A HoloLens Framework for Augmented Reality Applications in Breast Cancer Surgery

by

George Poothicottu Jacob

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2018

Abstract

This project aims to support oncologic breast-conserving surgery by creating a platform for better surgical planning through the development of a framework that is capable of displaying a virtual model of the tumour(s) requiring surgery, on a patient's breast. Breast-conserving surgery is the first clear option when it comes to tackling cases of breast cancer, but the surgery comes with risks. The surgeon wants to maintain clean margins while performing the procedure such that the disease does not resurface. This calls for the importance of surgical planning where the surgeon consults with radiologists and pre-surgical imaging such as Magnetic Resonance Imaging (MRI). The MRI prior to the surgical procedure, however, is taken with the patient in the prone position (face-down) but the surgery happens in a supine position (face-up). Thus mapping the location of the tumour(s) to the corresponding anatomical position from the MRI is a tedious task which requires a large amount of expertise and time given that the organ is soft and flexible.

For this project, the tumour is visualized in the corresponding anatomical position to assist in surgical planning. Augmented Reality is the best option for this problem and this, in turn, led to an investigation of the application capability of the Microsoft HoloLens to solve this problem. Given its multitude of sensors and resolution of display the device is a fine candidate for this process. However, the HoloLens is still under development with a large number of limitations in its use. This work tries to compensate for these limitations using the existing hardware and software in the device's arsenal.

Within this masters thesis, the principal questions answered are related to the acquiring of data from breast mimicking objects in acceptable resolutions, discriminating between the information based on photometry, offloading the data to a computer for post-processing in creating a correspondence between the MRI data and acquired data, and finally retrieving the processed information such that the MRI information can be used for visualizing the tumor in the anatomically precise position. Unfortunately, time limitations for this project led to an incomplete system which is not completely synchronized, however, our work has solidified the grounds for the software aspects toward the final goals set out such that extensive exploration need only be done in the imaging side of this problem.

Acknowledgements

After a process that went through 16 months, today I am writing the final piece to my masters thesis. The past months have been an intense learning experience for me on an educational and personal level. Knowing full-well that no small number of words would be enough in describing how grateful I am to the people who have continuously supported me and helped me make this happen, I hope that this page would be a start.

First and foremost, I would like to thank my Supervisor Prof. Oleg Michailovich and Co-Supervisor Prof. William D. Bishop without whom this thesis would not have been possible. During my time in the University of Waterloo they have helped me expand and deepen my knowledge through the process of my research, all the while being incredibly supportive and motivational. José, Amir, Hossein, Rui, Rinat and Ming, sharing an office with every single one of you has been an amazing experience.

To Robin, Shalabha, Vimala, Bharat, Mani, JJ, Hareesh, Sara, Vikky, Mahesh, Nirmal and Ranga, thank you for making Waterloo a home away from home. The past two years in this city were amazing all of you. If anything, the experiences we shared would be something I would always cherish in the years forward.

A special shout out to my buddies back in India - Akhila, Hemanth, Aby, Jacob and Thareeq, who despite the distance, always made me feel like they live right next door. Finally, I thank my family, for all the love, care and belief I've been lucky enough to receive.

From the bottom of my heart, I thank every single one of you, if not for which, this thesis would never have been complete.

Dedication

To Papa, Amma and Ichuppi.

Table of Contents

List of Tables

List of Figures

Abbreviations

AR Augmented Reality 2–7, 31, 37, 54

DLL Dynamic-Link Library 18

ISR Incremental Super Resolution 33, 43, 46–48, 55, 56, 62, 78

MRI Magnetic Resonance Imaging iii, 1, 2, 5, 7, 15, 19, 24, 33–35, 48, 53, 69, 70

MRTK Mixed Reality Toolkit 11, 17, 18, 20, 37, 39, 40

SDK Software Development Kit 6, 11, 17, 18, 24, 27, 35, 37, 39, 41

Chapter 1

Introduction

Breast Cancer is a widespread disease that is statistically shown to affect one in every eight women in Canada, with the disease estimated to be diagnosed in approximately 26,000 women in the year 2017 [5]. The mortality rate in records for breast cancer has seen a decline especially with the introduction of Mammography screening yet some cancers still remain occult to this process. This is especially the case when it comes to denser breast tissues or cancers that have spread to multiple foci. With this problem, patients, especially with a family history of the disease are referenced to MRI for diagnosis. MRI in itself is still far from being used as a screening tool but is standard procedure when it comes to surgical planning.

When it comes to tackling breast cancer through the process of surgery, it is possible to perform a Mastectomy, which results in the complete removal of the organ, thus preventing any relapse. However, from a patient's perspective, a breast-conserving surgery is always preferred over the complete removal of the organ. This, however, puts the surgeon in a bind. On one hand, there is an intention to preserve as much tissue of the breast as possible, while on the other, the surgeon has to make sure that there are clean margins of excision of the tumor(s) such that the risk of a relapse could be avoided preventing additional expenditure and emotional burden for the patient.

When it comes to the surgical planning of breast cancer, the surgeon consults the radiology team. A MRI is preferred in this scenario especially when it comes to cancers that are non-palpable or occult to mammography [59]. During the MRI data collection, the patient is placed in a prone position as seen in Figure 1.1, while the actual surgery takes place with the patient being in a supine position. Due to the significant softness of the organ in question, the deformation is substantial with the position change, adding another

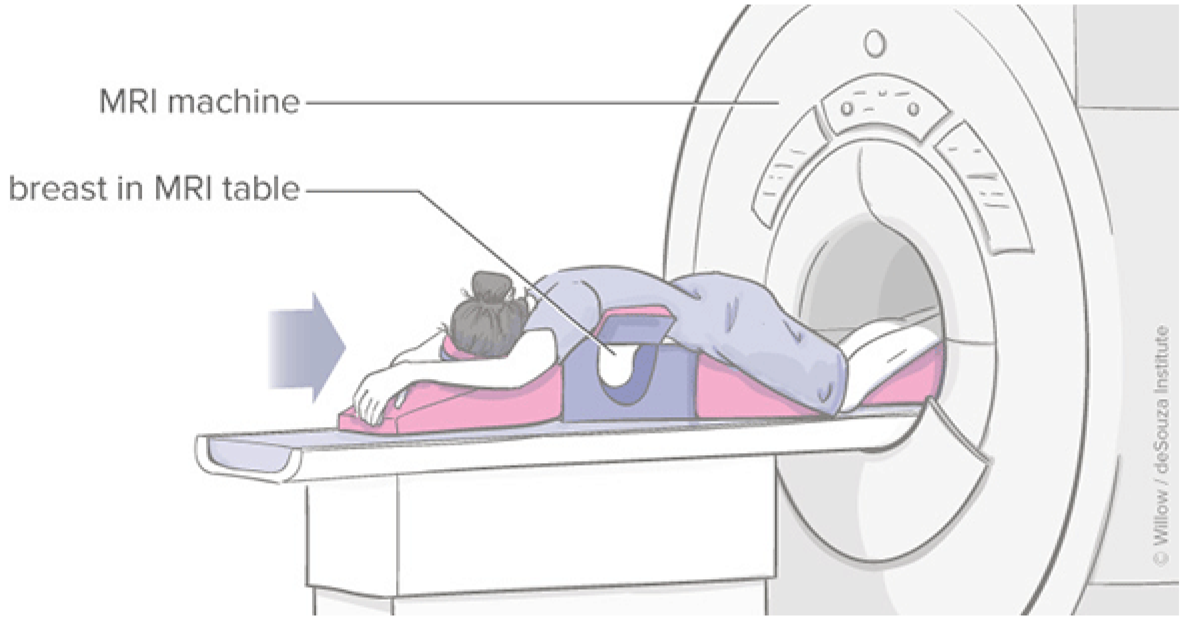


Figure 1.1: MRI Screening in the Prone Position [13]

layer of complexity for the surgical procedure with cancer occupying a different position with respect to the MRI. Moreover, the information acquired is 3D but conventionally visualized as 2D slices to the surgeon. Thus it is important to show the tumour in an anatomically accurate area during surgery. This creates the requirement for an AR display that performs this operation while not hindering the surgeon's process. An AR device provides a mixed reality interface to the user with virtual objects embedded in the real world. In contrast, virtual reality devices close off the real world to the user such that only virtual information is visible. Ideally, such a display of our requirements should be head-mounted, must contain sufficient computational power, thus bringing us to the Microsoft HoloLens. Figure 1.2 shows the capability of the HoloLens as used for educational purposes, especially in terms of understanding the human anatomy [2].

1.1 Augmented Reality and its Domain

Augmented Reality (AR) is a technology to create a reality-based interface that is moving into a staggering number of industries and consumers [60]. Since the late 90s, this technology has been its own distinct field of research. International workshops and symposiums

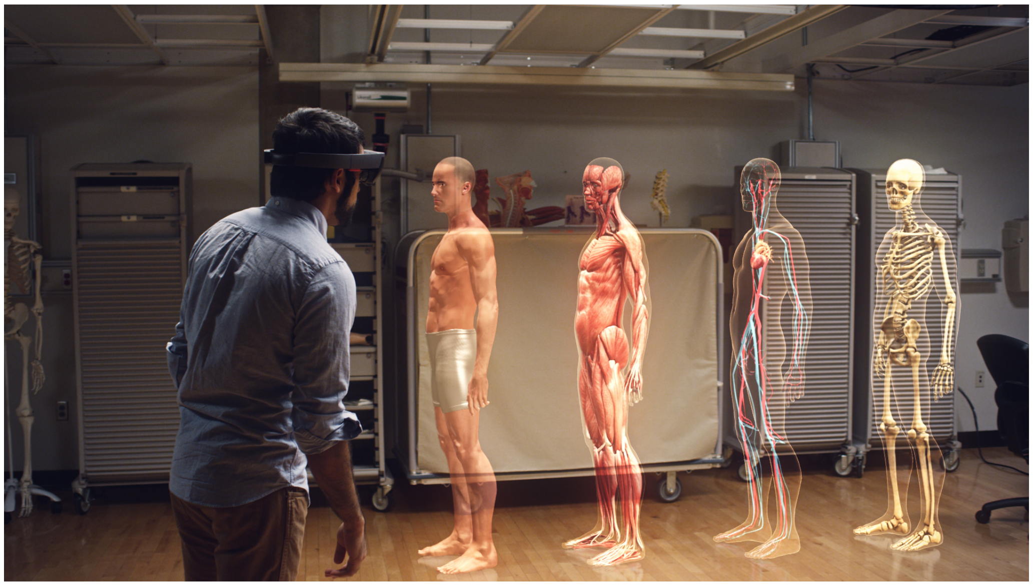


Figure 1.2: Microsoft HoloLens and its Augmentation Capability [2]

for mixed reality since then have steadily popped up in the technological scene [60]. According to Azuma and Ronald T. [23], in their 1997 survey on AR, any device employing the same must be capable of the following:

1. Combines virtual world with real world
2. The mixed reality must be capable of interaction in real-time
3. Virtual world is registered to the real world in 3 Dimensions

The capability of an AR device to blend the real world with the virtual makes its application domain vast and varied, overlaying areas such as gaming, education, mechanical assembly and medical science. Narzt et al. bases their paper on 'Augmented Reality Navigation Systems' under the basic principle that any AR equipment must be seamlessly integrated into the user's natural environment [50]. The paper talks about the utilization of a system that uses AR for navigation by considering a car as the AR equipment in itself. There has been a considerable use of AR in the education domain as well, from conducting trial studies of surgery through the use of haptic feedback in medicine [64] to the study of geometry through applications such as Construct3D [39]. AR can also assist in the design, assembly and maintenance of industrial and military projects [60]. In the area of design, AR can allow instant visualization and modification for models [60]. Fiorentino et al. talk about SpaceDesign, a mixed reality workspace to assist in design work-flow through the use of semi-transparent stereo glasses that augment prototype models [34]. The applications possible for this technology in the field of medical science is large in and of itself. Kuhlemann et al. discuss the use of AR in avoiding intermittent X-ray imaging during endovascular interventions [45]. Cui et al. talks about a system for using the Microsoft HoloLens for AR in Near-Infrared Fluorescence-based surgery. There are multiple works in the field of medicine for the utilization of augmentation in the assistance of Laparoscopic surgery [35, 29, 53].

Additionally, Van et al. [60] and Azuma et al. [23] assess in their respective surveys the applications for AR in industries such as personal assistance, advertisement, touring, combat simulations, maintenance and entertainment. Thus it is safe to say that AR extends its reach to most mainstream industries today and would be subject to considerable growth and popularity in the future.

1.2 Augmented Reality and Medical Science

Augmented Reality has been in use for surgery since 1986 [42]. Given the ability to create a mixed reality, AR would be crucial in terms of improving medical procedures by providing the kind of X-ray vision, once envisioned only in comic books, to surgeons in viewing anatomical parts of the human body in need of surgery. Over the years there have been several papers written on the work conducted within this area, including the application of this technology in real-life procedures and medical training. Wearable technology that allows AR is especially significant through its nature of being hands-free, thus only necessitating the wearing of the device without even being plugged into power (the Microsoft HoloLens exhibits this characteristic).

One of the initial attempts in this field, especially involving wearable technology was by Fuchs et al. [35] through the design and initial prototyping of a system that would assist with laparoscopic surgical procedures. The aim was to provide a view into the internal spaces of a patient with only minimal incisions on the skin and body wall. The design was also intended to solve issues in laparoscopy involving the lack of depth in imaging and limited field of view. The system used a 3D laparoscope in detecting depth to generate a real-time 3-dimensional mesh. However, depth extraction was slow and registration was faulty due to poor depth calibration.

Vogt et al. in 2006 conducted a study concerning the exploration and realization of the practical benefits of in-situ visualization [61] for image-guided procedures. They proposed the term in-situ visualization as the display of anatomical structures in their original location on the patient body. The design of the system used a video-see-through approach citing the precision and reliability possible from the same rather than projection on a semi-transparent system. The system design also had a custom single camera marker tracking system whose results were used for the purpose of calibrating the system as well as tracking. The primary purpose here was to do in-situ visualization based on CT or MRI data. The registration was done using MRI compatible markers during the scanning procedure and then tracking the marker positions later when in need of augmentation. However, trials on humans were not conducted for the same but results were collected on virtual models.

An advanced laparoscopic liver surgery using AR on a human subject was conducted as reported in a paper by Conrad et al. [29] to overcome the manual updating of the alignment between the pre-operative 3D reconstruction and the moving laparoscopic image of the liver (in displacement), thus taking the control of accuracy of overlay from the hands of the operator. The evaluation of the AR system was conducted through the first reported case of a laparoscopic rescue of failed portal embolization. The registration in this aspect was done through the tracking of the laparoscopic imaging and instrument while the alignment between the 3D reconstruction and the liver was achieved through the use of 4 anatomical markers. The view generated during this procedure merged a semi-transparent view of the 3D model over the laparoscopic view and the accuracy evaluated through the correct overlay of surface tumours. The accuracy had a deviation of approximately 5 mm which was also cited as a limitation since such a deviation would be critical for optimal clinical application due to the proximity of critical structures [29].

An AR system that helps surgical excision of tumours are especially relevant. Cui et al. [30] mention the heavy dependence of surgeons on traditional techniques in identifying cancerous tissue among healthy ones, which often lead to secondary surgeries due to the incomplete identification and removal of afflicted cells. The article also mentions the added advantage by which the radioactive sulfur-colloid used to visualize cancerous areas can be avoided thus preventing harmful ionizing radiations from affecting the patient. To solve this issue a near-infrared fluorescence (NIR) based image guided surgery was considered which uses tumour-targeted agents that bind to the cancerous tissue to be injected into the patient. The radiation emitted here is detected using a silicon-based camera. Further, a HoloLens was used to create a 2D AR environment that overlays a target area with the previously mentioned NIR visualization, allowing the surgeon to use their own perception overlaid with virtual information for surgery. Image acquisition and processing for this operation was done through a custom sensor and processing circuit board. Calibration for this procedure is done through the placement of a Hologram on top of a calibration plane. Once the hologram has been perfectly aligned, further transforms are calculated with respect to the calibration plane location and the coordinate system of the HoloLens.

Kuhlemann et al. [45] faces the challenge of visualizing the position and orientation of the catheter being inserted during endovascular interventions. In this type of surgery, the surgeon in traditional techniques needs to mentally overlay the 3D vascular tree with the 2D angiographic scene. Contrast agents and X-ray are the primary means of imaging here. The paper proposes the use of the Microsoft HoloLens to display the vascular structures in the field of view by using as inputs a landmark-based surface registration of a computed tomography (CT) scan in tandem with segmentation of the vessel tree through the use of Marching Cubes algorithm. Thus, eliminating the need for an X-ray in this aspect. Electro-magnetic markers are used to stream data into the AR device at near-real-time helping in registration and position tracking. The calibrations necessary for this process are extrinsic and are landmark based.

In 1998, Sato et al. [55] proposed a method for the image guidance of breast cancer surgery using 3D ultrasound images and AR, aimed at breast conservative surgery. The reconstruction of the ultrasound images is done through the use of an optical three-dimensional position sensor. The augmentation was achieved through the superposition of the reconstructed ultrasound images with live video images of the patient breast. However, the tumour segmentation and visualization were said to be still insufficient in efficiency and reliability, with reconstruction connectivity analysis and threshold selection being needed to be conducted multiple times. The paper also mentions that a real-time volume renderer would in the future be able to alleviate the issues mentioned.

Finally, Kotranza et al. [44] mentions the use of a virtual human and a tangible interface in the context of a virtual breast exam patient to effectively simulate interpersonal scenarios thus vouching for the degree to which virtualization created by a head-mounted display can be a substitute for reality.

1.3 Moving Forward

Given the vast domain and capabilities of AR and the HoloLens in the field of medicine as mentioned in the previous sections, the device is not without its limitations. The depth perception resolution of the device is limited to make way for a system that consumes less power and the Software Development Kit (SDK) is in its infancy, with changes continuously being released. This thesis, investigates the capabilities of advanced graphical visualization for pre-surgical planning. The principal work as performed and explained forward deals with overcoming the limitations of the HoloLens such that the device fits the objective. Primarily, the work accomplished the following:

1. **Data Acquisition**: Spatial information was collected over time for a breast mimicking surface such that the same can be used for correspondence with MRI generated information. This included the discrimination between the surface in question and the rest of the real world through photometry, as well as storing the acquired information for post-processing.
2. **Surface Reconstruction and Registration**: The data collected over the acquisition stage was offloaded into a computer which also possesses the MRI information. The collected data is then used by a surface reconstruction algorithm and further by a registration method that builds a correspondence with MRI such that the tumour location in the data acquisition coordinate system can be identified.
3. **Visualization:** The processed information in the previous step is then used to locate the tumour in the observer perspective and is sent to the HoloLens for display.

1.4 Summary

This chapter details the motivation towards developing an AR framework to assist in breast conserving surgery. Sections 1.1 and 1.2 briefs regarding the domain of AR and its history with medical science, demonstrating that this area of research is in a constant state of growth and that similar work involving the use of AR for surgical assistance have been on a gradual increase. Additionally, the outline regarding the kind of problems this thesis tackles are mentioned in Section 1.3. The next chapter details information that serves as the technical background to our work, including the hardware and software capabilities of the Microsoft HoloLens, its limitations and finally the background towards our surface fitting and registration algorithms.

Chapter 2

Technical Background

This chapter, discusses the technical background that need to be considered in building a framework for surgical planning in breast-conserving surgery. The following contains basic definitions, the type of expected inputs into the system and other relevant information which forms the base of the contributions to the problem at hand.

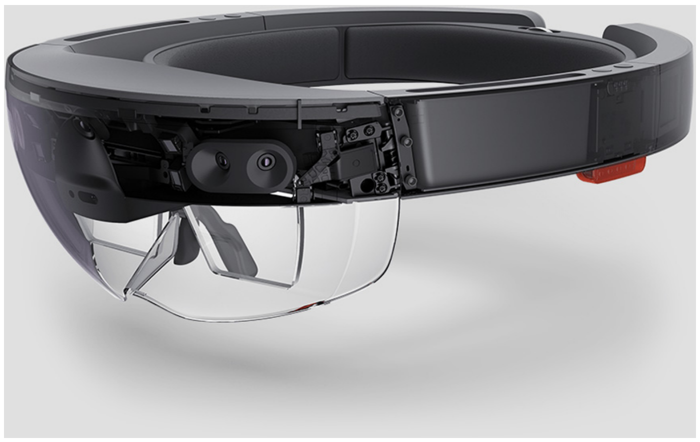
2.1 An Overview of the HoloLens Hardware

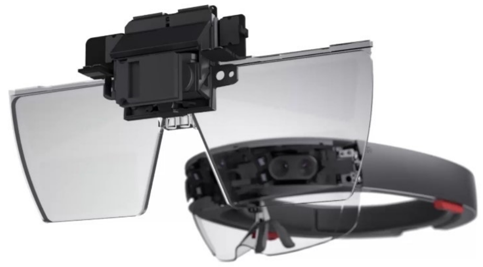
The following is a look into the hardware capabilities of the HoloLens primarily in regard to the types of sensors and optics available within the device as well as a brief look into the software development kit and tools available in leveraging its hardware.

The HoloLens has a large array of sensors that can be put to work on various tasks. Figure 2.1 displays the HoloLens and the sensor bar containing all the HoloLens sensors as well as its display module which is used to project holograms. The following subsections detail the sensors and optics in the Microsoft HoloLens as mentioned by the Microsoft Developer Website [7].

2.1.1 Sensors

Table 2.1 summarizes the built-in sensors in the HoloLens device.


(a) The Microsoft HoloLens


(b) HoloLens Display Module

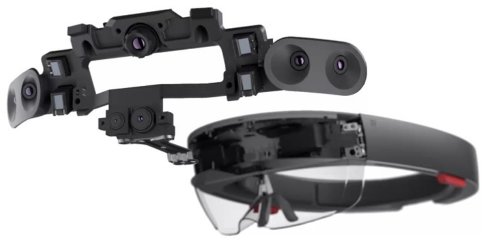

(c) HoloLens Sensor Bar

Figure 2.1: The Microsoft HoloLens and its Modules[7]

Table 2.1: HoloLens Built-In Sensors

Quantity	**Sensor Description**
1	Inertial Measurement Unit (IMU)
4	Environment Understanding Cameras
1	Depth Detection Camera
1	2 MP Photo / HD Video Camera
1	Mixed Reality Capture
4	Microphones
1	Ambient Light Sensors

2.1.2 Optics

The optics of the HoloLens includes see-through holographic lenses with two high-definition 16:9 aspect ratio, light engines. The device also contains automatic pupillary distance calibration for providing better focus as well as a holographic resolution of 2.3 M light points. The holographic density is quantified as being greater than 2.5 k radiants (light points per radian).

