Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelor Thesis

Peer-to-Peer Distribution of Mobile Simulations

Filip-Emanuel Birtar

Course of Study:	Informatik B. Sc.
Examiner:	Prof. Dr. rer. nat. Kurt Rothermel
Supervisor:	M. Sc. Johannes Kässinger
Commenced:	May 3, 2022
Completed:	November 2, 2022

Abstract

This bachelor thesis aims to explore different approaches of distributing a simulation on mobile devices organized in a Peer-to-Peer (P2P) network structure. The starting point is a muscle visualization Augmented Reality (AR) iOS application, which tracks a person using the camera and superimposes a 3D arm model on the left arm of the person. The muscle contraction of the arm is visualized in two ways, either through color, or through the deformation of the model. The application is extended in order to distribute the arm simulation, using Apple's Multipeer Framework as the building block for the P2P network.

Two measuring units which allow a comparison between mobile devices are introduced. The first evaluates a device's performance, while also taking into account the battery level. This unit is used to make a decision which device should perform inference using a Neuronal Network (NN). This device then shares the results to other devices in the network. Using this approach, the computational load of the devices which do not use the NN is reduced. The second evaluates the device's position in relation to the tracked person. It is used to decide either which device should run the NN, or which device should perform the arm tracking of the person. With these two approaches, either the accuracy of the NN results or the tracking accuracy are improved. Distributing the simulation comes at the cost of an increased energy consumption, stemming from the necessary communication. The communication is performed over Wi-Fi, which is expensive in terms of energy usage.

Contents

List of Figures

Acronyms

API	Application Programming Interface. 16
AR	Augmented Reality. 3
CPU	Central Processing Unit. 15
DHT	Distributed Hash Table. 14
DNN	Dense Neuronal Network. 14
FPS	Frames per Second. 39
GPU	Graphics Processing Unit. 21
NN	Neuronal Network. 3
OS	Operating System. 15
P2P	Peer-to-Peer. 3
PC	Personal Computer. 15
PerSiVal	Pervasive Simulation and Visualization with Resource- and Time-Constraints. 11
SoC	System on a Chip. 43

1 Introduction

Augmented Reality (AR) is a field that has seen more and more interest in recent times. It integrates digital information and computer generated content with the real environment, creating an interactive experience. AR applications use different sensors to assess the environment in order to superimpose a 2D or 3D model onto it. This has applications in many different areas, such as gaming, education, architecture or manufacturing, because it enables data visualization in real-time. AR applications can be improved through the use of Neuronal Networks (NNs), which provide a way to efficiently and accurately perform the calculations required for complex simulations. NNs are computing systems whose structure is inspired by biological brains. Artificial neurons are grouped in layers and connected to other neurons, from which they receive signals, which they process and send further. The first layer represents the input values, while the last layer represents the output.

Pervasive Simulation and Visualization with Resource- and Time-Constraints (PerSiVal) [29] is a project that combines AR technologies with different NNs with the purpose of simulating a bio-mechanical arm model. The simulation is performed on mobile devices, such as smartphones and AR glasses. This bachelor thesis explores the distribution of complex simulations on iOS mobile devices, using the arm visualization application as the starting point. The distribution happens by sharing important data for the simulation between multiple devices through a Peer-to-Peers (P2Ps) network. Additionally, two measuring units for comparing devices are introduced. These measuring units evaluate the devices based on their hardware specifications and on their position in the environment. They are intended to offer a quick way to determine which device is best suited to perform a specific task during the run-time of the application.

The thesis is structured in seven chapters, starting with the introduction. The second chapter offers important background information and describes the muscle visualization application in more detail. The third chapter provides an overview of other papers tackling load sharing in P2P networks, followed by the problem statement in chapter four. The fifth chapter presents the design and the implementation. It describes the tasks that will be distributed and it defines the two measuring units. The sixth chapter details the setup of three experiments, as well as the evaluation criteria and the results for each. The results are then explained and interpreted. The final chapter concludes this work and offers an outlook in the form of suggestions for future work.

2 Background

This chapter covers relevant background information for the simulation distribution. This includes a rough overview of AR and P2P networking, a description of the muscle visualization application, and a short presentation of the software used. Finally, it includes observations about the tracking performance of the PerSiVal application, as well as information about a series of benchmarks, both of which are important for defining the measuring units.

2.1 Augmented reality

AR is a technology that offers a real-time interactive experience by blending digital content generated by a computer with information stemming from the real world. Virtual objects are displayed over the environment, with the purpose of facilitating a better understanding and/or interaction with the real world through the additional information provided [3].

In order to be classified as AR, a system has to fulfill the following conditions:

- Real and virtual objects are combined in a real environment
- Interactive, runs in real-time
- Real and virtual objects are aligned with each other seamlessly [10]

In Milgram's reality-virtuality continuum [25], AR is categorized under mixed reality. The difference between AR and Augmented Virtuality and Virtual Reality is that, while in the latter two the surrounding environment is virtual, in AR it is real [10].

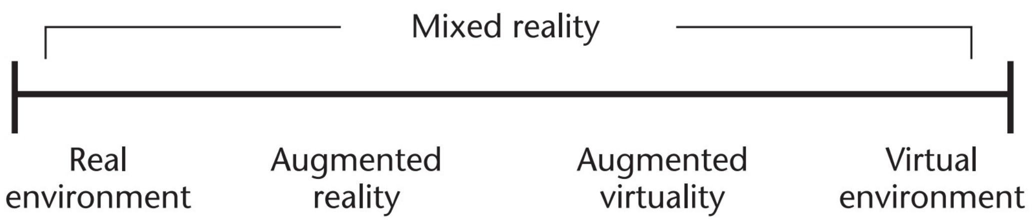


Figure 2.1: Milgram's reality–virtuality continuum [10], [25]

2.2 Peer-to-Peer Networking

According to Schollmeier [31], what defines a P2P network and differentiates it from a Client/Server network, is the ability of the nodes to act as server and client at the same time. This concept led to the introduction of the term *servent*, an artificial word derived from the term server and the term client.

P2P Systems can be classified into unstructured and structured systems. Unstructured networks do not impose any structure on the network, consequently they are easy to build and resilient to peer dynamics. Their weakness lies in locating unpopular files, since there is no network wide available table keeping track of the location of each file [22]. Structured systems impose a specific topology on the overlay network, usually by implementing a Distributed Hash Table (DHT). The nodes search for resources on the network using the hash table. This increases the difficulty of creating and maintaining the network, ascribable to the need to update the table in order to react to changes caused by peers connecting and disconnecting frequently [22].

Another way to distinguish P2P networks is splitting them into *hybrid* and *pure* P2P networks. Hybrid P2P networks include a central entity, which is necessary to provide parts of the offered services, and are, in this way, a combination of P2P and client/server models. The difference to a Client/Server network is the fact that the peers share resources between each other. Pure P2P networks do not include a central entity, meaning only servents are allowed [31].

2.3 PerSiVal

This bachelor thesis is part of a super-ordinate project called Pervasive Simulation and Visualization with Resource- and Time-Constraints (PerSiVal) [29]. The goal of this project is to simulate a complex bio-mechanical arm model. The model is superimposed on the body of an observed person in an augmented reality application which runs on mobile devices. The motion of the person is tracked using the camera of the mobile device in order to obtain the three-dimensional position of three joints, specifically the shoulder, the elbow, and the wrist of the left arm. The three points are used to position the arm model onto the arm of the person, as well as to calculate the angle of the hand. The previous angle is stored in a variable with the purpose of using the angle change to calculate the angular velocity of the arm. Similarly, the velocity change is used to obtain the angular acceleration. These three values, along with the weight the person is lifting, serve as input for a Dense Neuronal Network (DNN). The weight can take one of four values, namely zero, five, ten, and twenty kg, and it is either input by the user through the UI or it is selected before the application is built. Using these four values, the NN outputs five muscle activation values that represent their contraction. The project was carried out in two phases, which resulted in two different applications. Initially, a model with a static shape is overlaid on the person, while the activation values are visualized through the intensity of the color of the muscles. In this Bachelor thesis, this application will be referred to as the color application. The second application simulates the deformation of the biceps muscle, which was the final goal of the project. For this purpose, the biceps activation values calculated by the DNN are fed into a second NN, which outputs 2809 vertices that are used to visualize the deformation through a mesh structure. Calculating such a large number of vertices is costly, a fact that results in the need for a less computationally expensive alternative. In an alternative implementation, in lieu of calculating the position of all the vertices, a comparatively inexpensive reduced NN only calculates the position of 30 vertices. All the other vertices can be interpolated from the first 30 through an interpolation model. Further on, this application will be referred to as the deformation application.

2.4 Software

This section is dedicated to presenting a concise overview of the editing and game-engine software used for the implementation, such as Xcode and Unity, as well as the necessary frameworks.

2.4.1 Xcode

Xcode is an integrated development environment created by Apple for macOS. It provides a tool set designed to facilitate the development, testing, and distribution of programs and applications for all Apple platforms [35]. Among other features it offers Instruments [36], a performance-analysis tool that collects data about different aspects of the application, and debug gauges available in the debug navigator during run-time, which offer information on aspects such as the Central Processing Unit (CPU) and the memory, or the energy and network usage [34].

2.4.2 Unity

Unity is a game engine developed by Unity Technologies. The Unity editor is available for Windows, macOS and Linux, and it supports development for a variety of platforms, such as Personal Computers (PCs) running the Operating Systems (OSs) mentioned above, Android and iOS smartphones, Universal Windows Platform devices, PlayStation, Xbox and Nintendo Switch video game consoles, among others. It offers a C# scripting API which uses the Mono framework, allowing the usage of a multitude of C# libraries. Besides the video game industry, unity has made its way to other industries, including film-making, architecture, and engineering. The real-time 3D capabilities of the platform make it ideal for usage in simulations and visualization applications [33].

2.4.3 AR Foundation

AR Foundation is a cross-platform framework developed by Unity Technologies that provides Unity users an interface for building AR applications. AR Foundation does not implement AR features itself, but rather offers a common API which supports core functionality that is prevalent on multiple platforms. Said features are implemented by platform specific plug-in package providers. The officially supported provider plug-ins are Google's ARCore XR for Android, Apple's ARKit XR for iOS, and OpenXR for Hololens [2]. It is worth mentioning that not all providers implement all features. As an example, only ARKit supports body tracking. Unity Technologies offers a set of example projects that use AR Foundation in order demonstrate its functionality, called AR Foundation Samples. The samples are available in a public GitHub repository [8].

2.4.4 Apple ARKit XR Plug-in

ARKit is an Application Programming Interface (API) that enables the creation of AR experiences. For this purpose it combines motion tracking, camera scene capture, and advanced scene processing [9]. The ARKit XR Plug-in enables ARKit support for Unity. It implements a series of essential XR Subsystems, such as XR Session, XR Camera, XR Point Cloud Subsystem, XR anchor subsystem, and XR Human Body Subsystem. Through these subsystems, features such as Pass-through camera view, which is essential for AR, point clouds, device localization, or body tracking are possible, among others [1].

2.4.5 Multipeer Connectivity

Multipeer is an API created by Apple that supports P2P connectivity and nearby device discovery. It allows iOS devices to connect to each other, thus creating an ad-hoc, P2P mesh network. The communication is carried out using infrastructure Wi-Fi networks, P2P Wi-Fi, and Bluetooth personal area networks. This makes the creation of P2P networks possible in many different settings, even when traditional network infrastructure hardware is not available. An example of an application using the multipeer framework is AirDrop, an iOS feature that allows users to share files to nearby devices [26].

Applications using multipeer run in two phases. The first is the discovery phase, in which the application browses for nearby peers and simultaneously advertises its availability to nearby devices. The second phase is the session phase. The application can invite peers to join a session or accept their invitation to join a different session. Once a peer accepts the invitation and a session is established, the peers can communicate directly to each other [26].

2.5 Tracking

This section provides some observations about the way body tracking works and about its accuracy in a real environment, which were made during the implementation of the distributed muscle visualization application. The body tracking in the PerSiVal application is performed using the *ARHumanBodyManager* class of the ARFoundation API [12]. The ARHumanBodyManager component generates events when it detects a person, which can be subscribed to. Through these events, the position of body joints can be updated in every frame in which a person is visible to the camera. The relevant joints for the application are the left shoulder, elbow, and the wrist

There is no completely objective way to measure the accuracy. The real world positions are unknown, only estimations given by the device based on the camera input are available. For this reason, the tracking accuracy is evaluated based on where the arm model is positioned in different scenarios. A bad tracking accuracy results in an offset between the arm model and the real world arm. An evaluation of the tracking accuracy is relevant for the design of the experiment. As expected, the tracking accuracy is at its worst when one or more joints of the left arm are not visible to the camera. For this reason, tracking from behind a person is not ideal. In the next paragraphs it is distinguished between the tracking accuracy from different angles, assuming the person is facing the camera.

2.5.1 Tracking a person facing the camera

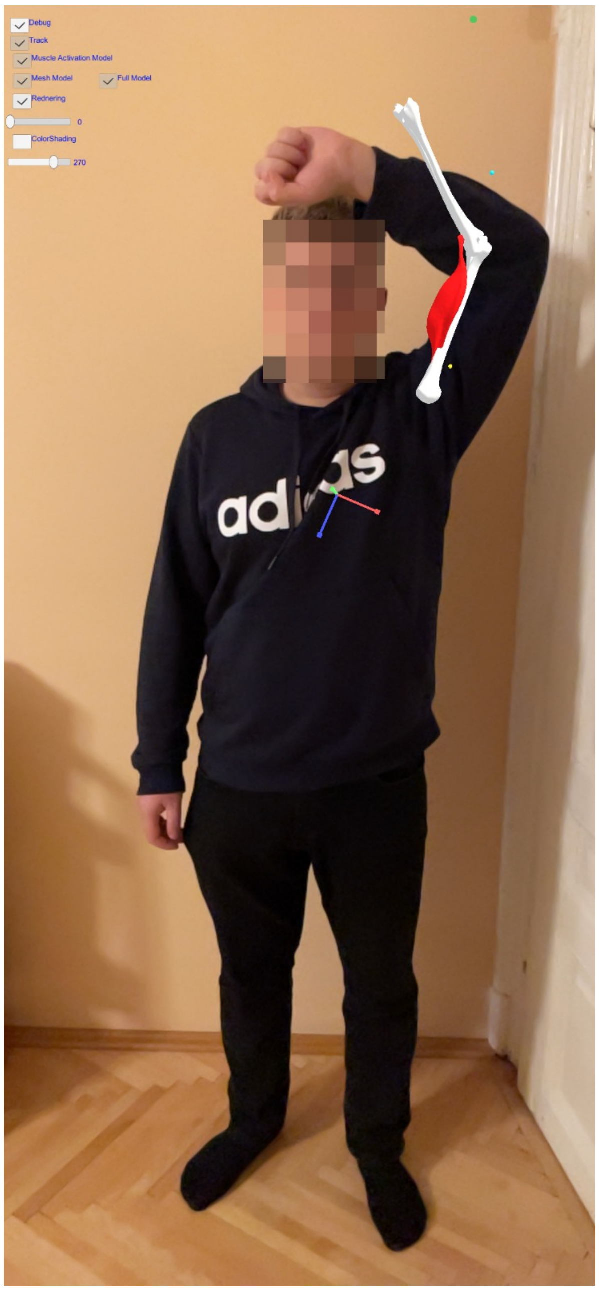
Assuming the person's face is visible, the tracking accuracy suffers when the device is pointed at the right side of the person, as the left arm is not visible or is only partly visible. Some screen captures from this position can be seen in the appendix A. When the device is pointed directly at the front side of the person, the accuracy depends on the arm position. The best accuracy is attained when the arm is in a low range - meaning in a position that is between the natural resting position of the arm, fully stretched downwards, and a position that is raised approximately 25°. In the middle range, from 25° to 90° the accuracy varies greatly depending on the rotation. If the arm is in front of the person and all joints are visible, the accuracy is good. However, if the elbow is close to the body, the accuracy worsens significantly. In the upper range, meaning the arm is between 90° and 180° from the resting position, the tracking performance is at the lowest point for this angle, illustrated in figure 2.2. In general, the best accuracy is attained when the device is pointed at the left of the person, because from this perspective the three joints are visible in the highest number of arm positions. This is exemplified by screen captures in the appendix A, which show the arm in different ranges.

