Representing ultrasound in Augmented Reality

Hanna Gjöthlén
Salem Lojic
Abstract

Representing ultrasound in Augmented Reality

Hanna Gjöthlén & Salem Lojic

This thesis covers the implementation of an augmented reality (AR) system where ultrasound images are captured from an ultrasound scanner (i.e. probe) and sent to an interface application through a server and then projected into the real world using AR technology. In practice this system will be used by holding a probe on to the body while watching a secondary screen where the ultrasound image will be displayed in the real world. This system makes it easier for students who are new to ultrasound readings to understand them.

This was done by using the game engine Unity along with the AR-engine Vuforia as the base for the AR capabilities. The server connecting the different parts is written in the programming language C# using the WebRTC protocol for sending images over the internet. The ultrasound images were processed using the interface software SimpliVue, Interson Cooperation.

The system performs well in the right conditions (with a frame rate of 26 frames per second) and performs worse in bad conditions (with a frame rate of 5 frames per second). The result shows that there are still improvements to be made in order for the system to be production ready, but the outcome of this project is a proof of concept, and has yet to be evaluated in a real world setting.
Acknowledgments

We want to thank our supervisor Orcun Göksel and subject supervisor Anna Klemm. Without you this would not be possible.
1 Introduction

Ultrasound scanners have been used by doctors for a long time, and the technologies behind them are always improved upon. [1] [2] To further improve the technologies, this thesis will cover the improvement of the user experience of these ultrasounds scans by introducing augmented reality. The first part of this project is to process the information from an ultrasound device. The second part is to send the processed data through the internet to an AR device.

The ultrasound device: Interson is what will be used in this project, along with its own API. The Transmission Control Protocol (TCP) is a data stream protocol and allows us to send and receive information via the internet. [3] It is arranged to send packets over the network and provides a secure delivery of data, in this case, the pre-processed data that will be forwarded to the AR device.

This is where the second part begins. The data first needs to be assembled before it can be projected into the real world using cameras and AR technologies. This will be done in the game engine Unity, which already has some support for AR. Together with Unity, a Software Development Kit (SDK) called Vuforia will be used, which has great functionality for deploying AR apps to the phone and generally good features for testing. [4]
1 INTRODUCTION

1.1 Purpose and Goals

The purpose of this academic paper is to describe the process of developing an AR-system. The system’s job is to process ultrasound data and transfer it reliably through the internet network and generate an AR representation of the ultrasound scan.

The purpose of this project is to process and transmit ultrasound data and to present it using AR-technologies in an intuitive way. The first part of this project is to process the information from an ultrasound device and then to send the processed data through the internet to an AR device.

1.2 Implementation

The final product will be implemented as shown in the figure below. To allow for modulization for every part of the system, different network protocols will be used to communicate between components/programs and to be able to run the different parts on different machines.

1.3 Outline

Section 2 covers background information where the project is divided in two parts. The first part is about sending images from the probe via WebRTC and receiving the streamed images in Unity. The second part is using the received images and then by using position tracking the program displays it in augmented reality. Section 3 covers the implementation of the whole project and testing. Section 4 brings up the results and performance of the final product and also covers some of the difficulties that were encountered during the development. Section 5 discusses the result, difficulties and not yet implemented solutions. The final section gives a summary of the report and concludes the thesis.
2 Background

This section presents tools, technologies, and knowledge that were used in this project.

The first part of the background section is about how the ultrasound software works and how it is used to pre-process the sending of the streams of images via WebRTC. The second part introduces the Unity Editor and how it receives the streamed images and represents them in AR with the Vuforia Engine inside Unity.

2.1 Ultrasound

An ultrasound scan works by sending high-frequency sound waves (higher than human hearing) from a small device which is attached to the body. The device will then recreate an image by measuring the echo of the sound. Since different types of tissues generate a different type of echo, one can construct a series of images displaying organs and tissues. The moving picture can be difficult to interpret for someone that has never seen such a scan before or is still learning to read them.

2.2 Interson

The ultrasound device that will be used is the Interson Probe 3, and it is a scanner that is connected to a computer through USB. It is a device that is portable, unlike many other types of ultrasound devices which require a stationary computer. It is easy to use and easy to install.