2.2 Basics for HoloLens Application Development

The following section explains the basics involved with programming in an application development environment for the Microsoft HoloLens, its software capabilities in leveraging its hardware as well as the types of information it handles.

2.2.1 Creating a Basic HoloLens Application

This section iterates the steps through which a basic holographic application can be created for the HoloLens.

Requirements

The basic requirements for developing applications for the HoloLens are the installation of Unity and Visual Studio. For up-to-date information refer the "Install the Tools" page within the Microsoft Developer website [8].

Creating and Building a New Project in Unity for the Microsoft HoloLens

1. Open Unity and start a new project. Provide a name for the project and set it as a 3D application and click the 'Create Project' button. The opening dialogue is as seen in Figure 2.2.

2. To install the Mixed Reality Toolkit (MRTK) go to `https://github.com/Microsoft/MixedRealityToolkit-Unity/releases/` and download the current release, unzip the same and copy the contents of the "Assets" folder into the auto-created "Assets" folder of your project.

3. Delete the default 'Main Camera' object in your project hierarchy panel. Go to "Assets/HoloToolkit/Input/Prefabs" folder (The name HoloToolkit may change in future versions of the MRTK), select the 'HoloLensCamera' prefab and drag and drop it into the hierarchy.

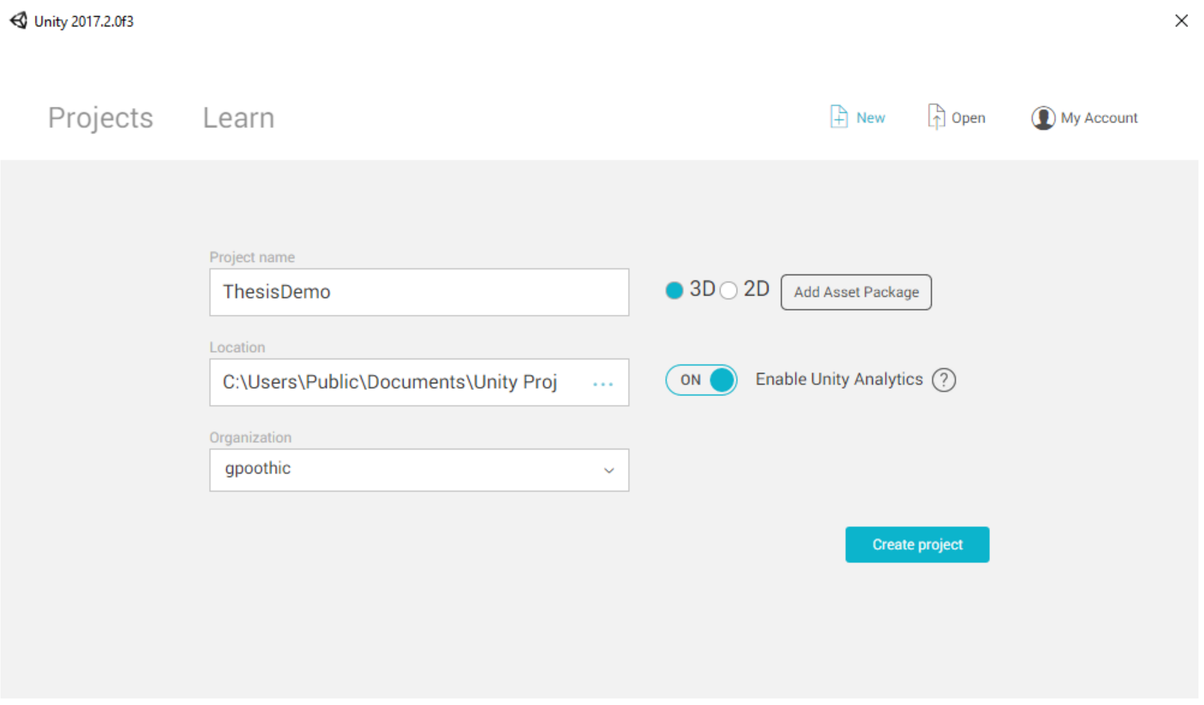
4. Create a new folder in the project "Assets" directory and name it "Scene". Go to the File menu and select 'Save Scene As' and give your current scene a name you desire.

5. Before we can build the project, slight changes must be made to the Unity development environment in terms of settings for HoloLens development purposes. From the 'Mixed Reality ToolKit' menu, select 'Apply Mixed Reality Project Settings' from the 'Configure' section as seen in Figure 2.3. This sets the current project Quality Setting to 'fastest' and build settings to Windows Store App, SDK Universal 10, Build Type D3D.

6. Select the 'Build Settings' menu from the 'File' menu. Click on 'Add Open Scenes' in the dialog box that pops up and see that the scene you previously saved shows up on the 'Scenes on Build' list in the dialog. Check the 'Unity C# project' and 'Development Build' check-boxes and finalize by clicking the 'Build' button. Create a folder in your main project directory called "App" and select the folder to start the build process. The Build dialog is as seen in Figure 2.4.

7. Once the build process is complete, open the Visual Studio (VS) solution file. In the VS toolbar, set the solution configuration to 'Release', solution platform to 'x86' and build target to 'Device' (Assuming that the HoloLens is plugged in and paired with your device). Now in the 'Debug' menu within VS, select 'Start without Debugging' or 'Start Debugging' based on your need to get the application going in your HoloLens.

Note: While working on a project we usually create a 'Scripts' folder within the project and an 'Assets' folder to store custom made scripts meant to be linked to game objects. These scripts would be built, during the building process and placed with the build solution within the sub-folders of the 'App' folder.



Project name: ThesisDemo

[✓ 3D] 2D

Location: C:\Users\Public\Documents\Unity Proj

Organization: gpoothic

ON Enable Unity Analytics

Figure 2.2: Unity 'New Project' Dialog

2.2.2 Unity Engine Environment Definitions

The following are some definitions of terms that would be encountered while working on the Unity end of the HoloLens application development environment

Materials: These are definitions of how a surface must be rendered with reference to information such as textures and colour tints

Shaders: Script files containing algorithms and mathematical calculations for finding the colour of the pixel to be rendered in an object based on the material configurations and lighting parameters

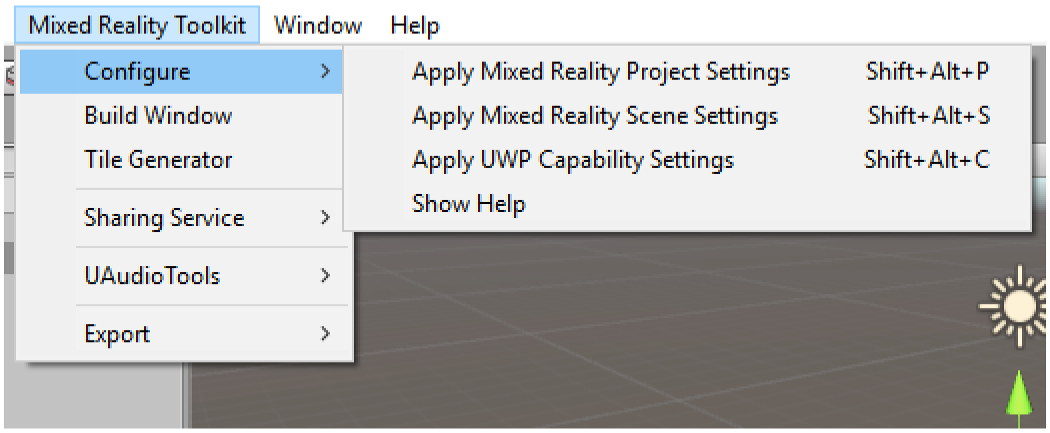
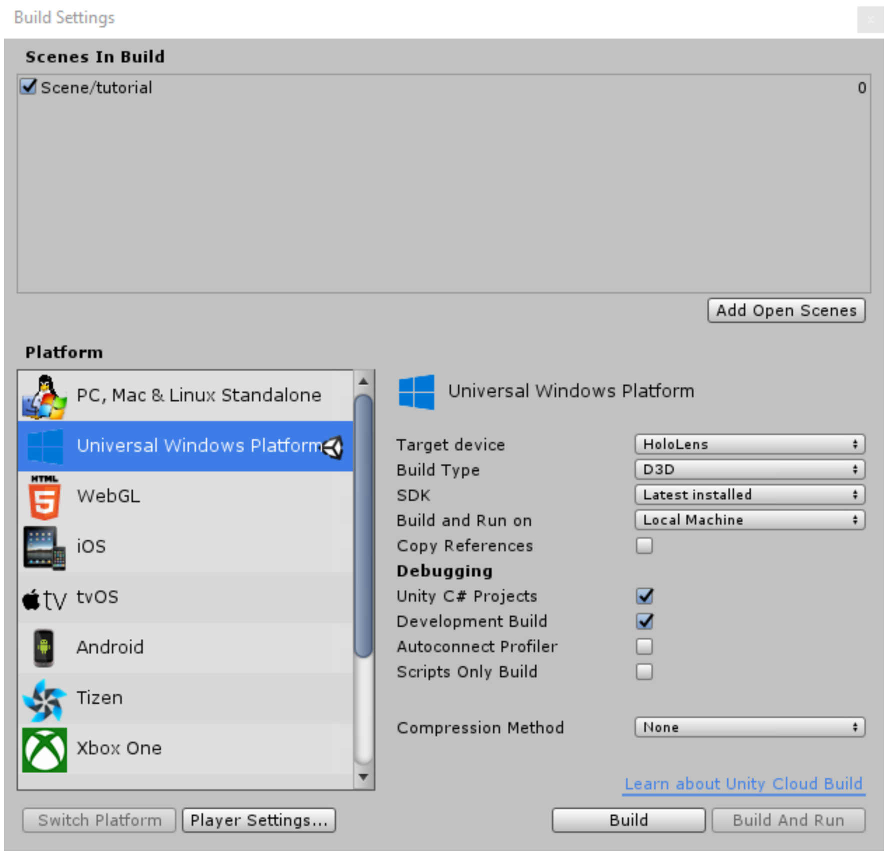

Mixed Reality Toolkit
Window
Help
Configure
Build Window
Tile Generator
Sharing Service
UAudioTools
Export
Apply Mixed Reality Project Settings Shift+Alt+P
Apply Mixed Reality Scene Settings Shift+Alt+S
Apply UWP Capability Settings Shift+Alt+C
Show Help

Figure 2.3: Applying MRTK Settings to the Current Project


Build Settings

Scenes In Build

✓ Scene/tutorial 0

Add Open Scenes

Platform

PC, Mac & Linux Standalone
Universal Windows Platform
WebGL
iOS
tvOS
Android
Tizen
Xbox One

Universal Windows Platform

Target device	HoloLens
Build Type	D3D
SDK	Latest installed
Build and Run on	Local Machine
Copy References	☐
Debugging	
Unity C# Projects	✓
Development Build	✓
Autoconnect Profiler	☐
Scripts Only Build	☐
Compression Method	None

Learn about Unity Cloud Build

Switch Platform | Player Settings... | Build | Build And Run

Figure 2.4: Unity Build Settings

Sub-shader: Every shader consists of one or more of these. Sub-shaders help in creating compatibility between multiple graphic cards with varying capability, thus making it possible to provide different options for rendering in different graphic cards. If there is only one sub-shader it is declared as generic across all graphic cards unless otherwise specified. This is also where properties such as 'ZTest' are specified which can control object transparency. The Unity Manual provides a rich documentation regarding the same[16].

Textures: Bitmap images that are referenced by a material for input to its shader in calculating an objects surface colour. Textures can also represent aspects such as reflectivity and roughness of a surface material

Ray Casting: A method available in the Unity API as well as increasingly used in the field of computer graphics. This functionality in Unity casts a ray from a given position towards a specified direction and records the journey of the ray including where it collides.

2.2.3 Object Representation in the HoloLens Environment

The representation of the topology of objects in 3D geometry for the HoloLens is discretized through the use of triangulated surfaces. This representation is referenced as a mesh and is defined as a set of inter-connected vertices/nodes which form triangle/faces. These triangles form the building blocks of what represents the topology of the object. We can represent a node N as follows:

$$N = \text{node} = (x, y, z) \in \mathbb{R}^3 \tag{2.1}$$

Here *node* represents the coordinate of each point on the mesh, $\mathbb{R}$ represents all real numbers and $\mathbb{R}^3$ represents the set of triplets of all real numbers. Given that there are L nodes, that is $\{N_i\}_{i=1=0}^{L}$, we can represent a face as a set of tuples of size three whose members are a non-repetitive subset of $\{0, 1, ..., L\}$ (which are indexes of the list of nodes). Now we can define a face as follows:

$$F = \text{face} = \{(l_1, l_2, l_3) \,|\, l_1, l_2, l_3 \in \{0, 1, ..., L\} \text{ and } l_1 \neq l_2 \neq l_3\} \tag{2.2}$$

Triangles formed based on the equation above are referred to as the faces of the representation. For example a face written down as $(1, 2, 3)$ represents the connection of vertices whose indexes are 1, 2 and 3 in the list of vertices. A triangulated representation for a surface is as seen in Figure 2.5.

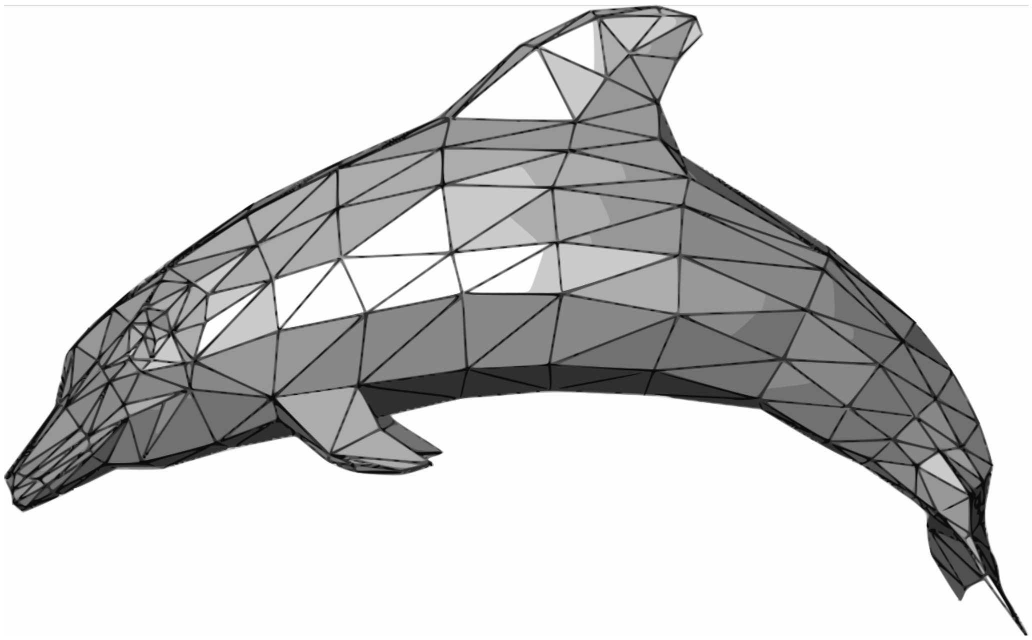


Figure 2.5: Triangulated Surface Representation [28]

2.2.4 Augmentation Basics and Real-time Mesh Generation

In terms of the end goal of the project, it is necessary to view a 3D model of a breast generated from an MRI scan in mixed reality. Even though a predefined model can be viewed in mixed reality by importing it into the Unity interface, the objective of the project demands that the model is generated during runtime based on the individual patient information.

For a model to be generated within runtime a mesh is required to be defined. This is the base form of a model that consists of nodes and faces. Figure 2.6 displays the 3D mesh generated of an office space through the use of the Microsoft HoloLens. The structures described in the image can be seen to be constructed from elementary geometrical shapes (in this case triangles). As mentioned in the previous sections a mesh structure is also the format by which we try to define the 3D model generated from the MRI.

```
Mesh mesh = new Mesh();
```

Listing 2.1: Initiates a fresh mesh object

Listing 2.1 states the code for a new mesh object in C# under the HoloLens-Unity development environment. Every script that needs to be updated over a period of time in

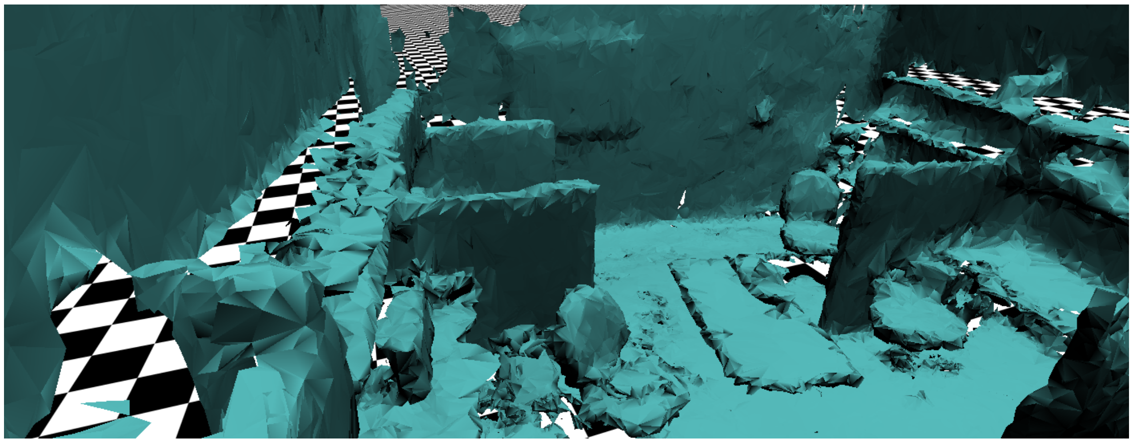


Figure 2.6: 3D Mesh of an Office Space as Scanned by the Microsoft HoloLens

the program derives from a base class called MonoBehavior which allows for methods such as the ones mentioned in Listing 2.2. These functions provide control over the activity cycle of an active object, referred to as a Game Object, in Unity. The base class Monobehavior allows the inheritance of certain variables that define the game object, such as direct references to the 'gameObject' Object and 'transform' Object. The game object connected to the mesh-generating script can be linked with the newly generated mesh. The mesh for a game object is stored within a 'Mesh Filter'. The mesh filter is inherited from the 'Component' class which is the base class for everything attached to the game object and can be referenced as mentioned in Listing 2.3. It also mentions how the newly instantiated mesh can be attached to this component.

```
void Awake() {...}
void Start() {...}
void Update() {...}
void FixedUpdate() {...}
void LateUpdate() {...}
void OnGUI() {...}
void OnDisable() {...}
void OnEnabled() {...}
```

Listing 2.2: MonoBehavior Derivatives

```
MeshFilter meshFilter = transform.GetComponent<MeshFilter>();
meshFilter.mesh = newMesh;
```

Listing 2.3: Referencing the MeshFilter and Attaching the New Mesh

The mesh component within the mesh filter object has a number of fields. The most important to specify in this case are the ‘vertices’, ‘triangles’ and ‘uv’ fields. The vertices are represented as a Vector3 structure array, triangles are an integer array and finally, uv is stored as an array of Vector2 structures.

Each element of the vertices array is a representation of a point in the world space or the space in relation to the game object associated with the mesh. Each set of 3 elements in the triangles integer array is a face, therefore each triangular face definition starts at array indices $0, 3, 6, \ldots$. The code in Listing 2.4 creates a model of a square with vertices at $a = (-1, -1, 0.1)$, $b = (1, -1, 0.1)$, $c = (1, 1, 0.1)$, $d = (-1, 1, 0.1)$. The faces for the square can be understood as (a, b, c) and (a, c, d). ‘uv’ here is a 2-dimensional vector representation of the texture co-ordinates of the mesh. Before moving further with the explanation of the application of ‘uv’, there are three key terms to be introduced in the scope of Unity [19].

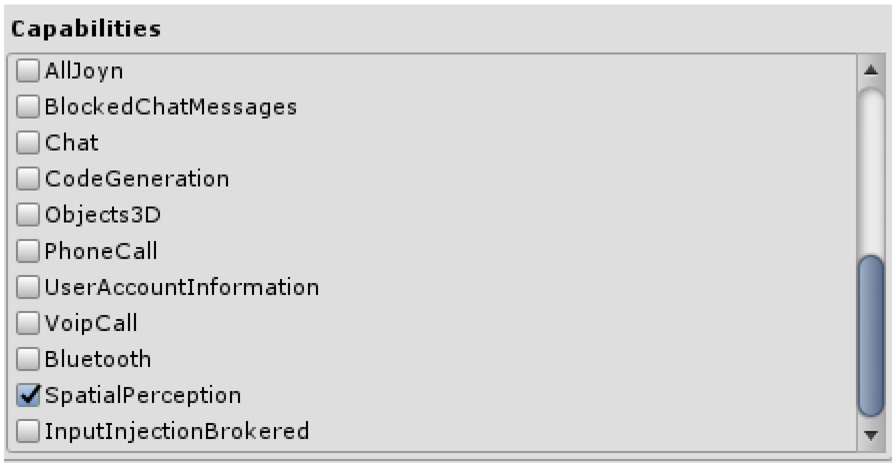
Here ‘uv’ acts as a map of each point in the mesh to a point in the corresponding texture. A texture can usually be assumed as a unit square with its upper-left corner at the coordinate $(0, 0)$ and its lower-right corner at $(1, 1)$.

```
Vector3 a = new Vector3(-1,-1,0.1f);
Vector3 b = new Vector3(1,-1,0.1f);
Vector3 c = new Vector3(1,1,0.1f);
Vector3 d = new Vector3(-1,1,0.1f);
Vector3[] vertices = new Vector3[]{a,b,c,d};
int[] triangles = new int[]{0,1,2,0,2,3};
Vector2[] uv = new Vector2[]{new Vector2(0,0), new Vector2(0,1),
   new Vector2(1,1), new Vector2(1,0)};
mesh.vertices = vertices;
mesh.triangles = triangles;
mesh.uv = uv;
```

Listing 2.4: Defining a Mesh Through Script

2.2.5 SDK and Other Software Tools

For the purpose of this project, we need the environment depth detected in real-time and then processed into a surface. The depth sensors of the device working-alongside the Software Development Kit (SDK) is known to provide a maximum spatial resolution of the range of 1500 to 2000 triangles per cubic meter. The Mixed Reality ToolKit (MRTK) is a collection of scripts and components intended to accelerate development of applications


Capabilities

- [ ] AllJoyn
- [ ] BlockedChatMessages
- [ ] Chat
- [ ] CodeGeneration
- [ ] Objects3D
- [ ] PhoneCall
- [ ] UserAccountInformation
- [ ] VoipCall
- [ ] Bluetooth
- [x] SpatialPerception
- [ ] InputInjectionBrokered

Figure 2.7: Unity Application Capabilities List

targeting Microsoft HoloLens. It provides a custom camera object to be used within Unity as well as several other prefabs (custom objects within Unity designed for specific purposes, such as the cursor or Gaze controls) and functions for application development. The toolkit is under active development and currently rolls in stable changes over the course of every 3 months.

