Three Dimensional Visualization through use of HoloLens

by Zachary Sather


Submitted to the

Department of Electrical Engineering and Computer Science in Partial Fulfillment of the

Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science at the

Massachusetts Institute of Technology

August 2017 [September 2017]

© 2017 Zachary Sather. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and to distribute publicly paper and
electronic copies of this thesis document in whole and in part in any medium now known or
hereafter created.

Author: 

Signature redacted

Department of Electrical Engineering and Computer Science
August 18, 2017

Certified by: 

Signature redacted

Reza Nezafat, Associate Professor of Medicine
Thesis Supervisor
August 18, 2017

Certified by: 

Signature redacted

Christopher Temkin, Senior Lecturer and Undergraduate Office
Thesis Co-Supervisor
August 18, 2017

Accepted by: 

Signature redacted

Christopher Temkin, Chairman
Masters of Engineering Thesis Committee
Abstract

For my Thesis I will be using Microsoft’s HoloLens in order to create 3D volumes of MRI data in the form of holograms. The MRI data is taken from heart scans. The goal of this project is to create high resolution images using late gadolinium enhancement (LGE) of myocardial scar. These images would be generated one of two ways: voxel-wise 3D scar rendering or surface projection of the scar. These images will be displayed using Microsoft’s HoloLens. HoloLens is an augmented reality device used to display holograms in the world around the user. These holograms can be then used to assist physician in their examinations or surgeons in their surgeries.

The goal of this project is to have holograms replace the current methods of looking at 2D slices of the heart. Instead of looking at 2D images, the medical professional would use HoloLens to inspect 3D holograms produced using the same data. The holograms would give the professional the ability to see any scar regions, both the interior and exterior of the regions. It would also allow them the ability to navigate the region in ways that they couldn’t before through use of the 3D space. The 3D hologram allows the doctor or surgeon the ability to see the complex 3D architecture of the scar and surrounding heart tissues.
1. Acknowledgements

I would like to acknowledge the BIDMC Cardiac MR lab and Reza Nezafat.
2. Contents

1. Acknowledgements .................................................................................................................. 5
2. Contents................................................................................................................................... 7
3. List of Figures............................................................................................................................. 11
4. Introduction ............................................................................................................................. 13
    Background ............................................................................................................................. 13
    Motivation ................................................................................................................................. 15
    Related.................................................................................................................................... 17
5. Implementation.......................................................................................................................... 18
    Hardware .................................................................................................................................... 19
        HoloLens .............................................................................................................................. 19
    Programs/Libraries.................................................................................................................... 21
        Unity ..................................................................................................................................... 21
        Microsoft Visual Studios ..................................................................................................... 22
        HoloToolKit ......................................................................................................................... 23
        Fo-DICOM ............................................................................................................................ 24
    Application ............................................................................................................................... 25
    Overview .................................................................................................................................... 25
    Backend ..................................................................................................................................... 25
        Image Loader ........................................................................................................................ 25
        Volume Ray Casting Algorithm ......................................................................................... 26
Frontend ................................................................................................. 27
Overview ................................................................................................. 27
Cursor ........................................................................................................ 27
Gesture Controls .................................................................................... 27
Voice Controls ........................................................................................ 28

6. Results ................................................................................................. 29
Data Set .................................................................................................... 29
First Pass ................................................................................................. 32
End Application ....................................................................................... 36

7. Future Work ....................................................................................... 40

8. References .......................................................................................... 41

9. Appendix ............................................................................................. 42
HoloLens Tech Specs ............................................................................... 42
Optics ........................................................................................................ 42
Sensors ....................................................................................................... 42
Human Understanding ............................................................................. 42
Input / Output / Connectivity ................................................................. 42
Power ......................................................................................................... 43
Processors ............................................................................................... 43
Weight ........................................................................................................ 43
Memory ..................................................................................................... 43
3. List of Figures

