The Creation Of 3-Dimensional Models From Image Sequencing.

ALEXANDER FARRELLY

Centre Number:
Centre Name:
Candidate Number:
Qualification Code:

20th April 2020

Table of Contents

Table of Figures

Analysis

Problem Identification

Currently, to create 3D models using image sequencing you need to purchase expensive computer and specialist hardware, and this is typically beyond a student or hobbyist or Small Medium Enterprises (SME's) looking to investigate how to create 3D models, without the need for expensive hardware. The emphasis for applications of 3D modelling has recently shifted from measurements to visualisation. New communication and visualisation technology, driven by the use of smart phones, have created an important demand for photo-realistic 3D content. This has created a lot of interest for image-based approaches. Applications can be found in e-commerce, real estate, gaming, post-production and special effects and simulation etc. For most of these applications there is a need for simple and flexible acquisition procedures, and this stimulates the use of consumer photo or video (smart phones) cameras to create the 3D models.

3D modelling using image sequencing has many benefits. An example of this, is that a few years ago it was mainly used for "robot guidance systems and visual inspection. Nowadays the focus is shifting. There is more and more demand for 3D models in computer graphics, virtual reality, augmented reality, and communication".[1]

The characteristics of a computer system, that would be necessary for a solution, would be a computer with video hardware with image processing capabilities, any camera (e.g., smart phone) which can create a live feed to the computer and an adequate operating system for the software to run on. The issue does lend itself to a computational solution due to it using complex algorithms to complete each task. Describing the process in layman's terms, the solution would segment a video taken of an object by the user into frames, then it would take each frame and create a DDEM (dense depth estimation map) of it to create a 3D scaled image on your computer of the 'object' the video was taken of.

The Stakeholders

The customers and demographic for this software would include architecture firms, virtual reality developers and guidance system firms. Due to the limited range of the application of this system the stakeholders will be a delegate sample, ranging from those looking to use and manipulate the models (virtual reality developers) to those looking to only use the model for a reference (guidance system firms). With this sample we will be able to test it in different situations with different objectives.

The stakeholders for this software are mostly the firms or groups that I have already mentioned however there are also people from the educational sector and personal users who could be potential stakeholders.

Almost the entire amount of modelling software used for virtual reality are Computer Aided Design (CAD) software, which are time consuming and costly with some even costing upwards of $1,500 (unity 3D). This type of software applications lends themselves to be replaced/supplement by my software, as a means to implement a real-life object into a Virtual Reality (VR) game takes a lot of skill and time. With this software, you will also need is a camera and time to explain how to use the software and explain how it should be used. The stakeholder for the VR developer's demographic will be Jonathan Ruffles and James Heely. Jonathan is a competent game designer working on his own game "Decay" which uses Unity 3D extensively. James Heely also works alongside Jonathan in creating visuals for VR games.

[1] Marc Pollefeys , Maarten Vergauwen and Luc Van Goo, AUTOMATIC 3D MODELING FROM IMAGE SEQUENCES, 2000

Another possible type of application that would be suitable for this software, is terrain mapping for guidance systems and / or terrains for gaming sequences. While VR modelling focuses mainly on the accuracy of the look of the object, the mapping would focus on the accuracy of the proportions / measurement of the 3D model. The stakeholder who will represent this area will be Martyn Farrelly, Martyn worked for years for the Ministry of Defence (MOD) designing targeting systems and working with MATLAB which is a CAD software that can be used for creating mathematical models for use in the creation of 3D terrains.

Why is it suited to a computational solution?

This problem is well suited for computational techniques of finding and implementing a solution for a diverse number of reasons. My solution to solve the problem will be a computer program, developed using open-source libraries and software developed by myself, that uses an off the shelf camera (e.g., webcam, smart phone camera) to capture both 2D photos and video. The computer program will need to run on either a computer or phone, as both are required to capture the image and process it in the manner needed. There is no solution that would not require a phone or a computer.

The elements of the solution that are controlled by the user would be the video capture device which is used to collect data / information about the target object. The software would then subsequently process the video and then deconstruct, followed by reconstruction into a 3D image on a computer.

Computational techniques that the solution is well suited for:

Problem recognition:

The problem as a whole is finding a way of taking each frame and creating correct depth maps and layering each one onto the other. To do this one needs to be able to understand where to segment the video to create the best perspectives and how to layer the image back onto the depth maps without overlapping them to make the most accurate 3D image.

Problem decomposition

The problem of creating renderings of models from images can be broken down into a set of smaller steps. My initial thoughts of what these steps will be:

1. Using the camera take a video of the target
2. Segment that video into frames
3. Projective reconstruction of each frame
4. Self-Calibration to align the images
5. Matching the depth maps
6. Layering back on the image

Once I have accomplished a solution, the process of how to create renderings of the models from discrete images will be finished. My aim is that it should be done precisely enough so that it is not possible to notice any major differences from the original target.

Divide and conquer

The steps mentioned above, while looking relatively small are still incredibly hard to accomplish. The approach I will take, will be to implement a divide and conquer strategy. I will segment each of the steps into smaller and smaller steps until each step is laid out in its simplest form, making it easier to accomplish my task.

Abstraction

A digital image includes an enormous amount of data / information based on the resolution and pixelization of that image. Nearly all of this data / information will not be required for the earlier stages of my software development. My aim is to start with an image that is blurred coupled with a

black and white filter. Applying the filter will limit the disparity of colour change in a small space. I can then apply a "mask", and this will be applied to eliminate the noise associated with the image.

The output of my program will be an 3D object which can be manipulated on a computer, which means that overall, the software will provide a way of creating that image without requiring the use of any 3rd party CAD software. This should allow for faster and more realistic representation of a physical object. This could be used for more immersive augmented / virtual reality applications or for a better understanding of the environment for guidance systems / gaming scenes for more realistic models.

Interview

Interview questions

I have summarised the most important questions that I will ask each stakeholder, however during the interview I may ask them to provide detailed information about a particular answer to aid with my understanding of their responses / requirements. The purpose of these questions is to find the stakeholders views about the software I propose creating; how they could use it in their applications and if there's anything that needs to be a mandatory requirement in order for my software to be of use in their application.

Virtual reality developers

My questions for Jonathan Ruffles, and James Heely (they both represent the Virtual Reality developers) are as follows:

1. Are you satisfied with the usability of your current 3D modelling software for your games?
2. Do you try creating realistic objects for your game?
3. If yes, what was your experience creating these objects and was it difficult?
4. What do you think to having physical objects reconstructed in 3D without the use of a CAD software but 3D image rendering software?
5. How would you want the said software to be operated?
6. Is there anything essential / mandatory a program such as this need?
7. Anything else to add / important to your application of this type of software?

Questions 1 and 2, I will use to establish their history with designing real life objects. This is crucial because it is imperative to know if their view is based on prior experience or is not and so determine if I can use their responses to help develop my solution.

Question 3, I will use to explore any problems that they might have had in designing games with physical objects and if they have experienced issues that I may need to overcome with my solution. I would also need feedback on what makes the current software they use the reason why they had chosen it? Was it ease of use and intuitive or any other fit, form or functionality?

Question 4, I will use begin understanding and to inquire about my project and the software I would need to develop. I will also ask whether they are satisfied with the main concept behind my proposed solution and if I need to consider any other options in order that it would fulfil the majority of their requirements.

Questions 5 and 6, I will start to examine the user experience of the computer software solution as well as how they would want / like to interact with it. I will focus / concentrate on both the user-interface and the way to operate the software as well as the core application for 3D models using image sequencing.

The questions will help me understand and come to a conclusion as to whether the solution I have proposed covers the majority of the requirements needed for a solution to support VR developers and whether I may have overlooked any must have set of functionalities.

Guidance systems developers

My questions for Martyn Farrelly are as follows:

1. Are you satisfied with the usability of your current terrain modelling software?
2. Do you find creating terrain with this software challenging?
3. If yes, what makes it difficult?
4. Have you ever used 3D modelling from image sequencing before?
5. Do you think people like you would benefit from those types of systems?
6. How would you like that type of software to be operated?
7. Is there anything essential a program such as this need?
8. Anything else to add?

Questions 1 and 2 will allow me to establish Martyn's history with designing 3D terrains as part of projects he has worked on. This is important because I need to understand if his view is based on prior experience or is not.

Question 3, I will use this to explore any problems that he might have had with designing 3D terrains with the chosen software (MATLAB) and what makes the software easy and good to use.

Question 4, I will use this to investigate if Martyn has ever come across and used the software or methods that I am going to create as part of my solution. This is important to me as I want to determine what are the features of the software that are needed and what are the must have / good features and what could possibly be improved.

Question 5 asks if from a professional opinion if the software I am proposing could be used or even worth tailoring towards using as part of a suite of software to create 3D terrain maps.

Question 6 and 7, I will use to investigate the usability of my proposed software solution and how it could be used, focusing on both the user interface and the way to operate the software and understand how it could compliment the way off the shelf software products are used in a professional environment.

The questions will help me understand and come to a conclusion as to whether the solution I have proposed covers the majority of the requirements needed for a solution to support 3D modelling of terrain maps and how my software could be used alongside a third-party software product such as MATLAB. It will also help me determine whether I may have overlooked any must have set of functionalities for my solution.

Interview

Virtual Reality Developers

Jonathan Ruffles

1. **Are you satisfied with the usability of your current 3D modelling software for your games?**

   "No while Unity's usability allows for quite quick developing of a prototype of an idea. It has problems when trying to create any mesh with other 900 vertices which limits the amount of precision we can have for the final entity.**"**

2. **Do you try creating realistic objects for your game?**

   "Yes, in the game I'm currently creating -Decay-, I'm focusing on enhancing the immersion that VR games already have by making the environment as real as possible almost like you were there yourself."

3. **If yes, how was your experience creating these objects and was it difficult?**

   "My experience creating those objects wasn't fun or easy. Firstly, it is already extremely difficult to create a convincing 3D object using any CAD software let alone Unity 3D having a limit of 900 vertices. Even if that was not a problem, any more than a handful of these (close to realistic) entities, the game lags when it tries to render them all."

4. **What do you think to having physical objects reconstructed in 3D without the use of a CAD software, but a 3D image rendering software?**

   "If you removed CAD completely, would it not be a bit limited. For it to work you would need to still have some controls to edit the objects. It would be more of a gimmicky way to gather textures and entities for games and does not sound viable in terms of graphical design."

5. **How would you want the software to be operated?**

   "I want to be able to switch on your software and have it work right away without the necessity of massive amounts of calibration."

6. **Is there anything essential that a program such as this need?**

   "Consistency, for a program such as this you can't have massively varying results. You need it to have constant accuracy."

7. **Anything else to add?**

   "I have used a rendering software before, it was called Agisoft, and it was terrible. I needed to constantly calibrate the system and even with all the help they provided I still couldn't properly use it. It ended up being less realistic than the graphic design I could already create."

8. **Do you believe that the lack of accuracy allowed in Unity 3D takes away that immersion you should get with VR experiences?**

   "Yes, completely."

James Heely

1. **Are you satisfied with the usability of your current 3D modelling software for your games?**

   "Not particularly. I do not struggle with my current software, but it is quite limited when I'm trying to design large scale projects such as buildings. While for small scale projects such as characters it can allow you to do accomplish a lot, but in a game, there is a lot more than just characters.**"**

2. **Do you try creating realistic objects for your game?**

   "I don't try to; this is not because I don't want my games to be as realistic as possible but it's because I know it's almost impossible to create hyper realistic objects and characters in the any software made for VR graphics**."**

3. **If yes, how was your experience creating these objects and was it difficult?**

   James had nothing more to add, above and beyond what I have captured above.

4. **What do you think to having physical objects reconstructed in 3D without the use of a CAD software but a 3D image rendering software?**

   "I believe that would be extremely useful as long as it's simple enough to use, the benefit of such a system would be great as long as the User Interface (UI) isn't overwhelming with different options and parameters.**"**

5. **How would you want said software to be operated?**

   "I really wouldn't mind how its operated and navigated as long as it's simple as I mentioned earlier."

6. **Is there anything essential a program such as this needs?**

   "The software would need to be able to render small objects and massive objects such as buildings, sculptures, etc… as what's the point in having realistic chairs and tables if the room they're in look completely fake.**"**

7. **Anything else to add?**

   "No, I don't have anything else to add.**"**

Analysis of the VR developers' interviews

Neither of the virtual reality stakeholders were satisfied with the current capacity for creating realistic 3D models for their virtual reality games. Jonathan has used 3D rendering software like this before and he had some major concerns, most of them revolving around the constant need for calibration the previous systems required. With this in mind, the solution would need a simple and robust calibration system which is self-explanatory or at the very least has a help section to guide users.

Another problem identified by the VR stakeholders was the problem of real-world likeness, they both testified to the point that current graphic development software cannot produce true realistic 3D

rendered images. To combat this concern the solution will need to be able to consistently produce images with realistic qualities.

Guidance systems developers

Martyn Farrelly

1. **Are you satisfied with the usability of your current terrain modelling software?**

   "Yes, I'm more than satisfied with our current software. It is an expensive product (MATLAB) for a single user licence, but gives the required amount of functionality and use of libraries to create the digital terrains required"

2. **Do you find creating terrain with this software challenging?**

   "No, I don't, while it does take a prior level of knowledge of the system we use (MATLAB) to be able to use it correctly and effectively. To get an in-depth understanding how to use the software takes time and experiment it provides help / guidance that means you can get up and running quite quickly."

3. **If yes, what makes it difficult?**

   Martyn did not have anything to say on this topic.

4. **Have you ever used 3D modelling from image sequencing before?**

   "Yes, I have, specifically I've used "Degsin.land" which is based is a mobile device application. It did not offer much in the form of practical use. It needed a lot of practise before I could use it effectively and even after that it was still more of a novelty application."

5. **Do you think people like you would benefit from those types of systems?**

   "I believe we would, however only in specific circumstances as a means to create an initial view rather doing an in-depth terrain map. While a system like that would help for visualising terrain without the ability to manipulate the images it would not be too practical for replacing our off the shelf applications."

6. **How would you like that type of software to be operated?**

   "I would like such a software to be simplistic and minimalistic. I wouldn't want a system such as this to have too many features that overload the user when they interact with it, and I would want the system to have very basic and easy to grasp controls."

7. **Is there anything essential a program such as this need?**

   "As this program couldn't replace such things as MATLAB it would need to be simple enough someone completely new to the system could pick it up almost instantly, and it should be quick enough to where the use of MATLAB wouldn't be quicker than just using such software. Maybe a use would be to create an image library based on the program for use in applications such as MATLAB"

8. **Anything else to add?**

   "Martyn had nothing more to add."

Analysis of Martyn's interview

Martyn has used 3D rendering software before, his main observations that the software he had used in the past was very feature rich and took a long time to get a good understanding of. He added that the systems that he had used before needed to much practise until he began to see it work consistently. This indicated to me that the applications he has used were complex and required a lot of experience with the application as a lot of user guidance needed to be understood / read before and when using the application. My solution would need to remove / reduce this problem by offering up an extremely simplistic GUI and having as much guidance as possible, to make each use of the application, even if it's the first time using my application, and ensure its use is as seamless and successful as possible.

Synopsis of my Analyses

From all the interview data captured from both the VR stakeholders (Jonathan Ruffles, James Heeley) and the Guidance systems developer (Martyn Farrelly), I have established some common concerns about 3D image rendering software. The first common concern was the lack of help and guidance given to the users; this problem is easily solvable by adding a detailed explanation (in a simplistic manner) of what everything does in the software as well as designing and developing my solution to be intuitive and follow a set of easily definable steps to create the 3D rendered images.

My next main concern was the lack of realism highlighted by the VR stakeholders. To better identify what they mean by the "lack of realism" I will need to conduct research into different software made for 3D image rendering namely,"Display.Land" referred to by Martyn Farrelly and "Agisoft" referenced to by Jonathan Ruffles. After I have gained better knowledge about these stated software's I will be able to better answer this concern.

As I move forward with further analysis and design of my solution, I will focus on obtaining information regarding the major concerns I have highlighted, while also trying to establish any potential weaknesses and any major strengths from the proposed similar solutions.

Research

Existing similar solution

Display.land

Overview:

Display.Land is a proprietary application which is only available on mobile phones, the application allows users to take video of a surface, object, room, etc… and then the application turns that video into a 3d image. Display.Land also acts like a social media platform where users can share different 3D models with each other, also allowing people to like and share their favourite models with other users. The software requires an android or iPhone that can access the latest app store variant[2].

The User Interface:

Figure 3 - Display.Land GUI

Figure 1 - Start Screen

Figure 2 - Final Image

Figure 3 is the first GUI you first see when you load the application, as you can see the app has a few functions accessible from the home page; the social media aspects (like, comment, share and etc), the image that another user uploaded and finally the button to add your own 3D image.

Upon pressing this button, I was greeted with a new overlay as you can see in Figure 1. This GUI gives me a range of new options, a help function, an exit function, and a start function. After pressing the begin button the application started recording and created a pop up which told me to walk slowly around the object/place I wanted to be created in 3D which went away after ten seconds or so.

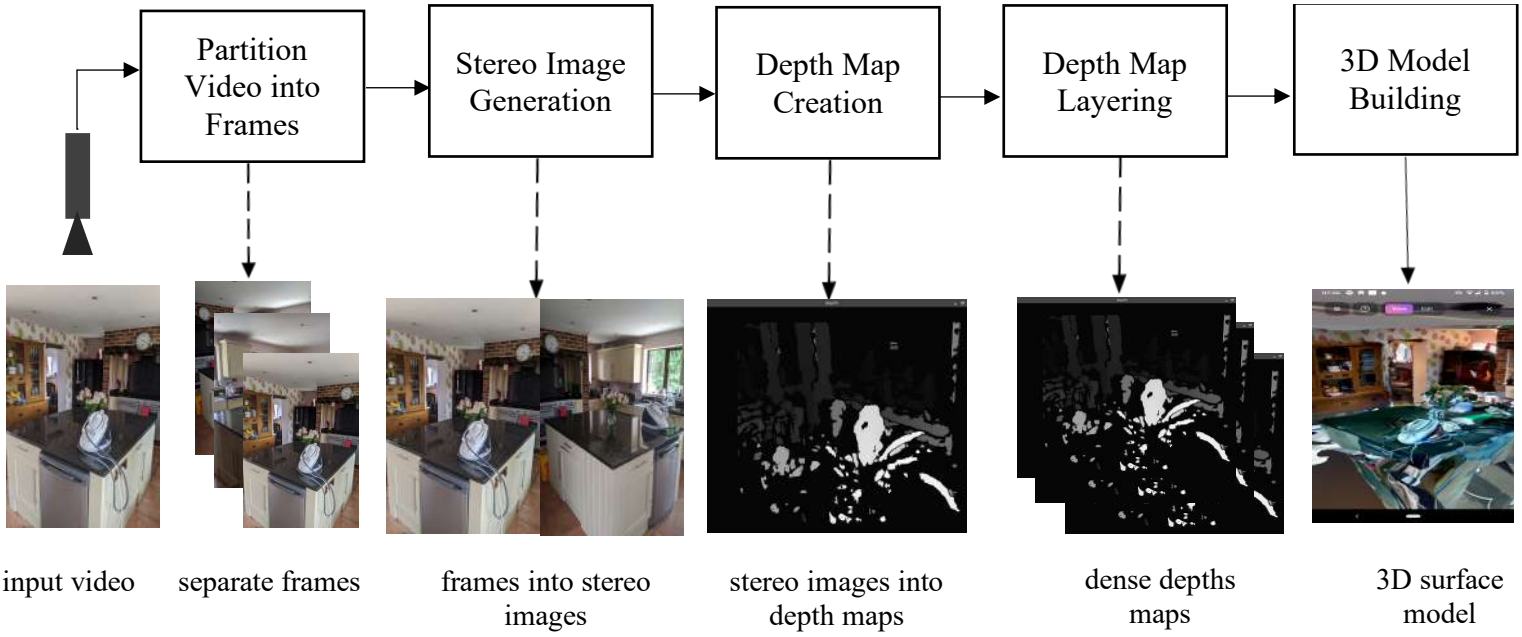
Whilst I walked around my kitchen (the object I wanted to recreate) the application periodically added little coloured dots around the room, this was to show me that the app is working and obtaining data to be used. After about 3 minutes of walking around the object, the application alerted me that I have entered the maximum amount data possible, it then took me to a loading page where it rendered all the data. I was stuck on this page for just under half an hour. After which it generated Figure 2.

[2] Note: The Display.Land app was sunset on 20th August 2020 and is no longer available for download.

I observed a few problems in using the application. Firstly, it looks nothing like my kitchen, plus there are a lot of voids in the image that leave blank spaces, and finally the textures seemed, on inspection looked to have meshed.

System / software components used by the application:

Figure 4 - Display.Land System Overview



Note: The camera is represented by the box and pyramid.

Partition Video into Frames:

The method of partitioning the video is the most simple and straight forward step of the process. The input video was filmed at 30 frames per second (FPS) (the default setting of the device I used), which means every second of the video is filmed for the application has 30 frames of data to use. The method I used to demonstrate how Display.Land could have accomplished this task was by using a python extension called open cv and creating this code:

```
import cv2
vforf = cv2.VideoCapture("mykitchen.mp4")
success, image = vforf.read()
count = 0
while success:
    cv2.imwrite("frame%d.jpg" % count, image)
    success, image = vforf.read()
    count += 1
```

After the code is executed it saves each frame to the default folder the code is in, here is the outcome, see Figure 5:

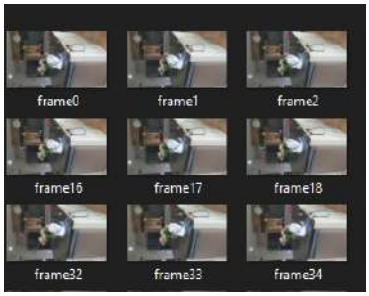


Figure 5 – Video partitioned into frames

Stereo Image Generation:

A stereo (stereoscopic) image is two images slightly offset from each other; this effect is used to mimic what our eyes do to perceive things in 3D. The difference between the two images or the disparity between them is what allows us to perceive its depth.

However, a stereo image to help perceive depth is not just getting two images with a disparity between the two, there is a lot of complex maths behind it. The main area of math involved in stereo images is Eipolar geometry. See Figure 6 which shows a basic setup of two cameras taking a picture of the same image (courtesy of OpenCV tutorials):

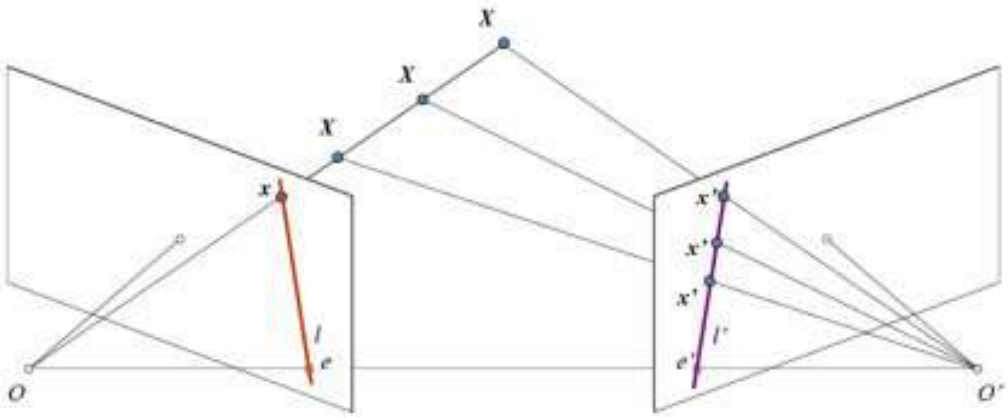


Figure 6 - Two Camera's taking a picture of the same image

Figure 6 shows a camera taking pictures from two different locations, the first location is labelled O and the second O'. If we were only using the first image taken at O, we would not be able to find any information about the depths of any parts of X as "every point on the line OX projects the same point onto the image plane. But now consider the picture from O' also. Now different points on the OX line project different points in reference to position O'. So, with these two images, we can triangulate the correct 3D points of an object"[3].

[3] Alexander Mordvintsev & Abid K, OpenCV-Python Tutorials Documentation, 2013

Depth Map Creation:

A "depth map" is an image that contains information relating to the distance of the surfaces of a scene from the place of the image being taken"[4].

The section above showed the basic idea of Epipolar geometry, it shows us if we have two images of the same scene with one being slightly offset from the other, we can gain depth information about that scene. Figure 7 is a simplified version of that diagram which has some mathematical equations and symbols which I will explain later:

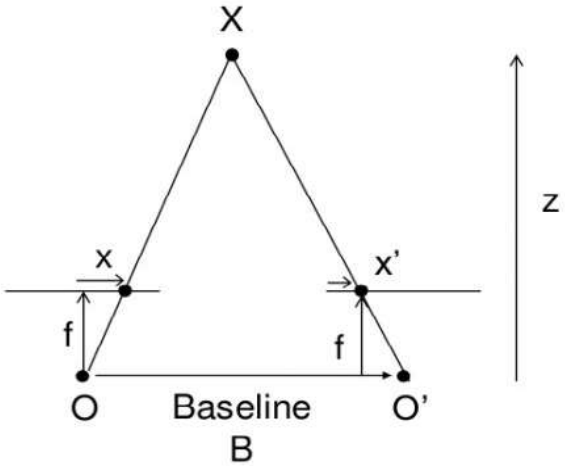


Figure 7 - Simplified Epipolar Diagram

Figure 7 is the same as Figure 6, however it is a top-down view of it. This allows us to have a simplified version that we can work with. Firstly, I've labelled different parts of the diagram: X is the object / surface that is being photographed, O and O' is the camera centres, f is the focal length of the camera, B stands for the distance between the two cameras (or more commonly referred to as the baseline), Z is the distance of the camera and the object, and finally x and x' are the distance between points in the image plane corresponding to the scene point 3D and their camera centre.

Using this diagram, we can simplify it to a singular equation for finding out the disparity of an image.

$$\text{Disparity} = x - x' = \frac{Bf}{Z}$$

This also shows us that the depth of a point in an image is inversely proportional to the difference between x and x'. This makes it possible to find the depths of all pixels in an image.

With the information of the depths of each pixel we use a method to visualise them, firstly we turn the image (usually the first image, so in this case O) into grey scale, then depending on how close part of that image is to the person taking the image the lighter the grey it is. For example, if something is far away from the image taker that part of the image would be black or dark grey, and if something was close to the image taker that part of the image would be a light grey and or white.

[4] Alexander Mordvintsev & Abid K, OpenCV-Python Tutorials Documentation, 2013

Depth Map Layering:

Depth map layering is a process that uses Epipolar geometry which we briefly touched upon in the stereo image generation section and another method called disparity estimating (which was based on the dynamic programming scheme).

The program is able to layer the depth maps by looking at the Epipolar lines and the scan lines. The scan line searches a small section of each image usual with a size of 5x5 pixels but can be up to 7x7 (these sizes are usually referred to as a kernel) the system looks for where the Epipolar lines and the depths in these images are about the same usually with a margin for error. Once the system finds two points which are the same it will warp the images to add the two depth maps together, so it creates one dense depth map.

