

REAL-TIME BODY TRACKING AND PROJECTION MAPPING IN THE  
INTERACTIVE ARTS

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Sydney Baroya

December 2020

© 2020  
Sydney Baroya  
ALL RIGHTS RESERVED

## COMMITTEE MEMBERSHIP

TITLE: Real-time Body Tracking and Projection  
Mapping in the Interactive Arts

AUTHOR: Sydney Baroya

DATE SUBMITTED: December 2020

COMMITTEE CHAIR: Christian Eckhardt, Ph.D.  
Professor of Computer Science

COMMITTEE MEMBER: Franz Kurfess, Ph.D.  
Professor of Computer Science

COMMITTEE MEMBER: Jonathan Ventura, Ph.D.  
Professor of Computer Science

## ABSTRACT

### Real-time Body Tracking and Projection Mapping in the Interactive Arts

Sydney Baroya

Projection mapping, a subtopic of augmented reality, displays computer-generated light visualizations from projectors onto the real environment. A challenge for projection mapping in performing interactive arts is dynamic body movements. Accuracy and speed are key components for an immersive application of body projection mapping and dependent on scanning and processing time.

This thesis presents a novel technique to achieve real-time body projection mapping utilizing a state of the art body tracking device, Microsoft's Azure Kinect DK, by using an array of trackers for error minimization and movement prediction. The device's Sensor and Bodytracking SDKs allow multiple device synchronization. We combine our tracking results from this feature with motion prediction to provide an accurate approximation for body joint tracking. Using the new joint approximations and the depth information from the Kinect, we create a silhouette and map textures and animations to it before projecting it back onto the user. Our implementation of gesture detection provides interaction between the user and the projected images.

Our results decreased the lag time created from the devices, code, and projector to create a realistic real-time body projection mapping. Our end goal was to display it in an art show. This thesis was presented at Burning Man 2019 and Delfines de San Carlos 2020 as interactive art installations.



## ACKNOWLEDGMENTS

Thanks to:

- Shanti Baroya, my grandmother, who I dedicate this thesis to and wish could have seen me finish
- My parents, brothers, and extended family, for their continual support and love
- Noah Paige, for always supporting and believing in me
- Christian Eckhardt, for introducing me to Computer Graphics and pushing me out of my comfort zone
- Irene Humer, for helping when needed and making me coffee when necessary
- Lucia Apocado, for allowing me to participate in her art show
- CPConnect Steering Committee, for awarding me 4 Azure Kinects DKs through the CPConnect Grant

# TABLE OF CONTENTS

	Page
LIST OF FIGURES . . . . .	ix
CHAPTER	
1 Introduction . . . . .	1
2 Background . . . . .	5
2.1 Projection-Based Augmented Reality . . . . .	5
2.1.1 Projection Mapping . . . . .	8
2.1.1.1 Body projection mapping . . . . .	8
2.2 User . . . . .	9
2.2.1 Passive . . . . .	9
2.2.2 Active . . . . .	10
2.3 Interaction Interfaces . . . . .	11
2.3.1 Natural User Interface . . . . .	11
2.4 Textures in Computer Graphics . . . . .	12
2.4.1 Image Masking . . . . .	14
2.4.2 Depth Map . . . . .	15
2.4.3 Point Cloud Image . . . . .	16
2.5 Related Works . . . . .	17
3 System Development . . . . .	19
3.1 Hardware and Software Requirements . . . . .	19
3.1.1 Microsoft Azure Kinect DK . . . . .	20
3.1.2 Azure Kinect Sensor SDK . . . . .	21
3.1.3 Azure Kinect Body Tracking SDK . . . . .	22

3.1.4	Multiple Kinect Synchronization . . . . .	24
3.1.5	Note on Latency . . . . .	25
3.2	Development Environment . . . . .	27
3.2.1	Visual Studio . . . . .	27
3.2.2	Unity . . . . .	28
3.3	Kinect Setup and Configuration . . . . .	29
3.4	Joint Tracking . . . . .	30
3.4.1	Body Tracking . . . . .	31
3.5	Minimizing Tracking Error . . . . .	31
3.5.1	Point Cloud Map . . . . .	32
3.5.2	2D Coordinate Rotation from Averaged Joint Angles . . . . .	33
3.6	Motion Prediction . . . . .	35
3.6.1	Body Approximation . . . . .	36
3.7	Gesture Detection . . . . .	37
4	Results . . . . .	41
4.1	Experimental Setup . . . . .	41
4.2	Analysis . . . . .	41
4.2.1	Averaging Joint Angles . . . . .	46
4.2.2	Motion Prediction . . . . .	47
4.2.3	Body Approximation . . . . .	48
4.3	Proof of Concept . . . . .	49
4.3.1	Burning Man 2019 . . . . .	49
4.3.2	Delfines de San Carlos: Un Proyecto de Esperanza 2020 . . . . .	51
5	Future Work . . . . .	56
6	Conclusion . . . . .	59

BIBLIOGRAPHY . . . . .	60
------------------------	----

## LIST OF FIGURES

Figure		Page
1.1	A snapshot from the color-changing chameleon effect designed for Burning Man 2019. The figure also includes the beginning of a fire animation that burns from the middle of the body outwards. . . . .	3
2.1	Possible image generations for augmented reality displays [5]. Each column categorizes displays as head-attached, hand-held, or spatial. The distance of each display are representative of how far away from eye's view they may require. The dashed arrow portrays the look at direction of an eye through displays of the real object. The black lines in and on the displays and object represent their surfaces. Notice that projectors can be in all categories and displayed at any distance away from eye's view onto the real object. . . . .	6
2.2	One of the quintet of singing busts in the Haunted Mansion shown (a) without and (b) with projection-based augmentation [23]. . . .	7
2.3	Passive user input/output layout [14]. The arrow direction indicates interaction flow. Virtual content affects visual, audio and/or haptic effects. The passive user can receive and experience the effects but cannot communicate with or influence them. . . . .	9
2.4	Active user input/output layout [14]. The arrow direction indicates interaction flow. Virtual content affects visual, audio and/or haptic effects. The active user can receive and experience the effects, as well as interact with and affect them. Based on the active user's interaction with the effects, they can change the virtual content. . .	10
2.5	A scene rendered both with (right) and without (left) shadow mapping [10]. While the normal mapping and color textures render the 3D objects to look realistic, shadow mapping clearly adds depth and more photorealism to the scene as opposed to not implementing it.	12
2.6	3D object in OpenGL with two textures: color texture and bump texture. The right picture depicts the object with only a color texture. The left picture shows the object with both the color texture and bump texture [28] . . . . .	13

2.7	An example of how to mask an image and an effect that is possible using the mask [3]. The masked image can be produced programmatically by the scene's depth value, by color value, and more. In this example, the masked image colors pixels black for the portion of the to mask and white otherwise, resulting in the masked image on the right with a transparent background and the bird left in the foreground. . . . .	14
2.8	Scene representations of the famous Stanford Bunny: (a) original model and (b) the depth map representation with red pixels representing a z-value closer to the camera and blue pixels having values further away from the camera [7]. . . . .	15
2.9	Scene representations of the famous Stanford Bunny: (a) point cloud with geometry information and (b) point cloud with attribute information in the form of a normal denoted with a blue line [6] . . . . .	16
3.1	The Microsoft Azure Kinect camera [20]. . . . .	20
3.2	This figure shows an example depth map (left) and a corresponding clean IR image (right) from the Azure Kinect. After casting modulated illumination in the near-IR (NIR) spectrum onto the scene, the Azure Kinect records an indirect measurement of the time it takes the light to travel from the camera to the scene and back [30]. The pixel colors in the image are mapped to the scene's depth values: a pixel closer to the camera (containing a small depth value) is mapped to blue and the pixel furthest away from the camera (containing a large depth value) is mapped to red. . . . .	22
3.3	An illustration of the array of joint locations and connections relative to the human body that the Azure Kinect creates. This skeleton includes a hierarchy made up of 32 nodes (joints) and 31 connections (bone) linking the parent joint with a child joint [36]. . . . .	23
3.4	The daisy chain device configuration of multiple Azure Kinect synchronizations. There is a maximum of eight subordinate devices synchronized to one master device [34]. This image also shows where to plug in the audio cables to distinguish between a master devices and subordinate devices. . . . .	24
3.5	System diagram of how our system communicates between each hardware and the user. The Kinects read data from a user, as a gesture and/or their joint information. The computer reads the data from the Kinects and sends it to the projector. The projector, in turn, projects the computer's images onto the user. . . . .	26

3.6	An outline result from our body projection mapping algorithm after being processed through a compute shader. The points in the point cloud map are rendered with scaled transformation matrix dependent on their distance compared to the closest and farthest body pixels in the map. The point with the smallest depth value are scaled by 0.3 and the point with the greatest depth value are scaled by 0.05. All points in between are mapped to values between 0.05 and 0.3 based on their distance to the the points with smallest and greatest depth values. Points on the outline of the body are further scaled by a factor of 0.6. . . . .	38
3.7	Body outline results from approximating body segment dimensions based on the predicted joint positions. . . . .	39
3.8	State diagram of how our system handles gesture detection. It tracks the movement of the user then switches states if a gesture is detected, otherwise it keeps trackign movement. If waving left or waving right is detected, then the skin textures effect (i.e. chameleon, see 4.3 for more) are changed. If arms up is detected, then the animation effects (i.e. dolphin swimming around, see 4.3 for more) are changed. Then, the projection mapping is updated and the cycle continues. . . . .	40
4.1	Dual axis line graph displaying the results of the right elbow joint's motion prediction and averaged angle. The x-axis is time measured in seconds with a tracking start time of about seven seconds. The left y-axis represents the rate of change over time of the length between the predicted position and the position computed from a 2D rotation of the right elbow joint, measured in meters. The right axis represents the change of angle, in degrees, over time at the right elbow joint, which is derived using the right shoulder, elbow, and wrist joints. The dashed lines are the angle averages. The solid lines are the position variance. Red colored lines are data from a three device arrangement. Green colored lines are data from a two device arrangement. Blue colored lines are data from a one device arrangement. . . . .	42

4.2	Dual axis line graph displaying the results of the right knee joint's motion prediction and averaged angle. The x-axis is time measured in seconds with a tracking start time of about seven seconds. The left y-axis represents the rate of change over time of the length between the predicted position and the position computed from a 2D rotation of the right knee joint, measured in meters. The right axis represents the change of angle, in degrees, over time at the right knee joint, which is derived using the right hip, knee, and ankle joints. The dashed lines are the angle averages. The solid lines are the position variance. Red colored lines are data from a three device arrangement. Green colored lines are data from a two device arrangement. Blue colored lines are data from a one device arrangement. . . . .	43
4.3	Dual axis line graph displaying the results of the right hip joint's motion prediction and averaged angle. The x-axis is time measured in seconds with a tracking start time of about seven seconds. The left y-axis represents the rate of change over time of the length between the predicted position and the position computed from a 2D rotation of the right hip joint, measured in meters. The right axis represents the change of angle, in degrees, over time at the right hip joint, which is derived using the chest, right hip, and right knee joints. The dashed lines are the angle averages. The solid lines are the position variance. Red colored lines are data from a three device arrangement. Green colored lines are data from a two device arrangement. Blue colored lines are data from a one device arrangement. . . . .	44
4.4	Body projection mapping outline depicting a snapshot from a frame of lifting the right leg up. The red colored outline is data from a three device arrangement. The green colored outline is data from a two device arrangement. The blue colored outline is data from a one device arrangement. The white pixels designates an overlap between all outlines. . . . .	45
4.5	Body projection mapping outline depicting a snapshot from a frame of walking to the right. The red colored outline is data from a three device arrangement. The green colored outline is data from a two device arrangement. The blue colored outline is data from a one device arrangement. The white pixels designates an overlap between all outlines. . . . .	46



4.6	Body projection mapping outline depicting a snapshot from a frame of landing after a jump. The red colored outline is data from a three device arrangement. The green colored outline is data from a two device arrangement. The blue colored outline is data from a one device arrangement. The white pixels designates an overlap between all outlines. . . . .	47
4.7	These figures portray the butterfly morphing effect we implemented. Over a span of a few seconds, each butterfly from (a) morph into the eyeballs in (b) at different times. . . . .	49
4.8	(a): A full view of the entire eyeball structure at Burning Man 2019. The structure is 7 feet tall and the eyeball is 3 feet wide. (b): A closer view of the structure's iris and 'pupil' hole where the Kinect and projector are looking out of. . . . .	50
4.9	The white dolphin structure that Apodaca's team constructed for our installation at the 2020 Delfines de San Carlos art show situated on El Mirador de San Carlos. A low polygon dolphin fixed on top of a wave-like base made up of recycled cycling wheels. Pictured from left to right: Noah Paige, Sydney Baroya, Irene Humer, Christian Eckhardt, Lucia Apodaca, Sara Valle (Mayor of Guaymas), and Enrique Gamez (comisario of San Carlos). . . . .	53
4.10	The psychedelic effect developed for the projection mapping on to the dolphin structure. This effect was triggered when an active user waved their arms left or right. Pictured is the team for our interactive installation at the Delfines art show from left to right: Noah Paige, Sydney Baroya, Christian Eckhardt, and Irene Humer. . . . .	54
4.11	The laptop/projector/Kinect setup for the 'Delfines' art show. The dolphin structure that we are mapping a projection onto is mounted on a tall base so we required a tall structure for the projector to sit on along with a platform to keep the laptop and Kinect at body level.	55

## Chapter 1

### INTRODUCTION

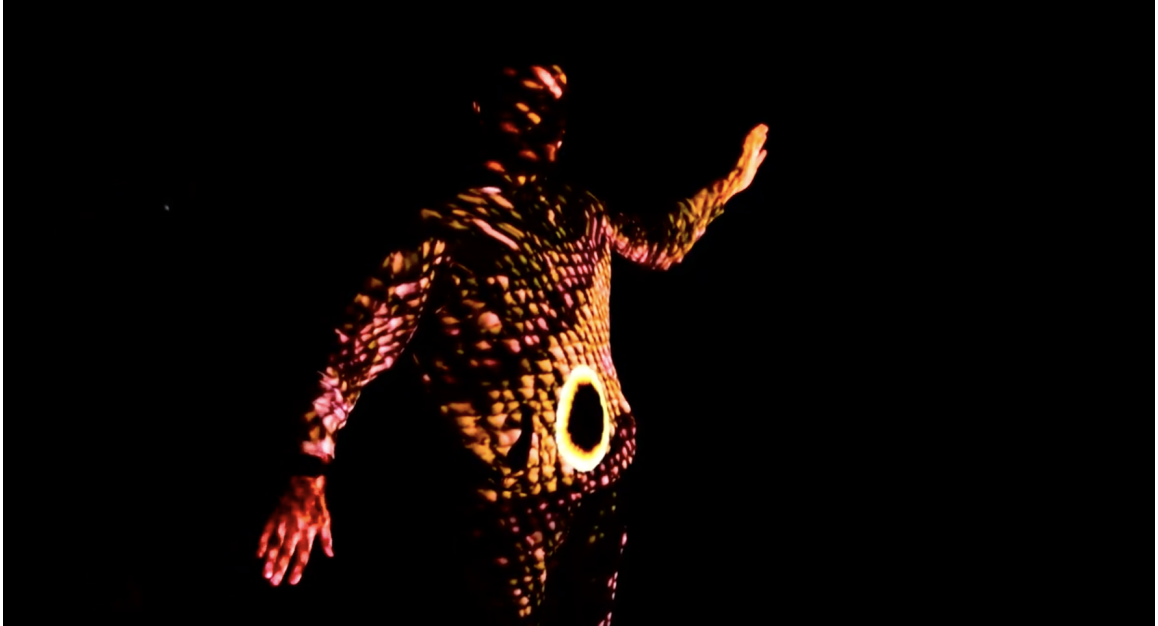
Computer graphics, a subtopic of computer science that explores digitally synthesizing and manipulating virtual content, has traditionally attempted to imitate the real world in a virtual environment. Methods like global illumination add to the realism of a virtual scene. As computer graphics have evolved, the reverse has been explored – virtual environments in the real world. From movies like *Tron* and games like *Pokémon GO*, extended reality(XR) has captured the wonder of many. XR is an umbrella term for immersive technologies including virtual reality(VR), augmented reality(AR), and mixed reality(MR). Instead of viewing virtual data through a 2D screen, we can now use our 3D world to portray the same data. This technology gives users an immersive experience by merging the physical and digital worlds. The interaction between imaginary, or digitally created, objects and physical objects is possible due to computer graphics and computer vision.