Figure 1: HoloLens (Microsoft, 2017) ........................................................................................................... 20
Figure 2: Unity Editor ..................................................................................................................................... 22
Figure 3: MSV editor ...................................................................................................................................... 23
Figure 4: Dataset rendered using up to 64 images ...................................................................................... 29
Figure 5: Dataset rendered using up to 128 images .................................................................................... 30
Figure 6: Dataset rendered using up to 256 images .................................................................................... 30
Figure 7: Dataset rendered using up to 512 images .................................................................................... 31
Figure 8: Dataset with no transparencies added .......................................................................................... 32
Figure 9: Start position of Hologram viewed through HoloLens .............................................................. 33
Figure 10: Hologram rotated long X-Axis ...................................................................................................... 33
Figure 11: Hologram rotated along Z-Axis .................................................................................................... 34
Figure 12: Hologram transformed and rotated long Y-Axis ........................................................................ 34
Figure 13: Hologram tilted towards the user ............................................................................................... 35
Figure 14: Hologram with low alpha in start position ................................................................................ 36
Figure 15: Low alpha hologram rotated ...................................................................................................... 37
Figure 16: Higher alpha hologram in start position ..................................................................................... 38
Figure 17: High alpha hologram rotated ...................................................................................................... 38
Figure 18: High alpha hologram rotated along Z-Axis ................................................................................ 39
4. Introduction

Background

Current imaging techniques rely on using late gadolinium enhancement (LGE) to gather data and create a stack of 2D slices. These slices are then inspected in order to get information about the scar and status of the tissue. The process of looking at 2D slices makes inspecting the 3D architecture of the scar very challenging due to the complex nature of the scars. The current methods also make it difficult to merge LGE data and create a full visualization of the scar and surrounding tissue. Data gathered using LGE techniques is a treasure trove of information that is made difficult to access due to the complexity of the 3D scars.

For my research I looked into using Microsoft’s HoloLens to aid in the viewing process. HoloLens is an augmented reality device. The device aims to supplement the user’s space through use of holograms that can be manipulated and seen through the HoloLens. HoloLens makes use of augmented reality device by allowing the user to still interact with the world around them while also allowing the addition of holograms projected on to the space around them.

Augmented Reality (AR) is sometimes confused with Virtual Reality (VR). AR uses the user’s world as a base for the scene and adding holograms to it. Whereas, in VR a new scene is made that the user can interact with instead of using the existing world around the user. AR and VR are sometimes used interchangeably, and though they both alter the world around us, the way they do it is an important difference that leads to different applications of similar technologies. In my case AR is extremely useful as it gives the care giver additional information and allows them to continue working in their environment. VR could also be useful, but cuts off the care giver from the world around them.
Augmented reality (AR) offers a solution to the problem of trying to build this complex 3D architecture in a 2D space. The current methods of analyzing and understanding the 3D architecture of the scars and surrounding tissue creates challenges of not being able to see everything the data has to offer at once. With the help of augmented reality, the full extent of the LGE data can be viewed and used in the medical process through the use of holograms. Holograms allow the user to use the space around them as a canvas to project a fully detailed 3D volume. Doctors and surgeons would be able to not only view, but also manipulate the hologram in order to gain a deeper understanding of the scars and how they interact with the surrounding tissue. It would also allow them to see how information changes with time. This is possible because holograms are able to change with time. They are not static pictures, but instead dynamic volumes that can be manipulated and updated with changing information as time moves.
Motivation

HoloLens can offer a wide variety of improvements to medicine. Current ways of viewing images collected through MRI scans involve using 3D software on a computer or simply viewing the 2D images taken. This is usually done at a computer and doesn’t give a full picture of the complexity of the volume without use of special tools. My research aims to find a way to allow HoloLens to help in this process. Through use of holograms, HoloLens can recreate 3D volumes that the image is representing. Holograms are useful for visualization because they allow the user to experience the 3D volume outside of a 2D surface. In most instances these 3D volumes would be viewed on a 2D screen. Despite displaying the same information holograms allow the user to experience the volume in 3D space rather than on a screen. This can help to give a fuller image of the scan data and allow the user to fully see what the data is displaying and how all the complex structures come together.

