Multi-Source Holoportation

Andy Pack

Mid-Term Report

Department of Electrical and Electronic Engineering
Faculty of Engineering and Physical Sciences
University of Surrey

Abstract

The scope and current state of the multi-source holoportation project is examined. The aim is to extend a suite of 3D video capture software from it's current capabilities supporting multiple sensors but a single captured environment to handling multiple surroundings concurrently during frame collection and display.

Currently the display methods have been developed in line with the specification in order to allow simultaneous display and arbitrary real-time placement within the display space.

The future work for the project is described including the current designs these endeavours. The bulk of this remaining work involves developing the network capabilities of the software to accommodate multiple sources, this is on track to be completed by May 2020.

Contents

List of Figures

Listings

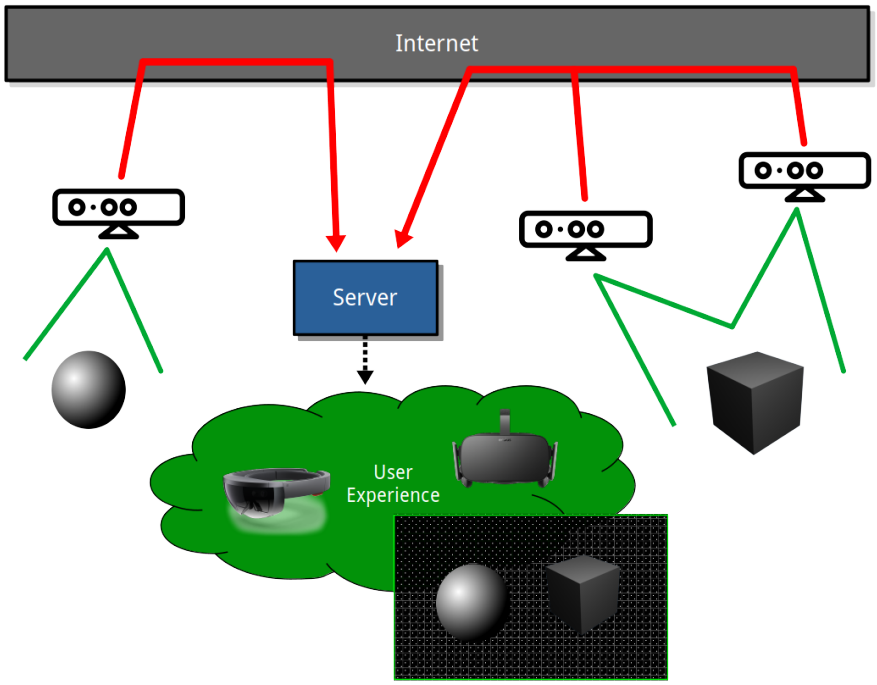

Figure 1: Demonstration of a multi-source holoportation system including single and multiple view camera configurations

1 Introduction

The aim of this project is to develop a piece of software capable of supporting multi-source holoportation (hologram teleportation) using the LIVESCAN3D[1] suite of software as a base.

As the spaces of augmented and virtual reality become more commonplace and mature, the ability to capture and stream 3D renders of objects and people over the internet using consumer-grade hardware has many possible applications.

This represents one of the most direct evolutions of traditional video streaming when applied to this new technological space.

A view of what multi-source achieves can be seen in figure 1. Both single and multi-view configurations of cameras are shown, the latter allowing more complete renders of the subject to be acquired. Both shapes are presented through the *user experience*, control schemes and visual language can vary between implementations across AR/VR and traditional 2D displays.

LIVESCAN3D is a suite of 3D video software capable of recording and transmitting video from client to server for rendering. The suite is fast and uses consumer grade hardware for capture in the form of XBOX KINECT cameras, it is used in various projects at the UNIVERSITY OF SURREY and has multiple setups in dedicated lab space.

LIVESCAN3D'S use of XBOX KINECT cameras allows the capture and stream of 3D renders in single or multi-view configurations using calibrated cameras however the server is only able to process and reconstruct one environment at a time.

The capability to concurrently receive and reconstruct streams of different objects further broadens the landscape of possible applications, analogous to the movement from traditional phone calls to conference calling.

2 Literature Review

The significance of 3D video like that captured and relayed using the LIVESCAN suite is related to the development of new technologies able to immersively display such video content. Therefore before discussing the specific extension that this project will make to the LIVESCAN software it is important to contextualise it within the space of 3D video capture while also considering it's implications for AR and VR applications.

2.1 Cross Reality (XR)

Cross reality is a broad term describing the combination of technology with a user's experience of their surroundings in order to alter the experience of reality. It is used as an umbrella term for virtual, mixed and augmented reality experiences and technology. Before continuing, the differences between these technologies is considered.

Virtual The replacement of a user's experience of their surroundings, rendering a new space that the user appears to inhabit. Typically achieved through face mounted headsets (*Facebook Oculus, HTC Vive, Playstation VR, Valve Index*).

Augmented The augmentation of a users surroundings by overlaying the environment with digital alterations. Can be achieved with translucent/transparent headsets *(Microsoft Hololens, Google Glass)* or through mobile experiences *(Android ARCore, Apple ARKit)* both when head mounted *(Google Cardboard, Google Daydream, Samsung Gear VR)* and handheld *(Pokemon GO)*.

Mixed A combination of virtual and augmented elements in order to allow interaction with an augmented reality. Can be achieved in different ways typically starting with either a typical AR or VR experience and including aspects of the other. At a higher level, mixed reality can be described as a continuous scale between the entirely real and entirely virtual with augmented reality occurring in between.

The burgeoning of these three forms of XR via consumer hardware such as the MICROSOFT HOLOLENS and OCULUS RIFT represents a new space for the consumption of interactive media experiences.