Virtual reality integrates users and virtual objects into the same virtual world. This process currently requires a headset that entirely covers the user’s sense of sight in order to fully submerge them into this new reality, which cuts off their perception of the real world. Users also need special handheld controllers in order to interact with the virtual objects in the world. A drawback with this technology is a common side effect of nausea. Virtual reality games and applications enable users to be in a fast-moving environment, like a roller coaster or driving a car. At the same time, their physical body is in a static state, causing motion sickness. This discomfort may make users uneasy and encourage them to stop or spend less time using virtual reality

On the other hand, augmented reality puts realistic overlays of virtual information on real world objects [8], most popularly through the use of mobile phones and see-through devices like a HoloLens. Users still have awareness of their surroundings and no consequences of a sickness. Instead of the user encountering an unknown virtual world, AR displays virtual information on billboards over physical objects or by overlaying it in a way that encases the real-world objects that are familiar to them. This can enhance the information from the surrounding world to provide a vivid interface to the viewers. Two challenges for augmented reality applications are displaying onto dynamic objects and full immersion.

With the help from depth sensors and computer vision, AR has the ability to present information onto a dynamic space. These technologies are able to follow the positions of physical objects in real time. In performing and interactive arts, there are attempts to use this technology to serve as a platform for artists to enhance their performance and to provide the audience with an elevated, visual experience [24]. Because users are limited to the reach and dimensions their AR device, they are required to keep the device within their eyesight. Projection-based augmented reality is able to relieve users of this hassle, using a projector to overlay unique visuals directly onto objects instead of through a handheld device. Projection mapping is a well-known projection-based AR technology for displaying computer generated light visualizations from projectors onto the real environment [23]. Real-time body mapping projection takes this idea and uses a body tracking device and software(i.e. depth sensor, motion capture suit) to project visualizations onto bodies as they move. However, a drawback is the communication lag. A communication lag occurs due to the body tracking information traveling from the body tracking device to the computer and then out through the projector. Therefore, current implementations of this subtopic of AR either follow movements from one spot or do not project it directly onto the body [24]. Another big obstacle for AR is user experience, especially in the interactive arts. If interact-

ing with an AR system is too complicated or looks unrealistic, then its immersion decreases and users become disinterested.



**Figure 1.1:** A snapshot from the color-changing chameleon effect designed for Burning Man 2019. The figure also includes the beginning of a fire animation that burns from the middle of the body outwards.

Our solution described in this work attempts to improve body projection mapping using motion prediction and tracking error minimization. We chose the state-of-the-art body tracking device Microsoft Azure Kinect Developer Kit as our hardware platform. A unique feature of this kit is its ability to synchronize multiple Kinect devices, which we take advantage of for minimizing tracking errors. We devised a method that uses multiple Kinects positioned in different positions to average similar joint angles and that performs a velocity prediction calculation to more accurately provide joint coordinates. We also explored a technique that utilizes the depth map from Kinect's depth camera to build a point cloud map. The results of our work was presented as art installations in Burning Man 2019 and Delfines de San Carlos:

Un Proyecto de Esperanza, see Figure 1.1 for a body projection mapping effect from Burning Man 2019.

## Chapter 2

### BACKGROUND

This chapter contains background information that we deem necessary to understand the mechanics of our thesis. Because the core of this thesis is projection-based augmented reality, we explore it and its subtopic projection mapping, in order to understand body projection mapping. For an interactive AR system, it is key to keep the user in mind and both types of users are considered in our project. Textures add photorealistic value to computer graphics applications. We discuss them generally and the ones we utilize are discussed in depth. Lastly, we describe related projects and their methods.

#### **2.1 Projection-Based Augmented Reality**

Consumer-based augmented reality is distributed through handheld or eyeworn displays. These applications include face filters apps, mobile apps, and AR glasses. However, some novel approaches have taken AR beyond these displays, using large spatially-aligned optical devices (i.e. video projectors). Therefore, this is referred to as spatial augmented reality (SAR) or projection-based augmented reality. Projection-based augmented reality is defined as the use of projection technology to enhance 3D objects and spaces in the physical world by projecting images and animations onto their visible surfaces [5]. This effect can reproduce or synthesize different surface attributes. Ramesh Raskar calls this mode of visualizing 3D computer graphics as shader lamps [29]. Computer graphics shaders give 3D virtual objects their surface

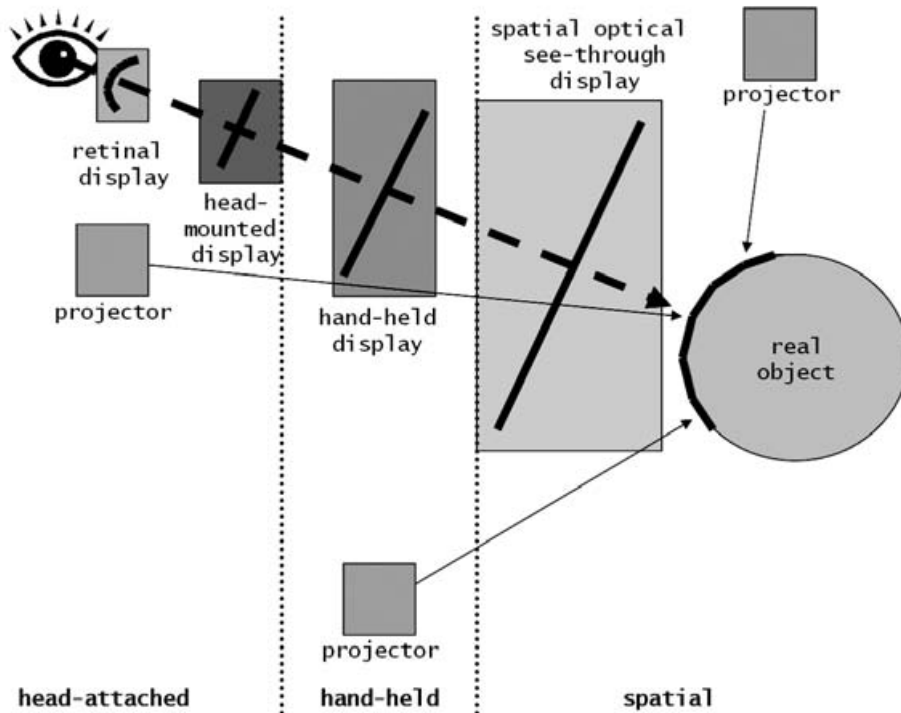
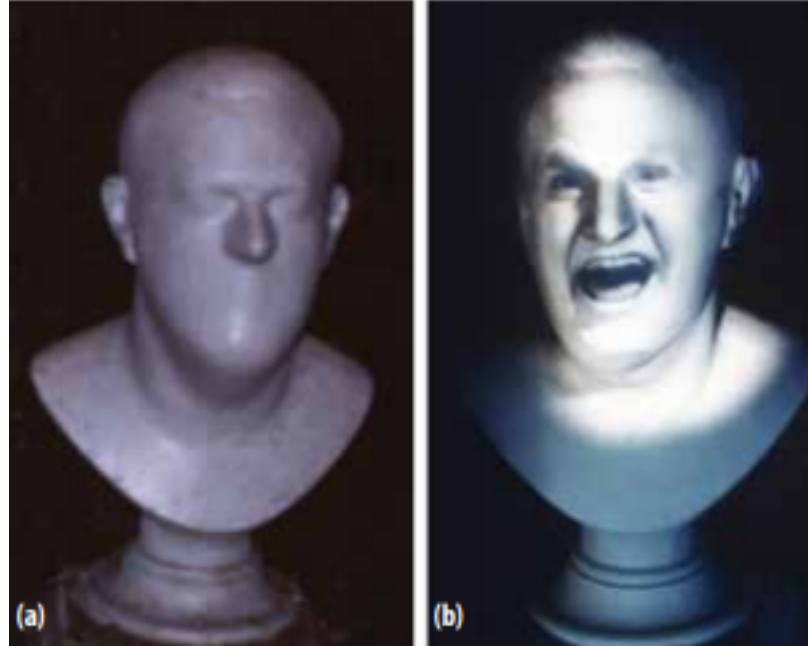


Figure 2.1: Possible image generations for augmented reality displays [5]. Each column categorizes displays as head-attached, hand-held, or spatial. The distance of each display are representative of how far away from eye's view they may require. The dashed arrow portrays the look at direction of an eye through displays of the real object. The black lines in and on the displays and object represent their surfaces. Notice that projectors can be in all categories and displayed at any distance away from eye's view onto the real object.

attributes and a projector lights up 3D physical objects like a lamp – hence the name shader lamps.

The first well-known instance of projection on arbitrarily complex surfaces dates back to Disneyland's 1969 opening of Haunted Mansion [4]. The Haunted Mansion ride featured a disembodied head inside a crystal ball, Madame Leota, and a quintet of singing busts, the 'Grim Grinning Ghosts' [23]. Optical illusions of 'living' statues were created by filming actors singing and talking and then projecting the films onto busts of their faces, see Figure 2.2.



**Figure 2.2:** One of the quintet of singing busts in the Haunted Mansion shown (a) without and (b) with projection-based augmentation [23].

The main advantage of SAR is that it can create beautiful dynamic environments and bring life to a static environment that would be difficult to achieve with traditional lighting. In addition, it creates a shared experience for an audience, advantageous for a performance or an interactive art installation with people passing by. In contrast, device-based AR (i.e. head-mounted displays or handheld devices) does not scale well since they are typically single-user. Because there is no need to constantly keep a handheld or eyeworn display in the eye's view, SAR displays overcome the technological and ergonomic limitations of traditional AR systems. Items as obvious as handheld mobile devices often conflict with the design and disenchant the illusion artists are trying to create.



### **2.1.1 Projection Mapping**

A subsection of SAR or projection-based AR is projection mapping. Projection mapping is a projection technique that creates “an optical illusion by analyzing three-dimensional objects, projecting images, and then precisely aligning them” [16]. The art field has adopted this technique but most implementations project onto fixed objects with a manual alignment between object and projection. For one of our art installations, we use this technique to project onto a dolphin sculpture, see Chapter 4.3 at page 53. Automatic alignment is used when projecting images onto dynamic objects. This can be 3D objects that are being moved around or the bodies of the performers/users. The other art installation we implemented uses this method to project onto the bodies of individuals interacting with our system, see Chapter 4.3 at pages 49 and 3.

#### **2.1.1.1 Body projection mapping**

Body projection mapping only augments a performer or user in range of the sensor being used. This effect can be used to decorate a body with different surface attributes or to illustrate a body as a silhouette within the scene. Recently, various body projection mapping systems that involve image or depth sensors for rendering in real-time interactions have been developed. In performing arts, there are attempts to use this technology to serve as a platform for artists to enhance their performance and to provide the audience with an elevated, visual experience.

Rendering in real-time requires huge computation, causing latency. As latency increases, the movement of the projected image becomes slower in comparison to the movement of the body. Consequently, “it generates a visual error, which reduces the immersion of audiences”[16]. To combat this error, the system rendering real-time

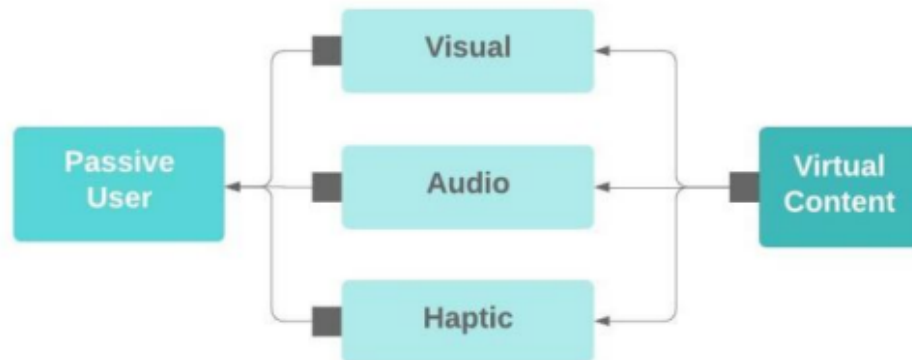
projected images needs to apply some type of motion prediction. We will go further into how we implemented motion prediction in Chapter 3.

Another thing to consider when it comes to body projection mapping is a flexible shape [16]. The methods mentioned previously have been applied to the surface of solid 3D objects. Flexible shapes, such as clothing, have silhouettes that are difficult to predict because their shapes change frequently. When a sensor is tracking a flexible shape, it can inaccurately predict there to be an edge where the shape is folding or moving around, causing the projected images to have a visual error.

## 2.2 User

An important component of interaction in AR systems is the user. There are two types of users in AR: the passive user and the active user.

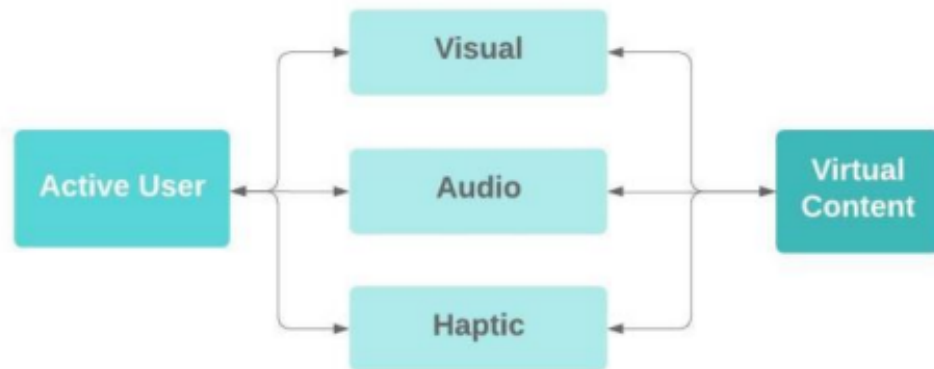
### 2.2.1 Passive



**Figure 2.3: Passive user input/output layout [14].** The arrow direction indicates interaction flow. Virtual content affects visual, audio and/or haptic effects. The passive user can receive and experience the effects but cannot communicate with or influence them.

A passive user refers to a user who consumes virtual content, receives visual, audio, and/or haptic feedback, and does not interact with the virtual content [14]. There can be many passive users as the interaction space for an AR system allows. For example, a system with haptic feedback gloves limits the amount of passive users to the amount of gloves available. This system can be used in a performing art show to allow a whole audience to be passive users. The advantage of using AR in a passive capacity is its ease of use – “there is no need to provide instructions on how to interact with the virtual content” [39].

### 2.2.2 Active



**Figure 2.4: Active user input/output layout [14].** The arrow direction indicates interaction flow. Virtual content affects visual, audio and/or haptic effects. The active user can receive and experience the effects, as well as interact with and affect them. Based on the active user’s interaction with the effects, they can change the virtual content.

An active user refers to a user who can interact with the virtual content in an AR system through a user interface with seamless control [14]. Like a passive user, they can receive visual, audio, and/or haptic feedback from the system but they also have the ability to control it. Usually there can only be one active user for an AR system. This is because if there were multiple active users simultaneously, especially

in a gesture detected interactive system, then the system may be overwhelmed with which user to take controls from.