MRTK - Spatial Mapping

The MRTK development packages contain a Spatial Mapping module for Unity3D which can be used to obtain and process the mesh data from the HoloLens. The functions contained in this module use the SurfaceObserver object that is part of the HoloLens SDK to obtain mesh information. Since this module uses a SurfaceObserver object, the Spatial Perception capability of the HoloLens (Figure 2.7) would need to be enabled. However, the Spatial Mapping output has drawbacks in terms of being coarse and non-continuous in nature.

MRTK - Spatial Understanding

The Mixed Reality Toolkit contains a better continuous meshing prefab called the 'Spatial Understanding' module. The Spatial Understanding module utilizes the mesh generated by the surface observer. However, the way the map is stored within the module through the use of the understanding DLL sets it apart. The mesh is continuous and is finer in comparison to the mesh generated by the spatial mapping module but runs a risk of oversimplifying surfaces in order to detect flatter surfaces. The understanding module has 3 stages in its existence as follows:

1. **Initiate Scan:** The module starts using the input from the Spatial Mapping Module to generate its own mesh
2. **Updating:** The module updates itself based on the previous meshes received from the Mapping module as well as the new ones.
3. **Finishing Scan:** The module finalizes the mesh by filling all holes (areas that the module did not receive enough spatial data for) in the current mesh and stops the scanning procedure. The finalized mesh would not change and would be permanent for that instance.

2.2.6 Mesh Storage Medium for Post Processing

During the course of the project, it is necessary to attain meshes generated by the HoloLens during real-time scanning and store them for later analysis or for the purpose of sending them to another device on the network in performing functions such as surface reconstruction or registration. Thus there is a necessity to store a given mesh in a logical file format that can be easily written and read by the HoloLens and external systems alike. Additionally, the format must be easily translatable to and from a 'mesh' object in Unity. It would be best if the format could be used to store the 3D object generated from an MRI. This would avoid conflicting files and unnecessary conversion from one file structure to another.

The Wavefront OBJ file format

This file format was developed by Wavefront Technologies and has been adopted by several other 3D Graphics Applications [47]. Wavefront OBJ files are a supported format in the Unity development environment that can be directly imported into virtual space as an object during development. The file can be written in ASCII or Binary format and can be used to declare multiple objects, specifying parameters such as coordinates, triangular faces, vertex normals and texture. The format is structured that each element of a model is represented as a set of key-value pairs. Every key letter specifies the information that follows in any given line.

Table 2.2: Key-Value Pairs in OBJ Files with Short Examples

OBJ File Keys and Descriptions			
Description	**Key**	**Value**	**Example**
Comment	#	-	#Comment on file format
Geometric Vertice	v	$(v_x, v_y, v_z[, w])$	v 0.123 0.456 -0.354 1.0 v 0.1 0.2 3.0
Texture Coordinates	vt	(t_u, t_v)	vt 0.2 1 vt 0.1 0.1
Vertex Normals	vn	(n_x, n_y, n_z)	vn -0.71 0.71 0
Face	f	(i, j, k)	f 1 2 3

Table 2.2 exhibits a brief introduction into the key-value pairs in an OBJ file. As seen here, the vertices are co-ordinates in the 3D plane. v_x, v_y and v_z are standard cartesian coordinates and w is an optional component for viewers that support colour and ranges from 0 to 1, with 1 being the default value. Vertex normals are represented by three coordinates, with each normal being represented as in the example section of Table 2.2. While reading the OBJ file the order in which each v, vn and vt come up are indexed respectively for their own type and are used to define the face. As seen from the example in Table 2.2 for face, each space-separated element after the key f is a connection element of the face. An example in building simple objects with OBJ files is mentioned in Appendix A.1

2.2.7 Spatial Mapping and Spatial Understanding within the Unity UI

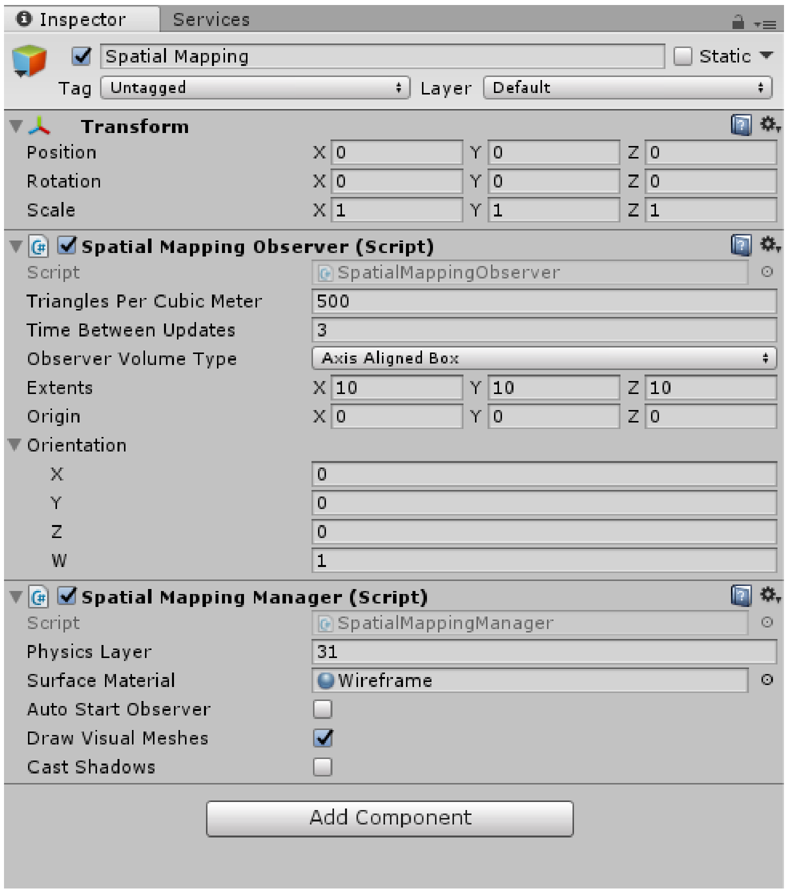
The HoloLens depth sensor does not directly provide raw depth data through their API. This is mediated by an object called the Surface Observer. Even though this object can be directly accessed to receive mesh information, the MRTK has a module that implements this object and its functionality to provide a mesh through a Spatial Mapping prefab in a very robust manner (A prefab is a prefabricated game object that can be directly imported into the Unity environment). The Spatial Mapping prefab contains two components as scripts, the Spatial Mapping Observer and the Spatial Mapping Manager as seen in Figure 2.8. There also exists a Spatial Understanding prefab in the MRTK which provides a smoother mesh based on accumulated information acquired from the Spatial Mapping prefab. The Spatial Understanding prefab consists of 3 custom components within it as scripts called the Spatial Understanding Custom Mesh, Spatial Understanding Source Mesh and Spatial Understanding.

Options in Unity UI for Spatial Prefabs

The Spatial Prefabs as mentioned in Section 2.2.7 has multiple options that could be set as their default settings in the Unity User Interface (Unity UI). These options can be leveraged to perform actions such as the automatic initialization of spatial scanning and the drawing of visual meshes.

Spatial Mapping Prefab: The Spatial Mapping Observer from the Unity UI provides options to adjust the detail of the mapping in terms of number of triangles per cubic meter (upper limit of 1500 - 2000 triangles per cubic meter), the time that can be allocated in between subsequent mesh updates, the observer volume (the volume around a specific point for which spatial data needs to be collected) and the orientation of the observer volume. The Spatial Mapping Manager within the Unity UI allows control for the kind of surface material used to construct a mesh, the physics layer of the application in which the mesh is attached to (which can be leveraged to isolate out the mesh from other objects in the game environment which would exist in different layers) and check-boxes that enable, the auto-initiation of the spatial observer, the drawing of visual meshes in the game environment and the casting of shadows by the mesh.

Spatial Understanding Prefab: The Spatial Mapping Source mesh is the script that accumulates information from the observer within the Spatial Mapping prefab. The Spatial Understanding Custom Mesh is the mesh that is built from the information acquired from the source mesh. This script provides options in the Unity UI to adjust the import period of new meshes from the source mesh, adjust the mesh material to be used to construct the understanding mesh, the maximum time per frame to be spend processing the accumulated meshes and a check-box that can be used to enable mesh-colliders such that if enabled, other objects within the game would be able to understand the custom mesh as a game object and interact with it such as for collision detection. The Spatial Understanding script controls the previously mentioned source and custom mesh scripts and provides options within the UI to automatically begin the scanning procedure for Spatial Understanding as well as fields to specify the amount of time that the Spatial Mapping Observer from the Spatial Mapping prefab needs to spend in providing updates, during the scanning process of Spatial understanding and after the scanning process is complete.



Spatial Mapping ✓ | Static ☐
Tag: Untagged | Layer: Default

Transform

	X	Y	Z
Position	0	0	0
Rotation	0	0	0
Scale	1	1	1

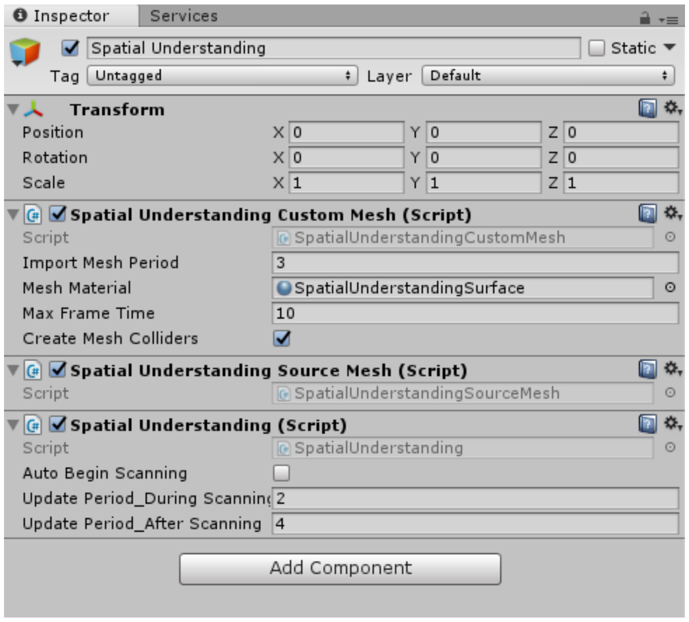
Spatial Mapping Observer (Script) ✓

Script	SpatialMappingObserver
Triangles Per Cubic Meter	500
Time Between Updates	3
Observer Volume Type	Axis Aligned Box
Extents	X 10, Y 10, Z 10
Origin	X 0, Y 0, Z 0
Orientation	
X	0
Y	0
Z	0
W	1

Spatial Mapping Manager (Script) ✓

Script	SpatialMappingManager
Physics Layer	31
Surface Material	Wireframe
Auto Start Observer	☐
Draw Visual Meshes	✓
Cast Shadows	☐

Figure 2.8: View of the Spatial Mapping Prefab in the Inspector Panel of Unity



Spatial Understanding Static
Tag Untagged Layer Default

Transform

Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Spatial Understanding Custom Mesh (Script)

Script	SpatialUnderstandingCustomMesh
Import Mesh Period	3
Mesh Material	SpatialUnderstandingSurface
Max Frame Time	10
Create Mesh Colliders	✓

Spatial Understanding Source Mesh (Script)

Script	SpatialUnderstandingSourceMesh

Spatial Understanding (Script)

Script	SpatialUnderstanding
Auto Begin Scanning	☐
Update Period_During Scanning	2
Update Period_After Scanning	4

Figure 2.9: View of the Spatial Understanding Prefab in the Inspector Panel of Unity

2.2.8 Accessing Spatial Understanding and Spatial Mapping Methods from External Scripts

Once Spatial Understanding or Spatial Mapping have been attached to Unity objects, it is possible to access information acquired by these components externally through scripts. This is possible by referencing the instantiated versions of the components in the applications. This is demonstrated in Listing 2.5. This would provide direct access to the game objects these components are attached to as well as other components of the same such as its renderer and transform.

```
SpatialMappingManager sm = SpatialMappingManager.Instance;
SpatialUnderstanding su = SpatialUnderstanding.Instance;
```

Listing 2.5: Accessing the Spatial Mapping and Spatial Understanding Instances from an external script

2.2.9 Viewing a Virtual Object within a Real-World Object

Generally, meshes or objects within the HoloLens SDK possess a property to occlude objects behind the point of view. This adds a sense of realism when mixing virtual objects with real objects. Since the surgeon must be able to view the tumour(s) within the patient's breast during the surgery it is important that a real-time meshing procedure be conducted such that we detect a surface that could be used for registration with MRI information (from which we acquire the location of the tumor(s)). It is possible, post-scanning process, to stop drawing the mesh so that it is not necessary anymore to handle the viewing of occluded objects in the scene, but this causes a problem for possible subsequent depth sensing to account for other attributes such as the movement of the breast during the process of surgery. In this scenario, we assume that a mesh generated in real-time (or a processed mesh from the real-time mesh) and MRI is registered over the patient breast. This means that the tumour is occluded from view because of a mesh surrounding its position. Figure 2.10 shows how the Spatial Understanding mesh (seen in the image as a green triangulated mesh) overlays a sphere.

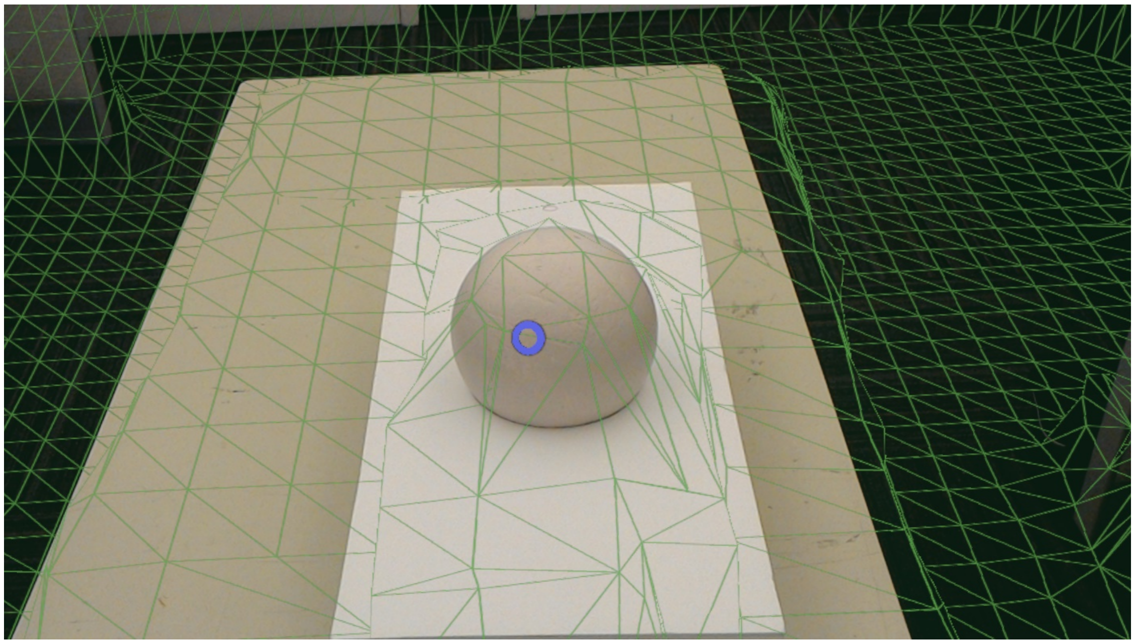


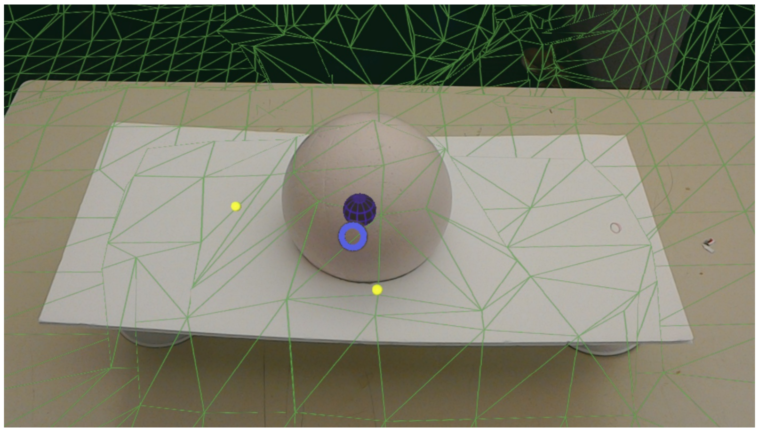
Figure 2.10: Spatial Understanding Mesh Overlaying an Object in the HoloLens view

The property of being occluded from view belongs to the material aspect of an object. Thus making an object see through would come through the modification of a materials shader script since it handles the mathematical calculations pertaining to the pixels generated for a material as mentioned in Section 2.2.4, and thus also handles how pixels are occluded as well. Every application in Unity while drawing objects in scene conduct a depth test. This ensures that only the closest surface objects are drawn within the scene. Our problem can leverage the 'ZTest' aspect of a depth test which controls how depth testing is conducted. The default value for this property is 'LEqual' which hides objects behind other objects in view.

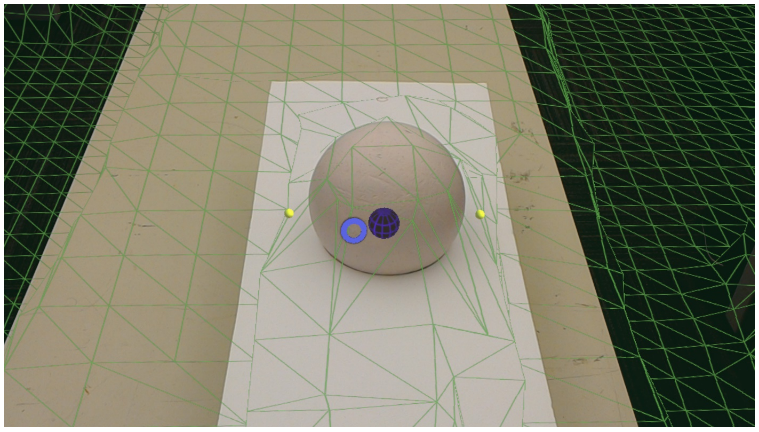
Specifying 'ZTest' to be 'Greater' allows the shader to specify that this object is visible even when occluded by another object. To also allow 'ZTest' to make this specification the property called 'ZWrite' must be 'Off'. 'ZWrite' controls if pixels for an object is written into the depth buffer for display. Turning it off allows the use of semi-transparent/transparent objects and dictates the system to ignore the object in the depth buffer and further passes the control over to 'ZTest' in drawing the object in the scene. More information regarding these properties are available in the Unity Manual [15]. Listing 2.6 demonstrates this action, attaching this to an object provides results as seen in Section 2.2.9 where the virtual sphere seen in purple appears to be inside the bigger real sphere and is not occluded by the Spatial Understanding mesh above it which surrounds the real sphere. The torus-like blue circle in the image is merely the cursor for the application.

```
Shader "Custom/OcclusionGrid"
{
  Properties
  {
    // Property specifications such as color go here
  }
  SubShader
  {
    Tags{"Queue" = "Transparent" "RenderType" = "Transparent"}
    ZWrite Off // To ignore the depth buffer and to pass power
       over to ZTest to determine if the object has to be rendered
    ZTest Greater // 'Greater' renders pixels that are occluded
  }
}
```

Listing 2.6: Manipulating the depth test within a Shader



(a) Front-view



(b) Side-view

Figure 2.11: View from the HoloLens of a Virtual Sphere within a Real Object

2.3 Limitations for the HoloLens in terms of the Project

The HoloLens limitations in regard to the project are as follows:

1. The meshes generated by the surface observer have a very limited resolution in terms of triangles per cubic meter (1500 to 2000).
2. The Microsoft HoloLens SDK does not provide access to raw depth sensor data or level of detail
3. There is no pre-implemented functionality in the HoloLens SDK that combines the sensor information from the RGB camera and depth sensor to provide an RGBD data set.
4. The HoloLens sensors require ambient lighting conditions and considerable open space to function according to its specifications. Low light and cramped spaces seem to hinder with the Spatial Understanding of the HoloLens in general, causing it to create multiple versions of the same space and store it in memory.

2.4 Basics of Surface Reconstruction

The following contains a brief overview of surface reconstruction techniques as well as important definitions and formulas that form the foundation of our proposed methodology.

2.4.1 An Overview of Surface Reconstruction

Fitting techniques are effective when we have information that is acquired from a surface through depth sensors in the form of point clouds, and we require to generate or reconstruct a surface from the same. A point cloud is defined as a set of unstructured points with no specific ordering and connection. In the context of two and three dimensions, these are represented by (x, y) and (x, y, z) coordinates respectively [46].