3D Model Building:

In the section above, I discussed how the images were set into grey scale and the lighter or darker parts of the image are relates to how close or far away in the image certain parts are. This is used primarily for this step in the process rendering images.

Once a depth map of an image is created and the depth maps are matched and lined up together (as outlined in the previous section), we can then finally overlay the original image. Each pixel of the original image will now be laid over the new greyscale match images. The position of each pixel in its new Z axis will collate to the lightness or darkness of that pixel's position in the depth maps.

Once all the pixels of the original image are laid over the depth map, they finally have created a 3D rendered image which for Display.Land was Figure 2.

Parts I can apply to my solution:

Display.Land is extremely similar to the solution I am proposing, in that it uses a similar method I was going to use (refer back to Figure 4). However due to the intended target audience of Display.Land there are still some differences between our approaches. Display.Land target market is just the average mobile phone user which leads to most the functionality of the app being centralised around convenience and simplicity, while this is very useful in any app, it also comes with major drawbacks. The main drawback for me is, as you can see in Figure 2, the lack of control in the camera led to significant warpage and voids in the image, this is due to the program blending together point clouds which don't align as perfectly as they should, which comes from the great disparity in the change in the x, y, and z axis of the camera caused by the user. With all this in mind my solution will use a similar method to create the rendered images however will use a stabilising device to allow the best possible image acquisition.

Features of proposed solution:

Initial concept of my solution considering this research:

My solution will be an application that can utilise either one or two cameras. The user will be able to take images or upload a video from a camera device. My solution will ask the user to input different values for the different parameters required. Once all this data / information is gathered, the system will create a 3D rendering of the images using the terms of the parameters.

Limitations of my solution:

The primary limitation of this solution will be that it requires almost perfect control in the camera as any slight deviation can lead to the rendered images becoming warped and distorted as we

demonstrated using the Display.Land app. A potential way to get round this is to not to tailor the solution / software for use by a novice (e.g., someone with no detailed knowledge of 3D rendering). However, to create a solution for professionals who already understand how 3D rendering software works, is supported by the feedback I received during my interviews with knowledgeable stakeholders.

Needed Specifications

Software and hardware specifications

Hardware:

A computer with the ability of running the application, with basic input and output peripheral device – The program will utilise an off the shelf camera (smart phone or other) and will use computer image processing to examine the images it gathers and create the 3D models. A standard off the shelf computer will satisfy the requirement as no specialist functions or graphic cards will be required. Nearly all laptops and desktop computers have this essential characteristic. The basic input and output devices consist of; a monitor: for displaying the software, and a keyboard and mouse: for circumnavigating the software.

A camera / cameras– The camera / cameras shall be employed to take a video/picture of the desired target, which will subsequently be examined and processed to render a 3D image. The camera has to be of an adequate resolution (no lower than 480p), the lower the resolution the lower the quality of the rendering, this is because if the resolution is too low the depth maps created may be too noisy and incorrect.

Software:

Windows, Linux, or Mac OS – these are the main operating systems which support python, allow it to be run and OpenCV (the image processing library used to examine and inspect the images).

Python Interpreter – The code shall be coded within pythons IDLE, so a python interpreter is completely essential to execute the code.

OpenCV for Python – This is an image processing library which I will be utilizing. It enables the straightforward and effective examination of images. This is important for creating both the stereo images and depth maps.

Matplotlib for Python – This is to be used in the displaying of point clouds.

NumPy for Python – "NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, Fourier transform, and matrices" (sourced from w3schools). Which are all utilised within OpenCV's functions.

Tkinter for Python – This is a module used for displaying the graphical user interfaces and interacting with the user's documents.

Pillow (PIL) for Python – This module is used for displaying the images / video the user chose to render.

CSV for Python – This module will be used to interact with external databases to both open and save data to.

Stakeholder requirements

Design

Requirement	**Explanation**
Simple main window	This is to allow the user to be able to use the program with minimal assistance as the layout is simple enough to understand.
Self-explanatory features.	Intuitive features will remove the need for a help section/guide for those new to the system.
Straightforward calibration options / layout.	The calibration needs to be self-explanatory; this is to allow the user, not to have in depth knowledge in order to calibrate the system.

Functionality

Requirement	**Explanation**
The software should be able to take "still" images to be rendered.	This is to allow users to render 3D images of objects either too big for a video or when there are objects obstructing the view of the required object e.g., people, cars, etc…
The software should be able to take a video to be rendered.	This is to allow a quicker way of gathering data points for the rendering of an image.
The software should offer the user to change its calibration options.	This is to allow the user to create better renderings and allow them to tailor each rendering to create an improved final 3D model.
The software should allow the user to save and open created renderings.	This is to not make the user recreate each render every time they would like to see it, as to render an image is a lengthy process.
The software should allow the user to save and open calibration alterations.	This is to allow the user, to not have to constantly alter the calibration options and every time they close the software and allow, them to open preferred calibration options.
The software should allow the user to create an account.	This is to allow the user to save pre-sets and data against their account.
The software should allow the user to login to an account.	This is to allow the user to access the data they have previously saved against their account.

Hardware and software

Requirement	**Explanation**
Basic input and output devices: computer with a keyboard, mouse, and a monitor.	The user must have basic input output devices, to both allow them to circumnavigate the program but also to allow them to fully use all the different features the program will offer.
A device to take videos and / or images that are at least 480p in resolution.	The device will be used to gather the information about the desired target so that it can be rendered, the higher the device resolution the less noisy and incorrect the rendering will be.
Windows, Linux, or Mac operating system.	The following operating systems that allow python to be executed.
Python with OpenCV, NumPy, Tkinter, Pillow and csv.	The program will be coded in and run-on Python utilising OpenCV which uses NumPy. Tkinter will be used for user interfaces, which also uses Pillow. The database interactions will use csv.

Success Criteria

- A simple to use GUI:
  - 1.1 The user interface should include the following features via log-in / user creation:
    - 1.1.1 Select Images, Select Video, Render Image, close program and minimize/maximize program buttons.
    - 1.1.2 Calibration options: minimum possible disparity, maximum possible disparity, uniqueness ratio and focal length.
    - 1.1.3 Display the images/frames the user has chosen to enter.
    - 1.1.4 Menu options to save and open changes in the program's calibration, save and open renderings and log out.
  - 1.2 The registration window should provide the following features:
    - 1.2.1 An entry box for; their username, password, and a check password.
    - 1.2.2 A button for; registering their data and showing their password.
  - 1.3 The login window should provide the following features:
    - 1.3.1 An entry box for; their username, password.
    - 1.3.2 A button for; logging in, creating an account, and for showing their password.
  - 1.4 All windows must have large buttons and clear text.

2 A functioning account system:

2.1 A registration window which allows the user to enter their details and to create an account.

2.1.1 The details need to be saved to an external database, with a unique user key.

2.2 A login window which allows the user to login to their account.

2.3 The user should be able to access the saved data they have previously made.

2.3.1 They should be able to save pre-sets and renderings against their account.

2.3.2 They should be able to open pre-sets and renderings against their account.

2.3.3 They should only be able to access the data linked to their user key.

3 A non-technical calibration:

3.1 The calibration needs to be straightforward and easy to use for new users.

3.1.1 Each option needs to be clear on the main page.

3.1.2 Each option will have text to describe their function.

3.1.2.1 The text needs to be clear and concise.

3.1.3 Minimal on-screen options to eliminate complexity of setting the calibration and avoid confusion.

3.2 The user should be able to calibrate:

3.2.1 minimum possible disparity.

3.2.2 maximum possible disparity.

3.2.3 uniqueness ratio.

3.2.4 focal length.

4 Render models:

4.1 The program should be able to render an image from two “still” photos.

4.1.1 The program should be able to take images as input data.

4.2 The program should be able to render an image from more than two images.

4.3 The program should be able to render an image from a video.

4.3.1 The program should be able to take a video as input data.

4.3.1.1 The program should be able to segment this video into the required number of frames.

4.4 From the input data the program should be able to create:

4.4.1 A stereo image that can be influenced by calibration.

4.4.2 The x,y,z,r,g,b values of each pixel.

4.4.3 A point cloud made from these values.

4.5 The renderings created must be clear and recognisable.

4.5.1 The stakeholders will determine if the renderings created are recognisable.

5 The program needs to be easy to use / intuitive to the end user.

5.1 Each stakeholder will be able to use the solution / program without requiring any detailed training.

Design

Initial concept of the program

My initial concept is to create a software program that has three main user interfaces:

- registration window: to allow users to create a profile.
- login window: to allow the user to access their account.
- main window: which will allow the user to render, save and open models, alter the calibration options which create the renderings and finally save and open created calibration pre-sets.

Using the program, you will be able to:

- input a selection of images – minimum of two images.
- import video as input data for renderings. If video is imported the program should be able to segment the video into each individual frame for processing.

Overall, my program should prevent any errors from crashing the program, and the software developed will need to be functional taking into consideration a simplistic and minimalistic approach. Using the computational methods of abstraction, I will create a hierarchy table of my initial concept of my idea, to visualise the basic functions I will need to design my interface around:

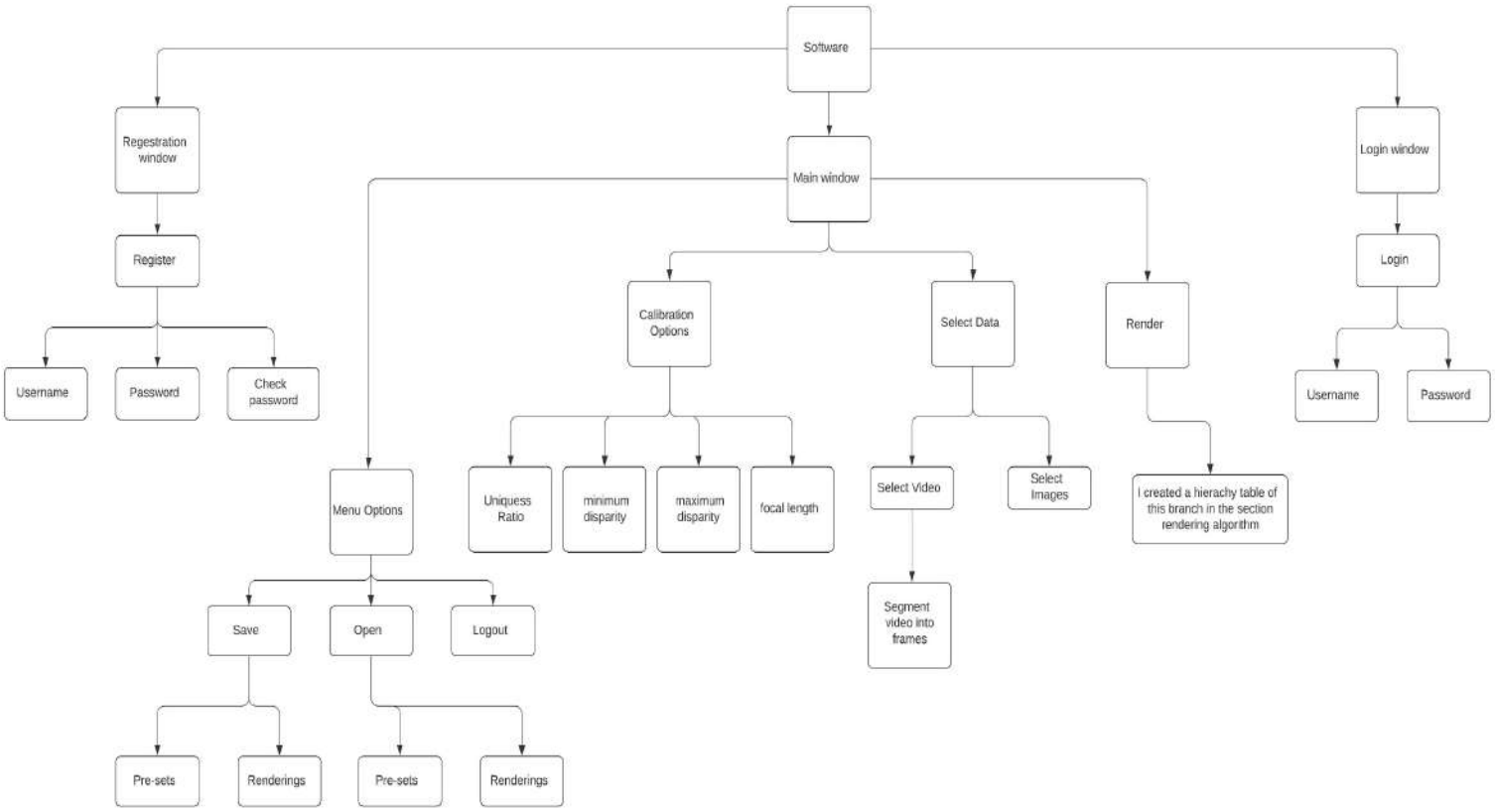


Figure 8 - Initial Concept Hierarchy Table

Figure 8 has enabled me to visualise in an abstract way what my program will need to include, as a minimum. The main window will be populated with the majority of the software functions:

- render: to render the images.
- select data: to allow the user to select a video or images to render.
- calibration options: to allow the user to alter the variables which influence the renderings.
- menu options: which will allow the user to logout, save and open both pre-sets and renderings (as per my initial concept). The hierarchy table has demonstrated to me, where I need to focus my effort and dedicate less time to the design and development of my registration and login windows, as both do not require as many functions and options.

My next step in the design process is to specify the context of my system, capture the requirements of it and better understand what I will need to design in order to meet my success criteria. I will need to visualise how my users will interact with the system. To accomplish this task, I will use methods of abstraction to create a use case diagram (using a UML) of each main section of program: user registration window, user login window, and the main window. Starting with a use case diagram and table for the user registration window.

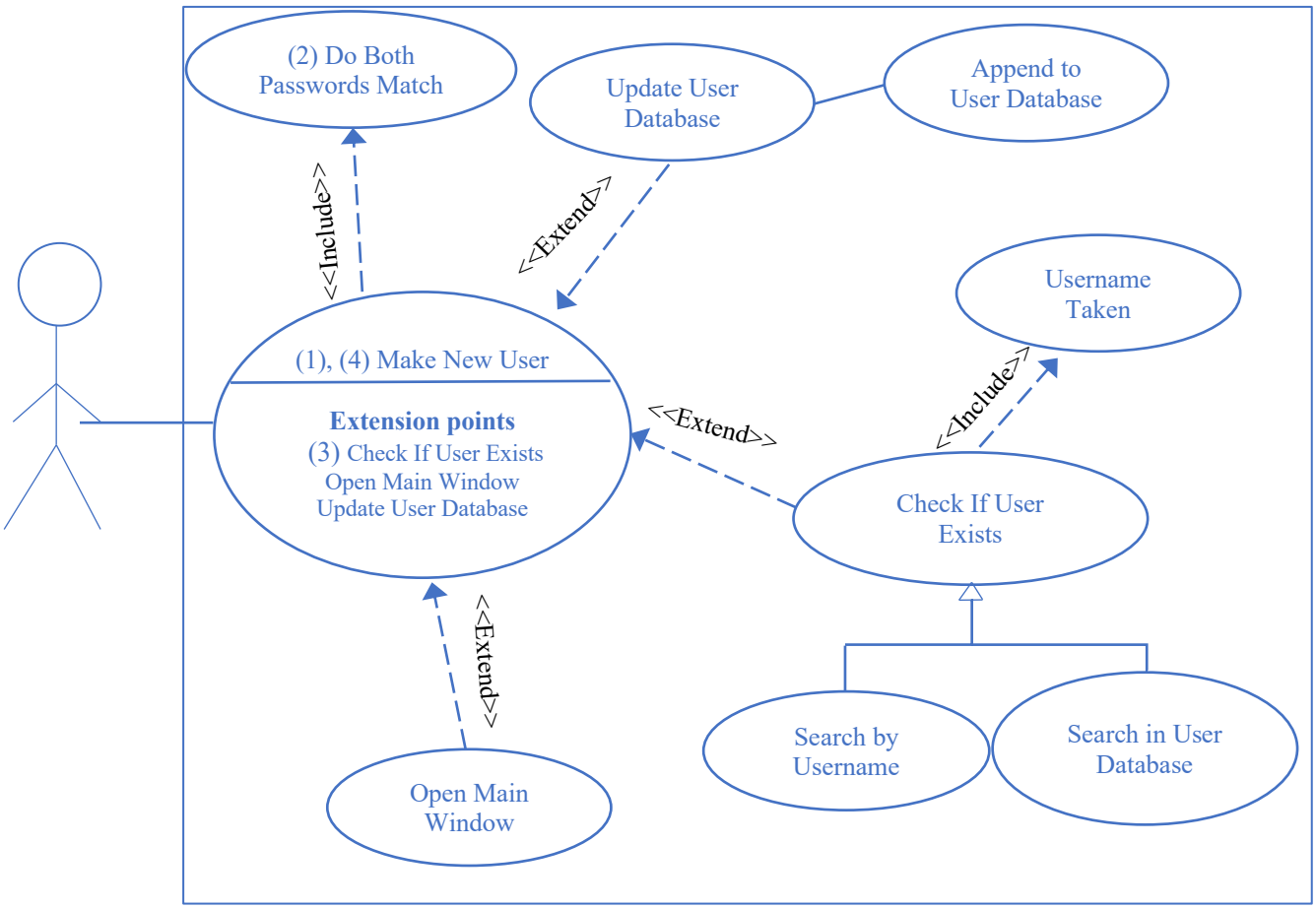


Figure 9 - User Registration Use Case Diagram

Here is a table to map the main scenarios represented in Figure 9.

Main Scenarios	**Serial No**	**Steps**
Actors/Users	1	Enter Username, password, and re-enter password.
	2	Do both passwords match?
	3	Is username already taken?
	4	Create user and allow access to system.
Extensions	2a	Passwords do not match, show error message.
	3a	Username taken, show error message.

Use Case Name	**Register window**
Use case Descriptions	A user registration system to create an account and access to the functionality of the system.
Actors	Stakeholders.
Pre-condition	System must be connected to the external database.
Post-condition	After a successful registration, the user can access the main window.

Figure 9 shows how the user interacts with the registration system, as you can see the main function inside of the registration system is Make New User. This function would include an error message if the passwords entered (password and re-entered password) do not match, the function will also be connected (extended) to other commands / functions, these include “open main window “(to open the main window once the account is created), “update user database “(to append user database with the new user’s data) and finally “check if user exists”. The function to check if user exists will also include an error message, if the username is already taken.

My next step in the design process, I will create a use case diagram of the login window, see Figure 10.

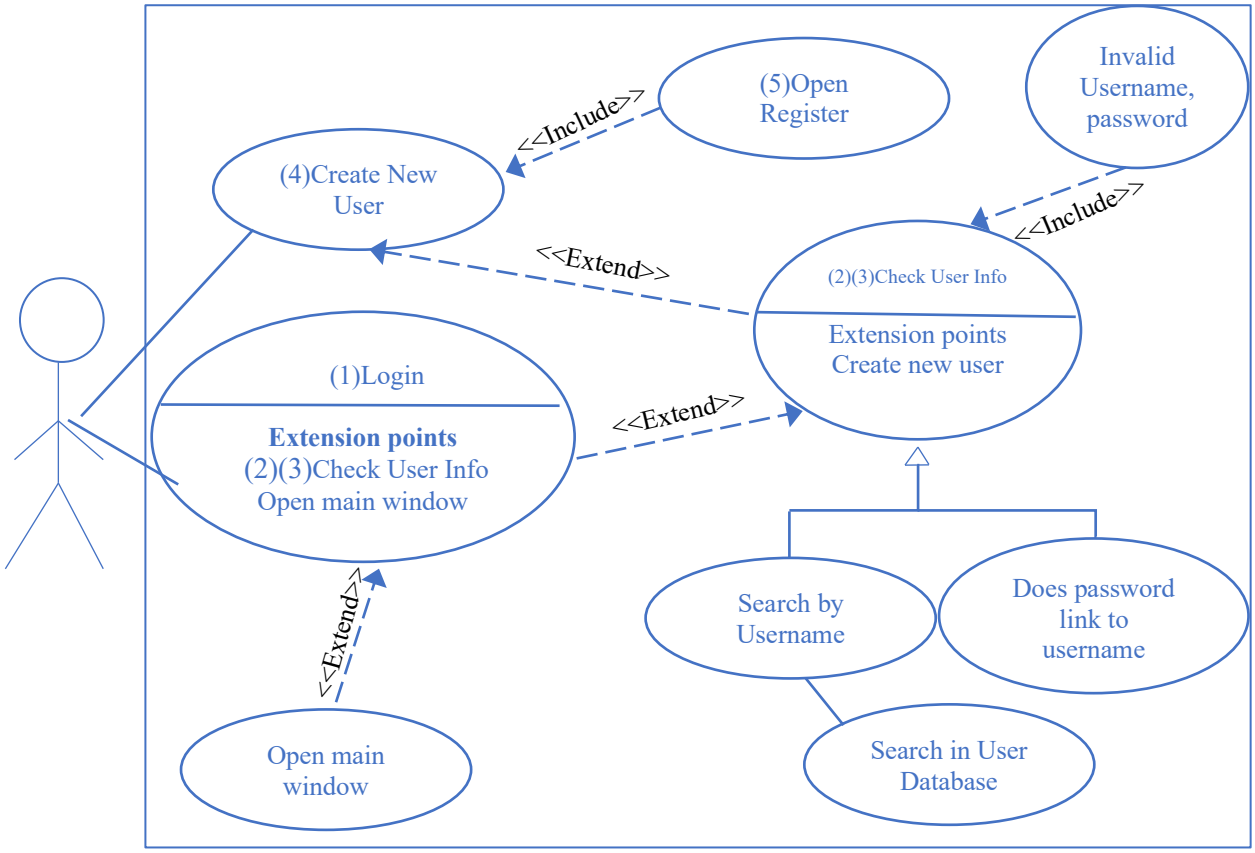


Figure 10 - Use Case Diagram for User Log-In

Here is a table to map the main scenarios represented the diagram.

Use Case Name	**Login window**
Use Case Descriptions	A user login process to access to the functionality of the system.
Actors	Stakeholders.
Pre-condition	System must be connected to the external database.
Post-condition	After a successful login, the user can access the main window.

Main Scenarios	**Serial No**	**Steps**
Actors/Users	1	Enter username, password.
	2	Validate username and password.
	3	Allow user access.
Actors/Users	4	Selects create new user.
	5	Open registration window.
Extensions	1a	Invalid usernames show error message.
	2b	Invalid passwords show error message.
	3c	Invalid password or username 3 times open register window.

Figure 10 shows how the user interacts with the login system. There are two functions inside of the login system that the user can interact with: create new user and log-in. The log-in function will be connected (extended) to other commands / functions and will include “open main window “(to open the main window once the account has been verified), and “check user info” (to search the user database for their account). The function check user info will include an error message if the username does not exist, or the password does not match the account. The function is also linked to the command create new user (to ask the user if they want to create an account, if they have had three or more invalid attempts).

My next step in the design process, I will create a use case diagram of the main window.

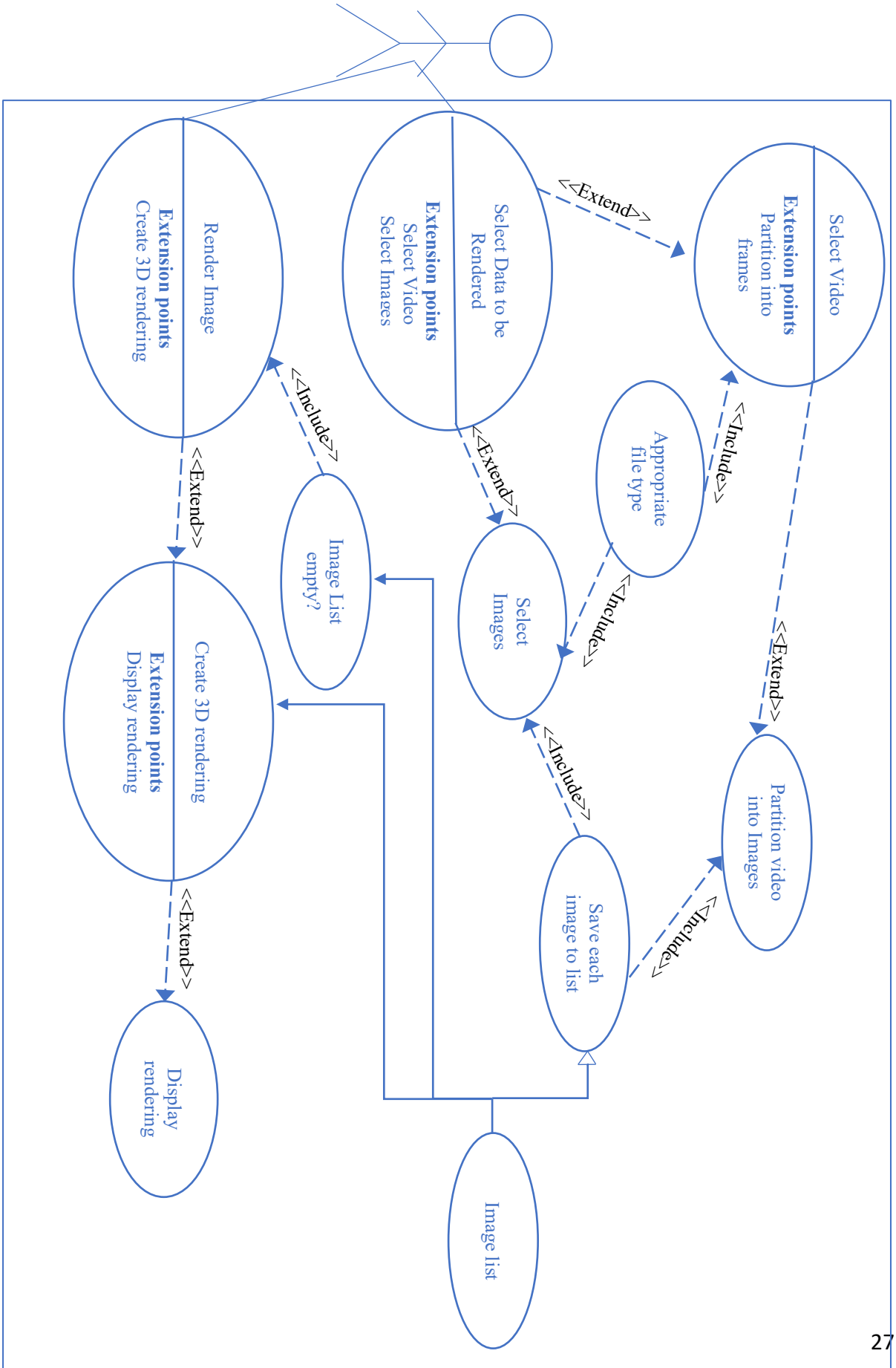


Figure 11 - Main Window Use Case Diagram

Here is a table to map the main scenarios represented in Figure 11.