Although VR and AR headsets have accelerated the development of XR technology, they are not the only way to construct XR experiences. Jones, Sodhi, Murdock, *et al.*[2] demonstrate *RoomAlive*, an AR experience using depth cameras and projectors (refereed to as *procams*) to construct experiences in any room. This is presented through games and visual alterations to the users surroundings. A strength of the system is it's self contained nature, able to automatically calibrate the camera arrangements using correspondences found between each view. Experience level heuristics are also discussed regarding capturing and maintaining user attention in an environment where the experience can be occurring anywhere, including behind the user .

A point is also made about how the nature of this room based experience breaks much of the typical game-user interaction established by virtual reality and video games. In contrast to traditional and virtual reality game experiences where the game is ultimately in control of the user or user avatar, AR experiences of this type have no physical control over the user and extra considerations must be made when designing such systems.

Traditional media consumption is not the only area of interest for developing interactive experiences, an investigation into the value of AR and VR for improving construction safety is presented by Li, Yi, Chi, *et al.*[3]. A broad look at the applicability is taken with assessments including VR experiences for developing worker balance to aid in working at elevation and AR experiences incorporated into the workplace for aiding in task sequencing to reduce the effect of memory on safety.

Lindlbauer and Wilson[4] demonstrate an example of mixed reality through the use of KINECT cameras and a virtual reality headset. Users are placed in a virtual space constructed from 3D renders of the physical environment around the user. Virtual manipulation of the space can then be achieved with visual, spatial and temporal changes supported. Objects can be scaled and sculpted in realtime while the environment can be paused and rewinded. The strength of mixed reality comes with the immersion of being virtually placed in a version of the physical surroundings, tactile feedback from the environment compounds this.

2.2 Kinect and RGB-D Cameras

Initially designed as a motion control accessory for the XBOX, the KINECT is a series of depth aware cameras produced my MICROSOFT. The device uses additional infrared lights and sensors alongside an RGB camera in a configuration referred to as a time of flight camera to generate 3D renders of a surroundings. The device also includes motion tracking and skeleton isolation for figures in view.

Following the release of an SDK for Windows in 2012, Zhang at MICROSOFT RESEARCH reflects on the original camera's capabilities and the applications to computer vision research in [5].

Here 3D conference calling of the type described in the introduction without AR or VR applications is presented, instead users watch a composite conference space on a screen with all participants rendered within. Work was undertaken to

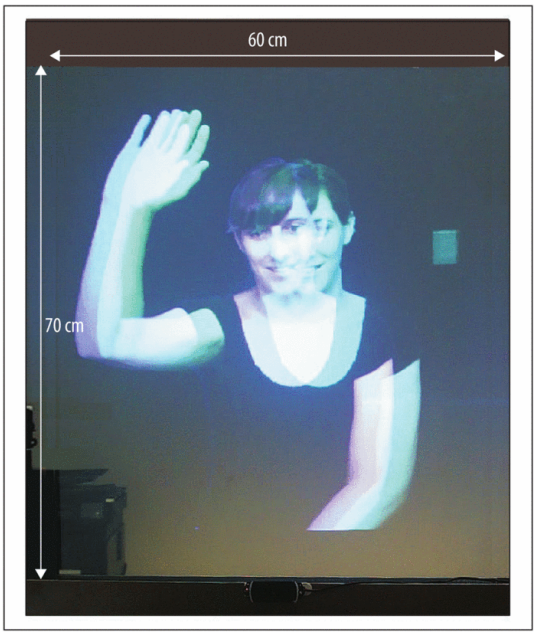

Figure 2: An example of stereoscopic projection of depth aware footage captured by Fuchs, State, and Bazin[11]

achieve mutual gaze between participants, a marked advantage over traditional conference calls where the lack of such aspects of group interaction make the experience more impersonal. Methods of achieving more natural virtual interactions or *telepresence* are covered in section 2.3.

A second version of the camera, v2, was released alongside the XBOX ONE in 2013 and presented many improvements over the original. A higher quality RGB camera captures 1080p video at up to 30 frames per second with a wider field of view than the original[6]. The physical capabilities of the camera are discussed by Lachat, Macher, Landes, *et al.*[7]. The second version of the camera was found to gather more accurate depth data than the original and was less sensitive to daylight. Wasenmüller and Stricker[8] found similar results with the v2 achieving higher accuracy results over the original. The second version did, however, achieve lower precision results than the v1 with recommendations made to include pre-processing on acquired depth images to control for random noise, *flying pixels* and *multipath interference*.

The KINECT is used successfully by Nissimov, Goldberger, and Alchanatis[9] for object detection in the context of an autonomous vehicle navigating a greenhouse. The depth information was used in conjunction with the RGB information to identify obstacles, while the paper lays out some limitations of the camera it was found to be effective in it's aim and was capable of running on a reasonable computer.

This second iteration on the KINECT is frequently used in computer vision experiments with many of the works cited here using it for acquisition.

2.3 Holoportation and Telepresence

The term Holoportation is defined and exemplified in a MICROSOFT RESEARCH paper by Orts, Rhemann, Fanello, *et al.*[10] where an end-to-end pipeline is laid out for the acquisition, transmission and display of 3D video facilitating real-time AR and VR experiences. The MICROSOFT RESEARCH paper builds on works including by Fuchs, State, and Bazin[11] 2 years earlier which describes attempts at achieving "*telepresence*", a term coined by Marvin Minksy to describe the transparent and intuitive remote control of robot arms as if they were the controllers own[12]. The term was broadened by Bill Buxton[13] to include the space of telecommunications to describe technology being used to make someone feel present in a different environment. In the context of holoportation this is through the use of 3D video reconstruction. The aforementioned work by Fuchs, State, and Bazin[11] used 10 KINECT cameras to capture a room before virtually reconstructing the models.

