu inzerce služeb v síti, které jsou poté nalezeny vyhledávačem na ostatních zařízeních. Tímto způsobem si jednotliví klienti můžou vyhledat hostitele a připojit se k běžící relaci aplikace na hostitelském klientu. Samotný přenos dat může probíhat buď v lokální Wi-Fi síti, přes Wi-Fi direct nebo Bluetooth. Výběr vhodné technologie probíhá automaticky.

Výše zmíněné návrhy jsou implementované v demonstrační aplikaci. Demonstrační aplikace umožňuje uživatelům umístění modelu rytíře do prostoru pomocí rozšířené reality. K jedné relaci může být najednou připojeno více uživatelů. S modelem rytíře lze volně pohybovat, umožňuje také změnu barvy jednotlivých části výzbroje. Všechny změny jsou přenášeny v reálném čase ostatním uživatelům, tedy pokud jeden z nich posouvá rytíře, celý průběh pohybu se zobrazuje i ostatním. Kromě toho aplikace implementuje i prvky uživatelského rozhraní, které pomáhají s procesem relokalizace. Tyto prvky navádějí uživatele pomocí nápověd k úspěšné relokalizaci do scény.

Demonstrační aplikace byla k dispozici na konferenci Excel@FIT 2019. V průběhu testování byla hodnocena robustnost, výkon a přívětivost navržených systému při provozu v různých prostředích. Testování prokázalo, že navržené systémy splňují svůj účel s připomínkami k implementaci UI. Navržený systém se tedy dá použít jako základ pro aplikace, kde například návrhář prezentuje své navržené řešení uživatelům v rozšířené realitě. Tento způsob prezentace umožňuje získávání uživatelského hodnocení jejich práce v reálném čase a cílovém fyzickém prostředí.

Multi-User Cooperation in Augmented Reality on iOS

Declaration

Hereby I declare that this bachelor's thesis was prepared as an original author's work under the supervision of Ing. Vítězslav Beran, Ph.D. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

.....
Adam Jurczyk
May 16, 2019

Acknowledgements

I would like to thank my supervisor Ing. Vítězslav Beran, Ph.D. for his time and assistance provided during the creation of this thesis. I would also like to express my gratitude to my colleague Martin Minárik for his cooperation during the creation of the demonstration application, as well as to all the people involved during the testing of the implementation.

Contents

Chapter 1

Introduction

In recent years technology enabling the usage of augmented reality has become more and more widespread. It is no longer necessary to purchase specialized head mounted displays with infrared cameras. Nearly every modern phone is capable of using and displaying augmented reality. It is being used in a wide variety of environments, ranging from educational aides to games. While most of these applications used to be limited to a single user in a given session, with the release of ARKit 2 by Apple and ARCore 1.2 by Google the ability to share these experiences with others has also become widely available.

The goal of this thesis is to design and implement a solution that allows multiple users to share and cooperate in a single AR experience. The solution allows for real time interaction and editing of 3D objects in a scene. This would for example enable developers to easily present their creations to potential customers in the intended real environment. Customers would be able to see the virtual work created by the designer on their own devices and explore it in the intended physical environment at their leisure. By also being able to interact with the placed object, customers will easily be able to show the developer their feedback and possible suggestions.

The solution uses the ARKit framework as its core, and thus operates on the iOS platform. The first part of this works outlines the theoretical knowledge necessary to understand how augmented reality works, with a focus on the methods used by the ARKit framework. It then discuses the challenges faced when creating a multi-user shared experience. Since the solution works over a network, common network architectures used in applications today are described. The native MultipeerConnectivity framework used for network sharing on iOS devices is presented.

The next chapter then makes use of the theory explained in the first part and an analysis of the possible usages of this work to establish a basic set of system requirements. A proposed solution to these requirements is then presented. User interaction and UI features necessary to help user successfully synchronize the AR scene are described. Network communication model is created along with a communication protocol to facilitate the exchange of information and control in the network. A proposal for a virtual object manager is made.

The final chapter describes the process of creation of the application using XCode and Swift. Implementation of the proposed solutions is made and combined into a demonstration application capable of multi-user cooperation in shared augmented reality. The effectiveness of the implementation and the quality of user experience is then evaluated based on testing in varied environments.

Chapter 2

Shared experiences in Augmented Reality

This chapter deals with the theory behind augmented reality, how we can achieve a shared experience and the main issues we face.

In order to understand the challenges a developer faces when trying to share an experience in AR, it is first necessary to outline few concepts behind how it works. This chapter first explores how mapping and localization come together to create the experience seen on mobile devices. Next it describes objects used by the ARKit framework that will be necessary for sharing, and the issues that a shared experience presents. Finally it describes available approaches to network architecture models.

2.1 Augmented Reality explained

Augmented reality is a combination of a virtually created scene and the real environment. This combined scene is then displayed on the target device, which can be some kind of a head mounted display, or in case of this work a mobile phone. In order to create a proper AR experience, it is necessary that the device is capable of properly combining the information obtained from the real world with the virtual scene. The key aspects of this process are the tracking of the camera location, acquiring feature points from sensors and visual data, and registration of all this data into a 3D representation. With all this information the system is then able to properly display the virtual content augmented onto the physical world with regards to the device's position. The following sections will describe algorithms and techniques used not only by ARKit to achieve this result.

Simultaneous localization and mapping

When an AR application starts, it has no information about the physical world it is in. It begins collecting and processing data from device sensors, mostly from the camera and sensors like the gyroscope and accelerometer. The goal of collecting this data is to create a map of the environment and to locate the device inside the map. To accomplish this goal, ARKit uses the Visual Inertial Odometry (VIO) system. This system tracks the location of the phone within the 6 degrees of freedom - the translation in space (XYZ coordinates) and orientation in space (pitch, yaw, roll). The VIO system is a combination of two tracking models: Inertial Odometry and Visual Odometry [4].

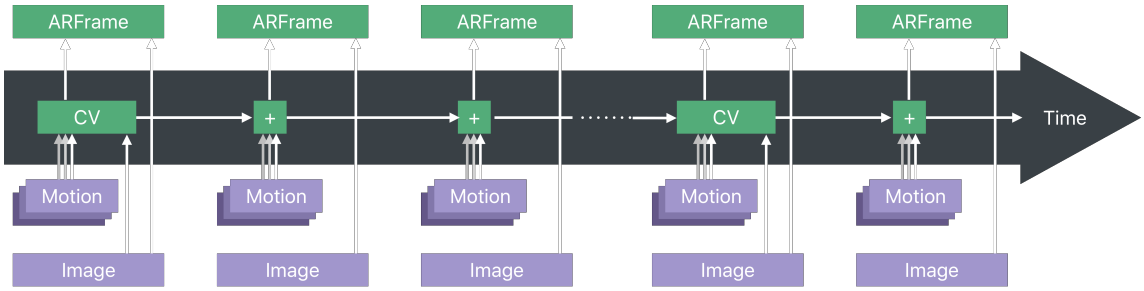


Figure 2.1: By combining the data from visual and inertial odometry, it is not necessary to always compute visual data to get camera position. Visual odometry is costly on system resources, and thus this approach saves computational power. Image source [4].

Inertial odometry uses the data acquired from the gyroscope and the accelerometer (together referred to as Inertial measurement unit) to track the orientation and movement of the device in space. IMU readings are computed around 1000 times a second [11]. While this provides great movement tracking for small periods of time or sudden movement, the data acquired from the sensors is not completely accurate. Because of the high rate of computation, even the tiniest of errors in measurement accumulate quickly, thus causing the calculated device position to deviate from reality. This makes it unsuited for tracking over long periods of time.

Visual odometry on the other hand uses frames captured from the camera. It performs measurements around the refresh rate of the camera. While this method is more accurate than simple inertial odometry, it is also more resource intensive to compute. In order to properly calculate the device position from a image, the method needs the environment to have enough visual fidelity. If the image is a simple white wall, this method does not work. It is also susceptible to fast motions, as they cause motion blur. The basic steps for visual odometry are as follows [17]:

1. Detect feature points in the first frame, for example by using the FAST algorithm.
2. Track the feature point in 2D from the previous frame, for example by using the KLT algorithm.
3. Determine the essential matrix between the current and previous frames from the feature correspondence with five point algorithm .
4. Recover the incremental camera pose from the essential matrix.
5. Triangulate 3D point locations from multiple observations of the same image feature. This is shown in figure 2.2. This process is called structure from motion (SFM).
6. Proceed to next frame.

Thanks to the triangulated 3D points, missing depth data is now calculable. It is necessary to note that if there is not enough translation between frames, the calculation will not be possible.

Using both the visual and inertial odometry allows the VIO system to compensate for cumulative errors in individual methods and for situations where their output is not ideal because of physical conditions. The output of these two systems is combined using

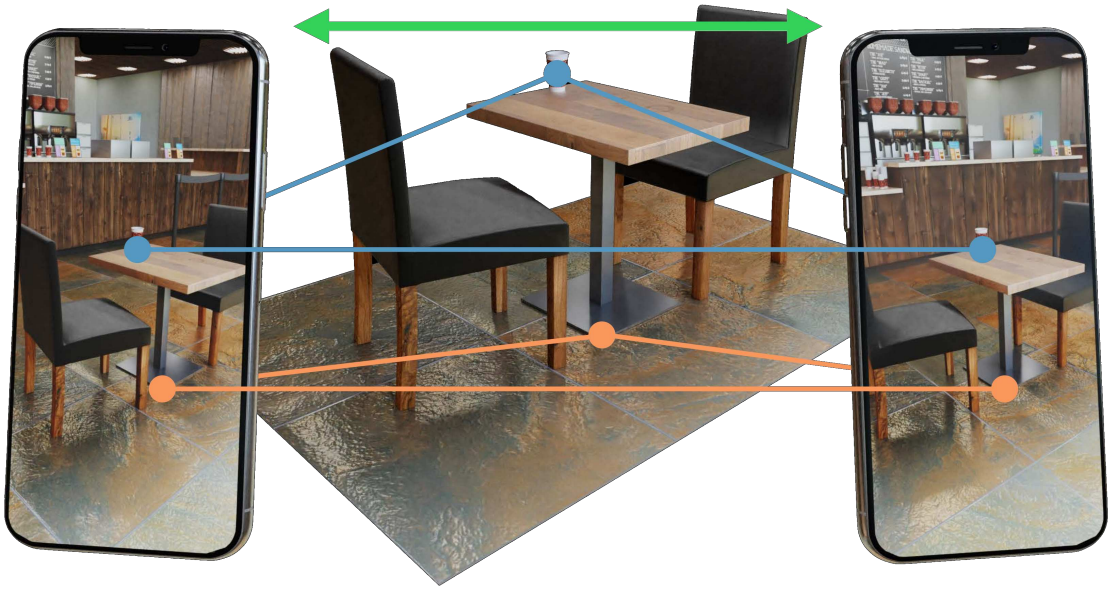
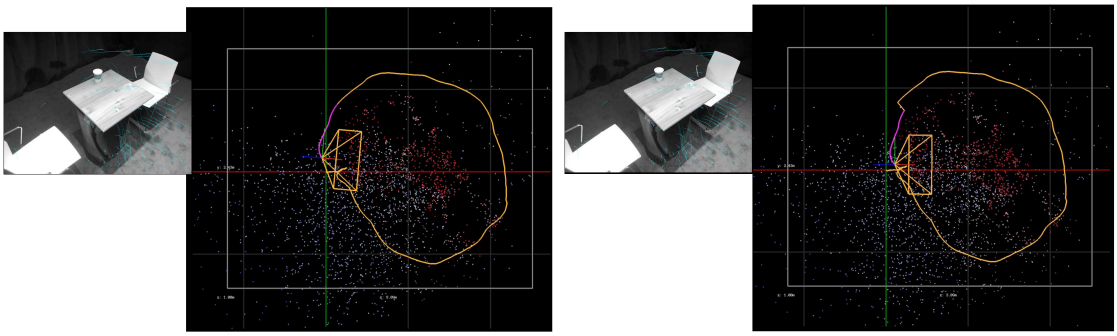


Figure 2.2: Triangulation between the same feature points in different positions. Orange and blue dots represent the feature points. Image source [4].

a *Kalman filter*, that determines which of the two systems is providing the best estimate of the current position at every given point in time [11]. An example pipeline combining both motion data and computer vision can be seen in figure 2.1.