2.3 API for the probe

The ultrasound device comes with an API, ArraySDK. This API was then used and modified to be able to send the streamed images through the internet using the network protocol WebRTC. The API, is made for Windows, and is written in the programming language C# and includes two Dynamic-Link Libraries, Interson.dll and Interson-Tools.dll. A dynamic-link library (DLL) is a module that contains functions that can be used by another module.
2.4 WebRTC

Web Real-Time Communication (WebRTC) is an open-source web protocol that can be used between devices, and it sends packages directly to the receiver without storing the package in a server. This protocol allows for a direct peer-to-peer connection for voice- and video-communication to work within applications. WebRTC use UDP to transport data, and it is the base for Real Time Communication, which the figure below shows. [9][10]

MixedReality-WebRTC is a compilation of different libraries to help the development of the network communications of a virtual reality application. [11]

Figure 1 shows the WebRTC network protocol stack [12]

As figure 1 shows, the UDP is the base foundation and at the top the RTCPeerConnection can be found. The RTCPeerConnection works as a webRTC connection between the local computer and a distant component/peer, it maintains and monitors the connection between these two parts and once the task is completed the connection/program/RTCPeerConnection closes.

A Session Traversal Utilities for NAT (STUN) server is a protocol within WebRTC that detects a public address and decides if there are any restrictions of the router that would make it impossible to create a peer connection. As figure 2 below shows, the client (Peer A) will send a request to the server and the server will respond if the client is available or not, and it will also reply with the client’s public IP address. [13]
2 BACKGROUND

Hanna Gjöthlén & Salem Lojic

Figure 2 shows how a STUN server works [13].

The NAT in the STUN server is the Network Address Translation and it gives your machine their public IP address. Every router has a public IP address and every machine that is connected to that router have a private IP address. There is no need for a different public IP for every machine because the requests will be translated from the private IP of the machine to the router’s public IP with an exclusive port and therefore is visible on the internet. [13]

2.5 Unity

Unity is a cross-platform game engine [15] and it has different built-in tools and features. The interesting feature for this project will be Vuforia, a software development kit (SDK) for creating and deploying AR apps on android, iOS, and Universal Windows Platform (UWP) to interact with the environment with the help of the integrated camera realistically. [16]
2.6 AR

Augmented reality (AR) is used to give the user of such a system an interactive experience of the real world by using a computer assisted device, like a phone or a computer. The goal with AR is to produce an experience that is as seamlessly interwoven with the physical world as possible. This concept is different from VR which produces a new reality by replacing the human senses with digital ones (i.e a VR-headset that replaces one’s eyes). The challenges with making AR is to locate where the physical device is oriented in relation to the real world and to incorporate objects in the AR space seamlessly by keeping lightning and proportions in mind. [17]
3 Implementation

The implementation begins with setting up the ultrasound software in order to integrate our code into it. Then the streamed images will be received in unity and the next implementation part is about position tracking and visualization in augmented reality.

3.1 The ultrasound interface

The first step was to download the program "simply view" on a windows operating system to display the streamed images from the probe. The source code from ArraySDK can be downloaded from Interssons official website with an api key. The source code we used in this project was handed to us by our supervisor, which was an older source code which is not supported by Intersson anymore. This was used anyway, since some key features were only used in this version of the source code. The key feature that was used was the support of the older types of ultrasounds.

3.2 The AR application

To begin using Unity, a new project needs to be initiated and set up with some proper settings, such as building targets and packet managing. A building target is a way to tell Unity which platform(i.e operating system and processor architecture) you want to compile the program to. For this project the targeted build is for Windows since no other computer-platform is currently being supported by Vuforia. Vuforia is easily installed through Unity’s own packet managing tool. The second package, mrwebrtc is easiest to install by using Mixedrealality’s own software which automatically set up all the necessary files and assets. When necessary settings have been established, then we are ready to begin using the tools and create the intended program.

3.2.1 Mixed reality WebRTC

Mixed reality’s implementation of WebRTC in unity is very straight forward to use. The main objects to use when setting up a WebRTC-connection are the Signaler-object and the peerconnection-object. Each of these objects has a few required fields to be connected, as seen in the figure (3) and (4). The important fields for the peerconnection object is the media line transceivers, whose task is to give the webrtc framework handlers of the transceiving video and the receiving video (if any) to the other parts of the framework. For this project we only use the tranceiving field since we are only interested in receiving
images. The important fields for the signaler object are the "local peer ID" and the "remote peer ID" fields. These fields are crucial for the framework, since a unique ID is needed for the different peers to be able to properly set up the connection. The last field in the signaler object is the field where the IP address for the server will be.