In service of demonstrating it's applicability to achieving *telepresence*, a figure was isolated from the surroundings and stereoscopically rear-projected onto a screen for a single participant, a result of this can be seen in figure 2.

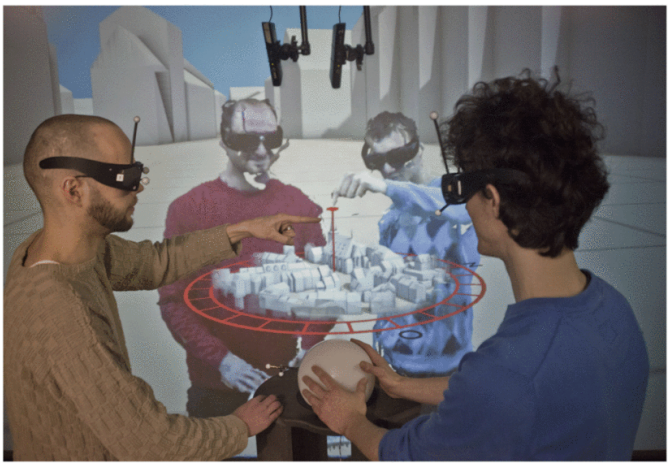

Figure 3: World in Miniature render demonstrated in a multi-source holoportation context by Beck, Kunert, Kulik, *et al.*[15]

The MICROSOFT RESEARCH paper demonstrates a system using 8 cameras surrounding a space. Each camera captured both Near Infra-Red and colour images to construct a colour-depth video stream, a more complex camera configuration than in the others cited.

Fender and Müller[14] demonstrates a similar holoportation experience to LIVESCAN3D capable of supporting multi-view configurations, it also supports both point clouds and meshes. Calibrating multiple view points is completed using the extrinsics and intrinsics of the camera. The extrinsics are the relative positions of each KINECT camera while the intrinsics describe the internal properties of each camera, the focal length and optical centre.

The intrinsics of the KINECT camera can be retrieved from the KINECT SDK while the extrinsics are obtained in one of two ways. Extrinsics can be imported and parsed from XML for manual selection or estimated using OPENCV and a checkerboard pattern. When considering holoportation systems of this kind, comparatively few implement multiple views as a result the increased complexity involved in calibration.

2.4 Multi-Source Holoportation

The space of multi-source holoportation has been explored by Beck, Kunert, Kulik, *et al.*[15] in the context of shared architectural design spaces in virtual reality similar to a conference call. Two groups of people were captured in 3D using clusters of KINECT cameras before having these renders transmitted to the other group. Each group reconstructs the other's render for display in virtual reality in conjunction with their own. In doing so a shared virtual space for the two groups has been created and it can be seen to implement the process of holoportation. The strength of the system as a shared architectural design experience is emergent of the semantics of the virtual space where a World in Miniature (WIM) metaphor is used.

The Worlds in Miniature is described by Stoakley, Conway, and Pausch[16] as a set of interfaces between the user and the virtual space they experience using tactile and visual tools. The interface involves providing the user with a miniature render of the world they are inhabiting that can interacted with in order to affect the full scale environment around them.

This navigation tool maps well to Beck, Kunert, Kulik, *et al.*'s[15] architecture groupware design, an image captured during the work can be seen in figure 3.

3 LiveScan3D

LIVESCAN3D is a suite of software developed by Marek Kowalski, Jacek Naruniec and Michal Daniluk of the Warsaw University of Technology in 2015[1]. The suite utilises the XBOX KINECT v2 camera to record and transmit 3D renders over an IP network. A server can manage multiple clients simultaneously and is responsible for processing, reconstructing and displaying the renderings in real-time.

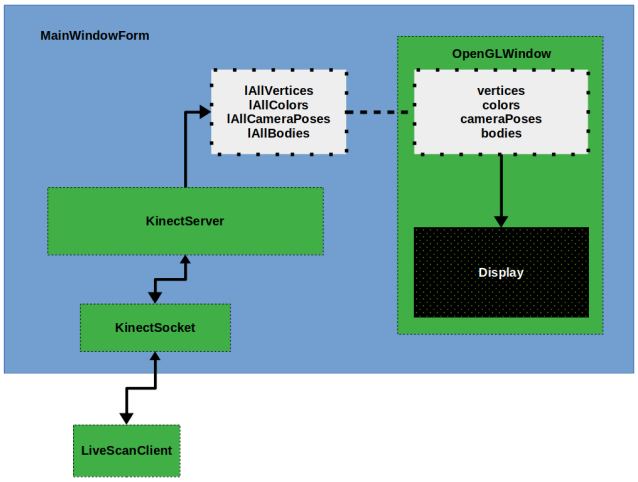

Figure 4: Initial architecture of the LIVESCAN3D server

These renderings take the form of a point cloud, a collection of 3D co-ordinates indicating the position of each voxel (3D pixel) and it's associated RGB colour value. As a result of it's analogous nature to a traditional frame of 2D video, the terms "render", "point cloud" and "frame" are used interchangeably from here.

The majority of the development being conducted in this project is regarding the server component of the software and as such this is covered in more detail.

3.1 LIVESCAN Client

The LIVESCAN Client is responsible for interfacing with the KINECT sensor via the KINECT v2 SDK and transmitting frames to the LIVESCAN Server. Body detection takes place client side, as does calibration when using multiple sensors.

Only one KINECT sensor can be connected to each computer as a result of the SDK's restrictions. A system used by multiple clients in this way lends itself well to multi-source configurations over the internet.