Another important part of a proper SLAM system is the ability to align newly gathered data with earlier acquired knowledge. This process is well documented in the paper *Globally consistent range scan alignment for environment mapping* [8] by Lu and Milios. It describes how over time misalignment occurs in a map due to the accumulated pose errors. A solution to this problem is to keep previous frames and compare how the relative pose of the objects changed. This then allows us to align and update the world scans to better match reality. This update usually occurs when the framework comes back to a frame that was already explored before [4]. The way the map updates during realignment is shown in figure 2.3.



(a) Original map (b) Corrected map

Figure 2.3: Alignment correction done in a real environment. Feature points and camera position is moved according to the calculated optimization. Image source [4].

Feature detection algorithms

This section provides a short description of the popular algorithms used in order to detect and extract features as described in visual odometry. These may not be what ARKit uses in its implementation, but serve well to ilustrate the principles of detection.

FAST - features from accelerated segment test

Proposed in a paper by Rosten and Drummond in 2006 [15], the FAST algorithm has become one of the most popular feature detection algorithms. The detector is highly suitable for real-time video processing applications, thanks to its computational efficiency and high-speed performance. FAST uses a discretized circle centered on the candidate point. A point is classified as a corner, if there exists a contiguous arc of pixels with sufficient contrast to the center pixel, which covers up to three quarters of a circle. There exist multiple variations of this algorithm, named after the arc length in pixels: FAST9, FAST10, FAST11, FAST12. The algorithm using FAST 12 is shown in figure 2.4.

As an improvement to the simple high-speed test, Rosten and Drummond also proposed a machine learning approach that creates a decision tree for determining the order of pixels to be tested, with the main goal being early exit. If the machine learning approach is not used, FAST12 is commonly employed [17].

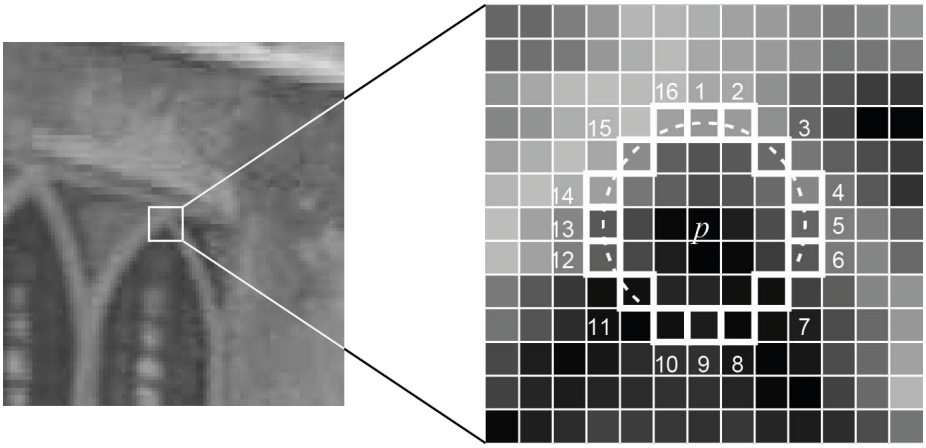


Figure 2.4: FAST12 algorithm uses a circle consisting of 16 pixels. A corner is found, if 12 contiguous pixels are all lighter or darker than the center by a threshold. The contiguous pixels are shown in the picture as the dotted arc. Rapid rejection can be determined by first testing the top (1), bottom(9), left(13) and right(5) pixels. Image source [15].

Kanade-Lucas-Tomasi tracking

The KLT feature tracker [9], [19], [7] is an approach to feature extraction. It is the classic method for incremental tracking, which extracts feature points from an initial image and then tracks them using optical flow. Optical flow are vectors that represent the translation of a particular pixel (represented by its local region) between frames.

The goal of the KLT tracking algorithm is to find parameters of a warp that transforms the template image to the input image. The warp is usually restricted to an affine transformation, which is enough to to model the deformation of the image patch that happened due to small motions. For such small, incremental motions, affine transformations

are very similar to perspective distortion effects, which would be much more expensive to compute [17].

Thanks to the algorithm, it is possible to efficiently track the motion of the feature point between multiple frames.

Five-point Algorithm for Essential Matrix

The goal of this algorithm is to determine how a camera has moved between frames from a 2D image. To achieve this, an essential matrix is computed from five point correspondences. In order for this algorithm to work, internal camera calibration must be known and the scene must be static. It is also necessary to have enough sideways or forwards movement between the two frames, if the camera is only rotated between the frames the method will fail [17].

Once an essential matrix is calculated, we can use the steps defined in Nister's algorithm [14] to calculate the rotation and translation of the camera between the two positions. With these results the new position of the camera is now known.

2.2 ARKit structures used in mapping

This section describes the most important structures that are obtained as a result of the tracking process. These structures are crucial to the final implementation of the solution.

ARWorldMap

The feature points detected during the tracking process are saved in a structure called *ARWorldMap*. All the obtained points are added to form an ARPointCloud, where each feature point is assigned a unique identifier and has its position in the map's coordinate space saved. The origin of the world map coordinate space is assigned as the position where the device was when the session was started. It is possible to update the origin point of a map if needed later. Aside from the origin point of the map, the ARPointCloud also contains information about the size of the mapped area in meters.

ARAnchor

Aside from feature points, the world map also contains anchors. Anchors are used to track the position of either real world or virtually created objects. A user can create an anchor and add it to the world map at any point in time. Every anchor contains a unique identifier and a transform of its position, scale and rotation relative to the map. An important feature of the anchors is the fact that they also undergo alignment correction when the map updates.

A special type of anchor created by the system is ARPlaneAnchor. These anchors are created when ARKit detects a real world plane in the obtained feature points. This representation contains the plane's position, geometry and size. The detected plane anchor area can increase in size automatically as more feature points are added when the device moves in the environment. This process is shown in figure 2.5. When different planes intersect, they can be merged into one singular plane. The newest versions of ARKit are capable of detecting both horizontal and vertical planes. If the device has an A12 processor or newer, the framework is also capable of classifying the detected plane (wall, floor, ceiling ...).

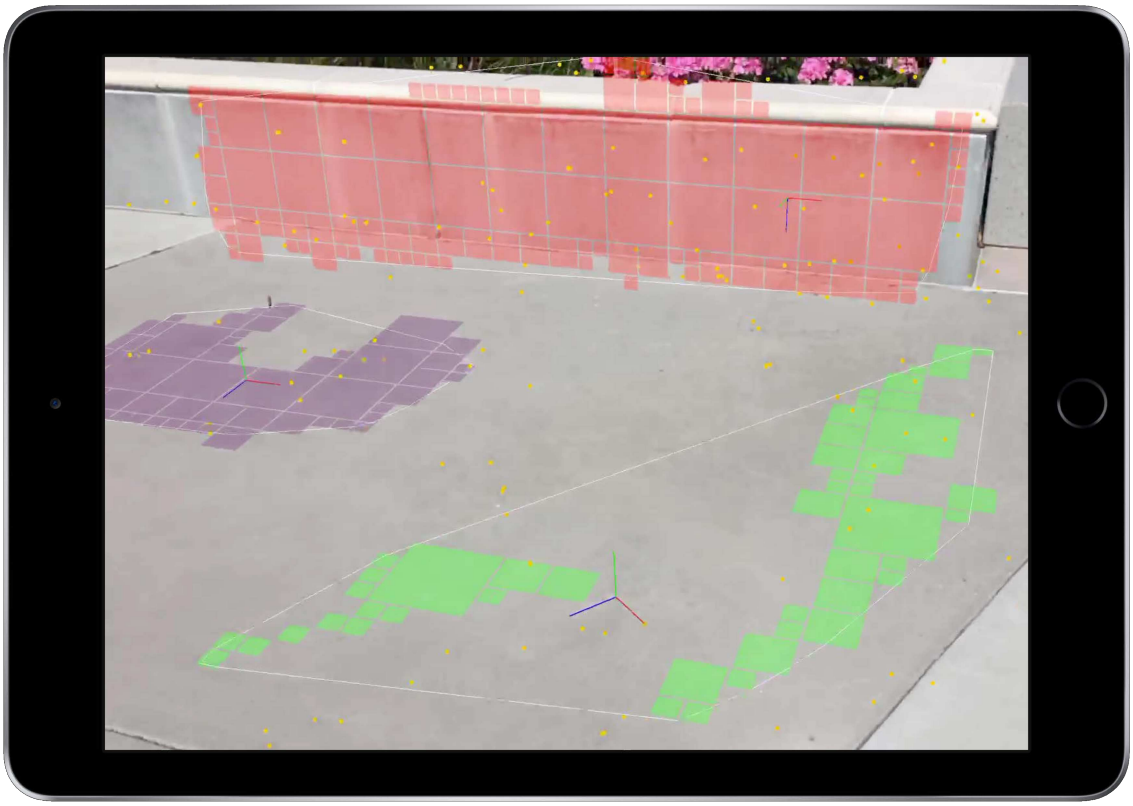


Figure 2.5: Visualisation of plane detection in progress. Areas where a plane was detected are colored. Image source [4].

2.3 Managing the AR experience

Physical world is a crucial part of a proper augmented reality. There is no meaning to the experience without it. However, a developer cannot control this aspect of the experience. Real world conditions can vary greatly and depending on the situation have a tremendous effect on the efficiency and quality of world tracking. For visual odometry to work, the images captured from the camera require a certain amount of visual fidelity. If there is not enough varied detail in the taken picture, ARKit cannot distinguish individual feature points. Lighting of the environment also plays an important role. It is thus imperative to communicate these shortcomings to the user and suggest possible solutions via a message. How these conditions can change during the session is shown in figure 2.6.

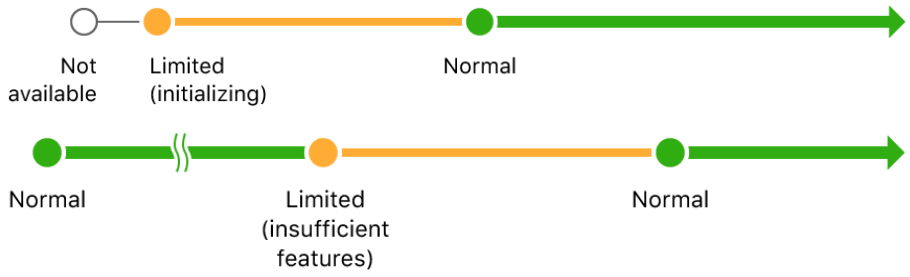


Figure 2.6: Typical lifecycle of an AR Session. Image source [2].

ARKit tracks the quality [2] of the capture in the ARCamera tracking state property. It also provides delegate methods that are called upon a change in the status. The basic states of tracking are:

- Not available - position tracking is unavailable.
- Limited - Some data is available, but the resulting device pose is uncertain. Reasons for this state are provided:
  - Initializing - the session has yet to gather enough data to provide an estimated position.
  - Relocalizing - session is in the process of relocalizing to a new world map
  - Excessive motion - device is moving too fast.
  - Insufficient features - captured environment is not textured enough, or lighting is insufficient
- Normal - everything is fine, tracking is working as intended

Aside from tracking status, the quality of the world mapping status is also monitored. The states are as follow:

- Not available - no world map is available
- Limited - tracking has not yet mapped the area around the current device position.
- Extending - recently visited areas have been sufficiently mapped, but tracking is still mapping around the current device position.
- Mapped - the visible area has been adequately mapped.

World mapping status is extremely important to the relocalization issue which will be described in the following section.

2.4 Challenges of multi-user application

This section describes the main problems that have to be solved in order to create a shared multi-user AR experience. One of the sources is the blog post by Miesnieks [12].

When creating a shared multi-user experience, or a multiplayer AR session, it is first necessary to get the devices into the same world map. As described in the previous section, once a session is started the device sets its initial recognized position as the origin point of the world coordinate system. If every device has a different world origin, sharing and cooperating with each other is impossible. Thus it is necessary for all the devices to synchronize their coordinate systems. One of the ways to achieve this is for the first device to share its SLAM map data with the rest. The other devices then adopt the coordinate origin point of the received map as the basis of their own coordinate systems and adjust their own device pose in it.

