UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS no. 1118

Network Traffic Analysis and Optimization of Augmented Reality Service

Matija Manđurov

Zagreb, June 2017.

Table of contents

Introduction

The recent advancements in virtual reality related technologies and affordable end-user devices paved the way for development of new services over the Internet, based on these technologies.

Since the invention of the telephone people have sought better and more advanced means of communication at a distance. The ultimate solution, one may imagine, would be to simulate and replicate the natural communication of two or more individuals as if they were physically in the same room. The future of communication could thus potentially look and feel like the Holodeck from Star Trek [1]. Now that the first commercially available mixed reality glasses have been made available, this may no longer be just wishful thinking.

The purpose of this thesis is to study the fundamentals of augmented (mixed) reality technology and create an experimental augmented reality service, enabling the users to have an audio conversation over the network, while seeing a virtual 3D image of each other through the Hololens display. For easier reference, this service will be called “HoloTalk”. The thesis presents an analysis and description of the characteristics of the information that is transferred while using HoloTalk, and the corresponding network data traffic. Algorithms for real-time data compression have been implemented and reviewed based on their suitability for this service. A laboratory prototype has been created using the appropriate resources and the performance of the compression algorithms has been measured and analyzed.

1. Overview of key technologies and selected studies

1.1 Selected studies

Prior to addressing the specific technologies applied in this thesis, a summary of the studies and projects already conducted with similar technologies, or a similar basic concept, is presented here. By “similar” technologies and concepts, we primarily consider those that enable the transfer of “live” 3D user image in real time.

The first project of interest is the Holoportation [2] project by Microsoft. The Holoportation project uses the Microsoft Hololens [3] as the smart glass device of choice. In addition to using Hololens as a smart glass device, this solution uses multiple depth cameras to record data about the user, which is then combined into a single 3D image and transferred to the Hololens.

Image 1. Microsoft Holoportation [2]

As shown in Image 1, the initial recording setup requires the whole room to be dedicated exclusively to the recording process, with multiple cameras deployed across the room. In contrast, the HoloTalk solution presented in this thesis only uses a single Kinect device for recording purposes, thus at least in part reducing the setup time and cost.

Another example of a solution similar to the one presented in this thesis is the Hololens-Kinect [4] GitHub project, where only the skeletal data and the joint information is sent across the network for the purposes of a 3D human avatar simulation (Image 2). More specifically, as the position of the person and the information on joint movements in space are sent over the network, a skeleton model mimics the joint movements on the Hololens side. This approach greatly reduces the bandwidth requirements for data transfer, since the only twenty or so joint positions stored in Vector3 [5] information format need to be transferred over the network.

Image 2. Hololens-Kinect view of a skeleton-based 3D character [4]

Lastly, one can refer to the LiveScan3D-Hololens [6] project on GitHub and its resulting paper [7]. It uses a network of Kinect devices, to record the initial 3D image, then processes it and sends it through a local network to the Hololens. The drawbacks of this project are the simplicity of the 3D image and no suggested ways to deploy this solution over the public Internet, thus limiting it to a local network transfer.

1.2 Mixed Reality

Mixed reality, as illustrated in Image 3, is usually described in general terms as merging of real and virtual worlds. More specifically, Paul Milgram and Fumio Kishino define mixed reality as "anywhere between the extrema of the *virtuality continuum*" [8], where the virtuality continuum extends from the completely real to the completely virtual environment, as illustrated in Figure 1.

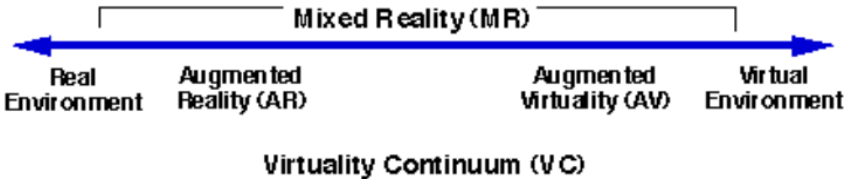

Figure 1. Virtuality continuum as defined by Milgram and Kishino [8]

Milgram and Kishino take the related term *augmented reality* to refer to "any case in which the otherwise real environment is 'augmented' by means of virtual (computer graphics) objects". One of the problems with defining mixed reality and other various variations on the commonly accepted reality is the lack of standards used through the scientific and development community resulting in new versions of terms being introduced in excess. For the purposes of this thesis, we adopt the mixed reality definition by the Milgram and Kishino, since it is the term which has gained the most widespread recognition and could be considered the standard in use.

Image 3. Example of mixed reality [9]

The main difference between mixed reality and virtual reality is that virtual reality places the user completely into a virtual environment, usually with the help of a head mounted display, while mixed reality combines the real world with computer-generated images, sounds and various other methods of sensory input to create a new experience. As such, mixed reality relies more on merging the two realities together and putting more emphasis on immersion. Virtual reality can create fantastic new worlds where the rules of physics do not necessarily need to be followed, while mixed reality remains in the real world. The key point to creating a realistic mixed reality is making sure that the virtual added elements merge seamlessly.

While it is generally difficult to pinpoint the first use of mixed reality in a modern setting, one of the first notable uses can be found in 1974 by the researcher Myron Krueger at the Space Science and Engineering Center of the University of Wisconsin-Madison in the USA. The research done by Krueger was coined Videoplace [10] and used projectors, video cameras, special purpose hardware and onscreen silhouettes of the users to place the users within an interactive environment (Image 4). The end purpose of Videoplace was to establish communication between two users in different rooms, using silhouettes to simulate the movements of the user in the other room. It does not require a great leap of imagination to see the future possibilities of Videoplace and, finally, HoloTalk as the solution of this thesis, conceptually does not differ that much from the original idea presented by Krueger almost half a century ago.

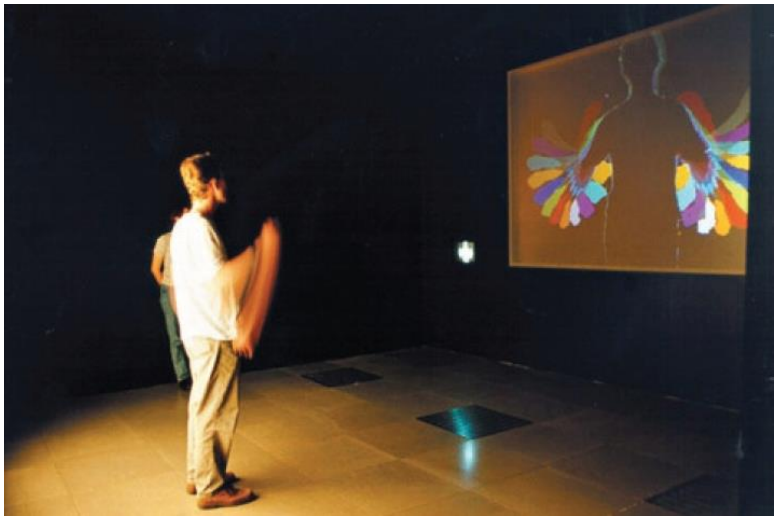

Image 4. Videoplace in action [10]

1.3 Microsoft Hololens

The central feature of HoloTalk is the ability to create 3D images in mixed reality (Image 5) and for that purpose the Microsoft Hololens was chosen as the display device for HoloTalk. The Hololens can be categorized as a pair of mixed reality smart glasses, which in turn can be defined as wearable computer glasses. The Hololens was chosen for its spatial positioning ability, similar to its competitors such as Google Glass [11], or in other words the ability to merge the virtual world and real world with sufficient precision. One of the main challenges mixed reality using smart glasses is to prevent the breaking of immersion due to faulty positioning of 3D objects in the real world. Immersion cannot be maintained if the 3D objects placed in the real world cannot maintain its position naturally once the wearer of the glasses starts moving around in space.

Image 5. Example of a 3D object in Microsoft Hololens [3]

The way the Hololens operates is through a set of two infra-red (IR) cameras mounted above the user's eyeline which constantly scan the area surrounding him or her and try to map out the physical world with the information they collect.

Furthermore, the Hololens includes 2 GB of RAM for processing purposes and a further 1 GB of Holographic Processing Unit RAM, known as HPU RAM, used exclusively for the processing of sensor data needed to map the surrounding space. Having two gigabytes of memory for processing purposes is also a unique new aspect to the smart glass solution, and an improvement over earlier version of Hololens, especially concerning the topic of this thesis, since a significant amount of that memory will be used on reconstructing the 3D image and texture data the Hololens will receive from the other user in the HoloTalk 3D conversation service.

Image 6. Microsoft Hololens [3]

On the other hand, as it can be observed in Image 6, the Hololens itself is an item which cannot be worn comfortably in public for an extended period of time. Most of its size can be attributed to the battery. The battery itself has an operating standard of about two hours (which is likely to increase in the future, as it is also expected that the size of the battery will decrease), but for now the Microsoft Hololens remains a relatively large object more suited for technical demonstration than for everyday use.

1.4 Microsoft Kinect

The Microsoft Kinect [12] is a motion sensing input device which combines various sensors to fully track a person's motions and gestures. It was originally intended only as a gaming addition to the Xbox platform, but soon found its primary use as a cheap and reliable depth camera for testing and research purposes. The original Kinect was released in 2010, but it still finds widespread use at the time of writing (2017) due to its low cost and advanced algorithms that went into the development of the original Kinect. For this thesis, a Kinect v2 was used due to the advancements made to the newer generations of Kinect.

At its core, the Kinect relies on its depth camera to map the surrounding area using IR technology, combined with a microphone to track and record the position of user(s) in front of the camera. What makes the Kinect stand apart from its competition at this price range, is the initial volume of the data used to train the original Kinect algorithms. The large original data set resulted in the ability of the Kinect to recognize and track specific gestures and movements by the user in front of the camera.