3D imaging is helpful in the medical process by providing care givers a way to see inside the human body. It can help see where a tumor is or where foreign objects are located and give the care giver a tool in the diagnosis process. Care givers already use 3D images in the diagnosis process, my research aims to help care givers see the information more clearly.

This application can also be used to help researchers who study these images by allowing them to see the full information and have access to the full reaches of the data.

The goal of my research is to create a volume where the scar tissue on the heart would be clearly visible. The hologram is created using LGE data to build interior and exterior volumes. With these volumes, doctors or surgeons would be able to look inside the heart and see the scar tissue. This hologram would give the surgeon a true image of the heart. The holograms would be able to be manipulated by the user in order to move around and explore the full extent of the data.
and see what different parts of the data look like as a whole. The use of a holographic space to do this allows the user to take full advantage of all the data collected using LGE while also preserving the accuracy of the data. Using AR also leads to an intuitive way to navigate the holograms produced by the data. Instead of looking at 2D slices, the user can see the volume as a whole and interact with it in ways that a normal 3D volume render simply doesn’t allow. Some of these ways include seeing how the data changes with time and having the ability to manipulate the hologram to explore any and all aspects of the data.
Related Work

HoloLens and the world of augmented reality is very young and not a lot of project have tried to do what my research does. The world of AR is mainly associated with Google Glass. Google Glass is a device that was announced by Google in 2013 and released to the public in 2015. (Google Glass, 2017)

Looking into research similar to mine, I found one project that aimed to open DICOM files on HoloLens.

Most work with HoloLens and AR revolves around creating games for both AR and virtual reality (VR). This is where the use of Unity, a game development software, became helpful.

Other applications aim to use HoloLens to create renders of the heart. A very recent application created by ANIMA RES (ANIMA RES, 2017) allows the user to see a heart in a few different state. The application creates a 3D hologram of a heart which the user can interact with by rotating, zooming, and moving around. The application has a few different simulations that show the heart in different states of healthiness. The application I am looking to make will do some similar things in the sense that I want to create a realistic image of the heart that the user can interact with in various ways. The main difference is that my simulation will be created using clinical data. So, it must be adaptive to different forms of data and generalized enough to visualize different data sets. My application will also be able to represent different scar tissues and allow the user to interact with the scarred region in order to gain a better idea of the state of the region.
5. Implementation

Overview

The application was developed using Unity. The main scripting language is C#. The application is split into a backend and a frontend with various sub parts. First, there is the backend which consists of the file reader and rendering code. Next there is the frontend which has the cursor, gesture, and voice controls in it. For this application a traditional GUI was avoided. This is because HoloLens is designed to be used in a non-traditional way. Mainly, without a mouse and keyboard as inputs, instead HoloLens relies on using the user’s gaze and hand motions to interact with the environment. In addition to hand motions HoloLens also makes use of voice commands in order to facilitate use of an application.
Hardware

HoloLens

HoloLens is an augmented reality device designed by Microsoft. While it was being developed it was known as Project Baraboo (reference). It was designed in the linage of Microsoft’s Kinect and was the first computer to run Windows Mixed Reality platform. The first versions of HoloLens shipped in 2016 in a developer’s package. HoloLens tech specs can be found in the appendix.

The device has a standalone operating system that runs Windows 10 applications. One limitation of the operating system is that it does not have a file explorer included. In
order to access files one has to use OneDrive or a specific companion application for HoloLens.

Some challenges of working with HoloLens was that HoloLens limits the amount of RAM applications of access to down to one gigabyte, which is half of the RAM included in the device. This is the total amount of memory allowed for every running application.
Programs/Libraries

Unity

Unity is a game development software that offers support for HoloLens and the Windows Universal API. Due to the support of the Universal Window’s Platform (UWP), Unity can be used to develop HoloLens applications. There are tutorials on Microsoft’s website that go through how to use Unity to create applications and the specific settings needed to run HoloLens. Unity works off of making a scene and populating that scene with objects in the game world. The developer is then allowed to add textures, shaders, and scripts to the objects. The addition of scripts allows the objects to be interacted with and used in conjunction with other objects. Some important settings are changing the background to a solid black because HoloLens then won’t render the background color and instead allow the world around the user to be viewed and only render the objects in the scene. Users interact with these objects through use of the camera object. The camera allows the user to view the different assets in the scene and interact with them through use of scripts that can be attached to the objects.
Microsoft Visual Studios

Microsoft Visual Studios (MVS) is what I used to do all of my coding in C#. One of Unity’s scripting languages is C# and this is the language that most of the tutorials for HoloLens are written in.