3.2 LIVESCAN Server

The server component of the LIVESCAN suite is responsible for managing and receiving 3D renders from connected clients. These renders are reconstructed in an interactive OPENGL window. When considering the code architecture of this application there are three main components.

OpenGLWindow Presentation layer of the application. Separate window spawned by the LIVESCANSERVER responsible for drawing point clouds and responding to user control.

KinectServer Network layer of the application. The main window make requests of this component to receive transmitted point clouds.

KinectSocket Child objects contained within the KINECTSERVER. A traditional network socket object representing a single TCP connection between the server and a client.

This structure can be seen in figure 4.

Received frames in the form of lists of vertices, RGB values, camera poses and bodies overwrite shared variables between the main window and the OPENGL window.

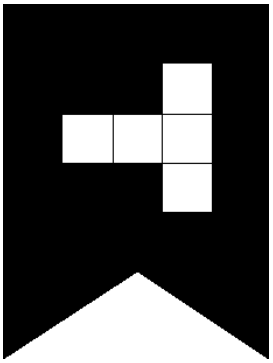

Figure 5: Example marker used within the LiveScan3D calibration process

3.3 Calibration & Multi-View Configurations

When using a single client setup, frames are transmitted in their own coordinate space with the origin defined as the KINECT camera and the captured scene rendered in front of it.

When using multiple sensors, the server would be unable to combine these unique Euclidean spaces without knowledge of the sensors positions relative to each other, the extrinsics of the system.

In order to make a composite frame a calibration process is completed client side following instruction by the server.

Calibration is completed in two steps, an initial estimation followed by a refinement process. The initial estimation is completed by informing the server of which calibration marker layouts are being used within the space. Client's identify possible visible markers like that seen in figure 5 using thresholding. Following this identification, the location of the marker can be found within the sensors coordinate system.

This information can be used to transform points from the cameras coordinate system to the markers frame of reference. As the relative locations of different markers are defined at the server, a world coordinate system can be defined as the centre of these markers. Typically 4 different markers are placed on the faces around the vertical axis of a cuboid allowing views in 360°.

This world coordinate space has shifted the origin from being the position of the single KINECT sensor to being a point in the centre of the calibration markers that each camera now orbits. As part of this calibration process the server distributes transformations to each client defining where they sit within this world coordinate space. Client's can now transform acquired renders from their own frame of reference to the world coordinate system at the point of capture and each point cloud can be merged coherently.

The refinement process is completed server side by requesting a single frame from each connected client and using Iterative Closest Points[17] (ICP) to improve the inter-camera relationships.

The OPENGL display space has it's origin within the centre of the visible box, this means that for single sensor setups this is also the location of the camera.

3.4 Design Considerations

When assessing LIVESCAN's suitability for extension to a multi-source context, the original network design should be investigated.

The original applications were best suited to a local environment as a result of many of the network functions being blocking. Should any delays or interruptions have occurred during a network operation, then the application would need to stop and wait for remediation before continuing. Interruptions of this type are more common when moving from a LAN environment to communicating over the open internet.

From a network perspective the need to make these actions non-blocking would present benefits for both multi-source and multi-view configurations.

Additionally, the network polling rates are higher than the frame rate of the produced video, when the server requests a frame before a new one has been captured by the client, the same previous frame is resent. This presents unnecessary bandwidth usage.

Moving to a multi-source context implies transmitting over the internet as opposed to local operation, this will make blocking actions and bloated bandwidth more dangerous to user experience.

Work has been undertaken that allows multiple concurrent TCP connections to be used by each client to increase bandwidth. Further work is being undertaken to un-block network actions.

4 Current Work

The required development to take the existing LIVESCAN codebase to the desired multi-source result can be split into two areas of concern.

Network The network layer of the LIVESCAN server must be updated in order to accommodate multiple clients logically grouped into "sources" for which separate frames are collected for display.

Display Finally the display element of the server should be extended to allow the simultaneous presentation of multiple point clouds. These objects should be individually arrangeable in the display space allowing both movement and rotation.

As of January 2020 the native method for displaying renderings, the server's OPENGL window, has been modified such that it can construct and render point clouds from multiple sources. To do so a dynamic sub-system of geometric transformations has been written in order to coherently arrange sources within the space when reconstructed. The default arrangements can be overridden with keyboard controls facilitating arbitrary placement and rotation of separate sources within the window's co-ordinate space.

4.1 Geometric Transformations

Within the LIVESCAN3D server source code are utility structures and classes which were extended in order to develop a wider geometric manipulation system. Structures defining Cartesian coordinates in both 2D and 3D spaces called POINT2F and POINT3F respectively are used in drawing skeletons as captured by the KINECT camera. There is also a class defining an affine transformation, the definitions for all three can be seen in appendix A.

Affine transformations are a family of geometric transformations that preserve parallel lines within geometric spaces. Some examples of affine transformations include scaling, reflection, rotation, translation and shearing.

The class definition is made up of a three-by-three transformation matrix and single 3D vector for translation, within the native codebase it is used for both camera poses and world transformations.

A camera pose is the affine transformation defining the position and orientation of the KINECT camera when drawn in the OPENGL space as a green cross. The world transformations are used as part of the calibration process when using multi-view configurations.

When considering how each source's render would be arranged in the space, the use of this class definition was extended. As the use of affine transformations is mostly limited to use as a data structure within the base source code, some utility classes and functions were required in order to fully maximise their effectiveness.

4.1.1 Transformer

The motivation in writing the TRANSFORMER was to create a generic framework of geometric transformations that could be utilised by the OPENGL display to arrange separate point clouds. At a high level this is done by implementing matrix arithmetic functions in the context of their use for applying linear transformations to Cartesian coordinates.

The TRANSFORMER class has static methods to apply AFFINETRANSFORMs to both POINT3F structures and lists of raw vertices as received from LIVESCAN clients.