The ability to separate the user image from its surroundings (Image 7) is an important factor in HoloTalk since it reduces the noise in the recording and the size of the recording itself. There is a large difference in needing to manually process and separate an entire depth frame and needing to only send an already processed person image which consumes only about one fifth of the data size of the overall frame.

Image 7. Example of a frame with a user (thesis author) already processed

1.5 Unity Game Engine

Unity Game Engine [13] was chosen as the primary working environment for the development of HoloTalk for a variety of reasons. The primary reason is that, as of Spring 2017, it is the only engine that supports all the various components necessary for HoloTalk to work. For the Microsoft Hololens there is a readily available software development kit (SDK) called HoloToolKit [14], which will be covered in more detail in a later chapter, concerning the implementation of HoloTalk. Likewise, there is also a publicly available SDK for the Microsoft Kinect [15] making integration of the Kinect with Unity possible. Finally, due to Unity's multi-platform nature it would be possible to potentially develop and deploy HoloTalk on a variety of platforms in the future, thus increasing the likelihood of HoloTalk being available for wider testing and demonstration purposes.

Unity engine was written in C and C++ programming languages as the base, while the developers working on a Unity project contribute to their projects with scripts written in either C# or JavaScript. HoloTalk was written using C# for the overwhelmingly large part of the code, with some minor parts written in JavaScript, such as the relay server necessary for the handshake in the public network transfer. Furthermore, due to Unity's extensively developed graphical user interface (GUI) (Image 8), it was possible to create and develop large segments of HoloTalk, using only the high-level Unity application programming interface (API). For more information about the high-level API, one can refer to the official documentation [16].

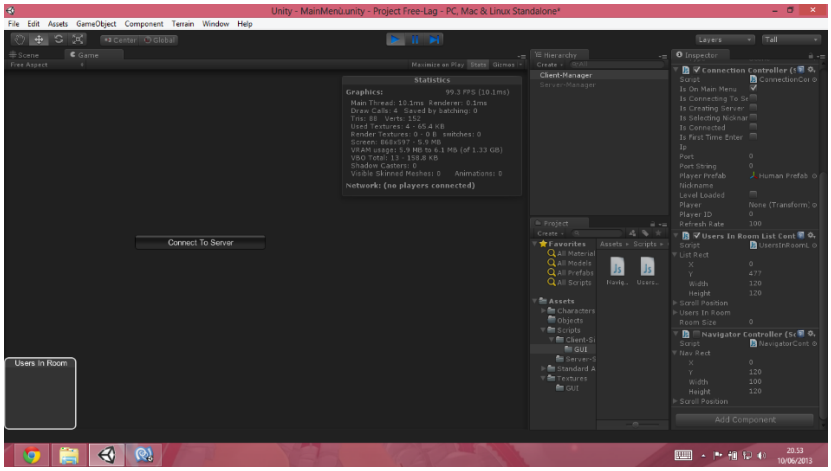

Image 8. Example of the Unity GUI

2. Functional requirements and system model

This chapter presents the functional requirements and system overview of HoloTalk.

2.1 Functional requirements and diagrams

The following functional requirements are integral to the design and implementation of HoloTalk.

- The sending user is able to record an 3D image with a recording device capable of recording standard video input and recording depth based data.
- The sending user is able to use an application on his PC for the processing and transmitting of data to the receiving user.
- The receiving user can view the resulting 3D image with mixed reality glasses.
- The receiving user can manipulate and adjust the received 3D image.
- The system shall provide network communication, including processing and compressing the data, between the sending user and receiving user.

A block diagram in Figure 2 provides a high-level view of the main functions which comprise HoloTalk.

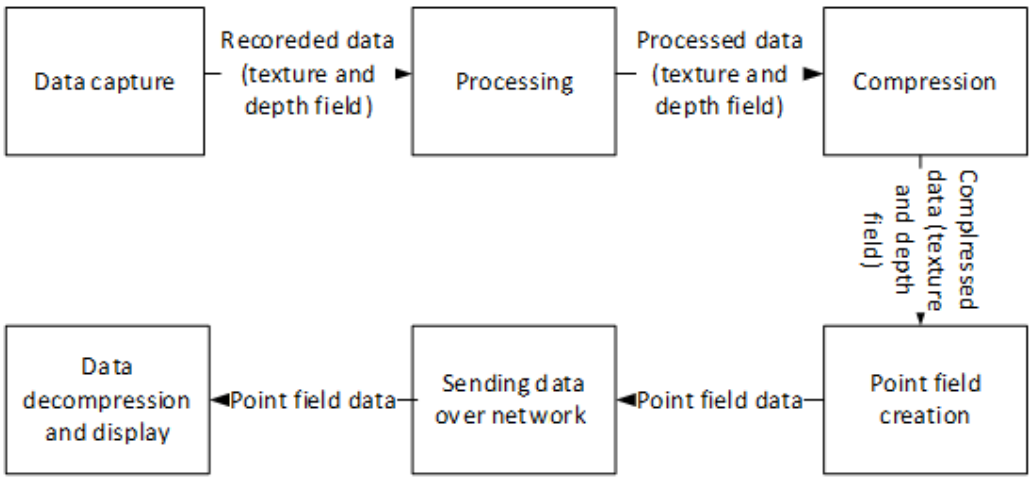

Figure 2. Main functions within HoloTalk

Workflow diagrams showing the perspectives of the sending user and the receiving user are given in Figures 3 and 4, respectively. (Only one-way communication is shown, which is simply "mirrored" to enable two-way communication).

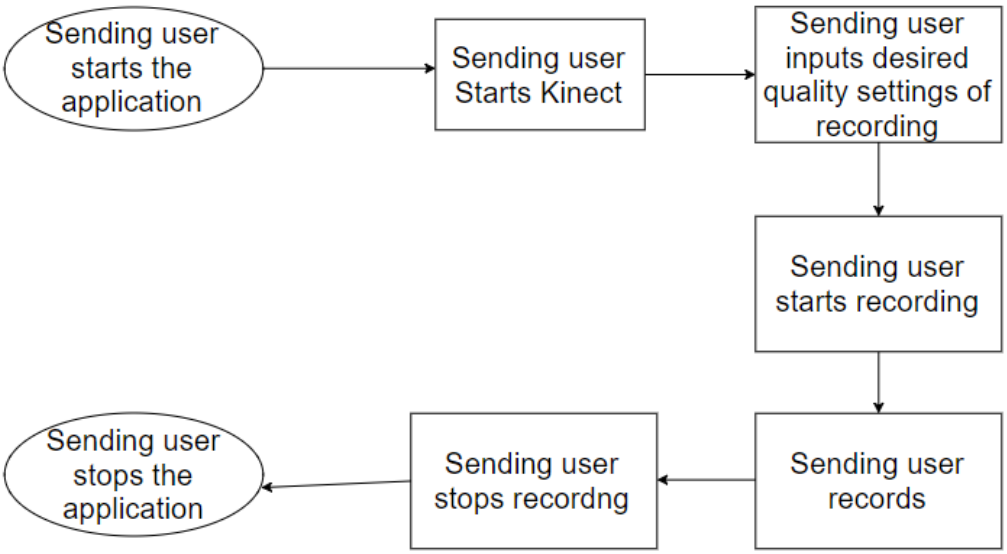

Figure 3. Workflow for the sending user

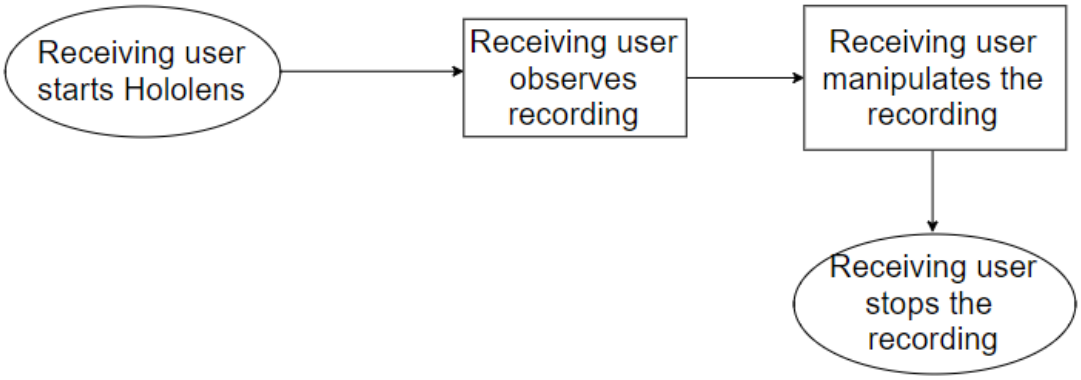

Figure 4. Workflow for the receiving user

2.2 Overview of the system design

Figure 5 presents a high-level view of two users using HoloTalk, with the sending user shown as *User A*, and the receiving user shown as *User B*. (As before, this figure only depicts one direction of communication, which in the final working solution would be implemented in both ways.)