C# is very close to C++ in that it is object oriented and runs at compile time. MVS offers a debugger for HoloLens that I made use of to help me implement the application. MVS was mainly used to write the scripts, but then the scripts were attached to Unity objects and the Unity application was compiled and built. Then the compiled Unity application was opened in MVS and HoloLens was connected to MVS. The Unity application was then compiled to HoloLens and sent to the device via USB. The application could then be run on HoloLens through either the debugger or standalone on the device.
HoloToolKit

HoloToolKit is an open source project (MixedRealityToolkit, 2017) created to make developing on HoloLens easier in Unity. It includes pre made scripts and assets that the developer can add to their scene that make use of HoloLens’s features. Some of the basic things it includes is a camera set up in the correct way for HoloLens, presets for the project properties, and a cursor along with the script to use the cursor. (MixedRealityToolkit, 2017)

The main assets I used were the cursor, camera, and input scripts. The input scripts helped to implement the voice and gesture commands. It also added the ability to track the user’s gaze along with tracking objects and creating the ability to interact with the volume. These scripts and assets were essential in the final implementation.
Fo-DICOM

Fo-DICOM is a library I used in order to convert raw DICOM files into a format that Unity can read. I used Fo-DICOM in the DICOM loader. Fo-DICOM is a C# library that has a version on Github.
Application

Overview

My application is split into a few parts. The first part is the Digital Imaging and Communication in Medicine (DICOM) Loader that takes in the DICOM files and converts to a format that can be used by Unity. DICOM files are the standard way to send information around in medicine. The next part is the Volume Ray Casting Algorithm. This part is where the images are converted into the 3D volume. The final part is the gesture and voice controls for use in HoloLens.

This application uses a variety of classes from Unity. Some of the important ones are Texture2D, Texture3D, and shaders. The Texture classes allow Unity to render both 2D and 3D images while shaders allow the images to be seen by the users.

All of the coding for the project was done through Microsoft Visual Studios in C#. The scripts were then added to a scene in Unity. The scene itself had two main objects, the camera and a cube where the volume would be rendered. Other objects exist in the scene but are not viewable by the user and contain scripts that manage inputs from the user.

Backend

Image Loader

The first part of my application is the DICOM loader. The DICOM loader takes in DICOM files, .dcm, and processes them into a format used by Unity, the Texture2D class. It does this by opening up the StreamingAssets file in application’s directory and reading the contents. StreamingAssets is a unique folder in Unity applications because whatever is put into before the application is built stays through after the application is built. This makes the folder useful for storing custom files such as DICOM files. The loader can then make a call to
“application.streamingassets” to get the path in the new location that the application is built to. Once it has the path it can iterate over all the files and feed them into Fo-DICOM. Fo-DICOM can then convert the “.dcm” files into Texture2D files that Unity can read. In order to save on memory, I stored the data as a DicomFile instead of the Texture2D’s. This is because I found that Texture2D’s took up a lot of memory compared to the DicomFile Class. These files are stored in an array where they can be used later by the Ray Volume Casting algorithm.

Volume Ray Casting Algorithm

For creating the 3D volumes, I used a Volume Ray Casting Algorithm. This algorithm works by scanning over the 2D images and mapping the pixels to a spot in a 3D volume. I implemented this algorithm by iterating over the height, width, and depth components of the volume. I had to sort the slices so that they appear in order. I then had to iterate over the list of DicomFiles and convert them to Texture2D so that I could take pixel data. Then I used some math to figure out where in the 3D volume the pixel was located based on the slice I was on and the height-width position in the image. With this information I could use the GetPixelBilinear function in order to get the pixel information at a particular position. Then save this information into a Texture3D.