Figure 6: Initial multi-source composite testing frame

Additionally there are utility functions to bidirectionally cast between POINT3F data structures and the lists of received vertices.

Finally static methods generate common rotation transformations about each axis given an arbitrary angle. This provided a foundation on which to define how the OPENGL space would arrange separate sources within it's combined co-ordinate space.

Currently missing is the ability to combine transformations into compound matrices. Applying multiple transformations to large numbers of coordinates would be more computationally expensive than applying one compound matrix and when running in realtime this should be considered. This is not yet included due to the current lack of need to apply multiple successive transformations. If the need were to arise following further refinements, it would be implemented as described here.

4.2 Separation of Network and Presentation Layer

During initial testing frames received from a live sensor were intercepted and serialized to XML in local storage. These frames were loaded into memory as the server was started and merged with those received live before display.

The composite frame can be seen in figure 6.

The objects can be seen to be occupying the same space due to their similar positions in the frame during capture. This is not a sufficient solution for displaying separate sources and so geometric transformations like those described above were employed to separate the two. The change in software structure at this stage can be seen in figure 7. A rotation of 180° in the vertical (y) axis pivoted the frames such that they faced those being received live, the results can be seen in figure 8.

At this point it was noted that transforming and arranging figures within the main window before passing the OPENGL window a complete point cloud spreads responsibility for the display logic to the main window.

LIVESCAN3D is capable of supporting more display methods than just the native OPENGL implementation with versions available for both MICROSOFT HOLOLENS and Mobile AR applications. Therefore when designing the multi-source capabilities, the separation of logic between the network and presentation layer is important.

The way in which the OPENGL window arranges the figures in it's display space should be defined by the OPENGL window itself. The network layer should be display agnostic and not make assumptions about how the display will process figures.

In order to follow this design the transformations were moved to instead occur within the OPENGL window class. To allow this the shared variables between the MAINWINDOW and OPENGL were changed. A Frame structure was defined to wrap an individual point cloud with a client ID to allow differentiation, the definition can be seen in appendix B.1. The structure holds fields for each of the lists previously shared between the two objects including a list of vertices (co-ordinates) and the RGB values for each as well as the camera poses and bodies.

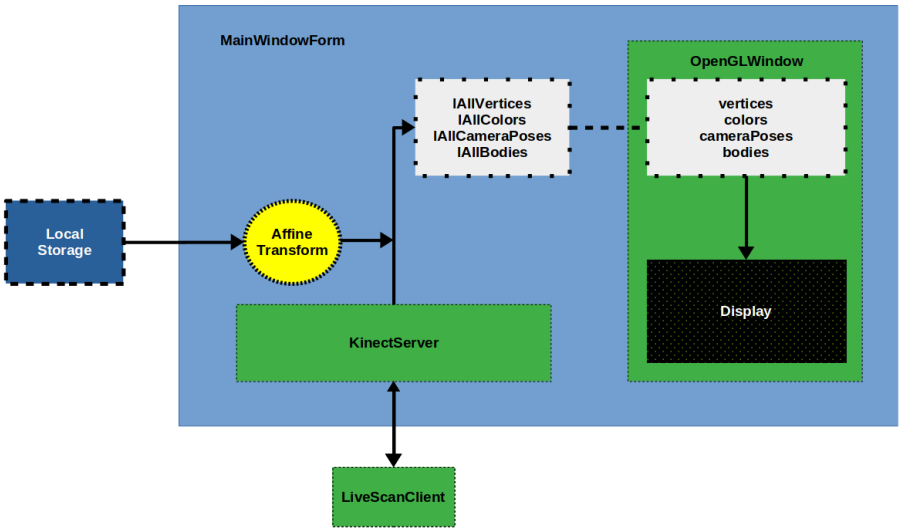

Figure 7: Initial testing process transforming frames loaded from local storage

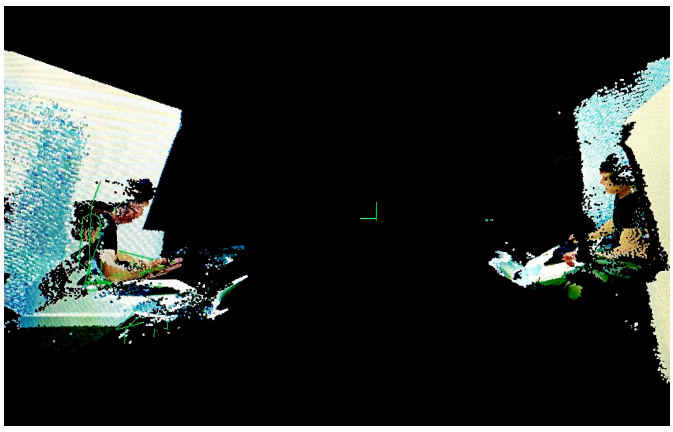

Figure 8: Composite testing frame following 180° rotation of recorded source in y axis

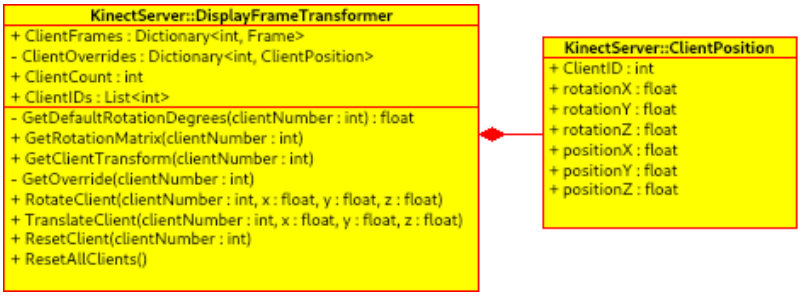

Figure 9: UML diagram for DISPLAYFRAMETRANSFORMER

The original LIVESCAN3D cleared each of these variables before retrieving a new frame, when moving to a multi-source architecture the ability to individually update source point clouds was prioritised. This would avoid blocking the entire display when unable to receive frames from a specific client, other clients would still be able to have frames updated promptly.