Liang et al. mention that surface reconstruction can be split based on either the characteristics of the data that is being used or the representation of the surface in question [46]. In terms of the input, a reconstruction can be considered easy if there is little noise and good sampling. While a reconstruction process is considered tough if the data emulates characteristics such as noise, non-uniform sampling and complicated topology.

Note: Noise means the false positives that can occur especially when it comes acquiring data from 3D Scanners or Light Detection and Ranging (LIDAR) measurements. A data set is considered well sampled if there are no holes in the information acquired or in other words, the lack of an abrupt discontinuity in the data set.

Based on the representation of surface employed by varying techniques, we can divide surface reconstruction into implicit and explicit techniques. Explicit Techniques represent surface geometry in an explicit manner. The surface is constructed through methods such as Voronoi Diagrams or Delaunay Triangulation (much like a mesh) being used to determine connections among its data points and subsequently using the same to construct a triangulated surface by connecting adjacent points [46].

Voronoi Diagrams: For a given number of points on a plane, a Voronoi diagram divides the plane based on the nearest-neighbour rule with each point being associated with the region having the highest proximity [22].

Delaunay Triangulation: This triangulation technique aims to minimize both the maximum angle and maximum circumradius of its triangles [31].

An explicit surface formulation can be very precise when using a dataset without considerable noise or holes, however, triangulated surfaces usually have problems when dealing with these parameters since the reconstruction process is based on the proximity of points from one another. A more compact and explicit representation can be found in parametric surfaces but it lacks global parametrization making the method less robust and flexible in dealing with complicated topology due to the lack of consideration of volumetric information. A global implicit surface reconstruction technique, however, that considers volumetric information would be able to adapt to non-uniform and noisy data [46].

Implicit techniques typically represent surfaces by means of a level set of a continuous scalar-valued function. The function is typically based on a grid and utilizes a level set function such as the signed distance function. There also exists grid free representations through the combination of an implicit surface which is defined through mesh-free approximation techniques such as radial basis interpolation functions [25, 25, 46].

Hoppe et al. [37] in 1992 developed an algorithm based on the determination of a zero set from a signed distance function. The proposed method was capable of inferring the topological type of the surface including the presence of boundary curves. However, the algorithm lacked in geometric accuracy. In 1994, another paper on Surface Reconstruction by Hoppe et al. [36] described a surface reconstruction procedure that provides accurate surface models for unorganized point data by determining the topological type of the surface and the presence and location of sharp features, but the paper does not contain experiments which consider sparse and non-uniform data.

Tang et al. [58] in a review and comparison paper in 2013 mention the same weakness to noise that can be acquired through real-time scanning from the techniques as proposed by Hoppe et al. [36, 37]. Additionally, it mentions that Delaunay/Voronoi based algorithms such as Alpha shape [33], Power Crust [21] and Cocone [20, 32] provide a theoretical guarantee but only under certain conditions which are tough to obtain in real-time scanning scenarios [58].

Kazhdan et al. in 2005 proposed a method called Poisson Surface Reconstruction [41] observing that the normal field of the boundary of a solid can be understood as the gradient of a surface indicator function which could be used for an implicit description of the surface we attempt to reconstruct [40]. The method described the three primary steps of reconstruction described as follows:

1. Transforming the oriented points (point normal information in this scenario) into a three-dimensional continuous vector field.

2. Finding a scalar function whose gradient would best describe the vector field.

3. Extracting the appropriate isosurface [40] from the scalar function by accessing its zero level set.

In 2013, Kazhdan et al. [40] proposed further improvements to this method to adjust for the sparse set of points, all the while including several algorithmic improvements that further reduce the time complexity of the solver used in finding the scalar function, thus enabling faster, high-quality surface reconstruction.

Liang et al. [46] in the paper titled Robust and Efficient Implicit Surface Reconstruction for Point Clouds Based on Convexified Image Segmentation, proposed a method which exploits the underlying geometric structure of the point cloud data combined with a global convexified image segmentation formulation. The method shows promise in terms of its ability in dealing with challenging point data with attributes such as complicated geometry and topology, as well as holes and non-uniformity. However, the algorithm does require prior information in the form of point normal data in exhibiting its stated accuracy.

2.4.2 Surface Fitting

Given our region of interest $\Omega_{in} \subset \mathbb{R}^3$, where Ω is the entire region under consideration such that $\Omega_{in} \in \Omega \subseteq \mathbb{R}^3$, we can describe the boundary of our region of interest as $\Gamma = \partial\Omega$. Γ is the interface between our region of interest and its surroundings. The parametric way of representing Γ is as follows:

$$\Gamma : D \subseteq \mathbb{R}^2 \to \mathbb{R}^3. \tag{2.3}$$

Here $\Gamma = (x(u,v), y(u,v), z(u,v))$, where $(u,v) \in D$. We can represent this interface based on a level set as well, that is

$$\phi : \mathbb{R}^3 \to \mathbb{R}. \tag{2.4}$$

This is defined such that Γ may be represented as follows:

$$\Gamma = \left((x,y,z) \in \mathbb{R}^3 \,|\, \phi(x,y,z) = 0\right). \tag{2.5}$$

This level set representation as mentioned above is the approach that we used to move forward. Suppose we want to fit a surface to measured data, we need to find Γ^* or equivalently as we follow the level set method, ϕ^* so that it agrees with our data and our prior beliefs. Usually, we define a functional E, which is a scalar valued function of ϕ, such that ϕ^* is obtained as a result of the minimization of functional E. That is,

$$\phi^* = \arg\min_{\phi} E(\phi), \tag{2.6}$$

where ϕ^* is a minimizer of E. Typically E has two parts. E_{ext} measures how well our solution agrees with our input data, while E_{int} represents our prior beliefs of how the zero level set of the function ϕ must look like. For example, if E_{int} represents the surface area, it is provided as an incentive to return smooth level sets. We can represent this set of combined energies as:

$$E(\phi) = E_{ext}(\phi) + E_{int}(\phi). \tag{2.7}$$

According to the Euler-Lagrange equation [63], the minimizer for E has to satisfy the following:

$$\left.\frac{\delta E(\phi)}{\delta \phi}\right|_{\phi=\phi^*} = 0. \tag{2.8}$$

Here $\frac{\delta E(\phi)}{\delta \phi}$ is the first variational derivative of $E(\phi)$. Given infinitesimal perturbation $\delta\phi$ of ϕ, the first variation can be written as:

$$\delta E(\phi) = E(\phi + \epsilon \cdot \delta\phi) - E(\phi), \tag{2.9}$$

where $0 < \epsilon \ll 1$. Based on the above, we can write:

$$\begin{aligned} \delta E(\phi) &= \left.\frac{dE(\phi + \epsilon \cdot \delta\phi)}{d\epsilon}\right|_{\epsilon=0} \\ &= \int \frac{\delta E(\phi(r)}{\delta\phi(r)} \cdot \delta\phi(r) dr, \end{aligned} \tag{2.10}$$

where $r \in \mathbb{R}^3$. Solving the Euler Lagrange condition normally is a difficult process, thus instead, we start with an initial guess ϕ_0 and then we update ϕ in the direction of a local decrease in $E(\phi)$ which is indicated by the value of $-\frac{\delta E(\phi)}{\delta\phi}$. This brings us to the following initial value problem (IVP):

$$\frac{\partial(\phi(r,t))}{\partial t} = \frac{\delta E(\phi)}{\delta\phi},\ \forall (r,t) \in \Omega \times [0, \infty), \tag{2.11}$$

$$\phi(r, 0) = \phi_0,\ \forall r \in \Omega,\ t = 0, \tag{2.12}$$

where t is not a physical quantity but rather a representation of algorithmic time. For example, if $E_{ext} = 0$ and E_{in} considers a minimum surface area:

$$\frac{\partial\phi}{\partial t} = \text{div}\left(\frac{\nabla\phi}{\|\nabla\phi\|}\right) \cdot \|\nabla\phi\|, \tag{2.13}$$

where this is done for all r and is referred to as Mean Curvature Flow (MCF) and attempts to reduce the mean curvature of the surface.

2.5 Summary

Through the sections within this chapter, we looked into the hardware capabilities of the Microsoft HoloLens and subsequently the fundamentals behind developing an AR applicationin the Microsoft HoloLens. This involved descriptions and definitions towards object representation in AR, mesh storage mediums for the device and the various components of the SDK that can be leveraged to acquire spatial information. We also discuss the limitations associated with the device for this project as mentioned in Section 2.3. Additionally, the chapter also provides an overview into the literature behind surface reconstruction and the theoretical background necessary in performing this process. The next chapter provides the principal contributions associated with the thesis, including the proposed methodology in solving our problem and the various components we have contributed in achieving this goal.

Chapter 3

Main Contributions

The previous chapter provided the technical background in developing a framework to assist in breast-conserving surgery. This chapter discusses our contributions which extends in a number of directions. We first propose a methodology in Section 3.1 termed as ISR for achieving our final goal which includes overcoming the hardware limitations of the HoloLens and completing our intended framework. To actualize our proposed methodology, we created multiple components as would be described in detail within subsequent sections.

3.1 Proposed Methodology

During the surgical procedure, the breast needs to be scanned in real-time so that a surface may be generated to be registered with the information acquired from the MRI (The registration procedure is required such that we can translate the tumor information as defined in the MRI based surface representation space to the coordinate space that the HoloLens is using for our application). However, the HoloLens, as mentioned in Section 2.3, lacks depth resolution. The figure details a method we propose called "Incremental Super Resolution (ISR)" which aims at compensating for this limitation of the Microsoft HoloLens. Sections 3.1.1 to 3.1.3 details the three stages that are continuously cycled throughout the ISR process. Figure 3.1 shows a diagram that that visually represents this methodology.

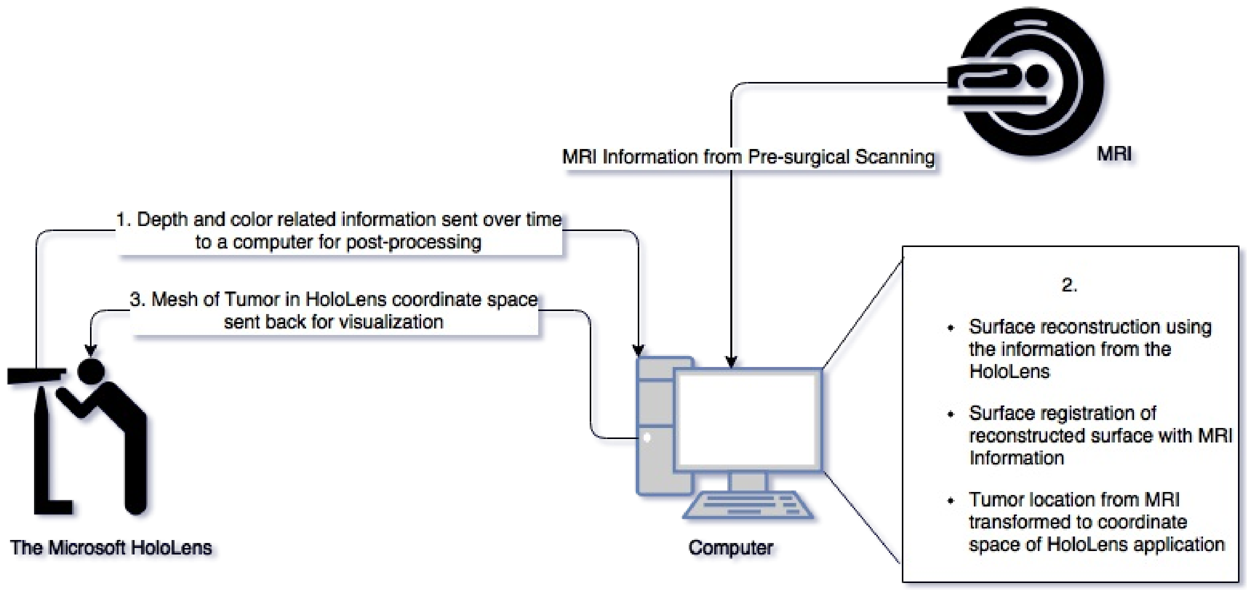

MRI Information from Pre-surgical Scanning
MRI
1. Depth and color related information sent over time to a computer for post-processing
3. Mesh of Tumor in HoloLens coordinate space sent back for visualization
2.
Surface reconstruction using the information from the HoloLens
Surface registration of reconstructed surface with MRI Information
Tumor location from MRI transformed to coordinate space of HoloLens application
The Microsoft HoloLens
Computer

Figure 3.1: Diagram Describing Proposed Methodology

3.1.1 Data Acquisition

The surface alignment procedure needs two meshes. One that would be provided through the MRI scans (and transformation procedure) and the other needs to be generated real-time during the viewing procedure. A depth sensor needs to be used to create a virtual surface of the breast for this process. The higher the quality of meshing offered, the better the matching that would happen during the surface registration phase of the project. Once a satisfying mesh has been created in terms of the real-world coordinates, this needs to be converted into the same mesh representation format as mentioned in Equations (2.1) and (2.2). The data acquisition phase consists of this collection of depth information from the HoloLens depth sensors over time. The vertex information of the mesh generated within the area being scanned is continuously collected over a period of time, paired with colour information from the Locatable Camera of the HoloLens and is sent over to a secondary device (such as a computer) for post-processing.

3.1.2 Post-Processing

The secondary device acquires the information flowing in from the HoloLens. The device is also in possession of the MRI data that is processed using finite element techniques such that we have a surface representing the topology of the breast in the supine position. The data which is continuously acquired is then cleaned using colour information to isolate the object under consideration (in this case the breast anatomy).

The continuous inflow of points enables us to use a surface fitting/reconstruction algorithm to generate a surface. This surface is then used to register with MRI information such that we get a transform which could be used to move the MRI surface into the HoloLens application space where the surgery is happening. Given this transform and the location of the tumours with respect to the MRI, we can use the former in acquiring the latter's coordinates in surgical space. The surface generated from the inflow of information from the HoloLens keeps improving over time based on the increasing amount of data being acquired, thus the transform creates a much more accurate localization of the tumour over a period of time.

3.1.3 Visualization

Through the process of surface registration, we acquire a transform function T that maps the nodes in MRI space $nodes_{MRI}$ to the surgical space coordinates $nodes_{surgical}$. This transformation can then be used to map the position of the tumour from MRI space to $surgical$ space. The mapped position of the tumour is then sent over to the HoloLens for visualization such that it appears in the anatomically accurate position inside the patient's breast.

3.2 Acquiring Depth and Color Information

The HoloLens does not provide direct access to the depth information it acquires through its sensors but provides access to processed mesh information. The following section details our contributions in acquiring data from the device including the access to depth information and creating a correspondence between depth information and colour.

3.2.1 Creating an Environment for File Saving in the HoloLens

As mentioned earlier, a mesh object stores mesh information within the HoloLens SDK. This means that when we want to store a mesh in view, the information would be processed as a mesh object. In the case of selectively saving attributes of a mesh, this object could be converted into an array or list of Vector3 objects.

While developing applications for the Microsoft HoloLens, it is important to note that not all 'System' and 'Windows' namespaces are available while in the Unity Editor environment but could subsequently be available in the build code. Thus we need to use #if directives while importing from certain namespaces such that the Unity Editor version of the code and the build code would be error free. When a C# compiler meets an #if followed eventually by an #endif, the code in-between is only considered if the specified symbol is defined. Listing 3.1 is an example for such a use case where the namespace mentioned within the #if and #endif are only imported if the environment where the script exists is currently not in the Unity Editor.

```
#if !UNITY_EDITOR
        using Windows.Storage;
#endif
```

Listing 3.1: An example for using #if

First, we need a file write system that uses an asynchronous process to do the saving process of the file. An asynchronous process unlike its counter-part works in parallel to other processes and does not hold up the system. This is important in the context of the HoloLens as it works in a frame-by-frame form. Any delay in the saving process can freeze the Holograms in view and create an irregular experience for the user. Typically, an asynchronous process is called through the use of 'System.Threading.Tasks' namespace. However, using the same without specifying to not consider the same within the Unity Editor environment would throw an error such as:

error CS1644: Feature 'asynchronous functions' cannot be used because it is not part of the C# 4.0 language specification

The best way to circumvent this situation is to specify to the Unity Solution Builder to ignore the code in the Unity Editor environment until we further build and deploy into the HoloLens from VS. The Mixed Reality Toolkit contains a method called 'MeshSaver' that implements a few lines of code based on this idea for saving information into a file during runtime in the HoloLens. The creation of a class that performs this operation with code segregation is mentioned in Appendix A.2.1. The file saving function is then given a string representation of the mesh object such that it could be saved for post-processing (Refer Appendices A.2.1 and A.2.3 for more code related details).

3.2.2 Obtaining Color Information and Acquiring Color of Vertices in a Mesh

When working out the problem of obtaining the mesh for an object from the depth sensors in the HoloLens, it is necessary to filter out unnecessary information or reduce noise in terms of other objects in visibility or inaccuracies in mesh formation. During the process of scanning meshes would be generated for everything in view or atleast for everything concerning the mesh filter contained in the observer volume. This would mean that a general reduction in observer volume would still encompass the mesh data concerning to a minimum span of 1 meter in all three coordinate spaces. To combat this problem we can filter out the vertices based on colour correlations. Thus it is necessary to use an RGBD (Red Green Blue + Depth) module which in theory would combine, the colour data that the device observes through its 'locatable camera' and the spatial information it receives from its depth sensors. Unfortunately, neither the HoloLens SDK nor the MRTK has a component that enables this process thus creating the need for solving this problem in the HoloLens.

Obtaining an Image from the HoloLens

The first part of solving this problem is the acquisition of an image during the process of scanning. This image can then be used to map the colour data to the mesh data we acquire in the same instance. The Locatable Camera in the HoloLens needs to be enabled for attaining this goal. This camera is mounted on the front of the device and enables applications to see what the user sees in real-time. For using the same in an application, one must first enable this capability in the capabilities section inside the player settings of Unity. The camera not only provides the colour information but also passes information concerning its position with respect to the scene by providing a transform. The processes necessary for photo capture are neatly wrapped into the HoloLens SDK with the method running asynchronously alongside other functions in the application. The C# code related aspects of performing this operation are discussed in Appendix A.3

Mapping Mesh Coordinates to Color

The camera that captures RGB information for the HoloLens and the meshes generated in AR exist in two different coordinate systems which can be termed as 'camera space' and 'world space'. The 'PhotoCapture' module provides a transformation matrix to bridge this difference, and in addition, provides a projection matrix in locating the pixel in an image that corresponds to any point in camera space. The following details the working of both matrices and an algorithm in obtaining the colour of any point in world space which is captured in an image.

Camera to World Matrix: The camera to world matrix mentioned in the previous section is a transformation matrix. Consider a point that is represented as (x, y, z) in the camera space, this is represented as a matrix

$$C = \begin{bmatrix} x & y & z & 1 \end{bmatrix}. \tag{3.1}$$

If the world space coordinates of C is represented as

$$W = \begin{bmatrix} a & b & c & 1 \end{bmatrix}, \tag{3.2}$$

for coordinates (a, b, c). A transformation matrix that works in 3 dimensions is a 4×4 matrix of the form

$$T_{CamToWorld} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & 0 \\ m_{21} & m_{22} & m_{23} & 0 \\ m_{31} & m_{32} & m_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.3}$$

The relation between the camera space, the world space and the transformation matrix can be written as

$$C \times T_{CamToWorld} = W. \tag{3.4}$$

Given this information we can also write,

$$C \times T_{CamToWorld} \times T^{-1}_{CamToWorld} = W \times T^{-1}_{CamToWorld}, \tag{3.5}$$

which on further simplification becomes

$$C \times I = W \times T^{-1}_{CamToWorld}. \tag{3.6}$$

Here I is the identity matrix

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.7}$$

Thus, we can write

$$C = W \times T^{-1}_{CamToWorld}, \tag{3.8}$$

which means that the inverse of the transformation matrix can be used to map coordinates from world space to camera space as well. If the inverse of $T_{CamToWorld}$ matrix can be referred to as $T_{WorldToCam}$[1], we can also write:

$$W \times T_{WorldToCam} = C. \tag{3.9}$$

Projection Matrix: The projection matrix is a 4×4 matrix that helps in transforming coordinates from the camera coordinate space to image space, that is, the transformation is from a 3D space to a 2D space. The matrix consists of parameters such as the focal length, skew and center of projection of the camera to assist in its transformational function. Multiplication of a matrix consisting of a camera coordinate (a, b, c) such as $C = \begin{bmatrix} a & b & c & 1 \end{bmatrix}$ yields a result $P = \begin{bmatrix} x & y & 1 \end{bmatrix}$ such that x and y are the pixel coordinates represented between a range $[-1, 1]$. If w and h represent the width and height of the image, then we can gain the actual pixel location P_{actual} from the matrix $P_{minimal} = \begin{bmatrix} x & y \end{bmatrix}$ by the following:

$$P_{actual} = \frac{1}{2} P_{minimal} + \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}. \tag{3.10}$$

Creating a World Space to Pixel Position Function: Given the information above, it is now possible to design a function that would be able to take as input a three dimensional coordinate and provide as output the pixel position associated with this point. Appendix A.4 describes the implementation for such a method in the Unity-C# HoloLens development environment.

3.2.3 Collecting Mesh Information via the HoloLens SDK and MRTK

Considering that the HoloLens SDK does offer a base class for accessing meshes, the Spatial Mapping and Spatial Understanding classes from the MRTK are well built and more robust in their approach of acquiring and keeping mesh information. Thus, accessing depth information from these modules is more reliable. The following is a description of the optimal method of achieving this goal.

[1]The transformations obtained from the Locatable Camera are affine in nature. The inverse of matrices representing these transformations are possible as long as the matrix determinant is non-zero. An example of a situation where inverting the transformation matrix would throw an error would be when attempting to invert a matrix that scales by 0. However this would necessarily not occur when mapping a point from world space to camera space given the knowledge that the point in question would be present in both spaces

Acquiring Mesh Filters directly and Drawbacks

In Section 2.2.8 we mentioned how to access the Spatial Understanding and Mapping modules externally from a script. Now we intend to access the mesh created in these instances in our script so that we may use them to identify and colour or even directly store them in OBJ files. The direct approach in accessing either mesh is through initiating a call from the respective instances for their mesh filters as follows.

```
List<MeshFilter> SMF =
   SpatialMappingManager.Instance.GetMeshFilters();
List<MeshFilter> SUF =
   SpatialUnderstanding.Instance.UnderstandingCustomMesh
   .GetMeshFilters();
```

Though this method is perfectly reasonable in its approach for the Spatial Understanding instance, it is unwise to use the same for Spatial Mapping. The Understanding Custom Mesh generated by the module is aligned in world space and can be directly sent to storage or processing based on the same belief. However, the Mapping Manager filters are not stored in relation to one another, that is, their inner meshes are stored in a different coordinate space and does not reflect their view of the world space.