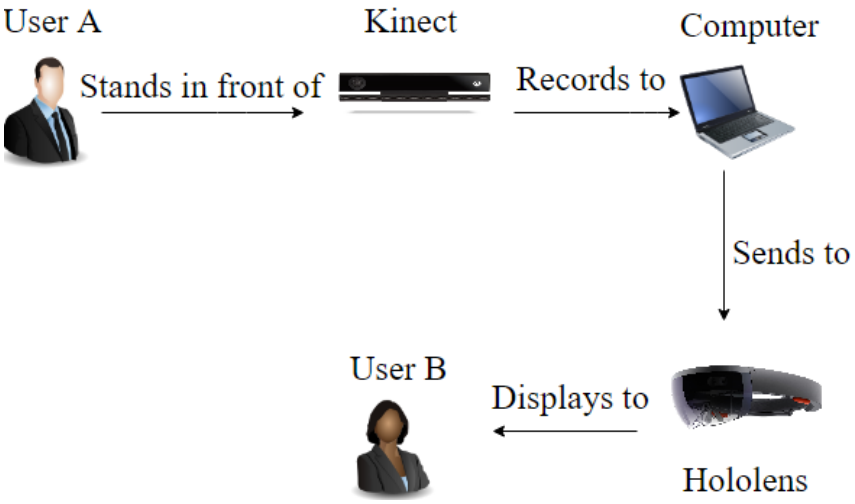

Figure 5. Using HoloTalk

Figure 6 shows the overview of the system design. The starting point of HoloTalk is the Kinect, used by the sending user, after the user has started the application on his local PC, which in turn starts the recording on the Kinect, the user positions himself in front of the camera. The Kinect then records the depth data, regular video data and audio data and stores them on the PC. Once on the PC, all the data collected is processed and packaged to be transferred over the network.

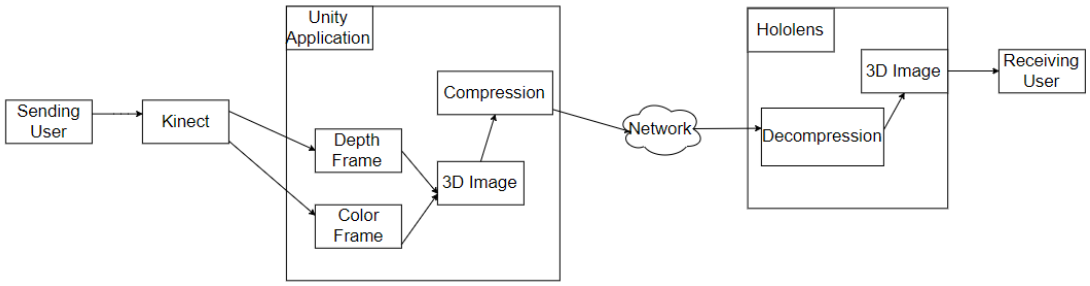

Figure 6. System components

Processing on the sending user's PC is handled by the Unity application at an interval between 15 and 30 frames per second (FPS). Both the depth data and color data are processed until a final compressed 3D image is created. Once the 3D image is finalized, depending on the type of network connection, the Unity application either starts a network transfer over LAN or contacts the Session Traversal Utilities for NAT (STUN) server for communication over the public Internet. (Using a STUN server allows real-time voice and video applications to detect and traverse network address translators (NATs) located along the path between the communication endpoints.) If the network connection is local, a direct connection is made to the Hololens through the Unity application. A separate application on the Hololens accepts the connection, and processes the incoming data and displays it to the receiving user.

If using network transfer over the public Internet, then (once the Unity application contacts the STUN server) it is directed to the location (IP address and TCP port(s)) of the user who is receiving the data. The receiving user has a computer ready with an active application which receives the data and forwards it to the Hololens for processing and visualization. Detailed description of the compression process and network transfer will be presented later in this thesis.

3. Implementation

Figure 7 shows the mapping of the main functions of HoloTalk to system components, and their implementation using Kinect, Hololens and Unity applications.

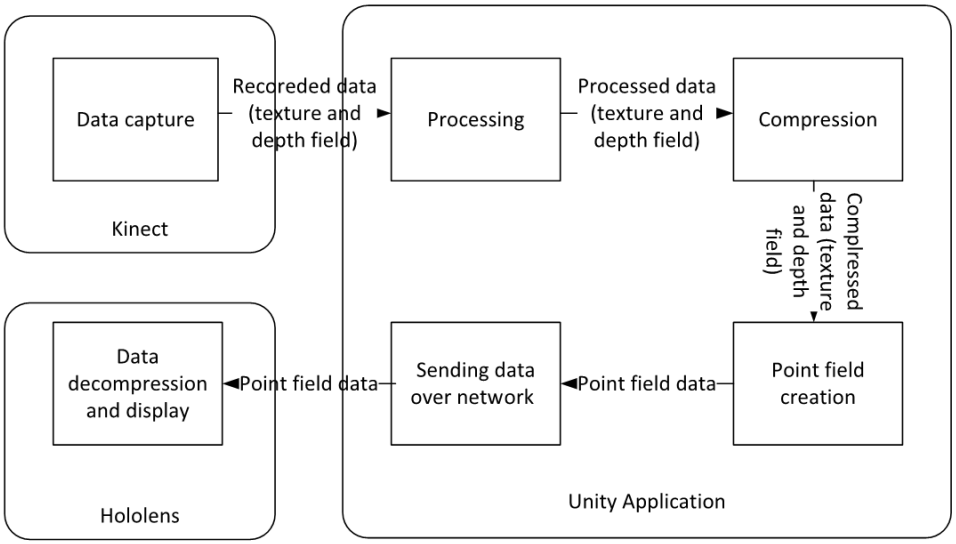

Figure 7. Implementation of HoloTalk

As evident from the figure, the implementation of HoloTalk requires many components to collaborate and coordinate for it to work. What follows is a short list of scripts responsible for the parts of the implementation, with descriptions of their respective purposes:

- Kinect Manager - Capture and recording of Kinect data,
- User Mesh Visualizer – Adjusting the depth frame to the final mesh data,
- Background Removal Manager – Adjusting the color frame to the final texture data,
- Compression- Implementation of all compression functions,
- Sender – Sending the data from the PC to the Hololens,
- Receiver – receiving the data from PC to the Hololens,
- Visualizer – Visualizing the received data for the Hololens,
- Stun Server – STUN server implementation.

3.1 Data capture and processing

A *point cloud* can be defined as a set of data points in some coordinate system. For this thesis, a three-dimensional coordinate system was used, to capture the resulting 3D image data. The first step necessary in creating HoloTalk was setting up the point cloud recording (Image 9). As previously mentioned, a single Kinect will be used as the recording device for generating the point cloud data, which will be transferred via USB 3.0 cable connection to a local PC which holds a Unity application for data processing. Ideally, the user should be standing one meter to a meter and a half away from the Kinect device to ensure that the entire person image is captured in point cloud information. The Kinect device itself is also capable of partially recognizing a person if one stands closer to the device, but it should be noted that the final point cloud data will contain only the parts of the person "visible" to the Kinect, so the closer one stands to the device, the larger percent of the person will be obstructed.

Image 9. Kinect recording setup

Using the Kinect SDK in combination with Unity Game Engine, a script was produced in C# called User Mesh Visualization which handled the incoming data from the Kinect. The Kinect transfers two relevant data streams to the Unity application:

- the color stream, and
- the depth stream.

The color stream transmits the RGB camera feed from the Kinect in byte array form. The RGB camera has two resolutions a 512×424-pixel resolution image, and a full HD image. Both resolutions will be used to provide for a product of various quality, the low-quality stream that saves on bandwidth usage but lacks picture quality, and the higher quality stream for cases when sufficient bandwidth is not a problem. The depth based stream returns a 512×424 sized frame, where each of the points contains within itself the information on the distance of the point from the Kinect point of origin. Using this depth stream is the basis for constructing the point cloud information. Figure 8 illustrates the processing of each individual frame.

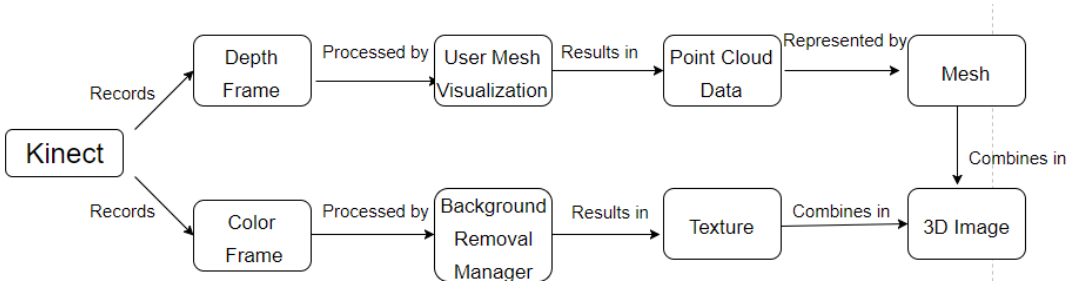

Figure 8. Processing of related depth and color streams

The two streams are delivered to the Unity application at the rate of 30 frames per second, and each new frame that is delivered is processed by the User Mesh Visualization. The way the User Mesh Visualization works is that it linearly goes through the depth frame, and locates each point that it considers belonging to the set of points comprising the image of the user and stores it, while discarding the rest of the points. Locating the points is performed by referencing the skeletal position tracking of the user performed by the Kinect. The Kinect, using its own internal algorithm, can position and locate the joints of the user it is currently tracking. Once the user is located in 3D space, every depth point within a set distance from one of the joints is considered a constituent part of the user. Unexpected side effects of this algorithm can sometimes result in adding unnecessary objects such as hats, or objects held in the hand to the result, but the algorithm can generally be relied on.

The resulting array is a collection of points representing the user's point cloud representation. In parallel to this process, the color frame is processed by a script called Background Removal Manager, provided by the Kinect SDK, which like the User Mesh Visualization, analyses the frame and removes the background noise leaving only the user against a black backdrop. An example is shown in Image 10.

Image 10. Output from the Background Removal Manager script

Once these two processes are completed, the resulting streams are processed together to create the frames shown within the Unity Application. The point cloud data is used as the basis for mesh construction, with the points used as vertices, and the triangles and UV coordinates constructed from the vertices. After the mesh is constructed, the resulting texture from the RBG feed, with the background removed, is overlaid on the mesh, resulting in the final “holographic” frame. A final 3D frame is shown in Image 11.