To accomplish this a dictionary was used as the shared variable with each client's frame referenced by it's client ID. In doing so only one frame per client is kept and each new frame overrides the last. During rendering the dictionary is iterated through and each point cloud combined. During combination a client specific transformation is retrieved from an instance of the DISPLAYFRAMETRANSFORMER class. This object is a member of the OPENGL window and is responsible for defining the orientation and position of each point cloud.

4.3 DisplayFrameTransformer

The DISPLAYFRAMETRANSFORMER is responsible for generating transformations for the sources displayed within the OPENGL window, a UML diagram for the class can be seen in figure 9.

Each client is assigned a default transformation which can be overridden using keyboard controls.

Clients are initially arranged in a circle around the origin in the center of the space. This is done by retrieving a transformation from the TRANSFORMER for a rotation in the y axis for each client, n. Each angle of rotation, α, is calculated using the below,

$$\alpha(n) = \frac{n}{\sum clients} \cdot 360°$$

Similar to the shared variables between the MAINWINDOW and OPENGL window, client transformations are stored within a dictionary indexed by client ID.

The DISPLAYFRAMETRANSFORMER also has methods to override these initial transforms with the RotateClient() and TranslateClient() methods. When these methods are called for the first time on a point cloud, an object defining the position and rotation is populated using the default rotation. From here the presence of a client override results in applied transforms being defined by these values as opposed to the default orientation.

This leaves the current architecture of the server application as described in figure 10.

4.4 Control Scheme

The movement of objects within the OPENGL space is implemented through keyboard controls. While using the mouse would allow fine-grained and intuitive control, the number of axes for motion and rotation available to objects makes defining specific keys for each more flexible. This additionally removes the need to redefine or overload the camera controls.

The "I" key is used to cycle through displayed sources, the currently selected source is the subject of each of the movement actions. Sources are moved across the horizontal plane (x, z) of the display space using a WASD-esque layout of the UHJK keys. Objects can be rotated about the vertical (y) axis using the B and N keys. Finally the placement of an object can be reset to default using the R key, the addition of the shift modifier resets all clients.

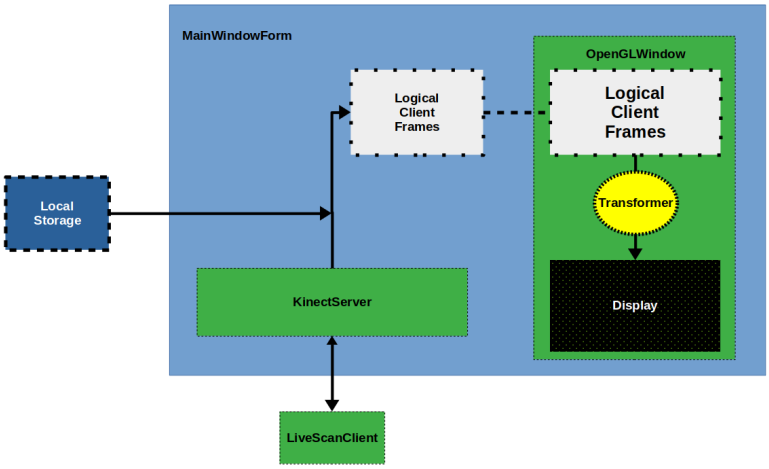

Figure 10: Current state of LIVESCAN server structure with OPENGL window-based transformer

Worth noting is that this represents arbitrary placement of sources in only two axes of position and one of rotation. This was a conscious choice as these are the most common and intuitive axes with which sources will need to be manipulated. The ability to allow movement in all axes would require only binding these actions to keys.

There is room to improve these controls as the directions of movement are in relation to the fixed axes of the display space as opposed to the view of the viewpoint camera. In practice this means that when moving objects in the display space the orientation of the space must be considered in order to identify which direction the object should be moved.

This is less intuitive than could be expected in other areas where such a control scheme is used such as video games or modelling software. In such implementations when moving objects the directions are typically taken from the camera's frame of reference. The feasibility of employing a similar control philosophy should be considered.

4.5 Challenges

The main challenge encountered throughout the project so far was initially intercepting the live frames and serializing these as XML files in local storage. With no previous experience developing in C#, the opening steps of the project involved both getting to grips with the language based on previous work in C-like languages (Java, C) and understanding the layout of the codebase.

Initial attempts to serialize the frame structures resulted in no output to the file system and the multi-threaded nature of the graphical application led to no feedback for debugging. This was fixed by removing the affine transformations representing camera poses from the frame structure for the testing process.

This would imply that something about the nature of the AFFINETRANSFORM class type is causing errors when serializing. Java requires classes implement a *serializable* interface in order to successfully save to file and further work will be required in order to determine whether the same concept is to blame in this situation. However for now the camera poses of local frames are not displayed in the OPENGL window.

5 Future Work

Following the extension of the OPENGL window, the network layer of the KINECTSERVER can now be developed with the advantage of a fully functional display method for debugging.

The aim of this aspect of the project will be to alter the the KINECTSERVER in order to allow the logical grouping of connected clients into sources for separate display.

When integrated together the server as a whole will then be able to collect discrete point clouds from different sources and coherently display them separately in the space, achieving the objectives for this project.

5.1 Network Layer Design Considerations

Some thought as to the design for the network layer has been undertaken. Although this has not yielded a final design for implementation, it has made apparent some of the conditions and constraints which must be considered.