## **2.3 Interaction Interfaces**

Since computers were made, it was necessary to interact with it physically or with another physical device attached to it. As technology has advanced, a wide breadth of interaction devices has been invented that do not require an attached physical device, such as a remote control, Bluetooth keyboards, and more. We have even gone so far as to not need anything in hand while interacting with our computers and consoles. AR systems also need an appropriate user interface to intuitively interact with the virtual content appearing in the world. Gesture recognition devices are “trying to break down technological barriers and . . . to transform technology” [13] to understand the users. However, due to the learning curve gesture technology brings, there is hesitation to use them.

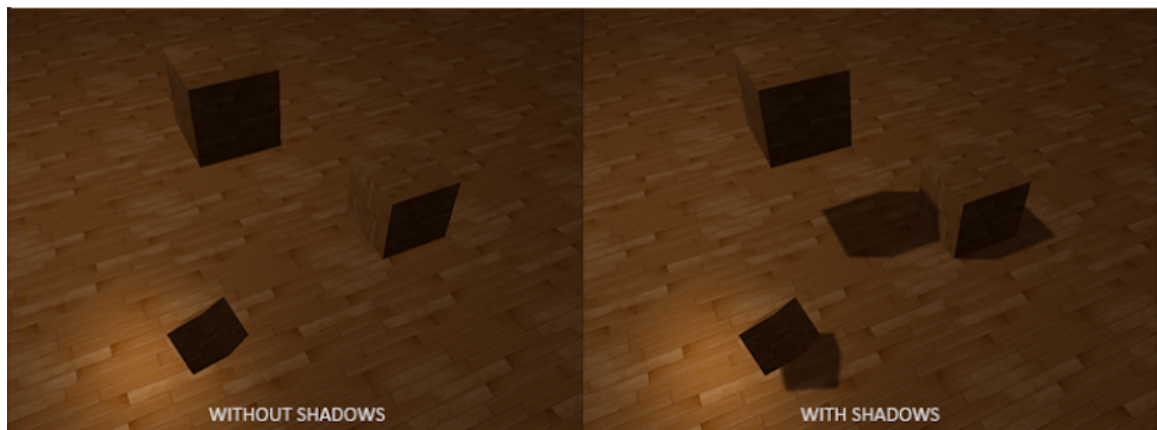
### **2.3.1 Natural User Interface**

A natural user interface uses a person’s natural movements as the input for interaction [14], requiring some type of gesture recognition. Gesture recognition is defined as an interface with computers using gestures of the human body, typically hand movements. This technology uses a camera to recognize the movements of the human body and forwards the data to a computer that uses the gestures as input controls. Using a sensor-based tracking system is an accurate method of tracking body motion, thereby providing a more accurate interaction [14]. However, this type of system restricts the user. For example, both a motion-capture suit and a sensor with a camera limits the area that motion can be tracked to the distance that these sensors are able

to communicate with a computer or view a user, respectively. Guiding systems for these systems are mainly designed in an ad hoc manner [11]. Even if isolated design characteristics exist, they concentrate on limited guidance. If gesture recognition is to advance, the gesture commands need to be self-revealing or a decided gesture language that is universal across all gesture-based devices.

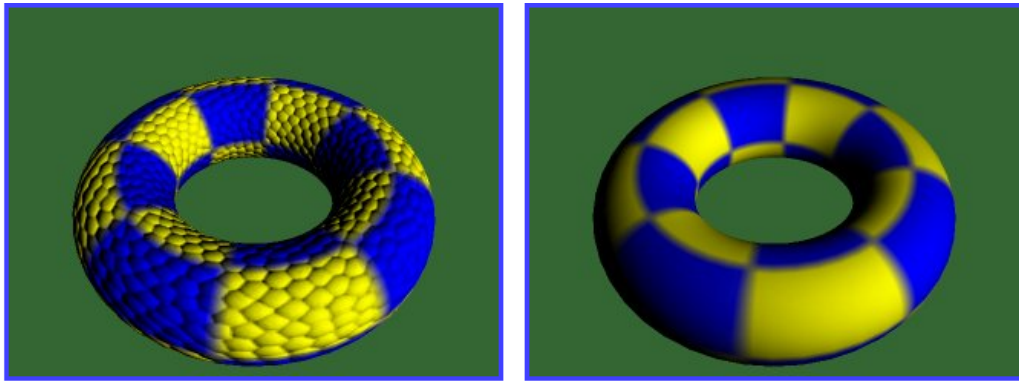
## 2.4 Textures in Computer Graphics

A key component of computer graphics is giving an object's surface different properties. Typically, there are three properties: diffuse, specular, and translucent. A translucent property allows an object to transmit and reflect light. Specular properties give an object a look that is smooth and shiny by assigning values to parts of an object where light should bounce off of it. In contrast to the other properties, light scatters with diffuse. This gives an object its color and roughness. A common way to give objects these properties is through the use of textures.



**Figure 2.5:** A scene rendered both with (right) and without (left) shadow mapping [10]. While the normal mapping and color textures render the 3D objects to look realistic, shadow mapping clearly adds depth and more photorealism to the scene as opposed to not implementing it.

Textures are able to contain any information about a pixel in the scene, not just about an object's surface properties. For example, depth map contains the depth values of each pixel away from the camera. When passed to a shader, it can be used to alter the scene based on a pixel's depth value. A shadow mapping also uses this technique except measures the distance from a light source to the objects in the scene and then using it to render the scene from the camera's point of view. Using these texturing techniques can significantly improve the photorealism of computer graphics applications, see Figure 2.5.



**Figure 2.6: 3D object in OpenGL with two textures: color texture and bump texture. The right picture depicts the object with only a color texture. The left picture shows the object with both the color texture and bump texture [28] .**

In OpenGL, “a texture can be used in two ways: it can be the source of a texture access from a Shader, or it can be used as a render target” [2]. When used as a texture access, it can contain information about color or other surface attributes stated above. An example of how an object looks with one or multiple surface attributes via textures is depicted in Figure 2.6. A texture used as a render target means that a shader will write information to a texture so that it can be accessed by the CPU and may be used as a texture for a different shader.

### 2.4.1 Image Masking

Image masking is a process that masks out certain pixels from an image [17], separating a certain part of the image from its background. Using this, a programmer may intend to place the part of the image over another background, replace the part of an image with something else, add a post processing effect on only the part of the image that is masked, and more. Figure 2.7 gives an example of masking an image and one effect that a programmer could use with the image mask. It is possible to pre-render

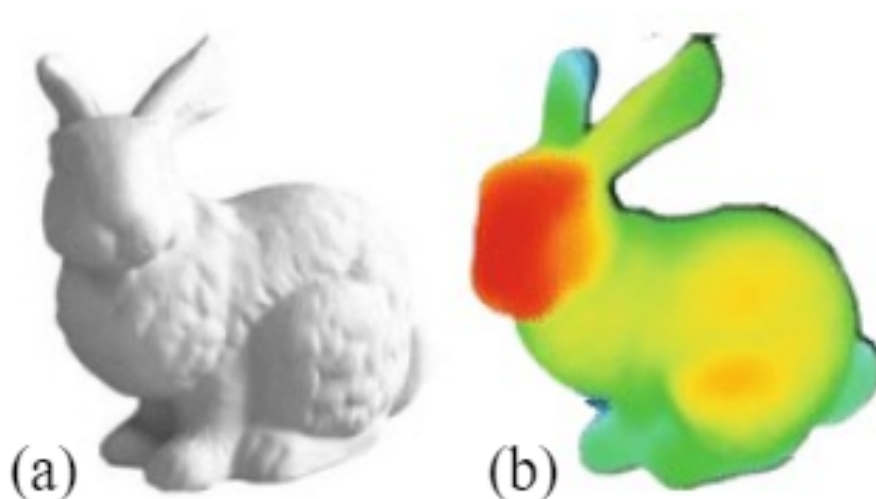


**Figure 2.7:** An example of how to mask an image and an effect that is possible using the mask [3]. The masked image can be produced programmatically by the scene’s depth value, by color value, and more. In this example, the masked image colors pixels black for the portion of the to mask and white otherwise, resulting in the masked image on the right with a transparent background and the bird left in the foreground.

the “Masked Image” in Figure 2.7. But if the bird was a 3D object and moved around the scene, then all images of the bird flying would have to be pre-rendered. Using a shader to mask the image, gives a real-time rendered mask and there is no need to continually pre-render the image when something in the scene changes.

### 2.4.2 Depth Map

Many computer graphics applications use a depth map for various visual effects like shadows, culling, and image masking. “A depth map is a set of Z-coordinate values for every pixel of the image” [31] measured away from the camera. It is analogous to the depth buffer or Z-buffer. This map can be created by calculating the distance from the camera to every point in the scene, see Figure 2.8 for a comparison of an image to its depth map.

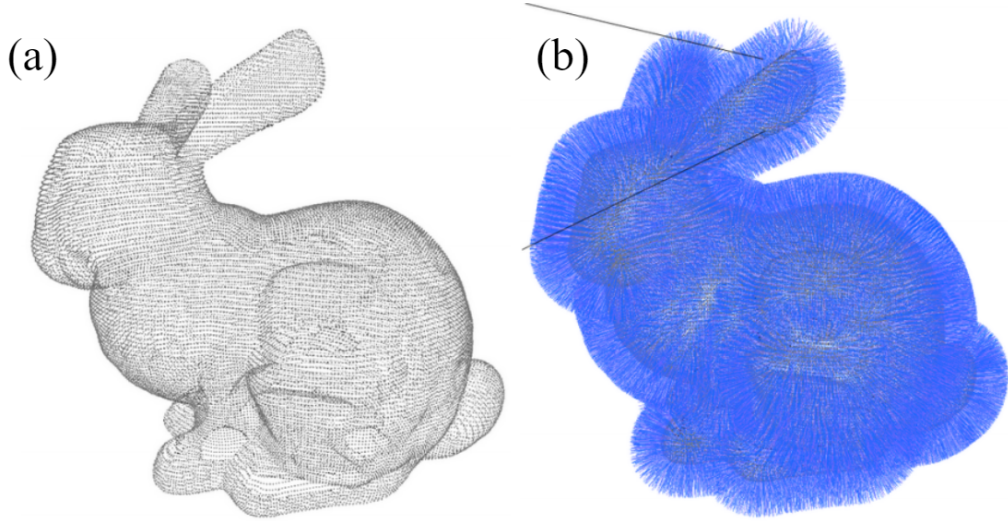


**Figure 2.8: Scene representations of the famous Stanford Bunny: (a) original model and (b) the depth map representation with red pixels representing a z-value closer to the camera and blue pixels having values further away from the camera [7].**

In order to create a depth map of tangible objects of an environment, a depth camera is required. Time-of-Flight is a principle some depth cameras, including the Microsoft Azure Kinect, implement to measure depth in the world. The principle is to cast modulated illumination in the near-infrared(NIR) spectrum onto the scene and then record an indirect measurement of travel time of the light from the camera to the scene and back [31]. This can be produced in a static or dynamic context. Within a dynamic context, a depth map is produced every frame. Latency is a challenge for



real-time depth map production due to the amount of time it takes to cast the light every frame.



**Figure 2.9: Scene representations of the famous Stanford Bunny: (a) point cloud with geometry information and (b) point cloud with attribute information in the form of a normal denoted with a blue line [6]**

### 2.4.3 Point Cloud Image

Although depth maps are valuable, they exclusively contain 2D information. However, we can reproject each pixel in the 2D depth map into 3D space, creating a point cloud. A 3D point cloud is a set of points, with each point being embedded in the 3D space and carrying both geometry and attribute information [6], see Figure 2.9. The geometry information refers to the Cartesian coordinate of the point's position relative to the camera. The attribute information may contain various information about the visual appearance of each point, such as the color and the normal vector. Similar to the depth map, point clouds can be produced in with static or dynamic content. With dynamic content, a different point cloud is considered at each frame. The term "cloud" reflects the unorganized nature of the set [27]. The number of points may

vary between frames, resulting in no point-to-point correspondence between clouds in successive frames.

## 2.5 Related Works

There are some similar approaches to real-time body projection mapping and interaction. The first method is purposed by Dubnov et al. using infra-red (IR) markers attached to the performer’s arms, legs, or both to detect gestures during practice/-training and are then fed as training input into a Hidden Markov Model (HMM) that tracks and recognizes the gestures relative to similar ones [12]. The Microsoft Kinect, since originally used for interactive Xbox games, already has functionality to recognize gestures but the added features will help reduce a system’s latency. Once the model is trained successfully, there is no need to use IR markers. During the testing/performing phase, Dubnov’s et al. method uses the depth camera from the Kinect to acquire an IR image and a depth image, derive the 3D coordinates of potential markers, and feed those coordinates into the HMM in order to predict the performer’s gestures [12].

The next method is proposed by Lee et al. where the real-time projection mapping is achieved by using a masking technique that tracks different actor’s silhouette, creates new masked images with the silhouette, composites video on top of the mask, and projects the newly generated images that aligns with the performer’s costume [16]. The Microsoft Kinect has a depth camera that is able to extract a masked image of bodies within their fields of view which is limited. For the project Lee et al. worked on, they found using an IR camera had a wider range, well suited for the large stage the performers were on [16]. So, they needed to create a real-time masking technique since the IR camera does not automatically mask the bodies in its image.

Morrison et al. proposes an interactive art installation using MAX/MSP/Jitter and a black and white digital surveillance camera with an IR filter to track real time movement of participants [25]. When participants step up to the installation, their silhouettes are projected onto a screen and an image of a young girl moves inside and across different silhouettes. Depending on the audience member's gestures, the girl will dance, skip, twirl, smile, lie down, and more. While the previous method is capable of detecting small gestures, this method uses larger movements as gestures for the system.

## Chapter 3

### SYSTEM DEVELOPMENT

Vovk states that there are three principles of AR and spatial computing that influence a fulfilling experience in AR: ergonomics of the devices being used, the design or interpretability of the environment, and the value of the content a user experiences [38]. While creating this project, we kept those three things in mind because the user determines how successful this project is. It is important to note that we capitalized Microsoft’s Azure Kinect Sample repository for the initial sensor and body tracking code organization [21, 22]. This allowed us to solely work on the novel parts of this project and remove focus from setting up the Kinect and organizing the code. In this chapter, we discuss the specifics of our implementation of real-time body projection mapping and interaction in interactive arts. The chapter starts with the hardware and software requirements in order to run the system. Then, we dive into the development environments we chose. Lastly, we explain our software implementations of minimizing both motion error and projection precision error. The sections will also include the steps we took for each feature as well as the gesture detection and texturing we applied to fit with our interactive art events.

#### **3.1 Hardware and Software Requirements**

We have not seen any documented research using the Microsoft Azure Kinect DK for body projection mapping, so we chose this hardware platform. The following sections describe the sensor’s hardware and software requirements and their benefits.

### 3.1.1 Microsoft Azure Kinect DK



**Figure 3.1: The Microsoft Azure Kinect camera [20].**

The original Kinect was launched as an Xbox accessory and never got very popular. Customers have commented that although the device is great for exercise-type games and party games, the gestures aren't as quick as short finger movements so didn't appeal to hardcore gamers or those who like complex games. Because of this, Kinect lost popularity and Microsoft decided to discontinue developing for it.

The Azure Kinect, on the other hand, was designed for developers, shown in Figure 3.1. Azure Kinect DK is a developer kit that “contains a depth sensor, spatial microphone array with a video camera, and orientation sensor as an all in-one small device with multiple modes, options, and software development kits (SDKs)” [33]. With this kit, Microsoft gives developers a platform to experiment computer vision. Each device purchased, comes with a power cord and adapter, a USB 3A cord to plug into a computer, and a cover to protect the device. The following list is the minimum host PC requirements<sup>1</sup> for the project we have worked on:

---

<sup>1</sup>There are different requirements for the Azure Kinect based on the SDKs being used. This project requires both the Sensor SDK and the Body Tracking SDK so the requirements are a combination of more stringent than if only the Sensor SDK was being used. See Microsoft's Azure Kinect system requirements [32] to view the requirements for each SDK.