For the system to work properly, each device needs to be aware of the other devices' current pose. As every device has its own tracker, the position data needs to be broadcasted to others for every frame. This requires the devices to be connected with each other via some type of a network connection. In addition to the location data, it is necessary to share

individual user actions in real time. Since there are now multiple users interacting with the virtual objects in a scene, a system that manages individual user permissions and the state of all the shared assets needs to be created.

In an ideal situation, the 3D understanding of the world that each device has is also shared as each device explores more of their surroundings. If one device manages to detect and map new planes, the other devices should be able to add this data to their own maps.

Relocalization

The hardest part of the previously described challenges is the synchronization of device world maps. The process used to achieve this is known as relocalization. When a device receives a map of an environment, it has no knowledge of its pose in this map. In order to orient itself in this new environment, it has to compare the received data with the information coming from its tracking mechanisms. The methods used to achieve this are very closely tied to methods used in Loop Closure in the SLAM methodology, since relocalization is one of the possible ways to accomplish it. The goal of loop closure is to detect when a device enters an area mapped previously, and align the map data accordingly as described in section 2.1.

Relocalization at its core is a search issue. An image is available from the camera and the goal of relocalization is to find this image in a SLAM map. A typical SLAM map usually consists of a set of keyframes and a sparse point cloud. When an image from camera is captured, it is compared to the keyframes available in the system. Alongside this process feature points are also extracted from the image. These extracted feature points are then being compared to the sparse point cloud in the map. In order to optimize and speed up this process, data from other sensors, such as GPS or IMU, can be used [12].

Relocalization in ARKit

The process of relocalization in a shared AR experience using ARKit begins when a host sends his ARWorldMap object to the other user. ARWorldMap supports easy serialization into raw data. When the other user receives the data, the world map is deserialized and the ARSession is completely reset with the received world map set as its initial map. All of the tracking and mapping data acquired on the first device is thus now available on the other user's device. All tracking and mapping data acquired locally before the reception of the new world map is lost.

In order for the other user to be able to relocalize into this newly received world map, the tracking data needs to have a certain level of complexity and variety. The physical environment needs to have plenty of visual complexity and distinct features, otherwise the relocalization will fail. The level of quality from the current point of view can be monitored using the World tracking status described in section 2.3. Simply put, in order to guarantee maximum chances of relocalization, it is necessary for the mapping status to have been in the Mapped state at least once. The relocalizing device also needs to look at the scene from the same spot where this Mapped status was achieved. An ideal world map is shown in figure 2.7

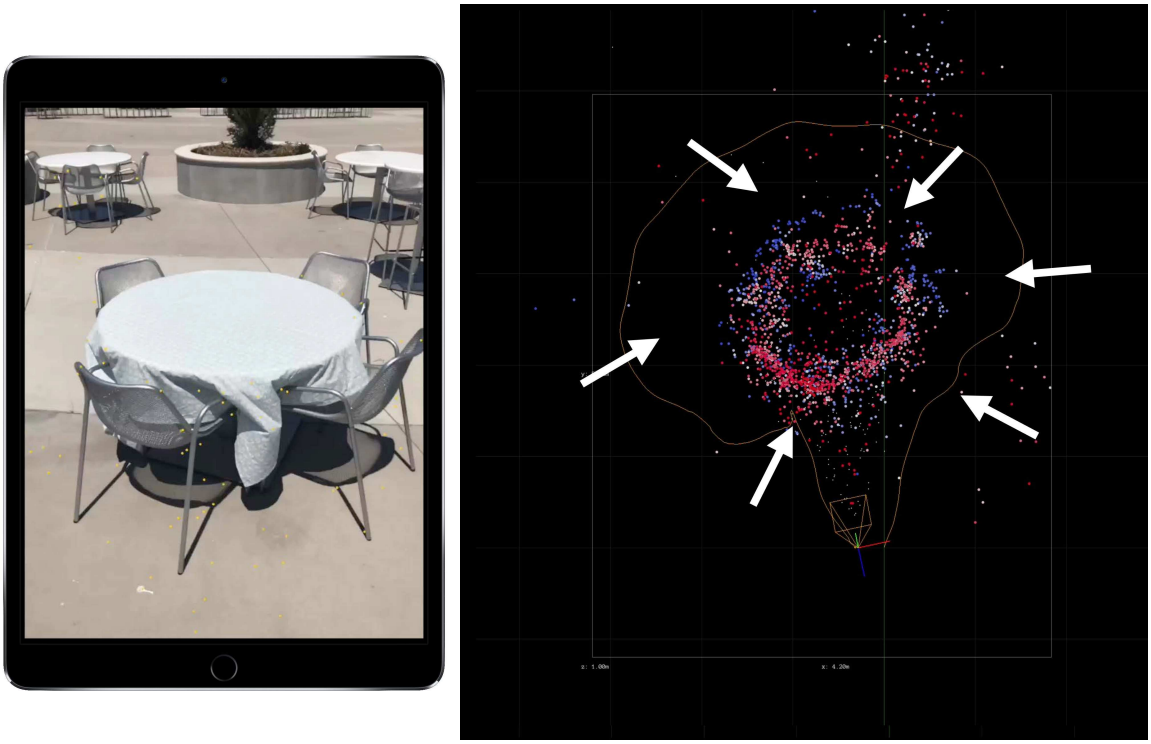


Figure 2.7: Ideal map created for localization. A static scene was mapped from multiple different angles, and a lot of feature points was acquired. Thanks to thorough mapping it is now possible for other users to relocalize from any angle. Image source [4].

2.5 Network architecture

In order to create an experience available to multiple users, it is necessary to choose a suitable network architecture. The paper by Schollmeier [18] classifies network architectures into three distinct categories:

- Client/Server architecture.
- Pure peer-to-peer architecture.
- Hybrid peer-to-peer architecture.

The following section will present the definition of each of these architectures, along with their typical use cases. Advantages and drawbacks of each approach will also be explored.

Client/Server architecture

Client/server model is the standard architecture used when communicating in a network or between processes [10]. A model of this architecture is shown in figure 2.8. A standard client/server connection consists of a client sending requests to the server. The server then processes the request and sends an answer. The server is the only provider of content and services. The client serves only to display the answers received from the server.

Communication between the processes is usually initiated by the client. Server simply runs on a loop awaiting requests from the clients. Most servers are designed to be able to

handle concurrent access, and thus are able to process requests from multiple connected clients at once.

A typical example of a client server application is an email client such as Thunderbird. The application installed on the user's computer establishes a connection with the mailing server using the SMTP protocol. The client sends requests to the server (EHLO, MAIL FROM, RCTP TO, DATA, QUIT). The server accepts the requests, performs the action associated with the request (loading of data, forwarding of mail) and responds back to the client via the SMTP protocol. The response could either be a confirmation of success or a request for more data.

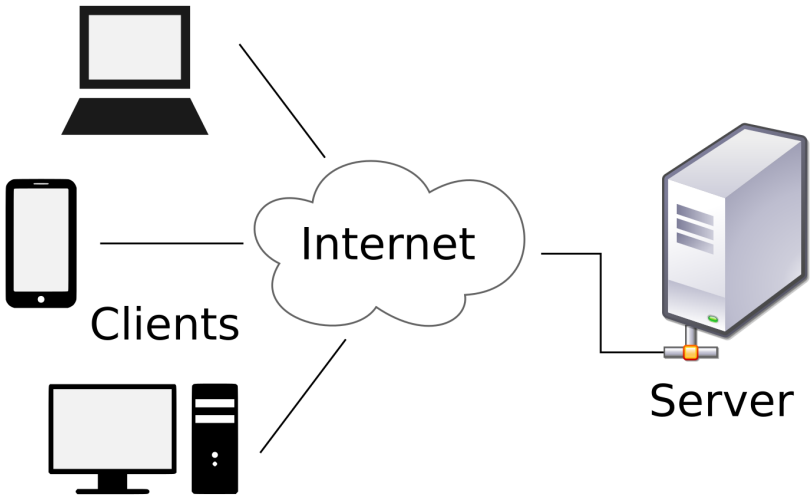


Figure 2.8: Typical client server model. Source[1]

Protocols

In order to correctly communicate between the client and server, a communication protocol needs to be established. The need for a communication protocol is common even to the peer-to-peer architectures. A protocol is a set of syntactic and semantic rules that define the exchange of information between at least two entities. The typical ways in which a protocol can be defined are:

- Finite-state machine
- Grammar - formal grammar, Structure Definition Language
- Graph models - a Petri net, MSC sequence diagram
- Process calculi

Advantages and drawbacks

Advantages of client/server approach:

[1] https://commons.wikimedia.org/wiki/File:Client-server-model.svg

1. *Centralization of services and content*: Since the server is the only point where services and processes are handled, it is easy to manage them. The server always has complete picture about all actions that are happening in the network. User permissions and access to content is easily handled in one place.

2. *Horizontal and vertical scaling*: When the load on the server becomes too great, it is easy to either move the server to a more powerful machine or add additional server machines.

3. *Data replication*: Because data is stored only on the server, there is no duplicate data stored on the clients

4. *Lightweight clients*: As everything is processed and stored on the server, the client applications are relatively small and do not need that much computational power.

Disadvantages of the approach:

1. *Congestion*: The server is the only point of exchange and processing, and the bandwidth available for incoming connections is limited. If too many requests arrive at once and overwhelm the available bandwidth, slowdown happens.

2. *Robustness*: Client/Server architecture severely lacks in system robustness. The server is the single point of failure in the architecture, if it crashes the clients are useless as nothing can be done. This can be somewhat mitigated by introducing redundant servers.

3. *Latency*: In application where the actions are very fast paced, such as multiplayer first person shooter or MMOs, the need for a server introduces latency issues. Since the actions are managed by the server, if network condition worsens, the user interaction becomes unresponsive. Commands are performed in a delayed mannner.

Pure peer-to-peer architecture

According to the definition by Schollmeier [18], Peer-to-Peer networking is defined by the fact that all the nodes in the network serve as both a client and a server. A new term „servent“ is introduced, which is a combination of the words server and client. The key aspect that distinguishes peer-to-peer network from a common distributed network architecture is the fact that every participant shares a part of their own hardware resources, be it in the form of processing power or storage capacity. These shared resources are necessary to provide the Service and content offered by the network. The resources are also accessible by all the peers without the need for passing through intermediary entities. All the peers in the network are both content providers, as well as content requestors.

In a pure peer-to-peer network there does not exist any entity that would provide some special services, any arbitrarily chosen peer can be removed from the network without suffering a loss in functionality. A model of a network without any central element can be seen in figure 2.9.

A representative of a pure peer-to-peer network is the file sharing protocol BitTorrent [5]. This protocol allows users to share files using peer bandwidth and storage. Files are distributed amongst different peers as pieces, and when a peer requests a download of a file via a tracker, he is connected to others who have the requested file. Once the user

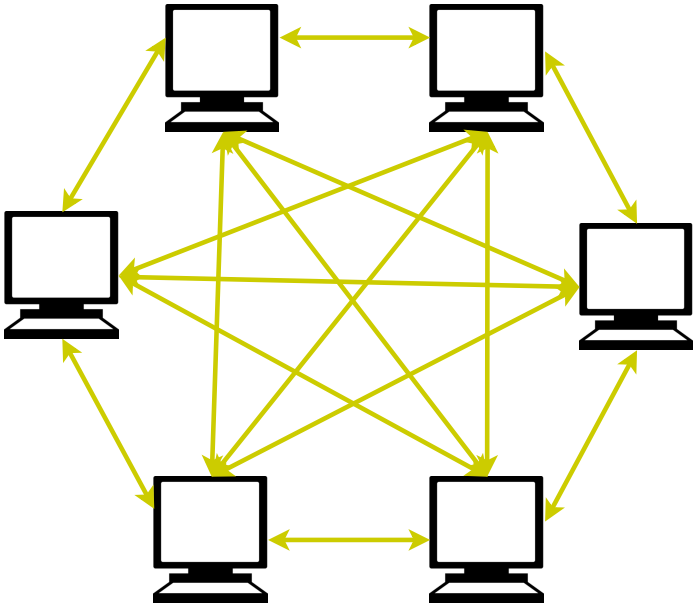


Figure 2.9: A simple model of a peer-to-peer network.

finishes downloading a piece of the file, he automatically also becomes an uploader for this particular piece. Other users can now download this piece from him.