2.5.2 Biceps function and lifting motion

Another consideration relevant for evaluating the tracking accuracy is the fact that the deformation application simulates the biceps deformation. The two main functions of the biceps are the flexion and supination (outward rotation) of the forearm [20]. In order to best represent the biceps deformation for these arm motions, the tracking is most accurate it is performed from the left. Additionally, the application allows the user to select different weights. The biceps deformation is influenced by the weight lifted. Flexing the biceps while holding a weight is a well known weight-lifting exercise called biceps curl. The best and most natural motion for lifting a weight is starting with the arms from the side, then flexing the biceps towards the direction of the shoulders, while keeping the elbow in the same position - close to the body [30]. For tracking a biceps curl of the left arm, the best angle is obtained by pointing the device at the left side of the person. When viewed from other angles, the elbow view is either partly or completely obstructed by the body, which reduces the tracking accuracy. Furthermore, if the biceps is flexed, the elbow view is blocked by the forearm when viewed from the front. The tracking accuracy for the flexion motion viewed from the front is noticeably worse than when it is viewed from the left, as can be observed in figure 2.4. Similarly, the accuracy of tracking a supination motion when the person is viewed from the left is better than the accuracy when the person is viewed from the front.

2.5.3 Consistency

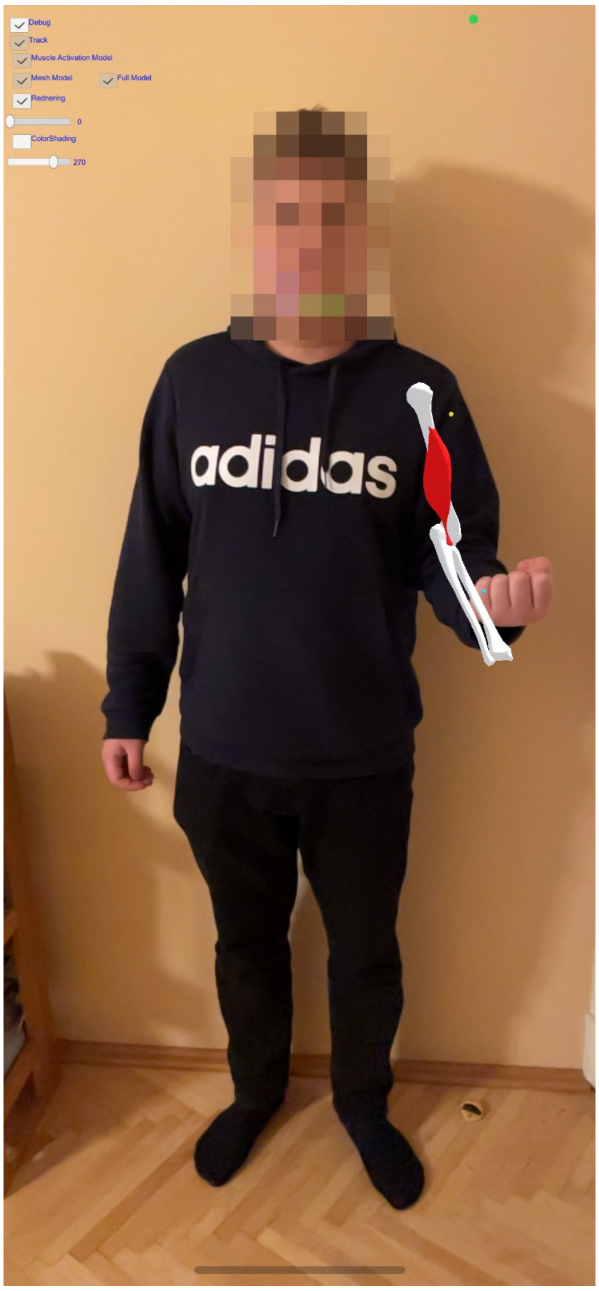
Another observation is the fact that the tracking is not always consistent. This can be illustrated through two examples. Firstly, if a person stands still in a certain position, the device might not track them, however if the person moves and then returns to the same position, the device identifies where the person is and is able to track them, even though they are in the same position as before. Secondly, the device sometimes stops tracking the person for a couple of seconds, especially if the device is in motion.



✓ Debug
✓ Track
✓ Muscle Activation Model
✓ Mesh Model ✓ Full Model
✓ Rednering
0
ColorShading
270

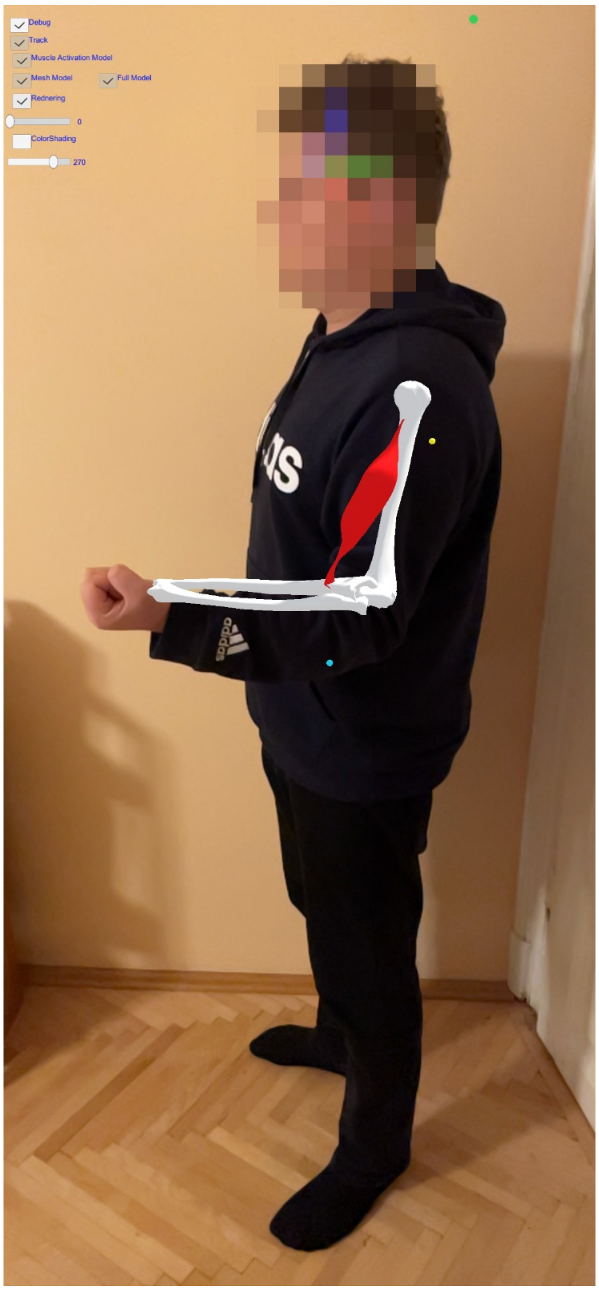
Figure 2.2: Front side, high range

To conclude this section, for the purpose of visualizing the biceps, considering its main functions, flexion and supination, as well as the typical weight lifting motion, the tracking will be assumed to be the most accurate when the device is pointing to the left side of the person. This assumption is also supported by the fact that all the three joints are visible for the highest amount of possible arm position when viewed from the left.



✓ Debug
✓ Track
✓ Muscle Activation Model
✓ Mesh Model ✓ Full Model
✓ Rednering
0
☐ ColorShading
270

Figure 2.3: Front side, flexion



[✓ Debug]
[✓ Track]
[✓ Muscle Activation Model]
[✓ Mesh Model] [✓ Full Model]
[✓ Rednering]
0
ColorShading
270

Figure 2.4: Left side, flexion

2.6 Benchmarks

This subchapter presents three benchmarks that evaluate a device's hardware specifications.

2.6.1 Geekbench 5

Geekbench 5 is the latest major release of the Geekbench multi-platform benchmarking software. It was developed by Primate Labs and it is available for Windows, macOS, and Linux, in addition to mobile operating systems, such as Android, iOS, and iPadOS. This benchmark program focuses on CPU performance, measuring both the single-core and the multi-core power, however it also offers a compute benchmark that tests Graphics Processing Unit (GPU) performance [13]. Geekbench maintains performance charts for categories of devices. The data used for the charts is submitted by users. The baseline score used for calibration is the score of the Intel Core i3-8100 CPU, namely 1000. A higher score is better, and the scoring is linear, meaning double the score indicates double the performance [14].

2.6.2 Geekbench ML

Geekbench ML is a benchmarking software designed to measure mobile inference performance. The machine learning capabilities of a device are evaluated based on the CPU, GPU, and neural engine performance. It utilizes computer vision and natural language processing tests for the evaluation [15]. A machine learning benchmark chart is available. The scores are calibrated against a score of 1500, which is the score of an Intel Core i7-10700 CPU. Similarly to the Geekbench 5 chart, a higher score is better, with a doubled score translating to a doubled performance [16].

2.6.3 PassMark Memory Mark Rating

PassMark is a software company that focuses on benchmark tests. Besides PC benchmarks, it has developed an application that tests and benchmarks mobile devices, called the PerformanceTest Mobile. The application offers CPU, disk, memory, 2D Graphics, and 3D Graphics tests. The user submitted scores are averaged to determine a rating for each of the five categories. The Memory Mark Rating evaluates memory performance [17].

3 Related Work

The concept of sharing a computational load between multiple peer nodes in a P2P network has been explored in different studies. This chapter summarizes related work in the field of distributed P2P networks. Perhaps the biggest challenge of distributing a load is finding the right balance in such a way that individual nodes are not over- or underutilized. There are three different architectures for P2P networks. Unstructured networks are the easiest to build and react the best to network changes. However, the lack of structure gives a disadvantage in the communication, since every node has to flood the whole network to find desired data. Structured networks solve this problem through a Distributed Hash Table (DHT). Peers can use the DHT to search for resources on the network. This comes at the cost of a more difficult set up and less flexibility regarding network changes. The third model is the hybrid model, in which a central server is used to connect peers to each other.

3.1 Dynamic Load Sharing in Memory Constrained Devices: A Survey [28]

The authors perform a review of common load sharing techniques for Internet-of-Things ecosystems, with a focus on devices with limited memory and very strict energy requirements. The three approaches they identify are Computational offloading, Energy-Based approaches, and Deep-Learning based approaches. Computational offloading means reassigning computationally intense tasks from devices with limited resources to ones with ample resources. Energy-Based approaches rely on analytical energy-saving models for the decision-making. Parameters such as the speed of the mobile device or the size of the data to be transmitted are taken into consideration for determining how to share the computational load. In Deep-Learning Based approaches, reinforcement learning models or other deep-learning techniques are used to allocate resources efficiently and to optimize offloading decisions.

The devices used for this bachelor thesis are iOS mobile devices, which are nowhere near as resource limited as Internet-of-Things devices. Similarly to some of the approaches presented in [28], the speed of the devices is taken into account for distributing the simulation, however the size of the transmitted data is constant and consequently does not influence the decision-making.

3.2 Dynamic Load Balancing in Unstructured P2P Networks: Finding Hotspots, Eliminating Them [37]

This paper presents a load balancing algorithm that does not require a specific P2P topology and can thus be applied to unstructured networks. The algorithm works by having nodes with relatively low workloads initiate random sampling operations. If the sampled node is a highly loaded node, the initial node performs a load movement in order to balance their loads. Through this algorithm the authors achieve results that are about four times faster than by using a uniform random load sharing scheme.

3.3 Mobile Computing - A Green Computing Resource [11]

H. Ba et al. describe a mobile computing system prototype named GEMCloud that uses smartphones and tablets instead of the conventional server hardware. They achieve comparable computing performance with an energy reduction between 55% to 98%. The prototype utilizes idle computing power to solve large parallelizable computational tasks. Since mobile devices are rarely turned off, they provide hours of unutilized computing resources. It consists of a network of users that need additional computing power, a server that organizes the network, a database that records information about the mobile clients, and the mobile devices that provide their unused computing resources. The server coordinates the tasks and maintains the database that stores important information such as the ID of the client, their IP address, their hardware capabilities and the task they are performing. When a user requires a task, the server assigns it to a mobile device based on the information it stored in the database. The mobile devices solve the task and send the results back to the server, which aggregates all the results and sends them to the user. The results of the study show that while smartphones and tablets have lower computing power, they have much higher energy efficiency than typical workstations. In order to make up for the lower computing power, more devices can be employed. This approach provides a more efficient and eco-friendly alternative to power-hungry server-based clouds.

In contrast to [11], the present work does not aim to offer a generalized distribution of computational load, but rather to share the results of specific tasks used in a simulation. Distributing a task itself rather than the results would result in extreme communication costs, because the tasks have to be performed every frame in order to obtain a real-time simulation.

3.4 Load balancing in dynamic structured P2P systems [32]

In most implementations of a Distributed Hash Table (DHT), objects are distributed randomly among different peer nodes. As a result, some outlier nodes have $\Omega(logN)$ times as many objects as an average node. In [32], the authors propose an algorithm for load balancing in dynamic, structured P2P networks. They use virtual servers as peers in the DHT, which are moved from heavily loaded physical nodes to lightly loaded physical nodes in order to balance the load.