Use Case Name		**Main window**
Use Case Descriptions		A system that allows users to select images or a video to then be used to create a 3D rendered model.
Actors		Stakeholders.
Pre-condition		User must be logged into an account.
Post-condition		A 3D model has been rendered.
Main Scenarios	**Serial No**	**Steps**
Actors/Users	1	Enter images.
Actors/Users	1.2	Enter video.
	2	Save each image selected to Image list.
	2.2	Partition each frame into an image and then save to image list.
Actors/Users	3	Run the sub routine Create 3D rendering.
	4	Display the rendering.
Extensions	1a	Invalid File type, show error message.
	1.2a	Invalid File type, show error message.
	2a	If create 3D rendering is run while the image list is empty, show error message.

Figure 11 outlines how the user will interact with the main functions of the main window. There are two functions that the user can interact with: select data to be rendered and render image. The select data to be rendered function will be connected (extended) to select images and select video. The function select video will be connected (extended) to partition video into images which will segment the video chosen into its individual frames and then saves these frames to image list (a list of the images). The function select images will allow the user to select images which will be also saved to image list. Both functions include a check to make sure that the files chosen to fit the correct file type (.jpg, .mp4, etc). The other function the user can interact with is a render image, this function is connected (extended) to the command create 3D rendering and includes a check to make sure the image list has images inside of it. The function create 3D rendering takes the Images from image list then creates the 3D model, after this it calls the command display rendering to show the user the 3D model created.

User interface design

Main Window

Now I have explained how I want users to interact with my program, I can now design a UI to support that requirement. I will start with the "Main window".

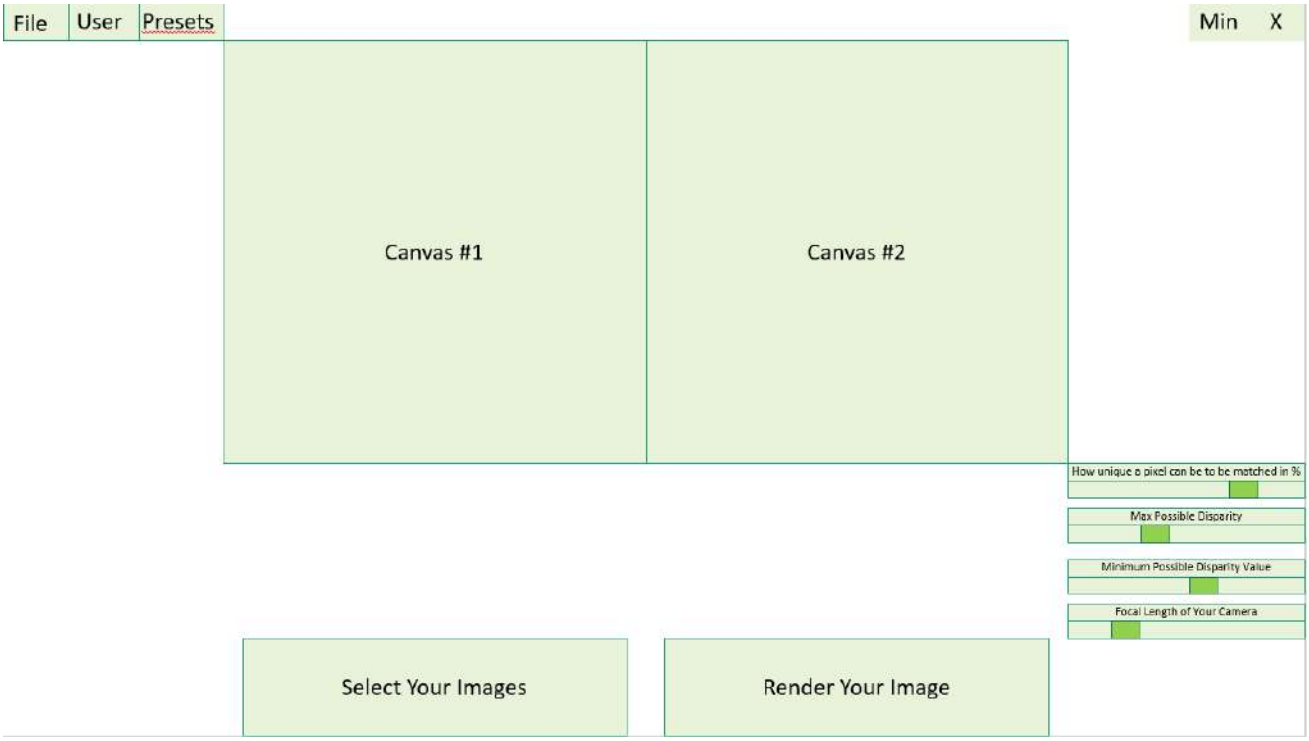


Figure 12 - Main Window UI Design

Figure 12 shows the concept of the design of the main window for my software program. This main window enables the user to see the images or videos they have selected, calibrate the program to create better renderings and most importantly the ability to render their chosen data. The design also has a top bar menu consisting of: file, user, and pre-sets. This menu will offer options to save and open a file (located under file), log out (located under user), and to save and open pre-sets (located under pre-sets). Under file, the user will have the option to change file type, this is to allow them to choose either video or images or vice-versa.

Links to the success criteria:

1. Simplistic design.

2. Main window that shows the user what they have chosen to render.

Canvas #1/2

The canvases will contain the first and last image inside of the list of images that the user has chosen to be rendered, however if the user has chosen a video file, the canvases will then contain the first and last frame of that video after it has been segmented into individual frames. This is to let the user make sure that the video / images they have selected are what they want / need.

Links to success criteria:

1. Be able to display either what images have been selected or what video has been selected.

Select your images button

The select your images button will open the users file directory to allow them to choose what files they want to be rendered. This is the same button that will be used if the user wants to select a video however the text to the button will be changed to “Select Your Video” when the user changes to a video file.

Links to success criteria:

1. Be able to take a video as input data.
2. Be able to take images as input data.

File option menu

The file option menu will contain three options, Open, Save and Change File Type. Open will allow users to open a previously saved rendering and Save will allow the user to save their current rendering. Change file type will allow the user to change their file from a still image-based rendering to a video-based rendering.

Links to success criteria:

1. The ability to save these renderings.
2. The ability to open previously saved renderings.

User option menu

The user option menu will contain one option, log-out. Log-out will allow the user to log out of their current account and will take them back to the log-in window.

Pre-sets option menu

The pre-sets option menu will contain two options, Save pre-set, and Open pre-set. Save pre-set will allow the user to save the alterations to the systems calibration that they have made, and open pre-set will allow the user to open previous saved pre-sets.

Links to success criteria:

1. The ability to save altering’s to the systems calibration to make a pre-set.
2. The ability to open a saved pre-set.

Calibration sliders

The calibration sliders are located bottom right of canvas #2, there are four different sliders all individually labelled, “How unique a pixel can be to be matched in %”, “Max possible disparity”, “minimum possible disparity” and “Focal length of your camera”. Each slider will be used in the creation of the renderings; however, the sliders will be pre-set to default values to allow the user to not need to calibrate the system.

Links to success criteria:

1. The option to change the minimum possible disparity value.
2. The option to change the maximum possible disparity value.
3. The option to change the focal length of the camera.
4. The option to change how unique a pixel can be to be matched with another.
5. Non-technical calibration.

Login Window

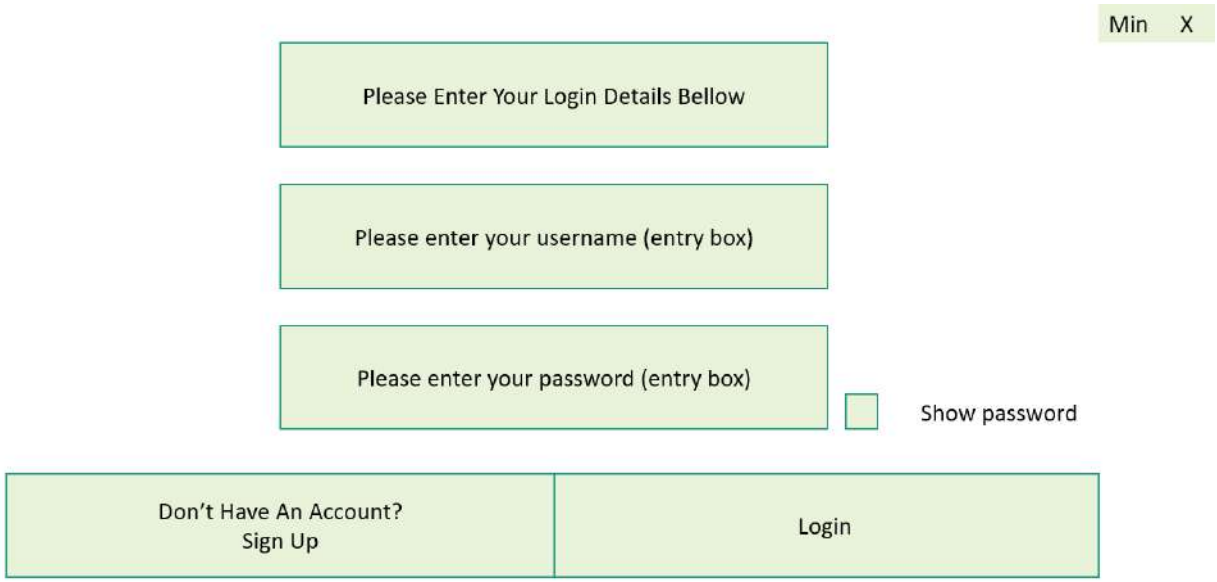


Figure 13 - Log-in Window UI Design

Figure 13 outlines the design of the login section of the program. This allows the user to enter the username and password of their previously created account; the design has both a button to login and a button to take them to the registration page. The design also has a check box to allow the user to show their password.

Link to success criteria:

1. A login window which allows pre-existing users to log back into their account.

Username entry box

The username entry box will allow the user to enter the username of their account to be later processed and checked to see if the account exists.

Password entry box

The password entry box will allow the user to enter the password of their account to be later processed and checked to see if the password matches the account they entered.

Show password check box

The show password check box will give the user to unhide their password so they can check if its spelt / entered correctly.

Don't have an account button

This button if selected will redirect the user to the registration page.

Login button

This button when selected will take the data entered into the entry boxes and will then cross reference with the user data base to see if the user exists and if they do open the main window.

Registration Window

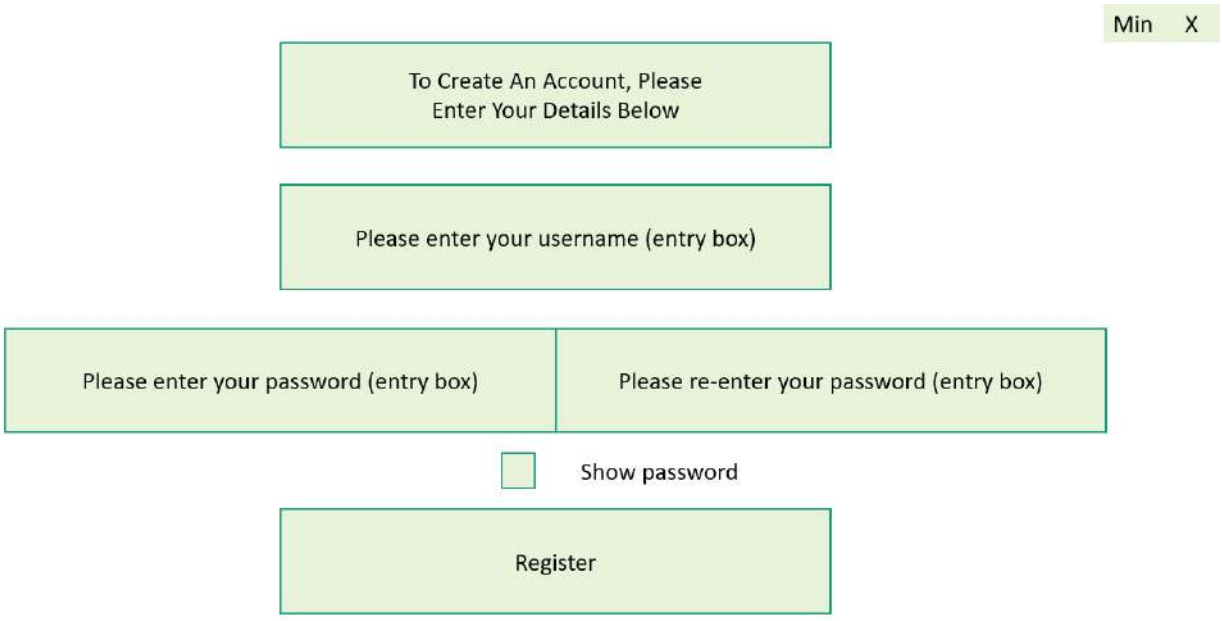


Figure 14 - Registration Window UI Design

Figure 14 outlines the design of the registration section of the program. This allows the user to enter a username, password and re-enter their password to create an account; the design has a button to register once they have filled out their details. The design also has a check box to allow the user to show their password.

Link to success criteria:

1. A registration window which saves the users data for logging in to their account.

Username entry box

The username entry box will allow the user to enter the username that they want their account to be called, this data will later be used to see if that username is already taken and will be used to append the user database.

Password entry box

The password entry box will allow the user to enter the password they want their account to have, this will be later processed and appended to user database.

Re-enter password entry box

This entry box be used to check if the password that the user has inputted is a duplicate of their first entry and validated against each other (password and re-enter password), if one if different from the other an error will occur, and the user will be asked to enter the password again.

Show password check box

The show password check box will give the user to unhide their password so they can check if its spelt / entered correctly.

Register button

The register button when selected will first check to see if username is already taken, if it is not, it will then append user database with both the username and password and will then open the main window.

Pop-Up Windows

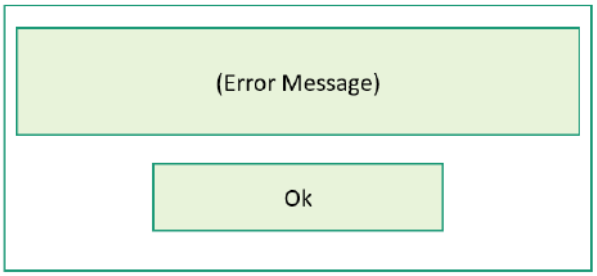


Figure 16 - Error Message Pop Up Window

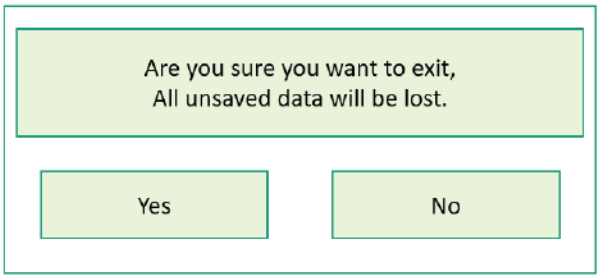


Figure 15 - Exit Pop Up Message Window

Figure 15 and Figure 16 outline the design of the pop-up windows, the pop-up window design (Figure 16) on the left will be used within all three of my windows, while the pop-up window design (Figure 15) on the right will only be used within my main window.

The error message label will be used to display any errors that occur throughout the use of the program, for example in the login window, if the password the user entered was incorrect it would say "incorrect password please try again".

All the buttons inside both the pop-up windows will terminate the window upon being selected, however on the right most design if the button "yes" were to be pressed all windows would be terminated.

User experience features

The user experience features that I have considered are mainly only applicable to the main window as this is where the user will spend most their time. These features will ensure that the system is simple to operate for all levels of users irrespective of experience. The layouts for each window that I have created all contain / have large text: so, it can be easily read by anyone using the program and will include large buttons: which allow the user to both easily see and click them, also allowing fewer miss clicks which could lead to starting over, this is especially useful considering that some renderings will take lengthy durations of time and if a miss click happened all that time would be wasted.

All the text present on the main window will need to be large and easy to read for individuals with sight or visual impairments / conditions. The main window will also have different buttons disabled at points in time, this is to further inhibit any accidental button presses which could lead to unwanted outcomes.

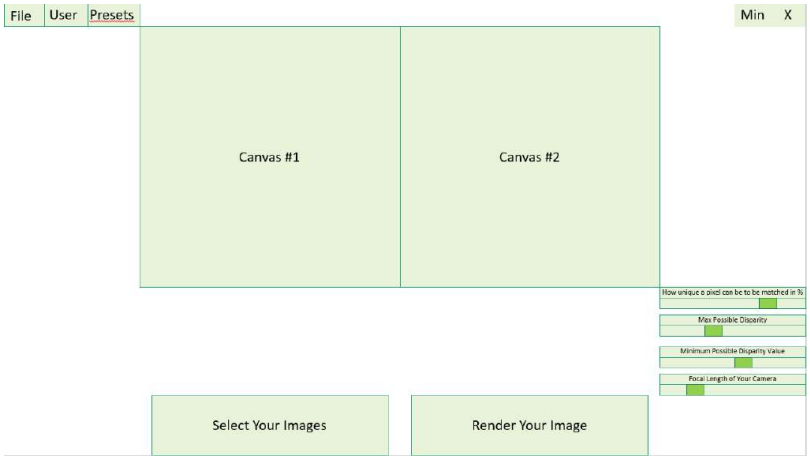


Figure 17 - UI Layout Design

Figure 17 outlines the layout design of the main window, it shows the large buttons, large menu options, large sliders, and clear text. However, what does need to be kept in mind is that the program will be taking up the full screen and so what may seem too small here will be of adequate size on the actual window. This layout design however may not be the same colour plan, but the text will still be easily visible.

The design of the main window will include features / functionality to prevent actions which can result in any unwanted outcomes. For this, I will be limiting the functions that the user can access and follow a process that will give the user prompts at given points. This will include the disabling of the render button until images or videos are selected, the displaying of the images / videos selected to alert the user, if they have chosen the correct data and a pop-up window to alert the user that all data not saved will be lost if they close the program.

The Stakeholder's Input

At this stage in the planning the development of my program, I already have a fairly consistent idea of how my final program is going to be presented and how it will function / operate. To cement my understanding and design thoughts, I have decided to discuss my progress with my stakeholders. I want to validate with my stakeholders if the interfaces will meet their requirements and to see if they my design supports the functionality, I discussed with them during the interviews I held.

I have emailed my stakeholders with the following / same email:

"Hello,

You will find attached the layout designs of the user interfaces. I have made sure that the design is simplistic and very easy to understand, all the buttons and calibration options are large and all easily available. The calibration will be simple and optional. The main two boxes in the main window will be to display what data you have chosen to be rendered. Other than a few other simple functions the rest will be handled by the computer. If you have anything you believe I need to add please share your thoughts.

Yours sincerely,
Alexander"

These were the responses I received:
Jonathan:
"The designs seem really good; however, I do have some concerns about the calibration options on the main window. My main concern is that have the options been too simplified. The worry is if losing options to change the programs calibration means creating a simpler calibration process, is it worth it? In summary the user interfaces for all the windows and pop-ups are more than satisfactory however I do have my concerns. However, if there isn't a loss in calibration options to create a simpler process then it's all perfect."

James:
"They all look great! I like how simple everything appears, and how it is not so cluttered that it becomes overwhelming with options and features. The calibration seems easy enough, and I do really like the idea that I won't have to do much work."

Martyn:
"Everything seems to be well thought through, in regard to sharing any of my thoughts I really can't think of much to else say, other than it looks good and if you wanted to add anything else to the user interfaces, I wouldn't mind it but need to make sure you can complete the development of the program."

Databases

The system will utilise three main databases, these databases will include UserData: which will include the data of the user log-in details, PresetData: which will include the data of the user's pre-sets that have saved, and FileData: which will include the rendered models saved by the users. Figure 18 shows the entity relationship diagram of how the databases will interact with each other.

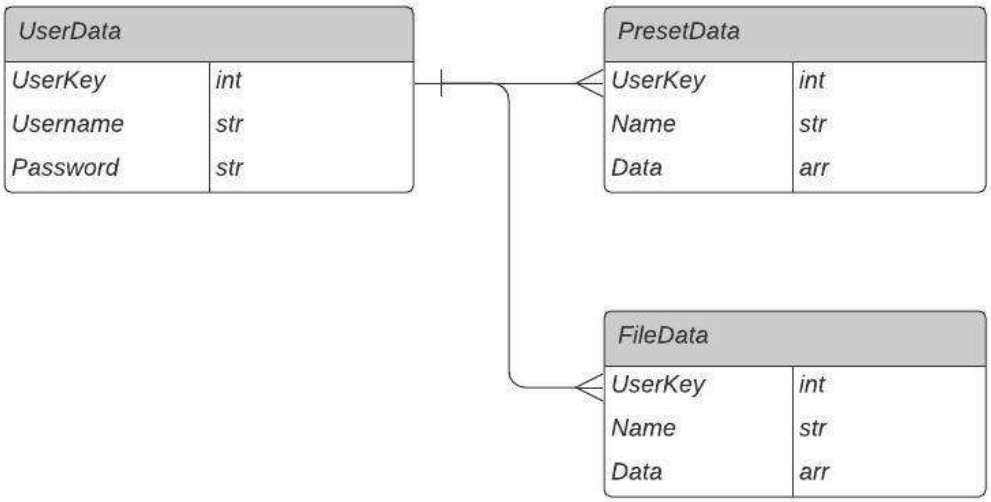


Figure 18 - Database Entity Relationships

Tables associated with the 3 main databases:

Table: UserData			
Field	**Key**	**Data Type**	**Validation**
Username	Primary	String	>0 Unique
UserKey		Integer	
Password		String	>0

The UserData database table will hold all the account data for the users. There are three separate fields: Username; UserKey and Password. Both Password and Username are of the data type string, and both must have a length greater than 0, however username must be unique and is the primary key (as it is what is searched for in the database). UserKey is an integer and will be generated by the username. As the UserKey is generated be it could not be the key for this database.

Table: PresetData			
Field	**Key**	**Data Type**	**Validation**
UserKey	Primary	Integer	
Name	Secondary	String	>0 Unique
Data		Array	

The PresetData table will hold all the saved pre-set data. There are three different fields, UserKey, Name, and Data. UserKey as mentioned previously, UserKey is an integer linked to the user's account. I have used it as the primary key in this database because when searching for the users saved pre-sets, it is very efficient to search for the UserKey and then look at the data linked to it. Name is defined as a string; this field must be both unique: so that data cannot be overlapped by another piece of data and has to have a length longer than 0: to make sure the string contains a value. Name is also the secondary key, this is because when a search is performed in the database, all the values linked to the user's key is searched and then followed by all the names which are allocated to the UserKey. Data which is defined as an array will contain the pre-sets data.

Table: FileData			
Field	**Key**	**Data Type**	**Validation**
UserKey	Primary	Integer	
Name	Secondary	String	>0 Unique
Data		Array	> 0

The FileData table will hold all the rendered models the user has saved. There are three different fields, UserKey, Name, and Data. All the data types and key types are the same as defined in the PresetData database. The difference between the two is what validation data has and what is stored in that field. The reason FileData must have a validation on Data, while PresetData does not, is because with PresetData no matter what the user does there will always be data to be saved. The sliders provide the current configuration (as set in the UI), while with FileData, a user might try to save a rendering before creating one first. Due to this scenario a check / validation is needed to make sure they have already created a rendering before they are allowed to save their data.

Rendering algorithm

The major challenge I have foreseen with the development of this program, is with the image processing and rendering methods using OpenCV. My overall target for this solution is shown in Figure 19:

Take a list of images/a video → Create a rendered 3D (pixel vectors with x, y, z) model

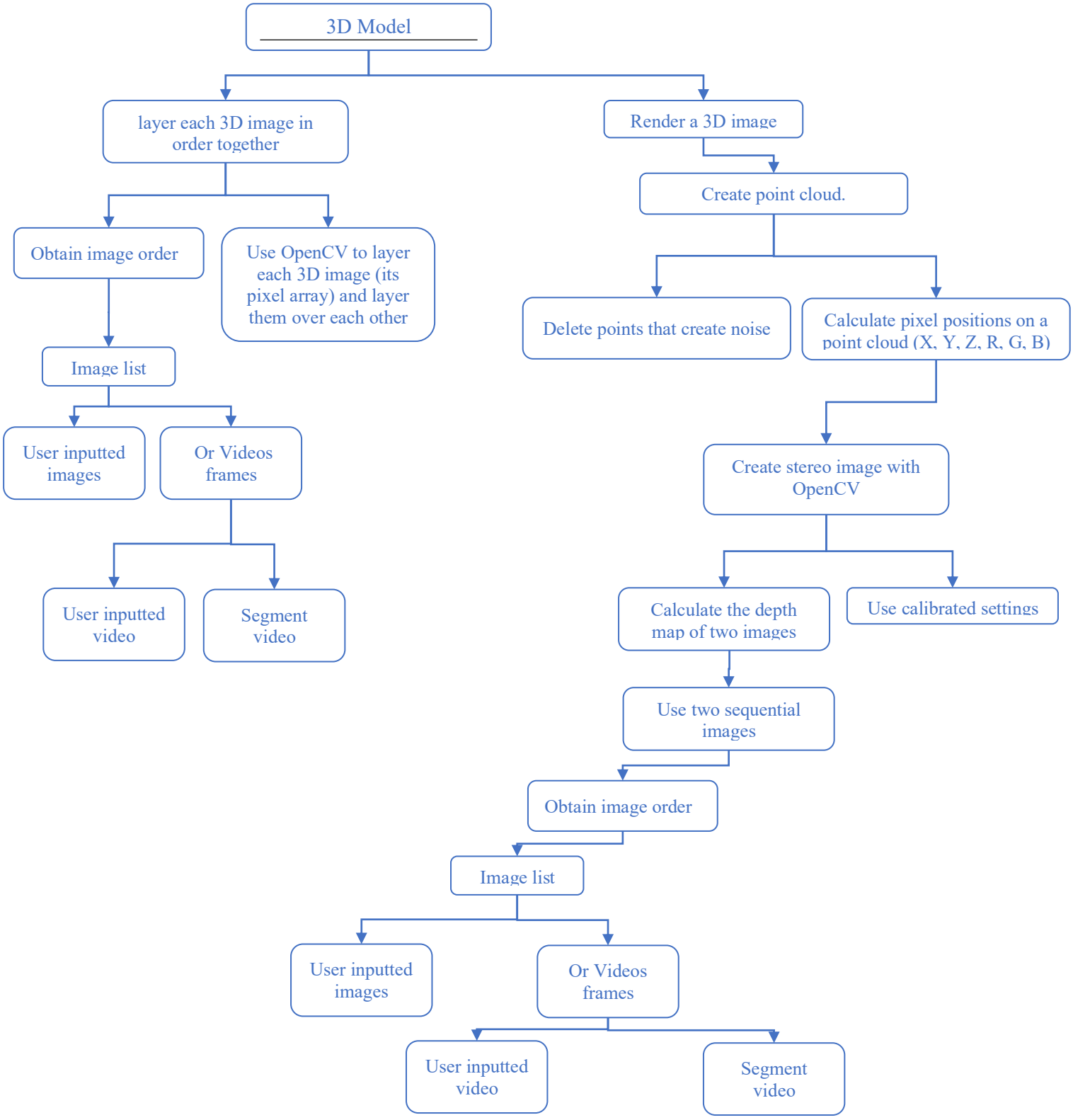


Figure 19 - Creation of a Rendered 3D Model

The process of creating a 3D model can be separated into 3 distinct sections:

- the creation of stereo images.
- creating a point cloud.
- layering each rendering / point cloud together.