The next part involves rendering the image. In order to get this to work another camera was added to the scene to allow for depth to be perceived Shaders had to be added to the back and the front of the target cube. A composite shader was also used. Its purpose was to meld with the render to allow to render to not take up the whole screen. The way the rendering process worked was it takes the current scene viewed by the camera and renders the image onto the target area. Without the composite shader the rendered image would take up the whole scene and nothing else, like the
cursor, could be rendered on top of the scene. The composite shader acts like a mask and allows other things to be rendered on top of the target cube.

**Frontend**

**Overview**

Originally I was thinking of implementing a GUI with buttons and sliders on it. I started implementing these features and after some testing decided to use Gesture and Voice controls instead. The main components of the frontend consist of the cursor, gesture and voice controls. The cursor is important because it lets the users know where they are looking and when they execute a select event. Gesture controls were useful for controls the direct of rotation during a rotation event and choosing where to move the object to in a translation event.

**Cursor**

The cursor is one of the most important features in terms of feedback to the users. In order to implement it I used the HoloToolKit which includes a default cursor and corresponding script. The cursor consists of a model for the cursor being on and off an object along with a different cursor to indicate a select gesture on an object. These models are used in a script that detects when the cursor is on or off an object. The cursor acts similarly to a mouse cursor except the movement is controlled by the user’s gaze instead of a mouse.

**Gesture Controls**

Gesture controls are the most natural way to interact with holograms with HoloLens. I implemented Gesture Controls with the help of HoloToolKit. HoloToolKit has a variety of tools to help with adding Gesture controls to an application. In order to add Gesture Controls, I needed to create an input manager to contain all of the input scripts. I also needed to make sure to add
interactive scripts to the game objects that I wanted to interact with, namely the cube that contains the 3D volume.

The first input script tracks to see where the user is looking and if they are looking at a hologram. The next script detects when the user “selects” through a tap gesture. Once a tap gesture is recognized, the script checks if the users gaze is over an object. If it isn’t over an object, no event is triggered. If it is over an object, then the corresponding event is triggered. This event can either be a move or rotation event. The event is concluded when the uses releases the select gesture or the users hand moves out of sight of the HoloLens sensors.

**Voice Controls**

Voice controls were added in order to allow the uses a more natural way to select what kind of move event they wanted. The keyword “move” allows the next select event to translate the object through the environment. I later added the words “X”, “Y”, and “Z” in order to the application to know what axis to rotate around. In order to add this, I make use of the HoloToolKit again. In this case, I added a keyword dictionary that contained the words and mapped them to their actions. I then had to add function calls to the mapped actions. The action would communicate with the object manager to see if an object was being looked at by the user. If the object was, then the action would trigger. In the case of move, the next select gesture of that object would start a move action that would allow the user to translate the object through their environment. The rotation commands would change some internal variables that are used to decide what axis to rotate around.
6. Results

Data Set

For testing I used a data set of a human heart. The data set has 250 high resolution images. All the images together create a full volume of a human heart. The images are grey scaled where the bright parts are the important component because they represent the fibrosis that a physician would be looking for.

A challenge of this data was that it was very memory intensive to process. The result was that I had to downscale the final volume to only use 128 of the images. I attempted to use more images, but the memory requirements out did what HoloLens is able to handle. In the end an under sampled image effectively using every other file was the best HoloLens could render.

Figure 4: Dataset rendered using up to 64 images
Figure 5: Dataset rendered using up to 128 images

Figure 6: Dataset rendered using up to 256 images
Figure 7 Dataset rendered using up to 512 images
First Pass

My first iteration had only half of the heart visible. This was because of dark tissue that could not be seen through. The solution was to set a minimum intensity so that anything below it would be set to fully transparent. This was done by setting the alpha value of the Color object to 0.0 if the intensity was below a certain number. After some experimentation I found that any pixel color below 0.25 (on a scale of 0-1) yielded a clear picture for the heart.

![Dataset with no transparencies added](image)

From here I noticed that it was really hard to tell what was going on inside of the heart. So I set a middle range up so that any value above the lower bound but below the upper bound would be set to be transparent. After some testing I found that setting the alpha value of the Color objects whose pixel value was in the middle range to be around 0.1 gave a good sense of what was going on inside of the heart while still allowing the user to see the structure.
Figure 9: Start position of Hologram viewed through HoloLens

Figure 10: Hologram rotated long X-Axis
Figure 11: Hologram rotated along Z-Axis

Figure 12: Hologram transformed and rotated along Y-Axis
At this point I added a section to allow the designer to change the high intensity colors and alphas. However, I decided not to change anything about it at this level.

From here I showed it to some people in the lab and was then shown a program that does a similar process to what I was trying to do on a computer. It was recommended I try some different color schemes and settled on red for the middle range and yellow for the high intensities. From here I tweaked the alpha values to made the structure of the heart to stick out more.

The next changes that I made were to make the voice commands more useable. Originally, the user could move and rotate the volume. The rotation was only along the y-axis. After some consideration I found this to be very limiting. So, to increase the usability, I added the ability to rotate along the x-axis and z-axis.
End Application

The end result has four voice commands. They are "move", "X", "Y", and "Z". The voice command "move" allows the user to select the volume and move it around in the environment. Once the selection is ended, the "move" action is ended. The "X", "Y", and "Z" commands all facilitate rotation. Once one of the rotation commands is said any subsequent selection action will result in the rotation along the said axis. The color palate we decided on put red on the middle range intensity and yellow on the high intensity. The cut offs for the intensities were 0.2 for the lower bound and then 0.85 for the upper bound.

Figures 14 and 15 show a low alpha value with red for the middle range and yellow for the high intensities. Figure 14 has the starting position as viewed through HoloLens once the application is started. Figure 15 is the rotated volume to see the high intensity parts more clearly.
Figures 16, 17 and 18 are the same intensity ranges for the transparencies but with a higher alpha value than in figures 14 and 15. Figure 16 is the hologram in the start position of the application. Figure 17 and 18 have the hologram rotated to see the high intensity parts on the bottom more clearly.

The end application is a standalone application that can take in DICOM files and render a 3D volume that the 2D images represent. This differs from other methods being explored in the lab by simply taking in the DICOM files directly and not needing any preprocessing. The advantages being that it is easy to use and does not require multiple programs in order to obtain the end volume. The application was well received by the lab members and overall is a success and a good step in using AR technology to aid in the medical process. (Microsoft, 2017)
Figure 16: Higher alpha hologram in start position

Figure 17: High alpha hologram rotated
Figure 18: High alpha hologram rotated along Z-Axis
7. Future Work

The application could benefit from a few additions. First, allowing the user to vary the intensities that become transparent and the level of transparencies would be useful. I didn’t add this due to memory issues. The way my application is built, the 3D volume needs to be redrawn in order to change the pixel borders or the alphas of the pixels within the border. Due to the fact that my application takes up almost off of HoloLen’s RAM on the first pass, I did not want to try to redraw the volume and risk causing the application to crash.

Another addition would be the ability to change the colors in the different intensities. Allowing a custom color palate that the user could change would allow the user more customization options. Currently you can only add three different colors, one for the low intensity, middle intensity and high intensity. If you change one of these ranges, you must rebuild the application.

One last addition would be displaying the 2D images along with the 3D structure. I considered trying to do this, but would need to severely optimize my code or wait until a more powerful version of HoloLens is released.
8. References


    https://github.com/Microsoft/MixedRealityToolkit-Unity
9. Appendix

HoloLens Tech Specs

Optics

• See-through holographic lenses (waveguides)
• 2 HD 16:9 light engines
• Automatic pupillary distance calibration
• Holographic Resolution: 2.3M total light points
• Holographic Density: >2.5k radiants (light points per radian)

Sensors

• 1 IMU
• 4 environment understanding cameras
• 1 depth camera
• 1 2MP photo / HD video camera
• Mixed reality capture
• 4 microphones
• 1 ambient light sensor

Human Understanding

• Spatial sound
• Gaze tracking
• Gesture input
• Voice support

Input / Output / Connectivity

• Built-in speakers
- Audio 3.5mm jack
- Volume up/down
- Brightness up/down
- Power button
- Battery status LEDs
- Wi-Fi 802.11ac
- Micro USB 2.0
- Bluetooth 4.1 LE

Power

- Battery Life
  - 2-3 hours of active use
  - Up to 2 weeks of standby time
  - Fully functional when charging
  - Passively cooled (no fans)

Processors

- Intel 32 bit architecture with TPM 2.0 support
- Custom-built Microsoft Holographic Processing Unit (HPU 1.0)

Weight

- 579g

Memory

- 64GB Flash
- 2GB RAM
Ray Casting Code

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.VR.WSA;
using System.IO;
using Dicom;
using Dicom.Imaging;
using Dicom.Log;

public class RayCasting : MonoBehaviour

    // [SerializeField]
    private int volumeWidth = 128; // 256 won't render in HoloLens
    // [SerializeField]
    private int volumeHeight = 128; // 256 won't render in HoloLens
    // [SerializeField]
    private int volumeDepth = 128; // 256 won't render in HoloLens
    [SerializeField]
    private LayerMask volumeLayer;
    // [SerializeField][Range(0, 2)]
    private float opacity = 1;
    // [SerializeField]
    private int downscale = 2;

    private Shader compositeShader;
    private Shader renderFrontDepthShader;
    private Shader renderBackDepthShader;
    private Shader rayMarchShader;
    // [SerializeField][Range(0, 99)]
    private int xclip = 99;
    // [SerializeField][Range(0, 99)]
    private int yclip = 99;
    // [SerializeField][Range(0, 99)]
    private int zclip = 99;
    // [SerializeField]
    private float opac = 0.02f; // transparency value
    // [SerializeField]
    private float col = 0.85f; // threshold for high intensity (anything above considered high intensity)

    private Vector4 clipDimensions;

    public Texture2D[] slices;
    private List<Texture2D> _slices;
    private Texture3D volumeBuffer;
private Camera _ppCamera;
private Material _rayMarchMaterial;
private Material _compositeMaterial;
private float oldOpac;
private float oldCol;

private string myPath;
private bool hasFiles;
private bool hasMesh;
private DicomFile[] images;

private void Awake()
{
    compositeShader = Shader.Find("Hidden/Ray Marching/Composite");
    _compositeMaterial = new Material(compositeShader);
    rayMarchShader = Shader.Find("Hidden/Ray Marching/Ray Marching");
    _rayMarchMaterial = new Material(rayMarchShader);
    renderFrontDepthShader = Shader.Find("Hidden/Ray Marching/Render Front Depth");
    renderBackDepthShader = Shader.Find("Hidden/Ray Marching/Render Back Depth");
    images = new DicomFile[250];
    hasFiles = false;
    hasMesh = false;
    getFiles(Application.streamingAssetsPath);
}

// Use this for initialization
void Start () {
    //GenerateMesh();
    oldOpac = opac;
    oldCol = col;
}

// Update is called once per frame
void Update () {
    if (oldOpac != opac || oldCol != col)
    {
        GenerateMesh();
        oldOpac = opac;
        oldCol = col;
    }

    if (!hasFiles) return; //Check if DICOM files have been processed
    if (hasMesh) return; //Check if Volume has been created
    GenerateMesh();
private void GenerateMesh()
{
    Texture2D slice;
    hasMesh = true;

    //create volume
    volumeBuffer = new Texture3D(volumeWidth, volumeHeight, volumeDepth, TextureFormat.ARGB32, false);
    var x = volumeBuffer.width;
    var y = volumeBuffer.height;
    var z = volumeBuffer.depth;
    var offset = (slices.Length - 1) / (float)z;
    var volumeColors = new Color[x*y*z];
    var count = 0;
    var floatCount = 0;