3.5 MoDNN: Local Distributed Mobile Computing System for Deep Neural Network [24]

DNNs are highly accurate, however this comes at the cost of high resource usage. MoDNN is a local distributed mobile computing system for DNN applications that allows DNNs to be run on mobile devices where the performance would have been unacceptable otherwise. The system comprises three components: 1) a network cluster of mobile devices in which one device acts as the group owner (GO) and the others as worker nodes; 2) a model processor that is responsible for partitioning an already trained NN onto multiple mobile devices over a WLAN; and 3) middleware on each device that has the purpose of scheduling the execution process. Each worker node is mapped with a part of a layer's inputs while the outputs are reduced back to the GO. The input neurons are partitioned according to the computing abillities of each worker node. In a DNN, Convolutional Layers contribute to the majority of computing time, while Fully-Connected Layers contribute to the majority of memory used. There are two partition schemes that minimize data delivery time based on the properties of these two layer types. The middleware performs data delivery and identification services of the DNN. In the evaluation, MoDNN has accelerated DNN computation by 2.17-4.28× while reducing the data delivery time by approximately 30%.

Rather than partitioning a NN so that it can run on multiple devices, the aim of this thesis is to share the output of the NN to multiple devices.

3.6 Placing the present work

What separates the present work from other scientific papers in the field of computational load sharing in P2P networks is the fact that it explores the distribution of an AR mobile simulation. In addition to the device performance, which is a common factor for decision-making, another criteria is used, namely the device's position in the environment. Most papers that were reviewed aim to parallelize or partition a task in a way that allows it to be distributed to multiple peers. This thesis aims to distribute a real-time simulation by assigning tasks to devices which are best suited to solve them, and by sharing their results with other devices. To achieve real-time capability, the tasks need to be performed in every frame. If a task itself were to be distributed to multiple devices, it would be needed to send multiple messages between those devices in each frame. This would result in a very high communication need, which would in turn result in a very high energy usage. If the results of the task are shared instead, the communication bandwidth is greatly reduced.

4 Problem statement

The PerSiVal project aims to enable the pervasive simulation of a bio-mechanical arm model on mobile devices. The mobile devices use sensors and different augmented reality techniques in order to track the movement of a real human arm and projects a three-dimensional, continuum-mechanical upper-arm model onto the real arm. The activation of different muscles is visualized through a changing color intensity or through the deformation of a mesh structure. The activation values are either computed locally or through a combination of local and remote computations. In a research project subordinate to PerSiVal [23], an approach that was considered was offloading a part of the computation to a remote server. Two NNs run in parallel, one on the mobile devices, and a more resource-intensive one on a powerful remote server. The mobile device can send information regarding the environment and request results from the server. This allows for improved accuracy while maintaining usable real-time performance.

In this bachelor thesis another approach will be explored, namely distributing the simulation between multiple mobile devices within the same local-area network in a P2P network structure. This approach provides several advantages: firstly, a reduced network latency due to the proximity of the mobile devices, and secondly, a higher flexibility than the fixed structure of a client/ remote server model. This facilitates the creation of ad-hoc networks and allows the system to react to changes such as devices joining or leaving the network without being dependent on a central server. Instead of each device running a simulation locally, data necessary for the simulation can be shared between multiple devices, either to reduce the computational load or to improve the performance. The aim is efficient resource management and improving the accuracy of the results, which leads to the problem statement of this Bachelor thesis, expressed as a question:

Can the PerSiVal muscle visualization application be extended to run as a distributed system on mobile devices, in such a way that either:

1. **the accuracy of the results is improved, or**
2. **the computational cost is reduced?**

5 Design and implementation

This chapter covers the design and the implementation of the distributed system. In the client-server design iteration of the muscle visualization application, a long short-term memory NN runs on a server, while a less resource-intensive dense NN runs on the mobile device. Both NNs give five activation values for muscles as their output. For the P2P variant, a different distribution of the workload is required. There are two questions which will be answered by the end of this chapter. First, which tasks of the simulation can be distributed, and second, what is a meaningful way to distribute these tasks among the devices?

5.1 Network

The following two sections describe what is required of the network architecture, which architecture fits the requirement the best, and which networking API provides the desired network architecture.

5.1.1 Peer-to-Peer Network architecture

The main requirement from the network architecture is having a highly flexible network that can easily react to changes in the connected peers and supports ad-hoc connections. Additionally, it has to be easy to set up, in order to allow the usage of the application without the hassle of manually establishing a connection. For these reasons, the most appropriate architecture is an unstructured, pure network, as described in section 2.2.

5.1.2 Multipeer connectivity

The most suitable networking API for the requirements of this application is the multipeer framework, which was presented in the section 2.4.5. Not only does it enable the creation of unstructured pure P2P networks, but it is also easy to use in the finished product. When starting the application for the first time, the users are asked for permission to search the network for peers, then, if other peers running the application are nearby, a multipeer session is created and a connection is established automatically. An issue with using multipeer is the fact that it is not officially supported in unity. However, in the ARFoundation samples [8], Unity has partly implemented the framework. While not all multipeer functionality is supported, enough functionality is included to fulfill the communication needs of the implementation.

5.2 Distribution

Without a central entity that decides how the simulation should be distributed and which peer should perform a certain task, there is a need for a different method to organize the distribution. For this reason, two decision algorithms are implemented. Each peer maintains a list of a measuring unit for all connected devices, including itself. The peers calculate the measuring unit and share it to all the other peers periodically or when the network structure is changed. By sorting this list in descending order, each device can determine if it is the most optimal device to perform a task. It then acts accordingly, either by performing the task and sending the result to its peers, or by using the results received from the most optimal device to render the arm model. This approach removes the need for a central entity that communicates the decision to the nodes. The measuring units are determined according to two criteria for distributing the simulation. Firstly, there is a unit that classifies the best device in terms of hardware specifications and battery level, and secondly, a unit that determines the device with the best position for tracking the observed person.

5.2.1 Device score

The first measuring unit used to determine the most optimal device is the device score, which is calculated by a scoring function. This measuring unit is intended to evaluate a device's performance, while concomitantly accounting for the battery status, thus providing a way to quickly compare devices.

Remapping method

A remapping method is needed for the scoring function, which is shown in listing 5.1. The scoring function uses benchmark scores, which can take values from different ranges. The benchmark scores are remapped to a range of 0 to 100 in order to achieve a consistent metric across the different benchmarks. The remapping method takes five parameters as input:

- *value*: the variable to be remapped
- *low1*, *high1*: the lower and upper bounds of the range *value* lies in
- *low2*, *high2*: the lower and upper bounds of the range to which *value* should be remapped

and remaps the given value to a specified range.

```
double Remap(float value, float low1, float high1, float low2, float high2)
    {
        double result = low2 + (value - low1) * (high2 - low2) / (high1 - low1);
        return result;
    }
```

Listing 5.1: Remapping method

Weighted average

The scoring function is a weighted average of the remapped values of three benchmark scores, in addition to the battery score. The battery score is determined by a function 5.1 using the battery level, which can be retrieved as a float between 0 and 1 using the *SystemInfo.batteryLevel* command. By multiplying it with 100 we obtain a value between 0 and 100. If the device is plugged in, the battery level is irrelevant, therefore the battery score is set to 100. The battery status can be checked with the command *SystemInfo.batteryStatus*. If the battery level is under 0.2, which is the default value for triggering the Low Power Mode on Apple mobile devices, the battery score is set to 0 to account for the fact that devices perform slower in Low Power Mode. In this case, the device score is also set to zero.

$$(5.1)\quad \text{battery score } B = \begin{cases} 100 & \text{, if device is plugged in} \\ 0 & \text{, if battery level} \leq 0.2 \\ \text{battery level} \cdot 100 & \text{, otherwise} \end{cases}$$

If energy consumption is not of any concern and only a performance evaluation is desired, the battery score can be changed to 5.2. In the experiments for this thesis, the first formula is used.

$$(5.2)\quad \text{battery score } B = \begin{cases} 0 & \text{, if battery level} \leq 0.2 \\ 100 & \text{, otherwise} \end{cases}$$

The benchmark ratings can be retrieved from a csv file that contains the identifier of supported iOS and iPadOS devices along with their ratings. The file includes iPhone models starting from the 8th iPhone generation and ending with the 13th generation, iPad models between the 6th and the 9th generation, and equivalent generations of iPad Pro, iPad Air, and iPad Mini devices, which are at the time of writing all the models that support the AR libraries used. The identifier is accessible by calling the command *SystemInfo.deviceModel*. Each rating is remapped from the range of the minimum and maximum possible ratings for all devices to the range of 0 to 100, with the help of the remapping method 5.1. This is done in order to achieve a consistent metric for each benchmarking category. The best device in each category has a rating of 100, while the worst device has a rating of 0.

$$(5.3)\quad \text{device score} \;=\; \frac{1}{\alpha+\beta+\gamma+\delta}(\alpha \cdot G + \beta \cdot Ml + \gamma \cdot M + \delta \cdot B)$$

The formula for the weighted average is shown in 5.3. G represents the remapped Geekbench 5 rating, as described in 2.6.1. This rating represents the CPU performance, which is an important metric in evaluating a device, however it is not the most important one for the tasks that need to be performed. For this reason, the weight α of G is set to 0.5. The remapped Geekbench ML rating 2.6.2 is represented through Ml. This is the most important metric for mobile inference, as it evaluates three hardware components, namely the CPU, the GPU, and the neural engine.

Consequently, the weight β is set to 2. M represents the remapped PassMark memory rating 2.6.3. Although the memory performance is important for reading and writing data such as the vertex positions, the ML rating is more important, hence the weight γ is set to 0.5. Finally, the weight δ for the battery score B is set to 1. The resulting formula for calculating the score is 5.4.

$$\text{device score} = \frac{1}{4}(0.5 \cdot G + 2 \cdot Ml + 0.5 \cdot M + 1 \cdot B) \tag{5.4}$$

Tiebreaker

If two identical devices have the same battery score, they will both have the same score. To solve this issue, the scores are rounded and then a value in the interval $(0, 0.5)$ is added. This value is calculated based on an unique device identifier, retrieved by the command *SystemInfo.deviceUniqueIdentifier*, ensuring that no devices will have the same final score.

5.2.2 Angle score

Based on the observations and reasoning from section 2.5, five angle ranges are defined as shown in 5.1. Each range is assigned a numerical value, meant to rank how good a device located in that range can perform the tracking. The angle range which is assigned the value 5 has the best visibility and tracking accuracy, while the angle range assigned the value 1 has the worst visibility and tracking accuracy.

An imaginary circle is drawn, with the middle being the ”root” of the tracked person. The root is a joint that can be tracked by the ARHumanBodyManager and represents the ”origin” or the center of the person. It is the joint that is the tracked the best from most angles. The cirlce is drawn in such a way that the projection of the camera lies on it. Furthermore, it is perpendicular to the vector pointing upwards from the root. 0° is defined in the point where the forward facing direction stemming from the root intersects the circle.

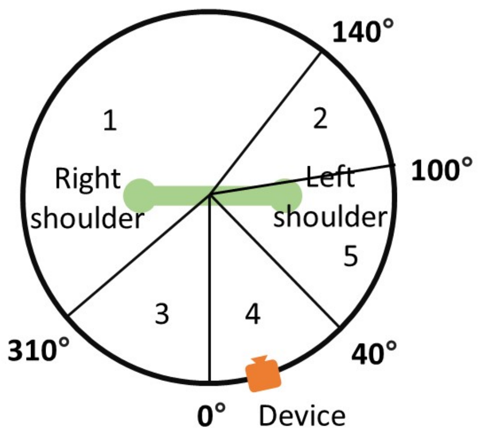


Figure 5.1: Angle classification

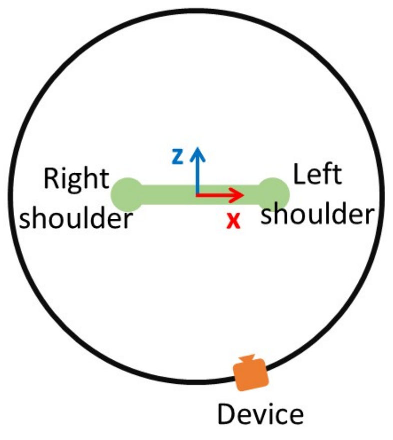


Figure 5.2: Root transform

The challenge now lies in determining at what angle the camera is situated on the circle, starting from 0° and going in a counter-clockwise direction. The root joint has a transform component, which has three normal vectors, $\overrightarrow{x}$, $\overrightarrow{y}$, and $\overrightarrow{z}$. $\overrightarrow{x}$ and $\overrightarrow{z}$ are illustrated in 5.2, while $\overrightarrow{y}$ is pointing upwards. The vector $\overrightarrow{h}$ from the root to the camera, which is shown in figure 5.3, can be obtained by subtracting the root position from the camera position. $\overrightarrow{h}$ is then projected on the plane in which the circle lies, defined by $\overrightarrow{x}$ and $\overrightarrow{z}$. The normal vector of this plane is $\overrightarrow{y}$. The result is $\overrightarrow{v}$, calculated through the formula 5.5 by subtracting the $\overrightarrow{y}$-component of $\overrightarrow{h}$ from $\overrightarrow{h}$.

$$(5.5)\quad \overrightarrow{v} = \overrightarrow{h} - (\overrightarrow{h} \cdot \overrightarrow{y})\overrightarrow{y}$$

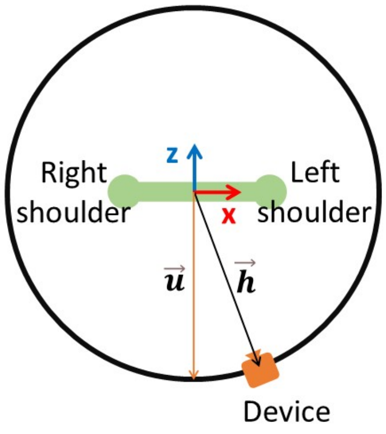


Figure 5.3: Root to Camera Vector

The vector $\overrightarrow{u}$ is obtained by starting from the root and going in the negative $\overrightarrow{z}$ direction with the magnitude of $\overrightarrow{v}$. $\overrightarrow{v}$ and $\overrightarrow{u}$ are shown in figure 5.4. The angle between $\overrightarrow{u}$ and $\overrightarrow{v}$ is defined as φ. In order to calculate φ, first it needs to be determined if $\overrightarrow{v}$ lies in the **I** or in the **II** half of the circle. This can be achieved through the dot product of $\overrightarrow{v}$ and $\overrightarrow{x}$, which is positive if $\overrightarrow{v}$ is in **I** and negative otherwise. The two halves are illustrated in 5.5. φ is calculated through the formula 5.6. The end result is the angle score **as**, which determined by assigning the value matching the range in which φ lies, in accordance to 5.1.

$$(5.6)\quad \varphi = \begin{cases} arccos(\frac{\overrightarrow{u} \cdot \overrightarrow{v}}{|\overrightarrow{u}| \cdot |\overrightarrow{v}|}) & \text{, if } \overrightarrow{x} \cdot \overrightarrow{v} \geq 0 \\ 360° - arccos(\frac{\overrightarrow{u} \cdot \overrightarrow{v}}{|\overrightarrow{u}| \cdot |\overrightarrow{v}|}) & \text{, otherwise} \end{cases}$$

Tiebreaker

To avoid having the same score, a value between $(0, 0.5)$ is added to the angle score. The value is obtained in the same way as in the calculation of the previous measuring unit 5.2.1.