The creation of stereo images will have to be repeated for each pair of images/frames, while the program will only need to create a point cloud for each stereo image and will only need to layer pairs of point clouds. Fractionating this into their individual subroutines shows the following.

Creation of Stereo Image:

- Take two images (left and right).
- Turn images into grayscale.
- Take in parameters relating to the images.
- Use these parameters to calculate the depth map.
- Layer the depth map back onto the left image.

Creating a Point Cloud:

- Use the stereo image.
- Calculate the position vector for each pixel (x, y, z).
- Deleting points which fall out of our pre-set limits.
- Projecting those points onto a point cloud with both x, y, z and R, G, B values.

Layering/Combining the Point Clouds:

- Take two-point clouds.
- Use OpenCV to add them together.
- Save as a new point cloud.
- Layer the next point cloud onto that.
- Return complete point cloud.

Once each of these processes are completed, my program will be able to display a 3D model. Let me explain each of these steps in more detail:

Creation of Stereo Image:

Take two images (left and right):

This step is necessary in the creation of stereo images, this is because a stereo image perceives depth like how our eyes do. We use our left and right eye (left and right image) to see the difference between the two frames of view, the disparity between the two allows us to perceive depth, this is the same method that stereo images use to calculate depth.

Turn images into grayscale:

While this step is not really necessary it makes the process a lot easier. By turning the images into greyscales, it allows the computer to quickly process noise that pixels with RGB values create. This allows the system to be more effective in creating stereo images. When we are creating hundreds of images at a time (if the user chose a video to render or a lot of images) this small change creates a big impact in processing time.

Take in parameters relating to the images:

This step is required for the creation of stereo images. The parameters that relate to this section are the minimum and maximum possible disparity between the two images, how unique a pixel can be, to be matched to another pixel, the maximum disparity variation within each connected component, the maximum allowed difference in disparity, and finally the parameters controlling the disparities smoothness.

Use these parameters to calculate the depth map:

In this step, all the previously named parameters will be used to calculate the depth map. This step is completely handled by OpenCV.

Layer the depth map back onto the left image:

In this step, the stereo image that was created by OpenCV is placed over the original left image as a mask, this is show what pixels in the image have what z values as they already have their x and y values inside of the left image.

Creating a Point Cloud:

Use the stereo image:

Use the stereo image created in the previous section to accomplish tasks within this section.

Calculate the position vector for each pixel (x, y, z):

Looking at the stereo images colour, the program can determine the z values of the image; this is because the stereo image is in grayscale and in the previous step it assigned pixels closer to the camera with a lighter colour (closest being white) and further away with a darker colour (furthest being black). With this data the program can assign each pixel an x, y, and z value.

Deleting points which fall out of our pre-set limits:

This step is essential to having a realistic rendering, this is because when rendering an image, it creates a lot of unnecessary data and this data can lead to both slowing the program down and creating an unrealistic rendering. So, this step deletes those points which create too much noise or fall out of a pre-set limit.

Projecting those points onto a point cloud with both x, y, z and R, G, B values:

Finally, this step (which again is handled completely by OpenCV) projects each pixel to a point cloud which is just a dataset which represents an object or space, this step also layers back on each pixels individual RGB values to give the rendering its colour.

Layering/Combining the Point Clouds:

Take two-point clouds:

In this step, if the user has chosen to render more than two images / frames, then the program uses two-point clouds which were created in the step above.

Use of OpenCV to add them together:

In this step, the program will utilise OpenCV to add together the point clouds with its built-in functions.

Save as a new point cloud:

This step will save the new point cloud as a new point cloud so it can be added to, if the user chose enough images to do so.

Layer the next point cloud onto that:

If there are more point clouds that have been created, each of the steps will be repeated until all point clouds have been combined. After this, the final point cloud should be returned and that is the 3D model.

Object Oriented Design:

To better understand the layout of my classes I will use UML. I will use a class diagram, see Figure 20, to show how each class will interact with both each other and their main routines:

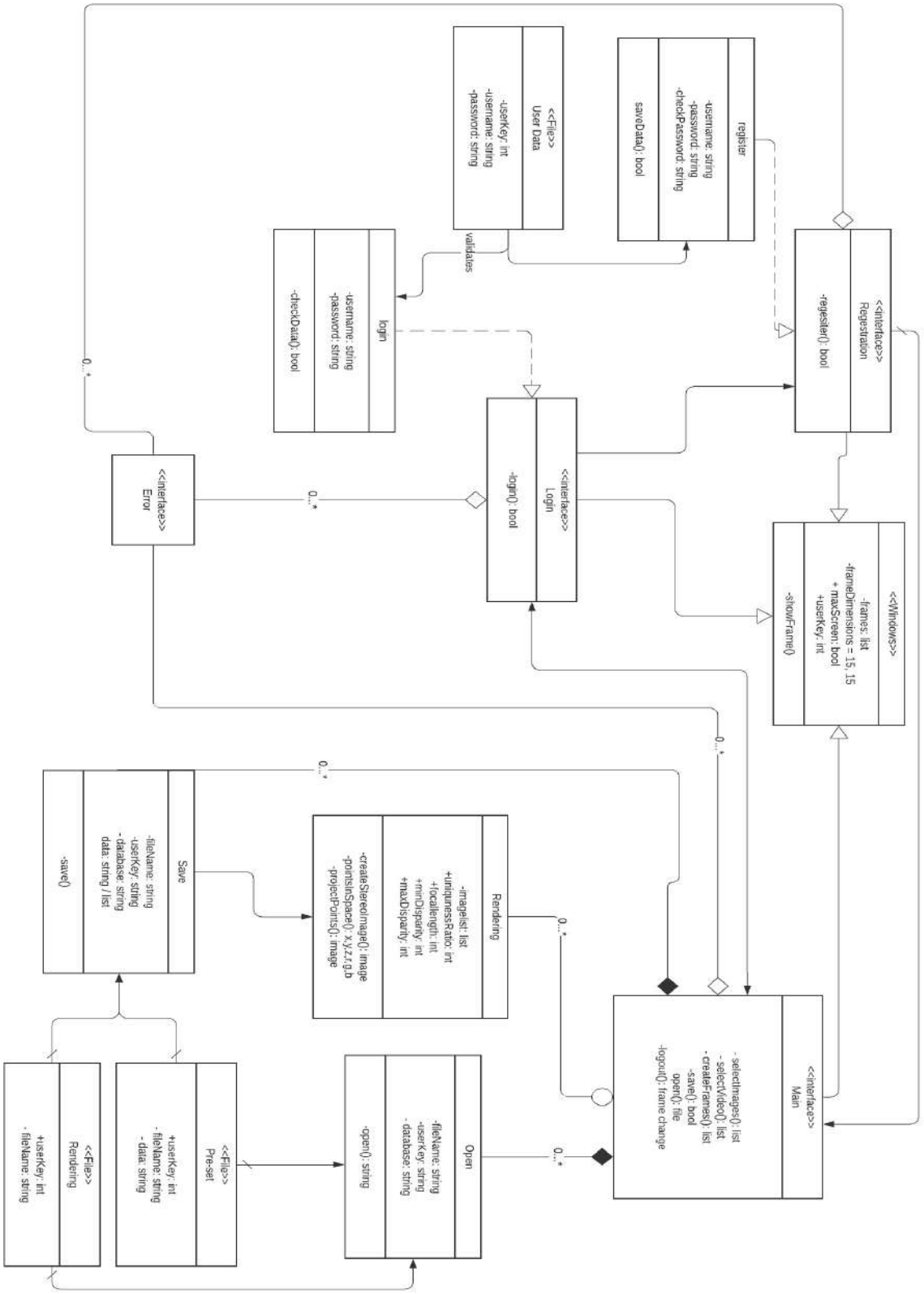


Figure 20 - Class Diagram for 3D Model Program

Figure 20 show the relationships I will have between my classes. I will have an abstract parent class (windows) which will be used by my main three interfaces: main, registration and login. All these interfaces will inherit their grid format and the function show_frame, which will be used to switch frames. Each interface has an aggregated relationship to the interface errors, which will be used to display error messages, whenever an erroneous action is performed or inputted.

The most important user interface “Main” contains multiple private functions, for example selectImages which outputs a list, and save which returns a Boolean answer as it either saves the file or it does not (true or false). Main also has a realisation relationship with a multiplicity of zero or more with the class rendering, as Main is the interface that implements the rendering class, and the user can render zero or more models. The classes open and save both have a composition relationship with a multiplicity of zero or more with Main, the reason for this is both open and save classes cannot exist without Main but Main can exist without them.

Save and open both contain the data from the files “Pre-set” (PresetData) and “Rendering” (FileData) these databases were both described in the section on databases. However, save also contains data from the class Rendering, this is because if the user chooses to save their rendering, the save function needs to access what rendering they have made / saved.

The next user interface registration has a singular private function register, this function is shown below registration. Registration has a realisation relationship with register, just like main and rendering, registration provides the UI for register. The last user interface is login, login like registration only has a singular private function login, this function is shown below login. The UI login has a realisation relationship with the function login, just like main and rendering, login provides the UI for login function.

Both the functions register, and login contain the data from the file user data (UserData), this database was described in the section on databases. While register only contains the values from user data, login is validated by user data, this means that while register can read and write to user data, login can only read the file to validate its inputs.

Main Subroutines

Now I understand how I will create the renderings and a design on how the whole program will function, I will now be able to plan the subroutines that I will use. The program will be programmed into one main file, both for the user interfaces and the rendering algorithm, this is to make it easier for me to follow the execution paths through my program and investigate / debug any errors when they arise.

Main window

The most important section of the program, as it allows the user to select the input data. This section will be needed for the rendering of the input data, as this is where the user will input their data.

Data Selection:

imageSelector

The program will need to allow the user to select as many images as they want, however does not allow them to choose any less than two images, it should also display the images the user selects.

Pseudocode algorithm

```
FUNCTION imageSelector():

        imagePathList = [] #Empty list to hold all the path directories

        imagePath = Tkinter.Open.FileDirectory()#Uses Tkinter to open their file directory

        WHILE length(imagePath) > 0 OR length(imagePathList) > 2#This to check if they have
chosen 2 images or chose a path
                imagePathList.append(imagePath)#this appends the imagePath to imagePathList

                imagePath = Tkinter.Open.FileDirectory()#Uses Tkinter to open their file directory

        LeftCanvas.image = image.open(limagePathList[0])#Sets the left canvas image to the first image
they chose
        RightCanvas.image = image.open(imagePathList[length(imagePathList) – 1])# Sets the right
canvas image to the last image they chose
```

videoSelector

The program will allow the user to select a singular video, it should also display the last and first frame of the video the user has selected.

Pseudocode algorithm

```
FUNCTION videoSelector():

        imagePathList = [] #Empty list to hold all the path directories

        video = Tkinter.Open.FileDirectory():#Uses Tkinter to open their file directory

        IF length(imagePath) > 0 #This to check if they have chosen chose a path

                createFrames(imagePathList, video)#Runs the function createFrames

                LeftCanvas.image = image.open(limagePathList[0])

                RightCanvas.image = image.open(imagePathList[length(imagePathList) - 1])
        ELSE:
                Print("Please select a video")
```

createFrames

This function will need to take into account the user's video and segment each frame. It will then need to save each frame to a folder on the user's computer and append each file path to imagePathList. This will then be used later used during the creation of stereo images.

Pseudocode algorithm

```
FUNCTION createFrames(imagePathList, video):

        Video = OpenCV.VideoCapture(video) #This opens the video

        success, frame = Video.read() #This reads the video

        count = 0

        WHILE success: #While the video has frames left to read

                OpenCV.imwrite("images/frame%d.jpg" % count) # this saves each frame to the file images with the name frame + whatever number count is

                imagePathList.append("images/frame%d.jpg" % count) #appends the directory to the list
                count += 1 #adds 1 to count

                success, frame = Video.read() #This reads the video
```

Figure 21 shows a flowchart to aide understanding of what this subroutine is doing:

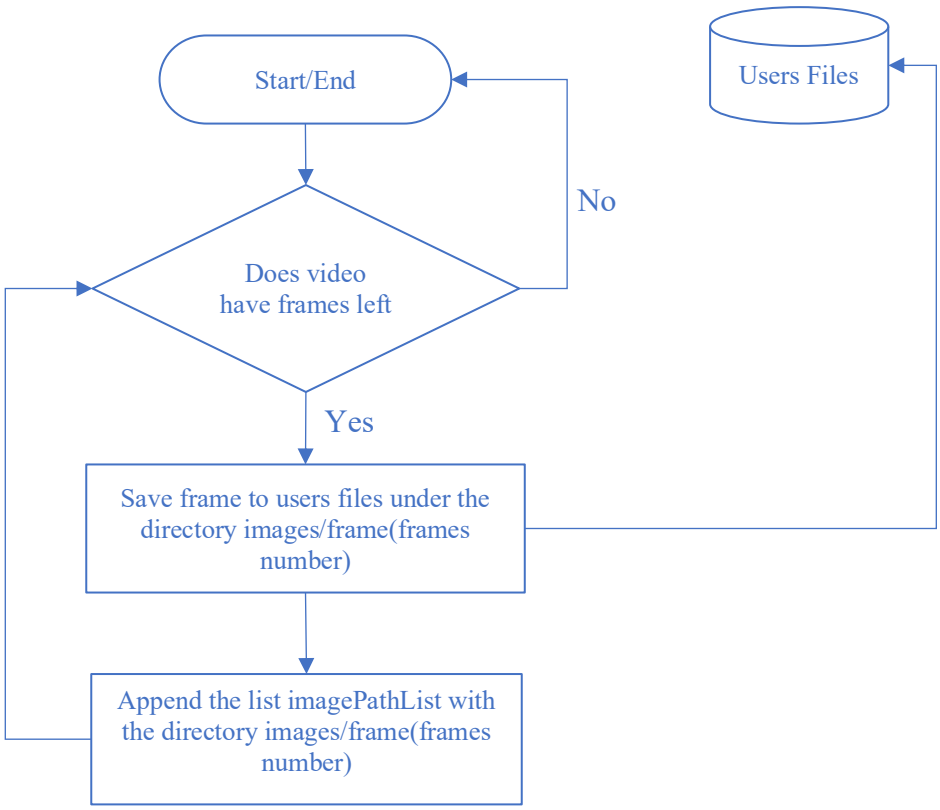


Figure 21 – createFrames Flowchart

Rendering Algorithm

This section of the program will be split into three sub-sections:

- Creation of Stereo Image.
- Creating a Point Cloud.
- Layering/Combining the Point Clouds.

As mentioned in the algorithms section, the system needs to use calibration data and images gathered by the main window, and then use that data to create the rendering.

Creation of Stereo Image

This function will need to take in two images and using OpenCV calculate the disparity between the two images. Inside this subroutine you will see the letter "n" repeated multiple times, this is to symbolise that an integer will go there however as of, yet I cannot assign them to anything (each n will be different).

Pseudocode algorithm

```
FUNCTION createStereoImage(left_image, right_image):
        window_size = n

                mindisp = n

                maxdisp = n - mindisp #Maximum disparity minus minimum disparity, this value has to be greater than 0

                stereo = OpenCV.Stereo_create(

                                minDisparity = n,       #Minimum possible disparity value

                                numDisparities = maxdisp, #Maximum disparity minus minimum disparity

                                uniquenessRatio = n,    #How unique a pixel can be (in %) to be matched with another

                                speckleWindowSize= n,   #Maximum size of smooth disparity regions to consider their noise
speckles and invalidate

                                speckleRange= n,        #Maximum disparity variation within each connected component

                                disp12MaxDiff= n,       #Maximum allowed difference (in integer pixel units) in the left-
right disparity check

                P1=n*window_size**n,    #The first parameter controlling the disparity smoothness

                P2=n*n*window_size**n,  #The second parameter controlling the disparity smoothness

                Stereo.compute(left_image, right_image) #This is a built in OpenCV function for computing stereo images
```

Creating a Point Cloud

To create a point cloud, I will need to employ multiple smaller functions and subroutines to create it, these will include finding the points in space: their x, y, and z co-ordinates, deleting points: which create noise, calculating the projected points and finally projecting the points to the point cloud.

Finding the Points in Space:

To find where to project each pixel onto the point cloud first, the program will have to analyse: the stereo image that we just created using the left most image that the user inputted. I will also have to create a 3D array using NumPy, this array will consist of; the height and

width of the left image, the focal length of the camera and finally the distance between where both images were shot (see Figure 5).

Pseudocode algorithm

```
FUNCTION pointsInSpace():
                Image = left_image #Sets image to be the left image

        Stereoimage = createStereoImage(left_image, right_image) #Calls the stereo image function

        Height, width, argument = image.shape # This is to find the images dimensions in the form of an array

        FocalLength = n #This is the focal length of the camera used

        camreaDisparity = n #This is the baseline

        D = np.array([[1, 0,0 -width/2], [0, 1, 0, -height/2],[0, 0, 0, focallength],[0, 0, -1/camreadisp, 0]]) #This creates an 4x4
        matrix with the specifications of the users

        points = cv2.reprojectImageTo3D(stereoimage, D).reshape(-1, 3) #using open cv it reprojects the disparity image to 3D
using the matrix D
                colour = image.reshape(-1, 3)  #this sets the value of colour to the left_image reshaped (changing the array but
not data)

         return deletePoints(stereoimage.reshape(-1), points, colour)      #This returns the value of what deletePoints returns
```

The reason for the argument in the line "Height, width, argument = image.shape", is because when gathering an array from an 2D image you receive 3 arrays back, one for the height, one for the width and finally one for the RGB values. Due to this we need a necessary argument to hold this array as without it the command ".shape" won't work however we don't need or want this array.

Deleting Points:

To remove noise and clean up, a point cloud to create a more precise version of it, we need to create a new array and lay it back over the both the image which contains the RGB values for image (colours) and lay it over the image which contains the x, y, and z values for the pixels (points).

Pseudocode algorithm

```
FUNCTION  deletePoints(stereoimage_arr, points, colour): # "_arr" meas the array of that image

    mask = ((stereoimage_arr > stereoimage_arr.min())
                &np.all(~np.isnan(points), axis=1)
                &np.all(~np.isinf(points), axis=1)) #This outputs a new array eg [[n,n,n], [n,n,n],[n,n,n]]

    return points[mask], colours[mask] #This returns the values of points and colour in respect to the new array "mask"
```

The commands inside of the line "mask = ((stereoimage_arr >Stereoimage_arr.min()) &np.all(~np.isnan(points), axis=1) &np.all(~np.isinf(points), axis=1))" do the following; _arr: finds the array of the image, _arr.min(): finds the minimum value of the array, np.all: checks wherever all array elements along a certain axis are equivalent to true, np.isnan: tests wherever an array is NaN (not a number) and returns a Boolean array, and finally np.isinf: this checks if the array is positive or negative infinity or is not either and returns a new Boolean array.

Calculate and Project the Points:

This will take in the points, colours, rotation vectors, translation vectors, the distortion coefficient, width, and height of the new image with the mask and using OpenCV's built in functions to project the points to a point cloud and by creating a new mask for both the linier cords and colours so they can be laid back onto the point cloud which as of yet only has the z cords on it, so we can finally have our finished 3D image.

Pseudocode algorithm

```
FUNCTION  projectThePoints(points, colours, rotationVector, translationVector, camreaMartrix, distortion, width, height):

  Projected = OpenCV.projectPoints(points, rotationVector, translationVector, camreaMartrix, distortion)     #projects the points with an openCV command

      linierCords = projected.reshape(-1, 2)

      mask = ((0 <= linierCords[: 0]) , (linierCords[: 0] < width) , (0 <= linierCords[: 1]) , (linierCords[:, 1] < height))
#Creates a new array to be layed over the colours and liner cords

      linierCords = linierCords[mask] # applys mask to the liner co ordanates

      colours = colours[mask] #Applys mask to colours

      image = np.zeros((height, width, 3), colours)

      image[linierCords[:, 1], linierCords[:, 0]] = arg

      cv2.imshow("Your Rendered Image", image)#Displays image
```

Testing these subroutines:

Due to the nature of these functions being drastically different to standard programming, i.e., the data that is being both handled and stored are images and 3D models. These and are not data types which lend themselves to trace tables and other forms of testing pseudocode, because of this I will have to test their functionality during the development phase. However, I would be obligated to do this anyway as the nature of rendering images means there will be a lot of small alterations to fine tune the system (as you can see in the subroutine createStereoImages with all the different "n" values I have to fine tune) which would benefit from continuous testing as it is visual in nature. The process for testing will be expanded upon in a future section.

Explanation and justification of this process

The problem that I have identified and undertaken to fix does seem extremely large and too complex for me to complete, however using the computational thinking methods of problem decomposition this task becomes increasingly simpler, if I adopt such a process. In this section, I will try and explain what my plan going forward will be.

I will create a single python file which will contain all the code for both the rendering algorithm and the user interfaces: main, registration and login window. I will first address the problem of creating the 3D model, as it is the single hardest part of my project. This task will include the creation of stereo images, the calculation of the point cloud, the removal of noise from the point cloud, projecting the image, and finally if there are more than two images selected then, layer each point cloud over each other to create a 3D model. A part of the task of rendering a model, which I would like to explain further, is the relationship between where each image is being taken to each other. Figure 22 will help me explain:

Figure 22 is an alteration (made by me) to the original, see Figure 7. Figure 22 shows an image (X) having 3 images taken of it (O'', O, O'). The relationship between these 3 points is O, this is because when creating the stereo images of these images O'' will use O to create its stereo image and O' will use O to create its stereo image. This would mean any time more than two images are entered the system would follow these rules: Stereo image "n" would use image "n" and image "n+1" to create it.

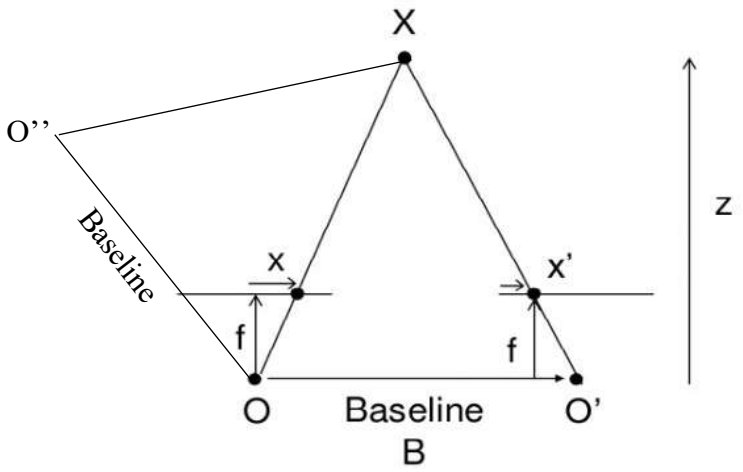


Figure 22 – Altered Epipolar Diagram

I will then create the user interfaces, each user interface will be created separately to test them and the functions which I create along with them, this is to ensure that my interfaces are laid out correctly and fit the designs I laid out earlier. This will also allow all the functions that I create to be adequately tested and iterated, as specified inside of the analysis section of this report.

By segmenting the program into these smaller sections and decomposing the problem, it makes it a lot easier for me to look at and investigate each individual section with more precision and scrutiny, giving the system the best chance of success of working. This all lends itself to creating a functioning modular program.

Inputs and Outputs

Main Window:

Input	Process	Output
Select images button.	Runs the select images algorithm.	A list of the image file directories.
Select video button.	Runs the select images algorithm.	A list of the frame file directories.
Render image button.	The program will take the calibration options and the images inside of the image path list and run the rendering algorithm.	A rendered 3D model/ image.
Menu Options (Save File/Pre-set, Open File/Pre-set, change file type, and logout).	Tkinter will open a new window for the bulk of the options. The change file type option will just change the command and text of the select image/video button.	The menu option chosen will create a pop-up window overlaying the main window. Or in the case of log out it will open the login page.
Exit button.	Runs exit algorithm.	Closes the program.
Minimize/Maximise button.	Runs Minimize/Maximise algorithm.	Minimize/Maximise the window.

Registration Window:

Input	Process	Output
Show password check box.	Runs show password algorithm.	Shows user their password.
Register Button.	Runs Register algorithm.	Opens main window.
Exit button.	Runs exit algorithm.	Closes the program.
Minimize / Maximise button.	Runs Minimize/Maximise algorithm.	Minimize / Maximise the window.

Login Window:

Input	Process	Output
Show password check box.	Runs show password algorithm.	Shows user their password.
Don’t have an account button.	Opens registration page.	Opens registration page.
Login Button.	Runs login algorithm.	Opens main window.
Exit button.	Runs exit algorithm.	Closes the program.
Minimize / Maximise button.	Runs Minimize / Maximise algorithm.	Minimize / Maximise the window.

Input validation

To make sure there are not any invalid responses, I will need to validate every single input condition that the program takes in, to make sure that either the information that the user has inputted is correct or the program does not crash with input errors. Here are the different ways / types of data the user can input data:

Image/Videos:

The only way that an image or video entered can be incorrect is if the file the user has chosen is not a valid video or image file. To make sure this does not break the program, I will need to use pythons try and except methods, this will allow me to create my own error pop-up window and place it within the except state. Whenever an erroneous data type is entered my program can create an error window instead of malfunctioning.

Entry Boxes:

The entry boxes are all handled by tkinter, this means that these entry boxes do not need validation as the system is already very strong against invalid responses. The only precaution I will need to make, is handling the data the user inputs correctly. By reading the data inputted as just variable and not allowing it to be read as a command, prevents the program and or its databases from being at risk. For example, if I were to use an SQL database this would prevent injections from taking place.

Sliders:

The sliders are also handled by tkinter, again making invalid inputs will be very rare. However, in the case of sliders erroneous data input is close to impossible, as the slider has a range that it works within. This is because with sliders they have boundaries that I can set, and because of this the user will only ever be able to enter a valid value. This provides the necessary validation checks.

Menu Options:

The option menus are also controlled by tkinter. When the menu options are not linked to a command, when selected nothing will happen. If the options are linked to command and not all the correct data is passed through, then there is a risk that the program will generate an error message.

Buttons/Check Buttons:

The last sub-section which needs input validation is button / check buttons. Like option menus, a button if not linked to a command, when selected will not do anything and will not create an error. However, if they are connected to a command and selected, if not all the correct data is passed through an error will occur. To help prevent this, I will create a try and except state with its own error window, so the program does not crash.