Image 11. A final 3D image frame

It is worth mentioning that the background removal and mesh visualization tasks are operating in their own separate threads, due to performance issues. The Kinect device is a performance intensive piece of equipment and as such, the multitude of tasks and calculations it performs for each frame affects the general performance. While placing the additional tasks in separate threads reduced the overall performance bottleneck, it also introduced a major synchronization problem, which will be further analyzed later in this thesis.

3.2 Compression process and algorithms

The algorithms that will be analyzed here are the custom created Sample Size process, Delta compression and Deflate compression, applied onto the color and the depth frames. A general overview of the compression process is shown in Figure 9.

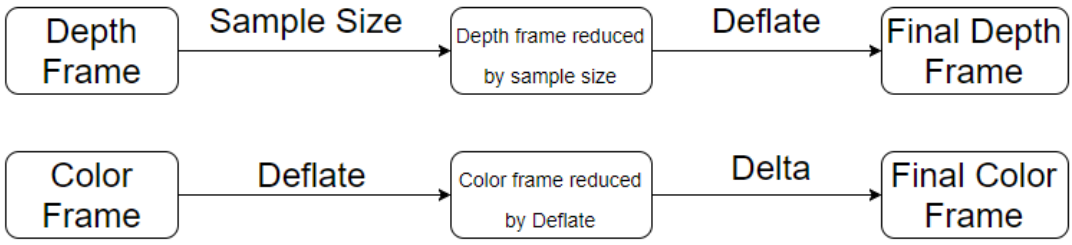

Figure 9. Compression overview

A new quality setting was added to the User Mesh Visualization script, called the Sample Size, with the setting having four levels: 1) Unstable High, 2) High, 3) Medium, and 4) Low. Depending on the setting chosen, the linear traversal across the depth frame array happens in different intervals, effectively down sampling the data by factor of 2: for example, on High setting every 2nd position would be read, on Medium setting, every 4th position, and on Low, every 8th. The list of quality levels and array sizes is presented in Table 1.

Quality Level	Array size
Unstable High	217,088
High	108,544
Medium	54,272
Low	27,136

Table 1. *Sample Size* four quality levels settings and their respective array sizes

The purpose of introducing this Sample Size setting was to reduce the maximal potential number of resulting point cloud information. As shown in Table 1, when using the lowest setting (*Low*), which reads only every 8th place, the maximal number of points to be read was 27,136, which was four times lower than the hard limit for the mesh, even in a case where the user's image takes up the entire screen. During filming and testing it was estimated that about 20 to 25 percent of the screen was occupied by the image of the person being recorded when the user was positioned at a distance of one meter from the Kinect device, which can be used to estimate the final mesh size to be at around 5,000 points, resulting in only a single percent of the original full frame size.

Naturally, it should be mentioned that reading anything but every single point constitutes a lossy compression, as the resulting mesh lacks the full information needed to recreate the user image. A visual comparison between the four quality states can be seen in Figure 10, as one can observe increasingly higher quality when going from the Low to the Unstable High quality settings.

Quality level: Low

Quality level: Medium

Quality level: High

Quality level: Unstable High

Figure 10. Comparison between Sample Size Quality settings

The next step in the compression process was to reduce the size of the resulting depth frame and color frame, after performing the initial Background Removal and User Mesh Visualization. At that point, the depth frame was already reduced by the process described above using the Sample Size factor, and the color frame already had the user image isolated, with the rest of the frame being replaced by a black background.

Since both frames included many identical values, namely the black background on the color frame, the next algorithm to be applied was the Deflate algorithm. The Deflate algorithm is a run-length encoding lossless compression algorithm which shortens the large sequences of identical characters in an array [17]. For the purposes of this thesis, it was implemented in C#, in the script called Compression, along with other implemented algorithms.

Finally, after both the Sample Size compression and the Deflate compression were applied to the depth frame, the size of the depth frame ended up being an order of magnitude smaller than the color frame, and the size (if necessary) could be made small enough to be comparable to modern video streaming services, with typical frame sizes ranging from 0.5 MB to 8 MB. On the other hand, the color frame still needed a final compression run through to reach the levels mentioned for modern video streaming services.

Thus, the last compression algorithm used in HoloTalk is the Delta compression, based on differential coding. The Delta compression takes its cue from video frame compression, sending a full color frame every set interval, while sending only the delta frame the rest of the time, with the delta frame consisting of changes to the frame compared to the last sent frame [18]. Image 12 illustrates the difference between the full and the delta frame.

Image 12. Example of a full frame (left) and a delta frame (right) in comparison

Delta compression algorithm passes through the array in linear way, and compares each point in the current frame to the one in the same position in the previous frame, storing the difference in the delta frame. The delta frame is then sent to its destination, where it is compared to the last received frame. Once the comparison is complete, the resulting frame (with the differences from delta frame added to the previous frame) is displayed. After the set interval is reached, a full frame is sent to effectively reset the image on the receiving side, thus making sure that no data is lost with too many delta frames sent in succession (i.e., accumulating the error).

Delta compression works best on a slow-moving set of frames, for example when a user is standing in front of the camera or only moving slightly since then the delta frame only contains small incremental changes. If a user moves too fast then the compression is of limited use since it is effectively sending a full frame on every frame due to the large number of changes made to each frame. (In general, this is a well-known problem in video compression, solved by introducing the motion estimation process.)

3.3 Implementation specifics of compression algorithms

One of the biggest challenges of implementing compression algorithms for HoloTalk comes from the architecture of the Hololens itself. While the Hololens architecture is based on the x86 [19], the overlaying application architecture is based on the Universal Windows Platform. The main problem with the Universal Windows Platform (UWP for short), is the lack of cross platform compatibility with previously implemented compression algorithms, or more specifically the DLLs needed for them to work. While that means is that it is not easy to find already implemented algorithms that can function in a setting where one device is using a PC based application, while the other is using an UWP application. Considering this problem, all the following compression algorithms were implemented from scratch.

The implementation for the Deflate algorithm is straightforward. The implementation goes through the array linearly marking down if a character starts to repeat itself. Once the character repetition comes to an end, it marks down the number of repetitions and replaces the original characters with a single character and the number of repetitions. For example, an array comprising the numbers [1,1,1,1,1,2,1,1,1,1,1], would be compressed using a Deflate algorithms as 1*5 2 1*5. As can be observed the Deflate algorithm works best on arrays with many identical repetitions, a great example of which can be found in the background removed color frame. The large number of zeroes representing the black color of the background can then be replaced by a single zero and the number of zeroes in succession, thus greatly reducing the size of the color frame array.

Image 13. Example of a frame suited for a Deflate compression

On the other hand, when dealing with arrays filled with random numbers that do not repeat in an organized fashion, the Deflate algorithm makes almost no improvements, often with the ending array being the same size as the starting array. While the Deflate algorithm is more suited for the color frame procession than the depth frame procession, there was no harm done in compressing the depth frame parallel to the process of compressing the color frame with Deflate. Since both tasks have their own threads and do not interfere with each other, there is no point in waiting on the depth frame thread for the color thread to run the Deflate so that the synchronization can continue, thus it was possible to also run the Deflate on the depth frame without a performance drop, even though the resulting compression was not as high as it was with the color frame.

On the other hand, as a part of the implementation of the Delta algorithm, a new quality setting was added to its algorithm as an attempt to cut down on the array size known as the Loss Percentage factor. The Loss Percentage is a sliding factor with a value from 0 to 1, which introduces lossy-ness to the process. The way that the Loss Percentage works is that depending on the size of the factor, it considers every change smaller than the given percentage to be “small enough to ignore”. Simply speaking, it runs the same process as generic Delta compression, but instead of marking down every change it comes across, it only marks down the changes which are greater than the percentage set as the Loss Percentage. For example, with the loss percentage factor set to 0.05, any difference within 5% would be ignored.

This step of the process, as mentioned earlier, introduces lossy compression, and as such reduces the quality of the final color frame. Furthermore, during the development time of HoloTalk, the Loss Percentage was added only as an experimental feature and its implementation still requires further fine-tuning to make sure it is working efficiently and producing the desired effects.

Unlike the Deflate compression, the Delta compression is only applied to the color frame, and not the depth frame. While the same general technique could be applied to the depth frame, during the testing period for HoloTalk, many errors and unexpected side-effects appeared during the reconstruction of the mesh from the depth frame when using the Delta compression. Combined with the fact that the depth frame was already at a size comparable to modern video streaming services before the Delta compression, it was decided to not further pursue the implementation of the Delta compression on the depth frame. It is also worth mentioning that due to the discrepancy in compression process between the color frame and depth frame, once the depth frame is finished with the Deflate process it would have to wait on its thread, while the color frame proceeded with the Delta compression. This difference in processing led to a desynchronization of threads and remains a problem to be dealt with in (potential) future versions of HoloTalk.

3.4 Sending data over local connection.

Once both the current color frame and depth frame are fully processed, compressed and located on the Unity Application running on the local PC, the next step was transferring both frames to a Hololens device over a LAN connection, where both the PC and Hololens were connected to the same local router.

As was the case with implementing the compression algorithms, the same problems arise here. Due to the specific architecture of Hololens, the existing networking solutions could not be used, since the Hololens depends on a different subnet of .NET framework, compared to the standalone PC application. What this meant was that the existing Unity Networking HLAPI [20] could not be used, and a custom TCP solution using sockets needed to be created. For the initial setup, the Hololens was chosen to act as the server, while the PC would act as a client. This setup was deemed appropriate since it enabled future scenarios where more than one 3D image could be displayed on the Hololens, thus enabling conference style scenarios.