The above images (figure 3 and 4) shows how a peer connection can be done using the peer connection object and signaler object.
3.2.2 Vuforia

The Vuforia library centers around the game-object "AR-camera". This object is what makes it possible to project 3d objects into the real world using the integrated camera. The AR-camera object is self-contained and does not need any further adjustments when using it in this project.

The most useful tool from the vuforia library/engine for this project is the built-in real time tracking capabilities when using the Image target-object, as seen in figure 5. There are many things to consider when using this object in order to get the best experience and performance out of it. A thing to consider is what type of target-shape is the most reasonable for the type of surface you are working with. There are different targets to choose from. The primary targets are the: 3D printed targets, plain flat targets and cylindrical targets. Since plain flat targets are the most efficient, it was used.

![Figure 5. Shows a screenshot from the program Unity](image)

The Vuforia engine gives better results when the provided pictures are: rich in detail, has good contrast, no repetitive patterns and are either PNG or JPG file format taking up no less than 2 mb of space. The picture also need to be angular dependent. If the picture is angular independent, the AR camera will not know the orientation of the picture if it is mirrored on every angle. For example, a perfect circle would be angular independent.
since every mirror is the same. The final produced image for this project is an image of Uppsala university’s logotype, which can be seen in figure 6 and its features in figure 7.

This image is then printed out and glued on the front side of the ultrasound scan so the AR camera can pick it up when using the probe naturally.
3.3 The Network connection

The network connection was done in Visual Studio and consist of three parts, namely; server, setup, and client.

3.3.1 Server

To use mixed reality’s WebRTC solution, a third party server is needed to handle the connection of the clients. The third party server could either be created by the user or the user can choose to use the simple implementation creates by mixed reality as shown below. The server is run with npm(node packet manager) and will be running on a different thread.

The code for the server is very simple. The server is supposed to work in a client-server type of network architecture. The code works by first allowing the HTTP module by requiring the essential parts, and then start the server by routing the incoming traffic to an arbitrary port.

```javascript
const http = require('http')
const finalhandler = require('finalhandler')
const debug = require('debug')('dss:boot')
const router = require('../index')

const server = http.createServer(function (req, res) {
  router(req, res, finalhandler(req, res))
})

const bind = server.listen(process.env.PORT || 3000, () => {
  debug('online @ ${bind.address().port}')
})
```

Mixed reality’s own implementation, as shown in figure 8 is useful for a quick setup and debugging, but this solution is not of high quality and should not be used in a production environment. There is no security, no authentication, especially over HTTP, since all connections are made in plain text, and knowing the identifier to connect with the remote collaborator is sufficient. WebRTC’s doesn’t support for encryption and therefore cannot be trusted. It is very easy for an attacker to hack a signaling device to bypass it. It should be replaced with a safer solution during production.
Figure 8 shows the WebRTC architecture [18]

3.3.2 Setup

To begin the use of WebRTC the user needs to install the WebRTC library to their corresponding platform. This comes with some problems, since mixed reality’s documentation regarding the installation does not completely cover all the possible ways of introducing the library into a project. The mentioned way of installing the library was not actually working in practice, so we need to find another way of linking the library. We used the following code in order to solve the issue.

```csharp
[DllImport("mrwebrtc.dll")]
public static extern int MessageBox(IntPtr hWnd, String text, String caption, uint type);
```

The code works by importing an external dll file when invoking the MessageBox function, which will automatically be called when executing the program.
3.3.3 Client (WebRTC)

The client utilizes the mixed reality’s structure of connecting two peers over the internet. The structure is made up of very clear and distinct parts.

- The first step is to set up a STUN server.
- The second part is to produce a function to handle the different types of incoming and outgoing messages.
- The last part is to connect all the parts and begin sending over the necessary information.