The main advantages of a peer-to-peer network is that it is easy to setup, as no special server is necessary. Since everyone acts as a servent, there is no single point of failure. With every new user joining the network, the resources available grow instead of diminish as it happens for a server.

The key disadvantages is that the network is hard to manage from user management perspective. Security is also a concern since each device has to manage itself. The is also no option for a central backup of available data.

Hybrid peer-to-peer network

A hybrid peer-to-peer combines features from both the client/server model and the peer-to-peer architecture. This approach aims to get the best of both worlds. In a hybrid peer-to-peer network, some nodes assume additional roles similar to those seen in the client/server architecture. Hybrid peer-to-peer network can be used to support file sharing, as Spotify had been doing until 2014 [16]. Spotify used central servers as well as peer-to-peer connections from users to distribute its music streaming service. Desktop computers with Spotify were both downloading and uploading the songs to others, in a fashion very similar to the BitTorrent protocol.

In multiplayer gaming hybrid architecture can be used to lower the load on servers [6]. While authoritative actions still need be synchronized via a central server, game updates such as player movement can be distributed via peers.

2.6 MultipeerConnectivity

In order to connect mobile devices, Apple has created the MultipeerConnectivity framework. This framework operates on top of Bonjour[2] protocol. Bonjour is Apple's implementation of a zero-configuration protocol, which enables automatic discovery of devices and network services available on a local network. MultipeerConnectivity enables applications to advertise and discover services, which enables direct communication between devices. The framework automatically selects a suitable networking technology to transmit data. It can work either via a Wi-Fi network if both devices are on the same network, or via Peer-to-peer Wi-Fi. It also supports communication via Bluetooth.

The basic objects used by the framework are [3]:

- Session objects support the communication between device. A session is created when the app adds peers to the session or when it accepts an invitation from another peer. The objects maintains a set of peerID objects with peers connected to the session.
- Advertiser objects tell nearby peers that the app is willing to join a session. The object creates a network service with a unique identifier and type.
- Browser objects search the network for advertised services of a particular type.
- Peer IDs uniquely identify an app running on the device to nearby peers.

The framework is used in two phases. The first phase is a discovery phase, where it looks for peers to invite to the session. User can then choose to invite the peer, and if the peer accepts the invitation, the browser establishes a connection. The session phase thus begins. In the session phase the app can perform direct communication with one or multiple peers within the session. Delegate methods serve to inform the application when a user joins or leaves the session.

[2] https://developer.apple.com/bonjour/

Chapter 3

Proposed solution

The goal of this work is to create a solution that would allow multiple people to cooperate and interact with 3D objects in a shared augmented reality. The solution counts on all the users being present in the same physical environment. The theory of how we can achieve this shared reality was described in chapter 2. The following sections will present the possible use cases of the application, define a set of requirements to achieve a suitable experience for the described situations and outline how these requirements can be met.

3.1 Uses of a shared AR experience

The typical use case for this application would be environment where there is a need to present a created solution to a second party. A good example for this is an interior designer. With the solution presented in this paper it would be possible for a designer to visit a client at the place being designed and work with him to achieve the ideal result. The client would either be provided with a device containing a copy of the application or would install it on his own device.

An application based on this work would contain a collection of 3D models of furniture and appliances. The designer would be able to place the furniture in the room. Client would immediately see these new objects on his own device. Both of them will be able to move, rotate and change the properties of these placed objects. These changes will be visible in real time on all devices. The designer will be able to present his vision to the client, and the client would be able to freely explore it at his leisure. Since they are both sharing the same AR session, it is easy for the client to suggest his desired changes and possible improvements to the developer via his device. If the client does not like the texture or color on the placed drawer, he can easily change it. If he would like some furniture placed elsewhere, he can move it himself and show his wishes to the developer.

Thanks to them both having their own device to see this design, the user experience is much smoother. If the AR scene could only be shown on one device, the client would need to stand close the designer and look over his shoulder. If the client wanted to change something, the developer would either have to pass the device over or the client would need to describe it to the developer. This makes the entire process clumsy and uncomfortable.

In summary the solution would simplify the process of gathering feedback as well as increasing potential customer satisfaction.

3.2 System requirements

Section 2.4 outlined the main challenges of creating a shared AR experience. In short, the main requirements are:

1. Users need to interact in a shared AR scene
2. User actions need to be shared in real-time amongst all users in the session
3. Virtual 3D objects need to be managed in order to avoid conflicts and irregularities due to their shared nature

The first point can be split further into two main points: the kinds of interaction with 3D objects that this solution allows, and how to assist with the process of relocalizing users to the same shared world map. The interactions with the 3D objects must feel natural to the user. According to the Apple Human Interface Guidelines for Augmented Reality [1], direct interaction via gestures should be preferred over controls hidden in menus or buttons. Since there are multiple people interacting with the same set of objects, this fact also needs to be displayed to the user. When a user starts interacting with an object, and the interaction takes a longer time to finish, other users need to be prevented from interfering with this action. This is done in order to promote clarity and to avoid user confusion and irritation.

Since there are multiple users each with their own view of the scene, it is crucial that the state of the scene remains synchronized across all devices. There should be no discrepancies amongst individual renditions of the scenes. This means that all the objects present in the shared AR scene need to be managed in a suitable manner. In order to achieve this, a hybrid Peer-to-Peer architecture with elements from the Client/Server model based on approaches used in multiplayer games was designed. This is further described in section 3.5.

Using the specified architecture will also solve the issue of managing user permissions during interactions. Since there is now a centralized point for all the clients, it is easy to manage when and how individual users are allowed to edit the properties of displayed 3D objects.

The following sections will describe proposed solutions in detail for the issues outlined in this part.

3.3 User interaction

In order for users to truly cooperate and make use of augmented reality, some form of interaction with the virtual objects augmented onto the scene is necessary. As the device used to view the AR is an iPad or an iPhone, the input medium for these interactions is the touch screen. This allows for controls to be handled either by touch gestures around the targeted object, or by some form of a button or a slider. Since the screen space on the device is limited, great care needs to be made in deciding the number of controlling elements present in the UI. Too great of a number of icons would clutter the screen and negatively impact the user experience, since the space available to view the actual content in the scene would be smaller.

The basic interactions with 3D objects available in the system are:

- Placement of an object in the scene
- Translation

- Rotation
- Scaling
- Editing of properties - change texture, color

Placement of an object

In order to place an object, it is first necessary for the user to select one from the pre-loaded collection of 3D models in the application. Once selected, the object can be placed into the scene. For best effect and immersion, the object should appear on a real physical surface, not just randomly floating in the air. To achieve this, the utilization of the plane detection available in ARKit as shown in figure 2.5 alongside hit testing methods is suggested. When a user taps a location on the screen, a straight 3D line is created from the device camera along a direction depending on camera and device orientation. The first detected real surface that intersects this line is then used as the anchoring point for the object.

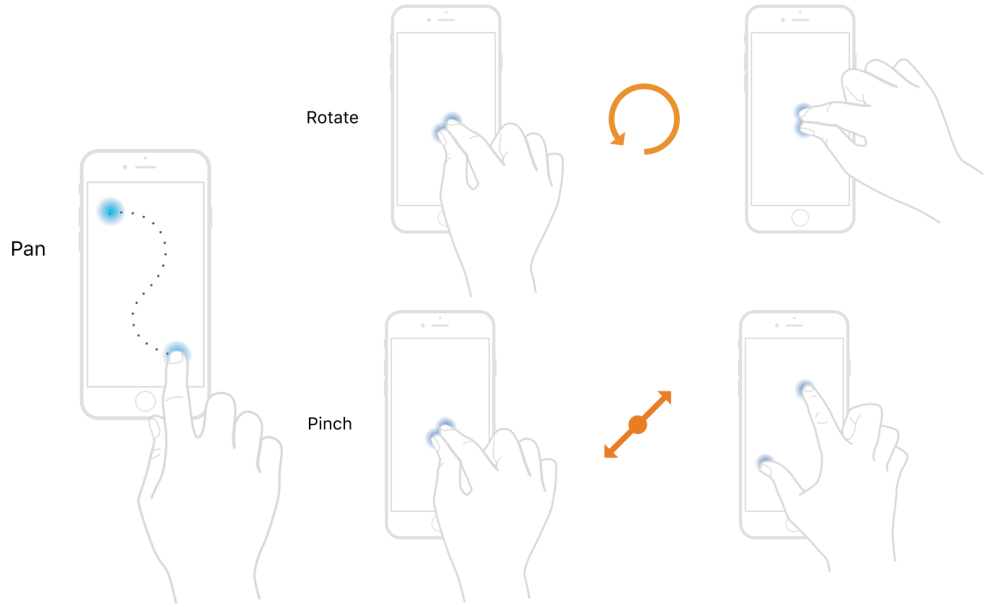


Figure 3.1: Typical gestures used on iOS touch devices. Images sourced from Apple[1].

Rotation, translation and scaling

For the implementation of object movement, rotation and scaling, the use of the classic gestures of panning, rotating and pinching is recommended. These gestures can be seen in figure 3.1. Nowadays, the vast majority of people use touch based phones or tablets. Since these gestures are used everywhere, the interaction will feel very natural and intuitive to the user.

[1] https://developer.apple.com/documentation/uikit/touches_presses_and_gestures/handling_uikit_gestures/handling_pan_gestures.
https://developer.apple.com/documentation/uikit/touches_presses_and_gestures/handling_uikit_gestures/handling_pinch_gestures.
https://developer.apple.com/documentation/uikit/touches_presses_and_gestures/handling_uikit_gestures/handling_rotation_gestures.

When a user performs these gestures near the targeted object, the desired action will happen. Since these actions are continuous and require some time to finish, it is required to lock other users from using them on this concrete object. Once the user making the gestures finishes, the object interaction will once again be available for everyone.

Aside from locking user interaction, it is also necessary to show the progress of the interaction to the other users. To achieve this, the position of the object must be sent during the progress of the gesture. The design of the communication is described in a later section.

The calculation of the new position is handled relatively to the gesture's positions. In translation, hit-testing methods similar to those in object placement are used to determine where the object should move in physical space. For rotations, the rotation of the object is changed based on the rotation of the gesture in radians. Scale is calculated relatively from the points of touch in the pinch gesture.

Editing object properties

An object can have multiple properties that can be changed. A kitchen counter can, for example, have different colored textures assigned to the body, door or counter top. To edit these properties, the use of long-press gestures is suggested. User taps and holds his finger over the object whose properties should be changed. After a while, a menu pops up displaying a list of changeable properties. User can then select which property to edit. Once the property is selected, another menu pops up with the available textures or colors.

Since the changing of properties happens instantaneously, it is not necessary to lock other users from editing the same object. However, remembering the state of every object property becomes crucial. The changes made need to be consistent for everyone, even if a user joins the session later after the object had been edited.

3.4 Assisting relocalization

A comfortable user experience is one of the pillars of a successful application. It does not matter how good an application is at its purpose, if the usability is poor and hard to understand no one will use it. As has been described in section 2.4, the biggest offender in a shared AR experience is the process of relocalization. The current version of ARKit requires the user to cooperate in the process of synchronization and to follow specific instructions. If no guidance is provided, it would be virtually impossible for a user to successfully connect to another user's session. Thus in this section the usual workflow from the point of host and the client is described. An analysis of issues that the user faces in each step is provided, as well as my suggestions for how the experience can be improved. These observations are based on Apple's Human Interface Guidelines for AR [1], as well as the results from testing how relocalization works during the development.

Host

The process of creating and joining a shared session is not simple. As has been described in section 2.4, relocalizing into other people's map is hard and very dependent on the environment. In order to ensure success, users need to be guided and sufficient information about the tracking state of the visible environment provided. The following section will outline the process from the session hosts point of view and analyze the issues from the

point of user experience. It is possible for multiple clients to be connected to a single hosted session.

Starting a session

When the host starts a session, the first thing he needs to do is to move around and map the environment in which the AR session takes place. In order to do this, information about the status of the world map and quality of the tracking environment are provided. Host can also begin placing and editing the virtual objects. So far, this process is the same as to any other AR application start. However, the issue begins once another user joins the session.

A client has joined

When a client connects to the session, he is not initially a part of the running AR session. In order for the newly connected client to begin interacting with the created AR scene, it is first necessary for the host to send him the world map of the environment. The shared map needs to be detailed enough for the connecting device to be able to relocalize. This step requires the cooperation of the host user. The host needs to send the map from a position where the world map tracking state is in the Mapped state (section 2.3). It is thus necessary to display this information in the UI, and even disable the option to send the world map if this is not achieved.