Testing method

During the development of my program, I will be systematically testing it along the way, every time I add a new function, section, or new part to a UI, I will execute the program to both make sure that it works as intended, and that everything is aligned correctly. I will start with the functional testing methods and then move onto the non-functional testing:

Unit testing:

Unit testing is the first stage of testing each of the modules created. Unit testing is the process of testing each section within the code with valid and non-valid input values. This stage of testing is essential, as if not done to an appropriate level, an error which was coded inside of a section might not be found until you test the system, which could drastically affect the program and result in needing to re-code a lot of the program.

Integration Testing:

The next step in testing will be the integration test. In this phase of testing, I will be combining / integrating two completed and tested units and testing their functionality as a combined set of modules. This will ensure that no functionality was lost by adding the two functions together.

System Testing:

System testing is the testing of the entire integrated system, for me this will include all the UI's and the rendering application. The system testing will determine if the system has met the criteria points, I specified during my analysis.

Acceptance Testing:

This stage of testing is to analyse if the program fits all the criteria points, I laid out inside of my analysis. This will also include all the criteria points which rely on input data and also the functionality required of the program to meet my design.

Usability Testing:

Usability testing will be used to test whether my considerations that I laid out in my user experience section, produce the desired outcome of an easy-to-use and intuitive system. This testing will be done preliminary by me, then handed to my stakeholders and receiving their feedback on the system and its usability.

The unit and integration testing will be done within the coding phases to iteratively improve my system and to allow checks for errors inside my code. However, the system, acceptance, and usability testing will all be carried out in the final testing phase once the entire code is completed.

Visual inputs

During the development of the rendering section, I will need to have selection of visual inputs (images) to adequately test my system. For the development of the rendering algorithm, I will test it with only two images and move onto more later on in the sections (most likely when I combine the main window with the rendering algorithm). This however is not as simple as just selecting any two images, there are many different variables that need to be considered.

1. Firstly, and most importantly the images need to be a left and right perspective of the same object. Without this being the case, the creation of a stereo image would be impossible.
2. Secondly, the images must be taken but the same camera, this is because the focal length of the camera you use is needed to create the matrix which is used to create the point clouds.
3. Thirdly, while not being essential, the testing would greatly benefit from the images being taken from the same height, as it would allow for a better creation of the stereo image.
4. Lastly, the testing would benefit from the object being large and simplistic. This is because the smaller and more intricate an object is the harder it is to create a precises depth map of its surface.

After careful consideration, I found the object which I believe could fit all the criteria necessary and when taking the images, I made sure to get a left and right perspective and keep the camera as level as possible. Here are the two images I gathered:

Left Perspective

Right Perspective

I will also use visual inputs to showcase how my system can display images, as these images are not being processed, just displayed, I will just use any images that I can find. I will also use videos to showcase my systems functionality, however I will show the videos I will use in the sections I process them in.

Testing check list

Once I have finished the development of my program, I will need to make sure that its main functions work. To do this I have set out a testing checklist which contains a list of all the functions inside the program I need to validate. I will use this checklist during my final testing phase.

Action to test	**Working?**
The “minimize/maximise” button minimizes or maximises the screen.	
The “exit” button closes all windows.	
The “show password” check button shows the user their password.	
The “login” button checks if the user entered correct details then either logs them in and opens main window or asks them to enter new details.	
The “Don’t have an account” button opens registration window.	
The “Register” button opens UserData and saves the users account information to the database if the username is not already taken.	
The menu option “Change file type” changes the data the user can upload from images to video or vice versa.	
The menu option “save” saves the rendering that the user just created to the database RenderedData.	
The menu option “open” opens a previously saved rendering that the user created.	
The menu option “logout” opens the login window	
The menu option “Open Pre-set” opens a previously saved set of calibration options.	
The menu option “Save Pre-set” saves a set of calibration options to the database PresetData.	
Any errors create a pop-up window instead of resulting in a python error.	
The “Select your images” button opens the users file directory to select images	
The “Select your video” button opens the user file directory to select a video.	
If a video is selected it should be separated into frames.	
You should be able to render an image from as few as 2 images.	
Selecting images displays both the first and last image selected on the canvases.	
Selecting a video displays both the first and last frame of the video on the canvases.	
The “Render image” button will run the rendering algorithm and return the rendered image.	
The program can produce a stereo image/images from the data selected.	
The program can produce a point cloud from the stereo images.	
The program can combine the point clouds	
The program can display the created 3D model/Image.	

Interface inputs and test tables

Buttons/Check Buttons:

While the buttons behaviour and functionality are not handled by me, e.g., by tkinter, I am not concerned with the buttons creating errors on their own (would have been tested by tkinter), however the commands and functions assigned to these buttons are all designed and created by me and will need to be tested. To make sure the functionality of these buttons works correctly, I will need to test each single button by selecting them to check if each has the desired effect.

Links to the success criteria:

- Be able to take images as input data. (The method to gather the images is through a button press).
- Be able to take a video as input data. (The method to gather a video is through a button press).

Sliders:

Once again, these sliders are all handled by tkinter and should not create any errors themselves (as discussed in the validation section). I will need to test the effect the sliders have on the rendering. Here is a test table I will follow:

Slider 1	**Slider 2**	**Slider 3**	**Slider 4**	**Result**
Minimum	Minimum	Minimum	Minimum	
Maximum	Maximum	Maximum	Maximum	
Random Value	Random Value	Random Value	Random Value	

Links to success criteria:

- The option to change the minimum possible disparity value.
- The option to change the maximum possible disparity value.
- The option to change the focal length of the camera.
- The option to change how unique a pixel can be to be matched with another.

Entry Boxes:

The entry boxes are also completely handled by tkinter, which means that they would have already had been tested / validated by tkinter and have a strong protection against typical input errors. Due to the nature of me saving and reading what the user enters to databases and requiring the users details to be different from others, to allow all the features of my system to work, I will need to test to see if my validation attempts were successful.

Here is a test table I will follow:

Entry Box Input	**Result**
Null (no data).	
Data that has already been taken.	
Random unused data.	

Menu Options:

The menu options are also handled by tkinter and like the buttons they should not create errors on their own, however the commands and functions assigned to these options are all designed and created by me. This means to make sure the functionality of these options works correctly I will need to test each option by selecting each of them to check if each has the desired effect.

Development and testing

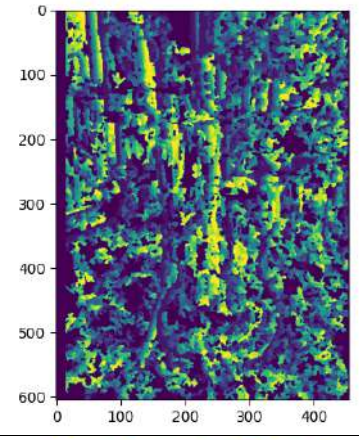
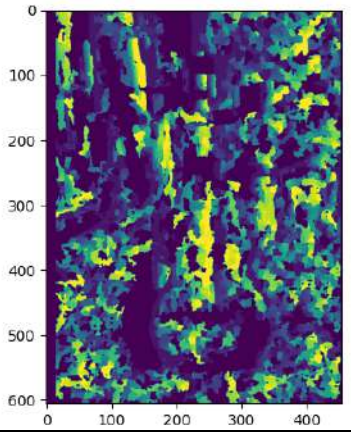
Coding Contents:

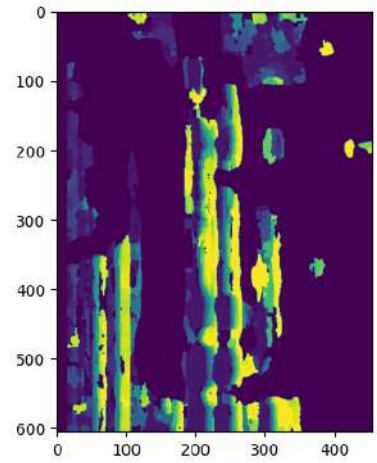
Phase 1, Rendering algorithm.

Now that I have discussed the theory behind how OpenCV can produce renderings of two images inside of my design phase, I will create my own method on how to accomplish this task, and hopefully improve upon the drawbacks I saw in the theory section. I will start with the creation of the stereo images:

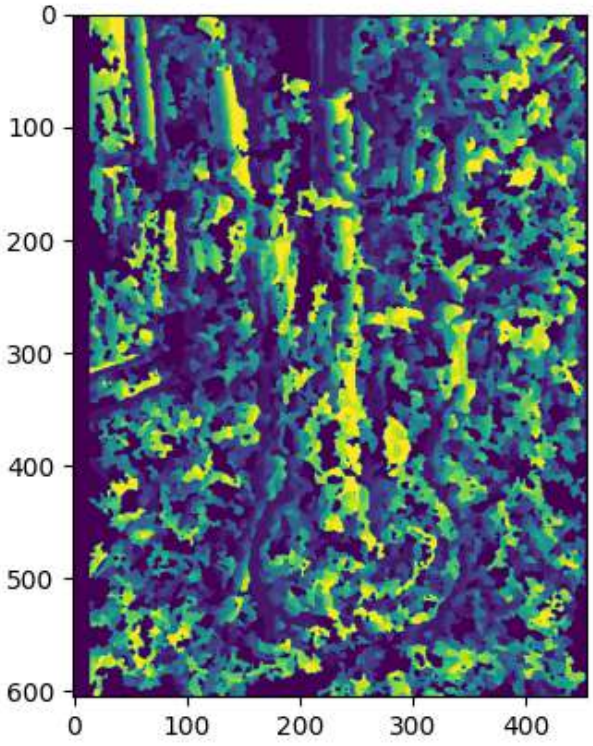
```
1  import cv2
2  import numpy as np
3  from mpl_toolkits import mplot3d
4  from matplotlib import pyplot as plt
5  #Opening the images
6  left_image = cv2.imread(r"C:\Users\alexf\OneDrive\Documents\tests\Screenshots\Testing\IMG-0884.jpg")
7  right_image = cv2.imread(r"C:\Users\alexf\OneDrive\Documents\tests\Screenshots\Testing\IMG-0885.jpg")
8
9  height, width, _ = right_image.shape#calculating the dimensions of the images
10
11 #From openCvs StereoSGBM page is where guidences for these values were given.
12 #However alot of trail and error was involved
13 stereo = cv2.StereoSGBM_create(minDisparity = -1,#Minimum possible disparity value
14                                numDisparities = 16,#Maximum disparity minus minimum disparity
15                                blockSize = 11, #This is to control how big the block of pixels is that gets matched
16                                uniquenessRatio = 1,#How unique a pixel can be (in %) to be matched with another
17                                speckleWindowSize = 200,#Maximum size of smooth disparity regions to consider their noise speckles and invalidate
18                                speckleRange = 2,#Maximum disparity variation within each connected component
19                                disp12MaxDiff = 10,#Maximum allowed difference (in integer pixel units) in the left-right disparity check
20                                P1 = 8*3*1**2,#The first paremeter controlling the disparity smoothness
21                                P2 =32*3*1**2)#The second parameter controlling the disparity smoothness
22
23
24 plt.imshow(stereo.compute(left_image, right_image))#This plots the created stereo image
25 plt.show()#This show the created stereo image
```

This method for creating stereo images is incredibly easy and straight forward, this is because it is a function built into OpenCV and will use it as designed. However, as you can see on line 14, I added an optional variable, not usually assigned in the method. This variable block size is instrumental in the creation of stereo images, this is because it controls how big the blocks of pixels that are matched are. I will now test the change in this variable to show how important it is, while simultaneously testing the creation of stereo images:

Block Size Value	**Output – Input data labelled in design**
`blockSize = 0,`	
`blockSize = 11`	

`blockSize = 23`	

As you can see this value has a significant role in the creation of stereo images and depth maps, my testing showed me that my initial value of 11 assigned to the variable was too large, while the base of the guitar is visible in the darkest area of the map, with the value assigned to 0 however it became slightly clearer but too much noise is around it, and after further trial and error I came to the value of 7 which produces the best stereo image in my opinion:



I can (while not so clearly) make out the shape of the guitars body, these stereo images are also a lot clearer than the ones created inside of theory's method, I attribute this to both the better visualisation method using matplotlib instead of OpenCV and the added variable which has allowed me to better fine tune my stereo image creation.

The next step is taking these points which already now have an x, y, and z coordinate and by applying a transformation matrix to them and projecting these transformed pixel vectors to 3D we have created a point cloud. However, first before I can code this myself, I will need to discuss how to derive the transformation matrix and how I will apply it to my solution:

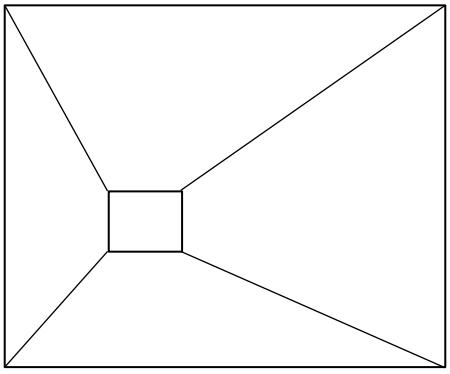
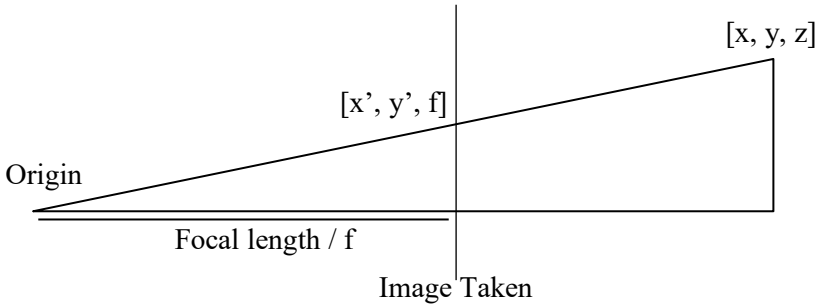


Figure 23 - Frustum Representation

The frustum that I have drawn, see Figure 23, is a simple representation of how an image (the smallest square), expands out to the real world (the largest square and everything between the two). This is import as if you turn this diagram to show only one of its sides you can work out the perspective matrix.

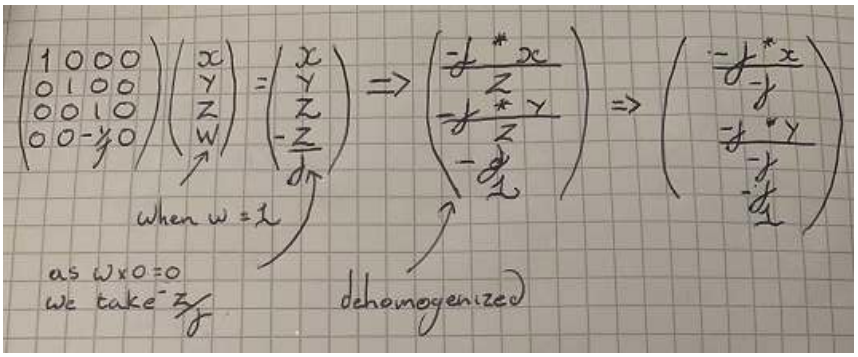


This is the diagram for deriving the matrix. The xyz are the real-world cords that have projected to the cords x', y', f. The relationship between x', y', f and x, y, z are as follows: $\frac{f}{x'} = \frac{z}{x}$ the x's are interchangeable with the y's.

This all then can be placed into a matrix which looks like this: $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{-1}{f} & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$

```
transformation = np.float32([[1,0,0,0],
                             [0,1,0,0],
                             [0,0,1,0],
                             [0,0,-1/f,0]])
```

however, we now need to de-homogenise this matrix to obtain for the proof this is the matrix:



This will result is a matrix with the value of the projected point, because of this I know I have achieved the correct transformation.

As you can see inside my code now, I have placed the variable transformation into my code, this variable as you can see contains the transformation vector, we derived from our perspective projection diagram.

Next step in rendering an image, is to derive both the translation and rotation vectors of the images. I will describe what is meant about translation and rotation vectors. A transition vector is the change of axis from the centre of one image to the other, then the rotation vector is the change in rotation from one image to the other. My main dilemma is I am making this software for an end user, this means that each image rotation and translation will vary, and so I cannot set a fixed value to it, unless the generalisation of them both is adequate.

There are two main methods for camera calibration in OpenCV: the functions calibrateCamrea and solvePnP. Both functions have their own requirements, inputs, weaknesses, and strengths. So, I will need to look all these factors before I even try and use them:

	SolvePnP	calibrateCamrea
Strengths	• It can take less time than other alternatives.	• Allows for very precise and accurate acquisition of the vectors.
Weaknesses	• Can be inaccurate. • It takes a long time to gather the t and v vectors.	• It takes a very long time to gather the t and v vectors.
Requirements	• Requires the program to know two common points to match together.	• Requires the user to take a picture of an already pre known image/object.
Inputs	ObjectPoints, imagePoints, camreaMatrix, distortion.	ObjectPoints, imagePoints, image Size, camreaMatrix, distortion.

This table has shown me I can use either of the main methods of vector acquisition, this is because each need real world cords that it can match and then adjust the vectors to fit each other. This means, I will have to find a work around and must derive the rotation and translation vector myself. To do this I will utilise the matrix I derived earlier on page 58.

The final form of the matrix I derived on page 58 is [[1,0,0],[0,1,0],[0,0,-f]], and this is what I will be using in conjunction with the camera's matrix, to find the rotation and transition vectors. First however I will describe the cameras matrix and how I defined it:

$$\mathbf{K} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

calibration matrix

5

This diagram shows the relationship between the focal length (f) of the camera and the dimensions of image that was taken. The matrix K or the cameras calibration matrix that is shown here is a universal camera matrix, while it may not be incredibly precise it still will be fairly accurate for my purposes. Now to implement this to my solution.

```
37 camreaMatrix = np.array([[f,0,width],
38                          [0,f,height],
39                          [0,0,1]])
```

[5] Kris Kitani, 16-385 Computer Vision, p.17 - http://www.cs.cmu.edu/~16385/s17/Slides/11.1_Camera_matrix.pdf

I next need to add the homogenized matrix that I spoke about earlier:

```
33 homogenized = np.array([[1,0,0],
34                         [0,1,0],
35                         [0,0,-f]])
```

Finally, I can utilise both these matrix's and by using OpenCV I can determine the rotation matrixes and transition vectors.

```
53 ret, RotationMatrix1, RotationMatrix2, transitionVectors = cv2.decomposeHomographyMat(homogenized, camreaMatrix)
```

Here I used the command called decomposeHomographMat, this command takes both the homogenized matrix and camera matrix and returns the values for possible rotation values and possible translation values. Here are the values it produces:

```
Rotation 1 :
[array([[-0.28047825,  0.95883405, -0.04437579],
       [ 0.95883405,  0.27774141, -0.05913514],
       [-0.04437579, -0.05913514, -0.99726317]]), array([[-0.28047825,  0.95883405, -0.04437579],
       [ 0.95883405,  0.27774141, -0.05913514],
       [-0.04437579, -0.05913514, -0.99726317]]), array([[-1.00000000e+00, -1.73979983e-16,  7.18223703e-10],
       [-2.89806964e-16, -1.00000000e+00,  9.57101065e-10],
       [ 1.24875500e-17,  9.99004001e-18,  1.00000000e+00]]), array([[-1.00000000e+00, -1.73979983e-16,  7.18223703e-10],
       [-2.89806964e-16, -1.00000000e+00,  9.57101065e-10],
       [ 1.24875500e-17,  9.99004001e-18,  1.00000000e+00]])]
Rotation 2 :
[array([[-470.17143817],
       [-626.55004426],
       [  28.99735742]]), array([[470.17143817],
       [626.55004426],
       [-28.99735742]]), array([[-470.21428569],
       [-626.60714283],
       [  27.        ]]), array([[470.21428569],
       [626.60714283],
       [-27.        ]])]
Translation :
[array([[0.00153034],
       [0.00203933],
       [0.99999675]]), array([[-0.00153034],
       [-0.00203933],
       [-0.99999675]]), array([[-4.62501852e-19],
       [-3.70001482e-19],
       [ 1.00000000e+00]]), array([[ 4.62501852e-19],
       [ 3.70001482e-19],
       [-1.00000000e+00]])]
```

Here is where my first instrumental decision will lie, as these are all just estimations there is no one rotation or translation matrix to choose from, however for now I will just use the first one from the Rotation 1 section and the first of the translation sector until I test it at the end, I will then try different combinations to find the best pair for estimation.

The last matrix I need to define until I can start projecting data to a point cloud is the distortion in the image, this is a straightforward task, as we presume there will be little distortion, we do not have to go to great lengths to find it, however, to make the system as refined as possible I will use OpenCV's built in functions to refine the cameras matrix values to create the best possible renderings.

```
41 distortion = np.array([[0.],
42                        [0.],
43                        [0.],
44                        [0.]])
45 camreaMatrix, _ = cv2.getOptimalNewCameraMatrix(camreaMatrix, distortion, (right_image.shape[:2]), 1, (right_image.shape[:2]))
46
```

On line 41, I defined what the distortion value is, I set it to the array [[0], [0], [0], [0]], as this states there is no distortion in the images. Next on line 45, I defined what the new camreaMatrix will be in terms of the distortion, I will now test this feature:

```
Old matrix~~~~~~~~~~~~~~~~~~~
[[ 28   0 454]
 [  0  28 605]
 [  0   0   1]]
New matrix~~~~~~~~~~~~~~~~~~~
[[ 28   0 453]
 [  0  28 604]
 [  0   0   1]]
```

While the difference is small, only decreasing the height and width values by one, the image I used was small and so the larger the image or the more images there are the more beneficial this step is.

Now that I have finished with the backbone of the rendering algorithm, I can move on to rendering the models:

```
47 #This re-projects the poits back to the world plane from their 2D plan
48 points = cv2.reprojectImageTo3D(disp, transformation)
49
```

On line 48, as you can see, I defined a new variable point, this variable contains both the stereo image I created and the transformation matrix I created. I then use the OpenCV command cv2.reprojectImageTo3D to take the stereo image and apply the transformation matrix to it, this will invert what happened when the image was taken and instead of the 3D real world points projecting to a 2D image plain, the 2D image points project to 3D points. This here should now output the real-world cords for the model, here is its test:

```
 [[  0.         528.5       -27.999998  ]
  [  0.87499994 528.5       -27.999998  ]
  [  1.7499999  528.5       -27.999998  ]
  ...
  [394.62497    528.5       -27.999998  ]
  [395.49997    528.5       -27.999998  ]
  [396.37497    528.5       -27.999998  ]]]
Traceback (most recent call last):
  File "C:\Users\alexf\OneDrive\Desktop\my own render.py", line 56, in <module>
    points = points[mask]
IndexError: boolean index did not match indexed array along dimension 0; dimension is 605 but corresponding boolean dimension is 274670
```

When executed this returned with an error, the reason why is I forgot to account for the arrays of arrays having more than just the three values per array I need for x, y, and z values. To fix this I will need to reshape these arrays from [x, y, z, x, y, z, …] to [[x, y, z], [x, y, z], […]].

```
49 points = points.reshape(-1,3)
```

This now uses NumPy to reshape points for me to create the desired array shape, here is the test to make sure that worked:

```
[[  0.           0.         -27.999998  ]
 [  0.87499994   0.         -27.999998  ]
 [  1.7499999    0.         -27.999998  ]
 ...
 [394.62497    528.5        -27.999998  ]
 [395.49997    528.5        -27.999998  ]
 [396.37497    528.5        -27.999998  ]]
>>>
```

The next step in rendering a model is allocating the RGB values for each, pixel, the method to this is very simple, I just need to gather the values from one of the images that was entered, however as we just saw I will reed to reshape these values in the same way as points:

```
48 RGB = left_image.reshape(-1,3)
```

Here is a test to see if it worked:

```
[[127 176 202]
 [128 179 205]
 [128 178 206]
 ...
 [124 175 218]
 [117 166 210]
 [125 174 218]]
>>>
```

Next, I will need to eliminate a lot of the noise from the image, this can be done in few ways however I will be only eliminating the noise which has no depth, and so I will need to apply a mask to points and RGB to eliminate any points they contain which have no depth, this will be done by looking at the disparity map and identifying which points if any have no depth, and if they do not removing that points data from the array of arrays.

```
51 #This is to eliminate any values with depth of 0.
52 deleteNoise = ((disp.reshape(-1)) > (disp.reshape(-1)).min())
```

As you can see, I assigned a new variable deletePoints, this executes what was described in the previous paragraph.

```
54 points = points[deleteNoise]
55 RGB = RGB[deleteNoise]
```

I then applied it to both points, as RGB to delete the noise is needed to create a clearer model.

```
56 projected, _ = cv2.projectPoints(points, RotationMatrix1[0], transitionVectors[0], camreaMatrix, distortion)
```

Next, I created a new variable projected, this will the contain the projected points. To determine the projected points, I used the OpenCV function projectPoints which takes in points, rotation vector matrix, the transition vector, camera matrix and the distortion. It then uses all this to project the points onto a 3D plain. Here is the test of this function:

```
cv2.error: OpenCV(4.4.0) C:\Users\appveyor\AppData\Local\Temp\1\pip-req-build-pz4stnv8\opencv\modules\calib3d\src\calibration.cpp:632:
error: (-5:Bad argument) Intrinsic parameters must be 3x3 floating-point matrix in function 'cvProjectPoints2Internal'
```

During investigations, I found that this error occurred because the camera matrix was out of bounds and the width and height variable had to be smaller. This is how I solved the problem and the tested it after I fixed it.

```
34 camreaMatrix = np.array([[f,0,width/10],
35                          [0,f,height/10],
36                          [0,0,1]])
37
```

```
Progected points~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[[[59.549572 75.78727 ]]

 [[59.511467 76.02838 ]]

 [[59.536896 75.86747 ]]

 ...

 [[29.651697 40.573837]]

 [[29.805687 40.707344]]

 [[29.834677 40.679058]]]
```

The co-ordinates for the x and y values cannot be used by the system just yet. This is because, I will later need to use projected as an index, and indices must be the type of integer or Boolean. The only way to do this is to utilise the astype command within python:

```
58 projected = projected.astype(int)
```

This command creates the output:

```
Progected points~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[[ 62 149]
 [ 65 171]
 [ 64 182]
 ...
 [ 23  32]
 [ 23  32]
 [ 23  32]]
```

Next, I have to now create and apply the final mask to points and colours to finally have a set of data that can be placed into a 3D field. The final mask will need to eliminate any points in projection that are not in the bounds of the image, this means any point smaller than 0, any x larger than the width of the image, and any y point larger than the height of the image. Now to create this mask:

```
60 xPoints, yPoints = projected[:, 0], projected[:, 1]
```

First, I assigned the x values of projected to the variable xPoints and then assigned the y values of projected to yPoints.

```
62 deleteNoise = ((0 <= xPoints & yPoints) &(xPoints < width)&(yPoints < height))
```

Next, I created the changed the mask inside deleteNoise, this mask contains the exact parameters I spoke about earlier, and now all I have to do is apply it to the projected:

```
65 projected = projected[deleteNoise]
```

To make sure this mask is working, as it should, I will look at the shape of projected image before and after deleteNoise is applied to it:

```
Before deleteNoise is applied~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
(198901, 2)
After deleteNoise is applied~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
(192045, 2)
```

As you can see this mask was essential to improve the operation. The mask removed 6856 different erroneous points which would have crashed the render process, as the points were out of bounds. Similar to the error I ran into before I reshaped points. The last part of rendering an image is now slicing the two arrays projected and RGB together.

```
66 #This image handeling was explained from numpys user guide
67 model = np.zeros((height, width, 3), np.uint8)
```

[6]

This piece of code creates model array which will be used to display the final image. This array named model is now shaped with the dimensions of the image and is populated completely with zeros, at the end line 67, I have added the data type uint. This data type represents any unidentified integer between 0 and 256. This is necessary, as until slice projected cords into model all the integers inside of it will be unassigned. Finally, I will need to slice the xPoints and yPoints with the RGB.