- Windows 10 April 2018 (Version 1803, OS Build 171134) release (x64) or a later version<sup>2</sup>
- Seventh Gen Intel® Core™ i5 Processor (Quad Core 2.4 GHz or faster)
- 4 GB Memory
- NVIDIA GEFORCE GTX 1070 or better
- Dedicated USB3 port (for each device being used)
- Graphics driver support for OpenGL 4.4 or DirectX 11.0v [32]

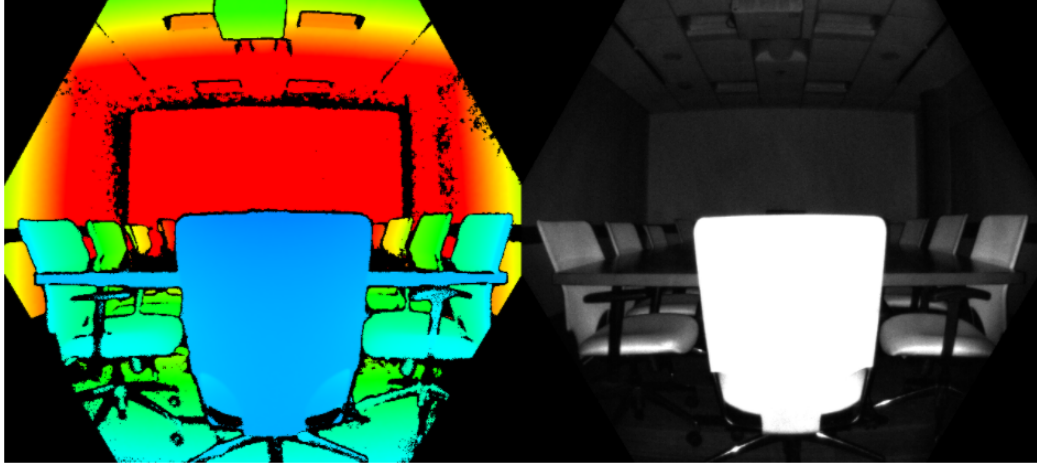
### 3.1.2 Azure Kinect Sensor SDK

The Azure Kinect Sensor SDK is required to communicate with the Azure Kinect. Microsoft developed this SDK for the Kinect specifically and it was not the focus of this project to provide a communication method for the sensors and the computer so we utilize it. This SDK opens and closes the devices; starts and stops the cameras in each device; and calibrates the device based on the developer’s specified configurations (e.g. depth mode for depth camera and color resolution for RGB camera). Once those steps are successfully run, then we can access the depth image in the Azure Kinect also using the Sensor SDK. Figure 3.2 shows an example of what the depth image and the corresponding clean IR image could look like.

The depth mode for our depth camera configuration is the NFOV 2x2 binned (SW). Even though the resolution of the image in this mode is only 320x288 and has a narrow field of view (75x65 degrees), we select it because of its ability to capture depth at a range of 0.5 - 5.46 meters. Because we choose to deploy multiple synchronous devices,

---

<sup>2</sup>Another supporting OS is a “Linux Ubuntu 18.04 (x64), with a GPU driver that uses OpenGLv4.4 or a later version” [32]. However, this project has not been tested with this OS so we’re not sure if it will support it.

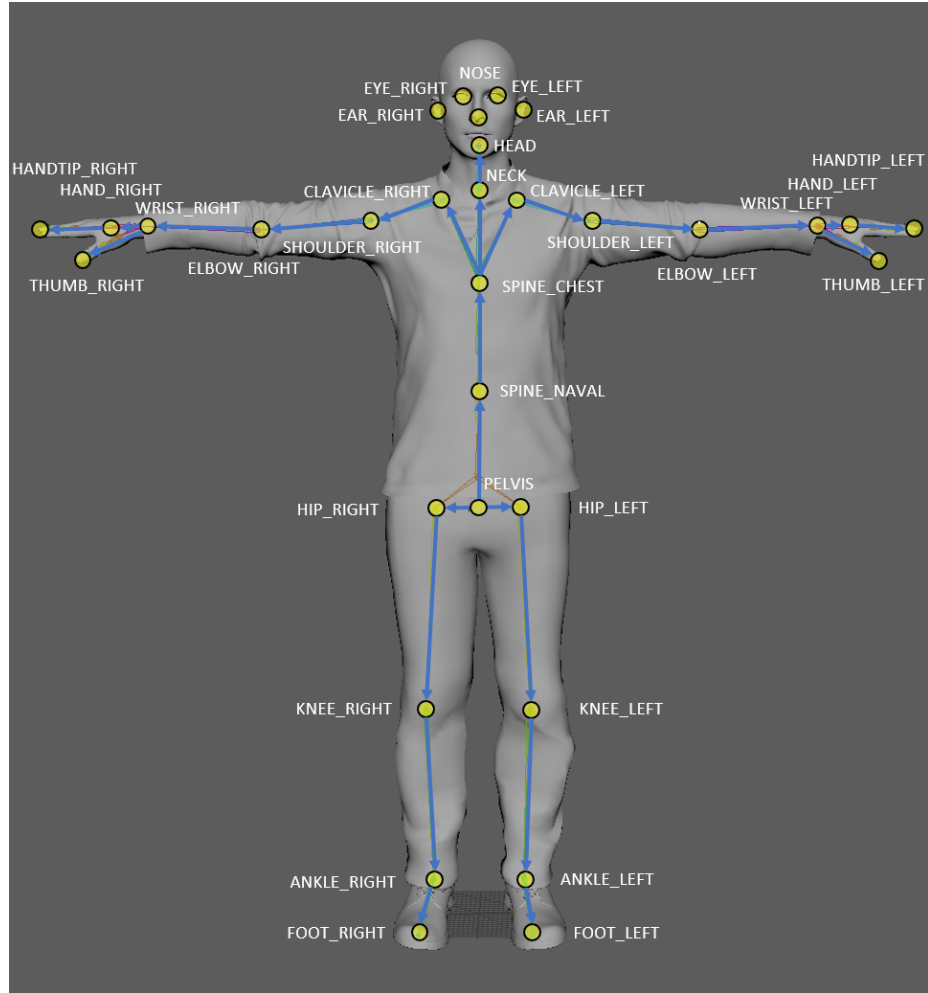


**Figure 3.2:** This figure shows an example depth map (left) and a corresponding clean IR image (right) from the Azure Kinect. After casting modulated illumination in the near-IR (NIR) spectrum onto the scene, the Azure Kinect records an indirect measurement of the time it takes the light to travel from the camera to the scene and back [30]. The pixel colors in the image are mapped to the scene’s depth values: a pixel closer to the camera (containing a small depth value) is mapped to blue and the pixel furthest away from the camera (containing a large depth value) is mapped to red.

the width of our field-of-view increases, so depth is our top priority. This depth range allows users and performers to be more immersed in our AR system and not collide with the hardware setup.

### 3.1.3 Azure Kinect Body Tracking SDK

Even though a main component of this project is tracking the bodies of the users/performers, it is not a novel part of this project. So, we use Microsoft’s Azure Kinect Body Tracking SDK in order to focus on the novel aspects. This SDK can create a tracker, use it to capture a frame with bodies, and track bodies using the image. We can track the bodies two different ways: a body index map and a body joint structure hierarchy.



**Figure 3.3:** An illustration of the array of joint locations and connections relative to the human body that the Azure Kinect creates. This skeleton includes a hierarchy made up of 32 nodes (joints) and 31 connections (bone) linking the parent joint with a child joint [36].

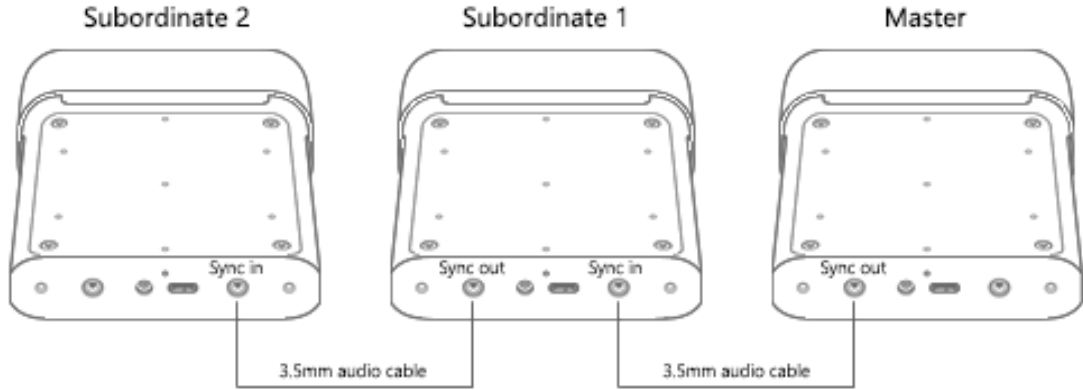
For every frame, the Kinect generates a segmentation map for each body in the depth camera capture resulting in the index map. Each pixel's value maps to either a number ID for which body the pixel belongs to (from one to the number of detected bodies) or the SDK's enum for the background [35]. Accessing the tracking for body joints is different from the index map. Instead of an image, the Kinect returns an ID, which may differ from the ID given by the index map, and a skeleton structure for each body within its frame. The skeleton structure, see Figure 3.3, contains an array of joint structures. Each joint structure holds information about its position in



camera space, orientation in normalized quaternion, and enum representing the level of confidence the sensor possesses in regards to tracking the joint.

### 3.1.4 Multiple Kinect Synchronization

Each Azure Kinect device contains synchronization ports (*Sync In* and *Sync Out*) that can be used to link multiple devices together. The synchronization ports are fitted for 3.5mm male audio cables, which are not provided with the device purchase. There are two device configurations that Microsoft recommends: daisy-chain and star [34].



**Figure 3.4:** The daisy chain device configuration of multiple Azure Kinect synchronizations. There is a maximum of eight subordinate devices synchronized to one master device [34]. This image also shows where to plug in the audio cables to distinguish between a master devices and subordinate devices.

We chose to use the daisy-chain configuration because it provides us with a wider view of our interest area. For every device in this configuration, a total of number of devices minus one audio cables are required. The audio cable connection determines the order

hierarchy in the program. The master device has an audio cable in the *Sync Out* port (the master's *Sync In* port must be empty) with a corresponding subordinate device with the same cable plugged into its *Sync In* port. Every other subordinate device follows suits, as depicted in Figure 3.4. The last subordinate device should have an empty *Sync Out* port.

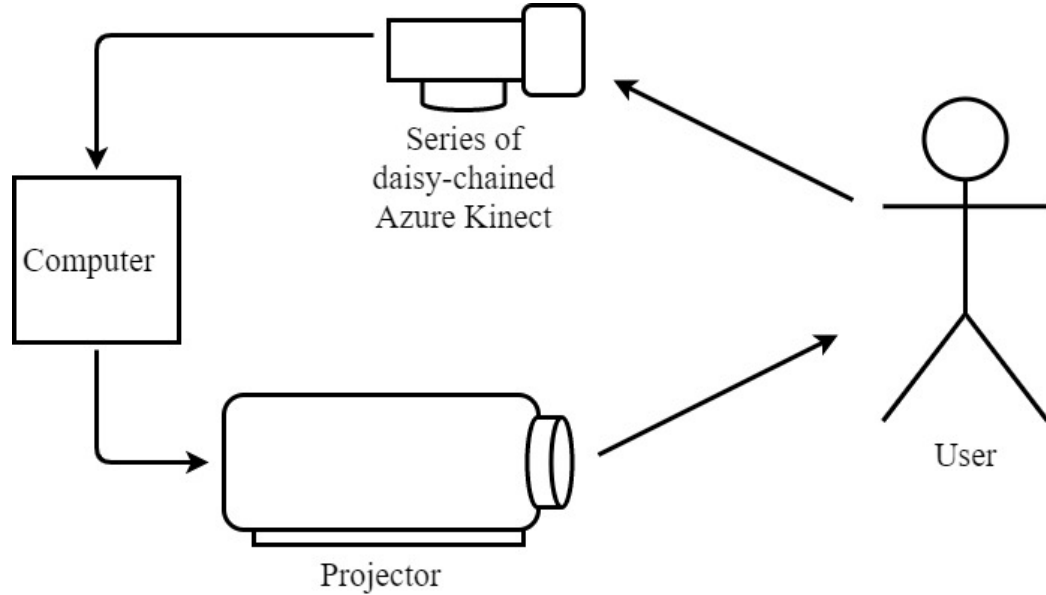
The following are benefits of using and synchronizing multiple Azure Kinects:

- Fills in occlusions that may occur from foreground objects, shadows, etc.
- Scans objects in three dimensions
- Increase the effective frame rate to a value that's greater than 30 frames per second (FPS)
- Capture multiple 4K color images of the same scene, all aligned within 100 microseconds ( $\mu s$ ) of the center of exposure
- Increase camera coverage within the space [34].

### 3.1.5 Note on Latency

Although the Azure Kinect DK is powerful, it produces a latency when body tracking is involved. The Bodytracking SDK processes tracking through a Deep Neural Network (DNN). As of developing this project, Microsoft has not provided details of the DNN supplied such as the type of propagation, the activation function, the cost function, and the number of hidden layers. While DNN is a robust machine learning algorithm, it is dependent on the quality and performance of the GPU for efficiency because of its core nature for parallel processing. Not only is there lag time in the software, but also the hardware. Microsoft measures the idle time for the depth sensor

as  $1450\mu s$  and the exposure time as  $12.8ms$  for the depth modes we consider [30]. As a note, we only consider the narrow field-of-view depth modes because they are faster and reach bodies at a longer distance than the depth mode for the wide field-of-view. We highlight the depth mode we use in Section 3.1.2. Spatial augmented reality requires a projector, which unfortunately also contains visual latency. For example, the Epson PowerLite Pro G5650W, the projector used for testing, its max sync rate (V x H) is  $85Hz \times 92kHz$  [1]. A system diagram for our hardware configuration and system communication is illustrated in Figure 3.5. These latency rates assured us that there’s a need to implement some sort of motion prediction.



**Figure 3.5:** System diagram of how our system communicates between each hardware and the user. The Kinects read data from a user, as a gesture and/or their joint information. The computer reads the data from the Kinects and sends it to the projector. The projector, in turn, projects the computer’s images onto the user.

## 3.2 Development Environment

There are three integrated development environments (IDEs) to choose from with the Azure Kinect as our hardware platform: Visual Studio, Unity, and Unreal. Microsoft has written base codes and plugins for Visual Studio and Unity to incorporate the Azure Kinect. There are plugins that non-affiliated programmers have developed for support in the Unreal engine. We have chosen not to use Unreal based on the advantages that Visual Studio and Unity provide which we discuss in the next sections.

### 3.2.1 Visual Studio

Visual Studio is an IDE that is used to edit, debug, and build code and then eventually publish it as an app. In addition to the standard editor and debugger that other IDEs provide for the software development process, Visual Studio includes compilers, code completion tools, graphical displays, and more. The following are the reasons why we chose this IDE to develop on with the majority of the project:

- IntelliSense for C++ and GLSL files
- Git management and repo creation in the IDE
- Debugger [19]

The IntelliSense is especially helpful when interacting with OpenGL's libraries. Microsoft made a git management that is easy to use and visually helpful in comparison to attempting to recall git commands and typing them on the command line of a terminal. However, the feature with the most impact, especially with computer graphics applications, is the debugger. Visual Studio allows you to pause code execution at

any moment or to insert a breakpoint in order to inspect a bug. There is even a feature for stepping back through the lines of code in case the debugger went too far forward or there was an unexpected change. Even though there is no way to pause code execution and debug inside the GLSL shader code, the debugger for the C++ code eases the software development process.