When considering the initial steps for the project, it was thought that the network layer should be developed first. The design would involve separating much of the logic contained within the KINECTSERVER object into a new KINECTSOURCE object which would represent a group of clients acting as a single source. It would function as a group of KINECTSOCKETs that could be individually polled for new frames using the same interface currently being used by the KINECTSERVER. The KINECTSERVER object itself would be reduced to simply managing these KINECTSOURCEs.

An advantage of this approach would be that it provide a suitable location to store additional information which should exist per source such as the calibration data and settings.

However it would also have represented a significant architecture change in the entire server application and without a functioning display method it would have been challenging to debug. This was the motivation for initially working on the display method.

Coming back to the network design following this work, a different design has been considered. A separate body of work currently being undertaken is investigating the network behaviour of the suite with a focus on unblocking the network sockets to aid in parallel operation.

In order to ease integration with developments in this work a less disruptive design was proposed.

5.1.1 Socket Handshake

The aim is to implement a method by which clients are grouped into sources that also allows them to identify themselves consistently when communicating over multiple sockets. Multiple sockets can be used by clients in order to make simultaneous connections to the server and increase bandwidth. However when doing so it is important to be able to identify which sockets represent which client when some may be at the IP address.

A method for doing so would involve a handshake process when new clients connect to the KINECTSERVER. The proposed handshake would be initiated by the client when connecting to the server, at this point they include which source they should be grouped with using an integer ID. The server groups the socket as such and, if one has not been received, responds with a random identifier string that should be used across all sockets to identify the client. Should the newly connected socket be for a client that is already connected then the client will respond with it's existing identifier to inform the server that this ID has been ignored. In doing so the client now has a method of identifying itself agnostic of socket, and the server has a way of identifying the source which each socket is representing.

5.2 Deliverables and Additional Goals

At this point in the project it is worth considering the viability of the final deliverables with relation to the time remaining. Based on the work completed so far the original objectives of multi-source holoportation remain viable with a round of complete testing undertaken.

This testing suite is yet to be defined but will comprise performance evaluation for both the network and display aspects of the software.

Should the original specification be delivered and evaluated with time remaining, additional goals and investigations should be examined. Initially, aspects already completed should be investigated for further refinement, namely the control scheme as mentioned above.

When considering the design principle of network and presentation separation in combination with the relevance of the technology to the spaces of AR and VR, an interesting analysis could be made into the applicability of multi-source network developments to additional display methods. Mobile AR and HOLOLENS display for LIVESCAN have both been demonstrated and either could prove interesting when considered in a multi-source context.

5.3 Final Report Structure

The final report for the project will, to an extent, be based off of this piece in terms of structure and content. The literature review will be expanded to include an investigation of simultaneous video streaming in order to contextualise the coming work on the network layer of the application suite.

A results section will describe the quantitative and qualitative results of the final product, these will then be evaluated and discussed in the discussion section. At this point a discussion can also be had about further developments that can be made, both to the LIVESCAN codebase and the multi-source version presented by this project. The structure will be as follows,

- Title Page
- Abstract
- Table of Contents
    - Including lists of figures, code listings
- Acknowledgements
- Introduction
- Literature Review
- Methodology and Developments
- Results
- Discussion
    - A description of possible further developments
- Conclusion
- References
- Appendices

6 Summary

Within this piece the process of extending the LIVESCAN3D software to include multi-source holoportation has been introduced. The use of such a system has many applications from those inherited from traditional 2D video such as conference calls to new utilisations that are wholly unique to the environment.

The literature review contextualises the LIVESCAN suite and the wider spaces of AR, VR, 3D video and multi-source holoportation itself. Previous examples of holoportation are described and their aims of achieving telepresence are discussed.

The current state of the project is laid out showing good progress through the required areas of development. Of these areas of concern, the display element has been extended in order to allow the rendering of multiple environments simultaneously with a dynamic sub-system of geometric transformations. The transformations are responsive to user input allowing arbitrary placement and orientation of individual sources within the display space. While this control interface allows free movement in the most naturally traversed axes it could use some additional tuning to make it feel more intuitive.

The next steps for the project leading up to its completion are presented, the initial and current plans for the remaining work is described and additional stretch goals are defined for any additional time. How the work will be presented in a final report is also described.

7 Conclusions

Holoportation and multi-source configurations thereof are important technologies for cross reality experiences with broad appeal to many applications. The use of consumer hardware, specifically the KINECT, have accelerated the space.

At roughly halfway through the time allowed for this project the native display has successfully been extended to meet the deliverable specification. This has resulted in the OPENGL window being capable of simultaneously rendering multiple sources with arbitrary placement and orientation within the display space.

From this point the network layer of the suite will be developed to also match the specification, allowing connected clients to be grouped into sources for polling and processing.

Following the development of the two, testing methodologies will be defined and carried out to gather quantitative results for the final product. A final report on the results will be available in May 2020.

References

[1] M. Kowalski, J. Naruniec, and M. Daniluk, "Livescan3d: A fast and inexpensive 3d data acquisition system for multiple kinect v2 sensors", in *2015 International Conference on 3D Vision*, Oct. 2015, pp. 318–325. DOI: 10.1109/3DV.2015.43.

[2] B. Jones, R. Sodhi, M. Murdock, R. Mehra, H. Benko, A. Wilson, E. Ofek, B. MacIntyre, N. Raghuvanshi, and L. Shapira, "Roomalive: Magical experiences enabled by scalable, adaptive projector-camera units", in *UIST '14 Proceedings of the 27th annual ACM symposium on User interface software and technology*, ACM, Oct. 2014, pp. 637–644. [Online]. Available: https://www.microsoft.com/en-us/research/publication/roomalive-magical-experiences-enabled-by-scalable-adaptive-projector-camera-units/.