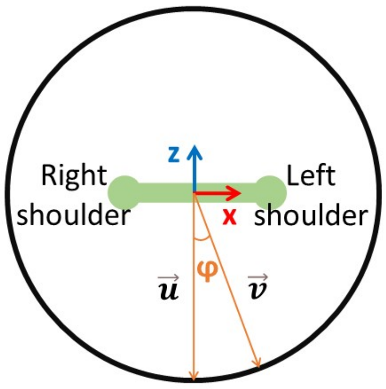


Figure 5.4: φ Angle

5.3 Distributing the tasks

By far the most resource-intensive task of the simulation is rendering the arm model, a task that has to be performed by each device. However, there are other tasks which can be performed by one device that shares the results with the other connected devices. In this section the two tasks of the simulation that will be distributed are presented. Going forward, for each task, the device that performs the task will be referred to as the main device, while the other devices on the network will be referred to as the peers. The criteria for choosing the main device was described in the sections 5.2.1 and 5.2.2.

5.3.1 Distributing the tracking

Tracking the left arm of a person is only accurate from certain angles. For this reason the positions from only one device can be used, specifically the one with the best view of the arm. Two approaches to distribute the tracking are explored, first using an anchor point, and second using a joint that can be tracked from all angles.

Anchor point

All the devices running the simulation need the positions of the shoulder, elbow, and wrist, which poses a problem. This problem can be solved by having the main device track the body, as well as an anchor point. The peers only have to track the anchor point, which is expected to be expensive in terms of computational load compared to body tracking. This is illustrated in 5.6.

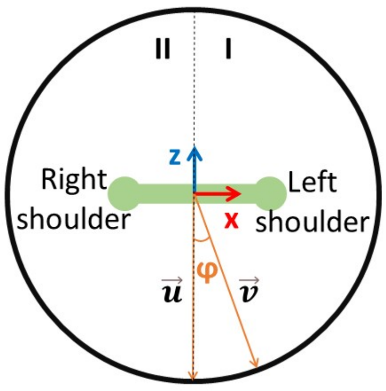


Figure 5.5: Determining φ

If all devices agree on the position of a common point, the main device can calculate the position $\overrightarrow{r}$ of a joint relative to the anchor point by subtracting the anchor position $\overrightarrow{a_1}$ from the joint position $\overrightarrow{j_1}$:

$$(5.7)\quad \overrightarrow{r} = \overrightarrow{j_1} - \overrightarrow{a_1}$$

It then sends this position to the peers. The peers transform the received position from anchor relative coordinates to world coordinates by adding the relative position $\overrightarrow{r}$ to the anchor position $\overrightarrow{a_2}$, resulting in the joint position $\overrightarrow{j_2}$:

$$(5.8)\quad \overrightarrow{j_2} = \overrightarrow{a_2} + \overrightarrow{r}$$

Repeating this process for each joint allows the peers to obtain the joint positions without the need to track the human body. To implement this, a way to determine and track an anchor point is needed. The implementation relies on the *ARCollaborationData* ARKit class. An object of this class holds session information about the environment. The data includes the poses of participants and feature points. Feature points represent notable features in the environment which are used to determine a device's location in the world, such as a knot in a wooden table. A AR Point Cloud Manager component creates point clouds, which are sets of feature points [7]. The point clouds are included in the collaboration data. Finally, an AR Anchor Manager component [6] is used to create an anchor. Anchors are particular points in space which the device should track. They are also included in the collaboration data. For this part of the implementation, an example from ARFoundation samples [8] has been used and built upon.

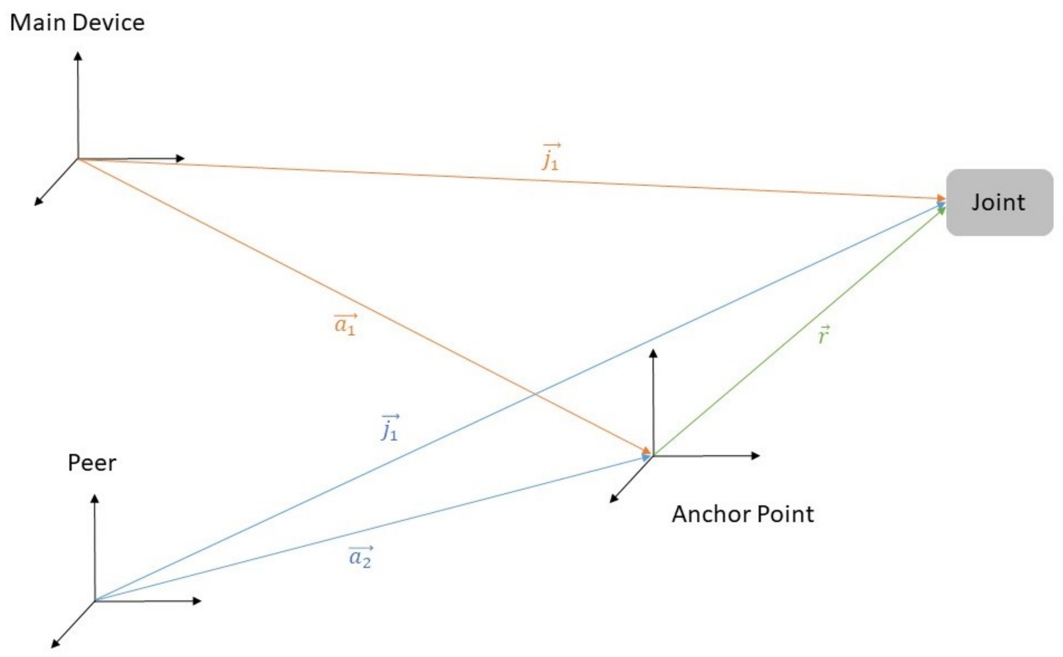
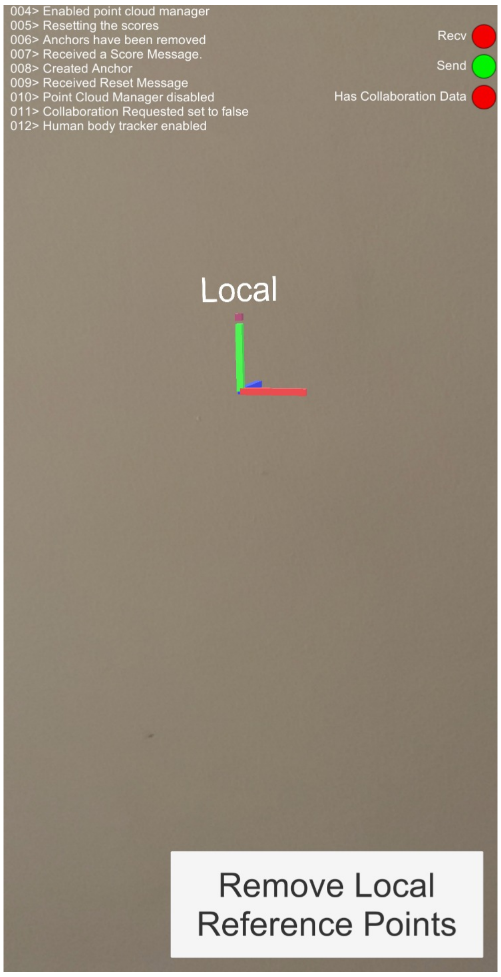


Figure 5.6: Tracking distribution

When the application is first started on a device, it runs locally. If another device joins, the multipeer session is established and either the score or the angle score is calculated and shared between the devices. Once the main device is determined, it creates an anchor 2 meters in front of where the application was initialized. Simultaneously, the two devices start sharing collaboration data. For this reason, the users have to rotate the camera around in order to track the environment. After a while, the peer has enough data to position the anchor. The anchor on the main device is labeled as local, while the one on the peer is labeled as remote, shown in the figures 5.7 and 5.8. Body tracking and environment tracking are not possible at the same time. Because of this, all devices disable body tracking once they are in a multipeer session. Once the peer have placed the remote anchor, they send a message to the main device. The main device keeps track of the received messages until their number matches the number of peers in the session. At this point, all peers have now placed the anchor point, and the main device enables the body tracking. The next step is that all devices disable the environment tracking. It is necessary for the main device to disable environment tracking in order to track the body. If a person is detected, the main device starts sharing the relative positions of the left arm, which the peer uses to update the model position and calculate the activation values used to visualize the muscle contraction. However, if the environment tracking is disabled, the anchor position does not get updated anymore, which leads to the anchor changing its position, and consequently to inaccuracy.

Root joint

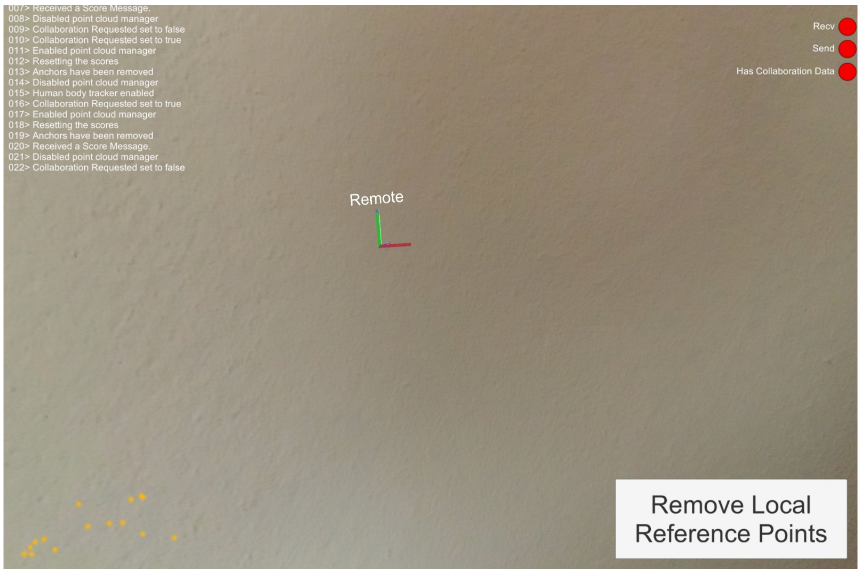
An approach to combat the issue of environment and body tracking not being possible at the same time is using the root joint position instead of an anchor. As stated earlier, the root joint is a body joint that represents the center of the human body and can be tracked accurately from all angles.



```
004> Enabled point cloud manager
005> Resetting the scores
006> Anchors have been removed
007> Received a Score Message.
008> Created Anchor
009> Received Reset Message
010> Point Cloud Manager disabled
011> Collaboration Requested set to false
012> Human body tracker enabled
```

Figure 5.7: Local anchor

All devices track the body, and the device with the highest angle score, as described in 5.2.2, shares the joint positions relative to the root. The root-relative position of a joint can be calculated in the same way as the anchor-relative positions by using the root position instead of the anchor position in the formula 5.7. In the implementation, the Unity method *Transform.InverseTransformPoint* is used to obtain the relative positions. This method transforms a given position from world coordinates to the local coordinates of a certain transform object, in this case the transform object of the root joint. The peers use the received positions to calculate the joint positions based on where they track the root. The same formula 5.8 as before can be used, replacing the anchor position with the root position. In the implementation, the Unity method *Transform.TransformPoint* is used, which performs the opposite conversion as the method mentioned before, namely it transforms a given position from the local coordinates of a transform object to world coordinates.



```
007> Received a Score Message.
008> Disabled point cloud manager
009> Collaboration Requested set to false
010> Collaboration Requested set to true
011> Enabled point cloud manager
012> Resetting the scores
013> Anchors have been removed
014> Disabled point cloud manager
015> Human body tracker enabled
016> Collaboration Requested set to true
017> Enabled point cloud manager
018> Resetting the scores
019> Anchors have been removed
020> Received a Score Message.
021> Disabled point cloud manager
022> Collaboration Requested set to false
```

Figure 5.8: Remote anchor

To showcase this approach, an implementation based on the color application is developed. The distribution criteria is the angle score. The accuracy issue with the anchor point approach, stemming from the inability to track both the environment and the body at the same time, was already noticed during the implementation phase of the bachelor thesis. For this reason, only the root joint approach is used going forward. This is also the only tracking distribution implementation included in the experiments which will be evaluated. A description of the experiment for evaluating this approach is provided in chapter 6.

5.3.2 Distributing the vertex coordinates

The first task whose results are distributed is the tracking. The second task is calculating the coordinates of the vertices necessary for simulating the biceps deformation. This task can be performed by one device, since with ideal tracking the muscle deformation would be the same regardless of the device running the application and of its position in relation to the body. The results of the first NN which calculates the activation values can also be shared. However, this NN is very light-weight in terms of computational costs, consequently the activations are calculated by each device locally.

Sharing the position of a vertex requires 12 bytes, as there are three coordinate values, each stored in one float variable. Multiplying this with 2809 vertices results in 33.708 bytes, or approximately 33 kilobytes necessary to share the vertices in one given frame. So as to achieve 30 Frames per Second (FPS), the required network bandwidth is almost 990 KB/s, while for 60 FPS this number jumps to 1980 KB/s. This represents a massive bandwidth demand which is not feasible given the constraints of the application and the characteristics of typical network hardware. To further confirm this, a test of multipeer transfer speed based on an application from a public GitHub repository [27] was performed using two iOS devices connected to a Wi-Fi network with Bluetooth on. The result was a transfer speed between approximately 1000 and 1200 KB/s. For this reason, the implementation with the 30 vertices is used. For this implementation the bandwidth requirement for 60 FPS is of only 720 bytes/s.

Two implementations for distributing this task are developed and evaluated, one based on the device score and the other based on the angle score. They are very similar to each other, only differing in the criteria for the decision-making. The experiment setup for evaluating them is described in chapter 6.