### **3.2.2 Unity**

Unity is a cross-platform game engine developed by Unity Technologies. Projects and games developed in Unity can be in 3D, 2D, virtual reality, augmented reality, and simulations. Features of the game engine include editors for audio, animation, graphics, UI, and more. However, the most prominent feature is Unity’s Assets from its Asset Store. A Unity Asset is “an item that you can use in your game or project” [37] in the form of a 3D model, audio file, image, etc. We were planning on developing entirely in Visual Studio. However, there was an opportunity to display this project, discussed in Section 4.3.2 with a short time frame and the access to Unity’s Assets gave us a quick turnaround. This is because instead of spending time and resources on creating 3D models and textures, we focused on the effects and gesture implementations the artist desired.

We choose to run our project on Unity 2019 because the assets we add perform better and more consistently than in Unity 2020. The Azure Kinect SDKs assets, a paramount asset for our system, often crashed Unity 2020.

### 3.3 Kinect Setup and Configuration

Microsoft’s base code contains an implementation to configure one device, but not for multiple synchronized devices. In order to initialize a sensor, there are a few basic steps:

- Open the device and save it to a device handler
- Set configuration parameters for the device and use it along with the device handler to start the camera
- Acquire the depth camera calibration and save it to a calibration handler
- Set configuration parameters for a body tracker and use it along with the calibration handler to create a body tracker

Because we are using multiple synchronous Azure Kinects, we loop through the above steps for the amount of devices available and we set specific device configuration parameters instead of using Azure Kinect’s Sensor SDK default parameters to configure a device.

The key parameter is called *wired\_sync\_mode* which configures the device to synchronize as a stand alone device (*K4A\_WIRED\_SYNC\_MODE\_STANDALONE*), a master device (*K4A\_WIRED\_SYNC\_MODE\_MASTER*), or a subordinate device (*K4A\_WIRED\_SYNC\_MODE\_SUBORDINATE*). Before step 2 from above, we use the Sensor SDK’s method called *k4a\_device\_get\_sync\_jack* which obtains the device jack status for the *Sync In* and *Sync Out* connectors as boolean values. Both connectors returning as false will occur when there is one device or multiple devices that are not connected synchronously and the device(s) will configure as a standalone

device. If the method returns with both connectors as true or only the *Sync In* connector is true, then the device is configured as subordinate. A return of false for the *Sync In* connector and true for the *Sync Out* connector is configured as the master device. The last requirement for device set up is waiting for all subordinate devices to finish steps 1-4 from above before performing steps 2-4 on the master device. To prevent the lasers from interfering with one another, Microsoft recommends the camera captures to be offset from one another by  $160\mu s$  or more [34]. For less latency, we use  $160\mu s$ .

In order to properly shutdown and destroy the trackers, stop the cameras, and close the devices, we keep store every device and tracker handler after their creation. We also create a new *tracked\_body* class we developed for every device to obtain and keep track of joint positions from each device. Every frame we call our update function from our *KinectSystem* class. Within it, we get a capture from the devices and use it in our body tracking process through either a body index map or a body joint structure.

### 3.4 Joint Tracking

Using the capture from the Kinect setup, we obtain body tracking information in the form of a joint hierarchy for each body in the frame. We document its tracking status, body ID, and skeleton structure (as described in Section 3.1.3). We designate the active user in our system to be the body closest to the device. The reason for having solely one active user it is simpler to have only one user controlling the system. Among all the bodies being tracked, we discover which is closest to the device, note its ID, and record all joint positions that have a decent confidence level. After each

body’s joints are tracked, we record the velocity of each joint based on a previously recorded position.

### 3.4.1 Body Tracking

Using the capture from the Kinect setup, we obtain body tracking information in the form of an index map. Although the map contains vital information for body tracking, it does not contain information about each pixel’s world position. The point cloud image, which we can obtain through the depth image, stores this information. We request the 320x288 depth image from the camera and transform it to a point cloud image using the Sensor SDK’s *k4a\_transformation\_depth\_image\_to\_point\_cloud*. This function returns the point cloud image as an array containing 3D cartesian coordinates of every pixel in the depth image. A masked image is created based on the point cloud image and index map. For each valid pixel (a pixel with a z-value that is not equal to zero and designated as a non-background pixel by the index map), we store the position in world space, a color based on the body ID the pixel belongs to, and the transformation matrix we want to apply to the pixel in the vertex shader. A pixel that is discerned as part of the background and not a body is given a black color property.

## 3.5 Minimizing Tracking Error

This section describes our software implementations in real-time body tracking and projection mapping. Our solution to improve this uses motion prediction and tracking error minimization. We explored a technique that utilizes the depth map from Kinect’s depth camera to build a point cloud map. We also devised a method that uses multiple Kinects positioned in different positions to average similar joint angles



and that performs a velocity prediction calculation to more accurately provide joint coordinates.

### 3.5.1 Point Cloud Map

We use our completed point cloud map and send it to the GPU to mark which pixels fabricate the outline of the bodies. The program performs a 2D compute operation with a work group size of (8, 18, 1) and a local size for each work group of (40, 16, 1). A compute shader with this work group configuration is able to execute 92,160 or 320x288, the resolution of our depth image, processes in parallel. Each invocation uses the global invocation ID variable (*gl\_GlobalInvocationID*) to calculate the current index and checks the current  $C(x, y)$  and neighboring pixel's color property. If  $C(x, y)$  is non-black, a new color property is assigned based on the following color map function:

$$C(x, y) = \begin{cases} \text{white,} & \text{if } C(x+1, y) = \text{black} \parallel C(x-1, y) = \text{black} \parallel \\ & C(x, y+1) = \text{black} \parallel C(x, y-1) = \text{black} \\ C(x, y), & \text{otherwise} \end{cases}$$

In other words, a pixel that is non-black and has at least one neighboring pixel that is black is marked as an outline pixel with a new white color property. Otherwise, the pixel's color property does not change. We record the GPU's execution time to average as 0.655 ms.

Based on the new outline information and previously retrieved depth values, we decrease the size of each point by different factors. If the pixel is marked as an outline pixel, the size of the billboard at the pixel location is decreased by 0.6. We also mark the point closest to ( $z_{min}$ ) and furthest away ( $z_{max}$ ) from the camera, record

those point's z-values, and map them to a scaling factor of 0.3 ( $s_{max}$ ) and 0.05 ( $s_{min}$ ), respectively. Every other pixel's scaling factor is mapped between this range created above based on their distance away ( $z'$ ) from the fore-mentioned points using this formula:

$$scale = ((z' - z_{min}) / (z_{max} - z_{min})) * (s_{min} - s_{max}) + s_{max}$$

This creates a tighter look on the body being tracked by containing a majority of the non-background pixels on the body. As Figure 3.6 shows, even with the flexible shape of a jacket, the algorithm portrays an accurate representation of the jacket's creases and deformation. Although we have not our point cloud map is not amalgamated with the other features, it was developed as separate research into minimizing tracking and projection mapping errors and aim to implement this in future effects, discussed in Chapter 5.

### 3.5.2 2D Coordinate Rotation from Averaged Joint Angles

Our second algorithm to reduce body tracking error involves averaging joint angles between each device's *tracked\_body*. Because each device has its own coordinate system, averaging joint positions from each device is not possible without multiple device calibration (discussed in Chapter 5). However, the angles made from the joints should be similar across devices. So, we developed an algorithm that generates and averages joint angles from each device and performs a 2D coordinate rotation using the newly averaged joint angles to obtain new joint positions.

First, we iterate through each device and generate a joint angle map for each *tracked\_body*. The map contains a Azure Kinect Bodytracking SDK joint enum as the key and a double for the calculated joint angle. The key corresponds to the joint for which we are calculating the angle for. To do so, we run our function *calculate\_joint\_angles*

that takes in three parameters: the joint position for the angle we are calculating ( $p_a$ ) and the two joint positions that most closely make up the body's natural angle with the first joint ( $p_b$  and  $p_c$ ). For example, if calculating the joint angle for the right elbow, then we would send the function the right shoulder ( $p_a$ ), right elbow ( $p_b$ ), and right wrist ( $p_c$ ) joint positions as arguments. In *calculate\_joint\_angles*, we create two normalized vectors: a vector from  $p_b$  to  $p_a$  which we called  $v_{ba}$  and a vector from  $p_b$  and  $p_c$  which we called  $v_{bc}$ . These vectors are calculated using the following formula:

$$v_{ba} = \text{normalize}(p_a.x - p_b.x, p_a.y - p_b.y, p_a.z - p_b.z)$$

The cosine of an angle between two vectors is equal to the dot product of the vectors divided by the product of each vector's magnitude. Using this relation, we derive the angle between our two resulting vectors. After normalizing the vectors, the derived formula is:

$$\alpha = \text{acos}(\text{dot}(v_{ba}, v_{bc}) / (\text{length}(v_{ba}) * \text{length}(v_{bc})))$$

We generate the joint angles for twelve joints that are paramount for gesture detection and performances, including the shoulders, elbows, wrists, hips, knees, and ankles.

After generating the joint angles for all *tracked\_body* instances, we average each corresponding joint angle between instances. This is calculated with a standard mean equation. The arccosine function only returns positive values:  $0 \leq \alpha \leq \pi$  in radians and  $0 \leq \alpha \leq 180^\circ$  in degrees. Therefore, a standard mean equation is valid because there are no coterminal angles. When any body joint is rotated, it can affect multiple joints. For example, a rotating left shoulder joint moves the left elbow, left wrist, and left hand joints. So in a map called *angleHierarchy*, we insert the joint as its key and the affected joints in a vector as its value. We then calculate the new joint positions using all the above angle computations:

1. Access the current joint's new angle average and old angle average
2. Calculate the delta angle ( $\theta$ ) between the two angles (new angle - old angle)
3. Iterate through all joints in *angleHierarchy* and calculate their new position using  $\theta$

In order to retrieve the new positions ( $x', y'$ ) of the joints affected by the current joint's newest angle change, we use the previous position of the affected joint *angleHierarchy* ( $p_i$ ), the position of the current joint ( $p_f$ ), and  $\theta$ :

$$\begin{aligned} x' &= ((p_i.x - p_f.x) * \cos(\theta) - (p_i.y - p_f.y) * \sin(\theta)) + p_f.x \\ y' &= ((p_i.x - p_f.x) * \sin(\theta) + (p_i.y - p_f.y) * \cos(\theta)) + p_f.y \end{aligned} \tag{3.1}$$

Once finished, we use the new positions to aid our joint position predictions in motion prediction.

### 3.6 Motion Prediction

Our motion prediction algorithm forecasts new joint positions with a change in velocity formula. It utilizes a basic change in velocity formula:

$$\Delta Velocity = \Delta Position / \Delta Time$$

With an initial velocity ( $v_i$ ), a final velocity ( $v_f$ ), an initial time ( $t_i$ ), a final time ( $t_f$ ), and an initial position ( $d_i$ ), we transform the formula in order to find an anticipated final position ( $d_f$ ):

$$d_f = d_i + (v_f - v_i) * (t_f - t_i) \tag{3.2}$$

The initial time is received from GLFW’s method *glfwGetTime*. After much trial and error, the best final time was found to be ten milliseconds or 0.01 of a second after the initial time or into the future from the current time. The joints tend to flutter when experimenting with larger values. For the initial velocity, we save each joint’s position up to the last five frametimes and use them to achieve a more accurate velocity for each joint. We not only use the speed of the joint in question, but also the speed of the two joints closest in the joint hierarchy to the current joint. We average the three velocities, obtain a delta velocity by subtracting the initial velocity from the average and multiply it by an error factor, and add the delta to the initial velocity, resulting in the final velocity. The initial position is recalled as the value of the joint position stored from body tracking. With all these values, we can plug it into the equation 3.2 and calculate our predicted final position.

### 3.6.1 Body Approximation

Because the point cloud image is not integrated with the motion prediction and angle rotation algorithms, we calculate an approximation of each body segment. As Figure 3.3 displays, the joints form a skeleton and do not align with the outline of the body. After the motion prediction process, we generate vertices using the final joint positions to construct billboards for each body segment, see Figure 3.7 for an example of the figure created from these body segment approximations. For example, a billboard constructed for the torso is calculated by using the elbow joints, shoulder joints, hip joints, and pelvis joint. For instances where only one body is guaranteed to be in view, such as testing or a dance audition with one performer, we provided a key callback tool to adjust the each body segment dimension’s scale factor to ensure a close alignment.

### 3.7 Gesture Detection

In order for active users to interact with the system, a simple gesture detection is implemented. Three gestures are detected: arms waving left, arms waving right, and hands above the head. We track them using the joint positions of the active user obtained after the motion prediction process. Waving left is detected when the *tracked\_body*'s left hand is above and to the left of its left shoulder. When the right hand is above and to the right of *tracked\_body*'s right shoulder, a waving right action is recognized. Otherwise, having both the left and right hands above the head joint signifies the hands up gesture. The active and passive users are then able to view the new effect being displayed. For both events detailed in Chapter 4.3, we designed 5-7 artistic effects for the both the active and passive users to view and for the active user to interact with. When the system recognizes a gesture from the active user, it rotates between each effect. Figure 3.8 is the state diagram of how our system interacts with gestures.

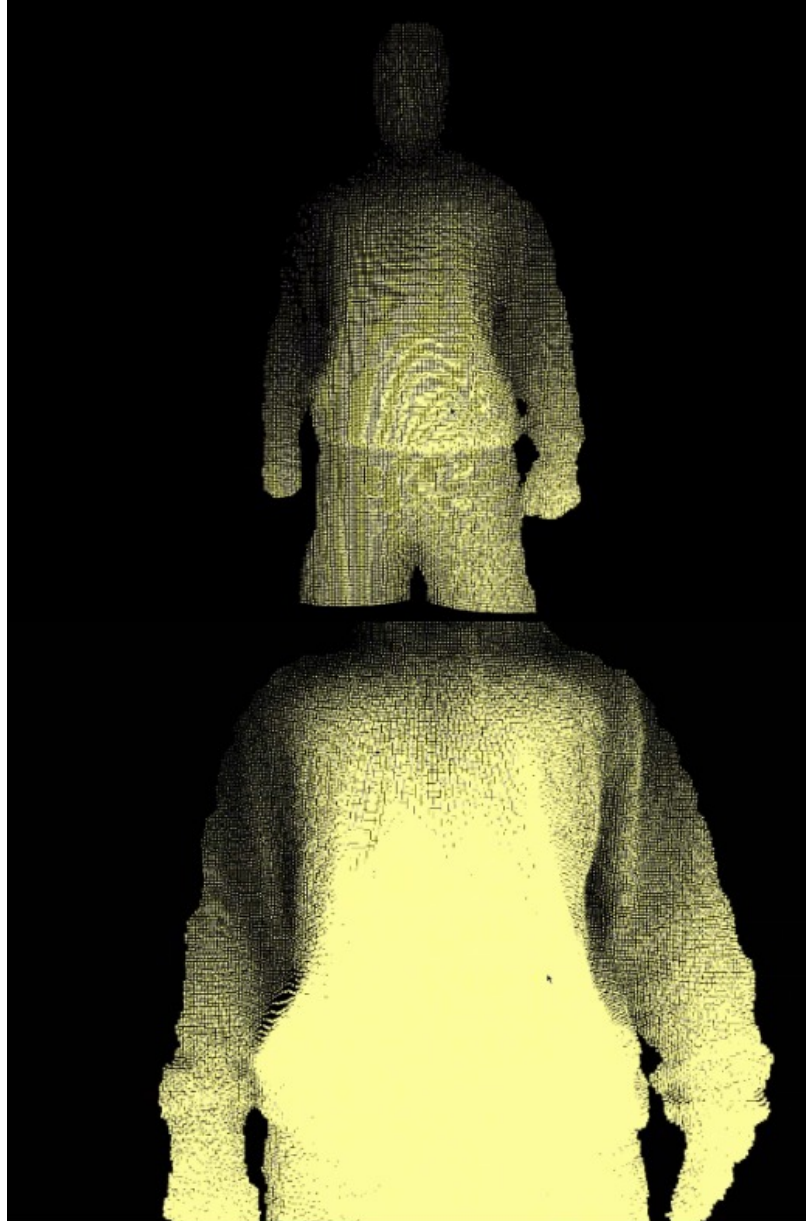


Figure 3.6: An outline result from our body projection mapping algorithm after being processed through a compute shader. The points in the point cloud map are rendered with scaled transformation matrix dependent on their distance compared to the closest and farthest body pixels in the map. The point with the smallest depth value are scaled by 0.3 and the point with the greatest depth value are scaled by 0.05. All points in between are mapped to values between 0.05 and 0.3 based on their distance to the the points with smallest and greatest depth values. Points on the outline of the body are further scaled by a factor of 0.6.