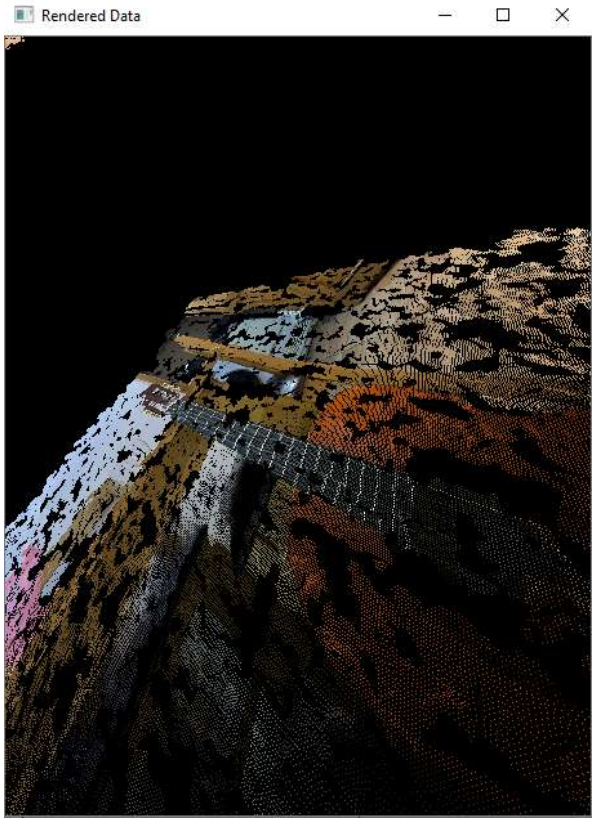
---

6 Martin McBride, 2017-05-12, URL: https://www.pythoninformer.com/python-libraries/numpy/numpy-and-images/

```
70 #All the points in projected are sliced with RGB
71 model[projected[:, 1],projected[:, 0]] = RGB[deleteNoise]
```

This code here takes each y and x cord and slices them with their corresponding RGB value. This now should create a point cloud which has an x, y, z, r, g, and b value. To show this image I will use OpenCV:

```
72 cv2.imshow("Rendered Data", model)
```



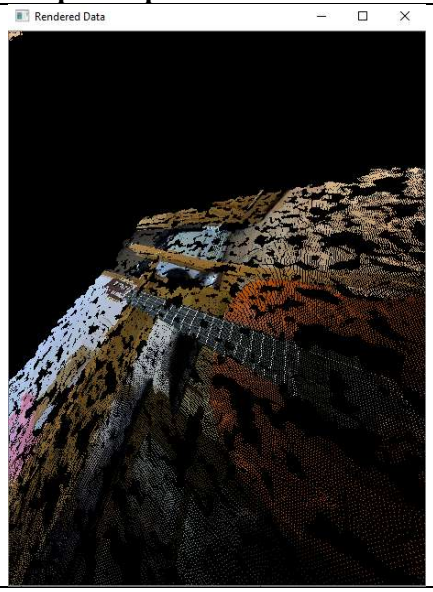
And here is its output:

As you can see the rendering process worked, however the rotation and translation vectors were off, however I expected this, as in the section I found the rotation matrices and translation vectors there were many to choose from, and so I will need to iterate and do some trial-and-error testing to allocate the best fit. However, this rotation has shown me that the blocks are on different levels, and so have depth which means I have created a 3D model.

Criteria points this links to:

- The program should be able to render an image from two still photos.
- A stereo image that also is influenced by the calibrations.
- The x y z r g b values of each pixel.
- A point cloud made from these values.

I will now set about testing the rotation and translation vectors to acquire the best fit for my system:

Rotation vector	Transition Vector	Output- Input data labelled out in design
RotationMatrix1[0],	transitionVectors[0],	

RotationMatrix1[1]	transitionVectors[1]	
RotationMatrix1[2]	transitionVectors[2]	
RotationMatrix1[3]	transitionVectors[3]	
RotationMatrix1[3]	transitionVectors[1]	

After many stages of trial and error I came to the best two vectors being the fourth matrix inside of the acquired rotation matrices and the second transition vector, which gained me the most realistic rendering of the input data.

How can I apply this to my solution?

Now that I have created a method for generating of images, I need to now apply this to my solution. To do this I will need to create an object to which will contain the rendering method, as explained in the design as part of the object orientated design section.

```
1  import cv2
2  import numpy as np
3
4  class RenderedimageData():
5      def __init__(self, left_image, right_image, f):
6          height, width, _ = left_image.shape
7          #Matrices~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
8          transformation = np.float32([[1,0,0,0],
9                   [0,1,0,0],
10                  [0,0,1,0],
11                  [0,0,-1/f,0]])
12
13         homogenized = np.array([[1,0,0],
14                  [0,1,0],
15                  [0,0,-f]])
16
17         camreaMatrix = np.array([[f,0,width/2],
18                [0,f,height/2],
19                [0,0,1]])
20
21         distortion = np.array([[0.],
22                        [0.],
23                        [0.],
24                        [0.]])
25
26 #Opening the images
27 left_image = cv2.imread(r"C:\Users\alexf\OneDrive\Documents\tests\Screenshots\Testing\IMG-0884.jpg")
28 right_image = cv2.imread(r"C:\Users\alexf\OneDrive\Documents\tests\Screenshots\Testing\IMG-0885.jpg")
29 RenderedimageData(left_image, right_image, 28)
```

I first created the class RenderedimageData, I passed both the left and right image and the focal length being used into the initialisation of the class. Then on line 6, I defined the heigh and width of the images, I next added all the needed matrices from line 8 to 24. This is all the universal data I will need to add inside the initialisation.

```
25
26         camreaMatrix = self.OptimalMatrix(camreaMatrix, distortion, right_image)
27
28     def OptimalMatrix(self, camreaMatrix, distortion, right_image):
29         camreaMatrix, _ = cv2.getOptimalNewCameraMatrix(camreaMatrix, distortion, (right_image.shape[:2]), 1, (right_image.shape[:2]))
30         return camreaMatrix
31
```

I next added a function OptimalMatrix into the class RenderedimageData, this class takes in the before set camreaMatrix, distortion and the right_image. I placed into this function OpenCV's method for obtaining the optimal camera matrix and returned the output value. This value was then set to the variable camreaMatrix, which is back inside the initialisation routine. To make sure this works, I will see what the camera matrix value is before and after line 26:

```
Before-----------------------------------------------

 [[ 28.     0.   227. ]
 [  0.    28.   302.5]
 [  0.     0.     1. ]]


After------------------------------------------------

 [[ 27.95371819   0.          226.62477529]
 [  0.          27.93832397 301.83368627]
 [  0.           0.           1.        ]]
```

As you can see the test was successful, and the camera matrix was altered to better fit the distortion as discussed on page 58.

```
34      def rt_Vectors(self, homogenized, camreaMatrix):
35          _, RotationMatrix1, _, transitionVectors = cv2.decomposeHomographyMat(homogenized, camreaMatrix)
36          return RotationMatrix1[3], transitionVectors[1]
```

I next created the function rt_Vectors, this function takes the values of the new camera matrix and the homogenized matrix. It then uses these two factors to discover the rotation and transition vector, as I discussed in the previous section. This function the returns the values of the fourth matrix in the array of matrices rotation matrix and returns the second vector is the array of vectors transition vector.

```
28          rVector, tVector = self.rt_Vectors(homogenized, camreaMatrix)
```

Then inside the class RenderedimageData, I assigned the values rVector and tVector to the output of the function rt_Vectors, to make sure this works I will print both rVector and tVector:

```
Rvec:
 [[-1.00000000e+00  1.73317470e-16 -4.59863259e-11]
 [-0.00000000e+00 -1.00000000e+00 -6.12789819e-11]
 [-0.00000000e+00 -1.99039973e-17  1.00000000e+00]]
Tvec:
 [[-0.00304899]
 [-0.00406309]
 [-0.9999871 ]]
```

I have therefore verified that the function works just as it should.

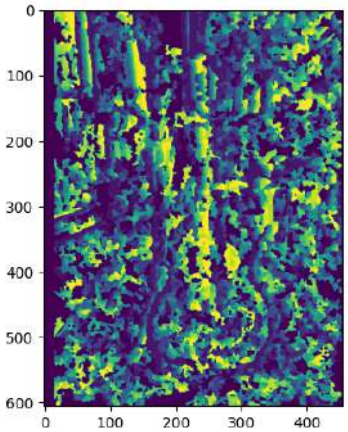
```
42      def depthMap(self, left_image, right_image):
43          #From openCvs StereoSGBM page is where guidences for these values were given.
44          #However alot of trail and error was involved
45          stereo = cv2.StereoSGBM_create(minDisparity = -1,#Minimum possible disparity value
46                                         numDisparities = 16,#Maximum disparity minus minimum disparity
47                                         blockSize = 7  , #This is to control how big the block of pixels is that gets matched
48                                         uniquenessRatio = 1,#How unique a pixel can be (in %) to be matched with another
49                                         speckleWindowSize = 200,#Maximum size of smooth disparity regions to consider their noise speckles and invalidate
50                                         speckleRange = 2,#Maximum disparity variation within each connected component
51                                         disp12MaxDiff = 10,#Maximum allowed difference (in integer pixel units) in the left-right disparity check
52                                         P1 = 8*3*1**2,#The first parameter controlling the disparity smoothness
53                                         P2 =32*3*1**2)#The second parameter controlling the disparity smoothness
54          return stereo.compute(left_image, right_image)
55
```

Next, I added the function for creating the stereo image / depth map, I changed nothing about the actual generation of the images from the last section other than I set the function to return the computed stereo image.

```
30          stereo = self.depthMap(left_image, right_image)
```

I next assigned the returned value of the function depthMap to the variable stereo, to make sure this function works I will be plotting the stereo image using matplotlib to hopefully get the same stereo image I got in the previous section:

```
31          plt.imshow(stereo)
32          plt.show()
```



As you can see the test was successful, the program as of yet functions completely as it should, however the main part is yet to be implemented into the class system.

```
59      def pointsTo3D(self, stereo, transformation):
60          points = cv2.reprojectImageTo3D(stereo, transformation)
61          points = points.reshape(-1,3)
62          return points
```

I next added the function pointsTo3D to the class RenderedimageData, I passed the transformation matrix and the stereo image that I had just created into the function. It the reprojects the points to a 3D plain and reshapes the array of arrays like I explained in the previous section.

```
35          points = self.pointsTo3D(stereo, transformation)
```

I then set the variable points to equal the output of this function, I will now test to see if this function also works as it should by printing the output:

```
[[  0.          0.        -27.999998  ]
 [  0.87499994  0.        -27.999998  ]
 [  1.7499999   0.        -27.999998  ]
 ...
 [394.62497    528.5      -27.999998  ]
 [395.49997    528.5      -27.999998  ]
 [396.37497    528.5      -27.999998  ]]
```

The test worked perfectly with the points being generated being the same as they were in the previous section.

```
 9          RGB = right_image.reshape(-1,3)
```

At the top of the initialisation of RenderedimageData, I added the variable RGB and just like the last section and I set it to be the RGB values of the right_image.

```
37          deleteNoise = ((stereo.reshape(-1)) > (stereo.reshape(-1)).min())
38          points = points[deleteNoise]
39          RGB = RGB[deleteNoise]
```

Then inside of the initialisation I also added the mask to delete noise in both the array points and RGB, to make sure this all still functions, I will print the outputs once again:

```
Points Before: (274670, 3)
RGB Before: (274670, 3)
Points After: (198901, 3)
RGB After: (198901, 3)
```

As you can see, this mask has obviously worked. The points have been reduced by over 75,000 erroneous points which would have disrupted the rendering process.

```
84      def projectPoints(self, points, rVector, tVector, camreaMatrix, distortion):
85          projected, _ = cv2.projectPoints(points, rVector, tVector, camreaMatrix, distortion)
86          projected = projected.reshape(-1, 2)
87          return projected.astype(int)
```

I next added the function projectPoints to the class RenderedimageData, this function finds the x and y cords of each pixel in the image, this step was explained completely in the previous system. The function returns the array projected but all its values are integers.

```
41          projected = self.projectPoints(points, rVector, tVector, camreaMatrix, distortion)
```

Finally, I set the variable projected's value to be the returned value of the function projectPoints. To test this all works I will print the value of projected:

```
[[  0 -22]
 [ -2 -26]
 [ -1 -26]
 ...
 [443 613]
 [444 613]
 [445 613]]
```

The test was successful, which means I can move on to the final stages of rendering a model.

```
82      def filterNoise(self, projected, RGB, xPoints, yPoints, right_image):
83          height, width = right_image.shape[:2]
84          deleteNoise = ((0 <= xPoints & yPoints)&(xPoints < width)&(yPoints < height))
85          return projected[deleteNoise], RGB[deleteNoise]
```

The penultimate stage in rendering an image with this class is filtering out all the noise to remove all erroneous data. I created a new function to filter the noise. As you can see, I passed; projected, RGB, xPoints, yPoints, and right_image into the function filterNoise, I then applied the filter deleteNoise to both projected and RGB, just as I did inside the previous section.

```
42          xPoints, yPoints = projected[:, 0], projected[:, 1]
43
44          projected, RGB = self.filterNoise(projected, RGB, xPoints, yPoints, right_image)
```

I then assigned the values for xPoints, yPoints, projected and RGB inside of the initialisation of the RenderedimageData class. I assigned projected and RGB to the returned values of filterNoise.

```
83      def createModel(self, height, width, projected, RGB):
84          model = np.zeros((height, width, 3), np.uint8)
85          model[projected[:, 1],projected[:, 0]] = RGB
86          cv2.imshow("Rendered Data", model)
87
```

The final function is createModel, this function slices together the both the projected values and RGB, it then finally displays the model. I passed the height, width, projected and finally the RGB values into this function.

```
46          self.createModel(height, width, projected, RGB)
47
```

I then added the command createModel inside the initialisation of RenderedimageData, and finally the class is finished. However, this is not the final prototype for the code. I will still need to create more code to augment this method to allow for the creation of multiple point clouds and the layering of them over each other. This will not be an easy task, but for now this will be the last changes I make to this version of the code.

Testing

Here is how I will be testing the method:

What I am testing	Data Entered	Data Type (erroneous, boundary, normal)	Result	Was this expected
The creation of a rendering with 2 images.	 	Normal		Yes

Code for this section:

```
import cv2
import numpy as np

class RenderedimageData():
    def __init__(self, left_image, right_image, f):
        #Image~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        height, width, _ = left_image.shape
        RGB = right_image.reshape(-1,3)
        #Matrices~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        transformation = np.float32([[1,0,0,-width/2],
                [0,1,0,-height/2],
                [0,0,0,-f],
                [0,0,-1/100,0]])

        homogenized = np.array([[1,0,0],
                [0,1,0],
                [0,0,-f]])

        camreaMatrix = np.array([[f,0,width/2],
              [0,f,height/2],
              [0,0,1]])

        distortion = np.array([[0.],
                     [0.],
                     [0.],
                     [0.]])

        camreaMatrix = self.OptimalMatrix(camreaMatrix, distortion, right_image)

        rVector, tVector = self.rt_Vectors(homogenized, camreaMatrix)

        stereo = self.depthMap(left_image, right_image)

        points = self.pointsTo3D(stereo, transformation)

        deleteNoise = ((stereo.reshape(-1)) > (stereo.reshape(-1)).min())
        points = points[deleteNoise]
        RGB = RGB[deleteNoise]

        projected = self.projectPoints(points, rVector, tVector, camreaMatrix, distortion)

        xPoints, yPoints = projected[:, 0], projected[:, 1]

        projected, RGB = self.filterNoise(projected, RGB, xPoints, yPoints, right_image)

        self.createModel(height, width, projected, RGB)

    def OptimalMatrix(self, camreaMatrix, distortion, right_image):
        camreaMatrix, _ = cv2.getOptimalNewCameraMatrix(camreaMatrix, distortion, (right_image.shape[:2]), 1, (right_image.shape[:2]))
        return camreaMatrix

    def rt_Vectors(self, homogenized, camreaMatrix):
        _, RotationMatrix1, _, transitionVectors = cv2.decomposeHomographyMat(homogenized, camreaMatrix)
        return RotationMatrix1[3], transitionVectors[1]

    def depthMap(self, left_image, right_image):
        stereo = cv2.StereoSGBM_create(minDisparity = -1,
                                       numDisparities = 16,
                                       blockSize = 7  ,
                                       uniquenessRatio = 1,
                                       speckleWindowSize = 200,
```

```
                                    speckleRange = 2,
                                    disp12MaxDiff = 10,
                                    P1 = 8*3*1**2,
                                    P2 =32*3*1**2)
        return stereo.compute(left_image, right_image)

    def pointsTo3D(self, stereo, transformation):
        points = cv2.reprojectImageTo3D(stereo, transformation)
        points = points.reshape(-1,3)
        return points

    def projectPoints(self, points, rVector, tVector, camreaMatrix, distortion):
        projected, _ = cv2.projectPoints(points, rVector, tVector, camreaMatrix, distortion)
        projected = projected.reshape(-1, 2)
        return projected.astype(int)

    def filterNoise(self, projected, RGB, xPoints, yPoints, right_image):
        height, width = right_image.shape[:2]
        deleteNoise = ((0 <= xPoints & yPoints)&(xPoints < width)&(yPoints < height))
        return projected[deleteNoise], RGB[deleteNoise]

    def createModel(self, height, width, projected, RGB):
        model = np.zeros((height, width, 3), np.uint8)
        model[projected[:, 1],projected[:, 0]] = RGB
        cv2.imshow("Rendered Data", model)

#Opening the images
left_image = cv2.imread(r"C:\Users\alexf\OneDrive\Documents\tests\Screenshots\Testing\IMG-0884.jpg")
right_image = cv2.imread(r"C:\Users\alexf\OneDrive\Documents\tests\Screenshots\Testing\IMG-0885.jpg")
RenderedimageData(left_image, right_image, 28)
```

Phase 1 Review:

What has been done?

I have developed and researched my own method for developing renderings and inside of a python file I created a class for the rendering of images, this class however is only able to create a singular rendering from two images and doesn't allow the user to alter the calibration options of the stereo image creation, however this will all be accomplished in later stages.

How has it been tested?

Each time I added a new section to the rendering algorithm, I ran the program to see if it worked and see there were any errors. The stereo image creation has been tested with normal data, and the final rendering algorithm was also tested with normal data (as laid out in the testing table), with each test I have provided what I was tested, the data that I entered, the result and if that result was expected.

How it meets the success criteria and user expectations

I have demonstrated that progress has been made and it's a good *start* to meet my design criteria:

- The program should be able to render an image from two still photos.
- A stereo image that also is influenced by the calibrations.
- The x y z r g b values of each pixel.
- A point cloud made from these values.
- The rendered image will have to be clear and recognisable.

Changes in the design that have resulted from this section.

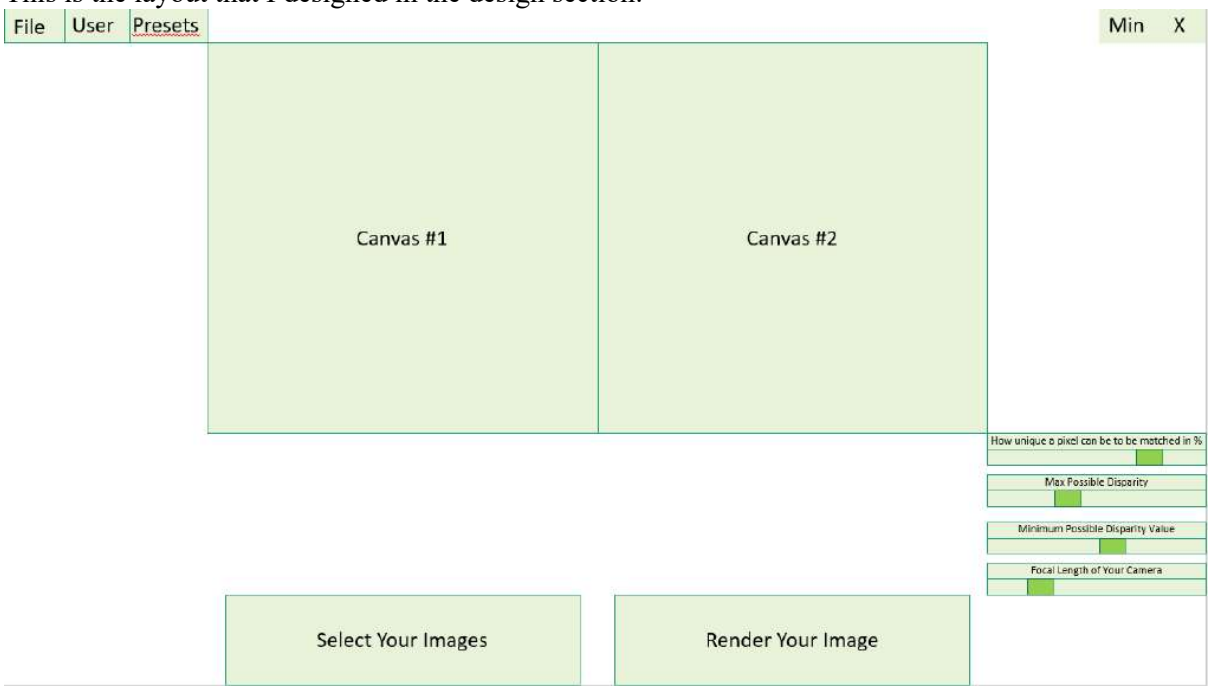
This section has followed the design so far – no changes yet to be identified.

Summary of the whole project at this phase

The rendering algorithm has been created and placed into a class; the program is only able to render two images at this point however videos and subsequently more images will be looked at later in the phases. The testing showed that the point clouds have some empty space and so a bit of black shows through on the rendered image, however the renderings are still recognisable and so this should not present me with any significant future development problems.

Phase 2, main window user interface

This is the layout that I designed in the design section:



I will be using Tkinter (a built-in python function) to create the user interface. I have made a simplistic Tkinter class that fits the specification of the layout above.

```
import tkinter as tk
import tkinter.font as font
from tkinter import filedialog
from PIL import ImageTk, Image
class windows(tk.Tk):

    def __init__(self):
        tk.Tk.__init__(self)
        container = tk.Frame(self)

        container.pack(side = "top", fill = "both", expand = False )
        container.grid_rowconfigure(0, weight = 1)
        container.grid_columnconfigure(0, weight = 1)

        self.frames = {}

        frame = mainwindow(container, self)

        self.frames[mainwindow] = frame

        frame.grid(row = 15, column = 15, sticky = "nsew")

        self.show_frame(mainwindow)


    def show_frame(self,cont):
        frame = self.frames[cont]
        frame.tkraise()

class mainwindow(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent, bg = "#596387")
        self.maxScreen = True
        #Fonts-------------
        ButtonFont = font.Font(size = 24, family="Helvetica")
        MenuFont = font.Font(size = 12, family="Helvetica")
        LabelFont = font.Font(size = 10, family = "Helvetica")
        #Buttons-------
        self.minmaxbutton = tk.Button(self,
                                      text = "Min", font = MenuFont)
        self.minmaxbutton.grid(row = 0, column= 12)

        self.exitbutton = tk.Button(self,
                              text = "X",  font = MenuFont).grid(row = 0, column = 14)
        self.RenderImageBtn = tk.Button(self,
                                        text="Render Your Image", state = "disabled", font = ButtonFont, height = 3, width = 30 )
        self.RenderImageBtn.grid(row = 13, column = 6 )
        self.SelectImageBtn = tk.Button(self,
                                        text = "Select Your Images", font = ButtonFont, height = 3, width = 30 )
        self.SelectImageBtn.grid(row = 13, column = 5)
        #Dropdown selections----------
        Fileoptions = ["Video File", "Save", "Open", "New File"]
        Useroptions = ["Log Out"]
        Presetsoptions = ["Save as Preset", "Open Preset"]

        self.Fileclicked = tk.StringVar()
        self.Userclicked = tk.StringVar()
        self.Presetsclicked = tk.StringVar()

        self.Fileclicked.set("File")
        self.Userclicked.set("User")
```

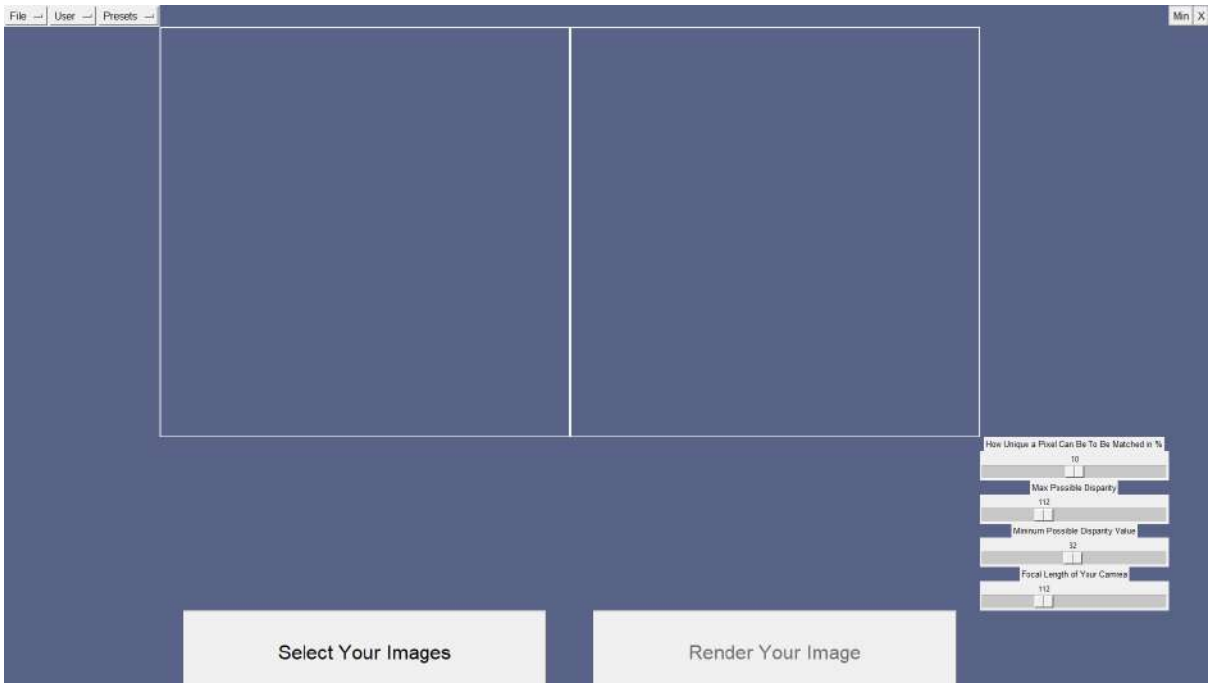
```
        self.Presetsclicked.set("Presets")

        filedrop = tk.OptionMenu(self,
                                 self.Fileclicked, *Fileoptions)
        filedrop.config(font = MenuFont)
        userdrop = tk.OptionMenu(self,
                                 self.Userclicked, *Useroptions)
        userdrop.config(font = MenuFont)
        presetsdrop = tk.OptionMenu(self,
                                    self.Presetsclicked, *Presetsoptions)
        presetsdrop.config(font = MenuFont)

        filedrop.grid(row = 0, column= 0)
        userdrop.grid(row = 0, column= 2)
        presetsdrop.grid(row = 0, column= 4)
        #Canvas----------------
        self.LCanvas = tk.Canvas(self,
                                 bg = "#596387", height = 650, width = 650)
        self.LCanvas.grid(row = 1, column = 5 )
        self.RCanvas = tk.Canvas(self,
                                 bg = "#596387", height = 650, width = 650)
        self.RCanvas.grid(row = 1, column = 6)
        #Sliders-----------------
        self.minDispSlider = tk.Scale(self,
                                      from_=1, to = 64, orient = "horizontal", length = 295, width = 20)
        self.minDispSlider.set(32)
        self.minDispSlider.grid(row = 10, column = 10)
        self.maxDispSlider = tk.Scale(self,
                                      from_=16, to = 320, orient = "horizontal", length = 295, width = 20)
        self.maxDispSlider.set(112)
        self.maxDispSlider.grid(row = 8, column = 10)
        self.focalSlider = tk.Scale(self,
                                      from_=16, to = 320, orient = "horizontal", length = 295, width = 20)
        self.focalSlider.set(112)
        self.focalSlider.grid(row = 12, column = 10)
        self.uniqueRatioSlider = tk.Scale(self,
                                          from_=5, to = 15, orient = "horizontal", length = 295, width = 20)
        self.uniqueRatioSlider.set(10)
        self.uniqueRatioSlider.grid(row = 6, column = 10)
        #Labels------------------------
        self.minDispLabel = tk.Label(self,
                                     text = "Minimum Possible Disparity Value", font = LabelFont).grid(row = 9, column = 10)
        self.maxDispLabel = tk.Label(self,
                                     text = "Max Possible Disparity", font = LabelFont).grid(row = 7, column = 10)
        self.focalLabel = tk.Label(self,
                                   text = "Focal Length of Your Camrea", font = LabelFont).grid(row = 11, column = 10)
        self.uniqueRatioLabel = tk.Label(self,
                                         text = "How Unique a Pixel Can Be To Be Matched in %", font = LabelFont).grid(row = 5, column = 10)

app = windows()
app.attributes("-fullscreen", True)
app.mainloop()
```

This code above creates this window:



I first imported the required libraries for the UI, that tkinter provides which is used in the basic display of the window, buttons, labels, canvases, and sliders. tkinters subsection font which is used to control the size and font family of the text, and Pillow which is used to display images libraries have also been used. I first created a class called windows, and in the initialisation of this class it creates and defines the basic parameters of each frame. In this class, I also defined how each frame will be shown with the sub-program show_frame:

```
26      def show_frame(self,cont):
27          frame = self.farmes[cont]
28          frame.tkraise()
```

When the name of the desired frame is called, frame.tkrasie raises that frame to the front.