The client was written using the standard .NET framework, and is a basic implementation of a client using TCP sockets; for further reading on the topic one can refer to a TCP socket server simple guide [21]. In the current implementation, it serves to open a TCP connection with a given IP address, stream the data over that connection, and close the connection. In potential future versions of HoloTalk, more complex functionalities could be added.

The server on the other side was implemented using the Windows.Networking library, which is a slightly more complex version of the .NET framework. While nothing in this library differs greatly from the standard networking library, it is worth mentioning for further Hololens development that this is the library needed for working with network transfers with the Hololens, and one cannot rely on previously written .NET solutions for the PC or other platforms. The server was written as a simple TCP server that operates on three ports.

Port 57777 was open for the color frame transfer, port 57778 was open for the depth frame transfer and port 57779 was open for the audio transfer. All three ports were chosen at random from the list of dynamic ports. Furthermore, when designing the user interface, all three port values were exposed as public variables for the user to change if he wishes, for the situation where the transfer is happening on a network than has only specific ports open.

With the overall system in mind (as presented earlier in Figure 6), here is the basic scenario for the “Network” part. First, the receiving user starts the Hololens and the appropriate receiving application. Right from the start, the application starts the TCP server and listens on the three predefined ports. While the Hololens is waiting for data, the sending user starts the Unity application on the PC and stands in front of the Kinect, going through the process described in the previous chapter. Once the first color and depth frame are finished processing, the PC application contacts the server and establishes the TPC connection. If a successful connection is made, all the subsequent frames are sent over the now established TCP connection.

TCP was chosen as a transport protocol over UDP because of its reliability, while no significant delays were found while transmitting data over the local network. In the next chapter, the move from HoloTalk over a LAN connection towards a connection over the public Internet will be covered.

3.5 Sending data over the public Internet

Development of the network transfer over the Internet came late in the timeframe allotted for this thesis, and as such the following chapter will just outline the problem and possible solution, rather than implement it. While a stable video chat was implemented by using Web-RTC [22] and a free Google STUN [23] server as the handshake point, a fully functional solution for 3D image transfer over the public Internet network remains on the list of potential further improvements to HoloTalk. The first step in transforming this solution from a LAN connection towards transfer over the Internet was the addition of a second PC to enable the use of the STUN server. The general idea of the solution is outlined in Figure 11.

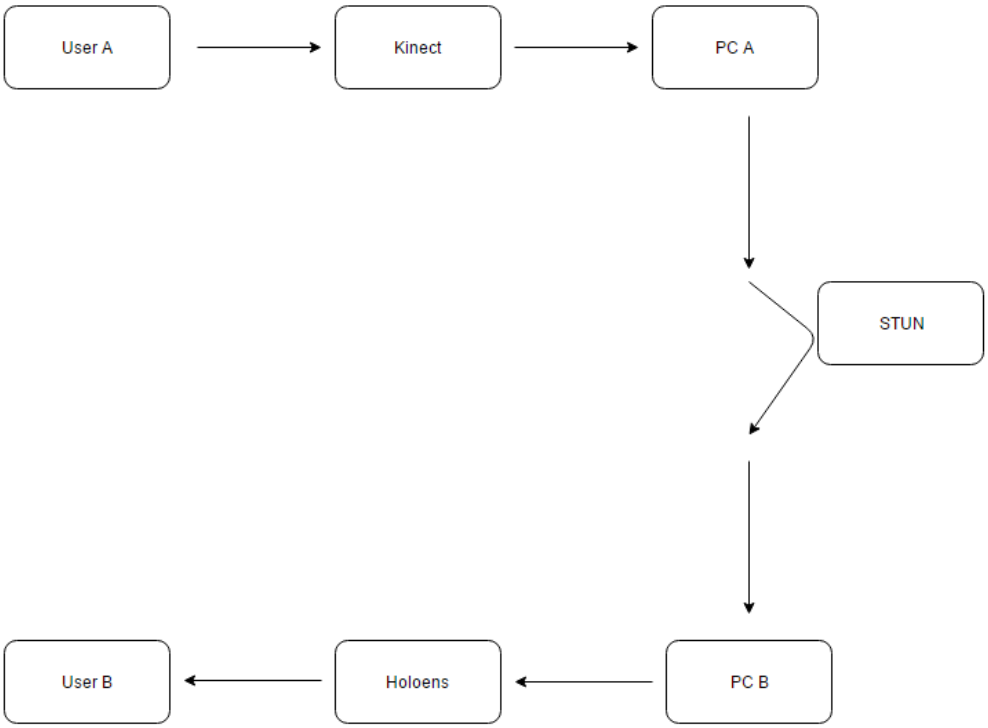

Figure 11. Service establishment over the Internet

The initial communication stays the same, the Kinect records all the necessary data, transmits it to the first computer (PC A) where the compression and processing is done in the same way. Once the processing is completed, instead of opening a TPC connection directly towards the Hololens, the local computer established a distant connection with the receiving computer (PC B) by using WebRTC. For this purpose, a relay point was added for the handshake. Google offers a list of publicly available STUN servers [24] for this purpose, while one can find an example implementation of a STUN server added to the source code of HoloTalk as a reference.

Once the STUN server establishes the initial connection between the two computers (PC A – PC B), all the remaining communication is handled directly between them, thus eliminating the need for any kind of permanent relay server. Choosing to go with a peer to peer solution with a STUN server compared to a traditional relay server, or NAT punching with a UDP protocol, proved to greatly reduce the latency in communication.

When the frames arrive on the receiving computer, the process described in the previous chapter is repeated, ending in a TCP connection between the receiving PC (PC B) and the Hololens. For the purpose of demonstrating a working prototype, a video chat application was developed, implemented and used to successfully transfer video data from the sending PC A to the resulting PC B, demonstrating that the solution works in thus limited settings. Due to the limited time, this demonstration was not expanded to include the initial Kinect (as a sender) and the final Hololens (as a receiver), but it indicated a possible direction for future work.

3.6 Data decompression and display

When the frame arrives on the Hololens, the process for reconstructing the data is straightforward. Using the same Compression script that compressed the original data, we use the inverse functions to decompress the data and revert it to its original form. The decompression process runs the algorithms in reverse order. First, the Delta decompression is applied on the color frame, bringing back either the full frame or adding the delta frame to the previously received frame resulting in the next frame to be displayed. After that, the Deflate algorithm is applied on both the color and the depth frames, bringing them both back to their original sizes. No reverse mechanism exists for the Sample Size compression on the depth frame, since any and all fillings of the missing points would be at best guesswork.

Once both frames are restored to their final size, the Visualizer script takes over and processes the frames according to their type. The color frame is converted from an array of bytes to a texture using a series of internal C# and Unity commands concerning byte to Texture transformation, while the depth frame is converted back to a Mesh using the Simple Mesh Serializer [25]. Simple Mesh Serializer was found as a publicly available tool on GitHub and converted to suit the specific needs of this project. As is the case with the PC application, both processes mentioned here are executed in their own threads, to retain the required performance and not to cause any deadlocks. It is also worth mentioning for any future Hololens development that writing thread operations on the Hololens requires the importing of the Windows.Threading.Task library and not the standard C# library for creating and maintaining new threads.

Finally, when both the resulting mesh and texture are created, they are displayed in front on the user wearing the Hololens, and approximately 33.5 milliseconds later the process starts again on the Kinect at the recording end.

3.7 Graphical user interface

Basic functionalities were added for the interaction with the resulting hologram and a basic GUI was developed for the HoloTalk. A simple GUI interface was developed for the PC application to make the initial setup easier for the end user. The sending user has the option of manually adjusting the Sample Size Quality, which, as previously stated, adjusts the quality of the mesh, and he or she can also adjust if the low quality or high-quality camera feed will be used, and if the Delta Compression will be used, in the case of a bandwidth network connection being available. Furthermore, the user has the option of manually inputting the IP address of the Hololens and the ports over which the application will try to connect over the network. Once the user picks the preferred settings and presses the Start button, the Kinect is activated, after which, by pressing Space on the keyboard, the recordings and the transmission of the recording starts.

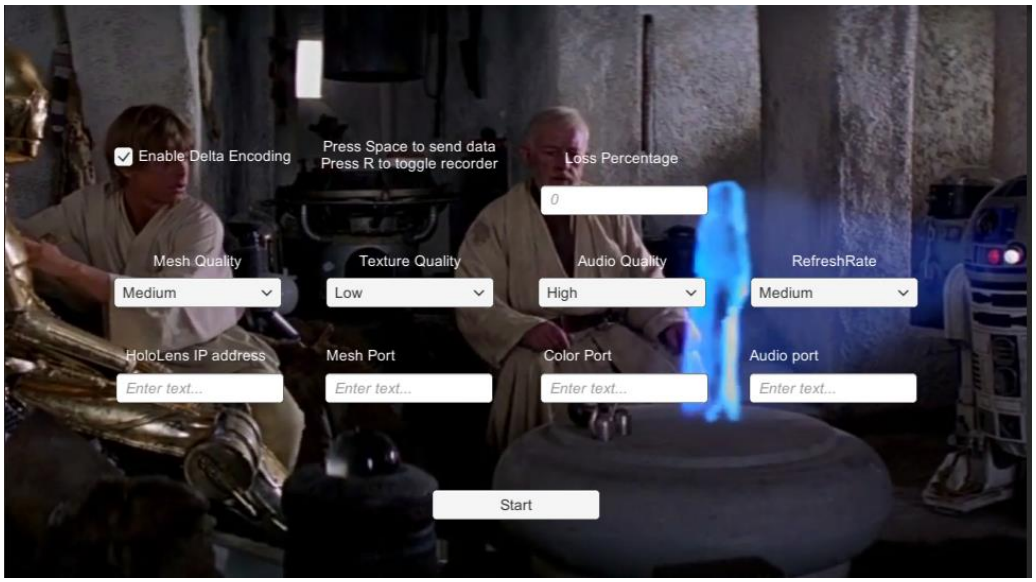

Image 14. GUI of HoloTalk

The receiving user's Hololens application, on the other hand, initially has no GUI, and only waits for the incoming connection from a client. Once the client successfully connects and relays the first frame, the 3D image ("hologram") of the sending user is displayed "in front" of the receiving user, on his/her Hololens display. Together with the 3D image, the control wireframe appears, which enables the receiving user to interact with the 3D image of the sending user by dragging and placing it somewhere in real space. As the manipulation and placement of 3D images in Hololens extends beyond the scope of this thesis, further information may be found in the HoloToolKit [26] documentation page.