Figure 3.7: Body outline results from approximating body segment dimensions based on the predicted joint positions.



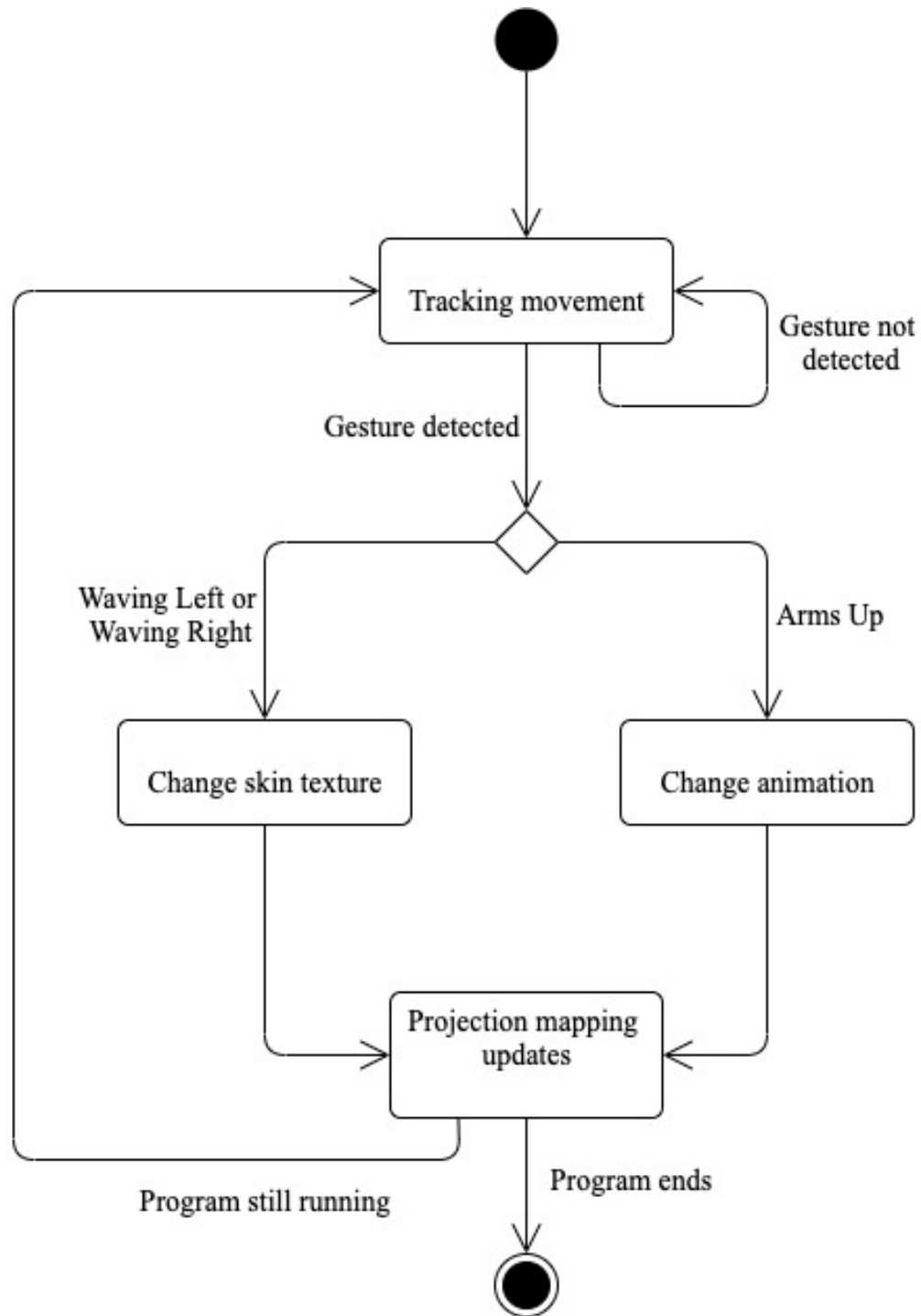


Figure 3.8: State diagram of how our system handles gesture detection. It tracks the movement of the user then switches states if a gesture is detected, otherwise it keeps trackign movement. If waving left or waving right is detected, then the skin textures effect (i.e. chameleon, see 4.3 for more) are changed. If arms up is detected, then the animation effects (i.e. dolphin swimming around, see 4.3 for more) are changed. Then, the projection mapping is updated and the cycle continues.

## Chapter 4

### RESULTS

This chapter provides an overview and a discussion of the findings of this thesis, including tests and events exhibiting the work.

#### 4.1 Experimental Setup

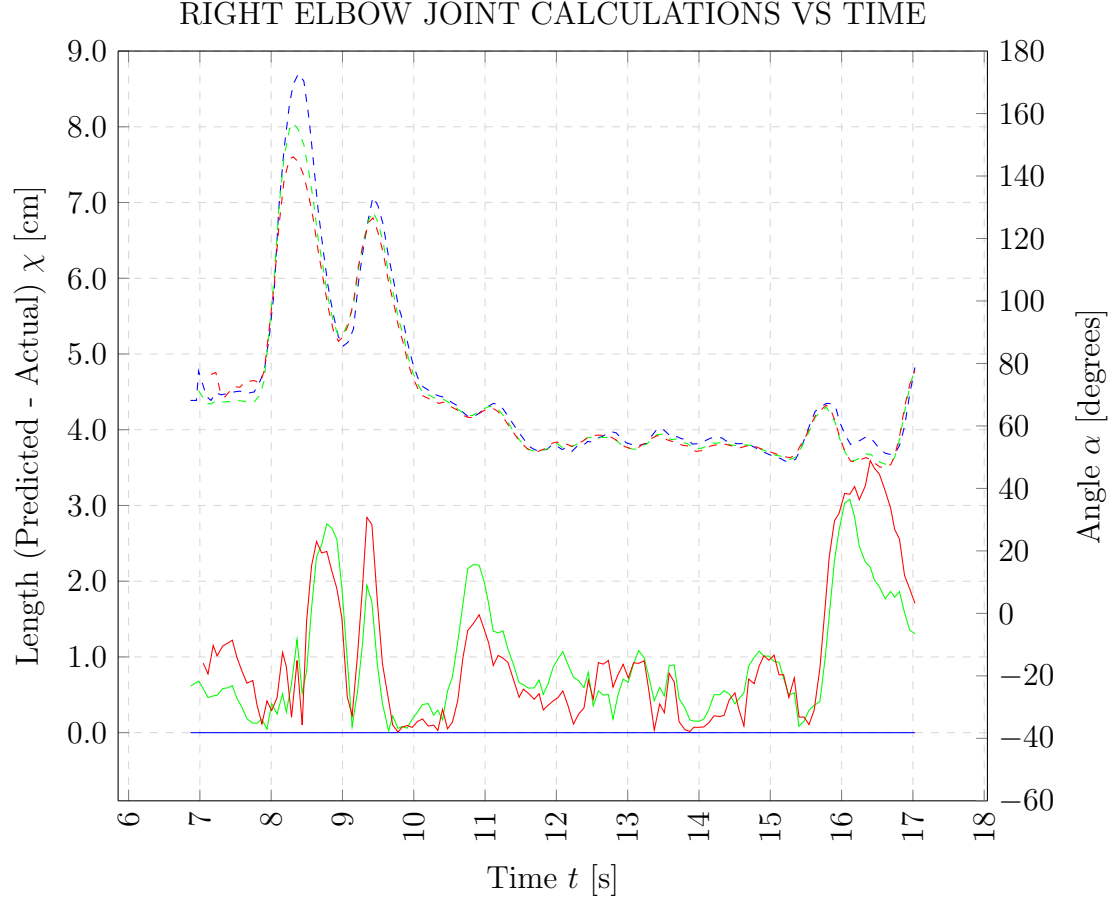
The tests were run on Windows 10 version 1909 with an i5-8400 CPU @2.80GHz processor and a 64-bit operating system. The graphics card we used was an NVIDIA GeForce GTX 1080, all of which fulfill the hardware requirements put forth by the Azure Kinect Sensor and Bodytracking SDKs. The latest SDK versions at the time were used:

- Azure Kinect Sensor SDK 1.4.1
- Azure Kinect Bodytracking SDK 1.0.1 (with its dependencies' versions at 0.9.1)

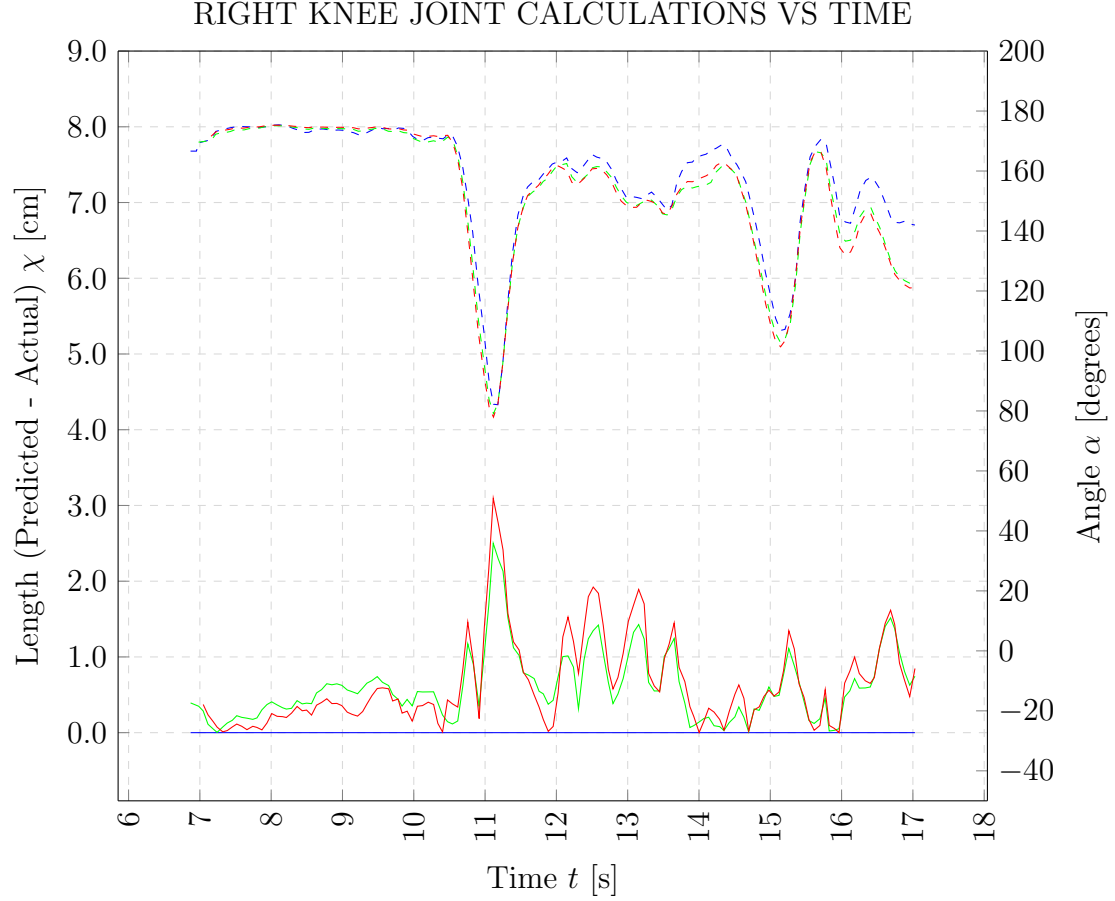
We compare body projection mapping results and execution times running configurations of 1-3 Azure Kinects.

#### 4.2 Analysis

In order to collect comparative data, we developed a test that runs on the same joint data points. We ran the program normally with a three device setup and a user in the area in front of it, recording all joint positions from each device. The movements

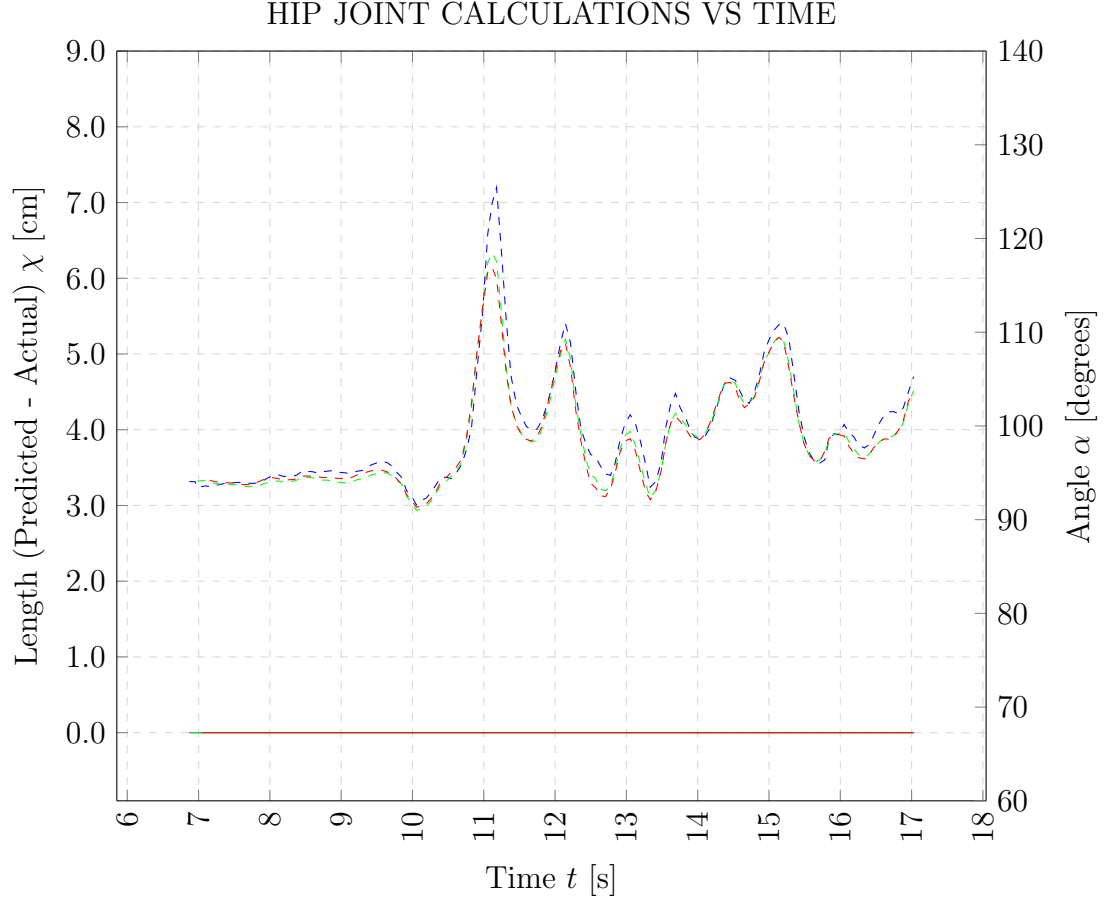


**Figure 4.1:** Dual axis line graph displaying the results of the right elbow joint's motion prediction and averaged angle. The x-axis is time measured in seconds with a tracking start time of about seven seconds. The left y-axis represents the rate of change over time of the length between the predicted position and the position computed from a 2D rotation of the right elbow joint, measured in meters. The right axis represents the change of angle, in degrees, over time at the right elbow joint, which is derived using the right shoulder, elbow, and wrist joints. The dashed lines are the angle averages. The solid lines are the position variance. Red colored lines are data from a three device arrangement. Green colored lines are data from a two device arrangement. Blue colored lines are data from a one device arrangement.