Accessing Meshes via Surface Objects

It is first important to understand that the list of mesh filters stored by both the mapping and understanding components are based on a system of surface objects. Thus, the call for mesh filters in the above function retrieves the meshes from a ‘SurfaceObject’ variable within each instance. To clarify, in this context a ‘SurfaceObject’ is a struct defined as follows in the MRTK.

```
public struct SurfaceObject
{
  public int ID;
  public GameObject Object;
  public MeshRenderer Renderer;
  public MeshFilter Filter;
  public MeshCollider Collider;
}
```

This structure is defined in the ‘SpatialMappingSource’ class and we can use the ‘SurfaceObject’ structure to map out the mesh based on its location in world space coordinates.

For this, we acquire the surface objects as follows:

```
SpatialMappingManager.Instance.GetSurfaceObjects(); // For the
   Spatial Mapping mesh
SpatialUnderstanding.Instance.UnderstandingCustomMesh
   .SurfaceObjects; // For the spatial understanding mesh
```

Acquiring meshes this way would help create a method that would work for both modules and is consistent with the results we desire. Note that the Spatial Mapping Manager has a function that returns the surface objects, while the understanding mesh returns the same using a public variable. Both calls return a

$$ReadOnlyCollection < SpatialMappingSource.SurfaceObject >$$

type that contains the surface objects on the scene and additionally is a construct of the results obtained from the 'SurfaceObserver' base class of the HoloLens SDK. This is a list variable that stores the specific structure as mentioned in the angle brackets. The world space coordinates are sometimes not available in the mesh filters (as these coordinates could also be stored in relation to the position of the parent game object). However, the game objects that these components are attached to are produced in the real world. As such, every game object possesses a 'Transform' component consisting of a 'TransformPoint' method which can take a point stored within it and transform it into the space that the said object is occupying. Thus we can take each vertice of a mesh in each surface object and apply the transform component to convert the vertice into a world space coordinate. For any given surface object so, if $soTransform$ is its transform component and $soFilter$ its mesh filter, we can acquire the same and implement this idea as follows:

```
MeshFilter soFilter = so.Filter;
Transform soTransform = so.Object.transform;
List<Vector3> worldVertice = new List<Vector3>();
Vector3[] tvertices = soFilter.mesh.vertices;
for (int i = 0; i < tvertices.Length; i++)
{
  Vector3 temp = soTransform.TransformPoint(tvertices[i]);
  worldVertice.Add(temp);
}
```

Listing 3.2: Applying transforms to coordinates in bringing them into the Application's World Space

Here $worldVertice$ is a list consisting of all the vertices in the surface object in world space coordinates. Since the list that stores the faces of a mesh are just the order of the vertices that are connected to one another, we do not need to do any transformation while importing these values from the surface objects. However, in the case of normals, a transformation has to be done to adjust the direction such that it is compatible with the new space the vertices of the mesh occupy. The 'Transform' component has a 'TransformVector' method to do just this action and can be used for any given normal vector expressed through the use of a 'Vector3' object. If $vNormal$ is a 'Vector3' object that represents the normal of a vertice in a given surface object with variables the same as in Listing 3.2, then we can transform them as follows:

```
Vector3 worldNormal = soTransform.TransformVector(vNormal);
```

Z-Filtering

During the process of collecting vertice information, the surface observer returns meshes throughout the area of observation irrespective of whether they are visible during the sampling step or not. This requires an additional layer of filtering to avoid taking in vertices that are blocked from view especially by the object under observation itself. For example, in Figure 2.10, we can see the mesh overlay on the sphere while there exist meshes behind the field of view which represents the sphere and would be acquired through the data acquisition steps we mentioned earlier. Thus this requires filtering based on the field of view and can be implemented as follows:

1. For each sample, while capturing the image, acquire the camera position
2. Do a ray cast from the camera position to each vertex and record the position of collision on the mesh
3. If the position of collision on the mesh in view, matches with the vertex to which the ray cast was intended, the vertex is safe to record, else, drop the vertex

An implementation of the Z-filtering method is mentioned in Appendix A.5

3.2.4 RGBD Module and Incremental Super Resolution

It is now possible to combine the knowledge we have discussed in Sections 3.2.1 to 3.2.3 to create a module that collects RGB color information (Red, Green, Blue), combine it with the mesh information to save for post-processing and complete the first phase of the ISR technique.

The RGBD Module

It is best to find the corresponding color based on the depth as the resolution of the depth sensor is lower than the resolution of the locatable camera, that is, the camera collects more points of information in the form of pixels than the depth camera can collect in terms of the scene that is in the field of view.

Program Flow: The program flow of a module as discussed above can first be summarized in the following steps:

1. Select a point around which spatial information needs to be acquired and surround it with a bounding volume of required size and shape
2. Collect spatial information within the observer volume in terms of a complete mesh that overlays the volume or just selective information (must contain vertice information) that occurs within it
3. Simultaneously initiate the photo capturing module to obtain an image of the scene in view using the RGB module discussed in Section 3.2.2
4. Pass the vertices obtained from the spatial information capture into the RGB module to obtain colour information
5. Store mesh information as an OBJ file along with colour information in a different or similar file based on requirements

Step 1 involves the selection of a point for which we need to create an observer volume around which we intend to collect the spatial information. Both the Spatial Mapping and the Understanding mesh is rooted in their depth information with the 'Spatial Mapping Observer' script which is an interface of the 'SurfaceObserver' base class. In addition, the script as mentioned in Section 2.2.7 provides options to specify an observer volume. The observer volume requires as input a shape and its size. There are several shapes that can be considered by the observer including boxes and spheres additionally for which we have to provide parameters such as orientation and size. Considering a box-shaped observer volume with its length, breadth and height specified as float values l, b and h respectively with its center c specified in 3-dimensional coordinates as a 'Vector3' object, we can request an observer volume for spatial observation as follows (The code also additionally specifies an integer parameter $TrianglesPerCubicMeter$, as the resolution of depth scanning in the observer volume, this value is inconsequential above the approximate range of 1500 to 2000):

```
SpatialMappingObserver SMO =
    SpatialMappingManager.Instance.SurfaceObserver;
SMO.TrianglesPerCubicMeter = TrianglesPerCubicMeter;
SMO.Origin = c;
SMO.ObserverVolumeType = ObserverVolumeTypes.AxisAlignedBox;
SMO.Extents = new Vector3(l, b, h);
```

Once the observer volume is in place, the spatial information we receive would only be regarding the surface objects that pass through this volume or contained in it. Step 2 is the acquiring of spatial information as specified in Section 3.2.3. The mesh filters that are obtained can then be filtered for the characteristic information we are seeking. Steps 3 and 4 involve using this vertice data to collect colour information as mentioned in Section 3.2.2, which is finally stored as an OBJ file. The colour information can be stored in an OBJ file as well with colour information substituted as vertice coordinates such that an OBJ file reader would read the same without a hitch.

Incremental Super Resolution

The HoloLens updates its mesh over a period of time that can be specified. Since raw-access to depth information is not allowed and only mesh information can be received the identification of the surface of an object is easier if the vertice information is collected over a period of time. The RGBD data for a specific volume covering the area of interest can be collected multiple times. This would result in a large set of points for which a majority of noise can be filtered out by using the colour information that we obtained. This filtered set of vertices can be used as input into surface fitting algorithms to generate a surface as mentioned in Section 3.3. Experimentation regarding this technique using a sphere as the model, the subsequent surface reconstruction and its registration with a simulated point cloud of a perfect sphere are discussed in Chapter 4.

Note: The Spatial Mapping mesh is the best option for this method. Even though this mesh is coarser than the Spatial Understanding mesh, the Understanding mesh is a product of the Spatial Mapping mesh and in addition focuses on detecting flatter surfaces such as tables, chairs, floors, walls and ceilings at this point. This would mean that curved surfaces such as a breast would run the risk of being oversimplified. Thus it is better to access the rawest form of information we can acquire, especially one that does not take shape priorities into context.

3.2.5 Pitfalls while Developing for the HoloLens

The HoloLens only allocates a finite amount of dynamically allocatable memory for its applications. Thus, any allocation that exceeds this amount would result in the application being killed from the outside. There are other pitfalls such as implementing the system such that a function runs in parallel to the frame rendering done by the HoloLens. In the context of this framework, following a set of precautionary measures can result in the smoother performance of the application.

Use Coroutines

While running a function it is natural that the frame of Holograms would be held in place until processing completes. For certain applications, especially in the context of the intentions of our data acquisition technique, a function that stores vertices from a mesh, correlates them with color and saves them to a file is extremely heavy on processing in the context of $\frac{1}{60}^{\text{th}}$ of a second (This assumes that our application is running at 60 frames per second).

Given the above problem, we would need a method to divide the processing of a function across frames. A coroutine is a function that has the ability to pause execution and return control to Unity but then to continue where it left off in the following frame [14]. In the context of C#, a coroutine is executed as follows:

```
IEnumerator DoStuff()
{
  for( ... ) // Executes some time consuming operation
  {
    ...
    yield return null;
  }
}
```

Here, when the coroutine needs to return control back to Unity, it stops at the line containing the 'yield' command until Unity returns control back to the coroutine, after which the method continues from where it left off. If necessary, it is also possible to provide a time delay in restarting execution by doing the following:

```
IEnumerator DoStuff()
{
  for( ... ) // Executes some time consuming operation
  {
    ...
    yield return new WaitForSeconds(2.3f);
  }
}
```

The above code makes the coroutine wait for 2.3 seconds before it continues its operation. The value 2.3 can be substituted with a measure of time, based on the requirements of the application. In context, the application we implemented for performing ISR waits 4 seconds per every sample[2]. This time would also be useful if the user considers moving from their original position for observing the volume under scanning from a different angle or position.

Memory Bottleneck

The limited memory available for dynamic allocation in the HoloLens can result sometimes in the application being killed once it exceeds this threshold. The following are a few ways this could happen, especially in the context of our application and possible solutions and precautions that can be taken to fix the same.

Instantiating Meshes: Instantiation is where an object is cloned completely including its inner components. More information regarding Instantiation is available within the Unity Scripting API documentation [17]. While working on an application that performs the ISR technique, it is possible to instantiate meshes so that they may be processed outside the Spatial Mapping Object. The utilization of the mesh in a coroutine is one such situation which might necessitate this usage. In the context of our application, the spatial observer is instructed to update itself every two seconds. Considering that we would be analyzing the mesh in parallel to the surface observers action it is possible that the original mesh object might change as mesh processing is happening. Continuous instantiation without the proper disposal of objects after processing would create a scarcity in memory and consequentially kills the application.

[2]This time delay is such that the user can move around to observe the surface from a different point of view. Changing the users point of observation during scanning can result in perks such as the inflow of fresh mesh information based on areas that may have not been in the field of view earlier

Using Textures: While using colours for filtering noise in the context of ISR, we would run into using Textures in storing the image we collect in parallel such that the world coordinates of said points can be mapped back into an image. Considering the kind of resolution we intend to capture the image in, textures would consume a large amount of memory per image which, at times, is greater than the objects that store the mesh information and thus proper handling of this variable is a necessity.

Defensive Measures Against Memory Bottlenecks: The following are a few measures one can adopt in avoiding issues concerning memory especially in the context of creating a system implementing ISR

1. Delete all instantiated objects and free all variables by setting them to null values.
2. Manually initiate the Garbage collector after a set number of samples are collected. This is possible through the system command "GC.Collect()" (More information can be found in the Microsoft Developer Network [6]). An example of running the garbage collector every 5 iterations is as follows:

```
using System;
...
for(int i = 0; i < TotalSamples; i++)
{
  ...
  if(i % 5 == 0)
  {
    GC.Collect();
  }
  ...
}
```

3. Consider limiting the observer volume and filtering vertices within the span of a single frame. Limiting the observer volume would result in a smaller mesh to analyze, thus making it possible to run the filtering action in a normal function rather than a coroutine.
4. Dispose textures after their use by doing the following:

```
...
Texture2D.Destroy(texture_variable);
...
```

3.3 Surface Reconstruction

The HoloLens even with the use of the ISR technique only yields about 1000 to 1500 points around the region of interest(30 cm × 30 cm × 30 cm box). After noise filtration through colour, this reduces to about 300 to 600 points. This poses a challenge for surface matching techniques as it is difficult to correlate the orientation and position of the scanned breast with respect to the 3D model generated from MRI. The specifications of the HoloLens additionally mentions that the upper limit of resolution for a mesh generated via the devices depth sensors is approximately 1500 triangles per cubic meter. The situation calls for the application of surface fitting techniques which provide a much smoother distribution of a point cloud for future matching.

3.3.1 Proposed Technique for Surface Reconstruction

There are multiple types of fitting techniques available for application in problems dealing with surface reconstruction as mentioned in Section 2.4.1. However, our data set is sparse, noisy and has a high chance of containing holes resulting in leaving out explicit surface representations. While there are several implicit methods that could be considered, our problem deals with extracting out the surface of a breast during real-time sampling. Thus rather than delving into generic solutions based on implicit representations, it would be best to formulate a method that would fit our problem. Thus, we propose a method where a functional representing a signed distance function is optimized through a total variation solver and gradient descent in creating a surface by accessing its zero level set.

In Section 2.4.2, we introduced a level set representation ϕ, which models the boundary of our region of interest which in this case is our breast mound. Given a set of K points, $\{r_i\}_{i=1}^{K}$ delivered from the HoloLens, we need a level set function that adheres to this data. Since the breast mound is smooth, it can be reasonably assumed that the mean curvature for the same is relatively small. As mentioned earlier, we thus need to minimize a functional $E(\phi)$, where its components E_{int} penalizes the smoothness of the zero level set of ϕ, while E_{ext} shows the adherence of the zero level set to our input data. It is seen to be useful to define ϕ to be a Signed Distance Function (SDF) viz,

$$\phi(r) = \inf \|r - r'\| \cdot \begin{cases} 1 & \text{if } r \in \Omega_{in} \\ -1 & \text{if } r \in \Omega \setminus \Omega_{in} \end{cases}, \forall r \in \Omega, \tag{3.11}$$

where r' is the set of points that lie on the boundary of our region of interest (the zero level set), $\partial\Omega$. Being a SDF, such ϕ should obey the Eikonal equation where

$$\begin{cases} \|\nabla\phi(r)\| = 1, \ \forall r \in \Omega, \\ \phi(r) = 0, \ \forall r \in \Gamma = \partial\Omega. \end{cases} \tag{3.12}$$

. Here ∇ denotes the gradient of ϕ at r. We now simplify Equation (2.13) as,

$$\frac{\delta E_{int}}{\delta\phi} = -\text{div}\left(\frac{\nabla\phi}{\|\nabla\phi\|}\right). \tag{3.13}$$

At the same time, we write E_{ext} as an expression of mean distance, to ensure a measure of adherence to our data, as follows:

$$E_{ext}(\phi) = \frac{1}{K}\sum_{i=1}^{K}|\phi(r_i)|. \tag{3.14}$$

Thus when we minimize the combined energy as mentioned in Equation (2.7) based on the components mentioned above, the process pushes the surface as close to the data as possible while ensuring that the surface we get as output is smooth.

To derive the first variational derivative of the external energy component, we do as follows:

$$\begin{aligned} \delta E_{ext} &= \left.\frac{dE_{ext}(\phi + \epsilon\cdot\delta\phi)}{\delta\epsilon}\right|_{\epsilon=0} \\ &= \frac{1}{K}\sum_{i=1}^{K}\left.\frac{d\left(|\phi(r_i) + \epsilon\cdot\delta\phi(r_i)|\right)}{d\epsilon}\right|_{\epsilon=0}. \end{aligned} \tag{3.15}$$

For simplicity, we define the derivative of the absolute value function $(|a|)'$ as $\text{sign}(a)$, which is true $\forall a \neq 0$. This is justified since $|a|$ can be approximated as

$$|a| = \lim_{\tau\to\infty}\frac{1}{\pi}\tan^{-1}(\tau a) \tag{3.16}$$

where the right hand side in the equation is 0 when $a = 0$. So from a numerical perspective, the assumption we made for the derivative of the absolute function as the sign of the value is quite acceptable.

Now continuing from Equation (3.15),

$$\begin{aligned} \delta E_{ext}(\phi) &= \frac{1}{K}\sum_{i=1}^{K} \text{sign}\left(\phi(r_i) + \epsilon \cdot \delta\phi(r_i)\right) \cdot \delta\phi(r_i) \bigg|_{\epsilon=0} \\ &= \frac{1}{K}\sum_{i=1}^{K} \text{sign}\left(\phi(r_i)\right) \cdot \delta\phi(r_i). \end{aligned} \tag{3.17}$$

Let D denote the Dirac delta function, for which one of its defining attributes is its sifting property. Specifically, given a continuous function $f(\tau)$, we have,

$$f(\tau) = \int f(\tau')D(\tau - \tau')d\tau'. \tag{3.18}$$

Using the sifting property, we derive the first variation for E_{ext} as

$$\begin{aligned} \delta E_{ext}(\phi) &= \frac{1}{K}\sum_{i=1}^{K} \int_{r\in\Omega} \text{sign}(\phi(r)) \cdot \delta\phi(r) \cdot D(r - r_i)\, dr \\ &= \int_{r\in\Omega} \left[\text{sign}(\phi) \frac{1}{K}\sum_{i=1}^{K}(r - r_i) \right] \cdot \delta\phi(r)\, dr. \end{aligned} \tag{3.19}$$

From this, we have $\frac{\delta E_{ext}}{\delta \phi}$ as,

$$\frac{\delta E_{ext}}{\delta \phi} = \text{sign}(\phi) \cdot \left[\frac{1}{K}\sum_{i=1}^{K} D(r - r_i) \right]. \tag{3.20}$$

In practical computation however, using the Dirac delta function is impossible, so we replace the same with an approximation as follows [27]:

$$D(r) \approx \frac{1}{(2\pi\sigma^2)^{3/2}} e^{-\frac{1}{2}\frac{\|r\|^2}{\sigma^2}} = D_\sigma(r). \tag{3.21}$$

Note that,

$$D_\sigma(r) \xrightarrow[\sigma\to 0]{} D(r). \tag{3.22}$$

Discretization

Given the above definitions, we now proceed to the discretization of the total gradient flow which now has the form of:

$$\frac{\partial \phi(r,t)}{\partial t} = \operatorname{div}\left(\frac{\nabla\phi}{\|\nabla\phi\|}\right) - \lambda\left[\frac{1}{K}\sum_{i=1}^{K} D_\sigma(r-r_i)\right]\cdot \operatorname{sign}(\phi), \tag{3.23}$$

where λ acts as the regularization parameter for the equation. For the sake of notational clarity, we term the component $\left[\frac{1}{K}\sum_{i=1}^{K} D_\sigma(r-r_i)\right]$ as $g(r)$ and re-write the equation as:

$$\frac{\partial \phi}{\partial t} = \operatorname{div}\left(\frac{\nabla\phi}{\|\nabla\phi\|}\right) - \lambda g\phi, \tag{3.24}$$

We now apply a semi-implicit discretization scheme as follows [62]:

$$\frac{\phi_{t+\Delta t}-\phi_t}{\Delta t} \approx \operatorname{div}\left(\frac{\nabla\phi_{t+\Delta t}}{\|\nabla\phi_{t+\Delta t}\|}\right) - \lambda g\cdot\operatorname{sign}(\phi_t). \tag{3.25}$$

Note that, here, t is not a physical unit of time, but a representation for an algorithmic step. Since semi-implicit scheme is more stable and less sensitive to a change in t, it further allows for larger discretization steps in time. Now we can write this equation as:

$$\phi_{t+\Delta t} - \Delta t\cdot\operatorname{div}\left(\frac{\nabla\phi_{t+\Delta t}}{\|\nabla\phi_{t+\Delta t}\|}\right) = \phi_t - (\Delta t\lambda)g\cdot\operatorname{sign}(\phi_t). \tag{3.26}$$

The left-hand side of the above equation can be expressed in terms of the action of an operator $\mathcal{A}$. Consequently, we have:

$$\mathcal{A}\{\phi_{t+\Delta t}\} = f, \tag{3.27}$$

where

$$f = \phi_t - (\Delta t\lambda)g\cdot\operatorname{sign}(\phi_t). \tag{3.28}$$

is available from t.

It is known that this equation amounts to the Euler-Lagrange optimality condition for the functional [54]:

$$E(\phi) = \frac{1}{2}\int |\phi(r)-f(r)|^2\,dr + \Delta t\cdot\int \|\nabla\phi(r)\|dr, \tag{3.29}$$

where the last term amounts to the total variation semi-norm of ϕ. Thus solving Equation (3.27) is equivalent to finding the minimizer of $E(\phi)$ which is characterized by the Euler-Lagrange condition,

$$\left.\frac{\delta E(\phi)}{\delta \phi}\right|_{\phi=\phi_{t+\Delta t}} = 0. \tag{3.30}$$

Additionally, since $E(\phi)$ is strictly convex, such a minimizer would be unique and can be found by a range of optimization algorithms such as PGM (Proximal Gradient Method), which in this case is also known as Chambole method [26]. So symbolically, the solution to Equation (3.27) is given by:

$$\phi_{t+\Delta t} = \mathrm{PGM}(f) = \mathrm{PGM}(\phi_t - \Delta t\, \lambda g\, \mathrm{sign}(\phi_t)). \tag{3.31}$$

Our Surface Fitting Algorithm

Now, we can proceed to build our algorithm as follows:

```
Data: $\{r_i\}_{i=1}^K$, An initial SDF guess $\phi_0$, $\lambda > 0$, $\Delta t > 0$, $\sigma > 0$, $0 < \beta \leq 1$, Iter $> 0$
Result: Minimizer of $E$, $\phi^*$
i = 0;
while $i$ less than Iter do
    $g(r) = \frac{1}{K}\sum_{j=1}^K D_\sigma(. - r_i)$;
    $f = \phi - (\Delta t \lambda) \cdot g \cdot \mathrm{sign}(\phi)$;
    $\phi = \mathrm{PGM}(f)$;
    $\phi = Redist(\phi)$;
    $\sigma = \beta\sigma$;
    $i = i + 1$;
end
```

Algorithm 1: Our Surface Fitting Algorithm

The 'Redist' operation mentioned above is a method termed as redistancing by which we make sure that the continuously evolving function within our algorithm remains a signed distance function [57, 56]. The β parameter modifies the σ in every iteration such that the delta function subsequently becomes tighter.