5.4 Communication

All communication between devices happens through Multipeer. For this reason, a message class is defined as in listing 5.2. Messages are converted to the JSON format [21].

```
public class Message
{
    public MessageType Type;
    public float Score;
    public Vector3[] Positions = new Vector3[3];
    public float[] Vertices = new float[5];
    public int Step;
}
```

Listing 5.2: Message class

Each message contains a type, as defined in the listing 5.3, which determines what fields of the message are set and how the information in the message is used by the receiving device. A Reset Message does not contain any other information. Its role is to trigger a recalculation of the score and thus an update of the score list. When a device joins a session for the first time, it sends a reset message to all devices in the network. A Score Message contains a float which represents the sender score, be it the score calculated with the scoring function or the angle score. In addition it contains the int Step, which keeps track of the frames. Tracking Data Messages contain an Array of vectors containing coordinates, as well as the Step. A Message of type Vertices contains the positions of 30 Vertices. Sync Messages contain the Step, which is used for determining the delay, as detailed in section 5.4.1.

```
public enum MessageType
{
   Reset, Score, TrackingData, Vertices, Sync
}
```

Listing 5.3: Message Types

5.4.1 Delayed Messages

Figure 5.9 exemplifies the communication between a main device and a peer. After the application is initialized, every device keeps track of the step it is at - a variable that starts from zero and is incremented by one in every frame. The main device sends a Sync message periodically, for example every δ steps. The message includes the current step the main device is at. When receiving a Sync message, the peer compares its own step to the received one and calculates their difference, which is the offset o. When a message of another type is received, the peer uses the offset to calculate the undelayed step, the step at which the main device should have sent the message, assuming the offset did not change. The delay is the difference between the undelayed step and the message step.

undelayed step = peer step + o
delay = undelayed step − message step

The delay is used to determine how recent the information is. If the delay is greater than 30, meaning the information is older than 0.5 s, the peer calculates the result locally. Similarly, if no message is received in a frame, the peer calculates the result itself.

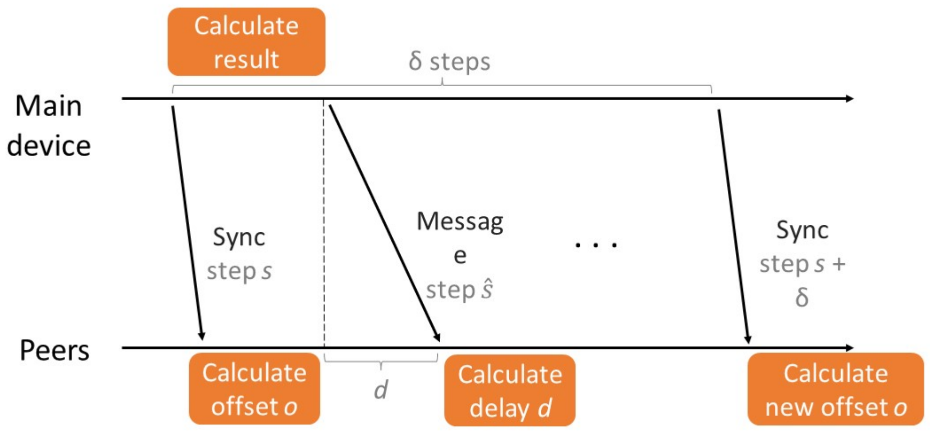


Figure 5.9: Delay calculation

6 Evaluation

This chapter describes the hardware which was used, as well as three experiments which were set up to evaluate the different approaches introduced in the previous chapter. The experiment results are then presented and interpreted.

6.1 Hardware

Two devices are used for testing and evaluating the implementations, an iPhone and an iPad. Their hardware specifications are described in the next paragraphs.

6.1.1 iPad Pro

The iPad Pro is a premium line of tablet computers developed by Apple Inc. The specific model used in this thesis is iPad Pro 12.9-inch (3rd generation) that was released in 2018 and comes with an Apple A12X Bionic chip and 4GB of RAM [18]. The A12X is a System on a Chip (SoC) that offers 8 cores which run at a maximum of 2.5GHz, and includes a 7-core GPU, as well as dedicated NN hardware, an 8-core Neural Engine [4]. The device uses the iPadOS Version 15.7 mobile operating system.

6.1.2 iPhone 12 Pro

The iPhone 12 Pro, released in 2020, is the flagship device of the iPhone line of smartphones developed by Apple Inc. It has an Apple A14 Bionic chip and 6GB of RAM [19]. The A14 SoC provides 6 cores that run at 1.8 - 3.1 GHz, and includes a 4-core GPU and a 16-core Neural Engine [5]. The iPhone uses the iOS Version 15.6.1 mobile operating system.

6.2 Distributing the tracking by angle score

The tracking distribution is done using the angle score measuring unit, introduced in 5.2.2. The root joint is used as the reference point for the joint positions. The design and implementation of the approach evaluated in this experiment is described in the section 5.3.1.

6.2.1 Experiment Setup

A person stands in the middle of the room, performing a flexing motion of the left arm. The iPad is set in place in a position where it has a good view of the left arm, meaning the angle score is 5. A second person holds the iPhone, pointing it at the first person, starting from a position where the left arm is not visible and the angle score is 1. The person holding the iPhone slowly rotates around the tracked person until the angle score is 5. The angle scores are given without the tiebreaker score.

6.2.2 Results

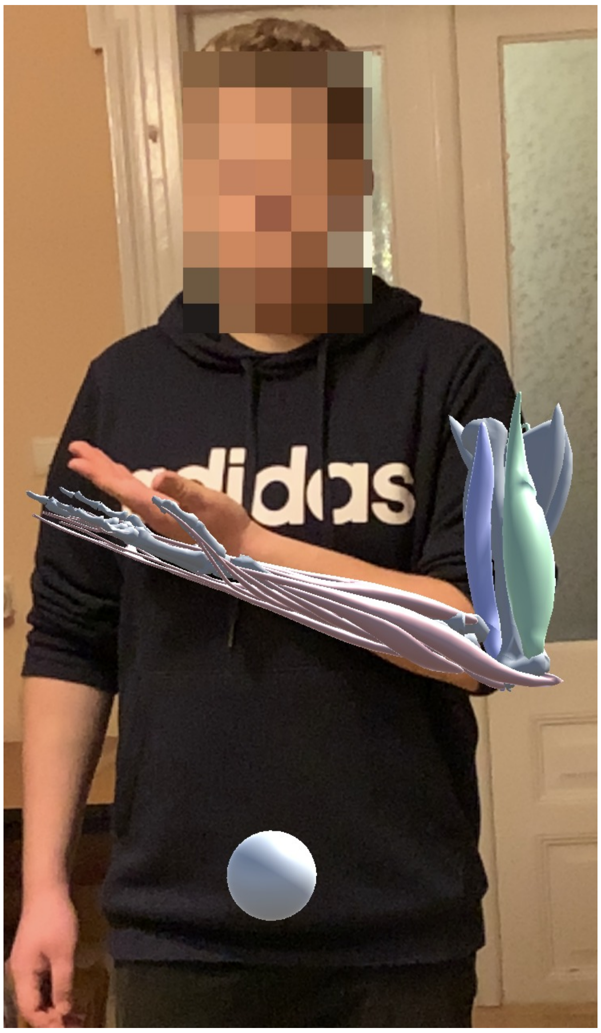


Figure 6.1: Distributed tracking front view

This approach allows an accurate visualization of the arm for a device that has a low angle score, even when the whole arm or some of the joints are not visible to a peer, for example if the tracked person has their back to the camera, on the condition that a main device is in a better position to track the arm.

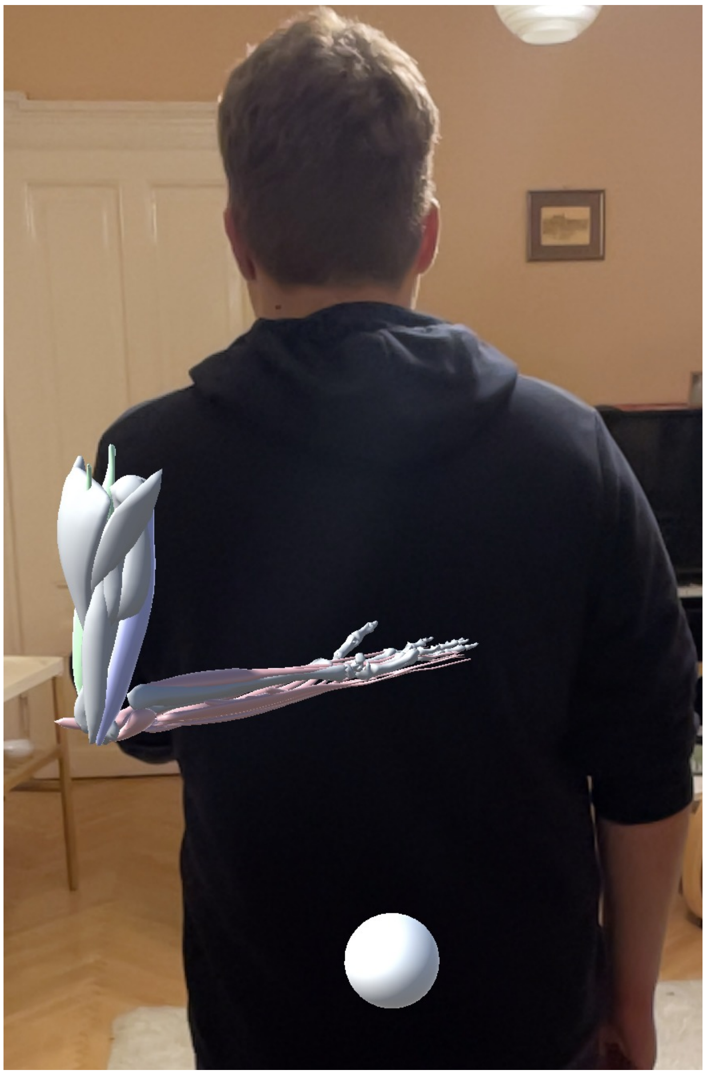


Figure 6.2: Distributed tracking back view

A screen capture from a device that has an angle score of 3 and is pointed at a person flexing their arm is shown in figure 6.1. The white sphere shows the root position, which can be tracked from any angle as long as the body is visible. Figure 6.2 shows a screen capture from a device with an angle score of 1. This device obtains the joint positions relative to the root position from the device with the angle score of 3. It then uses the received positions to update the arm model, which is placed in the correct position even though the wrist is not visible from this angle.

Figure 6.3 represents a graph of the angle between the position vectors as tracked by the iPhone, and the position vectors calculated in reference to the root joint, recorded over 1000 steps. The angle shows how close the position tracked by the iPhone and the position tracked by the iPad are. Overall, a downward trend for the angles can be noticed. At first, when the iPhone has an angle score of 1, the angle for each joint is at the maximum, taking values of around 3.4° for the shoulder, 3.1° for the elbow, and 3.2° for the wrist. As the iPhone is moved and gets a better angle score, the angles are decreased, until the iPhone and the iPad are in a similar position, which is shown starting from step 800. In this part of the graph, the shoulder angle is approximately 0.6° on average, the elbow angle is around 0.4° on average, while the wrist angle varies between a high of 1.2° and a low of 0.4°.

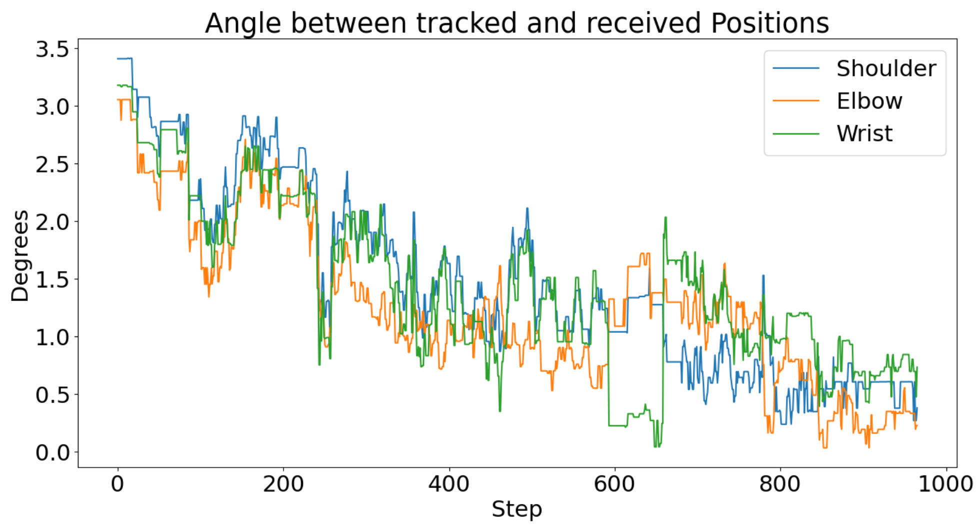


Figure 6.3: The angle between tracked and received positions

While the angles get closer to zero, at the final steps in 6.3, when both devices have a score of 5, the positions tracked by the peer and the positions the peer receives are still different from each other. The fact that the joint positions are different from each other at the end can be attributed to the fact that the iPhone and the iPad are not in an identical position in relation to the body. However, as the two devices get in a more similar position, the joint positions get more similar as well, which suggests that the tracking accuracy is improved when the iPhone has a low angle score. Tracking distribution works the best when the main device has a good angle score, such as 5, while the peers have a bad angle score, such as 1. When both devices are close to each other, the peer tracks the arm better by itself.

6.3 Distributing the vertex coordinates by angle score

The distribution of the vertex coordinates is described in section 5.3.2. In this experiment, the distribution based on the angle score is evaluated.

6.3.1 Experiment Setup

Similar to the previous experiment, a person stands in the middle of the room, performing a flexing motion of the left arm. The iPad is set in place in a position where it has a good view of the left arm, meaning the angle score is 5. A second person holds the iPhone, pointing it at the first person, starting from a position where the left arm is not visible and the angle score is 1. The person holding the iPhone slowly rotates around the tracked person until the angle score is 5. Because of the tie breaking score, even when the iPhone has an angle score of 5, it still receives the vertex coordinates from the iPad.

6.3.2 Results

To evaluate the vertex distribution, the peer calculates two vertices. The first is the average vertex of the 30 points it receives from the main device, while the second is the average vertex of the 30 points it calculates locally. The average vertex is calculated through component-by-component addition of the 30 vertices, followed by dividing each component by 30. The angle between the two vertices is used in order to quantify their similarity. This angle is shown in figure 6.4 for almost 1000 steps.