One Device:	Two Devices:	Three Devices:
— Position Discrepancies	— Position Discrepancies	— Position Discrepancies
- - Angle Averages	- - Angle Averages	- - Angle Averages

**Figure 4.2:** Dual axis line graph displaying the results of the right knee joint's motion prediction and averaged angle. The x-axis is time measured in seconds with a tracking start time of about seven seconds. The left y-axis represents the rate of change over time of the length between the predicted position and the position computed from a 2D rotation of the right knee joint, measured in meters. The right axis represents the change of angle, in degrees, over time at the right knee joint, which is derived using the right hip, knee, and ankle joints. The dashed lines are the angle averages. The solid lines are the position variance. Red colored lines are data from a three device arrangement. Green colored lines are data from a two device arrangement. Blue colored lines are data from a one device arrangement.



**Figure 4.3:** Dual axis line graph displaying the results of the right hip joint's motion prediction and averaged angle. The x-axis is time measured in seconds with a tracking start time of about seven seconds. The left y-axis represents the rate of change over time of the length between the predicted position and the position computed from a 2D rotation of the right hip joint, measured in meters. The right axis represents the change of angle, in degrees, over time at the right hip joint, which is derived using the chest, right hip, and right knee joints. The dashed lines are the angle averages. The solid lines are the position variance. Red colored lines are data from a three device arrangement. Green colored lines are data from a two device arrangement. Blue colored lines are data from a one device arrangement.



**Figure 4.4:** Body projection mapping outline depicting a snapshot from a frame of lifting the right leg up. The red colored outline is data from a three device arrangement. The green colored outline is data from a two device arrangement. The blue colored outline is data from a one device arrangement. The white pixels designates an overlap between all outlines.

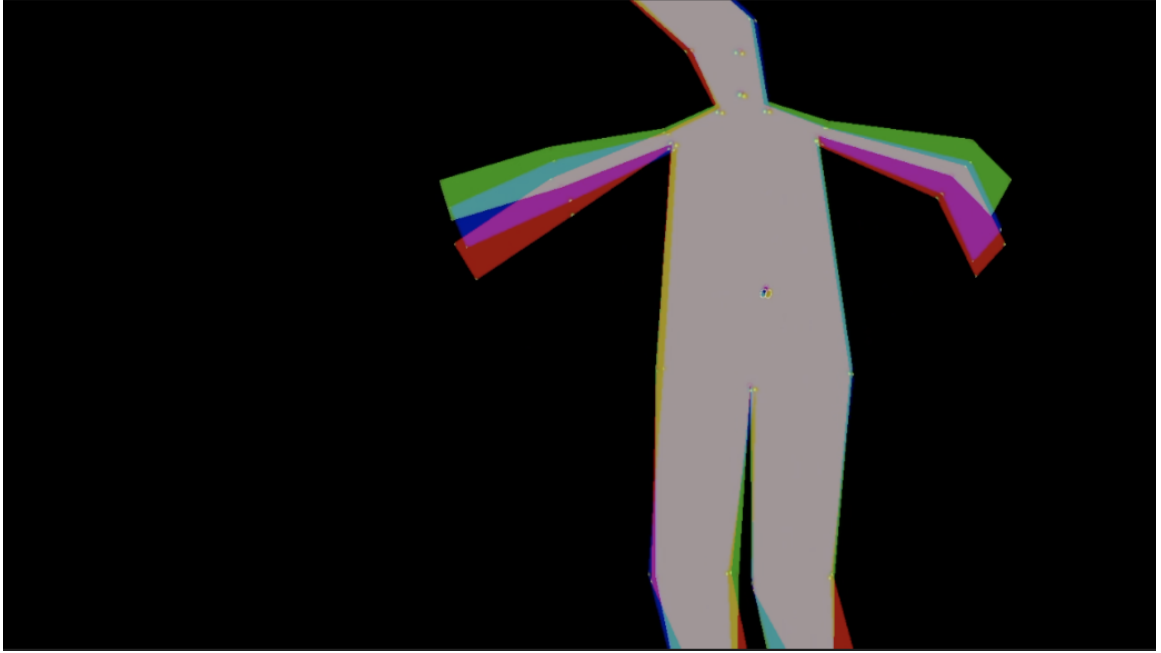
performed were waving both arms up and down, stepping with high knees to the left and right, and jumping up. We examine three dominant right joints in body movement: right elbow, right knee, and right hip. The entire animation produced 147 data points for the following graphs. The x-axis indicates time between seven and seventeen seconds. However, the program measured start time to be once it runs, not once the body was tracked and displayed. We only started collecting data after the body was tracked, so around seven seconds is our start time. Each device configuration is represented by the same color throughout all the next six figures: one device setup is colored blue, two devices setup is colored green, and three devices setup is colored red.



**Figure 4.5:** Body projection mapping outline depicting a snapshot from a frame of walking to the right. The red colored outline is data from a three device arrangement. The green colored outline is data from a two device arrangement. The blue colored outline is data from a one device arrangement. The white pixels designates an overlap between all outlines.

#### 4.2.1 Averaging Joint Angles

Once the positions were recorded for the current frame, we processed the positions through our algorithm as all three device configurations: one device (only consider data from master device), two devices (consider data from the master and one subordinate), and three devices (consider all data). Graphs 4.1, 4.2, and 4.3 display the results from the elbow, knee, and hip angles, respectively, with a dashed line and the y-axis on the right. In each graph, the overall change of rate is similar. There are slight angle discrepancies between device configurations. However, this is expected because only an orthogonal placement relative to the two vectors in the angle calculations can obtain a complete view of the angle. A camera placed at other points may process the vectors to form a differently sized angle.



**Figure 4.6:** Body projection mapping outline depicting a snapshot from a frame of landing after a jump. The red colored outline is data from a three device arrangement. The green colored outline is data from a two device arrangement. The blue colored outline is data from a one device arrangement. The white pixels designates an overlap between all outlines.

#### 4.2.2 Motion Prediction

Before averaging the angles, the elbow, knee, and hip joints from the master device were processed through our motion prediction algorithm, producing an anticipated joint position 0.01 seconds into the future. We used a timer that, during each frame, increased by the total elapsed time between frames. We waited until the timer reached 0.01 seconds to compare the master device's predicted values to the positions we calculated after performing the two-dimensional rotation based on the angle averages, described in Section 3.5.2. This is visualized in Graphs 4.1, 4.2, and 4.3, respectively, with a solid line and the y-axis on the left. Each graph shows our motion prediction algorithm produced an 100% accurate position 0.01 seconds into the future for the master device. In each observed joint for the one device configuration, the the



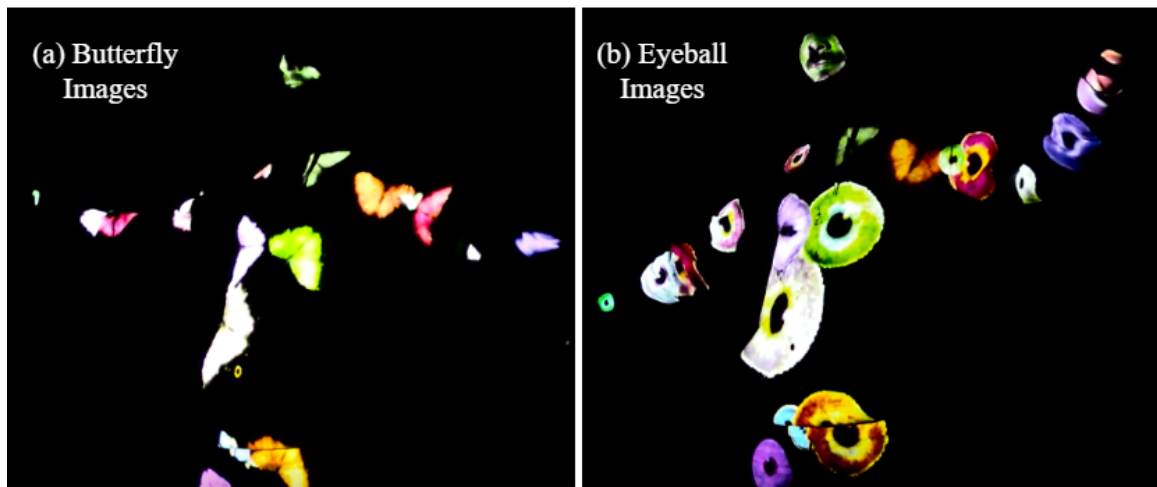
predicted position and the actual position are the same value, confirming our motion prediction algorithm. With the exception of the hip joint, the two and three device configurations contain some disparities. The elbow joint’s predictions remains less than  $4.0cm$  away from the actual prediction in both setups. The knee joint’s predictions are at most  $3.0cm$  off compared to the actual positions. The differences between the two and three device setups are not too large; in some cases, the three device setup provides a more accurate prediction. Referring to the hip joint graph, it shows that even with different angles there are certain joints that are predicted better than others. This joint was predicted positions with 100% accuracy across all device configurations. This may be because the sensors were tracking the hip positions better than the knee and elbow or because the hip movements were slower compared to the movements of the other joints. Overall, we see highly accurate predictions for joint positions using our motion prediction algorithm, with no significant accuracy discrepancies between each sensor configuration. However, we can see the slight angle and position differences from the body approximation images.

### 4.2.3 Body Approximation

We have shown the statistical differences between camera arrangements. When aligned by framerate, there are also visual differences. We compared the body projection mapping from our resulting joint angle and position computations of each device configuration, see Figures 4.4, 4.5, and 4.6. While the assessed positions and joints are slightly differing in data, the visual comparisons by frame are noticeable, especially the joint angle measurement.

### 4.3 Proof of Concept

This section discusses our results from exhibiting our body projection mapping in events we specifically designed it for. We planned to organize a performing art show with computer science students who minored in dance. We also auditioned to be in the established California Polytechnic University Dance Show for Spring Quarter 2020. However, due to social distancing protocols from COVID-19, we put planning a performance on hold and the university’s dance show was cancelled. Below we discuss the results of attending two events where the project was displayed as an interactive art installation. The two events are Burning Man 2019 and Delfines de San Carlos art show 2020.



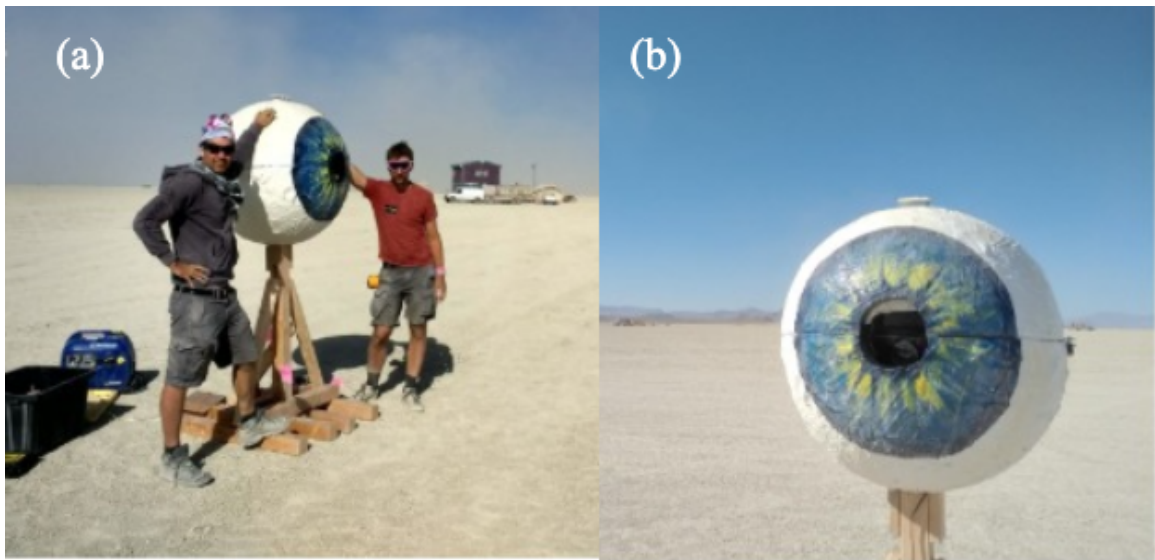
**Figure 4.7:** These figures portray the butterfly morphing effect we implemented. Over a span of a few seconds, each butterfly from (a) morph into the eyeballs in (b) at different times.

#### 4.3.1 Burning Man 2019

The project participated at the Black Rock City Art installation camp at Burning Man 2019 [26]. With an art theme of metamorphosis, we created effects that artisti-

cally correlate with the theme: butterflies that morph into eyeballs, a skeleton with a beating heart, chameleon skin that changes colors, and an animation that makes the body appear to burn away from the inside. Figures 4.7 and 1.1 depict a couple of the effects.

Due to climate and weather conditions Burning Man experiences in the middle of the desert, we needed to build a structure that could withstand the elements and accommodate a projector, Kinect sensor, and a laptop. We committed to our favorite effect and constructed an eyeball structure made up of wooden slabs for the base and fiberglass, epoxy, and paint for the containment unit, see Figure 4.8. Originally, the eyeball was an inflated yoga ball which we then pasted several layers of fiberglass using epoxy and when the final layer dried we painted it. Apart from the artistic addition, the eyeball also encases the laptop, projector, and sensor. A hole was shaped where the pupil should be located for the camera to peer out from and the projector to project out of.



**Figure 4.8:** (a): A full view of the entire eyeball structure at Burning Man 2019. The structure is 7 feet tall and the eyeball is 3 feet wide. (b): A closer view of the structure's iris and 'pupil' hole where the Kinect and projector are looking out of.

As Burning Man attendees walked past a giant eyeball structure, they were curious to investigate what metamorphosis art they could experience. A dark environment with a real-time interactive projection caught their attention and allowed them to be fully immersed with their metamorphosis experience. Burning Man noticed our popular, novel technology and invited us back as official artists for an art installation camp at Burning Man 2020. Unfortunately, Burning Man 2020 was cancelled. We are hopeful and anticipating our project to be invited to a Burning Man 2021 art installation camp.

#### **4.3.2 Delfines de San Carlos: Un Proyecto de Esperanza 2020**

Lucia Apodaca, an artist from San Carlos, Sonora, Mexico, was exposed to the art and technology of our project and showed interest in showcasing it at her art show Delfines de San Carlos: Un Proyecto de Esperanza on September 26, 2020 [15]. The art show consisted of low-polygon dolphin structures, each individually painted by various artists from Mexico and the United States.

Her team constructed an all-white dolphin structure (Figure 4.9) to map projections of animated and textured effects onto it and allow the show's guests to interact with it. The gestures described in Section 3.7 are used for the guests to switch between the seven effects in real-time, see Figure 4.10 for an example from the art show.

The other structure depicted in Figure 4.11 performs a similar function to the eyeball structure from Burning Man 2019: a place to mount the laptop, Azure Kinect, and projector. Because there was only a month of development time, we modified our project for the event to run in C# and Unity in order to take advantage of Unity's prefabricated assets. Doing so, permitted us time to tweak the variety of effects and refactor our C++ code into C#.

The attendees of the art show enjoyed watching and taking pictures of the dolphin transforming its effects in real time. Although the hardware configuration was exposed and may have decreased the quality of immersion, it sparked curiosity and discussion of how the project worked. A discussion about the art on the other dolphins also occurred, so our set up may have been advantageous for this type of event. Even though this event did not involve any body projection mapping, we were still able to showcase our body tracking algorithm when detecting gestures and were given the opportunity to projection map onto a unique static object.