To ensure the success, it is also necessary for the receiver of the world map to look upon the scene from the position where the map was sent. This can be achieved by either directing the connecting user to stand beside the host, or by sending a photo taken during the moment when the map was sent.

Client has localized

Once the connected user's device successfully localizes in the host's world map, everyone can freely interact with and edit the virtual objects placed in the scene.

Client has left

No special actions are necessary.

Another user joins during an active session.

When another user joins, it is necessary to send him a world map for synchronization into the shared space. However, care must be taken during this process to send it only the newly joined user, so as not to disrupt the already connected ones. Thus it is necessary to take note of which user's and devices have already successfully localized into the session.

Client

This second part of the workflow process will describe and analyze the same situation from the connecting users (clients) view.

Searching for a session

When the user starts the application, he will be presented with a list of currently running AR sessions in his vicinity. Upon the selection of one of the options it is possible to join the given session. If the connection succeeds, he will join the host. If the connection fails, the user can try again or select a different session.

A session was joined

The user now needs to wait for the host to send him a world map. This fact needs to be communicated to the user clearly. Aside from waiting, there is not much for the user to do. Once a map is received from the host, the user should be prompted to move to a position close to where the map was sent. This can be achieved by showing a photo of the desired view as seen in figure 3.2 or by prompting the user to move next to the host. In order to successfully relocalize, it may be necessary for the user to move his device slightly around.

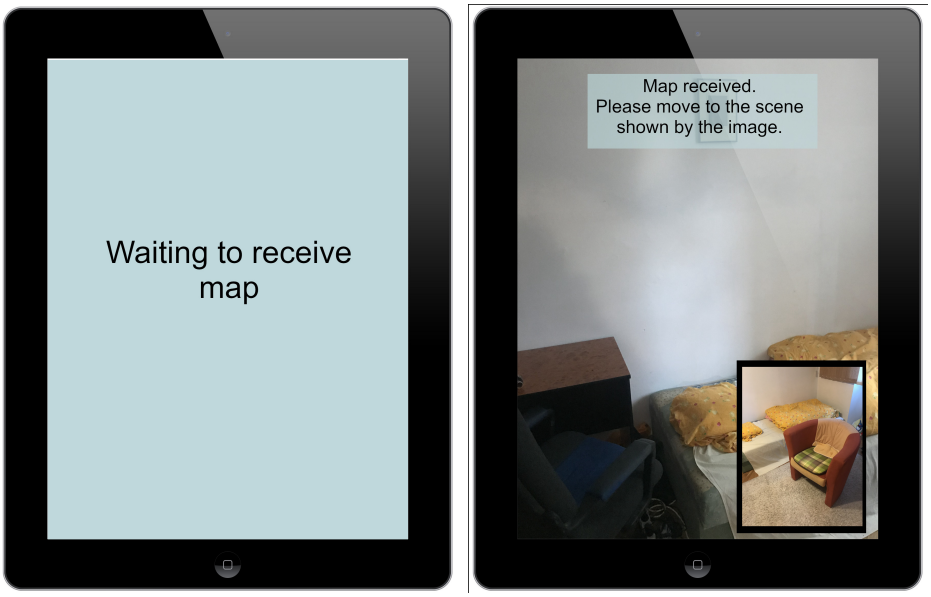


Figure 3.2: Once a map is received, the UI view will show the frames captured from camera along with an image of the scene where the map was captured.

Relocalization was successful

Once the relocalization succeeds, the user will be able to see the same scene as the host of the session. The user is now free to move around and interact with the scene. Any actions taken will be displayed to the other people in the session. Actions taken by the other users will also be visible.

Host ends the session

When the host ends the session on his end, the user will be informed about this. The user can remain in the AR session, but the experience will not be shared with anyone else. Objects will remain interactable, but no changes will be transmitted to any other users that were still present during the ending.

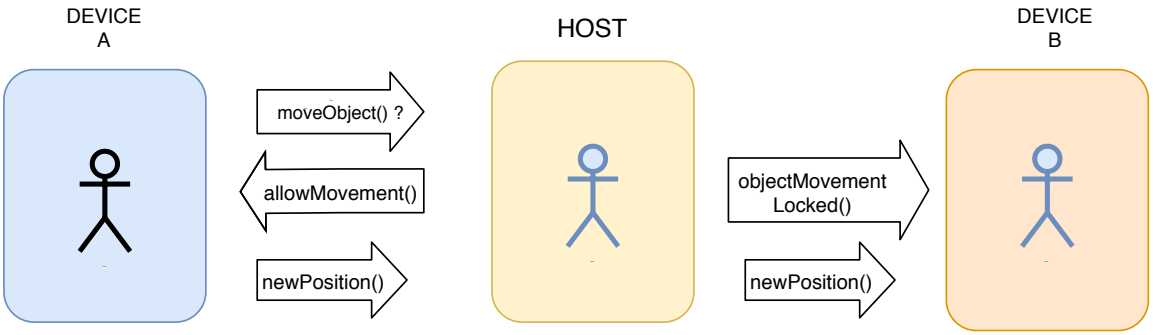


Figure 3.3: One of the first designs of how a network architecture could work based on the client/server model

3.5 Network communication

In order to properly facilitate multi-user cooperation in a shared environment, it is important to select the correct network architecture. In section 2.5, three different models were presented. In the following parts, the chosen architecture will be adapted to suit the purpose of this work. Reasons for choosing this specific architecture will also be explained. The final part of this section contains the communication protocol used between different devices in the shared session.

Choosing an architecture

Section 3.2 specified the key aspects of the proposed system. The first that matters from the point of network architecture is the ability to manage and control the state of objects present in the scene. The second aspect is the management of the permissions of different users to interact with them as time goes on. These requirements naturally lead to the selection of the Client/Server architecture, as it perfectly satisfies these conditions. Since everything is managed in one place it is easy to ensure that all object states remain synchronized. However, the pure Client/Server model approach does not suit the way AR interaction, especially the processes of translation, rotation and scaling, is shared. An initial design of the architecture can be seen in figure 3.3.

As has been stated in section 3.3, these user actions are continuous and in order to ensure the best user experience, the progress of their movement needs to be shared with the rest of the users connected to the session. This means that incremental updates of position have to be sent during the entire process as the controlling user for example moves the object with the pan gesture across the screen. If one were to use the Client/Server architecture, the whole set of these movements would have to first be sent to the server, which would only then share this information with the rest of the connected clients.

This introduces an unnecessary middle step, as the server cannot do anything to improve the quality of the position updates, since these updates are based on the tracking data from the sending peer. It only serves as a distributor of this data. Having to go through a server first in order to reach the other clients also means that the latency of updates is basically doubled at best. The increased latency then causes undesired delays in the actions displayed across the devices.

In order to avoid this unnecessary latency, it is best to avoid the server for these kinds of updates altogether. This fits well with the advantages of using the peer-to-peer network

approach, as the incremental updates can be shared directly with the other peers in the session. When taking all of the above points into account, it is best to use the key points of both of the approaches. This led to the creation of a hybrid peer-to-peer network with an added aspect of user permission and object state management controlled by the peer that hosts the entire session. A representation of how this architecture looks can be seen in figure 3.4.

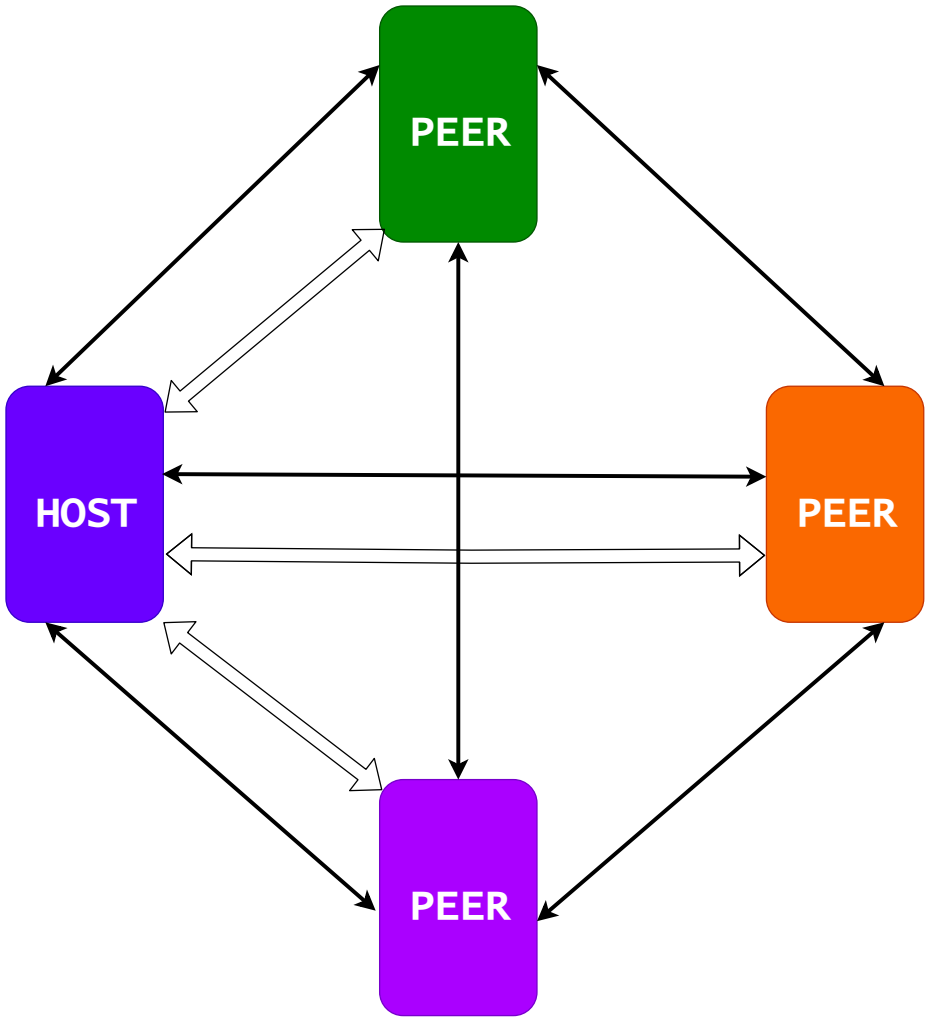


Figure 3.4: Representation of how peers communicate in the network design. The black arrows represent the communication of object state update such as translation or rotation. The white arrows represent the authoritative part of the communication, such as requests to move an object.

Designing the architecture

The layout of the network architecture is simple. The peer who starts the entire session is selected as the hosting node/server. Other peers then connect to the session created by the host. Host has the same capabilities as any other peer in the network, but additionally serves as the point where other „client“ peers ask permission for the interaction with the different objects placed in the AR scene.

In order to discover peers and connect them, the MultipeerConnectivity framework is used. The basic architecture is that when a peer chooses to host the session, he also begins advertising the service to the network. Peers who want to join the session browse nearby services, and once a service is found they connect to it. The host automatically accepts any incoming requests to join the session. The peer IDs available via the framework are used to distinguish between individual devices in the session.

A typical sequence of events that happens once a peer establishes connection with the host is as follows:

1. Registration - the peer registers with the server and is assigned a color. This color is later used to associate the peer actions in the UI.

2. The peer receives map data, a snapshot from the scene and other data necessary to initialize the shared experience from host.

3. Peer is now capable of receiving and sending information about the different user interactions and the changes in individual objects values, such as the position or their color.

```
1 {"type": MesssageType, "data": DATA}
```

Listing 3.1: Default JSON message format

In order to facilitate the communication I have designed a communication protocol using the JSON format. The base format of the messages is shown in listing 3.1

Communication protocol

Several message types were designed in order to accomodate all required situations. Each type of message has its own kind of data structure specified. Data can either contain a simple type, or another JSON formatted object. The basic message types are as follows:

- Register - used when a peer connects to the host. Peer sends the message with empty DATA, and in return receives a message of the same type, but with the assigned color as data.

- Initialize - used for the sending of data necessary to begin the session. Data contains the ARWorldMap object and a snapshot of the scene from which the map was captured. It also contains data from the Virtual Object Manager described in the later section 3.6. Message is sent from the host to a target peer.