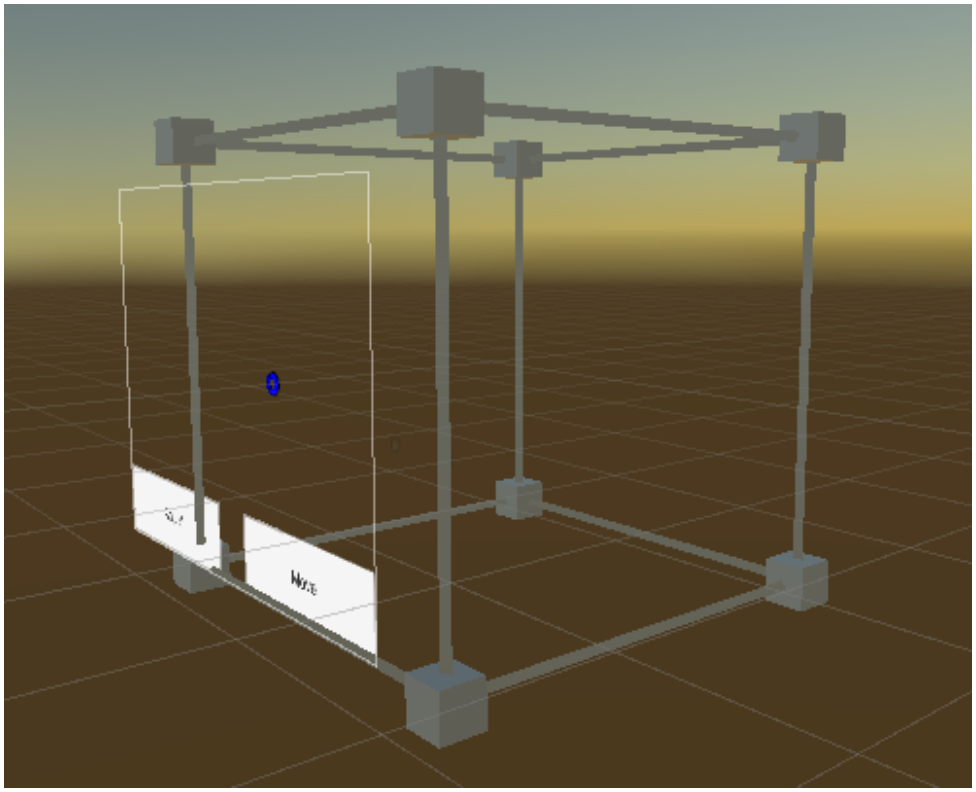

Image 15. Hologram frame for scaling and placement
of the received 3D image in the real world

3.8 Further advancements and remarks

Before analyzing the performance of the developed solution, one should note that there are many possible improvements to advance HoloTalk from a research prototype to a “polished” and finished solution. As previously mentioned, the largest single improvement that can be made is the full implementation of HoloTalk over a public Internet connection. While for performance purposes the implemented video chat could be used to test the limitation of the implemented compression algorithms, a fully functional 3D conversation application over a network connection would provide more relevant network data. Thusly, most of the testing done in the next chapter was made over a LAN connection for the purpose of measuring the amount of network traffic needed to make HoloTalk work.

Other than that, most of the other improvements are bug resolves, unexpected errors and results, such as the high-quality feed from the Kinect camera not aligning properly with the mesh resulting in only half the mesh being covered by the texture. This and other known bugs and errors have been listed in the software documentation accompanying this thesis.

4. Performance evaluation methodology

In this section, we describe the methodology which was applied for measurement and evaluation of the solution.

4.1 Performance

The first round of performance tests was carried out to record and analyze the performance of HoloTalk itself, and record the amount of processing power needed to ensure the stability of both the PC application and the application on the Hololens. For the PC application, the test machine ran an Intel Core i7-4702MQ 2.2 GHz CPU with 16 GB DDR3 L RAM, and had an NVIDIA GeForce GTX 850M graphics processing unit. Based on these specifications, the testing PC can be considered an “above average” machine, or even a lower-end high-performance machine. Due to the currently high price and limited availability of Hololens, a higher performance PC was selected, considering that an average user who has access to a Hololens would also have access to such a PC.

The testing scenario is as follows: The user powers on the PC application, and goes through all the possible quality settings while communicating with the Hololens application for a limited time, in this instance, for 30 seconds. It obvious that for every increase in quality, a drop-in framerate was experienced, but what is worth mentioning is the amount of performance decrease experienced and the bottleneck points that were detected in the testing procedure. A baseline measurement was established to work as a starting point. Using the following combination of medium Sample Size, low Texture Quality, medium Audio Quality, enabling the Delta processing and medium Refresh Rate a base framerate was created. Shown in Image 16 is the baseline combination of parameters positions next to the actual person (tester), as viewed from the Hololens.

Image 16. 3D image (baseline parameters) placed next to the actual tester

4.2 Compression

Comparing the compression algorithms cannot be made directly, since each of the algorithms deals with different aspects and even different frames, so this chapter will deal with the general efficiency of the algorithms and conditions where they achieve their maximum and minimum. The efficacy of the algorithms was measured by recording the amount of data that is being transmitted between the devices as described in previous chapters. Total data was recorded for the 30 seconds that every quality setting was active and the numbers were compared.

The most straightforward element of the compression procedure was the Sample Size process, since it is a process with a fixed compression efficiency and is not affected by the type of depth frame it receives (as shown in Table 1, the array size for each quality level is double the size of the next lower level).

Next comes the Deflate algorithm, which can provide an interesting exercise in further improving the basic algorithm. Defining the maximum size and minimum size of the resulting array is trivial. The Deflate algorithm achieves its minimum when working with an array filled with identical values, for then an array of length n is reduced to an array of fixed length 3. On the other hand, the maximum is achieved when there is not a single repetition, or the maximum repetition increment is 2, due to the necessity to note down the character and number of repetition. A character that repeats only twice in a row cannot be compressed with the Deflate algorithm.

Finally, there comes the Delta compression. As with the Deflate compression, defining the minimum resulting size and the maximum resulting size is fairly simple. If there is no change made between two delta frames, then the next resulting delta frame is non-existent. Comparably, the maximum can be achieved by changing the entire delta frame, for example if the frame 1 is a solid color blue background, and the subsequent frame 2 is a solid color red background, then the entire delta frame needs to be resent, meaning that the full frame is resent.

4.3 Network traffic

As mentioned earlier, the developed solution was tested only over a LAN connection. On the other hand, as mentioned in chapter 3.4, an additional prototype was built for video chat communication over the Internet, using Web-RTC technology.

The prototype was tested using a connection with the capability of having 1Gb upload and 1Gb download speed, due to their availability, and naturally the prototype worked well in that environment. After the initial connection was established, Wireshark [27] analysis was conducted to observe the network data, in addition to collecting other information on the amount of data necessary. The Wireshark analysis was conducted on the 3 ports mentioned earlier in this thesis, and the amount of data send over a TCP connection open over those ports was recorded for a period of 30 seconds for each of the four Sample Size settings.

Furthermore, this prototype is reliant on the availability of the STUN server. In the server fails or goes offline, it is not possible to make the initial handshake; thus, it is important to always maintain the availability of the server.

The primary problem of measuring the amount of network data needed for HoloTalk to work is the inconsistency of the results, with a wide variety of measurements gained as a result of the testing. What this means is that an extensive testing procedure should be made in future analysis of HoloTalk.

5. Results

5.1 Performance

Even with a high-end PC, getting HoloTalk to perform at an acceptable framerate was not a trivial task, demonstrating the massive amount of data processing needed to ensure all the elements ran smoothly. For the baseline combination of factors mentioned in the previous chapter (medium Sample Size, low Texture Quality, medium Audio Quality), a frame rate of 90 FPS was achieved. It is also worth mentioning that for this baseline combination the user was situated a meter and a half from the Kinect device.

The resulting 90 FPS was deemed acceptable for the working conditions of HoloTalk for several reasons. The primary reason was that the maximum refresh rate for HoloTalk was capped at 30 FPS, due to the amount of data that needed to be processed, as it would have been a daunting task to present a working solution faster than 30 FPS. While the conclusion that 30 FPS is enough to provide a smooth video for the user is derived from the fact that the PC application has hardly any interaction after the initial setup, and requires no user attention once the recording starts, further testing should be performed to confirm this framerate, if needed.

The first factor that was altered was the Sample Size (Image 17). Changing the Sample Size to Low resulted in a massive improvement in the framerate, with the resulting framerate reaching as high as 180 FPS, primary because of the much-decreased size of the depth frame. On the other hand, the quality of the mesh faced the biggest drop in quality. It is a strong suggestion for future research to conduct a large-scale test to determine the mesh quality level that will be acceptable to the average user. Since there was no time for such a testing during the time allocated for this thesis, what follows is the summary of informal opinions of the people who either worked on HoloTalk or helped to test it. The overwhelming majority agreed that the drop in quality was much larger than the increase in framerate, and that the low quality should only be used in instances where either the performing computer is subpar or operating on a network limited by upload speed.