**Figure 4.9:** The white dolphin structure that Apodaca's team constructed for our installation at the 2020 Delfines de San Carlos art show situated on El Mirador de San Carlos. A low polygon dolphin fixed on top of a wave-like base made up of recycled cycling wheels. Pictured from left to right: Noah Paige, Sydney Baroya, Irene Humer, Christian Eckhardt, Lucia Apodaca, Sara Valle (Mayor of Guaymas), and Enrique Gamez (comisario of San Carlos).



**Figure 4.10:** The psychedelic effect developed for the projection mapping on to the dolphin structure. This effect was triggered when an active user waved their arms left or right. Pictured is the team for our interactive installation at the Delfines art show from left to right: Noah Paige, Sydney Baroya, Christian Eckhardt, and Irene Humer.



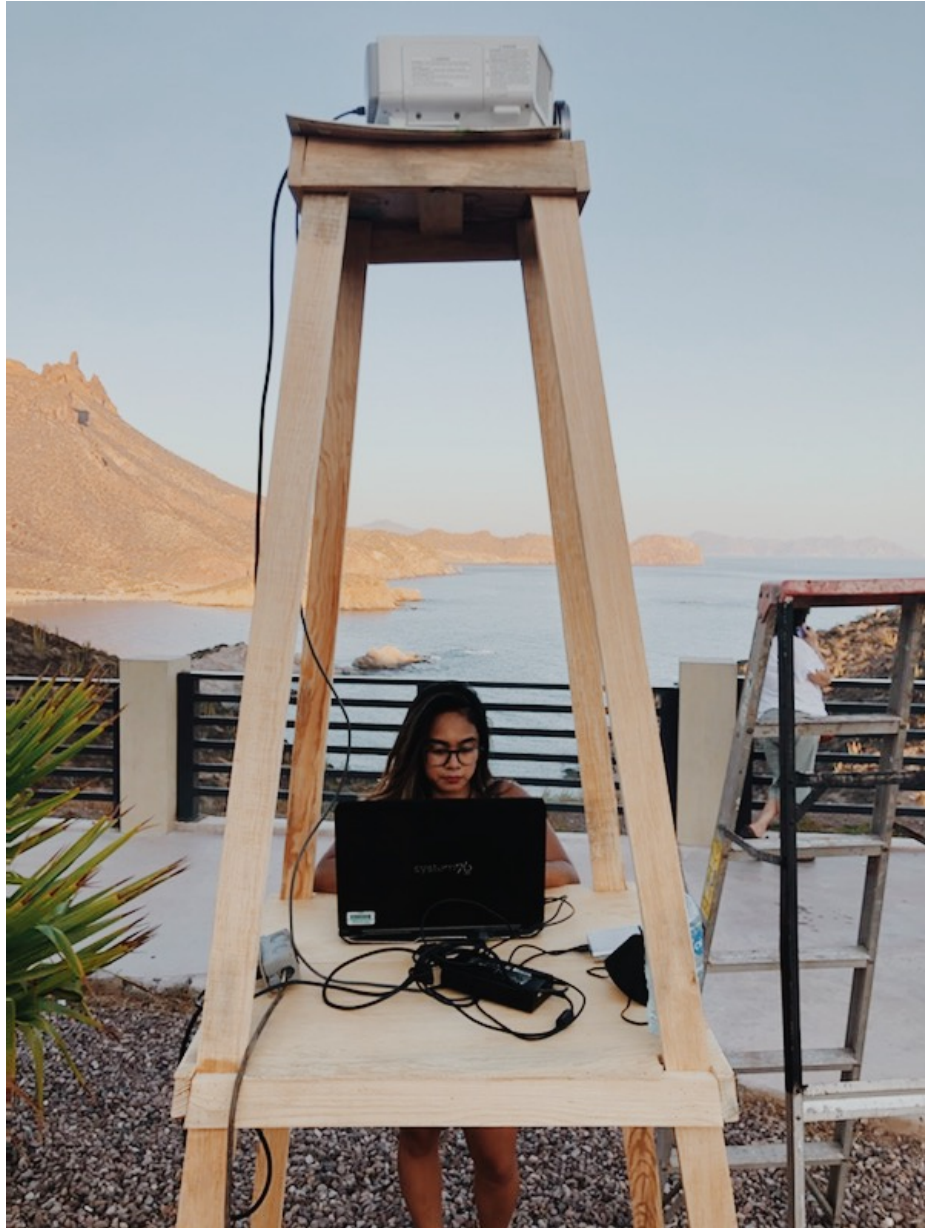


Figure 4.11: The laptop/projector/Kinect setup for the ‘Delfines’ art show. The dolphin structure that we are mapping a projection onto is mounted on a tall base so we required a tall structure for the projector to sit on along with a platform to keep the laptop and Kinect at body level.



## Chapter 5

### FUTURE WORK

Although the results and the proof-of-concept events are successful, we have identified a few features that would enhance the project. The main work we would like to develop pertains to our point cloud image. First, we would calibrate all the devices to produce images in a single domain. In a single device, the depth and RGB cameras are factory calibrated to work together. When multiple devices are synchronized, they have to be calibrated to transform an image from the domain of the camera that captured it to the domain of the camera you want to use to process images [34]. We would develop an algorithm that could calibrate devices in order to produce a cohesive depth map and consequently a cohesive point cloud map. Second, we would combine the results from averaging joint angles and motion prediction with our point cloud image. To do so, we would use the point cloud image as a mask texture and employ it to mask the body approximated figure, see Figure 4.4.

The next two features are associated with the projector. Taking the resolution and lag rate of a projector into account could provide a sharper image and further combat latency. Although we do predict motion, any reduction of latency would improve our work. Another projector related improvement involves adding a multi-projector configuration. This could create various viewing angles that do not vary in quality. Creating an effective and efficient multi-projector system is an ongoing research topic. This feature would update our hardware set-up to place projectors surrounding the object(s) of interest and place 1-2 Kinect sensors by each projector, giving each projector the “view” they are to project. We would also need to develop an algorithm that fixes effects from overlapping pixels between projectors.

One factor that affects virtual objects to look natural is shadowing and specular highlights. With a static light, shadowing can be accomplished in a single render pass. However, specular highlights can be complicated in multi-user systems. Lighting effects are typically rendered from one view – the camera view – because users view it from a 2D screen. When augmented reality projects onto a 3D canvas like a body, it is viewable by multiple users at various angles, complicating the light rendering. We would need to update specular lighting as users move. To decrease the complexity, we can utilize the real-time detection for the body closest to the camera as our view instead of distorting all views. This method would not be viable in a performing art show, but an interactive art show’s immersion will be increased with this implementation.

While our program is able to project around the outline of flexible shapes, the projected effects may be distorted on the non-rigid surfaces. The next feature we would implement takes deformed non-rigid surfaces into consideration so that the art stays cohesive in spite of surface deformations. Users and performers will be dressed with clothes or costumes, so adding this update is a necessary step to increase our immersion.

Although we are confident with our motion prediction and gesture detection, they can be upgraded for an improved result. Wayne McGregor from “Google Arts & Culture and Studio Wayne McGregor” uses artificial intelligence (AI) in the form of a neural network to predict a dancer’s movements [18]. Implementing a similar artificial intelligence could be helpful for a planned performance. With time for gathering training input, the AI could learn the individual styles of each dancer in a performance and predict their movements. In an interactive art installation, the AI would improve the general motion prediction with each user. Similarly, an addition

of AI can enhance the gesture detection process, either in a performing art show or an interactive art installation.

## Chapter 6

### CONCLUSION

This thesis presents augmented reality and human computer interaction in the interactive arts through body projection mapping. To the best of our knowledge, this is the first solution in this area implemented on Microsoft’s Azure Kinect Developer Kit. Using the skeleton joint hierarchy from multiple Azure Kinects, we obtain an accurate, real-time body approximation with our 2D coordinate rotation of averaged joint angles and joint motion prediction algorithm. Combining the body index segmentation map and the depth maps from the Azure Kinect, a masked point-cloud image is achieved that is able to detect flexible shaped clothing. We present our results from averaging joint angles from different device configurations and from prediction joint movement 0.01 seconds into the future. We also provide our results from exhibiting our project at Burning Man 2019 and Delfines de San Carlos: Un Proyecto de Esperanza art show 2020. While we produced a high-quality body projection mapping, there are updates for each stage and additional features that can enhance it. We look forward to future research that takes advantage of the Azure Kinect features and any future updates to the Azure Kinect Sensor and Bodytracking SDKs.

## BIBLIOGRAPHY

- [1] Epson Powerlite Pro G5650W - 3LCD projector - LAN Specs.  
<https://www.cnet.com/products/epson-powerlite-pro-g5650w-3lcd-projector-lan/>.
- [2] OpenGL Texture. <https://www.khronos.org/opengl/wiki/Texture>.
- [3] Texture masks using a shader, Oct 2017.  
<http://pirron.one/playingincanvas/texture-masks-using-shader>.
- [4] J. Baham. *The Unauthorized Story of Walt Disney's Haunted Mansion*. Theme Park Press, 2016.
- [5] O. Bimber and R. Raskar. *Spatial augmented reality: merging real and virtual worlds*. CRC press, 2005.
- [6] C. Cao, M. Preda, and T. Zaharia. 3d point cloud compression: A survey. In *The 24th International Conference on 3D Web Technology, Web3D '19*, page 1–9, New York, NY, USA, 2019. Association for Computing Machinery.
- [7] C. Chen, B. Wang, C. X. Lu, N. Trigoni, and A. Markham. A survey on deep learning for localization and mapping: Towards the age of spatial machine intelligence. *arXiv preprint arXiv:2006.12567*, 2020.
- [8] J. Chen, T. Yamamoto, T. Aoyama, T. Takaki, and I. Ishii. Simultaneous projection mapping using high-frame-rate depth vision. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4506–4511, 2014.

- [9] J. Chen, T. Yamamoto, T. Aoyama, T. Takaki, and I. Ishii. Real-time projection mapping using high-frame-rate structured light 3D vision. *SICE Journal of Control, Measurement, and System Integration*, 8(4):265–272, 2015.
- [10] J. de Vries. Shadow mapping.  
<https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping>.
- [11] W. Delamare, C. Coutrix, and L. Nigay. Designing guiding systems for gesture-based interaction. In *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 44–53, 2015.
- [12] T. Dubnov, Z. Seldess, and S. Dubnov. Interactive projection for aerial dance using depth sensing camera. *Proceedings of SPIE - The International Society for Optical Engineering*, 01 2014.
- [13] L. Frum. First impressions:Microsoft’s Kinect gaming system, Nov 2010.  
<https://www.cnn.com/2010/TECH/gaming.gadgets/11/03/kinect.video.game/>.
- [14] Y. Ghazwani and S. Smith. Interaction in Augmented Reality: Challenges to Enhance User Experience. In *Proceedings of the 2020 4th International Conference on Virtual and Augmented Reality Simulations*, pages 39–44, 2020.
- [15] S. Gobo. Interactive Art ”Delfines” Sydney Baroya. YouTube video, Sept. 27 2020. <https://youtu.be/Rl8RIrEMia4>.
- [16] J. Lee, Y. Kim, M.-H. Heo, D. Kim, and B.-S. Shin. Real-time projection-based augmented reality system for dynamic objects in the performing arts. *Symmetry*, 7(1):182–192, 2015.

- [17] S. Lee, N. U. Islam, and S. Lee. Robust image completion and masking with application to robotic bin picking. *Robotics and Autonomous Systems*, page 103563, 2020.
- [18] D. Leprince-Ringuet. Google’s latest experiment teaches ai to dance like a human, Dec 2018. <http://www.wired.co.uk/article/google-ai-wayne-mcgregor-dance-choreography>.
- [19] Microsoft. Visual Studio 2019: Download for free.  
<https://visualstudio.microsoft.com/vs/>.
- [20] Microsoft. Buy the Azure Kinect developer kit – Microsoft, 2019.  
<https://www.microsoft.com/en-us/p/azure-kinect-dk/8pp5vxmd9nhq?activetab=pivot%3Aoverviewtab>.
- [21] Microsoft. microsoft/Azure-Kinect-Samples, Jun 2019.  
[https://github.com/microsoft/Azure-Kinect-Samples/tree/master/body-tracking-samples/sample\\_helper\\_libs/window\\_controller\\_3d](https://github.com/microsoft/Azure-Kinect-Samples/tree/master/body-tracking-samples/sample_helper_libs/window_controller_3d).
- [22] Microsoft. microsoft/azure-kinect-samples, Jan 2020.
- [23] M. R. Mine, J. Van Baar, A. Grundhofer, D. Rose, and B. Yang. Projection-based augmented reality in disney theme parks. *Computer*, 45(7):32–40, 2012.
- [24] S. A. Mokhov, D. Li, H. Lai, J. Singh, Y. Shen, J. Llewellyn, M. Song, and S. P. Mudur. ISSv2 and OpenISS distributed system for real-time interaction for performing arts. In *ACM SIGGRAPH 2019 Posters*, pages 1–2. 2019.
- [25] A. J. Morrison, P. Mitchell, and S. Viller. Evoking gesture in interactive art. In *Proceedings of the 3rd ACM international workshop on Human-centered computing*, pages 11–18, 2008.

- [26] music vizards. 2019 Art Installations, 2019.  
[https://burningman.org/culture/history/brc-history/event-archives/2019-event-archive/2019-art-installations/?yyyy=.](https://burningman.org/culture/history/brc-history/event-archives/2019-event-archive/2019-art-installations/?yyyy=)
- [27] J. Otepka, S. Ghuffar, C. Waldhauser, R. Hochreiter, and N. Pfeifer.  
 Georeferenced point clouds: A survey of features and point cloud management. *ISPRS International Journal of Geo-Information*, 2(4):1038–1065, Oct 2013. <http://dx.doi.org/10.3390/ijgi2041038>.
- [28] Paul. Simple Bumpmapping.  
<http://www.paulsprojects.net/tutorials/simplebump/simplebump.html>.
- [29] R. Raskar, G. Welch, K.-L. Low, and D. Bandyopadhyay. Shader lamps: Animating real objects with image-based illumination. In *Eurographics Workshop on Rendering Techniques*, pages 89–102. Springer, 2001.
- [30] T. Sych and B. Allen. Azure Kinect DK hardware specifications, Feb 2020.  
<https://docs.microsoft.com/en-us/azure/kinect-dk/hardware-specification>.
- [31] T. Sych, B. Allen, and P. Meadows. Azure Kinect DK depth camera, Jun 2019.  
<https://docs.microsoft.com/en-us/azure/kinect-dk/depth-camera>.
- [32] T. Sych, B. Allen, P. Meadows, and C. Edmonds. Azure Kinect Sensor SDK system requirements, Mar 2020.  
<https://docs.microsoft.com/en-us/azure/kinect-dk/system-requirements>.
- [33] T. Sych, P. Meadows, and D. Coulter. About Azure Kinect DK, Jun 2019.  
<https://docs.microsoft.com/en-us/azure/kinect-dk/about-azure-kinect-dk>.
- [34] T. Sych and T. Sherer. Synchronize multiple Azure Kinect DK devices, Feb 2020.  
<https://docs.microsoft.com/en-us/azure/kinect-dk/multi-camera-sync>.



- [35] T. Sych, Y. Wang, B. Allen, S. Leavitt, J. Parente, J. Borsechnik, P. Meadows, and M. Bradley. Azure Kinect body index map, Jun 2019.  
<https://docs.microsoft.com/en-us/azure/kinect-dk/body-index-map>.
- [36] T. Sych, Y. Wang, J. Dunn, P. Meadows, M. Bradley, and K. Toliver. Azure Kinect body tracking joints, Jul 2019.  
<https://docs.microsoft.com/en-us/azure/kinect-dk/body-joints>.
- [37] U. Technologies. Quick guide to the unity asset store.  
<https://unity3d.com/quick-guide-to-unity-asset-store>.
- [38] A. Vovk. [dc] dimensionality of Augmented Reality Spatial Interfaces. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 1377–1378, 2019.
- [39] L. Vural. *Instructional Methods*, pages 107–145. 11 2016.