[3] X. Li, W. Yi, H.-L. Chi, X. Wang, and A. Chan, "A critical review of virtual and augmented reality (vr/ar) applications in construction safety", eng, *Automation in Construction*, vol. 86, 2018-02-01, ISSN: 0926-5805. [Online]. Available: http://search.proquest.com/docview/2012059651.

[4] D. Lindlbauer and A. D. Wilson, "Remixed reality: Manipulating space and time in augmented reality", in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, ser. CHI '18, Montreal QC, Canada: Association for Computing Machinery, 2018, ISBN: 9781450356206. DOI: 10.1145/3173574.3173703. [Online]. Available: https://doi.org/10.1145/3173574.3173703.

[5] Z. Zhang, "Microsoft kinect sensor and its effect", eng, *IEEE MultiMedia*, vol. 19, no. 2, pp. 4, 10, 2012-02, ISSN: 1070-986X.

[6] J. Jiao, L. Yuan, W. Tang, Z. Deng, and Q. Wu, "A post-rectification approach of depth images of kinect v2 for 3d reconstruction of indoor scenes", *ISPRS International Journal of Geo-Information*, vol. 6, p. 349, Nov. 2017. DOI: 10.3390/ijgi6110349.

[7] E. Lachat, H. Macher, T. Landes, and P. Grussenmeyer, "First experiences with kinect v2 sensor for close range 3d modelling", eng, 5, vol. XL-5/W4, Gottingen: Copernicus GmbH, 2015, pp. 93, 100. [Online]. Available: http://search.proquest.com/docview/1756968652.

[8] O. Wasenmüller and D. Stricker, "Comparison of kinect v1 and v2 depth images in terms of accuracy and precision", Nov. 2016. DOI: 10.1007/978-3-319-54427-4_3.

[9] S. Nissimov, J. Goldberger, and V. Alchanatis, "Obstacle detection in a greenhouse environment using the kinect sensor", eng, *Computers and Electronics in Agriculture*, vol. 113, no. C, pp. 104, 115, 2015-04, ISSN: 0168-1699.

[10] S. Orts, C. Rhemann, S. Fanello, D. Kim, A. Kowdle, W. Chang, Y. Degtyarev, P. Davidson, S. Khamis, M. Dou, V. Tankovich, C. Loop, Q. Cai, P. Chou, S. Mennicken, J. Valentin, P. Kohli, V. Pradeep, S. Wang, and S. Izadi, "Holoportation: Virtual 3d teleportation in real-time", Microsoft Research, Oct. 2016. DOI: 10.1145/2984511.2984517.

[11] H. Fuchs, A. State, and J. Bazin, "Immersive 3d telepresence", *Computer*, vol. 47, no. 7, pp. 46–52, Jul. 2014, ISSN: 1558-0814. DOI: 10.1109/MC.2014.185.

[12] E. Ackerman and E. Guizzo. (Feb. 2016). Marvin minsky (1927-2016) and telepresence, International Society for Presence Research, [Online]. Available: https://ispr.info/2016/02/01/marvin-minsky-1927-2016-and-telepresence.

[13] W. Buxton, "Telepresence: Integrating shared task and person spaces", in *Proceedings of Graphics Interface '92*, ser. GI 1992, Vancouver, British Columbia, Canada: Canadian Information Processing Society, 1992, pp. 123–129, ISBN: 0-9695338-1-0. DOI: 10.20380/GI1992.15. [Online]. Available: https://www.billbuxton.com/TelepShrdSpce.pdf.

[14] A. Fender and J. Müller, "Velt: A framework for multi rgb-d camera systems", Nov. 2018, pp. 73–83. DOI: 10.1145/3279778.3279794.

[15] S. Beck, A. Kunert, A. Kulik, and B. Froehlich, "Immersive group-to-group telepresence", *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 4, pp. 616–625, Apr. 2013, ISSN: 2160-9306. DOI: 10.1109/TVCG.2013.33.

[16] R. Stoakley, M. Conway, and Y. Pausch, "Virtual reality on a wim: Interactive worlds in miniature", Feb. 1970. DOI: 10.1145/223904.223938.

[17] P. J. Besl and N. D. McKay, "A method for registration of 3-d shapes", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, Feb. 1992, ISSN: 1939-3539. DOI: 10.1109/34.121791.

A Existing Data Structures

Listing 1: Cartesian coordinate in 2 dimensions

```
public struct Point2f
{
    public float X;
    public float Y;
}
```

Listing 2: Cartesian coordinate in 3 dimensions

```
public struct Point3f
{
    public float X;
    public float Y;
    public float Z;
}
```

Listing 3: Affine transformation matrix with translation

```
[Serializable]
public class AffineTransform
{
    public float[,] R = new float[3, 3];
    public float[] t = new float[3];

    public AffineTransform()
    {
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                if (i == j)
                    R[i, j] = 1;
                else
                    R[i, j] = 0;
            }
            t[i] = 0;
        }
    }
}
```

B New Data Structures

B.1 Frame

Listing 4: Point cloud with Client ID

```
public struct Frame
{
    public List<Single> Vertices;
    public List<byte> RGB;
    public List<Body> Bodies;
    public int ClientID;
    public List<AffineTransform> CameraPoses;

    public Frame(List<Single> vertsin,
```

```
                    List<byte> rgbin,
                    List<Body> bodiesin,
                    int clientID,
                    List<AffineTransform> cameraPosesin)
    {
        Vertices = vertsin;
        RGB = rgbin;
        Bodies = bodiesin;
        ClientID = clientID;
        CameraPoses = cameraPosesin;
    }
}
```