I then created another class for main window called mainwindow, and in the initialisation of this class the layout is created. Each item in this window has at least two lines dedicated to it; one is used to define what it is (for example self.SelectImageBtn = tk.Button(self, text = "Select Your Images", font = ButtonFont, height = 3, width = 30) and the second states where it will be placed in the window (for example self.SelectImageBtn.grid(row = 13, column = 5)). For this window, I have used tkinters grid layout system which gives each item is given a row number and column number.

The layout of mainwindow is the same as I designed previously. The two large black spaces are placeholder images until the user enters their own images which as of yet is not working.

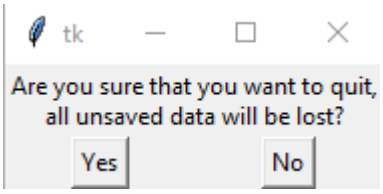
This is the image which will be used to fill the blank space:

I will not be addressing this issue now; however, I am going to look at a solution later because I do not know yet what the best way of doing this is. Later in the development of my program, I will be using these spaces to display videos which may require a completely different method to just displaying static images. I will need to investigate this.

I will now create the pop-up windows needed for the exit button. To do this, I will need a new class for this window and a function in the class mainwindow to open it.

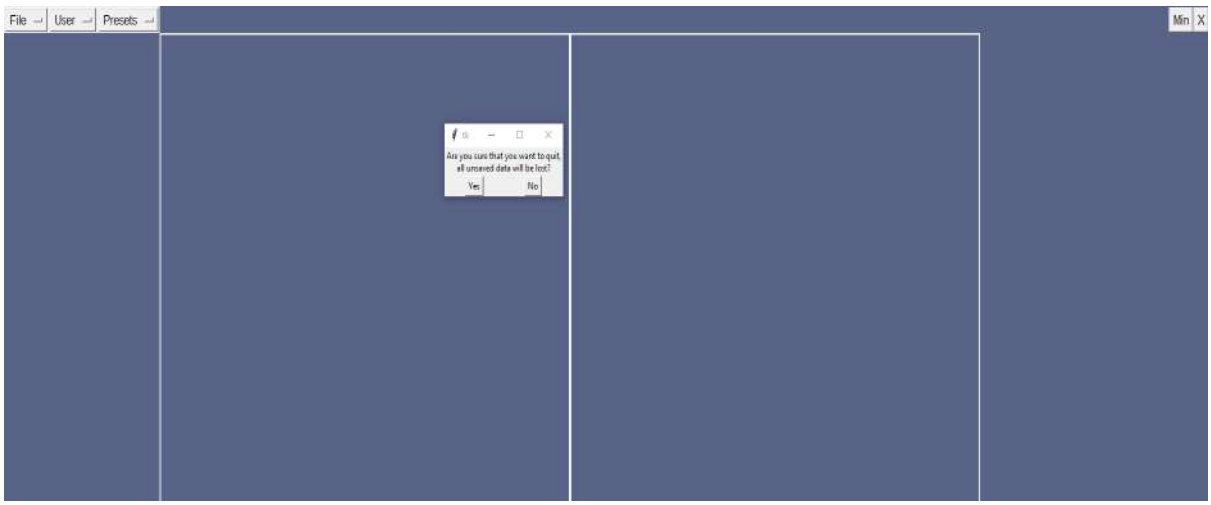
```
112 class quitWindow():
113
114     def __init__(self):
115
116         self.quit = tk.Toplevel()
117
118         Message = tk.Label(master=self.quit,
119                            text= """Are you sure that you want to quit,
120 all unsaved data will be lost?""").grid(row=1, column=1, columnspan=2)
121         quitButton = tk.Button(master= self.quit ,
122                            text="Yes",
123                            command = self.quitALL).grid(row=2, column=1)
124         cancelButton = tk.Button(master= self.quit,
125                            text="No",
126                            command = lambda: self.quit.destroy()).grid(row=2, column=2)
127
128     def quitALL(self):
129         app.destroy()
130
```

I created a class called quitWindow, and in the initialisation of this class the layout is created and the raised. On line 116, the pop-up window is created and is raised to the top level, then on line 118 a warning message is created. The message states that if the user has not saved their file it will be lost, then on lines 121 and 124, two buttons are defined both with commands; firstly, on line 121 the quit button is established with the command set to the sub-subprogram quitALL which as you can see on line 128-129 clears all windows. On line 124, the cancel button is established which clears only the pop-up window. This is the interface quitWindow creates:



Then in the mainwindow I have append a command to the exit button (added command = quitWindow):

```
43        self.exitbutton = tk.Button(self,
44                                    command = quitWindow, text = "X",  font = MenuFont).grid(row = 0, column = 14)
```



The end result is this pop-up window which appears in front of the main window. Using this method of creating these windows, allows for the pop-up window to be called by any window which I create, saving time taken in writing a closing sequence for all windows.

Next button I will be addressing is the minimization button found on lines 39-41. For this button, I will need to create another class, that when called changes the attributes of the window from full screen = true to false.

Inside of the class windows (found at line 7), I have added a new global variable maxScreen, which is set to true, this is because my windows will start off as full screen.

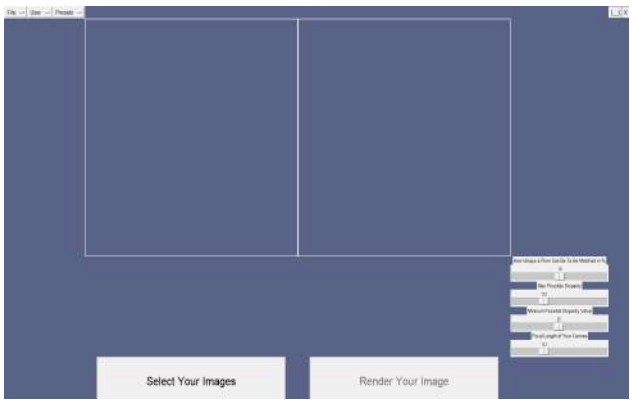
```
7  class windows(tk.Tk):
8
9      def __init__(self):
10         tk.Tk.__init__(self)
11         container = tk.Frame(self)
12
13         container.pack(side = "top", fill = "both", expand = False )
14         container.grid_rowconfigure(0, weight = 1)
15         container.grid_columnconfigure(0, weight = 1)
16
17         self.farmes = {}
18
19         frame = mainwindow(container, self)
20
21         self.farmes[mainwindow] = frame
22
23         frame.grid(row = 15, column = 15, sticky = "nsew")
24
25         self.show_frame(mainwindow)
26
27         global maxScreen
28         maxScreen = True
```

Next, I have created a class named minmax_frame, and in the initialisation of this class, I reference the global variable again and then used a basic if, elif statement to check what the status of maxScreen was (True/False):

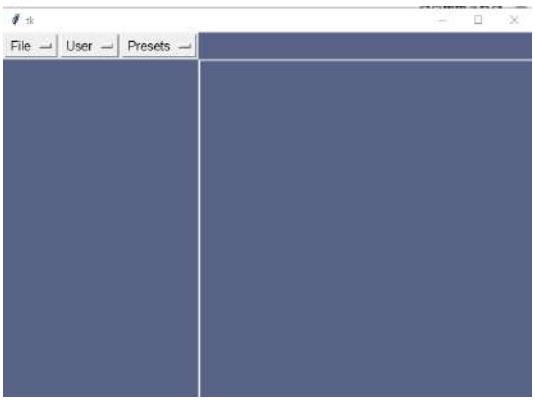
```
136 class minmax_frame():
137         def __init__(self):
138             global maxScreen
139             if maxScreen == True:
140                 app.attributes("-fullscreen", False)
141                 maxScreen = False
142             elif maxScreen == False:
143                 maxScreen = True
144                 app.attributes("-fullscreen", True)
145
```

Then in the mainwindow I have appended a command to the minmax button (added command = minmax_frame and changed text = [__]):

```
44         self.minmaxbutton = tk.Button(self,
45                                       command = minmax_frame, text = "[__]", font = MenuFont)
46         self.minmaxbutton.grid(row = 0, column= 12)
```



Before



After

The final result is a button which can appear on every page that minimizes or maximizes the page, this again like the exit button is useful in a class method, as it's a lot more effective than rewriting the same code for each window.

The next and penultimate part of the main window, I will be addressing the select your images button. For this button, I will need to append to the class mainwindow and create a sub-program inside of it. This will allow users to select the images they want to use.

```
115
116     def imageSelector(self):
117         self.imagePathStr = []
118         imagePath = filedialog.askopenfilename()
119         while len(imagePath) > 0 or len(self.imagePathStr) < 2:
120             self.imagePathStr.append(imagePath)
121             imagePath = filedialog.askopenfilename()
122         length = (len(self.imagePathStr) - 1)
123         print((self.imagePathStr))
124         try:
125             while length >= 0:
126                 imageCheck = Image.open((self.imagePathStr[length]))
127                 length -= 1
128             pathL = Image.open((self.imagePathStr[0]))
129             pathR = Image.open(self.imagePathStr[(len(self.imagePathStr)-1)])
130
131             NewPathR = pathR.resize((650,650))
132             NewPathL = pathL.resize((650,650))
133
134             self.RCanvas.image = ImageTk.PhotoImage(NewPathR)
135             self.RCanvas.create_image(0,0, image = self.RCanvas.image, anchor = "nw")
136
137             self.LCanvas.image = ImageTk.PhotoImage(NewPathL)
138             self.LCanvas.create_image(0,0, image = self.LCanvas.image, anchor = "nw")
139
140             self.RenderImageBtn.configure(state = "active", bg = "light green")
141         except:
142             errorPopup = tk.Toplevel()
143
144             errorMessage = tk.Label(master=errorPopup,
145                                     text= """Error: One Of the file directorys given
146 was not an image please try again""").grid(row=1, column=1, columnspan=2)
147             cancelButton = tk.Button(master= errorPopup,
148                                     text="Ok",
149                                 command = lambda: errorPopup.destroy()).grid(row=2, column=2)
150
```

First an empty string is created called imagePathStr (this is to hold all the image directory files), then the program opens the users file directory utilising tkinters file dialog import. Next a while loop is created to check that the user has chosen a directory path with a length of more than zero characters, however if they have not done that and they have already chosen two or more images (two images is the minimal amount required to render an image) the program starts a try and except statement. In this statement, if the user has chosen any file paths which do not contain images, the imageCheck found on line 126, will generate an error and a pop-up window will be created as is shown in lines 142-149.

When tested with this erroneous input data as you can see the correct response was given.

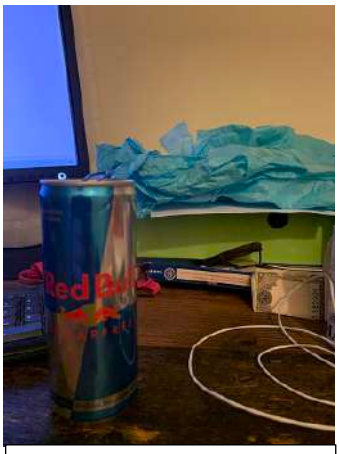
However, if correct data was inputted the program should set the first image in the string imagePathStr to the left canvas, as it should be the left most image in the sequence and should set the last image in the string to the right canvas.

```
= RESTART: C:/Users/alexf/OneDrive/Documents/tests/Screenshots/GUIS/Main GUI.py
['C:/Users/alexf/Downloads/IMG-1184.jpg', 'C:/Users/alexf/Downloads/IMG-1185.jpg',
'C:/Users/alexf/Downloads/IMG-1186.jpg', 'C:/Users/alexf/Downloads/IMG-1187.jpg']
```

With this data inputted:

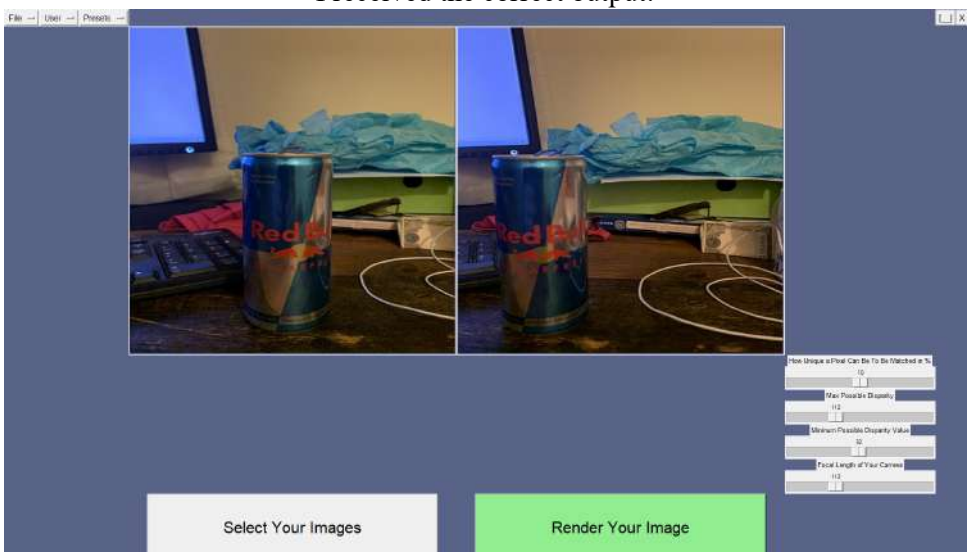
These should be the respective left and right canvas pictures.

IMG-1184.jpg



IMG-1187.jpg

I received the correct output:



The final part of phase 2, will be creating the function which allows the user to choose a video instead of a selection of images. For this, I need to append to the mainwindow class and define a new sub-routine to allow the user to choose a video instead of images.

```
150
151     def videoSelector(self):
152         self.video = filedialog.askopenfilename()
153         if len(self.video) > 0:
154             self.imagePathStr = []
```

As in imageSelector, the sub-program utilises the file dialog import of python, and once again checks if the length of the path give is longer than zero characters and if it is, we define imagePathStr as an empty string. After this we need to partition the video selected into its individual frames.

```
166     def createframes(self, video):
167       Video = cv2.VideoCapture(self.video)
168       self.maxFrames = int(Video.get(cv2.CAP_PROP_FRAME_COUNT))
169       success, image = Video.read()
170       self.count = 0
171       while success:
172         image = cv2.resize(image,(650,650))
173         cv2.imwrite("images/frame%d.jpg" % self.count, image)
174         self.imagePathStr.append("images/frame%d.jpg" % self.count)
175         self.count += 1
176         success, image = Video.read()
```

At line 116, I created a new sub-routine called createframes inside the class mainwindow, into this routine I passed through the video selected by the user. Next utilising OpenCV, I selected the video passed it into cv2.VideoCapture which reads the video, then I assigned the variable maxFrames to the value of the total frame count of the video. On line 169, both variables success and image were defined, both were assigned Video.read, which looks at each frame of the chosen video, while there are frames in the video left to read the while loop at line 171 will continue. Each time it loops, the frame is resized to a more manageable size for both time efficiency and convenience later in the program, then the image is saved in the file directory “images/frames%.jpg” with the % standing for what number the frame is. I need to now call this function inside of the videoSelector routine.

```
152
153     def videoSelector(self):
154         self.video = filedialog.askopenfilename()
155         if len(self.video) > 0:
156             try:
157                 self.imagePathStr = []
158                 self.createframes(self.video)
159
160                 pathL = Image.open((self.imagePathStr[0]))
161                 pathR = Image.open(self.imagePathStr[(len(self.imagePathStr)-1)])
162
163                 NewPathR = pathR.resize((650,650))
164                 NewPathL = pathL.resize((650,650))
165
166                 self.RCanvas.image = ImageTk.PhotoImage(NewPathR)
167                 self.RCanvas.create_image(0,0, image = self.RCanvas.image, anchor = "nw")
168
169                 self.LCanvas.image = ImageTk.PhotoImage(NewPathL)
170                 self.LCanvas.create_image(0,0, image = self.LCanvas.image, anchor = "nw")
171
172                 self.RenderImageBtn.configure(state = "active", bg = "light green")
173                 print (self.imagePathStr)
174             except:
175                 VideoerrorPopup = tk.Toplevel()
176
177                 errorMessage = tk.Label(master = VideoerrorPopup,
178                                         text= """Error: The file directory given
179     was not a video please try again""").grid(row=1, column=1, columnspan=2)
180                 cancelButton = tk.Button(master = VideoerrorPopup,
181                                         text="Ok",
182                                          command = lambda: VideoerrorPopup.destroy()).grid(row=2, column=2)
183
```

I have also appended to the sub-routine videoSelector a try and except statement, this is to prevent errors breaking the program and to show a dialog box with the error that has occurred. I also added a method of showing the user what video they uploaded, I did this in the same way as in imageSelector with the left canvas being the first frame and the right canvas being the last.

The next task required is to append the mainwindow class again and create another sub-program, this time to manage the option menu selections, in particular the option Change File Type (an option under File menu) to allow the user to choose to select a video.

```
37 class mainwindow(tk.Frame):
38     def __init__(self, parent, controller):
39         tk.Frame.__init__(self, parent, bg = "#596387")
40         self.ImageFile = True
```

Inside of the initialisation of the class mainwindow, I have added a new variable, ImageFile. I have used this variable to help determine what file type the program is (Image or Video). The variable will start off beginning assigned as True, this will indicate that the file is accepting a sequence of images, and when the variable is set as False this will show the file is accepting a video instead.

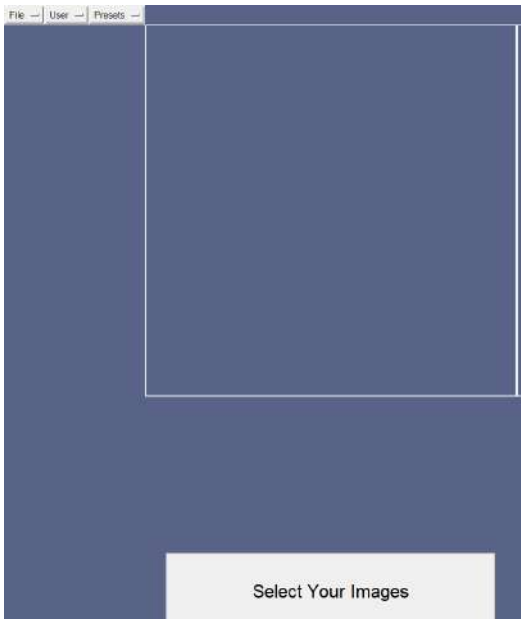
```
198
199     def changeFileType(self):
200         if self.ImageFile == True:
201             self.ImageFile = False
202             self.SelectImageBtn.config(command = self.videoSelector,
203                                        text = "Select Your Video")
204         else:
205             self.ImageFile = True
206             self.SelectImageBtn.config(command = self.imageSelector,
207                                        text = "Select Your Images")
208
```

Inside of the class mainwindow, I defined a new subroutine, changeFileType, this routine is responsible for changing the text and commands of the button SelectImageBtn. When called the routine looks to determine if ImageFile is True or not, and as discussed previously, depending if it's true or false it will either change the command and text of the button to fit, if you wanted to upload an image (lines 205-207) or if you were to upload a video (lines 200-203).

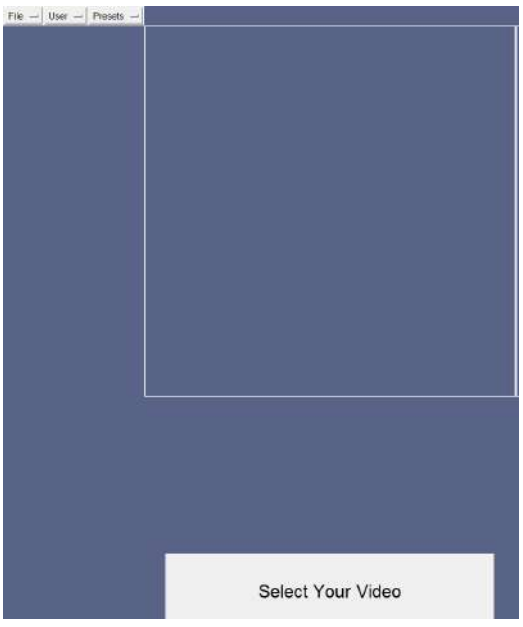
```
208
209     def OptionChange(self, event):
210         if self.Fileclicked.get() == "Change File Type":self.Fileclicked.set("File"), self.changeFileType()
211
```

Inside of the class mainwindow I have defined the event OptionChange. When the option is clicked it executes a command. In this case, when Change File Type is clicked inside of menu File, the File menu icon is set to remain as File and the subroutine changeFileType is executed.

