CS3249 User Interface Design

**Final Report**

**The Microscope**

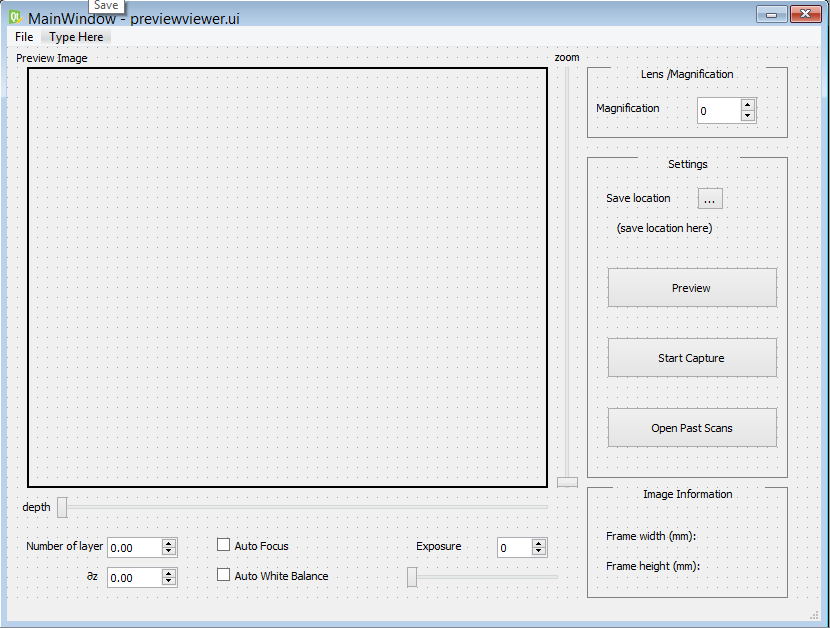
Team QX

Kit Wei Min

Wong Wei Jie Bob

Oh Qi Xuan

# GUI Design



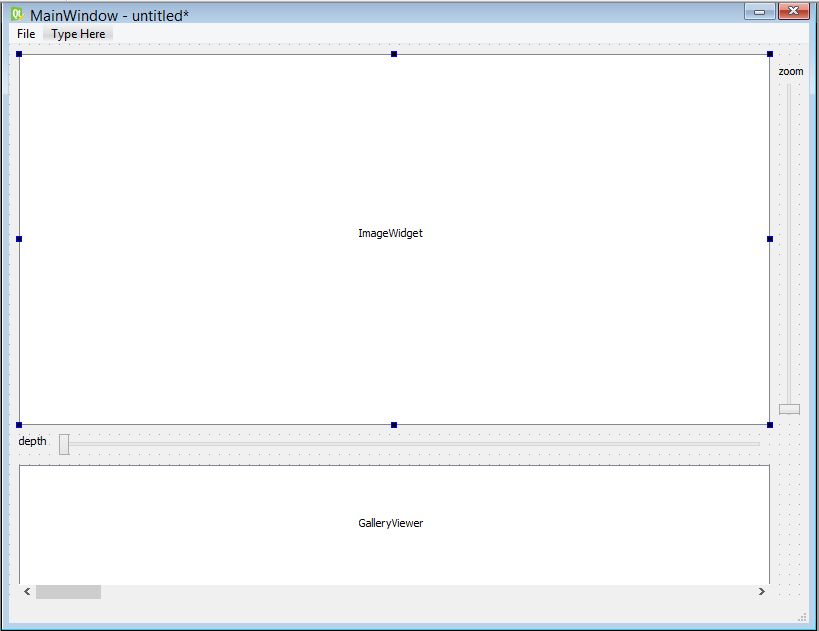
Based on the initial microscope design from tutorial 1, we are proposing a revamped GUI with several elements that emphasize aspects of the program important to the user, and de-emphasize lesser-used but still important elements.

First of all, instead of a large preview button, a frame intended to contain the preview image takes up most of the space. This frame updates only when the Preview button is pressed, to allow the user a fixed frame of reference when adjusting settings. We believe this is the main focus of the application and should be sized accordingly, as opposed to being assigned to another button as in the prior design. Allowing the frame to be in the same window as most of the application’s options enable fast review of changes. The frame is positioned on the upper left, as this allows it to be viewable easily at eye level as opposed to being positioned lower down, which may necessitate looking down at a slight incline and aggravate neck strain and user irritation.

Magnification is located on the top right of the GUI, as an important but infrequently used component. Though a slider may intuitively be an option, we believe that this would only introduce unnecessary clutter to the interface, as most uses of microscopes have different magnification options purely for versatility in perusing different samples, and do not require fast iteration through different magnification options. A spin box is sufficient to both allow modification of the magnification and display it.