- Translate, Rotate, Scale - used to update the position, rotation or scale of the given object. The data contains the name used to identify the object, the state of the transformation (start, end, update) and the vector with values necessary to update the object transformation.

- AssetQuery - this message type is used to manage object state permissions. Data contains the name of the object as well as a query identifier. The peers ask the host with the identifier *request*, and host replies with either *allow* or *deny*.

[1]URL: https://www.json.org/

- assetLock - this message is used by the host to control the peers permissions to edit the given object. The data contains the name of the object and either a *lock* or *unlock* state.
- assetColor - used to change the color of an object. The data contains the object name, name of the object property to be changed and a string with the name of the color to change to.

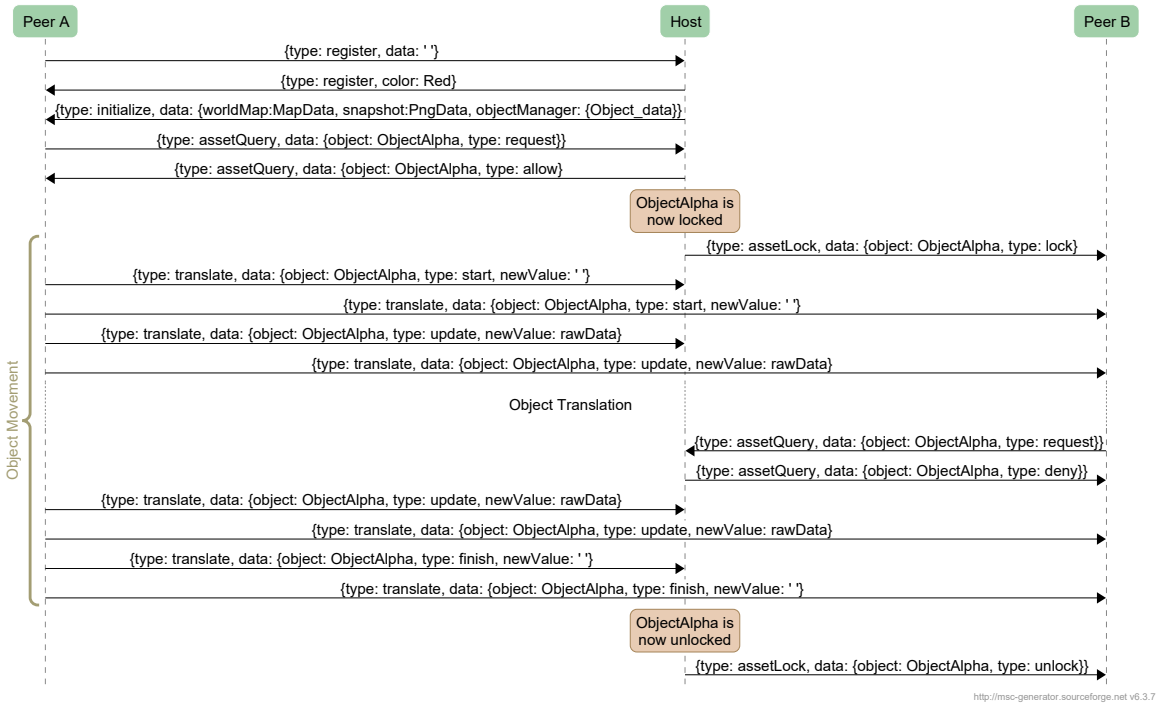


Figure 3.5: This diagram showcases a typical use case when a peer joins a session in progress

Message sequence diagram

The way a typical situation of user joining and moving an object is handled using the designed protocol is illustrated in figure 3.5. A typical conflict resolution when two users try is showcased in the section denoted by the vertical brace. Peer A received permission from the host and was able to move the requested objected. All other users including the host had the interaction locked. When Peer B tried to interact with the object, the request was denied by the server.

3.6 User action and virtual object management

In the final section of this chapter the options on how to deal with virtual object management and user permission are described. An outline of the programmable objects that are used for this task is provided, as well as reasoning behind how and why they are designed in such a way.

Analysis

When dealing with a scene where multiple editable virtual 3D objects are present, an object management system is of significant importance. A virtual object usually consists of multiple parts. A model of a door would for example be composed of a frame, a door body and a door handle. All of these parts can have multiple variations of texture color. One of these textures is set as a base version for when the object is loaded into the scene. In a shared AR session, this can create issues. If an object was edited after placement, the texture would be correct for users who were present before the change happened. However, if a user joined later, the door would be rendered with the base texture. This would create a discrepancy between the user experiences and go against the system requirements that were outlined in section 3.2. The other values that are necessary to track is the current object rotation and scale. Position in the scene is not really an issue, since virtual objects are always rendered on top of anchors. These anchors are stored in the world map and are included when the map is shared to other users.

The other important part is user action management. As I described in section 3.3, there are actions which can create conflicts and user confusion during their execution. As such, a way to store permissions to allow interaction with the virtual objects is required. In the use case that is being considered in this work, there is no need for an overly complex user permission system. Since the solution requires the users to share physical space, and the goal of this work is to enable user cooperation, it is not necessary to limit actions behind levels of permissions. The only permission that is truly necessary is the ability to lock and unlock access to object transformation.

Virtual object manager

Based on the previous analysis, a virtual object manager with these properties is proposed as the ideal solution:

- Capable of storing and identifying all virtual objects present in the scene
- Retains information about specific objects and their properties.
- Allows for locking and unlocking object interaction.

If a system implements a manager fulfilling these designations, multi-user cooperation in a shared AR experience will be possible. The state of virtual objects and their properties will be consistent across all devices.

Chapter 4

Implementation

The proposals and methods described in the previous chapters were used to create a simple demonstration application. This application is simplified in certain aspects, but still retains all of the core features described in this work. This demo application was created by myself and Martin Minárik. His thesis focused on creating an application capable of visualizing animated 3D objects in Augmented Reality on iOS. Based on his work, he implemented the 3D model used in the demo and the insertion of said model into a scene. Next he implemented the parts focused on translation, rotation and scaling of the model, along with the touch based gestures used to control them.

My contribution to the implementation focused on the core tenets of my work, meaning I implemented the parts dealing with the shared experience. I designed the UI and management of the virtual object properties and permissions with focus on sharing. Next I implemented the networking architecture used to connect and share the augmented reality experience. My work on the networking architecture involved the implementation of the communication protocol, serialization and deserialization of data sent via the network and the selection of proper moments to transmit the virtual object transformations.

The resulting application was presented as a part of the Excel@FIT 2019[1] conference [13]. The visitors of the conference were able to try out the implementation on two iPad devices.

The following sections will describe and show the way the shared AR experience was implemented and an evaluation of how successful the implementation was from the point of user experience.

4.1 The Knight demo

The goal of this demo is to showcase the methods with which a multi-user cooperation in a shared augmented reality can be achieved. The application allows multiple people to view and edit a single virtual 3D object in a shared scene from their own devices. The changes made to the virtual object are transmitted instantly to all the users connected to the session. The virtual object used in this demonstration is a model of a knight. The knight can be moved, rotated and scaled by any user in the session. It is also possible to edit the individual properties of the model. In the case of this demo parts of the knight's body, such as the helmet or shield, can have their texture color changed. Multiple people

[1] URL: http://excel.fit.vutbr.cz/

can thus cooperate together to create their ideal version of the knight based on the group's preferences.

The demo also implements the virtual object management system. Thus the object's parts stay consistently colored no matter when a user joins the session. The entire project is implemented in Swift and uses only frameworks native to iOS.

4.2 UI Design implementation

The UI used in the demonstration application was implemented using the Interface Builder available in XCode IDE[2]. The interface builder is called Storyboards and it allows for easy development of UI by separating the interface into a set of scenes with described relations. The storyboard used in the implementation can be seen in figure 4.1. Individual scenes are easily built by using simple interface blocks available in the environment. The changing of scenes is described using transitions called segues. Storyboards also allow the for the usage of the Auto-layout system, which defines mathematical relationships between individual elements using constraints. Well defined constraints allow for automatic resizing and movement of the element based on the device screen size. The bulk of the design was however tested on the iPhone 7 Plus and an iPad, so the smallest devices may have some issues with scaling.

Apple interfaces follow the Model-View-Controller architecture pattern. Models manage the data and logic of the application, views display information based on data acquired from models, and controllers serve to manipulate the model logic and respond to input from the view interface.

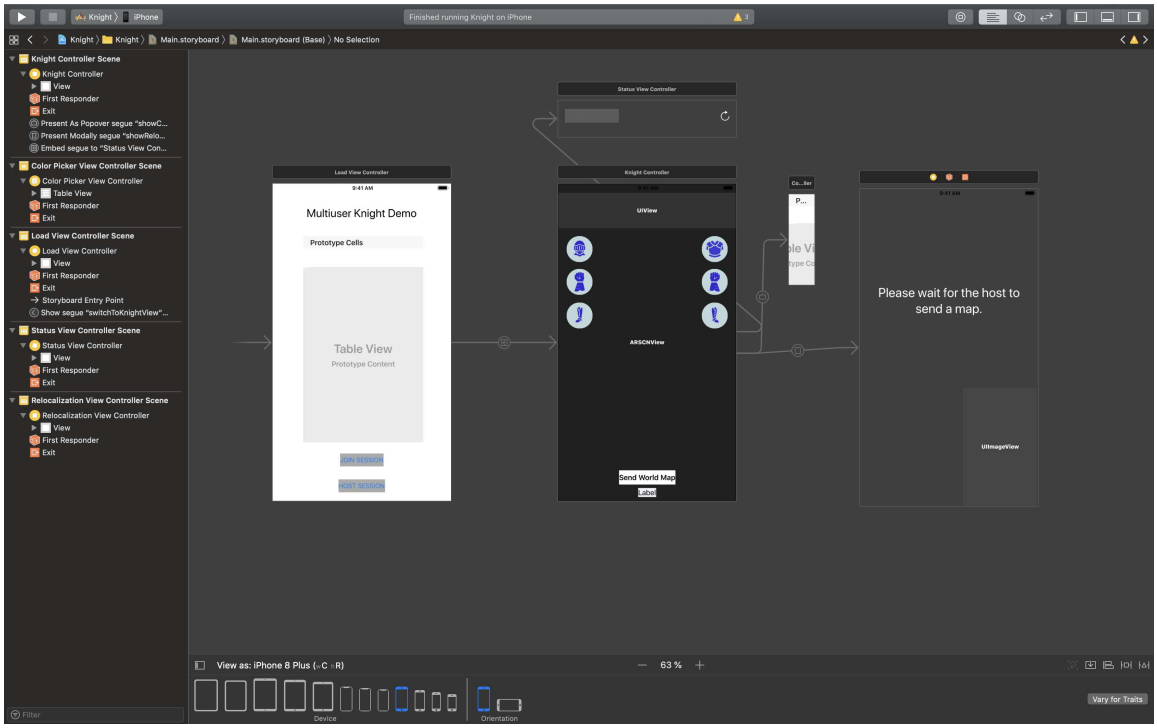


Figure 4.1: Storyboard of the final UI used in the project. It shows the scenes and relations used to build the interface.