3.4 Surface Registration

In registering the surface that we acquire through the process as described in Section 3.3 with the actual surface information of the patient we acquire from MRI, we use an intensity-based automatic registration technique. Consider that the fitting process as mentioned in the previous section yields an interface Γ_{HL}. Additionally, we possess an interface Γ_{MRI} from the MRI data. Our objective here is to register Γ_{HL} to Γ_{MRI}.

Given a set of affine transforms $T : \mathbb{R}^3 \rightarrow \mathbb{R}^3$, our objective is to find an optimal T that could be applied to the MRI interface such that it minimizes the distance between both interfaces, i.e.,

$$\min_T \left[\text{dist}(\Gamma_{MRI} \circ T, \Gamma_{HL})\right]. \tag{3.32}$$

Before specifying the distance measure, we specify the following Gaussian approximation of the Dirac delta function as follows:

$$D_\sigma(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{x^2}{\sigma^2}}. \tag{3.33}$$

Indeed, the registration can alternatively be done between the SDFs ϕ_{MRI} and ϕ_{HL} corresponding to Γ_{MRI} and Γ_{HL}, respectively. Such an approach would allow us to use relatively simple and convenient distances such as $D_\sigma(\phi_{MRI}(r))$ and $D_\sigma(\phi_{HL}(r))$. $D_\sigma(\phi(r))$ in essence forms a fuzzy indicator of the zero level set of the SDF. Here σ is a scalar, positive value dictating the thickness of intensity around the surface, with a larger value corresponding to a higher thickness. Thus, our problem becomes the minimization of the following based on an affine transform T:

$$\min_T \int |D_\sigma(\phi_{HL}(T(r))) - D_\sigma(\phi_{MRI}(r))|^2 \, dr. \tag{3.34}$$

The calculated value of the Dirac Delta function from the corresponding ϕ can be assumed as an intensity based representation of the surface. Thus, it is now possible to use a registration technique built for this purpose. We use an intensity-based automatic registration technique for which more information can be found in the MathWorks documentation[4, 9]

Once the registration technique provides an acceptable T, the same transform can be used in locating the tumour location within the breast. This method however is under the assumption that a transform that registers based on surface information can be used to extrapolate the motion of the interiors as well. The surgeon can then proceed to place physical markers that identify these locations with precision and proceed to surgery.

3.5 Summary

The chapter details the proposed methodology in realizing the AR framework that would assist in breast-conserving surgery. The sections mentioned above also detail the contributions we have made towards the same. This in terms of data acquisition involves the collection of depth information from a specific area, merging data collected from the HoloLens depth sensors with the color information acquired from the device's Locatable Camera. In the side of surface reconstruction, we suggest algorithms that are capable of reconstructing the observed surface from the data points collected through the HoloLens (given that this data represents a sparse point cloud that is convex in nature). Finally, we suggest a method to register this surface with the MRI data. The following chapter details the experimental phase of our project based on the techniques mentioned in this chapter. This is facilitated through the utilization of a hemisphere as a breast-mimicking surface.

Chapter 4

ISR Experimentation

This section details the proof of concept experiment in registering the collected data of a spherical object with simulated data and finally visualization. Since the HoloLens is particularly weak in terms of understanding curved surfaces through its depth sensor, registering a sphere through the ISR technique allows for detecting curved surface data and subsequently provides promise in assisting conservative breast cancer surgery.

4.1 Setup for Data Collection

The understanding of the lower-resolution of depth sensitivity led to the consideration of a spherical structure as the subject of experimentation. A sphere approximately 14 cm in diameter was used and was placed in the centre of a black poster paper. Based on the discussions above, the following is a list of parameter settings before scanning the sphere:

- Observer Volume Shape: Box
- Observer Volume Size: $50 \times 50 \times 50\,\text{cm}^3$ (Even though this setting would result in considering a larger surface object that overlies the area)
- Observer Volume Origin: *Can be selected at the beginning of scanning - Usually, a point observed by the depth sensor on the surface of the sphere*
- Vertex Filter Size: $30 \times 30 \times 30\,\text{cm}^3$
- Vertex Filter Origin: *Same as Observer Volume Origin*

- Number of Samples Collected: 50

The setting for the experiment can be seen in Figure 4.1. The uniform black color around the sphere would help in filtering out noise through the analysis of color that we attach with each point.

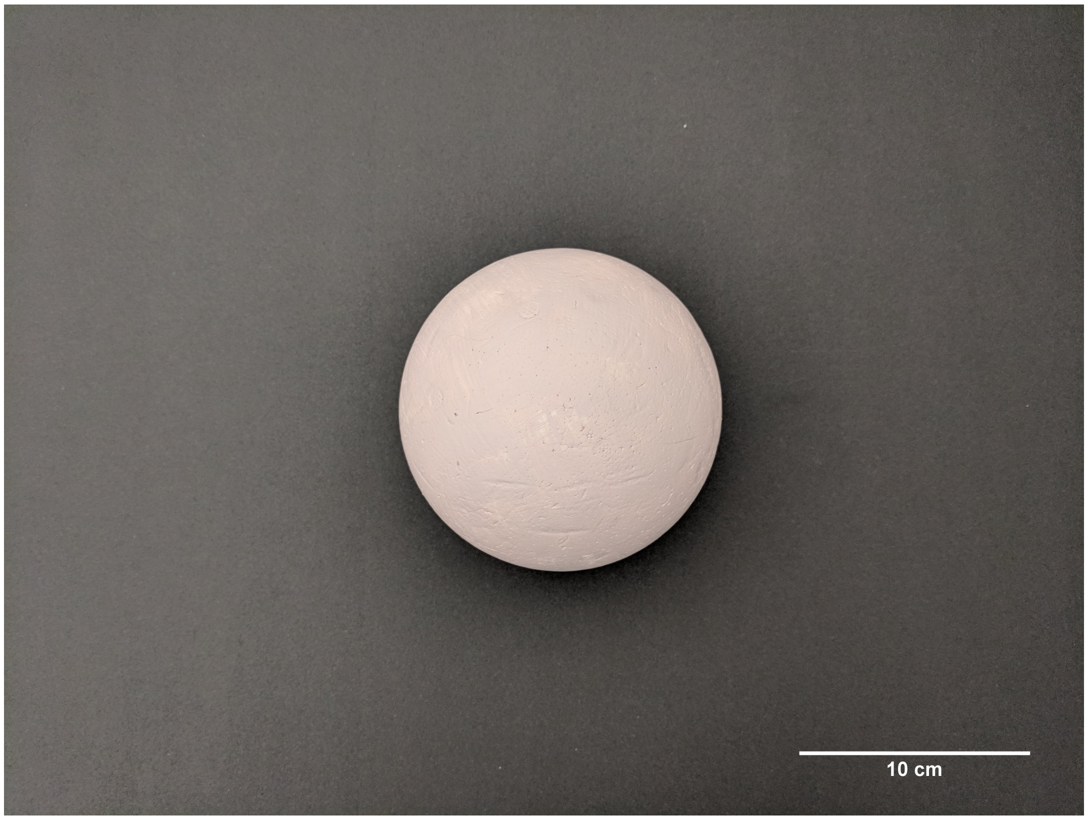


Figure 4.1: Setting used for Spatial Scanning in Conducting the ISR Experiment

4.2 Data Acquisition Program Flow

Given the setup, we collect the data for ISR based on the following algorithm:

1. Start a coroutine and initiate the process of generating the spatial mapping mesh, set the iteration counter to zero and initialize a vertex and colour list. The mesh overlay on the object during scanning is as seen in Figure 4.2.

2. While the iteration counter is less than total sample requirement, move on to the next step, else go to step 9.
3. Capture a photo and save it in a texture using 'RGBCapture' as mentioned in Section 3.2.2.
4. Capture vertex information from surface objects within world space coordinates as mentioned in Section 3.2.3.
5. Filter vertices based on the vertex filter size and origin.
6. Filter vertices based on the Z-filtering method as mentioned in Section 3.2.3.
7. Map filtered vertex data and correlate with colour information from the image captured in Step 3 as mentioned in Section 3.2.2.
8. Append captured data into appropriate lists as mentioned in Step 1 and increment the iteration counter.
9. Wait for 5 seconds using the yield functionality of coroutines as mentioned in Section 3.2.5 and then move to Step 2.
10. Store the information from the vertex and colour lists into OBJ files using the methods discussed in Section 3.2.1 and then exit the coroutine.

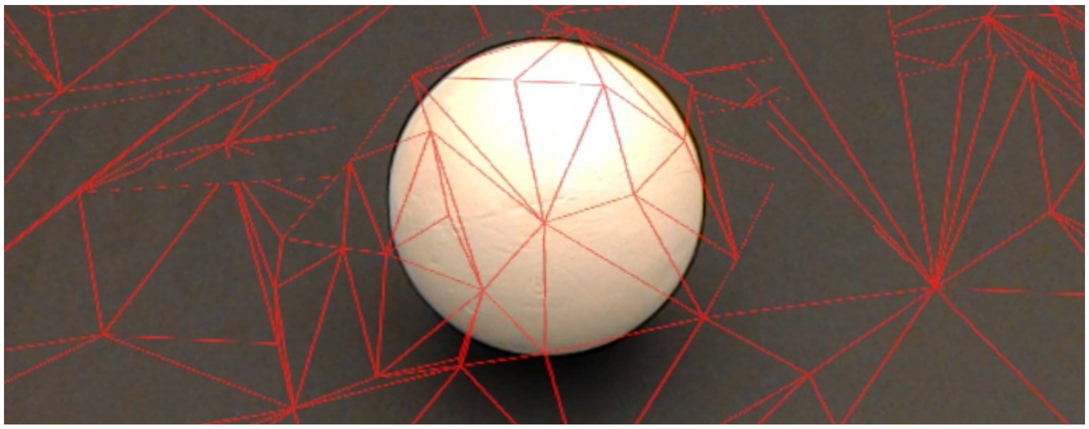


Figure 4.2: The Spatial Mapping Mesh Overlay as seen from the HoloLens During the Scanning Procedure

4.3 Filtering Noise Using Color

The data acquired is then filtered first by color. Since the surroundings of the sphere are black in color, a majority of the noise acquired from the mesh would have a corresponding color that is close to the color value of black ($(0,0,0)$). Thus a simple distance metric used on the colors can be used to filter out noise. In this case a euclidean metric is used as follows:

$$\text{dist} = \sqrt{x^2 + y^2 + z^2}. \tag{4.1}$$

Here (x, y, z) are the corresponding red, green and blue components of the color that we collect. For data collected for N points, we can define the data C as:

$$C = \{(x, y, z) | x, y, z \in [0, 1]\}. \tag{4.2}$$

Now, if each (x, y, z) in C can be represented as C_i we can define the threshold t to filter the colors as:

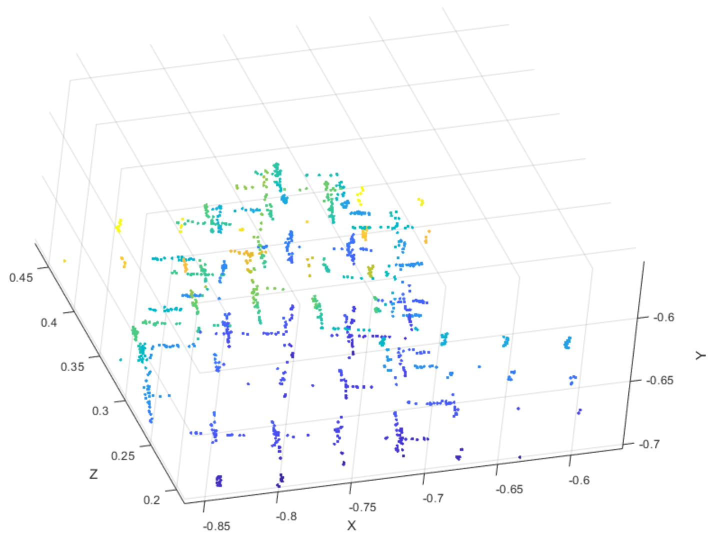
$$\text{mean}(\text{dist}(C)) = \frac{\sum_{i=1}^{N} \text{dist}(C_i)}{N}, \tag{4.3}$$

$$\text{std}(\text{dist}(C)) = \sqrt{\frac{\sum_{i=1}^{N} [\text{dist}(C_i) - \text{mean}(\text{dist}(C))]^2}{N-1}}, \tag{4.4}$$

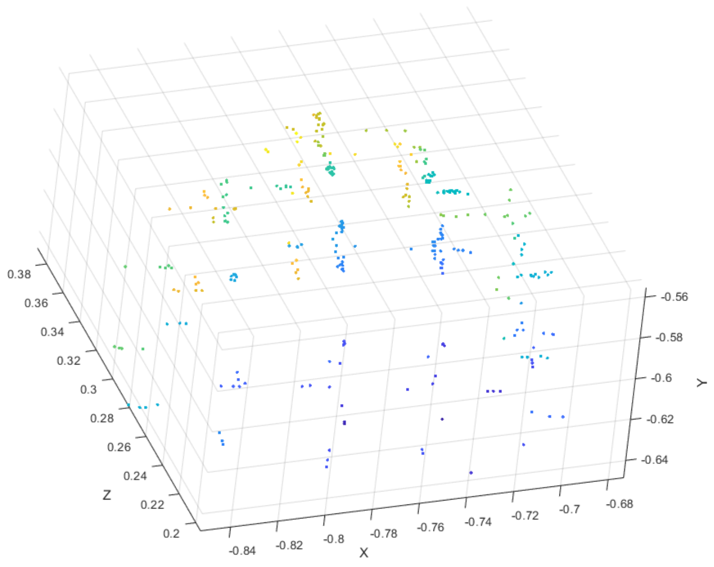
$$t = \text{mean}(\text{dist}(C)) - \text{std}(\text{dist}(C)). \tag{4.5}$$

The threshold as seen in Equation (4.5) is just the difference of the mean and standard deviation values of the distance of each color from the origin. Since the colors from the black background serves as the source of color for a majority of noise, the points associated with it would be closer to the origin and thus the threshold helps in filtering out the same. Once the colors have been filtered, we also apply cut out values from the base of the sphere so as to get a hemisphere. This can be done by filtering out a set of axis values from the center of the sphere.

The unfiltered and filtered scatter figure for the point cloud would look like Figure 4.3. The figure exhibits a point cloud of 364 points collected over 50 samples of the Spatial Mapping mesh. A single sample in contrast only contains approximately 30 viable points. The samples collected over our data acquisition technique can now be used in our surface reconstruction algorithm to generate a surface that can be used to map against another set of higher resolution data.



(a) Point Cloud Before Filtering



(b) Point Cloud After Filtering

Figure 4.3: MATLAB Plot of Points Before and After the Filtering Process Mentioned in Section 4.3 (X, Y and Z are 3D Coordinates Representative of Distance in Meters)

4.4 Surface Reconstruction - Setup and Results

We prototyped our surface reconstruction algorithm in MATLAB. In addition to the algorithm mentioned in Algorithm 1, the following steps were taken as part of initializing the process:

1. The convex hull C of the point cloud was derived using the inbuilt 'convhull' function in MATLAB as seen in Figure 4.4. Points in red are the ones selected in forming the shape of the hull.

2. A three dimensional grid G is generated of size $N \times N \times N$ based on the filtered point cloud F (where $(x, y, z) \in F$) containing equally spaced values on the x, y and z coordinates in the range of $[\min(x) - o, \max(x) + o]$, $[\min(y) - o, \max(y) + o]$ and $[\min(z) - o, \max(z) + o]$ respectively. Here $o > 0$ and is set to the equivalent of 1 cm, that is 0.01 in ensuring that a set of points remain above and below the convex hull as padding.

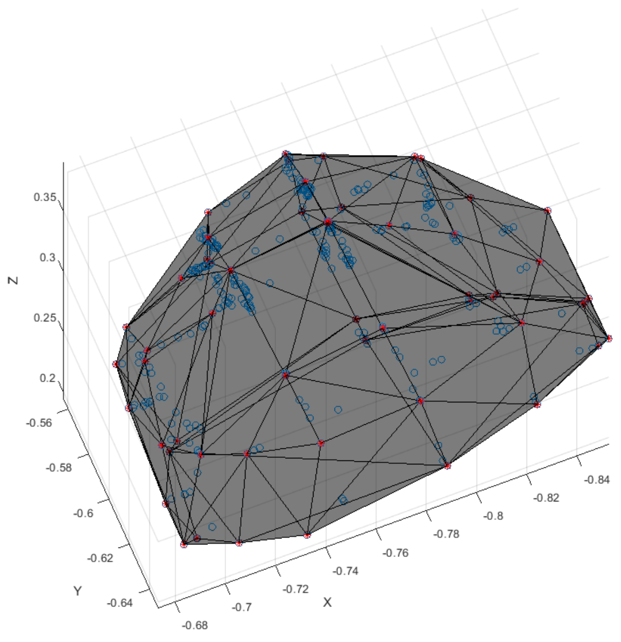


Figure 4.4: Convex Hull Created for the Filtered Point Cloud Using MATLAB (X, Y and Z are 3D Coordinates Representative of Distance in Meters)

3. A function called 'intriangulation' [43, 52] is used to identify if each point on the grid lies inside or outside the generated convex hull C, in effect creating a binary mask. This mask indicates the initial set of points in G which we would further reduce, as seen in Figure 4.5.

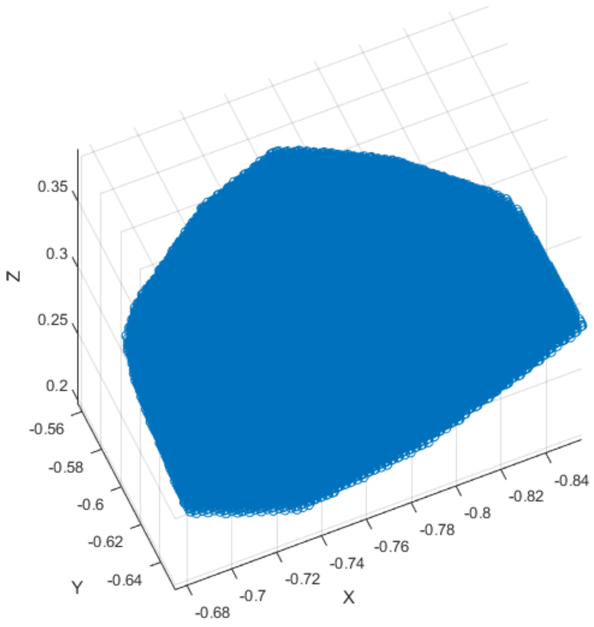


Figure 4.5: Points from the Grid Correlating to a Logical 1 Value in the Mask (X, Y and Z are 3D Coordinates Representative of Distance in Meters)

4. A function called 'bwdistsc' [49, 48] is used to calculate the three dimensional signed distance function of each point in the mask, forming the ϕ_0 component of the algorithm mentioned in Algorithm 1.

5. Subsequently the constants based on Algorithm 1 are set as $\lambda = 200$, $\tau = 10$, $T = 20$ and $\sigma = 10$, with λ, σ and τ being reduced by three percent of their own value in each iteration[1].

[1]These values were selected based on trial and error results of the algorithm, obeying parameter constraints, until a surface of satisfactory form was attained

The result of the above experiment through the access of the zero level set of the optimized function ϕ identifies the shape of our surface in G as shown in Figure 4.6.

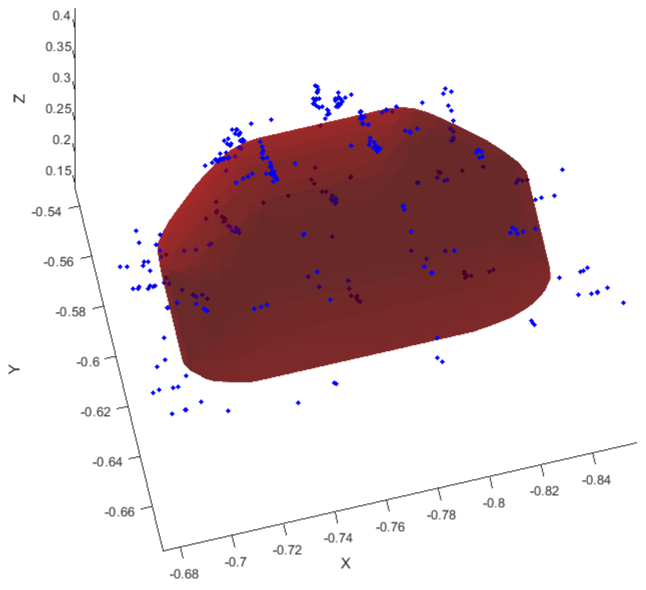


Figure 4.6: Surface G corresponding to the Zero Level Set of our Optimized ϕ with the Original Data Points Represented in Blue (X, Y and Z are 3D Coordinates Representative of Distance in Meters)

4.5 Surface Registration - Setup and Results

Using the results from Section 4.4 we can now proceed to register a simulated point cloud of a hemisphere to assess our ISR technique. The following are the steps involved in setting up the registration process:

1. Given a point cloud F in grid G_F as the result of our reconstruction experiment and

the optimized signed distance function (SDF) result ϕ_F, we generate the point cloud M and surrounding grid G_M of a hemisphere of similar size as the sphere we scanned, which in this case is 13 cm in diameter, containing 1711 points. We consider the point cloud F as the fixed point cloud and M as the moving point cloud to be registered with the position of F. The surface point cloud M forms as seen in Figure 4.7.