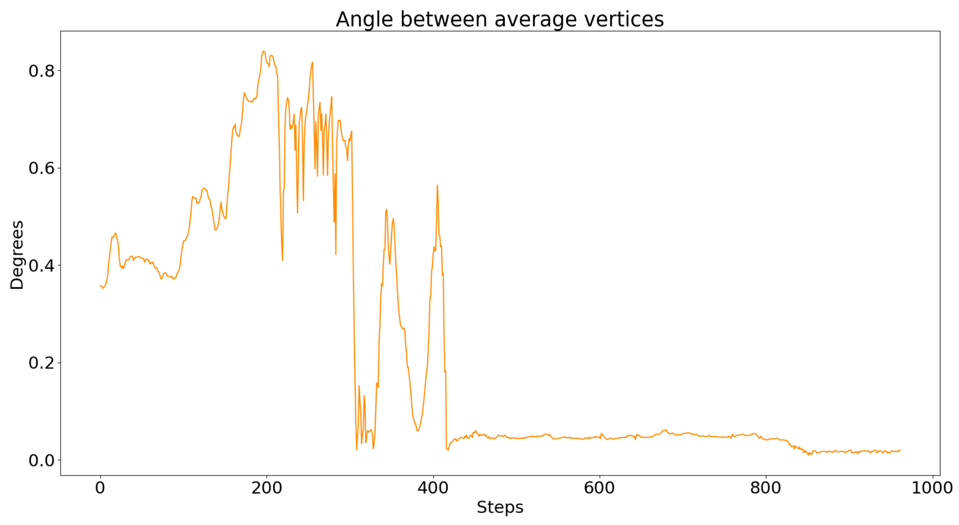


Figure 6.4: The angle between the average vertices

The largest angles, with a maximum of 0.84°, are between step 0 and step 300, when the iPhone has an angle score of 1. Between steps 300 and 400, it has an angle score of 3. In this part there are some large jumps, the lowest point being at 0.03° and the highest at 0.56°. These jumps can be attributed to the fact that the tracking is sometimes inconsistent during motion. Starting from step 415 up until step 800, the angle between the two average vertices is relatively constant around 0.04°. This coincides with an angle score of 4. From step 800 onwards is when the iPhone has the best view of the arm and an angle score of 5. As the iPhone gets closer to the iPad, the average vertices are very similar, with an angle between them of only approximately 0.02°. This shows that as the two devices get nearer to each other, the positions of the vertices they calculate are getting increasingly similar. Relying on the assumption that the most accurate results are calculated by the device with the best angle for tracking, which is determined through the angle score, it is advantageous for the peer to use the results obtained from the main device, because it is very similar to what the results the peer would obtain from the same angle as the main device.

The vertex coordinates are calculated based on the five activation values, which only depend on the weight, the angle, the angular velocity, and the angular acceleration. For this reason, even if the devices estimate the real world positions of the joints differently, the activation values are similar or the same if the angle and the angular velocity and acceleration are similar. This means that if the position estimations by two devices are different, but at each frame the angle between them is the same, the activation values will also be the same, as will be the vertex coordinates.

6.4 Distributing the vertex coordinates by device score

A third experiment is designed in order to evaluate the distribution of the vertex coordinates based on the device score.

6.4.1 Experiment Setup

For this experiment, a person's motion was recorded from two angles, one recording being from 45° to the left and the other from 45° to the right. The person walks around while moving their arms in different ways. In the recordings, the left hand is visible at some points and obstructed at others, as the person turns around. The devices are set in a fixed location with the camera pointed at a screen playing the two recordings, which are synchronized for 30 seconds. The iPad is pointed at the recording from the left, while the iPhone is pointed at the recording from the right. To measure CPU usage on the iPad, the recordings are looped for 3 minutes. First, the application runs on both the iPad and the iPhone, then only on the iPad. To measure the energy usage on the iPad, the recordings are looped for 10 minutes. First, the application runs on both the iPad and the iPhone, then only on the iPad. To measure the computation time of the NN inference for the 30 vertices, the application runs locally on both devices. Before the start, the devices have been charged battery level of 75%, with the exception of the last evaluation, in which the local simulation runs on the iPhone twice, once normally at a battery level of 75%, and once in Low Power Mode at a battery level of 15%.

6.4.2 CPU usage

Figure 6.5 shows a comparison of the CPU usage of the local and the distributed deformation application with the reduced network on the iPad. The figure is the result of merging two charts from the Xcode debug CPU gauge. The CPU usage in Xcode is shown as a percentage that represents the addition of the usage of all cores. Since the iPad has 8 cores, the shown value is out of 800%.

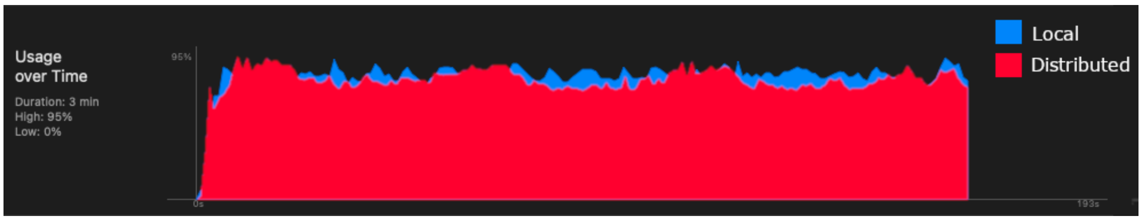


Figure 6.5: iPad CPU usage comparison

The usage of the distributed application is shown in red, while the usage of the local application is shown in blue. Both hit a maximum of 95%, however the distributed CPU usage is slightly lower on average. This difference can be attributed to the fact that in the distributed version of the application, the iPad does not calculate the 30 vertices using the neuronal network, receiving them from the iPhone instead. However, the fact that the difference is so small, and that at some points the CPU usage on the local application is very close or even exceeds the usage of the distributed application, shows the fact that performing inference using the NN is not resource intensive compared to other processes used for this simulation.

6.4.3 Energy usage

Figures 6.6 and 6.7 show the energy usage on the iPad. The figures are screen captures of the Xcode energy usage debug gauge. The energy usage of the local simulation is lower than the energy usage of the distributed simulation.

This can be attributed to the fact that, while the CPU usage for the distributed simulation is slightly lower, the energy that would be saved by the iPad not having to calculate the 30 vertices is insignificant and certainly not enough to offset the supplementary energy usage stemming from distributing the simulation. Distributing the simulation introduces additional overhead, such as the devices using the Wi-Fi module to communicate and to constantly search for peers. Running the reduced NN has a minor impact on both the energy usage and the CPU usage, as shown in section 6.4.2. Other tasks and services, such as the ones needed for the distribution, or rendering the model contribute far more to the energy usage.

6.4.4 Computation Time

At a battery level of 75%, the iPad has a device score of 65, while the iPhone has a device score of 76. The device score for the iPhone at 15% battery level is 61. These scores are given without the addition of the tiebreaker value. The iPhone was released more recently, it has a newer and better performing CPU, and performs better in benchmarks, which explains the higher score.

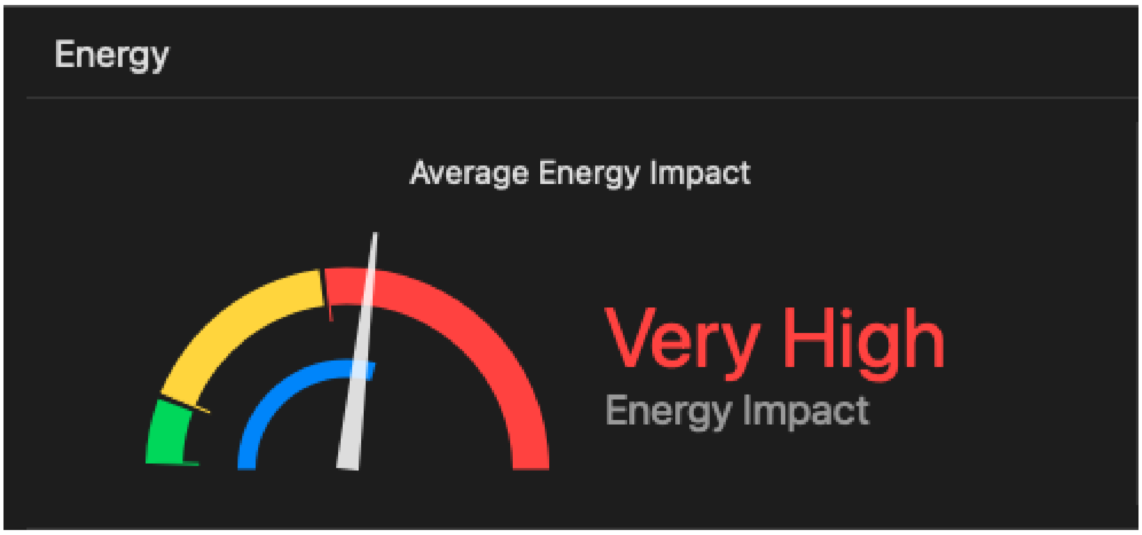


Figure 6.6: iPad energy usage local

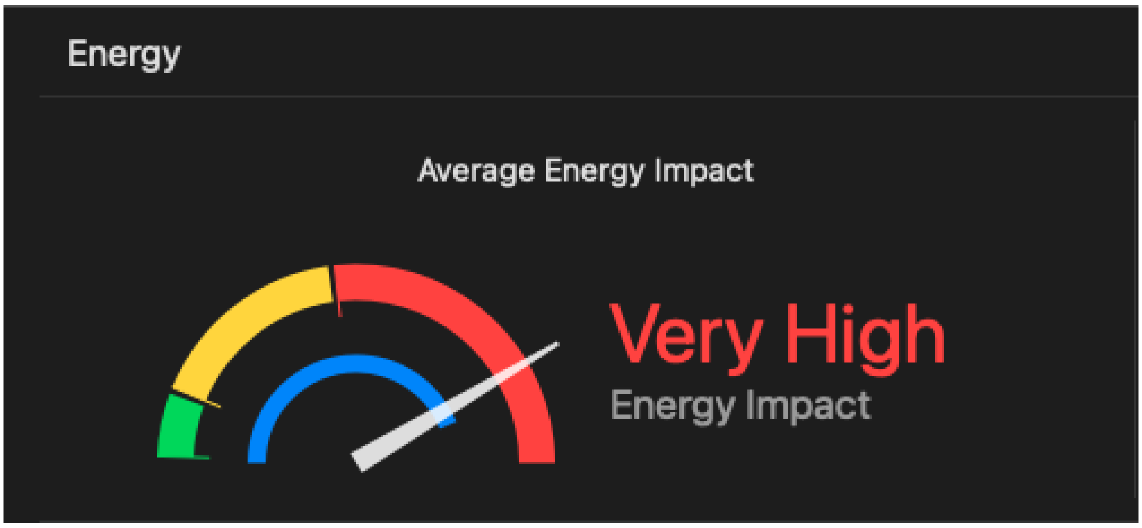


Figure 6.7: iPad energy usage distributed

A comparison between the computation times for the first 1000 steps of running the local simulation is shown in figure 6.8. The computation time is obtained by calculating the difference of the time at which the inference is started and the time at which it is completed, and it is measured in milliseconds. As expected based on the device scores, the iPhone performs better than the iPad under normal circumstances. When it is in Low Power Mode, it performs worse than the iPad. The average computation time across the 1000 steps is 0.328 ms for the iPhone, 0.575 ms for the iPad, and 0.605 for the iPhone in Low Power Mode, as illustrated in figure 6.9. The device with the highest device score performs the best, while the lowest score corresponds to the lowest performance.

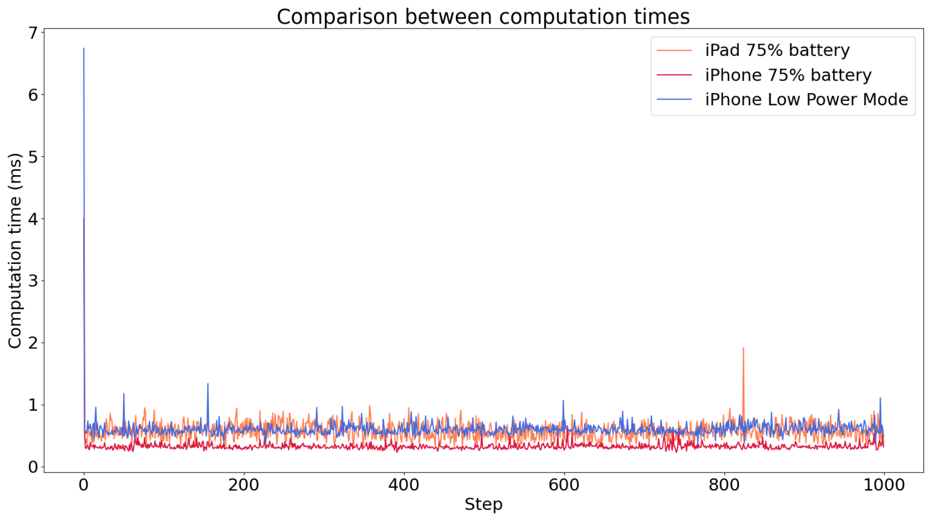


Figure 6.8: Computation times comparison

The iPhone with a battery level of 21% has a score of 66, while the iPad with a battery level of 100% has a score of 71. In this case, the iPhone is not in Low Power Mode and performs better than the iPad, even though it has a lower device score. However, because the battery level of the iPhone is relatively low, it is still preferable that the iPad performs the additional computations. If energy usage were not a factor, the battery score could have been calculated using 5.2 instead of 5.1.

Thus, for the two devices used, the device score is able to predict which device will perform the inference better in the three scenarios. The data points of this experiment are however restricted by the number of devices used, which is a limitation of the present work.