[2] https://developer.apple.com/xcode/

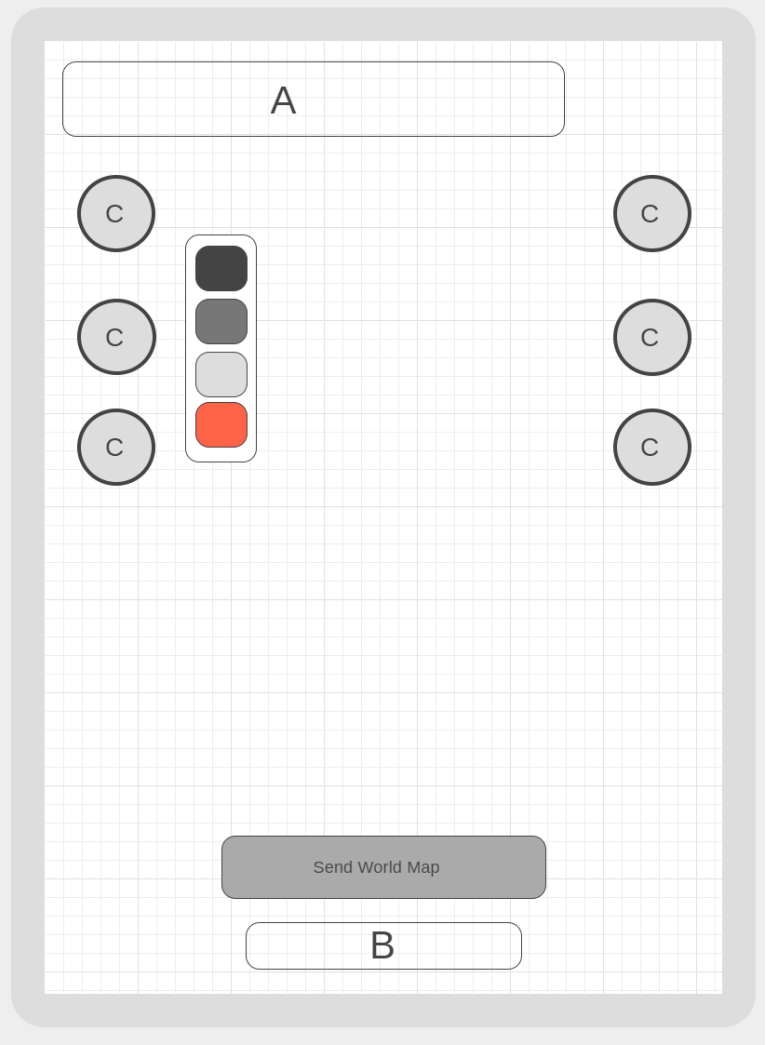


Figure 4.2: Wire-frame showing the layout of control elements. A is the area to display guidance messages based on tracking. B is the area to display world mapping status. C are controls used to modify object properties.

The UI was implemented using two main scenes. The first scene manages the connections and selection of whether the user wants to host or join a session. The second scene is where most of the user interactions happen. It displays the augmented reality scene and contains the control elements of the UI. The wire-frame of these user controls is shown in figure 4.2. The second scene uses a combination of 4 different views to satisfy the requirements set in section 3.3.

Load View scene

The loading view is very simple, and consists of a table of discovered sessions and buttons to join or host the session. The service discovery process begins immediately after launching the application. The view controller responsible for this scene is *LoadViewController.swift*. The view controller implements delegate methods from the *MCNearbyServiceBrowser* ob-

ject to display the available peers in the table view. By clicking the Join or Host session button the entire scene shifts to the next view.

Knight View Scene

This is the most important view in the entire implementation. It combines 4 different views to achieve all the functionality required of a proper multi-user AR experience application. At the core of it lies the *ARSCNView* controlled by the *KnightViewController*. This view controller is responsible for tying all the different models and views together. It handles the setup and initialization of the *ARSession* and rendering of the 3D model into the scene. It also initializes gesture handlers for user interaction.

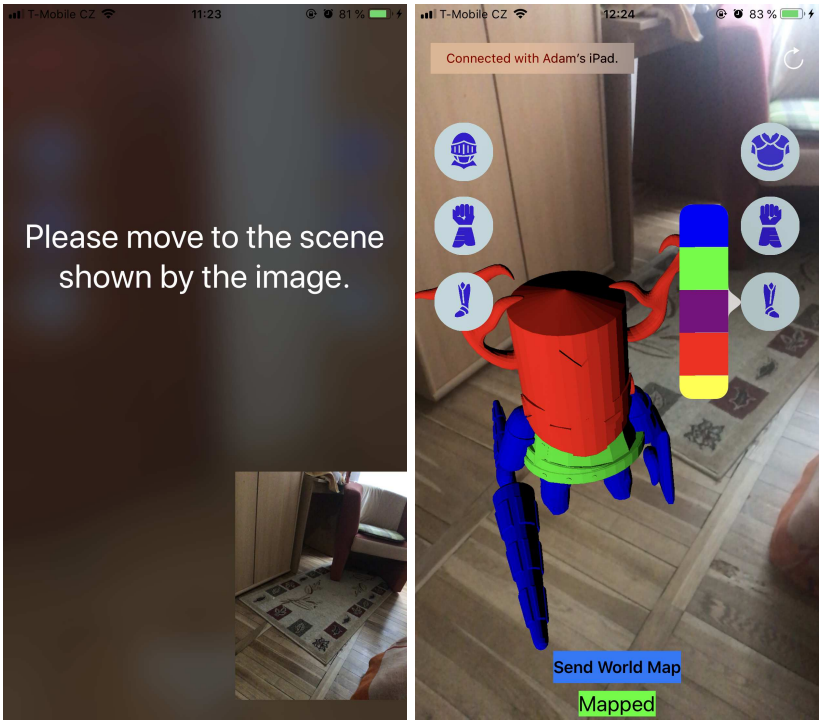


Figure 4.3: The left figure shows the screen that a client sees when he receives a map object from the host. The right figure shows the arrangement of the UI in the main scene.

The view implements user controls for the selection of object properties, in this case colors, and the sharing of the world map. These controls can be seen in figure 4.3. Two informative elements are also present to indicate the state of the world mapping status and the general tracking state. The content of these elements is set by implementing an *ARSessionDelegate*, which is being called with status updates for every frame. The delegate methods are implemented in an extension of the *KnightViewController* class in file *KnightViewController+ARSessionDelegate.swift*.

The 3 view controller scenes that are either embedded or overlaid on top of the main view are:

- *ColorPickerViewController* - a popover view that displays a choice of colors.

- *StatusViewController* - embedded as a child view on the top of the screen, it handles the display of messages containing suggestions and information based on the current tracking state.
- *RelocalizationViewController* - this view appears modally over the base view and serves as a guideline for newly connected peers to achieve relocalization.

Figure 4.1 nicely shows how these individual views are arrayed.

4.3 Virtual object manager

The virtual object manager that I implemented for this demo adheres to the properties set in section 3.6. Because there is only one virtual object, there is no need to implement methods to manage multiple objects. The manager in this application focuses solely on the management of the knight model properties and their individual lockability.

The manager is implemented in the file *VirtualObjectManager.swift*. The implementation consists of a set of structures that adopt the *Codable*[3] protocol. This protocol enables easy serialization of the properties of the manager, and thus it is easy to share the information contained within across the network. The structures are defined in the listings 4.1, 4.2. The different types of nodes monitored are defined in listing 4.3

```
public struct VirtualObjectManager: Codable {
    var transformation: ObjectTransformation
    var managedNodes = [ManagedNode]()

    init(transformation: ObjectTransformation) {
        self.transformation = transformation
    }
}
```

Listing 4.1: VirtualObjectManager is the base structure that is used in ViewControllers

```
struct ManagedNode: Codable {
    var interactionState: LockState = .unlocked
    var type: NodeType = .undefined
    var color: Color = .blue
}
```

Listing 4.2: ManagedNode retains information about individual parts of the knight model.

The *managedNodes* array is filled with the editable parts once the *Knight View* loads. The individual values of properties such as the object rotation or part color are updated whenever a change to the model occurs. Thus they are being kept always in sync. This is important, since the entire *VirtualObjectManager* structure is being sent along with the world map when a new peer joins the shared session. The information contained within this data is then used to initialize the object to the proper state on the new device.

[3] https://developer.apple.com/documentation/foundation/archives_and_serialization/encoding_and_decoding_custom_types

```
enum NodeType: String, Codable {
    case mask
    case body
    case arm_left
    case arm_right
    case leg_left
    case leg_right
    case knight
    case undefined
}
```

Listing 4.3: An enumeration of body parts available to edit in the knight model

4.4 Network implementation

Network implementation will be presented in three distinct parts. The first subsection will show how the communication protocol and individual message encoding/decoding were implemented. The second part will deal with the objects used to manage the network connection. The final part will showcase how the transformation in the object movement is extracted and sent.

Communication protocol and message encoding

The key aspects in the implementation of the messaging infrastructure were the *Codable* protocol and Swift language's native JSON encoders. Using these two techniques together allows for a very clear-cut realization of the communication protocol. This approach was already described in the previous subsection. The structures that are key are the *MessageCore* and *MessageData*. These structures are partially defined in listings 4.4, 4.5. *MessageCore* as can be seen in the listings is the base from which all messages are formed. The usage of optional members within the *MessageData* structure in listing allows for a dynamic range of messages to be used. The listing 4.6 shows the process in which a message is created and encoded. Listing 4.7 shows the resulting JSON string.

```
public struct MessageCore: Codable {
    let type: MessageType
    let data: MessageData
}
```

Listing 4.4: Structure used to encapsulate the entire message.

The Message object is an implementation of an object class in which the *JSONEncoder* and *JSONDecoder* are hidden inside class functions.

Network management objects

Two network management objects were implemented. The first is the the *MultipeerSession* object, which contains implementation of the MultipeerConnectivity framework. It is designed as a singleton that is used to send data to peers on the network. It can send data

```
public struct MessageData: Codable {
    var text: String?
    var assetData: AssetData?
    var transformData: TransformData?
    var worldMap: Data?
    ...
}
```

Listing 4.5: Structure used to encapsulate varied types of data payloads.

```
var msgData = MessageData()
msgData.transformData = TransformData(object: "knightNode",
type: .update, newPosition: data)
let msg = MessageCore(type: .scale, data: msgData)
let jsonMsg = Message.encode(msg)!
```

Listing 4.6: Creation of a message object and encoding it to JSON

```
1 {"type":"translate","data":{"transformData":{"type":"update","
    object":"knightNode","newPosition":DATA}}}
```

Listing 4.7: Resulting JSON string

to all peers, all peers except one or directly to a chosen peer. It contains an array of all connected peers and handles connections and disconnections of individual peers.

The second object I implemented is the *NetworkManager*. The main role of the *NetworkManager* is to handle incoming messages, decode them and call appropriate delegate methods. The delegate protocol is adopted in an extension by the *KnightViewController*. This way the received data can be incorporated into the AR session. The network manager on the hosting peer also administrates the information about peers connected to the session.

Transmission of object transformation

The transmission of updates of the virtual object's transformation in space while the user manipulates them is handled using the individual translation/rotation/scale types of messages. These messages always contain information about the new transformation as an encoded vector consisting of 3 float values. This transformation is calculated based on the user's gestures and the result of the calculation is shared to the other peers. How this transformation is calculated is described in section 3.3. The transformation information is sent continuously during the gesture progress. Because the devices are located in the same coordinate space, they can simply apply these results in their own views whenever they receive them.

A long user gesture carries within itself information about the state of the gesture - start, changed and ended. These gesture states are used to trigger the sending of specific messages in the communication protocol, such as *assetQuery* during the started state, or translation with new position during the updated state. Gesture's updated state can be triggered multiple times as the movement on screen progresses. They can also be used to

trigger the highlighting of the object based on the user who is performing the action, thus providing more clarity to what is currently happening with the virtual object.

World mapping data and object placement

The implementation featured in this work does not share map updates between devices past the initial sending of the map. However, once the devices successfully relocalize into the shared experience, the coordinate systems of maps are synchronized. Thus, when individual devices increase their understanding of the environment by moving around, the representation of the same physical environment inside their local SLAM maps should be mostly similar. There is, however, no guarantee of accuracy, as this depends solely on the quality of the tracking data collected by the user.

Virtual objects are placed into the scene using anchors described in section 2.2. Thanks to the ARKit framework, the anchors automatically align themselves into the SLAM map accordingly to the quality and state of mapping of the environment. Thus a virtual object tied to an anchor can have its position in the scene improved when more mapping data becomes available. The method to place a virtual object is described in section 3.3.

With these two features, it is possible for the application to correctly display virtual objects that have been placed outside of its currently mapped environment. This is useful in situations where, after relocalization, one of the peers (user A) leaves the initially mapped area. This user can then place a virtual object in a place that the other peer (user B) has not yet mapped. Once the object is placed, the implementation sends the ARAnchor object to the other peers in the session. Thus, when user B moves to the scene where user A placed the object, the virtual object is displayed on user B's device in an approximately similar position.

The accuracy of the displayed virtual object's position between devices depends solely on the tracking data acquired by the individual devices. In order to improve this accuracy, a solution using some sort of a central Cloud architecture to create a SLAM map based on the combined tracking from all connected devices would be necessary.