3.3.4 Client (Image handling)

Mixed reality’s implementation of WebRTC also includes capabilities and blueprints for producing & sending frames of images when a connection is already established. By using the "ExternalVideoTrackSource" class, it is possible to inject a frame producing callback function which is used to generate frames and inform the WebRTC module when a new frame is ready to be pushed out to the remote peer.

```csharp
public void FrameCallback(in FrameRequest frameRequest){
    if (global_thread_safe_bitmap == null){
        return;
    }
    using (var ms = new MemoryStream()){
        try{
            global_thread_safe_bitmap.Save(ms, ImageFormat.Png);
            Bitmap bitmap = new Bitmap(ms);
            Byte[] byte_bmp = BitmapToByteArray(bitmap);
            Argb32VideoFrame new_frame = new Argb32VideoFrame()
            {
                data = GetByteIntPtr(byte_bmp),
                height = global_thread_safe_bitmap.Height,
                width = global_thread_safe_bitmap.Width,
                stride = global_thread_safe_bitmap * 4
            };
            frameRequest.CompleteRequest(new_frame);
        }
        catch{
            Console.WriteLine("Error");
        }
    }
}
```
The way the framecallback function was implemented in this project is shown in the code above. Since this function is called on constantly from the beginning of initialization, the first if statement prevents the function from trying to generate a frame when the ultrasound image has not yet been produced from the probe. If there however is a ready image, the callback function will convert the image to a PNG image, produce a byte array from the image and then make a new frame with that data before calling the "completeframerequest" function which notifies the program that a new frame is ready to send. The reason for converting the bitmap into a PNG is that the "GetbyintPtr" functions requires it.

3.4 Integration

The client was coded as a module to make it easier to handle and integrate the code for future updates of the interson program.

The important part of the code is in the formScan2D class and it is held in the IntersonArray.dll library. This is where the entire interface is drawn, but also where the ultrasound images are being produced. Due to the codes being delicate and not being executed in sequence, we decided to run the integrated code on a new thread. Since there is only one important shared memory part in the system namely the bmpout image object, we decided to make a deep copy of the bmpout image object every time the original one was updated. When the image is copied, the program is free to operate on the new image without interfering with the Interson program.
3.5 Testing

Some parts of the code are not isolated and therefore harder to test. Therefore, we only tested the isolated parts. Such an example would be the "BitmapToByteArray" function, which takes a bitmap and returns a Byte array representation of it.

```csharp
public static byte[] BitmapToByteArray(Bitmap bitmap){
    BitmapData bmpdata = null;
    try
    {
        bmpdata = bitmap.LockBits(new Rectangle(0, 0, bitmap.Width, bitmap.Height), ImageLockMode.ReadOnly, bitmap.PixelFormat);
        int numbytes = bmpdata.Stride * bitmap.Height;
        byte[] bytedata = new byte[numbytes];
        IntPtr ptr = bmpdata.Scan0;
        Marshal.Copy(ptr, bytedata, 0, numbytes);
        return bytedata;
    }
    finally
    {
        if (bmpdata != null)
            bitmap.UnlockBits(bmpdata);
    }
}
```

We tested the code by calling the function with different types of pictures to check whether the function was returning a correct byte array of the bitmap by translating the byte array back into a bitmap and compared it with the original image.

When testing the user experience, where user confirmation was not as accessible, we observed if the pixels from the image streams arrived in a correct way.
4 Result

4.1 The final product

The image below (figure 9) shows how the System will look like when using it.

Figure 9 shows the final product in action.

The above picture depicts how the AR application will look like when running the system. As you can see the picture is in action while the two demonstrators (us: Hanna and Salem) uses the probe on the arm. This is how one would typically use the applications when using the probe and the associated software.

The following two picture depicts the other two components that are used in the final product, namely the interface for the ultrasound software and the ultrasound probe itself.
Figure 10 shows the ultrasounds probe with the Uppsala University logo glued on.
Figure 11 shows the Interson interface

The system works by opening the three different parts of software. The first step is to initialize the server by running the nodejs server with the command "npm run". When the server is set up, the next step is to start the unity application first and then start the Interson application.

The unity program runs when pressing the app icon, and the unity logo will appear on the screen. After a couple of seconds of loading time, the webcam will activate and displayed on the screen. During the loading time, the program will also connect to the node server.

The Interson interface (figure 11) has not been changed and works as usual, where one can press the different buttons and the image will change depending on which button. The only difference is that a new thread will begin to execute the injected code in the background, which the user will not experience.