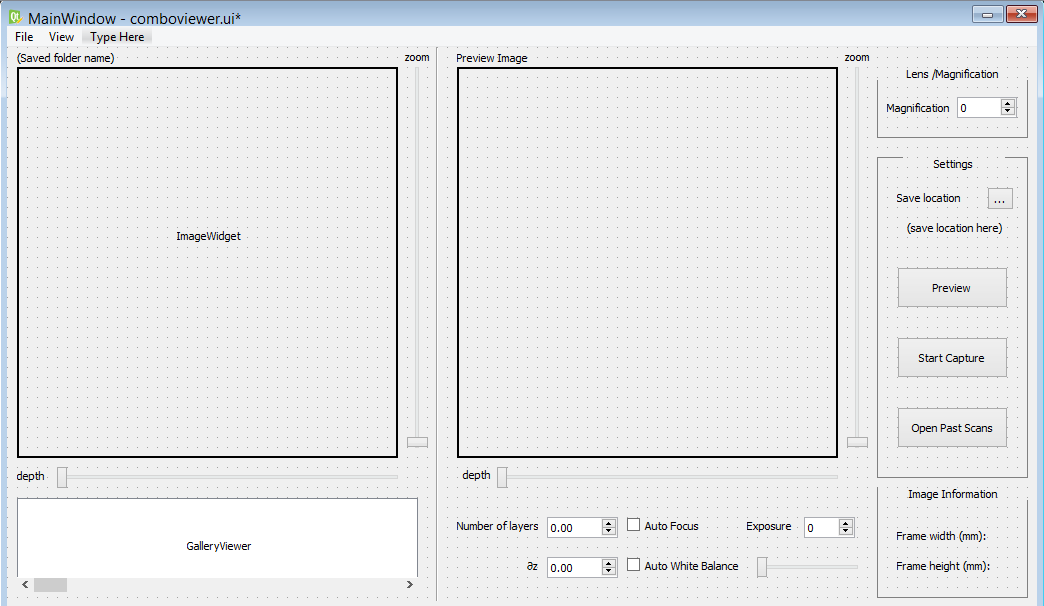
Settings are located below magnification, and contain options for adjusting the image save preferences. These include auto focus, auto white balance, exposure and the number of layers. To allow for a more intuitive adjustment, we have also provided a slider for modifying the exposure. While modifying the settings, the user will see the changes reflected in the preview image shown on the left, giving them more ease of use.

Buttons for Preview, Start capture and Open past scans are located at the bottom right hand corner, as they are the last things that the users will look for when they use the program. The Preview button updates the preview image with a new static image for users to peruse; selecting Start Capture opens up a dialog that informs the user that the scan is ongoing; selecting Open Past Scans opens the file dialog for the user to select the folder storing the scans that the user is interested in. The GUI will then be updated to a new view as shown below.



The main viewer now occupies the bulk of the window box. Sliders are used to zoom in on the image and to scroll through the different layers, though for advanced users, controls using the mouse and keyboard will be implemented as well. Users can also get a preview of all the different layers with the photo gallery located below the main viewer, and can select a particular layer by clicking on the image as well. The scrolling of the photo gallery will be synced with the slider that controls depth, so that the user can have a view of the surrounding layers while scrutinising a particular area in the main viewer.

We realise that users might want to be able to do side-by-side comparisons between past scans and what they are observing real-time, and thus we are offering a third view, with both views combined and placed alongside each other for easier comparison. This is shown below.



# Software Architecture

# softwareArchitecture.png

The ImageWidget is a common widget used by both the PreviewViewer and the ImageViewer. It communicates with its respective parent (and sibling) widgets through QT's slots and signals, allowing for loose coupling between the components. For instance, the communication between ImageWidget and GalleryViewer in ImageViewer is accomplished by the widgets emitting a signal when the depth is changed. As they are both parts of the ImageViewer widget, this widget in turn facilitates this communication between the widgets, therefore avoiding tight coupling.

FileHandler handles all files used by our program, and is an interface to interact with the files in the drive. This allows for changes to be made to the type of files that the Microscope can read without affecting the rest of the program, allowing the program to be extended without affecting the rest of the program.

Overall, the PreviewViewer is the main widget, which facilitates the using of the Microscope. It functions as a controller, controlling the user interaction with the Microscope and the rest of the widgets(ImageViewer, FileHandler).

# Implementation Strategy

PreviewViewer is the main window of the Microscope and will be shown when the program is first opened. QMainWindow is thus used for it. With the ImageWidget being the main point of the PreviewViewer, it is set as the central widget, and set larger in size. PreviewViewer interacts with the user via both preview and settings options. It also functions as the controller, and PreviewViewer is the widget that calls both the Microscope API to function the Microscope, and the FileHandler to communicate with the storage.

ImageViewer is a QWidget that can be shown and hidden from the PreviewViewer, and is called when a directory for viewing past scans is given. The second ImageWidget used here is a child of ImageViewer, and is independent from that in the PreviewViewer. GalleryViewer is another child of ImageViewer, used to show the list of the images of different depths in thumbnails. This is done via a QListWidget, which stores the image thumbnails as QIcons. ImageWidget and GalleryViewer interact via signals and slots, ensuring both low coupling and the synchronization of the showing of the image with its associated depths.

We use the original Qt FileHandler class which can also include custom formats, and therefore is able to serve the purpose of detaching the main program from the storage. It is also already implemented in Qt and simply requires the applying of the different appropriate functionalities.

The Microscope API is the list of API for adjusting the microscope. To simulate the microscope API, we created our own Microscope API that functions as a stub, returning pre-fixed values depending on the different functions called.