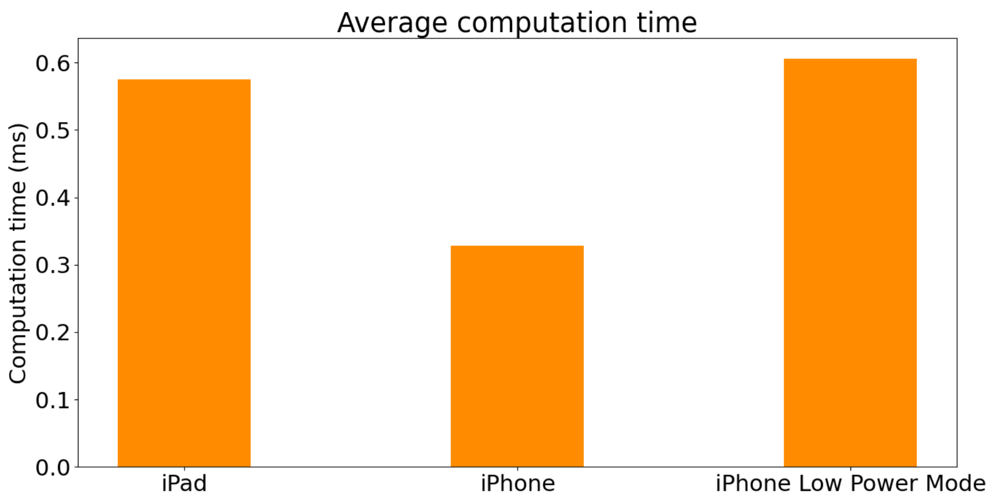


Figure 6.9: Average computation time

7 Conclusion and Outlook

The goal of this bachelor thesis is to explore methods for the Peer-to-Peer (P2P) distribution of a mobile simulation. The starting point is an iOS muscle visualization application which superimposes an arm model onto the arm of a person observed through the camera. The application tracks the person and, using the positions of the joints in the left arm, shows the contraction of arm muscles either through colors or by changing the deformation of the arm model. There are different tasks that have to be performed in order to run this simulation. One important constraint is the fact that the application has to run in real-time. For this reason, all necessary tasks have to be completed in each frame. This means that the tasks themselves can not be partitioned and distributed, as that would result in the need to communicate multiple times in each frame, which would increase the network requirements and the energy usage significantly. Instead, a task is assigned to a certain device, which performs it and shares the results. Consequently, there needs to be a way to compare multiple devices and decide which device should perform a task. For this purpose, two measuring units are introduced, an angle score and a device score. The angle score evaluates how good the position in which a device finds itself in relation to the tracked person is for tracking the left arm specifically, while the device score evaluates the hardware specifications and the battery level. The devices are connected to each other in an unstructured P2P network and as such, there is no central entity responsible for decision-making. Each device maintains a list of the scores of other devices in the session, which allows them to decide individually if they should perform a task or not. There are two tasks whose results are distributed. First, the task of tracking the person, and second, the task of calculating the vertex positions of a mesh used to show the deformation through a NN.

The first experiment evaluates the distribution of the tracking task. The decision of which device should perform the tracking is made using the angle score. The angle score is calculated by a function that assigns a score to each device based on the angle it finds itself in relation to the tracked person. This approach allows the visualization of the arm model from angles in which the left arm is not visible if another device is at a better angle and shares the joint positions. This is the scenario in which the tracking distribution works best. In the case that both devices are at a similar angle, the results are better if each device performs the tracking itself. Going back to the first point of the problem statement, this approach improves the accuracy of the results for devices that are not in a position with a good visibility of the left arm. The task of calculating the vertices is distributed using both the angle score and the device score, resulting in two additional experiments. The first implementation of the vertex distribution uses the angle score. A device that is located at a bad angle is able to obtain accurate vertex positions from a device that is at a better angle. When the two devices are in a similar position, the vertex positions they calculate are also similar. Hence, distributing the vertex task results in improved accuracy if the distribution is done based on the angle score, which fulfills the requirement set in the first point of the problem statement. The second implementation is very similar, the only difference being the fact that the device score is used in lieu of the angle score. The device score can predict which device out of the two tested will perform this task better, including when the device with the highest score normally is in Low Power

Mode. The performance is measured using the computation time for each query. The CPU usage of the device with the lower score is slightly reduced by distributing this task, but this comes at the cost of increased energy usage, caused by the additional services needed for the distribution. The mobile devices are powerful enough to where running the reduced NN does not significantly affect the energy usage or the CPU usage. For the application used in this thesis, the reduced CPU usage is not enough to offset the additional energy cost. The distribution costs only vary depending on the message size, and would be constant for two different applications, provided their network usage would be the same. Thus, if a NN would take up a large part of the computing resources of a device, sharing its outputs would indeed be advantageous and result in reduced energy consumption for the other peers in the network. Nevertheless, as a proof-of-concept, this experiment shows that distributing the simulation reduces the computational cost for the devices that do not perform this task, providing an answer for the second point in the problem statement. It is expected that for a different application, in which a more costly NN is used, the additional energy usage caused by the distribution would be offset by the reduced CPU usage.

To conclude the thesis and provide a final answer to the problem statement, the simulation can be distributed on mobile devices in a way that improves the accuracy or reduces the computational costs, but not without the disadvantage of a higher energy consumption caused by the communication. The first and the second experiment show that the accuracy of the results is improved for devices that are not in an optimal position for viewing the left arm. The third experiment shows that, while on the one hand the devices which do not perform the vertex task have reduced CPU usage, on the other hand the distribution leads to increased energy costs. In other words, there is a trade-off between accuracy and energy consumption that has to be taken into consideration before distributing a simulation.

Outlook

There are multiple possible ways to extend the approaches presented in this thesis in future work. The first is implementing the same distribution methods on an application with a more costly NN, which is expected to lead to better results in terms of reducing the computational costs. The second is testing a large number of iOS devices in order to observe how the device score correlates to the performance in different tasks, while optimizing the coefficients in the weighted average of the scoring function based on real world results. The third is training a NN to calculate angle scores using a data set which contains the angle at which a device lies and an assessment of the accuracy from that angle. This would result in a more granular angle score, for example in the range between 0 and 100, allowing for a better comparison between devices. In this thesis only the biceps deformation is simulated; another way to build on the present work is simulating the deformation of five muscles, for each activation value that is calculated, and distributing the NNs that infer the vertices for each muscle to different devices, in such a way that each device is responsible for one muscle. Finally, the P2P distribution could be combined with the approach presented in the research project [23], in which a mobile device runs a reduced NN and periodically requests more accurate activation values from a server that runs a more complex NN. The main device would be the one communicating with the server, as it would be able to send the most accurate input values to the server. The main device would then transmit the result received from the server to the peers and all devices would combine the server results with their local results using the merging algorithm introduced in the project. As a result, the accuracy would be improved for all devices, while at the same time the communication costs would be reduced, since the communication would not happen every frame.

Bibliography

[1] *About Apple ARKit XR Plug-in*. URL: https://docs.unity3d.com/Packages/com.unity.xr.arkit@5.0/manual/index.html#about-apple-arkit-xr-plug-in (cit. on p. 16).

[2] *About AR Foundation*. URL: https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@5.0/manual/index.html (cit. on p. 15).

[3] D. Amin, S. Govilkar. "Comparative Study of Augmented Reality Sdk's". In: *International Journal on Computational Science & Applications* 5 (Feb. 2015), pp. 11–26. DOI: 10.5121/ijcsa.2015.5102 (cit. on p. 13).

[4] *Apple A12X Bionic*. URL: https://www.notebookcheck.net/Apple-A12X-Bionic-SoC.354571.0.html (cit. on p. 43).

[5] *Apple A14 Bionic*. URL: https://www.notebookcheck.net/Apple-A14-Bionic-Processor-Benchmarks-and-Specs.494578.0.html (cit. on p. 43).

[6] *AR anchor manager*. URL: https://docs.unity.cn/Packages/com.unity.xr.arfoundation@4.2/manual/anchor-manager.html (cit. on p. 36).

[7] *AR Point Cloud Manager*. URL: https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@3.0/manual/point-cloud-manager.html (cit. on p. 36).

[8] *Arfoundation Samples GitHub Repository*. URL: https://github.com/Unity-Technologies/arfoundation-samples (cit. on pp. 15, 29, 36).

[9] *ARKit Documentation*. URL: https://developer.apple.com/documentation/arkit (cit. on p. 16).

[10] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, B. MacIntyre. "Recent advances in augmented reality". In: *IEEE Computer Graphics and Applications* 21.6 (2001), pp. 34–47. DOI: 10.1109/38.963459 (cit. on p. 13).

[11] H. Ba, W. Heinzelman, C.-A. Janssen, J. Shi. "Mobile computing - A green computing resource". In: *2013 IEEE Wireless Communications and Networking Conference (WCNC)*. 2013, pp. 4451–4456. DOI: 10.1109/WCNC.2013.6555295 (cit. on p. 24).

[12] *Class ARHumanBodyManager Documentation*. URL: https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.1/api/UnityEngine.XR.ARFoundation.ARHumanBodyManager.html (cit. on p. 16).

[13] *Geekbench 5*. URL: https://www.geekbench.com/ (cit. on p. 21).

[14] *Geekbench iPhone, iPad, and iPod Benchmarks*. URL: https://browser.geekbench.com/ios-benchmarks (cit. on p. 21).

[15] *Geekbench ML*. URL: https://www.geekbench.com/ml/ (cit. on p. 21).

[16] *Geekbench ML Benchmarks*. URL: https://browser.geekbench.com/ml-benchmarks (cit. on p. 21).

[17] *iOS Devices - Memory Mark Rating*. URL: https://www.iphonebenchmark.net/memmark_chart.html (cit. on p. 21).

[18] *iPad Pro 12.9-inch (3rd generation) - Technical Specifications*. URL: https://support.apple.com/kb/SP785?locale=en_US (cit. on p. 43).

[19] *iPhone 12 Pro - Technical Specifications*. URL: https://support.apple.com/kb/SP831?locale=en_US (cit. on p. 43).

[20] M. James Myhre & Dennis Sifris. “The Anatomy of the Biceps”. In: *Verywell Health* (2022). URL: https://www.verywellhealth.com/biceps-anatomy-4688616 (cit. on p. 17).

[21] *JavaScript Object Notation*. URL: https://www.json.org/json-en.html (cit. on p. 41).

[22] X. Jin, S.-H. G. Chan. “Unstructured Peer-to-Peer Network Architectures”. In: *Handbook of Peer-to-Peer Networking*. Ed. by X. Shen, H. Yu, J. Buford, M. Akon. Boston, MA: Springer US, 2010, pp. 117–142. ISBN: 978-0-387-09751-0. DOI: 10.1007/978-0-387-09751-0_5. URL: https://doi.org/10.1007/978-0-387-09751-0_5 (cit. on p. 14).

[23] D. Lieb, F. Birtar, S. Aydin. “Distributed Neural Networks for Continuous Simulations”. In: *Institute of Parallel and Distributed Systems, University of Stuttgart* (2022) (cit. on pp. 27, 54).

[24] J. Mao, X. Chen, K. W. Nixon, C. Krieger, Y. Chen. “MoDNN: Local distributed mobile computing system for Deep Neural Network”. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. 2017, pp. 1396–1401. DOI: 10.23919/DATE.2017.7927211 (cit. on p. 24).

[25] P. Milgram, F. Kishino. “A Taxonomy of Mixed Reality Visual Displays”. In: *IEICE Trans. Information Systems* vol. E77-D, no. 12 (Dec. 1994), pp. 1321–1329 (cit. on p. 13).

[26] *Multipeer Connectivity Documentation*. URL: https://developer.apple.com/documentation/multipeerconnectivity (cit. on p. 16).

[27] *MultiPeer-Progress-iOS by GitHub user ”shawn-frank”*. URL: https://github.com/shawn-frank/MultiPeer-Progress-iOS/commits?author=shawn-frank (cit. on p. 39).

[28] E. Nwafor, M. Robson, H. Olufowobi. “Dynamic Load Sharing in Memory Constrained Devices: A Survey”. In: *2021 IEEE 7th World Forum on Internet of Things (WF-IoT)*. 2021, pp. 586–591. DOI: 10.1109/WF-IoT51360.2021.9594948 (cit. on p. 23).

[29] PerSiVal. *Pervasive Simulation and Visualization with Resource- and Time-Constraints*. URL: https://www.simtech.uni-stuttgart.de/exc/research/pn/pn7/pn7-1/ (cit. on pp. 11, 14).

[30] P. Rogers. “How to Do Biceps Curls”. In: *Verywell Fit* (2020) (cit. on p. 17).

[31] R. Schollmeier. “A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications”. In: *Proceedings First International Conference on Peer-to-Peer Computing*. 2001, pp. 101–102. DOI: 10.1109/P2P.2001.990434 (cit. on p. 14).

[32] S. Surana, B. Godfrey, K. Lakshminarayanan, R. Karp, I. Stoica. “Load balancing in dynamic structured peer-to-peer systems”. In: *Performance Evaluation* 63.3 (2006). P2P Computing Systems, pp. 217–240. ISSN: 0166-5316. DOI: https://doi.org/10.1016/j.peva.2005.01.003. URL: https://www.sciencedirect.com/science/article/pii/S0166531605000167 (cit. on p. 24).

[33] *Unity*. URL: https://unity.com/ (cit. on p. 15).

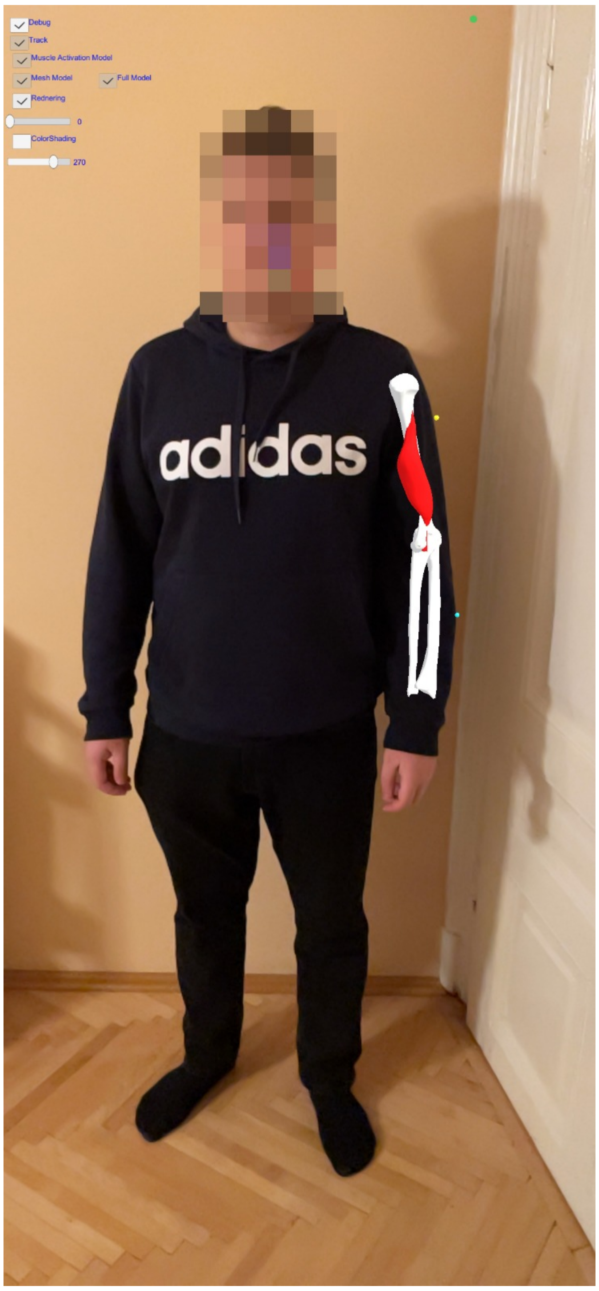
[34] *Xcode Debug Gauges*. URL: https://help.apple.com/xcode/mac/current/index.html?localePath=en.lproj#/devf7f7c5fcd (cit. on p. 15).