2. We calculate the convex hull and use the same to generate the mask of point clouds M as M_{mask}, in the same set of steps as previously mentioned in Section 4.4. The signed distance function ϕ_M is calculated for the points.

3. The intensity is calculated for all points in both point clouds M and F as I_M and I_F respectively. The intensity calculation involves the use of the combined data set of the grid and SDF (G_M, ϕ_M) and (G_F, ϕ_F) as stated in Equation (3.33).

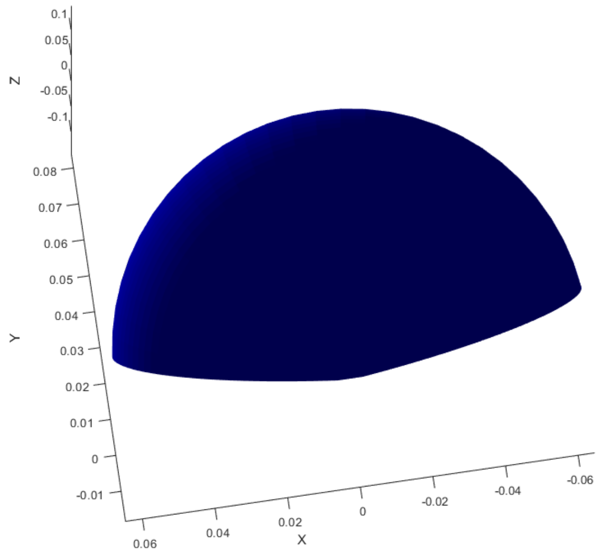


Figure 4.7: Simulated Hemisphere with Identical Size as the Scanned Object (X, Y and Z are 3D Coordinates Representative of Distance in Meters)

Given the techniques in generating the intensity, the algorithm for acquiring the transform for registration of M to F is as follows:

1. Set initial σ based on Equation (3.33) as 20.[2]

2. Create a three dimensional image to world coordinate reference for both point clouds R_F and R_M by passing in values corresponding to G_F and G_M in the MATLAB function 'imref3D' [12].

3. Retrieve optimizer(Opt) and metric(Met) configurations based on the image capture modality('unimodal' in this case as both point clouds exhibit readings from an object surface with similar brightness and contrast, post surface reconstruction) using the MATLAB function 'imregconfig' [3].

4. Set total number of iterations N as 10, initialize iteration counter n as 0.

5. When $n < N$, proceed to step 6, else exit.

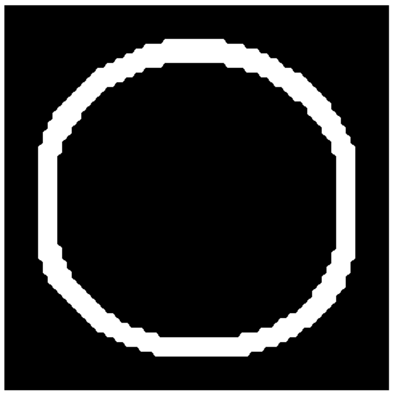
6. Calculate initial intensity for point clouds M and F as I_M and I_F.

7. If $n = 1$, then the first transform generated T_1 is found using the 'imregtform'[4] function in MATLAB, where values $[I_M, R_M, I_F, R_F, Opt, Met, \text{'affine'}]$ are passed. Else, the current transform T_n is found by using the same function by passing the values $[I_M, R_M, I_F, R_F, Opt, Met, \text{'affine'}, T_{n-1}]$. ($T_{n-1}$ is passed as an initial transform, 'affine' is passed as the type of transform we expect from the function[1]).

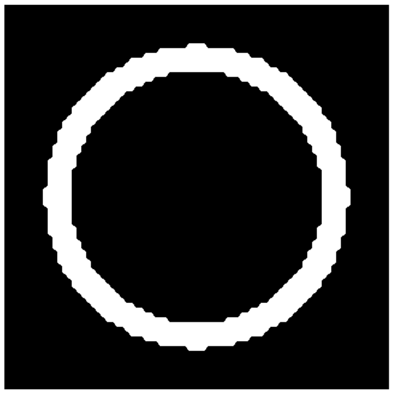
8. $\sigma = 0.8\sigma$, Go to step 5.

The cross-sectional view of the surface registration is seen in Figure 4.8. The surfaces are seen to overlap one another about their respective cross-section, providing a satisfactory registration.

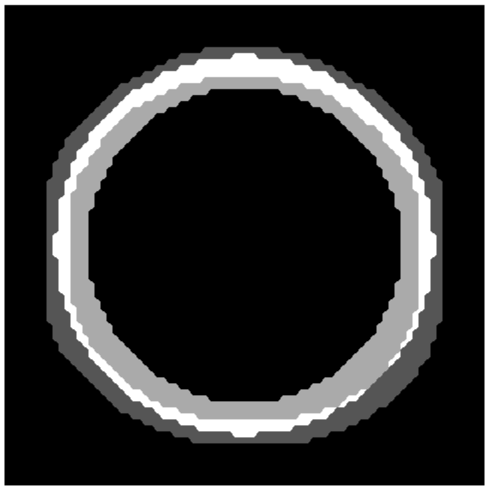
[2]This value is based on trial and error results. The higher value of σ at the start gives a rougher estimation of the surfaces that are then registered. Subsequent iterations reduce the σ (resulting in a more sharper estimate of the surface) and additionally uses the prior registration information to refine the acquired transform



(a) Cross-sectional view of Scanned Model



(b) Cross-sectional view of Simulated Model



(c) Cross-sectional view of Registered Model

Figure 4.8: Cross-Sectional View of Registered Models

4.6 Visualization

Finally, considering that the simulated surface has a sphere of 2 cm diameter within it representing a tumour, with its centre at $(0.03, 0.03, 0.03)$ as seen in Figure 4.9, we apply the transform generated from the registration procedure to the cloud in translating this into the HoloLens application space. This transformed position is then transferred to the device such that a tumour can then be visualized in the real world within the scanned sphere. The results for the same are as seen in Figure 4.10, where the tumor is seen to be accurately sitting on the upper half of the sphere we considered for reconstruction. Additionally, the sphere is also oriented towards a quadrant as is the simulated tumour model and is seen to be accurate to the specified position.

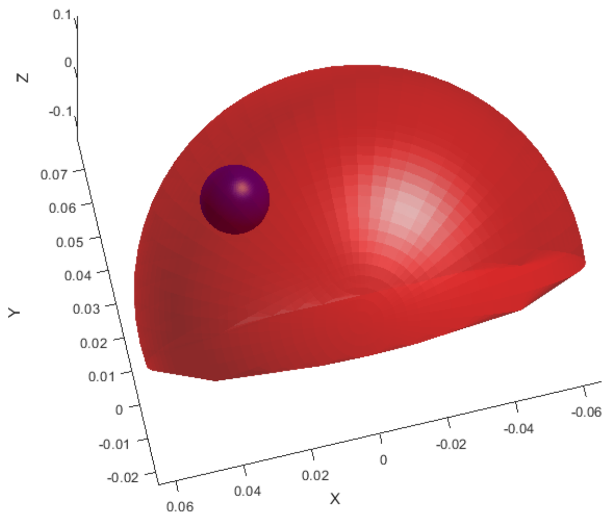
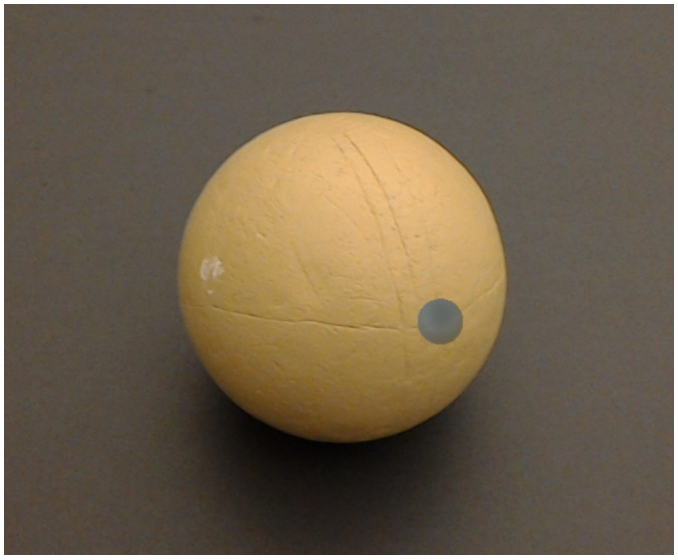
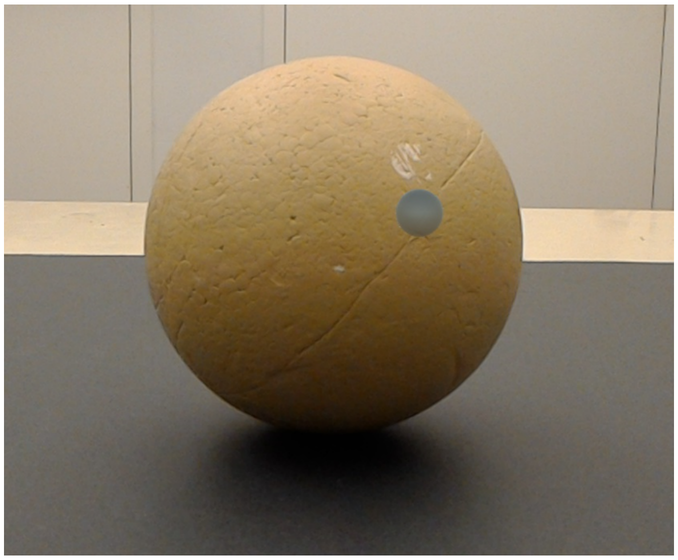


Figure 4.9: Representation of a Tumor within the Simulated Surface (X, Y and Z are 3D Coordinates Representative of Distance in Meters)



(a) Top View



(b) Side View

Figure 4.10: Top and Side View of the Simulated Tumor Visualized within the Real World Object

4.7 Summary

We started initially by scanning a sphere of 13 cm in diameter using our HoloLens, over the course of 50 samples using the technique mentioned in Sections 3.2.4 and 4.2. The resulting point cloud consisted of 1776 points which included noise and a small portion of the table that the sphere was set up on. Subsequent colour filtration and further culling of the lower half of the sphere as explained in Section 4.3 yielded the point cloud of a hemisphere with 364 points. We passed this point cloud into the surface reconstruction algorithm devised in Section 3.3 and executed in Section 4.4, to generate a surface. A hemisphere is then simulated with size 13 cm in diameter at the origin containing 1711 points. An intensity-based automated registration technique is then used to register the simulated point cloud into the space of the scanned point cloud as mentioned in Sections 3.4 and 4.5. Finally a representative body for a tumour is simulated within the space of the simulated point cloud of 2 cm in diameter with its origin at $(0.03, 0.03, 0.03)$, and the transform obtained from the registration process is applied to the same to move the tumour into the HoloLens application space. The transformed coordinates are then sent to the HoloLens where they are visualized in the real world, creating a correspondence between the simulated point cloud and the scanned object.

Through this experiment, it is clear that the HoloLens can be used to effectively read depth information from a surface such that the scan is used in registration with another three-dimensional surface based on a point cloud.

Chapter 5

Conclusion and Future Work

This thesis provides a method in which the Microsoft HoloLens could be used in a framework that enables breast-conserving surgery. Attaining this goal meant that a surface or mesh needed to be generated for the target surface so that it may be registered with the information we receive from the MRI, which included the location of the tumours. The HoloLens with its portability and ease of use is an incredibly viable candidate in this aspect, but its sensors have poor observability of the depth resolution. To address this problem, we proposed a method termed "**Incremental Super Resolution**" which leverages the continuously changing mesh information in the HoloLens by accumulating it over time in a space of interest and further using this data within a surface reconstruction algorithm for convex surfaces, to help in registration. Further, we demonstrate an experiment through which the viability of the information collected from the HoloLens can be validated by registering a simulated curved convex surface with data collected from a similar surface in the real-world.

Our initial intention with this project was to provide contributions towards the software end of this problem's spectrum. However, the hardware limitations as mentioned above, led us into the quest for methods into overcoming the same. Thus there exists a large avenue along which work needs to be done in perfecting our proposed method. The future work based on this thesis would be to experiment with the system in terms of objects that possess a varying topology. The methods discussed within this thesis can also be adapted into a continuous system that cycles through the three outlined steps as mentioned in Section 3.1, rather than the batch data collection and processing we performed in Chapter 4. Additionally, experiments for such a framework, would come in using the HoloLens to capture information from its real-world scenario of use. This collection of data from a live patient and then registering this information with the MRI information would contribute as a final validation step. More investigations into the scalability and robustness of the surface reconstruction algorithm, including the involvement of varying prior information such as vertex normal data can be performed. The registration techniques for such a use case could also be modified, optimized, and improved through experimentation with surfaces of complicated topology. The current filters we use in reducing the noise from the point cloud we acquire from the HoloLens are inadequate in a real-world scenario and can be reworked by using prior information that utilizes the skin colour of the patient for isolation of the breast. Moving forward, the system can be modified to work during the process of surgery, where the tumor location moves, adjusting itself, as the breast is moved. Finally a complete framework that could be used in surgical scenarios can be constructed which scans data in real-time, sends it wirelessly to a processing system (for reconstruction and registration), which in turn responds with the MRI based surface registered in the surgical world coordinate space, thus locating the tumor.

References

[1] Affine transformation - MATLAB & Simulink. https://www.mathworks.com/discovery/affine-transformation.html. (Accessed on 11/26/2017).

[2] Case western reserve university - Hololens. http://case.edu/hololens/. (Accessed on 12/13/2017).

[3] Configurations for intensity-based registration - matlab imregconfig. https://www.mathworks.com/help/images/ref/imregconfig.html. (Accessed on 11/26/2017).

[4] Estimate geometric transformation that aligns two 2-D or 3-D images - MATLAB imregtform. https://www.mathworks.com/help/images/ref/imregtform.html. (Accessed on 11/26/2017).

[5] Facts & stats. http://www.cbcf.org/central/AboutBreastCancerMain/FactsStats/Pages/default.aspx. (Accessed on 12/13/2017).

[6] Gc methods (system). https://msdn.microsoft.com/en-us/library/system.gc_methods%28v=vs.110%29.aspx?f=255&MSPPError=-2147217396. (Accessed on 11/22/2017).

[7] Hololens hardware details. https://developer.microsoft.com/en-us/windows/mixed-reality/hololens_hardware_details. (Accessed on 11/04/2017).

[8] Install the tools. https://developer.microsoft.com/en-us/windows/mixed-reality/install_the_tools. (Accessed on 11/08/2017).

[9] Intensity-based automatic image registration - MATLAB & Simulink. https://www.mathworks.com/help/images/intensity-based-automatic-image-registration.html. (Accessed on 11/26/2017).

[10] Locatable camera. https://developer.microsoft.com/en-us/windows/mixed-reality/locatable_camera. (Accessed on 11/11/2017).

[11] Objexporter - unify community Wiki. http://wiki.unity3d.com/index.php?title=ObjExporter. (Accessed on 11/08/2017).

[12] Reference 3-D image to world coordinates - MATLAB. https://www.mathworks.com/help/images/ref/imref3d.html. (Accessed on 11/26/2017).

[13] Screening for breast cancer canadian cancer society - breast cancer support and information programs. http://support.cbcf.org/get-information/hereditary-breast-ovarian-cancer-hboc/managing-your-breast-cancer-risk/screening-for-breast-cancer/. (Accessed on 12/14/2017).

[14] Unity - manual: Coroutines. https://docs.unity3d.com/Manual/Coroutines.html. (Accessed on 11/21/2017).

[15] Unity - manual: Shaderlab: Culling & depth testing. https://docs.unity3d.com/Manual/SL-CullAndDepth.html. (Accessed on 11/10/2017).

[16] Unity - manual: Shaderlab syntax. https://docs.unity3d.com/Manual/SL-Shader.html. (Accessed on 11/10/2017).

[17] Unity - scripting API: Object.instantiate. https://docs.unity3d.com/ScriptReference/Object.Instantiate.html. (Accessed on 11/22/2017).

[18] https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/, 5 2017. (Accessed on 11/08/2017).

[19] Materials, shaders and textures. https://docs.unity3d.com/Manual/Shaders.html, Oct 2017. (Accessed on 11/06/2017).

[20] Nina Amenta, Sunghee Choi, Tamal K Dey, and Naveen Leekha. A simple algorithm for homeomorphic surface reconstruction. In *Proceedings of the sixteenth annual symposium on Computational geometry*, pages 213–222. ACM, 2000.

[21] Nina Amenta, Sunghee Choi, and Ravi Krishna Kolluri. The power crust. In *Proceedings of the sixth ACM symposium on Solid modeling and applications*, pages 249–266. ACM, 2001.

[22] Franz Aurenhammer. Voronoi diagramsa survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991.

[23] Ronald T Azuma. A survey of augmented reality. *Presence: Teleoperators and virtual environments*, 6(4):355–385, 1997.

[24] Jonathan C Carr, Richard K Beatson, Jon B Cherrie, Tim J Mitchell, W Richard Fright, Bruce C McCallum, and Tim R Evans. Reconstruction and representation of 3D objects with radial basis functions. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 67–76. ACM, 2001.

[25] Jonathan C Carr, W Richard Fright, and Richard K Beatson. Surface interpolation with radial basis functions for medical imaging. *IEEE transactions on medical imaging*, 16(1):96–107, 1997.

[26] Antonin Chambolle. An algorithm for total variation minimization and applications. *Journal of Mathematical imaging and vision*, 20(1):89–97, 2004.

[27] Tony F Chan and Luminita A Vese. Active contours without edges. *IEEE Transactions on image processing*, 10(2):266–277, 2001.

[28] Wikimedia Commons. File:dolphin triangle mesh.svg — Wikimedia Commons, the free media repository, 2012. [Online; accessed 16-December-2017].

[29] Claudius Conrad, Matteo Fusaglia, Matthias Peterhans, Huanxiang Lu, Stefan Weber, and Brice Gayet. Augmented reality navigation surgery facilitates laparoscopic rescue of failed portal vein embolization. *Journal of the American College of Surgeons*, 223(4):e31–e34, 2016.

[30] Nan Cui, Pradosh Kharel, and Viktor Gruev. Augmented reality with Microsoft Hololens holograms for near infrared fluorescence based image guided surgery. In *Proc. of SPIE Vol*, volume 10049, pages 100490I–1, 2017.

[31] Jesús A De Loera, Jörg Rambau, and Francisco Santos. *Triangulations Structures for algorithms and applications*. Springer, 2010.

[32] Tamal K Dey and Samrat Goswami. Tight cocone: a water-tight surface reconstructor. In *Proceedings of the eighth ACM symposium on Solid modeling and applications*, pages 127–134. ACM, 2003.

[33] Herbert Edelsbrunner and Ernst P Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics (TOG)*, 13(1):43–72, 1994.

[34] Michele Fiorentino, Raffaele de Amicis, Giuseppe Monno, and Andre Stork. Spacedesign: A mixed reality workspace for aesthetic industrial design. In *Proceedings of the 1st International Symposium on Mixed and Augmented Reality*, page 86. IEEE Computer Society, 2002.

[35] Henry Fuchs, Mark A Livingston, Ramesh Raskar, Kurtis Keller, Jessica R Crawford, Paul Rademacher, Samuel H Drake, Anthony A Meyer, et al. Augmented reality visualization for laparoscopic surgery. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 934–943. Springer, 1998.

[36] Hugues Hoppe, Tony DeRose, Tom Duchamp, Mark Halstead, Hubert Jin, John McDonald, Jean Schweitzer, and Werner Stuetzle. Piecewise smooth surface reconstruction. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 295–302. ACM, 1994.

[37] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. *Surface reconstruction from unorganized points*, volume 26. ACM, 1992.

[38] Etienne G Huot, Hussein M Yahia, Isaac Cohen, and Isabelle L Herlin. Surface matching with large deformations and arbitrary topology: a geodesic distance evolution scheme on a 3-manifold. In *European Conference on Computer Vision*, pages 769–783. Springer, 2000.

[39] Hannes Kaufmann and Dieter Schmalstieg. Mathematics and geometry education with collaborative augmented reality. *Computers & graphics*, 27(3):339–345, 2003.

[40] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (TOG)*, 32(3):29, 2013.

[41] Michael M Kazhdan et al. Reconstruction of solid models from oriented point sets. In *Symposium on Geometry Processing*, pages 73–82, 2005.

[42] Patrick J Kelly, Bruce A Kall, Stephan Goerss, and Franklin Earnest IV. Computer-assisted stereotaxic laser resection of intra-axial brain neoplasms. *Journal of neurosurgery*, 64(3):427–439, 1986.

[43] Johannes Korsawe. intriangulation(vertices,faces,testp,heavytest) - file exchange - MATLAB Central. https://www.mathworks.com/matlabcentral/fileexchange/43381-intriangulation-vertices-faces-testp-heavytest-. (Accessed on 11/26/2017).

[44] Aaron Kotranza and Benjamin Lok. Virtual human+ tangible interface= mixed reality human an initial exploration with a virtual breast exam patient. In *Virtual Reality Conference, 2008. VR'08. IEEE*, pages 99–106. IEEE, 2008.

[45] Ivo Kuhlemann, Markus Kleemann, Philipp Jauer, Achim Schweikard, and Floris Ernst. Towards x-ray free endovascular interventions–using Hololens for on-line holographic visualisation. *Healthcare Technology Letters*, 2017.

[46] Jian Liang, Frederick Park, and Hongkai Zhao. Robust and efficient implicit surface reconstruction for point clouds based on convexified image segmentation. *Journal of Scientific Computing*, 54(2-3):577–602, 2013.

[47] Kenton McHenry and Peter Bajcsy. An overview of 3D data content, file formats and viewers. *National Center for Supercomputing Applications*, 1205:22, 2008.

[48] Yuriy Mischenko. 3D Euclidean distance transform for variable data aspect ratio - file exchange - MATLAB Central. https://www.mathworks.com/matlabcentral/fileexchange/15455-3d-euclidean-distance-transform-for-variable-data-aspect-ratio. (Accessed on 11/26/2017).