Now whenever the unity program picks up on the targeted image (an image of Uppsala University logo on the probe as seen in figure 10) through the webcam, the program will display the ultrasound image from the Interson program on the targeted image. The location tracking works well in well lit rooms with pictures that have a high contrast, since unique patterns are easier to distinguish from the rest of the room.

4.2 Performance

The machine that was used for bench marking was a Intel® Core™ Pentium Silver N5000 CPU @ 1.1GHz with 2 cores (2 hyper-threads per core), 8 GB RAM, 0.13 MB L1, 1 MB L2, 6 MB L3, running Windows 10.

The camera that was used for the bench marking was the iPhone 11 Pro integrated standard zoom camera with the following settings: 120 fps, 1080p, and HEIF\HEVC video format.

To measure the performance of the system, we used a technique called frame-counting. This is done by recording a video of the system in use and count the video frames of the recorded video when the first ultrasound video frame reaches the client to when the second ultrasound video frame reaches the destination. When the sought for number is acquired, we can calculate the time it took to send the pictures by multiplying the number of frames by how many frames per second the recording was shot in. This gives us a fairly accurate picture of the performance of the system if the camera used to record with has a high frame per second count.
To get a broader picture of how the system is performing, we tested to run the different parts of the system on different machines. This will greatly improve the overall performance since less computing time is being required for each unit/computer. Figure 12 below depicts a graph showing the different scenarios and how it impacts the latency. Where the notation (unity) + (Interson) + (server) stands for the three parts being on different computers and (unity + Interson + server) being on the same computer.

Figure 12 shows the latency of the different scenarios.

The system performs well when the right conditions are maintained, with a frame rate of 26 frames per second. When the conditions are worse, the system performs with a frame rate of 5 frames per second.
5 Discussion

5.1 Discussion of results

We successfully created a program that portraits ultrasound images into the augmented reality world. Therefore, the result shows that the proof of concept holds.

We ran the system on very low end laptops, which could greatly impact the performance when benchmarking the system. The benchmark test should be repeated with a computer with better computing power in order to get a better picture of the performance of the system.

5.2 Difficulties

In the beginning, we sent images over the network by creating our own client- and server program. This was not optimal due to the complications with optimizing the program to an acceptable level within a reasonable time frame. The amount of work needed to implement a new protocol would be a thesis project on its own. This was instead resolved by using the WebRTC protocol where we got to an acceptable performance level as can be seen in figure 12.

At first the goal was to use a different probe from Clarius, but after a while of trying to connect the probe with the API we decided to leave this idea because of it not being able to transfer any images since it presumably was broken. Instead of using Clarius as the probe, we instead got the Interson model from our supervisor. Along with changing the probe, the protocol was also changed during the same development time frame since it being easier to use, as mentioned above.

The next example of a difficulty that we came across was about the interpretation of the documentation. In the documentation, it says that offers can be created from either the remote or the local peer. But when creating the offer from the remote side (unity program) the media lines in the local peer (ultrasound program) does not get any handles and therefore does not produce any colored frames. The only way it worked was when the local peer created the WebRTC offer.
5.3 Yet to be implemented solutions

Further development would be needed before implementing the system for commercial use. For example, the interface of the unity program could be more intuitive. The security of the internet connection needs to be worked on, since there is not any further encryption of the messages that are being sent. There are also some low light concerns to be addressed, since the image target-finding does not work well in dull lighting as when it is bright and well lightened.

Other areas of development would be for example using virtual reality headset instead of augmented reality cameras like the Microsoft’s Hololens glasses which would further improve the ease of use of the system since it is easier to wear a pair of glasses than holding a phone in your hand while operating the ultrasound probe.
6 Conclusion

In this thesis we have shown that it is possible to visualize ultrasound images in an augmented reality world with the tools that were given/available. The main tools that were used to make this possible were Unity, Vuforia, mixed reality WebRTC and the ultrasound program Interson which together created a good enough design of the system. Although there is more room for improvement such as security and user experience, this system is sufficient for proving that the concept holds.
References


[12] Figure 1. August 2021. Available at https://princiya777.wordpress.com/2017/08/19/webrtc-architecture-protocols/


[18] Figure 7. September 2021. Available at https://krispcall.com/blog/what-is-webrtc-how-does-it-work/.