4.5 Evaluation of the implemented solution

The designed and implemented solution was presented as a demo application available to the attendees of the Excel@FIT 2019 conference. The attendees were able to try out the application on two iPad devices. Around 15 people decided to test it. The users were observed during their interaction with the application and then presented with questions about their experience. Aside from the testing conducted during the conference, additional tests were conducted by myself with a friend in different types of environments.

The conditions of the environment during the conference were not ideal for relocalization purposes. Our stand was located in a hallway with monotonously toned tiles and white walls. Random people were also constantly walking past. This led to the lengthening of time needed to be spent on the synchronization of the augmented reality sessions. Once connected, the session worked fine and without too many issues, with only occasional glitches and disappearing of the virtual object.

The main criteria observed during the testing can be summed into three categories:

1. Intuitiveness of the designed user interface.

2. Responsiveness of the system to user actions and the action synchronization across the connected devices.

3. Ability of the system to guide the user to resolve issues in non-ideal conditions.

Intuitiveness of the user interface

The overall experience during the usage of the application can be split into two parts. The first is the process at the launch of the application before the AR session is connected and synchronized. The users needed to perform specific actions in order to connect the sessions. The UI proved to not be sufficient, since most of the users could not achieve this without external guidance. In the demo version of the application available during the conference, there was no special view available for the joining peers to assist them with the process of relocalizing into the host's map.

Once the users managed to connect the sessions and a model was displayed in the shared scene, the interface controls based on gestures and some buttons were sufficient. The users were able to control and interact with the scene without any suggestions necessary from my side. They were able to cooperate and edit the colors of the knight to their liking, as well as move and scale the model according to their gestures.

Responsiveness and consistency of user actions across devices

The synchronization of the actions such as movement of the knight across the scene were smooth and with minimal delay. Users expressed no complaints about the speed of the synchronization. The locking of other people's ability to move the virtual object when it was already being interacted with served well in preventing user action collision. Some users, however, remarked that some sort of visual notification when the object was being interacted with by the other users would be welcome, as it was confusing for them not being able to suddenly interact with the object.

While most of these test were performed with two users present in the scene, a connection using 4 devices at once was also tested. Even with the added devices, the movement and position of the knight model stayed consistent across all devices. There was no noticeable increase in the delay.

Ability to guide the user

The user interface provides information about the state of the tracking when an issue occurs. It also provides information about the quality of the mapping information gathered from the tracking of surroundings. If the tracking data is insufficient, the users are not able to share their world map to others.

While the information is provided to the users, most of the participants did not know how to use it. Once an explanation was provided as to how the application can be used, the users were able to successfully share and synchronize their AR sessions. This revealed an issue that should be resolved in future improvements.

In order to improve this aspect of the user experience, a tutorial should probably be provided when the user first launches the application. In an ideal situation, with an environment that is visually varied and well mapped, relocalization was successfully achieved from opposite points of view.

Chapter 5

Conclusion

The goal of this work was to create a solution that would allow multiple people to cooperate in a shared augmented reality scene. The solution would provide the ability to interact with 3D virtual objects placed in the scene to the users. The individual peers would be able to see the interactions done by the other participants in real-time, and the content displayed would be consistent for all users no matter when they joined. The experience should use the ARKit framework and be based on the iOS devices.

In order to accomplish the goal, a theoretical basis was provided for the techniques that enable a device to be able to display and interact in augmented reality. The inner workings of the ARKit framework were explained, which allowed for the definition of issues that a multiplayer shared experience faces. In order to implement the network requirements, classical network architectures and ways to share data on iOS were presented.

Based on the theoretical background, a set of basic system requirements was defined. The following sections then focused on proposals with which these requirements could be met. Basic workflow of the application was defined along with user actions that should be available to the users in the system. A hybrid peer-to-peer network architecture was chosen to be used as the basis for the network communication. A communication protocol was also created, along with systems to manage virtual objects.

The proposed solutions where then implemented in a demo application. The demo allows user to place a model of a knight in a shared reality scene. The users can freely move the object using gestures and edit the color of individual parts of the body using a set of buttons. These actions are synchronized to all the users in the session. The demo application was tested by multiple users in a variety of real environments.

While the demo itself is quite simple in terms of available interactive features, it demonstrates the feasibility of the different systems proposed in the work. Based on testing, the designed systems and network architecture can work for this type of applications. The work can be used as a basis for further development of multi-user interactive AR applications. Future works could also focus on measuring the efficiency and scalability of the designed network architecture for this type of workloads.

The biggest hurdle for the development of more interactive applications are the limitations of tracking and mapping systems. A more robust approach than the one used in ARKit needs to be created in order to achieve fully immersive and interactive shared experiences. Once the relocalization process to join a shared map becomes possible without any previous preparations made by the user, will the era of true Augmented Reality begin.

Bibliography

[1] Apple Inc.: *Augmented reality - System capabilities - iOS - Human Interface Guidelines - Apple Developer*. [Online; Visited 07.05.2019].
Retrieved from: https://developer.apple.com/design/human-interface-guidelines/ios/system-capabilities/augmented-reality/

[2] Apple Inc.: *Managing Session Lifecycle and Tracking Quality*. [Online; visited 07.05.2019].
Retrieved from: https://developer.apple.com/documentation/arkit/managing_session_lifecycle_and_tracking_quality

[3] Apple Inc.: *MultipeerConnectivity | Apple developer documentation*. [Online; visited 07.05.2019].
Retrieved from:
https://developer.apple.com/documentation/multipeerconnectivity

[4] Apple Inc.: *Understanding ARKit Tracking and Detection*. 2018. [Online; Visited 06.05.2019].
Retrieved from: https://developer.apple.com/videos/play/wwdc2018/610/

[5] Cohen, B.: *The BitTorrent Protocol Specification*. 2008. [Online], visited 8.5.2019.
Retrieved from: http://www.bittorrent.org/beps/bep_0003.html

[6] Jardine, J.; Zappala, D.: *A Hybrid Architecture for Massively Multiplayer Online Games*. In *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*. NetGames '08. New York, NY, USA: ACM. 2008. ISBN 978-1-60558-132-3. pp. 60–65. doi:10.1145/1517494.1517507.
Retrieved from: http://doi.acm.org/10.1145/1517494.1517507

[7] Jianbo, S.; Tomasi, C.: *Good features to track*. In *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. June 1994. ISSN 1063-6919. pp. 593–600.

[8] Lu, F.; Milios, E.: Globally Consistent Range Scan Alignment for Environment Mapping. *Autonomous Robots*. vol. 4, no. 4. Oct 1997: pp. 333–349. ISSN 1573-7527. doi:10.1023/A:1008854305733.
Retrieved from: https://doi.org/10.1023/A:1008854305733

[9] Lucas, B. D.; Kanade, T.: *An Iterative Image Registration Technique with an Application to Stereo Vision*. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI'81. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.. 1981. pp. 674–679.

[10] Matoušek, P.: *Síťové aplikace a jejich architektura*. Brno: VUTIUM. first edition. 2014. ISBN 978-80-214-3766-1.

[11] Miesnieks, M.: *Why is ARKit better than the alternatives?* July 2017. [Online; visited 07.05.2019].
Retrieved from: https://medium.com/6d-ai/why-is-arkit-better-than-the-alternatives-af8871889d6a

[12] Miesnieks, M.: *Multiplayer AR – why it's quite hard*. January 2018. [Online; visited 04.05.2019].
Retrieved from:
https://medium.com/6d-ai/multiplayer-ar-why-its-quite-hard-43efdb378418

[13] Minárik, M.; Jurczyk, A.: *Multi-user interaction with 3D objects in Augmented Reality*. In *Proceedings of Excel@FIT 2019 Conference*, vol. 5. FIT VUT. April 2019.

[14] Nister, D.: An efficient solution to the five-point relative pose problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. vol. 26, no. 6. June 2004: pp. 756–770. ISSN 0162-8828. doi:10.1109/TPAMI.2004.17.

[15] Rosten, E.; Drummond, T.: *Machine Learning for High-Speed Corner Detection*. In *Computer Vision – ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I*, *Lecture Notes in Computer Science*, vol. 3951. Berlin, Heidelberg: Springer Berlin Heidelberg. 2006. ISBN 978-3-540-33832-1. pp. 430–443.

[16] der Sar, E. V.: *Spotify Starts Shutting Down Its Massive P2P Network*. 2014. [Online; Cited 8.5.2019].
Retrieved from: https://torrentfreak.com/spotify-starts-shutting-down-its-massive-p2p-network-140416/

[17] Schmalstieg, D.; Höllerer, T.: *Augmented Reality: Principles and Practice*. Boston: Addison-Wesley Professional. June 2016. ISBN 978-0-321-88357-5.

[18] Schollmeier, R.: *A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications*. In *Proceedings of the First International Conference on Peer-to-Peer Computing*. 09 2001. ISBN 0-7695-1503-7. pp. 101 – 102. doi:10.1109/P2P.2001.990434.

[19] Tomasi, C.; Kanade, T.: *Detection and Tracking of Point Features*. Technical report. Carnegie Mellon University Technical Report CMU-CS-91-132. April 1991.

Appendix A

Contents of the included media disc

The following layout of contents is present on the included disc:

- **knight_sources** – this folder contains the XCode project and source files of the application.
- **thesis_sources** – this folder contains the .tex files and images needed to compile the thesis.
- **multimedia** – this folder contains the poster and video.
- **installation_manual.txt**
- **xjurcz00-ar-multi-user-cooperation.pdf** – thesis text.

Appendix B

Installation manual

In order to compile this work, a system capable of running XCode 10.0 and at least two devices running iOS 12.0 with the A9 or later processor are necessary (iPhone 6S and later, iPad 2017(5th edition) or later). A developer profile created on the Apple website is also necessary (can be used with the free version). Installation steps:

1. Open the project located on the included disc in the *knight_sources* folder in XCode.
2. Open project settings and set the developer profile in Signing.
3. Connect the iOS devices to the computer.
4. Allow the device to trust this computer.
5. Install the application by selecting the device as a target and running the project.
6. If this is the first usage of the given developer, it is necessary to enable trust on the iOS device. This can be done in Settings/General/Device Management. The device needs to be connected to the internet during this process.
7. The application will be installed under the name Knight and can be now run.

Appendix C

Poster

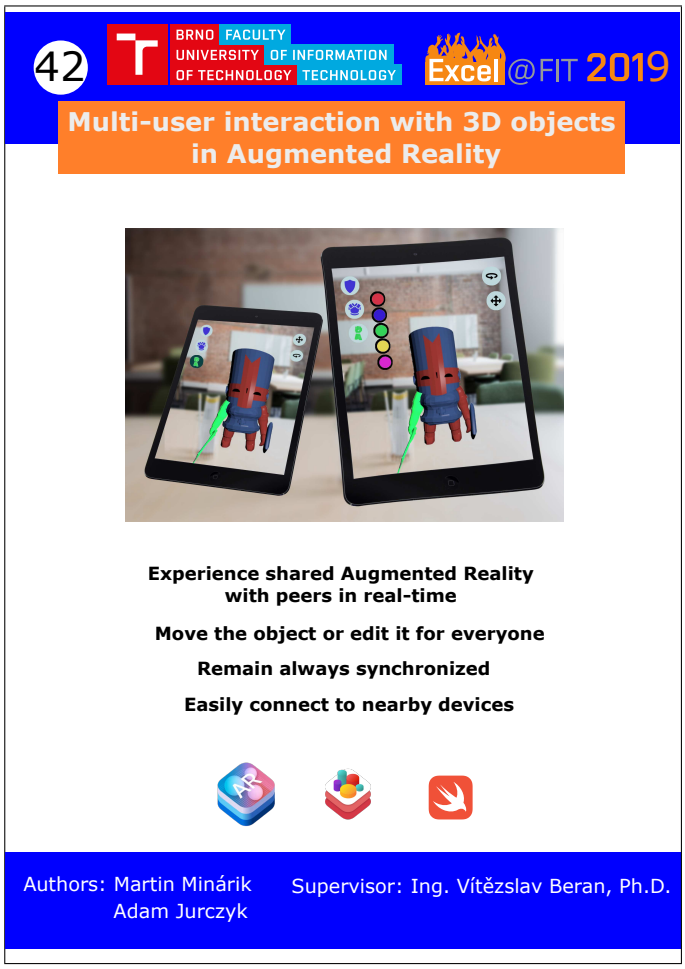


Figure C.1: Poster created for the Excel@FIT 2019 conference showing the work of this thesis.