[49] Yuriy Mishchenko. A fast algorithm for computation of discrete Euclidean distance transform in three or more dimensions on vector processing architectures. *Signal, Image and Video Processing*, 9(1):19–27, 2015.

[50] Wolfgang Narzt, Gustav Pomberger, Alois Ferscha, Dieter Kolb, Reiner Müller, Jan Wieghardt, Horst Hörtner, and Christopher Lindinger. Augmented reality navigation systems. *Universal Access in the Information Society*, 4(3):177–187, 2006.

[51] Stanley Osher and Nikos Paragios. *Geometric level set methods in imaging, vision, and graphics*. Springer Science & Business Media, 2003.

[52] Sandeep Patil and B Ravi. Voxel-based representation, display and thickness analysis of intricate shapes. In *Computer Aided Design and Computer Graphics, 2005. Ninth International Conference on*, pages 6–pp. IEEE, 2005.

[53] Peter Pietrzak, Manit Arya, Jean V Joseph, and Hiten RH Patel. Three-dimensional visualization in laparoscopic surgery. *BJU international*, 98(2):253–256, 2006.

[54] Leonid I Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1-4):259–268, 1992.

[55] Yoshinobu Sato, Masahiko Nakamoto, Yasuhiro Tamaki, Toshihiko Sasama, Isao Sakita, Yoshikazu Nakajima, Morito Monden, and Shinichi Tamura. Image guidance of breast cancer surgery using 3-D ultrasound images and augmented reality visualization. *IEEE Transactions on Medical Imaging*, 17(5):681–693, 1998.

[56] Mark Sussman and Emad Fatemi. An efficient, interface-preserving level set redistancing algorithm and its application to interfacial incompressible fluid flow. *SIAM Journal on scientific computing*, 20(4):1165–1191, 1999.

[57] Mark Sussman, Peter Smereka, and Stanley Osher. A level set approach for computing solutions to incompressible two-phase flow. *Journal of Computational physics*, 114(1):146–159, 1994.

[58] Renoald Tang, Setan Halim, and Majid Zulkepli. Surface reconstruction algorithms: Review and comparison. In *The 8th International Symposium On Digital Earth (ISDE 2013)*, 2013.

[59] Fabienne Thibault, Claude Nos, Martine Meunier, Carl El Khoury, Liliane Ollivier, Brigitte Sigal-Zafrani, and Krishna Clough. MRI for surgical planning in patients with breast cancer who undergo preoperative chemotherapy. *American Journal of Roentgenology*, 183(4):1159–1168, 2004.

[60] DWF Van Krevelen and Ronald Poelman. A survey of augmented reality technologies, applications and limitations. *International Journal of Virtual Reality*, 9(2):1, 2010.

[61] Sebastian Vogt, Ali Khamene, and Frank Sauer. Reality augmentation for medical procedures: System architecture, single camera marker tracking, and system evaluation. *International Journal of Computer Vision*, 70(2):179, 2006.

[62] Joachim Weickert. *Anisotropic diffusion in image processing*, volume 1. Teubner Stuttgart, 1998.

[63] Eric W Weisstein. Euler Lagrange differential equation. MathWorld - A Wolfram Web Resource http://mathworld.wolfram.com/Euler-LagrangeDifferentialEquation.html. (Accessed on 11/07/2017).

[64] Steve Chi-Yin Yuen, Gallayanee Yaoyuneyong, and Erik Johnson. Augmented reality: An overview and five directions for ar in education. *Journal of Educational Technology Development and Exchange (JETDE)*, 4(1):11, 2011.

APPENDICES

Appendix A

Semantic and Syntactical References

The sections detailed below describe semantic and syntactical references to the topics discussed in the main body of this thesis in developing a framework performing ISR.

A.1 Building Objects in an OBJ File

Listings A.1 and A.2 demonstrate the construction of 2 elements, one a 2D square in 3D space and the other a cube, through the use of parameters supplied through an OBJ file. The corresponding output of the constructed OBJ files for the square and cube are as shown in Figures A.1a and A.1b respectively.

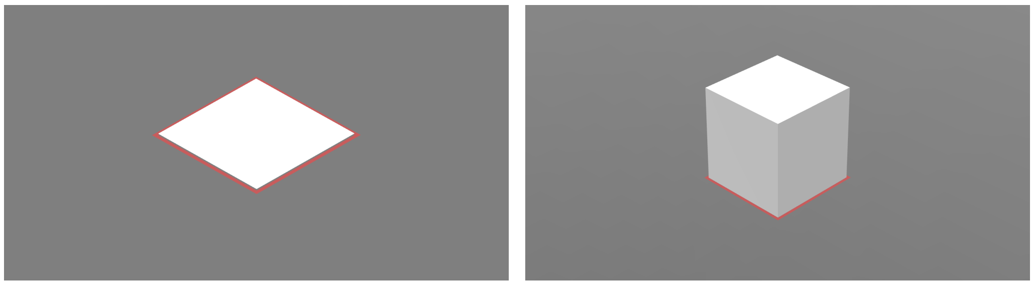
```
v -1 -1 0.1
v  1 -1 0.1
v  1  1 0.1
v -1  1 0.1
f  1  2  3
f  1  3  4
```

Listing A.1: Example of OBJ file structure representing the square from Listing 2.4

```
v  0.0  0.0  0.0
v  0.0  0.0  1.0
v  0.0  1.0  0.0
v  0.0  1.0  1.0
v  1.0  0.0  0.0
```

```
v   1.0   0.0   1.0
v   1.0   1.0   0.0
v   1.0   1.0   1.0
f   1   7   5
f   1   3   7
f   1   4   3
f   1   2   4
f   1   5   6
f   1   6   2
f   3   8   7
f   3   4   8
f   5   7   8
f   5   8   6
f   1   5   6
f   1   6   2
f   2   6   8
f   2   8   4
```

Listing A.2: Example of OBJ file structure representing a unit cube



(a) OBJ File View of Listing A.1 (b) OBJ File View of Listing A.2

Figure A.1: OBJ File Viewed through Microsoft's 3D Builder

A.2 Creating an OBJ File Exporter

The following details the techniques in creating an OBJ file exporter, including compilation segregation using #if definitions to ensure a clean build in Unity, and methods that translate C# mesh objects into strings that could be written to file.

A.2.1 Writing to a File and Code Compile Segregation

Listing A.3 details code that is written such that certain lines only compile outside the Unity editor environment, This ensures that the project builds into the VS solution such that it can further be seamlessly built and deployed from there into the device. This ensures that the 'Stream' object would be initiated correctly and would perform without glitches in the methods that request for the same from the 'OpenFileForWrite' function.

```
/* namespace imports that need to be ignored in the Unity Editor
   environment pre-build */
#if !UNITY_EDITOR && UNITY_WSA
using System.Threading.Tasks;
using Windows.Storage;
using Windows.Storage.Streams;
#endif

public class ObjExporter
{
  public static string MeshFolderName
  {
    get
    {
#if !UNITY_EDITOR && UNITY_METRO
      return ApplicationData.Current.RoamingFolder.Path;
#else
      return Application.persistentDataPath;
#endif
    }
  }

  private static Stream OpenFileForWrite(string folderName,
     string fileName)
  {
    Stream stream = null;
#if !UNITY_EDITOR && UNITY_METRO
    Task<Task> task = Task<Task>.Factory.StartNew(
        async () =>
        {
          StorageFolder folder = await
             StorageFolder.GetFolderFromPathAsync(folderName);
```

```
            StorageFile file = await
               folder.CreateFileAsync(fileName,
               CreationCollisionOption.ReplaceExisting);
            stream = await file.OpenStreamForWriteAsync();
          });
      task.Wait();
      task.Result.Wait();
#else
      stream = new FileStream(Path.Combine(folderName, fileName),
         FileMode.Create, FileAccess.Write);
#endif
      return stream;
     }
}
```

Listing A.3: Writing a file during run-time in the HoloLens (Compile code segregation using #if as modified from the Mixed Reality Toolkit's MeshSaver functionality)

A.2.2 Converting C# Mesh Objects to String

From the above sub-section, we have a Stream object which we can leverage in saving a file into the HoloLens file system during application runtime and a file structure we intend to utilize for storing information. The next step is to create a generic method that can translate a 'MeshFilter' object into a string. The Unity3D Wiki has an ObjExporter function to do this but the file saving section of the code is not compatible with the HoloLens [11] because of the VS solution build issue. We can modify the same code to handle multiple mesh filters or we can create a function to handle storage of specific information from a mesh such as a list of vertices in the form of a Vector3 object list.

```
public class ObjExporter
{
        ... //Other code

        public static string MeshListToString(List<MeshFilter>
           mfList)
    {
        StringBuilder sb = new StringBuilder();
        int verticeCount = 1;
        foreach (MeshFilter mf in mfList)
        {
            Mesh m = mf.mesh;
            Material[] mats = mf.GetComponent<Renderer>().sharedMaterials;

            sb.Append("g ").Append(mf.name).Append("\n");
            foreach (Vector3 v in m.vertices)
            {
                sb.Append(string.Format("v {0} {1} {2}\n", v.x, v.y, v.z));
            }
            sb.Append("\n");
            foreach (Vector3 v in m.normals)
            {
                sb.Append(string.Format("vn {0} {1} {2}\n", v.x, v.y, v.z));
            }
            sb.Append("\n");
            foreach (Vector3 v in m.uv)
            {
                sb.Append(string.Format("vt {0} {1}\n", v.x, v.y));
            }
            for (int material = 0; material < m.subMeshCount; material++)
            {
                sb.Append("\n");
                sb.Append("usemtl ").Append(mats[material].name).Append("\n");
                sb.Append("usemap ").Append(mats[material].name).Append("\n");

                int[] triangles = m.GetTriangles(material);
                for (int i = 0; i < triangles.Length; i += 3)
                {
                    sb.Append(string.Format("f {0}/{0}/{0} {1}/{1}/{1} {2}/{2}/{2}\n",
                     triangles[i] + verticeCount, triangles[i + 1] + verticeCount, triangles[i + 2] + verticeCount));
                }
            }

            verticeCount += m.vertices.Length;
        }
        return sb.ToString();
    }


    public static string Vector3toString(List<Vector3>
       vectorList, int option)
    {
        //option 1: vertice
        //option 2: normal
        StringBuilder sb = new StringBuilder();
        foreach(Vector3 v in vectorList)
        {
            if(option == 1)
            {
                sb.Append(string.Format("v {0} {1} {2}\n", v.x,
                   v.y, v.z));
            }
            else
            {
                sb.Append(string.Format("vn {0} {1} {2}\n", v.x,
                   v.y, v.z));
            }
        }
        sb.Append("\n");
        return sb.ToString();
    }
}
```

Listing A.4: Converting Objects and Components into OBJ format String (Modified from the OBJExporter functionality as mentioned in the Unify Community Wiki [11])

Listing A.4 features 2 methods, 'MeshListToString' which takes a list of mesh filters as input and 'Vector3toString' which takes as input a list of Vector3 values as well as an integer value.

MeshListToString

This function iteratively takes the mesh filters in the string for which the mesh component is extracted. This mesh component is then broken down into objects such as vertices and faces and a string is constructed from the same logic. This function is particularly useful when we intend to store more than one MeshFilter in a single file.

Note: A room scanned by the Microsoft HoloLens is stored as multiple MeshFilter components so that a large amount of space can be modified at select locations based on iterative scanning results from the device, without loading all the mesh concerning the room at once. That is, when the depth sensor of the HoloLens picks up new information regarding an area of a room that is scanned, modifications would have to be made. These modifications can only be made if the mesh corresponding to the area is loaded into memory and edited based on the internal mesh modification algorithms. Thus, storing the entire mesh in a single filter would then be a bad idea because even minor updates to a single area would necessitate the import of the entire mesh for the room.

Vector3ToString

This function is meant to exclusively encode a List of Vector3 data into either the format of a vertex or a vertex normal based on the option provided into the function. This helps in selectively filtering out information from the mesh that we need rather than storing everything in one go, thus making any function that calls this method more efficient while in parallel reducing the load on the HoloLens system.

A.2.3 Completing the OBJ Exporter Class

The combination of the ideas mentioned above and the code as mentioned in Listings A.3 and A.4 is a class called 'ObjExporter' that provides the functionality to store data as OBJ files. Listing A.5 demonstrates such a method that could be built into the ObjExporter class such that it would leverage both the 'OpenFileForWrite' method as well as the 'MeshListToString' method that has been mentioned above.

```
public class ObjExporter {
... //Other code
    public static string SaveAll(string fileName,
       List<MeshFilter> meshFilters)
    {
        if (string.IsNullOrEmpty(fileName))
        {
            throw new ArgumentException("Must specify a valid
               fileName.");
        }

        if (meshFilters == null)
        {
            throw new ArgumentNullException("Value of meshFilters
               cannot be null.");
        }

        // Create the mesh file.
        String folderName = MeshFolderName;
        using (Stream stream = OpenFileForWrite(folderName,
           fileName + fileExtension))
        {
            // Serialize and write the meshes to the file.
            using (StreamWriter sw = new StreamWriter(stream))
            {
                sw.Write(MeshListToString(meshFilters));
                sw.Flush();
            }
        }

        return Path.Combine(folderName, fileName + fileExtension);
    }
}
```

Listing A.5: Function to Save an Incoming List of Mesh Filters

A.3 Initializing and Using Photo Mode

Listing A.6 demonstrates the coding process where first the ‘PhotoCapture’ object is initiated as null and as a global variable such that other methods in the class have access to it without explicitly being passed into the method itself. This is followed by the call of the ‘CreateAsync’ static function from the ‘PhotoCapture’ class.

The ‘CreateAsync’ function asynchronously creates a new ‘PhotoCapture’ object and takes 2 parameters as input. A Boolean value (valid values are ‘true’ or ‘false’) that specifies if the Holograms in view need to be considered while capturing the image (which if specified false would ignore Holograms) and also a callback.

Thus the ‘CreateAsync’ method once it has initiated a ‘PhotoCapture’ object calls the function ‘OnPhotoCaptureCreated’ which takes as input a ‘PhotoCapture’ object. This is automatically passed in by the ‘CreateAsync’ function and here the callback assigns this object to our global variable. Once this has been completed the callback function also initializes a camera parameters variable and specifies the output we expect. The parameters supplied include settings such as the camera resolution we need in capturing the image and the pixel format in which the image is captured. Once these settings are copied, we can call the ‘StartPhotoModeAsync’ method which takes as input the Camera parameters and another callback. The callback is called once the photo mode of the HoloLens is initiated.

‘StartPhotoModeAsync’ calls its callback with a variable that conveys if the photo mode has successfully initiated. The callback (‘OnPhotoModeStarted’), given the result, if successful, then commands the HoloLens to take a photo by starting the ‘TakePhotoAsync’, else throws an error. Once a photo has been taken the callback provided to ‘TakePhotoAsync’ is called with the results regarding the success of the photo capture as well as a ‘PhotoCaptureFrame’ object that contains the photo. If the photo capture is a success a texture object is initiated which has the capability to store image information, with the width and height of the resolution we specified earlier. This texture is uploaded to the image data and then the ‘cameraToWorldMatrix’ and the ‘projectionMatrix’ for the image are obtained.

The camera-to-world matrix translates any three-dimensional position in terms of the ‘locatable camera’ perspective into the three-dimensional coordinate system used by the application in drawing objects into the scene (our mesh is drawn/can be translated, in terms of these coordinates). The projection matrix helps in converting the three-dimensional camera space coordinates into the pixel coordinates of the image. More regarding these matrices would be discussed below. Finally, once all the information is obtained the ‘StopPhotoModeAsync’ method is called, that stops the photo mode of the HoloLens and also initiates a callback which could be used to dispose of the photo capture object and free memory.

More information regarding the Locatable Camera can be found in the Microsoft Developer documentation [10].

```
using HoloToolkit.Unity;
using System.Linq;
using UnityEngine;
using UnityEngine.XR.WSA.WebCam;

public class RGBCapture {
    PhotoCapture pCO = null; \\ The photo capture object global variable
    Texture2D targetTexture = null; \\ The Texture object where we intend to store the image information
    Matrix4x4 cameraToWorldMatrix = Matrix4x4.identity;
    Matrix4x4 projectionMatrix = Matrix4x4.identity;

        public void capturePhoto()
    {
        PhotoCapture.CreateAsync(false, OnPhotoCaptureCreated);
    }

    private void OnPhotoCaptureCreated(PhotoCapture captureObject)
    {
        pCO = captureObject;

        Resolution cameraResolution = PhotoCapture.SupportedResolutions.OrderByDescending((res) => res.width * res.height).First();
        CameraParameters c = new CameraParameters(); \\ The camera parameters variable for storing the photo requirements during capture
        c.hologramOpacity = 0.0f;
        c.cameraResolutionWidth = cameraResolution.width;
        c.cameraResolutionHeight = cameraResolution.height;
        c.pixelFormat = CapturePixelFormat.BGRA32;

        captureObject.StartPhotoModeAsync(c, OnPhotoModeStarted);
    }

    private void OnPhotoModeStarted(PhotoCapture.PhotoCaptureResult result)
    {
        if (result.success)
        {
            pCO.TakePhotoAsync(OnCapturedPhotoToMemory);
        }
        else
        {
            Debug.LogError("Unable to start photo mode!");
        }
    }

    void OnCapturedPhotoToMemory(PhotoCapture.PhotoCaptureResult
       result, PhotoCaptureFrame photoCaptureFrame)
    {
        if (result.success)
        {
            // Create our Texture2D for use and set the correct
               resolution
            Resolution cameraResolution =
               PhotoCapture.SupportedResolutions.OrderByDescending
                ((res) => res.width * res.height).First();
            targetTexture = new Texture2D(cameraResolution.width,
               cameraResolution.height);
            // Copy the raw image data into our target texture
            photoCaptureFrame.UploadImageDataToTexture
               (targetTexture);
            photoCaptureFrame.TryGetCameraToWorldMatrix(out
               cameraToWorldMatrix);
            photoCaptureFrame.TryGetProjectionMatrix(out
               projectionMatrix);
        }
        // Clean up
        pCO.StopPhotoModeAsync(OnStoppedPhotoMode);
    }

    void OnStoppedPhotoMode(PhotoCapture.PhotoCaptureResult
       result)
    {
        pCO.Dispose();
        pCO = null;
    }
}
```

Listing A.6: Creating a 'PhotoCapture' object and Initiating Photo mode (Modified from the Locatable Camera documentation from the Microsoft Developer website[10])

A.4 A Function to Map Points from World Space to Pixel Space

This method mentioned below can be placed in the same class as the other methods described in Listing A.6. The code in Listing A.7 assumes that the global variables mentioned in Listing A.6 have been populated by the process of taking a photo.

The function in Listing A.7 first finds the transformation matrix for world-to-camera space by finding the inverse of the camera-to-world matrix for the process as mentioned in Equations (3.5) and (3.9). This transformation matrix and the world space position passed into the function is multiplied to find the position of the point in the camera space coordinates. Further, the projection matrix and the camera space position are multiplied and normalized into the $\begin{bmatrix} x & y & 1 \end{bmatrix}$ format by dividing the third parameter in the matrix obtained throughout the entire matrix, where x and y are the pixel positions expressed in the range of $[-1, 1]$ making them members of the $P_{minimal}$ matrix mentioned in Equation (3.10).

An example for when normalization would be required would be, if the result obtained from multiplication with the projection matrix is

$$\begin{bmatrix} a & b & 2 \end{bmatrix}$$

Here we divide the entire matrix by 2 to obtain

$$\begin{bmatrix} \frac{a}{2} & \frac{b}{2} & 1 \end{bmatrix}$$

Finally, Equation (3.10) is used to find the correct pixel position of the world space coordinate.

```
// Other initializations
public class RGBCapture {
        // Other code

```

```
    public Vector2 WorldToPixelPos(Vector3 WorldSpacePos)
    {
      Matrix4x4 WorldToCamera = CameraToWorldMatrix.inverse;
      Vector3 CameraSpacePos =
         WorldToCamera.MultiplyPoint(WorldSpacePos);
      Vector3 ImagePosUnnormalized =
         projectionMatrix.MultiplyPoint(CameraSpacePos); // Get
         unnormalized position (Vector3 format with Vector2
         numbers, z position is not 1
      Vector2 ImagePosProjected = new
         Vector2(ImagePosUnnormalized.x,
         ImagePosUnnormalized.y)/ImagePosUnnormalized.z; //
         normalize by z, gives -1 to 1 space
      Vector2 ImagePosZeroToOne = (ImagePosProjected * 0.5f) +
         new Vector2(0.5f, 0.5f); // good for GPU textures
      Vector2 PixelPos = new Vector2(ImagePosZeroToOne.x *
         targetTexture.width, (1 - ImagePosZeroToOne.y) *
         targetTexture.height);
      return PixelPos;
    }

}
```

Listing A.7: World Space to Pixel Position Function (Modified from the Locatable Camera documentation from the Microsoft Developer Website [10])

More information for camera space conversions are available at the Microsoft Developer documentation online [10].

A.5 Implementing Z-filtering

Listing A.8 exhibits a simple example of this procedure where the method described takes as input the camera position and a vertex to be judged as occluded in view or not. The layer mask is set such that only layer 31 is considered for collision with the ray (layer 31 is the layer the spatial mapping mesh is assigned to by default). If a hit happens and the collision occurred in close proximity to the intended point, the method returns a true value. Else, the vertex is flagged as occluded by returning false.

```
Boolean RayCastVerification(Vector3 cameraPos, Vector3 vertPos)
```

```
{
    RaycastHit hitinfo;
    Vector3 heading = vertPos - cameraPos;
    Vector3 direction = heading / heading.magnitude; // Unit
       vector calculation of direction vector for a ray cast
    int layermask = 1 << 31;  // Layers to be focused on during
       the RayCast procedure
    if (Physics.Raycast(cameraPos, direction, out hitinfo, 10.0f,
       layermask))
    {
        if (Math.Abs((hitinfo.point - cameraPos).magnitude -
           heading.magnitude) > 0.006)  // If the hit did not
           happen in proximity of  the intended location
        {
            return false;
        }
        else
        {
            return true;
        }
    }
    else
    {
        return false;
    }
}
```

Listing A.8: Ray Cast verification procedure