On the other end of the spectrum, a High value of Sample Size setting was concluded to be the state that HoloTalk should aspire to achieve while maintaining

the working framerate. It achieves the best increase in quality while still maintaining an acceptable framerate, in this case close to 35 FPS. Once the quality level was moved to Unstable High the first bottleneck appeared. Not only does the framerate drop down to an unacceptable 15 FPS, because of the massive size of the array that needs processing, but here the risk of a memory overflow is not insignificant. As mentioned before, at this quality level there is a risk of the size of the array being larger than the maximal allocated size given to a Unity mesh, which if overflows breaks the application and the user experiences a drop-in communication.

In conclusion, currently the High quality setting represents the limit that can be achieved on a high-performance machine and it presents the appropriate benchmark to be used for potential future development of HoloTalk.

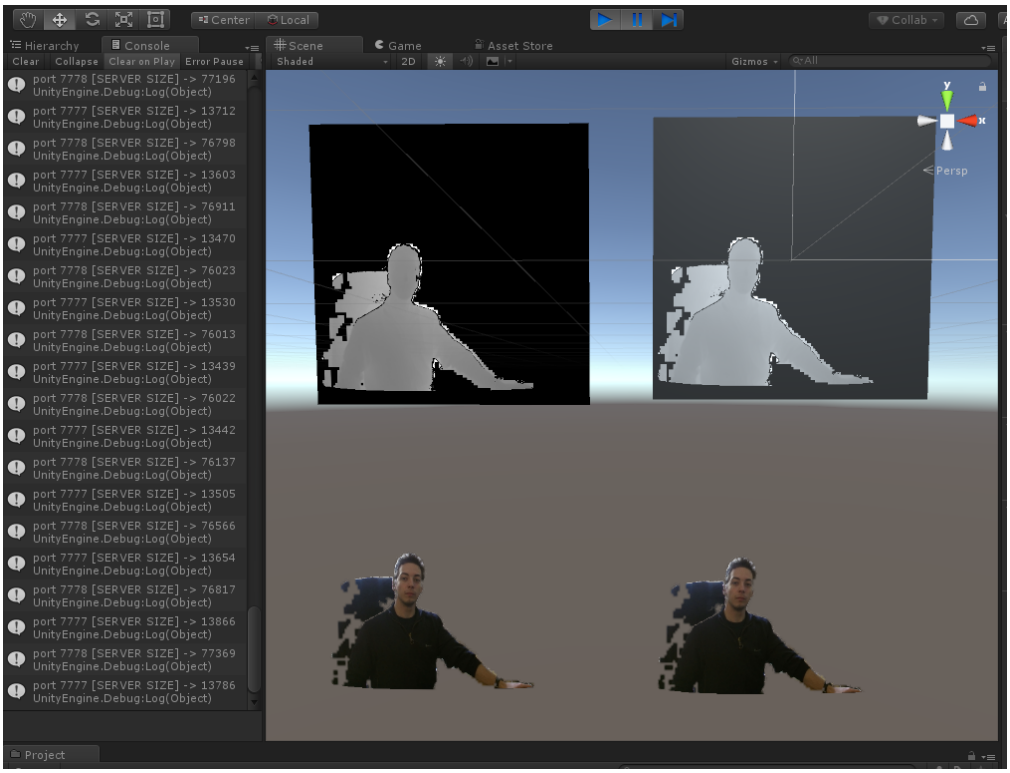

Image 17. Framerate testing with mesh quality settings

After that, testing moved on to the Texture Quality settings. The only option here, other than the starting point, was to increase the texture quality to High. The resulting increase did not only severely deteriorate the performance, but also exposed a bug due to the different ratio size between the low-quality and high-quality setting as described in previous chapters. The currently implemented solution had trouble with attaching the texture to the correct point on the mesh, resulting in a 3D image where half of the user representation is not aligned correctly. Image 18 shows a frame illustrating this problem. As such, not much testing was done to compare and it had to be left to be improved in future versions of HoloTalk.

Image 18. Incorrectly aligned 3D image

Similarly, Audio Quality had almost no effect on performance increase or decrease, as the implementation of the audio was a straightforward, out-of-the-box implementation, unlike the one used for the mesh and texture components.

Finally, testing was done on the Refresh Rate parameter. As mentioned before, further testing needs to be made to confirm this observation, but observing the drop-in quality with the Low and Medium quality setting proved to be far greater than the drop in performance quality. In conclusion, on a high-powered machine using this version of HoloTalk, a user can run the application with Sample Size, Audio Quality, and Refresh Rate set on High, and the Texture Quality set on Low, while still maintaining an acceptable user experience. This level of quality is likely to be downgraded on a lower-performance computer, or over a slower network connection.

One final step was made, to observe what happens when the user disables the Delta compression option. While disabling this option greatly increased the required network bandwidth (due to sending full frames all the time), it provided no great increase in performance. As mentioned earlier in the thesis, the process of the compression itself was parallelized early in the process, to ensure that performance does not drop.

Finally, there is the problem of the Kinect bottleneck itself, it appears that once the user approaches the Kinect to closer than one meter, a notable drop in performance appears – as much as two or even three times the reduction in framerate. If this is a problem with the Kinect or the SDK responsible with communication to Unity remains to be seen, but it was deemed not to be related to HoloTalk as such, and should be analyzed further in projects that deal with similar topics.

5.2 Compression

While the maximum and minimum in the Deflate Algorithm are rarely achieved in practice, it is worth noting that there are types of arrays that are close to the minimum which would benefit greatly from an improved algorithm, at the loss of converting this algorithm from a lossless to a lossy compression. Let us observe the following array as an example [1,1,1,1,2,1,1,1,1]. If we apply the algorithm "as is", it will compress the first four elements, skip the fifth element, and compress the last four elements. Now if we imagine that these values represent color values, ignoring for a second that the numbers are not ideally presented in a RGB format, one can conclude that the color difference between the fourth, fifth and sixed elements is negligible and would not be seen easily with the human eye. (This concept is similar to quantization of a matrix of DCT coefficients within JPEG compression process.)

What could be done to improve the algorithm is that while the algorithm linearly passes through the array (which it must do anyway), it checks the closest neighbor of each elements and if an element is surrounded with identical elements which are within a pre-defined threshold that element is changed to the surrounding values. The resulting array with this new improvement would then be [1,1,1,1,1,1,1,1,1], resulting in a minimum solution and a fixed array size of 3, greatly improving the compression rate, while still sacrificing only a small portion of the resulting quality. Naturally this is only a general idea and a practical implementation would be a more complex solution, considering variables such as two-dimensional arrays, the value of the threshold itself and other not yet detected problems and unexpected side effects.

In practical terms, the Deflate algorithm, when used on the color frame in this solution, was concluded to provide an adequate compression rate. Due to the initial background removal applied to the color frame, most of the resulting color frame is a uniform black color, very suitable for the Deflate algorithm. An average compression rate was found to reduce the original 512×424 image which resulted in 217,088 pixels to about 100,000 pixels depending on an unusual factor. Interestingly enough, the primary factor in the efficiency of this algorithm in this specific implementation is the position of the user itself compared to the origin point of the Kinect.

As can be observed in Image 19, depending on where and how the user is standing with respect to Kinect, there can be larger number of repeated characters detected within the image. For example, if the user is standing in the center of the Kinect's "view", then the array will effectively be divided vertically (and the repeated sequence broken at that point), while if the user is positioned so that only his or her upper body part is "seen" by the Kinect, the unobstructed upper part of the array can be reduced to the minimum size. What this means is that the users can affect the compression rate by different behaviors, for example by only showing the upper body when the network bandwidth is low, thus improving the Deflate compression rate.

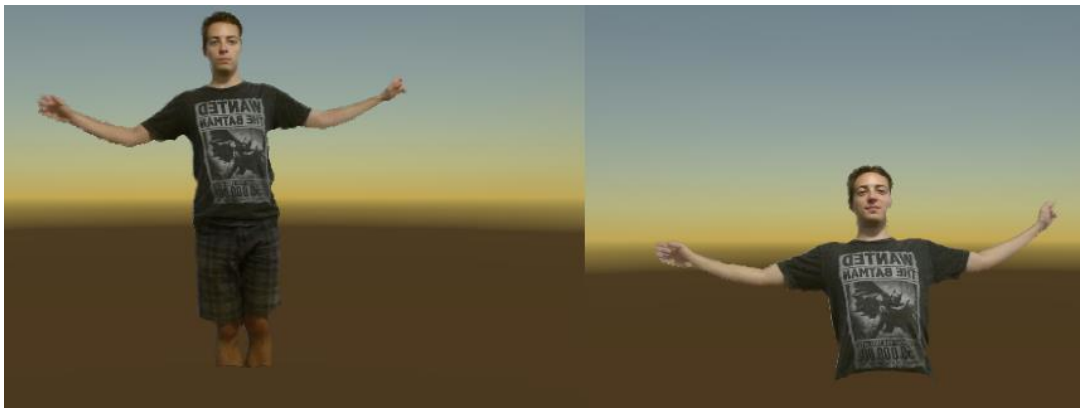

Image 19. Positions of user with respect to the Kinect (whole body, centered on screen, as opposed to the upper body only with half of the screen free)

As with the Deflate algorithm, the user can change the compression ratio with his or her positioning and movement with the Delta algorithm as well. Grand sweeping motions and swift movements in front of the Kinect will ensure that most of the frames sent over the network will be full frames, while slow and steady movement will benefit the Delta compression. In conclusion, the algorithms implemented here work best in scenarios where a normal conversation simulation is presented, with a steady, no movement, only upper body image shown on screen.