[35] *Xcode IDE*. URL: https://developer.apple.com/xcode/ (cit. on p. 15).

[36] *Xcode Instruments Overview*. URL: https://help.apple.com/instruments/mac/current/#/dev7b09c84f5 (cit. on p. 15).

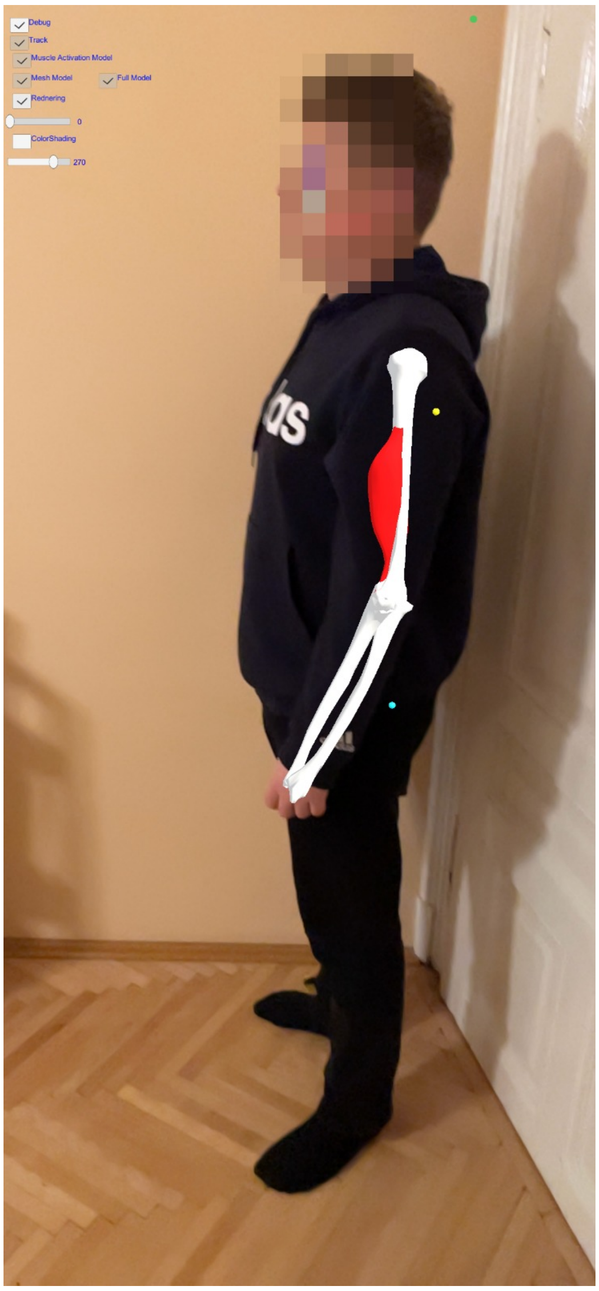
[37] M. Zhong, K. Shen, J. Seiferas. “Dynamic load balancing in unstructured peer-to-peer networks: Finding hotspots, eliminating them”. In: (Jan. 2007) (cit. on p. 23).

A Appendix



- [✓] Debug
- [✓] Track
- [✓] Muscle Activation Model
- [✓] Mesh Model [✓] Full Model
- [✓] Rednering
- 0
- [ ] ColorShading
- 270

Figure A.1: Front, Low



✓ Debug
✓ Track
✓ Muscle Activation Model
✓ Mesh Model ✓ Full Model
✓ Rednering
0
ColorShading
270

Figure A.2: Left, Low

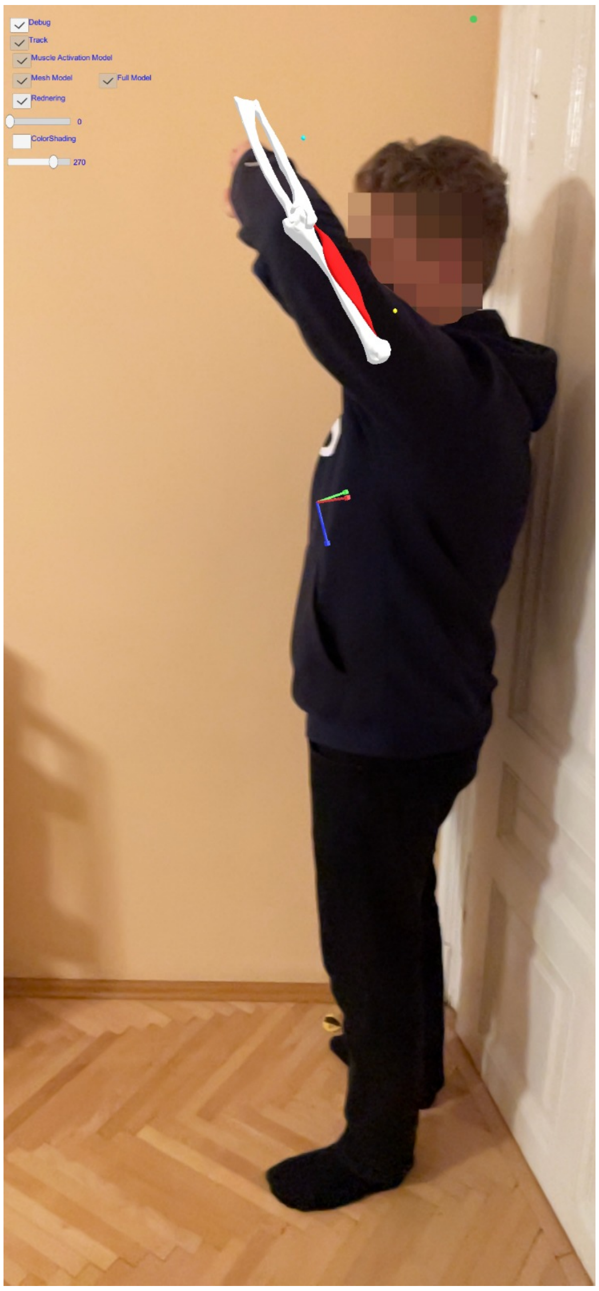
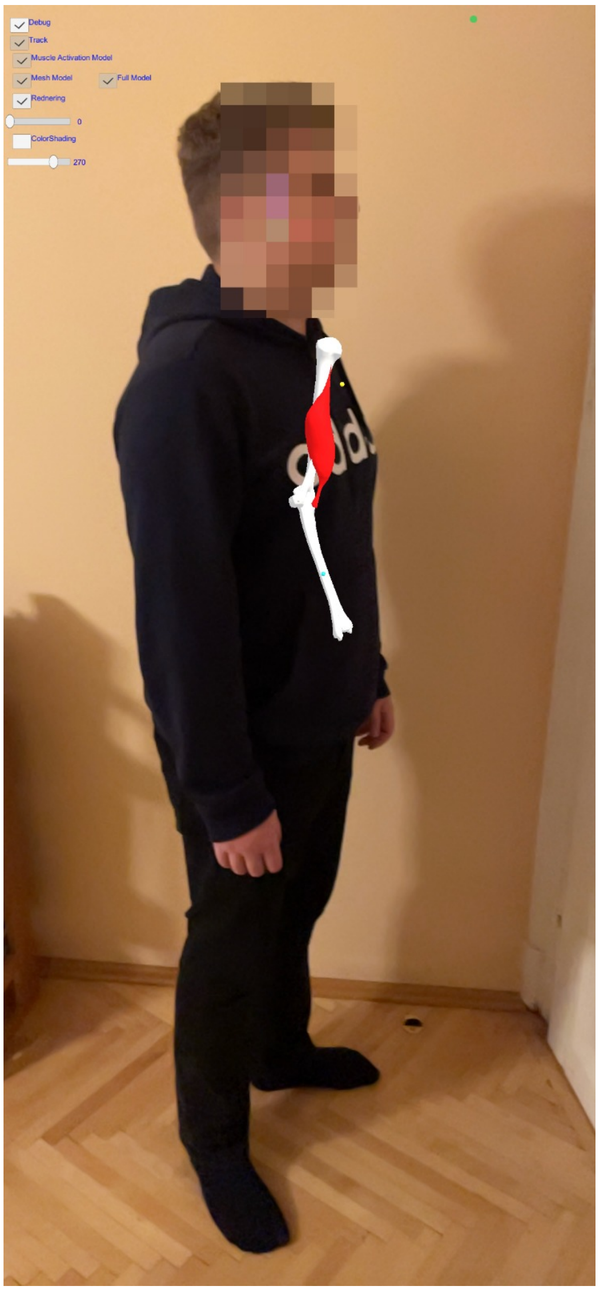
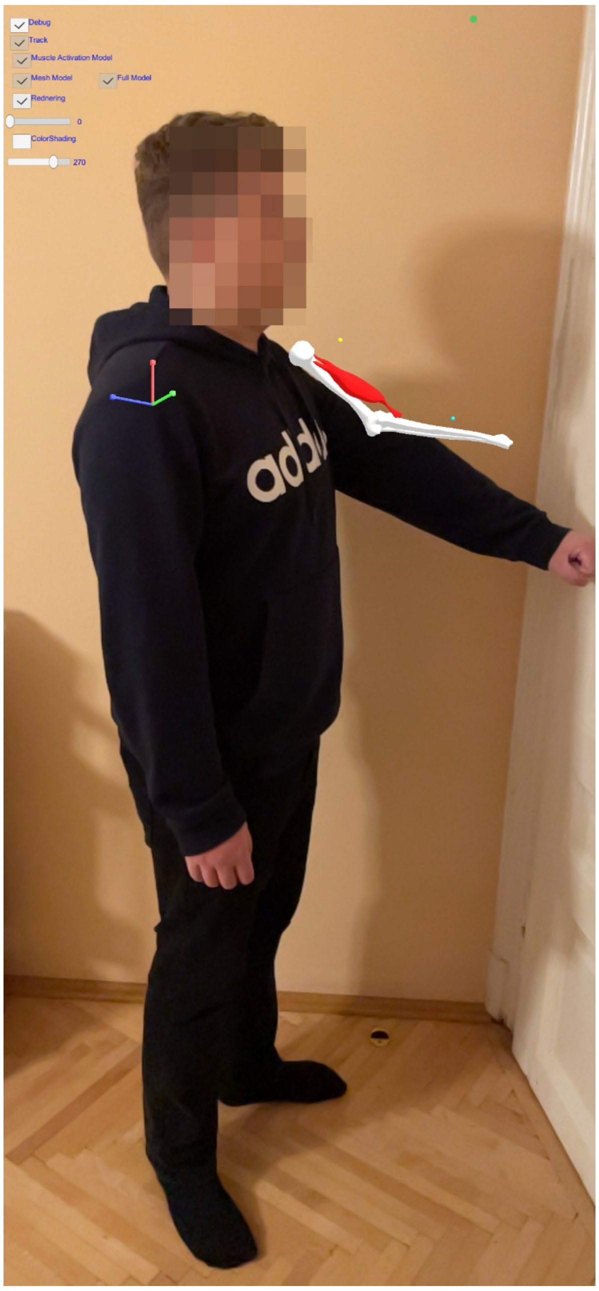

✓ Debug
✓ Track
✓ Muscle Activation Model
✓ Mesh Model ✓ Full Model
✓ Rednering
0
☐ ColorShading
270

Figure A.3: Left, High



✓ Debug
✓ Track
✓ Muscle Activation Model
✓ Mesh Model ✓ Full Model
✓ Rednering
0
☐ ColorShading
270

Figure A.4: Right, Low



✓ Debug
✓ Track
✓ Muscle Activation Model
✓ Mesh Model ✓ Full Model
✓ Rednering
0
ColorShading
270

Figure A.5: Right, Middle

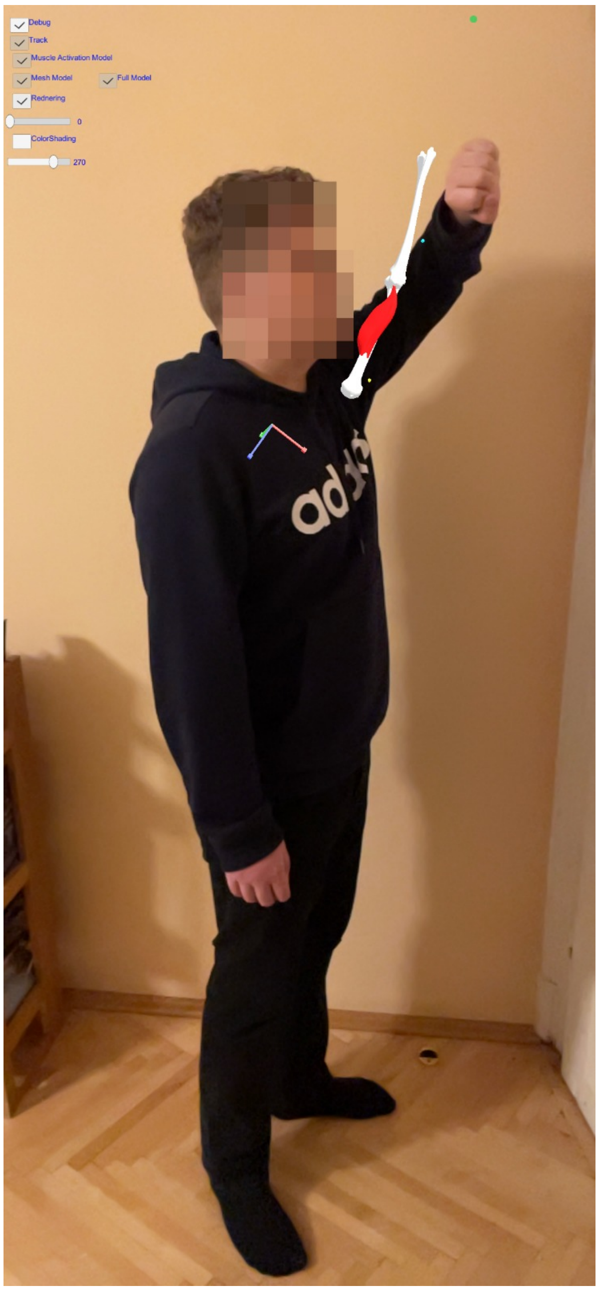

✓ Debug
✓ Track
✓ Muscle Activation Model
✓ Mesh Model ✓ Full Model
✓ Rednering
0
ColorShading
270

Figure A.6: Right, High

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature