CMPT 371 – TEAM 3

Programmer's Documentation

Virtual Reality Medical Imaging Software with Luxsonic Technologies Inc.

April 6th, 2016

Introduction

This document outlines the necessary information required for other programmers to pick up and continue developing the Virtual Reality Medical Imaging Software developed by CMPT 371 Team 3 and put forth by Luxsonic Technologies Inc. The project consisted of eleven members from Dr. Nathaniel Osgood's computer science class. All features delivered with this project will be outlined below, as well as ways to improve and continue working on them. Additional features that were not implemented in the final build will be explained in detail. While developing these features, several dead-ends, issues and failures were discovered. Documenting these has allowed us to provide future developers insight into what avenues of development to avoid.

System Specifications

In order to run both the executable program and the software, several programs and specifications must be met.

Hardware specifications

- Graphics Card: NVIDIA GeForce GTX 1070
- Hard drive space required: Minimum 8GB
- Required USB ports: 3 x 3.0 USB ports
- HDMI port: 1 x directly to the graphics card
- Oculus Headset
- Left and right Oculus controllers
- 2 sensors required (one for controllers, one for the headset)

Software specifications

- Unity 5.5.1f1
- Operating System: Windows 7 or higher (recommended Windows 10).
- Oculus Software Development Kit

Licenses for Libraries

For our program to work, a few external libraries were used. All libraries that have been incorporated into our project are free to use in projects that can be monetized.

Fellow Oak DICOM

While not a feature we developed from scratch, DICOM parsing is an important part of our system. Due to time constraints, the Fellow Oak by Cureos was downloaded from Unity's asset store. It allows the reading and writing of DICOM files with uncompressed images. It works with Unity to retrieve information from DICOM files needed to display to the user.

The library is licensed under the Microsoft Public License (MS-PL).

Full documentation can be found at their API documentation.site
https://fo-dicom.github.io/html/632e5303-a1e0-492f-8f6a-8b78e9037c40.htm

Unity 5

The project utilises the Unity 5 Engine heavily. The project has been started on the Free license for Unity. Information on this license can be found at the following links:

http://download.unity3d.com/unity/licenses

https://unity3d.com/legal/terms-of-service

Project Organization

The Unity project is contained within the repository in the 'Luxsonic Project' folder. It contains the following folders:

- Assets - Contains the major pieces of the project, including the code developed by the team.
- ProjectSettings - Contains setting files for Unity. Generally, these files should not be modified directly; project attributes should be changed from within Unity.
- Packages - Contains downloaded packages and libraries for the project; in this project, the major packages added are NUnit and Fellow Oak DICOM.

The code base for this project is contained within the 'Luxsonic Project/Assets' folder. This folder contains the following folders:

- 'DICOM Converter' - Contains the compiled executable for the DICOM Decoder used to decipher DICOM files.
- 'OVR' - Contains all of the Oculus Rift assets required for development with Unity.
- 'OvrAvatar' - Contains the prefabs necessary to display the avatar hands in the workspace.
- 'Plugins' - Contains the plugins we have chosen to use for the project, specifically the OVRGamepad bundle.

- ‘Prefabs’ - Contains the saved GameObjects we have created for the workspace such as the Copy objects, Display, and Dashboard, along with any other objects we required for the scene.
- ‘Resources’ -Contains folders for our custom shaders, materials, and images that have been created for use with the prefabs.
- ‘Scenes’ -Contains all of the Unity scenes we have created for the system.
- ‘Scripts’ -Contains all of the C# scripts needed for the program to function. These are the scripts that we attach to the prefabs to give them functionality.
- ‘UnityTestTools’ -Contains the NUnit Testing framework, along with all of our testing scripts in the ‘UnitTesting/Editor’ folder.

Current Features

File Browser

Current State:

Upon pressing the load button in the Dashboard, a file browser will be brought up for the user to search for DICOM files on their computer. The file browser currently works by displaying interactable buttons for each file and directory in the current directory. Each of these buttons stores a string representing a path to the specified folder or file. Selecting any file button will retrieve the file at that location and send it to the DICOM Converter. Selecting a directory button will call functions to erase the current set of buttons and replace them with those of files and directories in the new directory. The parent directory is always stored as a path in the back button. The current directory will display scrolling buttons that will move the list of directory and file buttons up and down. Any that fall outside a specified range will be hidden, showing only the buttons within the user’s field of view. The file browser can also load entire directories of DICOM images, where it will create a coroutine for each file (to reduce latency) and load them in from the converter.

Ideas For Improvement:

There are several areas for improvement in the file browser that we did not have the time to enhance. Although the file browser can take in files of any type, it does nothing to files that are not DICOMs. It would be beneficial if it had the ability to take in and load non-DICOM images. It then be necessary to have checks made before calling the DICOM converter to see if the image is not DICOM, and handle the image appropriately.

It would also be good if the file browser displayed an indication that files were being loaded. Since it takes time for the program to load a selected image onto the display, the user may think that nothing is occurring and select the file again. Adding a check to prevent the user from selecting a file button more than once would help with this.

Another feature that would improve the visual representation of the file browser would be to have the scrolling be a gradual motion. Currently when scrolling, files and directories jump to their next position instantaneously. Unity has functions that allow objects to move towards a position over a specified period of time. Applying these functions to scrolling would enhance the visual representation of the file browser.

DICOM Image converter

Current State:

The DICOM Image converter is an external compiled console program we made as a work-around to the limitations Unity 5 has for some JPEG encodings. Its sole purpose is to extract images from DICOM files, and save them as JPEGs in a particular path. It takes in 3 arguments, and will only function when all three parameters are sent in, and when they are all valid. The parameters are all string arguments: either an “f” or a “d” for whether the program is converting a single file or an entire folder, the target path for conversion, and the destination path for where the extracted JPEGs should be saved.

Ideas For Improvement:

In order to make it work better with the internal DICOM Information Parser, it would be better and more simple if the external program did not have two modes, and would only convert individual files. That way images can be loaded individually in tandem with the DICOM Information Parser, making it easier to pair an image with it’s information.

Display

Current State:

Currently the Display has functionality to show up to three images to the user at a time. The user can scroll through these images to view any image currently loaded into the workspace via the left and right scrolling buttons displayed when three or more images have been loaded. The Display is automatically updated when the user loads a new DICOM file. Each image in the Display is accompanied by patient information that is attached in the DICOM file. The sizes of the Display images may be tweaked in the Unity editor, along with the number of images that are shown in the display.

Ideas For Improvement:

The Display may benefit from a more natural scrolling technique, such as a gesture and an animation that gives visual feedback to the user when the images are scrolling. This would eliminate the clutter of the scrolling buttons in the space, as well as give a more comfortable and natural experience to the user.

Another feature that may benefit the Display is the ability to create Copies from it, rather than creating copies from the tray. The user would use a ‘laser pointer’ to point at the image in the display that they want a copy of, and select a button on the Touch controllers to activate that Display image. This would create a copy of that image just as the Tray does now. This may be a more natural way of creating copies as the Tray can be small and finicky. The Tray would then only be used for navigating the currently loaded images in the workspace.

Dashboard

Current State:

The Dashboard currently contains two panels with buttons that the user can interact with. The menu panel contains the Load, Quit, and Minimize buttons. The user can select Load to bring up the file browser to load a new DICOM file into the workspace. The Minimize button will hide all items in the workspace, except for the Copy objects. The Quit button will terminate the application.

The Copy button panel contains buttons used to manipulate the properties of the images in Copy objects. These buttons include: Brightness, Contrast, Resize, Saturation, Invert, Zoom, Close, Select All, Deselect All, and Restore. The Resize, Brightness, Contrast, and Saturation buttons all have similar functionality.

When pressing one of these buttons, the selected button will depress and the properties of any selected copies can be changed by moving the right thumbstick left or right. The Zoom button is currently not implemented and will have no affect to the workspace. Invert will invert the colours of the images on the selected copies. The Close button will close any of the selected copies from view. The Select All and Deselect all buttons will select all copies in the scene, or deselect all selected copies in the scene, respectively. The Restore button will return all selected copies to the state they were loaded into the workspace.

Ideas For Improvement:

Currently the Dashboard feels a bit clunky and could be improved on. Replacing the Dashboard panels with a ‘quick select’ menu that is mapped to the hands of the user may make the Dashboard more ergonomic. The menu would be invisible to the user until a button on the Touch controllers is pressed, at which point the options on the dashboard would be presented as a wheel around the user’s hand. The user would use the thumbstick to select the option they would like and the menu would disappear. This would make the workspace very clean as virtually no buttons would be taking up space. This would also provide a more comfortable experience for the user as they would not constantly need to reach out to press buttons.

Image manipulations

Current State:

The selection and grab features allow a user to move and select images within the virtual reality environment. A user selects images by making the point gesture and touching the desired images. A user grabs an image by making the grab gesture and touching an image.

To improve selection and grabbing accuracy, two colliders were placed within the hand, one along the index finger for pointing and one within the palm for grabbing. When the hand collides with the image while making the grab gesture, the image is made a child of the hand in the Unity hierarchy and moves with the hand until the gesture is stopped. When the hand collides with the image while making the pointing gesture, the image is highlighted by setting the child plane behind the image to visible and by setting the image to selected in Copy.cs and Dashboard.cs so that manipulation can happen on those selected images. The image is unselected with a second collision while using the pointing gesture.

Image manipulation can be achieved by using the set of manipulation buttons that are located in front of the user. These buttons are labelled by the manipulation they perform to an image, such as brightness, contrast, invert, resize, and saturation. Brightness, contrast, invert, and saturation manipulate the images by the use of Unity shaders. Shaders were chosen as the method to manipulate images because they render the manipulated image in a fast and efficient way. The shader used is called ImageEffects and can be found at Assets/Resources/Shaders/ImageEffects.

A user applies one of these manipulators to images by selecting the desired images and then pressing the desired manipulator buttons. The copy script will then set the corresponding attributes within the shader, which will render the image with the desired manipulation.

Ideas For Improvement:

Selection and grabbing images are a core part of this program, so the accuracy of these features are important. Two colliders were added to the hand to improve accuracy, though it caused other problems such as images/buttons being selected twice because they would collide with both colliders. As a result, extra checks had to be implemented within the scripts to make sure this double selection doesn’t happen.

An alternative solution to having two colliders and writing extra checks in code, while still keeping hand gesture accuracy, would be to have only one long collider within the hand that is the width of the index finger, which extends from the tip of the index finger to the middle of the palm of the hand. This would allow a point to still be accurate as it is the same width of the pointing finger, and still allow images to be grabbed around the palm of the hand.

The process to manipulate an image can be improved. Having to select buttons to manipulate an image causes the user to stay in a stationary place that allows him/her to reach the buttons. This replicates the work environment of a radiologist at a computer, which defeats the purpose of having an interactive, mobile system.

A suggestion to solve the issue is to have manipulation done with the Oculus Rift controllers . This could be implemented by pressing a couple buttons on the controllers that pops up a selection menu where the user can then select the manipulator they want with the controllers. This would allow the user to use the whole virtual reality space to place their images, with no restrictions on where they sit/stand. It also would improve the user experience because they would not have to make big arm movements so often. This corresponds with the improvements suggested for the Dashboard above.

Gestures

Current State:

Currently, the only gesture that is functional but not implemented is the ability to resize images with hand movements. This feature is located in the ID5_gestures branch. The feature will currently take the start positions of the controllers. Whenever the user holds the buttons on both controllers down, they enter a resize state where the change in position of their hands will be used to determine if the image should get bigger or smaller. The image will get bigger if the distance between the user's hands is greater than their initial start position. If the distance is smaller, the image is shrunken.

The current issue with this feature is that sometimes the user will have to cross their hands in order for them to shrink an image. Further investigation into what causes this state is required.

Ideas For Improvement:

Bug fixing on the feature itself is still necessary. We also had plans to implement a 'swipe' function which will allow the user to move through display images as opposed to clicking the left and right buttons. Work on this has been done, but unfortunately did not make it into the final product due to an issue where swiping would swipe the images too fast.

Features For Future Development

Dictation

While viewing cases, radiologist will typically either dictate or write up reports on their observations and assessments. Without a keyboard (which is what most use), recording information becomes quite difficult. Large virtual keyboards are currently cumbersome in virtual reality, and small keyboards require more subtle movement that virtual reality is still poor at handling. So to deal with recording information, dication is the only realistic option.

Dictation was labeled as one of our “nice to have” features, and was seriously considered being integrated as part of the system. Some preliminary research was spent into investigating its feasibility, and we had determined it would be the next feature in line to be implemented. Due to delays and time constrictions, this feature was not attempted. The Oculus Rift is equipped with a microphone so recording is feasible. Additional dictation software that allows the conversion of text to speec could be looked into.

3D Models

Ideally, 3D medical imaging would be one of the most useful applications of this software. It would make the best use of the 3D space provided in VR, and would make the software more engaging overall.

We have been able to import 3D models in the “.obj” format, and have been able to manipulate them in the workspace similarly to how 2D images work. The main issue stems from the actual conversion from the series of images to an “.obj” format, or having some way to convert the series of images to the 3D mesh in Unity.

Save Workspace

As the user makes changes to images and their workspace, it may be useful for them to be able to save the current configuration of their space. This feature would ensure that the user could leave the virtual reality without risk of losing progress on their work.

Implementing this feature would require a way of representing the current state of the workspace, including all image settings and positions, in such a way that allows it to be saved to disk. For example, this could be represented as a collection of tuples storing image sizes, positions and rotations. The file would then need to be able to be loaded in by the software to allow the user to continue where they left off. There is currently no infrastructure built in for this feature.

Locking Grid

Currently, users are able to grab copy images and place them wherever they want in any orientation. While most user testers liked this feature, there were a few who did not. Those that didn't said they preferred the idea of grabbable images locking to a position around them when they let go of the image. To address this issue, we came up with the concept of having an invisible dome around the player. The dome would contain 'anchor' points spaced out around the user. When an image is brought close to one of these points and the user lets go of the image, it will snap to the closest anchor and orient itself to face the user. This may require a button press to toggle on and off, to prevent accidental anchoring of images.

Connect To PAC Server

While this was a requirement laid out to us by Luxsonic, we decided to table this feature due to time constrictions. Currently, the user can only access and view DICOM files that are currently on the computer the program is run on. It would be a significant advantage if the program were able to connect to the PAC server and access any DICOM files the user has authorization to view. Due to the potential legal issues and time consumptions such a feature would have, no progress or ideas were attempted to implement this.

Defect Summary

Throughout the project defects were discovered through the processes of user testing and bug parties. These issues have been entered into the issue tracking database provided by GitHub. Issues have been tagged with relevant labels describing various attributes such as severity. The nature of this system yields four major categories of issues: Hardware, Usability, Refactoring/Best Practices, and Unity-related issues.

Many of the issues discovered during this project were related to usability. The VR aspect of the system poses many new challenges for creating a natural, seamless experience for the user. These issues have been recorded by user testing.

Hardware issues are issues that come with the Oculus workstations required to run the software. There are few of these issues, but the newness of the Oculus Rift comes with unknown bugs and issues.

Refactoring/Best Practice issues are issues related to optimizing code and keeping a clean code base that is easy to understand and build upon. These issues arise from misunderstanding of design as well as the learning curve required to develop for the Oculus.

Finally, Unity-related issues are issues that are inherent to Unity and the versions of the packages we are using to develop with Unity. These include issues with testing frameworks and continuous integration servers. These issues are generally out of the scope of this project, but still have great effect on the functionality of the system.

Technical Difficulties Encountered

Coming into this project, no members of our team had any experience dealing with virtual reality or creating 3D objects. Most of us had some experience coding in C# and Unity, but it was strictly limited to 2D games. So learning how to use virtual reality was a huge hurdle. The following section will outline the difficulties we have faced and explain what we did to work around them.

Graphical User Interfaces

We needed to know which packages from Unity were required to run and interface with the virtual reality software of the Oculus Rift. It was also important to know which features built into Unity were not compatible with the Oculus software.

Most notably, we discovered that the built in graphical user interface system in Unity is not compatible with virtual reality equipment. The problem stemmed from the fact that the Oculus uses two cameras to render images to the wearer. Unfortunately, GUI interfaces in Unity using the onGUI functionality are made to work exclusively with one camera. While GUIs will render when running the project in the editor, it does not work while viewing from the actual headset.

Canvas elements are a more appropriate fit for VR projects, but must be positioned carefully in World Space; Canvases can not be positioned in Screen Space-Overlay mode in VR.

Hardware difficulties

In addition, we needed to learn how to install, hook-up, and troubleshoot any issues encountered with the Oculus Rift. Unfortunately, we encountered several difficulties while using the hardware that was at our disposal. One of the workstations used for development would throw built-in assertions of Quaternion Matrix Errors whenever images were grabbed. Buttons would randomly enlarge or get contorted, as well as the image rendering of the user’s hands. Initially we were not sure if this was due to problems in our code, or if it was something wrong with that particular computer or Oculus Rift. We did, however, investigate the matter by conducting some tests to determine the source of the problem.

After swapping which machine ran which Oculus, we determined that the source of the problem was indeed the actual computer. Several of the problems were able to be replicated on the other computer by intermittently obstructing the sensors. This lead us to believe that the flow of data between the sensors and the computer would get frequently interrupted.

Another hardware issue we had which limited our time significantly was our inability to commit and push from the computers at our disposal. All computers at the university have an educational license attached to their Unity versions. This will prevent any product produced off of these computers from being monetized in the future. Luxsonic was kind enough to obtain their own hardware for us to use for the last couple weeks of development. This allowed us to directly update and commit to our repository without any repercussions.

OVR Shader Assets

The OVR package required for developing comes with shader assets used to render images to the screen. During development, we had issues with one of the shader assets being compiled for the wrong configuration of GPU. As the Oculus tools for Unity are quite new, we had a difficult time finding out what could fix this issue. We later discovered that we could edit the shader code to compile it for the correct GPU settings.

Image Manipulations

When we began implementing the brightness and contrast features, we initially started by using unity lighting objects to adjust the images. We soon found out that this is an insufficient way to manipulate images as it does not properly change the actual properties of the image. We redesigned our implementation to work with Unity shaders, which run GPU specific code to specify how to draw an image to the screen. We implemented functions in a shader to modify the brightness, contrast, and saturation in the ImageEffects.shader file. These not only provide an accurate way of manipulating the image, but also an efficiently optimized piece of code that runs specifically on the GPU.

Usability

Virtual Reality comes with the inherent difficulty of usability. Creating a natural, elegant, and simple to use interface is more difficult than most applications. Much care is needed to ensure all interactable objects are comfortably within the reach of the user. Health risks such as eye strain and motion sickness also play a big role in designing the UI. Many hours were spent tweaking the positions of the UI to ensure each button is in a comfortable place that is easy to access.

The general rule of thumb in VR is to create a sphere around the user that is centered at the front of their body. It is important to keep all UI elements within a small turning radius so the user does not need to strain their neck to view pieces of the UI. The ability to look around in VR is useful, but must be used sparingly to provide a comfortable experience to the user. The UI is a difficult part of the design process as you must account for the entire body of the user, rather than just the screen they are looking through.

Unit Tests

Due to the underlying testing framework for Unity, some of the features we have unit tests for do not run with the editor tests runner. These issues come up when using some of Unity's built-in functionality such as the SendMessage() method for process communications. We have written the unit tests for these functions, but have ignored the ones that fail for these reasons. All tests that fail for these reasons are marked as ignored with a description of the issue pertaining to that test. When running the tests in EditorMode, all tests pass, but the software cannot be run in editor mode regularly as it will edit the saved prefab objects permanently. We currently see no way to patch this issue and as such will leave the affected unit tests marked as ignored.