5.3 Network traffic

Doing some basic calculations, we first estimate the largest possible amount of data. The hard limit for the mesh as stated is 65,536 vertices, stored as a Vector3 (4-byte sized) variables, with the corresponding data rate of 262,144 bytes per frame or 15 Mb per second. The color frame is a fixed 512×424 sized array of 4-byte variables which corresponds to 868,352 bytes per frame, or 192 Mb per second. Thus, there is both a large difference between the data traffic necessary to send the color frame compared to the depth frame.

Fortunately, due to the nature of the color frame and the large black areas created by the Background Removal, the Deflate algorithm presents a dramatic decrease in color frame size. Depending on where the user is standing with respect to the Kinect, about 20 to 25 percent of the frame size, at worst, is occupied by the user image, meaning that the rest of the frame can be reduced drastically in size. Assuming some margin of error, let us presume that the Deflate algorithm decreased the initial color frame to one third of the original size, leaving us with 64 Mb per second. Now this is where the Delta helps, if the user is standing still or making small movements the resulting delta frame is of insignificant size, and could be reduced in size by an order of magnitude smaller than the full frame.

If the resulting delta frame rate is 8 Mb per second, not taking into account the need to resend the full frame once every set interval, which in this solution was set at every three seconds. It is worth mentioning that the compression algorithms presented here are clearly inferior to commercially available solutions, due to the nature of the architecture explained in previous chapters. With additional time and expertise, all the algorithms presented here could be improved resulting in the number quoted for the Web-RTC solution. Following that, results shown in Figure 12 for all four levels of Sample Size quality settings averaged from the Wireshark recording, described in the previous chapter. As one can see, they are in line with the conclusion drawn in this chapter.

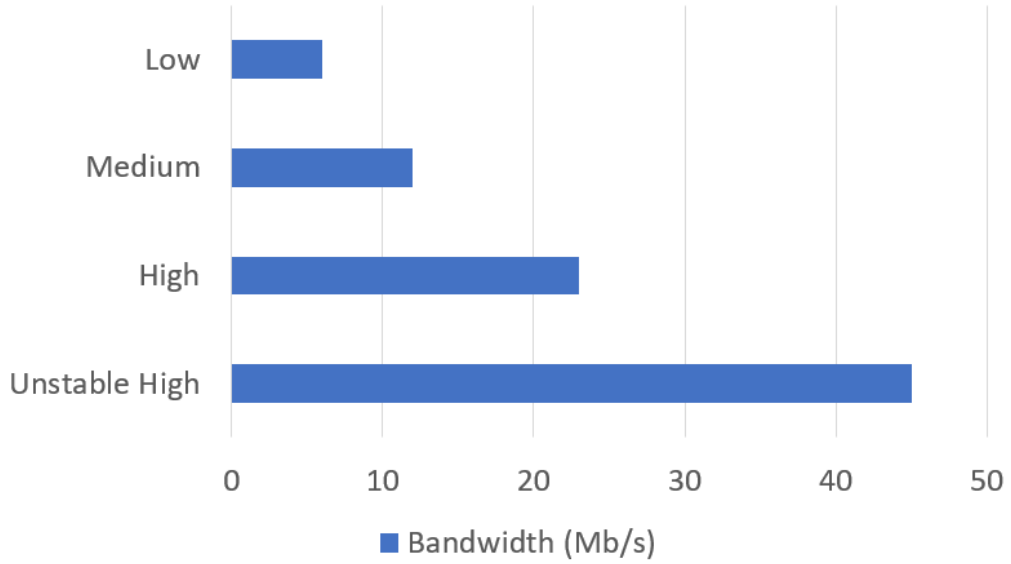

Figure 12. Average bandwidth for the four Sample Size quality settings

5.4 Further advancements and remarks

HoloTalk is a proof of concept solution and its functions are not optimized. This is in part due to the fact that most of the software had to be written from scratch, as dictated by the requirements of the Hololens architecture and software compatibility issues.

Further testing regarding the user quality of experience could also improve the current solution to gain insight on where cuts could be made on the performance while still maintaining a desired quality, and where there needs to be greater improvement in the performance. As such, HoloTalk presented here can serve as a foray into the further research in this topic.

6. Conclusion

In conclusion, having presented the research done in this thesis, the potential for 3D conversation is there, if not fully researched just yet. At its currently level of implementation, HoloTalk can serve as a proof-of-concept for further research into the field. The current state of HoloTalk is a prototype system that can establish 3D conversation with another user over a local area network with an average amount of data being less than 10 MB per second.

Furthermore, the compression algorithms used in this thesis can also serve as a starting point for further improvements to be made on the network and transfer level. While the network traffic can be reduced to 6 Mb/s on the setting low of Sample Size, for future advancements it would benefit HoloTalk greatly if the network traffic was reduced to amounts similar to those achieved by modern video streaming services.

Finally, the technologies used to build and create HoloTalk are in their infancy, and it can be assumed that they will evolve in the coming years. Potentially, one can speculate that down the line the need for having smart glasses would disappear and a fully holographic solution be possible.

Literature

[1] Star Trek Holodeck - http://memory-alpha.wikia.com/wiki/Holodeck

[2] Microsoft Holoportation - https://www.microsoft.com/en-us/research/project/holoportation-3/

[3] Microsoft Hololens - https://www.microsoft.com/en-us/hololens

[4] Hololens-Kinect - https://github.com/michell3/Hololens-Kinect

[5] Vector3 - https://docs.unity3d.com/ScriptReference/Vector3.html

[6] LiveScan3D-Hololens - https://github.com/MarekKowalski/LiveScan3D-Hololens

[7] Kowalski, M., Naruniec, J., Daniluk, M. "LiveScan3D: A Fast and Inexpensive 3D Data Acquisition System for Multiple Kinect v2 Sensors," in Proc. of the 2015 International Conference on 3D Vision (3DV), 2015, Lyon, France, 2015 [Available: https://www.researchgate.net/publication/308807023_Livescan3D_A_Fast_and_Inexpensive_3D_Data_Acquisition_System_for_Multiple_Kinect_v2_Sensors]

[8] Milgram, P. and Kishino, F. "Taxonomy of Mixed Reality Visual Displays," IEICE Transactions on Information and Systems, vol. E77-D, no. 12, December 1994, pp. 1321-1329 [Available: https://www.researchgate.net/publication/231514051_A_Taxonomy_of_Mixed_Reality_Visual_Displays]

[9] Ghost in The Shell example hologram http://www.empireonline.com/movies/features/ghost-shell-exclusive-vfx-breakdown/

[10] Videoplace- http://thedigitalage.pbworks.com/w/page/22039083/Myron%20Krueger

[11] Google Glass - https://www.google.com/glass/start/

[12] Microsoft Kinect - https://developer.microsoft.com/en-us/windows/kinect

[13] Unity Game Engine - https://unity3d.com/

[14] HoloToolKit - https://github.com/Microsoft/HoloToolkit-Unity

[15] Kinect SDK- https://www.microsoft.com/en-us/download/details.aspx?id=44561

[16] Unity Documentation - https://docs.unity3d.com/Manual/index.html

[17] Hololens specification - https://www.windowscentral.com/hololens-hardware-specs
[18] Deflate Explanation - https://zlib.net/feldspar.html
[19] Delta Explanation - http://www.dspguide.com/ch27/4.htm
[20] Unity Networking HLAPI - https://docs.unity3d.com/Manual/UNetUsingHLAPI.html
[21] Writing TCP sockets in C# - https://msdn.microsoft.com/en-us/library/system.net.sockets.tcpclient(v=vs.110).aspx
[22] WebRTC- https://webrtc.org/
[23] Stun server - https://www.3cx.com/pbx/what-is-a-stun-server/
[24] Stun server list - https://gist.github.com/zziuni/3741933
[25] Simple Mesh Serializer - http://wiki.unity3d.com/index.php?title=MeshSerializer2
[26] HoloToolKit documentation - https://github.com/Microsoft/HoloToolkit-Unity/blob/master/GettingStarted.md
[27] Wireshark - https://www.wireshark.org/

Summary

This master thesis briefly presents the basics of mixed and augmented reality, followed by the design and implementation of an experimental augmented reality service for 3D conversation at a distance, called HoloTalk. Functional blocks of the service as described, focusing on capturing, processing, compression and transfer of 3D user data representation in real time over the communication network. The current version of HoloTalk uses local network communication, while a possible solution for use in the public Internet is outlined in the thesis. The developed proof-of-concept version of HoloTalk is based on Kinect as a motion tracking device, Unity as a virtual reality/augmented reality game engine, and Hololens mixed-reality glasses as display. The thesis also presents a performance analysis of HoloTalk for different quality settings, as well as the characteristics of its compression algorithms and network data traffic.

Key words:

Augmented reality, network transfer, data compression, 3D conversation

Sažetak

Diplomski rad ukratko opisuje osnove miješane i proširene stvarnosti te dizajn i implementaciju eksperimentalne usluge proširene stvarnosti za 3D razgovor na daljinu, nazvane HoloTalk. Razrađeni su funkcijski blokovi usluge, s naglaskom na prikupljanje, obradu, kompresiju i prijenos podataka za 3D prikaz sudionika razgovora u stvarnom vremenu putem komunikacijske mreže. Izvedena je komunikacija putem lokalne mreže te okvirno opisano rješenje za komunikaciju putem javnog Interneta. Programska izvedba koncepta HoloTalka zasniva se na uređaju Kinect za praćenje pokreta, pogonskoj platformi Unity za virtualnu i proširenu stvarnost i Hololens zaslonu za miješanu stvarnost. U radu su prikazane izmjerene su, odnosno analizirane performance razvijene inačice usluge HoloTalk za različite postavke kvalitete te karakteristike kompresijskih algoritama i mrežnog podatkovnog prometa.

Ključne riječi:

Proširena stvarnost, prijenos podataka, kompresija podataka, 3D razgovor