    //iterate over all the slices
    //each iteration map pixel into volume
    for (var k = 0; k < z; k++)
    {
        floatCount += offset;
        count = Mathf.FloorToInt(floatCount);
        slice = new DicomImage(images[count].Dataset).RenderImage().AsTexture2D();
        slice.Compress(false); //Needed to same memory
        for (var i = 0; i < x; i++)
        {
            for (var j = 0; j < y; j++)
            {
                var ind = i + j * x + (k * (x * y));
                Color pix = slice.GetPixelBilinear(i / (float)x, j / (float)y);
                if (pix.b < 0.2f || pix.g < 0.2f && pix.r < 0.2f) //Consider switching to or's
                {
                    //continue;
                    pix.a = 0.0f;
                }
                else if (pix.b < col || pix.g < col && pix.r < col) //middle intensity settings
                {
                    pix.a = opac;
                    pix.b = Color.red.b;
                    pix.g = Color.red.g;
                    pix.r = Color.red.r;
                }
                else if (pix.b > 0.72f || pix.g > 0.72f && pix.r > 0.72f) // high intensity settings
                {

pix.b = Color.yellow.b;
pix.g = Color.yellow.g;
pix.r = Color.yellow.r;
}
volumeColors[ind] = pix;

private void OnRenderImage(RenderTexture source, RenderTexture destination)
{

    //save volume to be rendered
    volumeBuffer.SetPixels(volumeColors);
    volumeBuffer.Apply();

    //Save to texture
    _rayMarchMaterial.SetTexture("_VolumeTex", volumeBuffer);
    hasMesh = true;
}

    //get texture
    //set scaling
    //get volume target
    //render saved volume in the volume target
    clipDimensions = new Vector4(xclip, yclip, zclip, 0);
    var width = source.width / downscale;
    var height = source.height / downscale;
    var volumeTarget = RenderTexture.GetTemporary(width, height, 0);
    if (_ppCamera == null)
    {
        var go = new GameObject("PPCamera");
        _ppCamera = go.AddComponent<Camera>();
        _ppCamera.enabled = false;
    }
    _ppCamera.CopyFrom(GetComponent<Camera>());
    _ppCamera.clearFlags = CameraClearFlags.SolidColor;
    _ppCamera.backgroundColor = Color.white;// was white
    _ppCamera.cullingMask = volumelayer;
    var frontDepth = RenderTexture.GetTemporary(width, height, 0, RenderTextureFormat.ARGBFloat);
var backDepth = RenderTexture.GetTemporary(width, height, 0, RenderTextureFormat.ARGBFloat);

_rayMarchMaterial.SetFloat("_Dimensions", slices.Length);

// Render depths
_ppCamera.targetTexture = frontDepth;
_ppCamera.RenderWithShader(renderFrontDepthShader, "RenderType");
_ppCamera.targetTexture = backDepth;
_ppCamera.RenderWithShader(renderBackDepthShader, "RenderType");

// Render volume
_rayMarchMaterial.SetTexture("_FrontTex", frontDepth);
_rayMarchMaterial.SetTexture("_BackTex", backDepth);
_rayMarchMaterial.SetVector("_ClipPlane", Vector4.zero);

_rayMarchMaterial.SetFloat("_Opacity", opacity); // Blending strength
_rayMarchMaterial.SetVector("_ClipDims", clipDimensions / 100f); // Clip box

Graphics.Blit(null, volumeTarget, _rayMarchMaterial);

_compositeMaterial.SetTexture("_BlendTex", volumeTarget);
Graphics.Blit(source, destination, _compositeMaterial);

RenderTexture.ReleaseTemporary(volumeTarget);
RenderTexture.ReleaseTemporary(frontDepth);
RenderTexture.ReleaseTemporary(backDepth);

private void getFiles(string myPath)
{
    DicomFile image;
    DirectoryInfo dir = new DirectoryInfo(myPath);
    FileInfo[] info = dir.GetFiles("*.dcm*");
    FileInfo f;
    int j = 0;
    for (var i = 0; i < info.Length; i++)
    {
        f = info[i];
        string _f = Path.Combine(Application.streamingAssetsPath, ":" + f);
        if (_f.Contains(".meta")) // Running in unity creates .meta files for every file
        {
            continue;
        }
    }
image = DicomFile.Open(_f);
images[j] = image;

j++;  
if (j >= images.Length)  
{  
    break;
}

hasFiles = true;
}  
}