After selecting the menu File, then clicking the option this was the outcome:

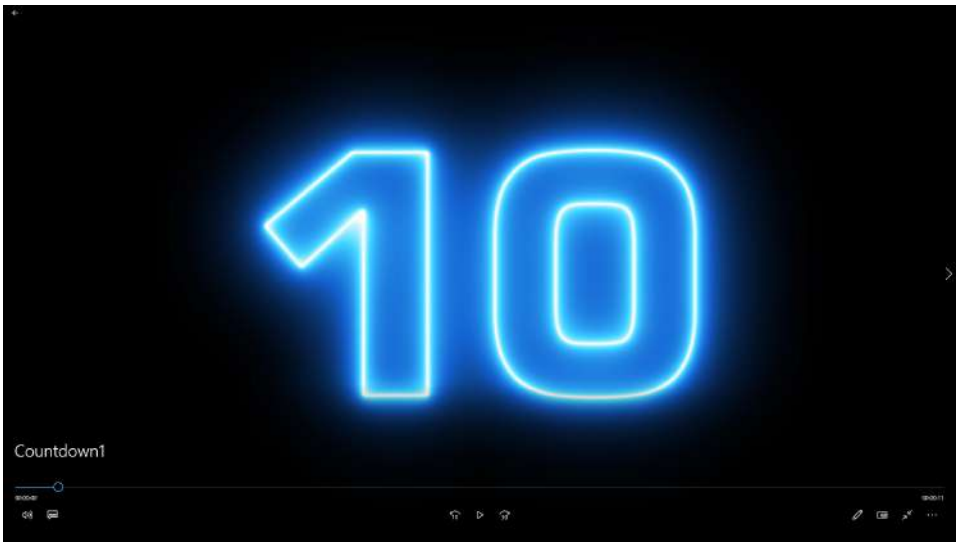


Before



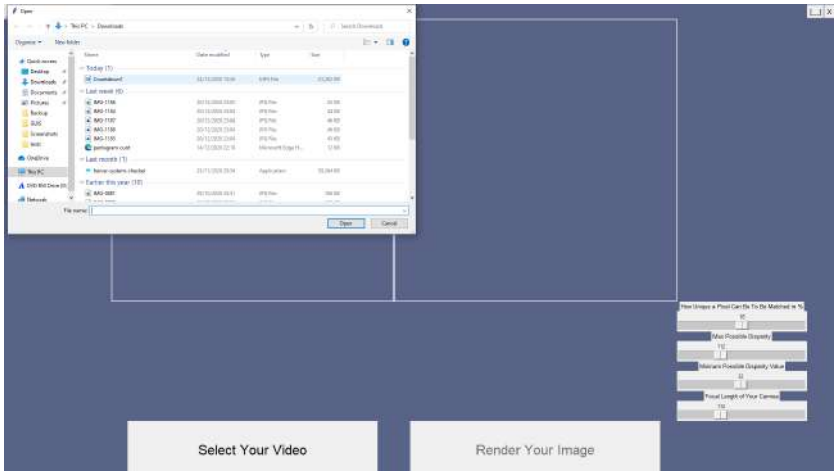
After

Next, I will be testing if the selection of a video works, for this I will pass through a video I found online of a ten second count down. The video looks as follows:

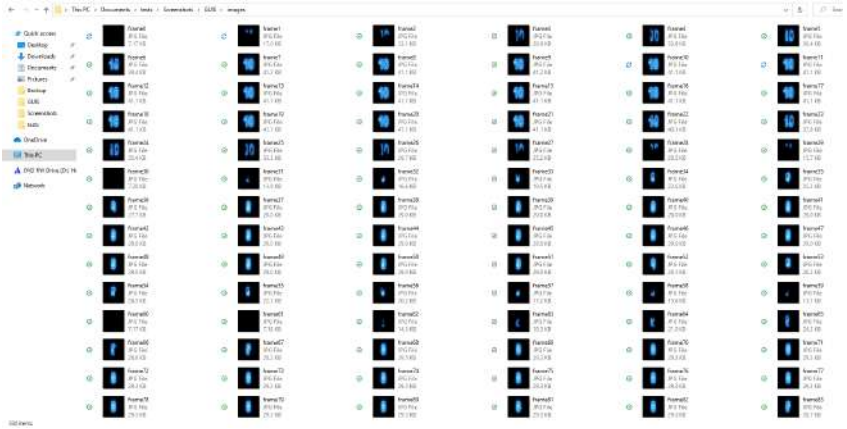


When the image is selected the program should save all frames to a folder named images located in the same folder as the code, and once all the frames have been spilt apart and saved the program should display the first and last frame of the video. Here were the results:

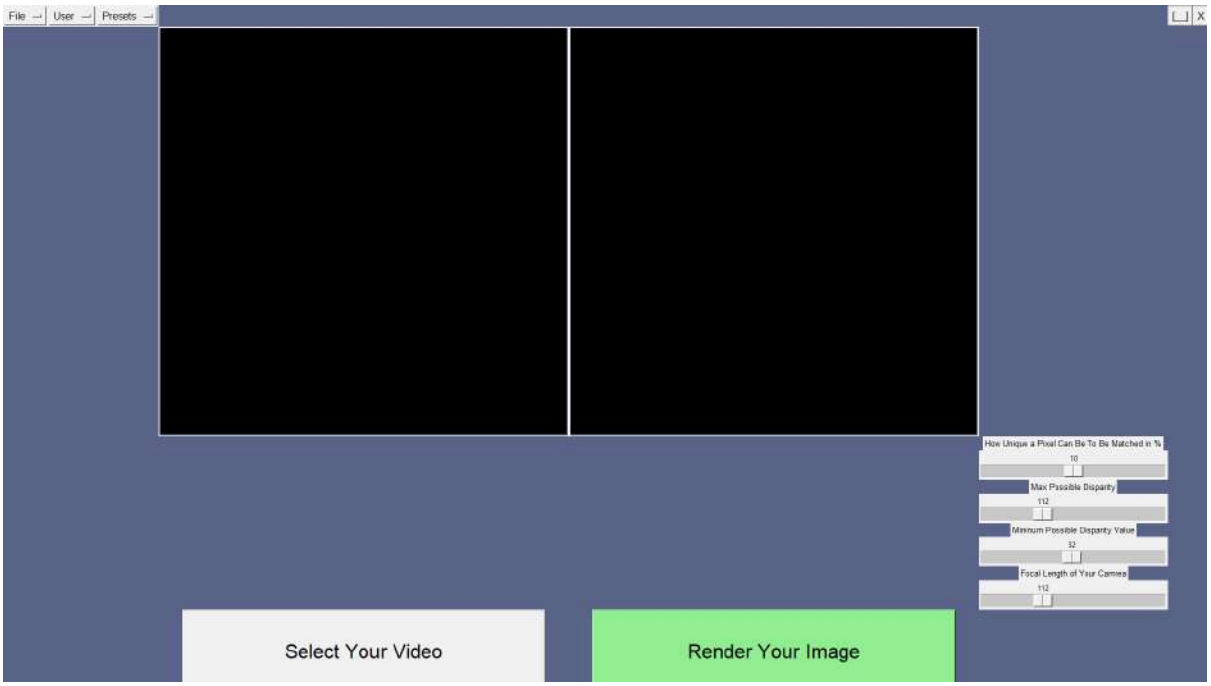
First, I selected the video:



This is a sample of the frames saved to the folder as images:



Layout and functionality of the final UI:



The program functioned as per the design of the UI.

Main Class code:

```
import tkinter as tk
import tkinter.font as font
from tkinter import filedialog
from PIL import ImageTk, Image
import cv2
import time


class windows(tk.Tk):

    def __init__(self):
        tk.Tk.__init__(self)
        container = tk.Frame(self)

        container.pack(side = "top", fill = "both", expand = False )
        container.grid_rowconfigure(0, weight = 1)
        container.grid_columnconfigure(0, weight = 1)

        self.farmes = {}

        frame = mainwindow(container, self)

        self.farmes[mainwindow] = frame

        frame.grid(row = 15, column = 15, sticky = "nsew")

        self.show_frame(mainwindow)

        global maxScreen
        maxScreen = True


    def show_frame(self,cont):
        frame = self.farmes[cont]
        frame.tkraise()

class mainwindow(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent, bg = "#596387")
        self.ImageFile = True
        #Fonts-------------
        ButtonFont = font.Font(size = 24, family="Helvetica")
        MenuFont = font.Font(size = 12, family="Helvetica")
        LabelFont = font.Font(size = 16, family = "Helvetica")
        #Buttons--------
        self.minmaxbutton = tk.Button(self,
                                      command = minmax_frame, text = "[__]", font = MenuFont)
        self.minmaxbutton.grid(row = 0, column= 12)

        self.exitbutton = tk.Button(self,
                                    command = quitWindow, text = "X",  font = MenuFont).grid(row = 0, column = 14)
        self.RenderImageBtn = tk.Button(self,
                                        text="Render Your Image", state = "disabled", font = ButtonFont, height = 3, width = 30 )
        self.RenderImageBtn.grid(row = 13, column = 6 )
        self.SelectImageBtn = tk.Button(self,
                                        command = self.imageSelector, text = "Select Your Images", font = ButtonFont, height = 3, width = 30 )
        self.SelectImageBtn.grid(row = 13, column = 5)
        #Dropdown selections-----------
        Fileoptions = ["Change File Type", "Save", "Open", "New File"]
        Useroptions = ["Log Out"]
        Presetsoptions = ["Save as Preset", "Open Preset"]
```

```
        self.Fileclicked = tk.StringVar()
        self.Userclicked = tk.StringVar()
        self.Presetsclicked = tk.StringVar()

        self.Fileclicked.set("File")
        self.Userclicked.set("User")
        self.Presetsclicked.set("Presets")

        self.filedrop = tk.OptionMenu(self,
                                      self.Fileclicked, *Fileoptions, command = self.OptionChange)
        self.filedrop.config(font = MenuFont)
        userdrop = tk.OptionMenu(self,
                                 self.Userclicked, *Useroptions)
        userdrop.config(font = MenuFont)
        presetsdrop = tk.OptionMenu(self,
                                    self.Presetsclicked, *Presetsoptions)
        presetsdrop.config(font = MenuFont)

        self.filedrop.grid(row = 0, column= 0)
        userdrop.grid(row = 0, column= 2)
        presetsdrop.grid(row = 0, column= 4)
        #Canvas----------------
        self.LCanvas = tk.Canvas(self,
                                 bg = "#596387", height = 650, width = 650)
        self.LCanvas.grid(row = 1, column = 5 )
        self.RCanvas = tk.Canvas(self,
                                 bg = "#596387", height = 650, width = 650)
        self.RCanvas.grid(row = 1, column = 6)
        #Sliders-----------------
        self.minDispSlider = tk.Scale(self,
                                      from_=1, to = 64, orient = "horizontal", length = 295, width = 20)
        self.minDispSlider.set(32)
        self.minDispSlider.grid(row = 10, column = 10)
        self.maxDispSlider = tk.Scale(self,
                                      from_=16, to = 320, orient = "horizontal", length = 295, width = 20)
        self.maxDispSlider.set(112)
        self.maxDispSlider.grid(row = 8, column = 10)
        self.focalSlider = tk.Scale(self,
                                      from_=16, to = 320, orient = "horizontal", length = 295, width = 20)
        self.focalSlider.set(112)
        self.focalSlider.grid(row = 12, column = 10)
        self.uniqueRatioSlider = tk.Scale(self,
                                          from_=5, to = 15, orient = "horizontal", length = 295, width = 20)
        self.uniqueRatioSlider.set(10)
        self.uniqueRatioSlider.grid(row = 6, column = 10)
        #Labels------------------------
        self.minDispLabel = tk.Label(self,
                                     text = "Minimum Possible Disparity Value", font = LabelFont).grid(row = 9, column = 10)
        self.maxDispLabel = tk.Label(self,
                                     text = "Max Possible Disparity", font = LabelFont).grid(row = 7, column = 10)
        self.focalLabel = tk.Label(self,
                                   text = "Focal Length of Your Camera", font = LabelFont).grid(row = 11, column = 10)
        self.uniqueRatioLabel = tk.Label(self,
                                         text = "How Unique a Pixel Can Be To Be Matched in %", font = LabelFont).grid(row = 5, column = 10)

    def imageSelector(self):
        self.imagePathStr = []
        imagePath = filedialog.askopenfilename()
        while len(imagePath) > 0 or len(self.imagePathStr) < 2:
            self.imagePathStr.append(imagePath)
            imagePath = filedialog.askopenfilename()
        length = (len(self.imagePathStr) - 1)
        print((self.imagePathStr))
        try:
            while length >= 0:
                imageCheck = Image.open((self.imagePathStr[length]))
                length -= 1
            pathL = Image.open((self.imagePathStr[0]))
            pathR = Image.open(self.imagePathStr[(len(self.imagePathStr)-1)])

            NewPathR = pathR.resize((650,650))
            NewPathL = pathL.resize((650,650))

            self.RCanvas.image = ImageTk.PhotoImage(NewPathR)
            self.RCanvas.create_image(0,0, image = self.RCanvas.image, anchor = "nw")

            self.LCanvas.image = ImageTk.PhotoImage(NewPathL)
            self.LCanvas.create_image(0,0, image = self.LCanvas.image, anchor = "nw")

            self.RenderImageBtn.configure(state = "active", bg = "light green")
        except:
            errorPopup = tk.Toplevel()

            errorMessage = tk.Label(master=errorPopup,
                                    text= """Error: One Of the file directorys given
was not an image please try again""").grid(row=1, column=1, columnspan=2)
            cancelButton = tk.Button(master= errorPopup,
                                     text="Ok",
                                     command = lambda: errorPopup.destroy()).grid(row=2, column=2)


    def videoSelector(self):
        self.video = filedialog.askopenfilename()
        if len(self.video) > 0:
            try:
                self.imagePathStr = []
                self.createframes(self.video)

                pathL = Image.open((self.imagePathStr[0]))
                pathR = Image.open(self.imagePathStr[(len(self.imagePathStr)-1)])

                NewPathR = pathR.resize((650,650))
                NewPathL = pathL.resize((650,650))

                self.RCanvas.image = ImageTk.PhotoImage(NewPathR)
                self.RCanvas.create_image(0,0, image = self.RCanvas.image, anchor = "nw")

                self.LCanvas.image = ImageTk.PhotoImage(NewPathL)
                self.LCanvas.create_image(0,0, image = self.LCanvas.image, anchor = "nw")

                self.RenderImageBtn.configure(state = "active", bg = "light green")
                print (self.imagePathStr)
            except:
                VideoerrorPopup = tk.Toplevel()

                errorMessage = tk.Label(master = VideoerrorPopup,
                                        text= """Error: The file directory given
    was not a video please try again""").grid(row=1, column=1, columnspan=2)
                cancelButton = tk.Button(master = VideoerrorPopup,
                                         text="Ok",
                                         command = lambda: VideoerrorPopup.destroy()).grid(row=2, column=2)
```

```
184
185     def createframes(self, video):
186         Video = cv2.VideoCapture(self.video)
187         maxFrames = int(Video.get(cv2.CAP_PROP_FRAME_COUNT))
188         success, image = Video.read()
189         count = 0
190
191         while success:
192             image = cv2.resize(image,(650,650))
193             cv2.imwrite("images/frame%d.jpg" % count, image)
194             self.imagePathStr.append("images/frame%d.jpg" % count)
195             count += 1
196             success, image = Video.read()
197
198
199     def changeFileType(self):
200         if self.ImageFile == True:
201             self.ImageFile = False
202             self.SelectImageBtn.config(command = self.videoSelector,
203                                        text = "Select Your Video")
204         else:
205             self.ImageFile = True
206             self.SelectImageBtn.config(command = self.imageSelector,
207                                        text = "Select Your Images")
208
209     def OptionChange(self, event):
210         if self.Fileclicked.get() == "Change File Type":self.Fileclicked.set("File"), self.changeFileType()
211
212 class quitWindow():
213
214     def __init__(self):
215
216         self.quit = tk.Toplevel()
217
218         Message = tk.Label(master=self.quit,
219                            text= """Are you sure that you want to quit,
220 all unsaved data will be lost?""").grid(row=1, column=1, columnspan=2)
221         quitButton = tk.Button(master= self.quit ,
222                            text="Yes",
223                            command = self.quitALL).grid(row=2, column=1)
224         cancelButton = tk.Button(master= self.quit,
225                            text="No",
226                             command = lambda: self.quit.destroy()).grid(row=2, column=2)
227
228     def quitALL(self):
229         app.destroy()
230
231 class minmax_frame():
232         def __init__(self):
233             global maxScreen
234             if maxScreen == True:
235                 app.attributes("-fullscreen", False)
236                 maxScreen = False
237             elif maxScreen == False:
238                 maxScreen = True
239                 app.attributes("-fullscreen", True)
240 app = windows()
241 app.attributes("-fullscreen", True)
242 app.mainloop()
```

Phase 2 Review:

What has been done?

In this section, I used the design I created inside of my design phase and created the software code that populated the main window page. In this page I added text, canvases, buttons, option menus, and sliders as per my design, but not all these have been linked to any functions yet. The buttons for selection images and videos are completed, however the rest of the program needs to be completed to finish off my proposed design.

How has it been tested?

Every time that I have introduced a new part to the main window, I have executed the program to see if it functioned correctly, and if it was in the correct placement. Every function which linked directly to my success criteria was tested and the results shown after each test was performed. I liken this to a continuous testing activity that I perform when any changes / updates are made to my software code.

How it meets the success criteria and user expectations

The main window has a straightforward layout and allows the users to select both images and videos. This process is the *start* of my project journey to meet the following design criteria:

- Clear front end.
    - The main user interface should provide the following features:
        - Select Images, Select Video, Render Image, close program and minimize / maximize program buttons.
        - Calibration options: minimum possible disparity, maximum possible disparity, uniqueness ratio and focal length.
        - Display the images/frames the user has chosen to enter.
        - Menu options to save and open changes in the program's calibration, save and open renderings and log out.
- A non-technical calibration.
    - The calibration needs to be appropriate for new users.
        - Each option needs to be clear on the main page.
        - They all need text which describe their function.
            - This text needs to be clear.
        - Only a few options required in order to eliminate the chance of confusion.
- The program should be able to take images as input data.
- The program should be able to take a video as input data.
    - The program should be able to segment this video into its required frames.

Changes in the design that have resulted from this section.

This section has followed the design so far.

Summary of the whole project at this phase

So far, I have completed two phases: the rendering algorithm and the main window. Each page has been completed using their own files, neither of them have the ability to be integrated completely together as of yet, however in the next phase of my project, I will do just that.

Phase 3, Joining phases 1 & 2.

In phase 1, I created the class for the rendering of two still images. In phase 2, I created the user interface that will allow the user to select two or more still images or allow them to select a video to be segmented and rendered. In phase 3, I will add / append the main file that the main UI was created in and add the rendering algorithm into it.

```
407     def RenderImage(self):
408         images = self.imagePathStr
409         mindisp = self.minDispSlider.get()
410         maxValue = self.maxDispSlider.get()
411         focal_length = self.focalSlider.get()
412         uniquenessRatio = self.uniqueRatioSlider.get()
413         RenderedimageData(images, focal_length, maxValue, mindisp, uniquenessRatio)
414 
415 #Rendering method-------------------------------------------------------------------------
416 class RenderedimageData():
417     def __init__(self, images, f, maxValue, mindisp, uniquenessRatio):
```

Inside the class mainwindow, I will create a new function RenderImage, inside this function is where I assign both the values for what will be passed through RenderedimageData and call the class. I have assigned the value for images, this value is the imagePathStr, which will contain all the paths to the images that will be rendered, then all the slider values will be assigned to their corresponding variable name (e.g., focal_length = self.focalSlider.get()). The class RenderedimageData is called with these values passed through to render the image.

```
226     self.RenderImageBtn = tk.Button(self,
227                                     text="Render Your Image", command = self.RenderImage, state = "disabled", font = ButtonFont, height = 3, width = 30 )
228     self.RenderImageBtn.grid(row = 13, column = 6 )
```

I next assigned the function RenderImage to be called as the command to the render image button. To see if it works, I will pass through two pictures to make sure it is called and functions correctly:


```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\alexf\AppData\Local\Programs\Python\Python38-32\lib\tkinter\__i
nit__.py", line 1883, in __call__
    return self.func(*args)
  File "C:\Users\alexf\OneDrive\Desktop\Main GUI.py", line 392, in RenderImage
    RenderedimageData(images, focal_length, 100, maxValue, mindisp, uniquenessRa
tio)
  File "C:\Users\alexf\OneDrive\Desktop\Main GUI.py", line 404, in __init__
    disp = self.createStereoImage(left_image, right_image, maxValue, mindisp, un
iquenessRatio)
  File "C:\Users\alexf\OneDrive\Desktop\Main GUI.py", line 443, in createStereoI
mage
    return stereo.compute(left_image, right_image).astype(np.float32) / 16.0
NameError: name 'np' is not defined
>>>
```

The reason for this error. I forgot to import numpy into the main file, the result after I amended this error was:

```
1  import tkinter as tk
2  import tkinter.font as font
3  from tkinter import filedialog
4  from PIL import ImageTk, Image
5  import numpy as np
6  import cv2
7  import csv
8  import time
9
```

Now that the function is shown to be passed through correctly, I will now append the main file to allow the creation of models with more than two images. To do this, I will utilise both the method for creating the xyz and rgb values discussed in phase one, however I will now use another method in unison to allow the creation and combination of multiple point clouds.

```
1 import tkinter as tk
2 import tkinter.font as font
3 from tkinter import filedialog
4 from PIL import ImageTk, Image
5 import numpy as np
6 import cv2
7 import matplotlib.pyplot as plt
8 from mpl_toolkits import mplot3d
9 import csv
```

I have imported two new needed libraries; matplotlib: for plotting the model, and mplot3d: as an extension to matplotlib to allow it to handle 3D models.

```
416 class RenderedimageData():
417     def __init__(self, images, f, maxValue, mindisp, uniquenessRatio):
418         homogenized = np.array([[1,0,0],
419                     [0,1,0],
420                     [0,0,-f]])
421 
422         distortion = np.array([[0.],
423                        [0.],
424                        [0.],
425                        [0.]])
426         if len(images) > 2:
427             totalImages = (len(images) - 1)
428             currentLeftImage = 0
429             while totalImages > currentLeftImage:
430                 pathL = images[currentLeftImage]
431                 pathR = images[(currentLeftImage + 1)]
432                 left_image = cv2.imread(pathL)
433                 right_image = cv2.imread(pathR)
434         #Image~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
435                 height, width, _ = left_image.shape
436                 RGB = right_image.reshape(-1,3)
437         #Matrices~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
438                 camreaMatrix = np.array([[f,0,width/2],
439                                          [0,f,height/2],
440                                          [0,0,1]])
441                 transformation = np.float32([[1,0,0,-width/2],
442                                              [0,1,0,-height/2],
443                                              [0,0,0,-f],
444                                              [0,0,-1/100,0]])
445 
446                 camreaMatrix = self.OptimalMatrix(camreaMatrix, distortion, right_image)
447 
448                 rVector, tVector = self.rt_Vectors(homogenized, camreaMatrix)
```

I then appended the class RenderedimageData and created an if statement, which queries if the string of images contains more than two images, if it does it then creates a new variable totalImages which is the total amount of images minus one (as we start the counting of a list/string at zero) and then creates the variable currentLeftImage, which is set to zero (as that's the first image we start with).

Then a while loop is created, it executes the code while the total images is larger than the current left image which allows each image to have a pair. While the total amount of images is larger than the current left image the program loops through each image.

```
449
450                 stereo = self.depthMap(left_image, right_image)
451
452                 points = self.pointsTo3D(stereo, transformation)
453
454                 deleteNoise = ((stereo.reshape(-1)) > (stereo.reshape(-1)).min())
455                 points = points[deleteNoise]
456                 RGB = RGB[deleteNoise]
457
458                 c = [ [i] for i in points]
459                 d = [ [i] for i in RGB]
460                 a = 0
461                 largeArray = []
462                 for i in points:
463                     pointsArray = np.array(c[a])
464
465                     coloursArray = np.array(d[a])
466
467                     mainArray = np.hstack((pointsArray, coloursArray))
468                     if a == 0:
469                         largeArray = mainArray
470                     else:
471                         largeArray = np.vstack((largeArray, mainArray))
472                     a +=1
```

Next to create the coordinates for the point cloud I called depthMap to create a stereo image of the first two images in the image list. Then using the stereo image, I called points to get the xyz values (points) and rgb values (colours).

When creating a point cloud, you need a large array of different arrays containing the x, y, z ,r , g, b values for each pixel([[x, y, z, r, g, b], [x, y, z, r, g, b]…]), with the xyz values being matched with their corresponding RGB values, however at the moment points and colours are two different array of arrays. OpenCV usually would handle this, however because OpenCV can't handle the combination of multiple point clouds on its own, I needed to combine them manually.

To do this I first assigned the variable "c" and "d" to each individual array in points (c) and colours (d), then I created a new array called largeArray which will contain the final combined x, y, z, r, g, and b values. For each array in points, take the two arrays with matching positions in points and colours (position 0 and 0, 1 and 1 ….) and using the NumPy command hstack combine the two arrays onto the same row. The last step in creating the full array is combing each newly created arrays together, to do this this I first checked if the array that was created and was the first one created, as you cannot stack an empty array with a populated one. If it is, I set largeArray to equal the newly created one, however if it is not, I set the large array to equal itself stacked vertically with the new array this then takes points and colours which look like: [[x,y,z],[x,y,z],…] and [[r,g,b],[r,g,b],…], and turn them into:

[[x, y, z, r, g, b],
[x, y, z, r, g, b],
….]

```
473
474             points = largeArray[abs(largeArray[:,2]-(np.mean(largeArray,axis=0)[2]))<2]
475
476             axis, RGB = points[:,:3], points[:,3:]
477             project = plt.axes(projection='3d')
478
479             project.scatter(axis[:,0], axis[:,1], axis[:,2], c = RGB/255, s = 0.01)
480             plt.show()
```

To render this data, I made points be equal to largeArray where the absolute value of the z values in the array minus the mean of the height of all the z values are smaller than 2, this is used to help remove noise and outlying values.

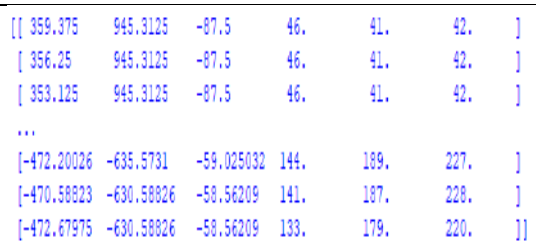
Then I set the variable axis to be equal to the first three values inside of all the arrays (x y and z) and colours to be equal to the last three values inside of points (r g and b). I then created the 3-dimensional graph utilising matplotlib to project the points onto. I then projected the points onto the graph using matplotlibs scatter function and plotted the graph.

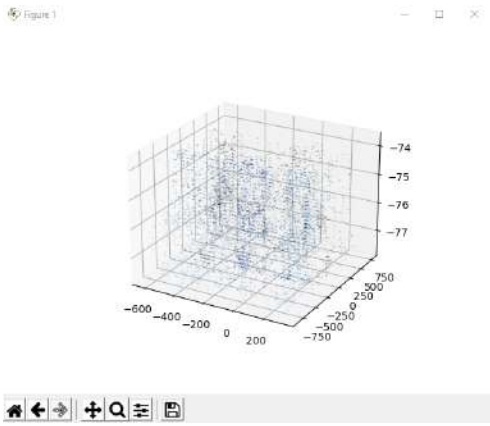
However, if the image list only contained two images then a different method is used.

```
482         else:
483             pathL = images[0]
484             pathR = images[1]
485             left_image = cv2.imread(pathL)
486             right_image = cv2.imread(pathR)
487             height, width, _ = left_image.shape
488             RGB = right_image.reshape(-1,3)
489 
490             camreaMatrix = np.array([[f,0,width/2],
491                                      [0,f,height/2],
492                                      [0,0,1]])
493             transformation = np.float32([[1,0,0,-width/2],
494                             [0,1,0,-height/2],
495                             [0,0,0,-f],
496                             [0,0,-1/100,0]])
497 
498             camreaMatrix = self.OptimalMatrix(camreaMatrix, distortion, right_image)
499 
500             rVector, tVector = self.rt_Vectors(homogenized, camreaMatrix)
501 
502             stereo = self.depthMap(left_image, right_image, maxValue, mindisp, uniquenessRatio)
503 
504             points = self.pointsTo3D(stereo, transformation)
505 
506             deleteNoise = ((stereo.reshape(-1)) > (stereo.reshape(-1)).min())
507             points = points[deleteNoise]
508             RGB = RGB[deleteNoise]
509 
510             projected = self.projectPoints(points, rVector, tVector, camreaMatrix, distortion)
511 
512             xPoints, yPoints = projected[:, 0], projected[:, 1]
513 
514             projected, RGB = self.filterNoise(projected, RGB, xPoints, yPoints, right_image)
515 
516             self.createModel(height, width, projected, RGB)
```

The method that is being used here is the same process that happened in the Phase 1. The reason I have had to use two different methods is due to how both these different functions create a point cloud. OpenCV uses something called a block matcher, which when the system identifies two pixels to match it takes the block of colours around it and layers that over it, which creates an output of an image which looks more realistic with less data points needed, however when adding more and more images, the blocks of colours overlap and become very noisy and unrealistic and so I could not use it for multiple image rendering. Matplotlib on the other hand (using its 3D function) is the complete opposite. It needs a lot of input data as it plots each pixel with its individual colour not a block of colours, this means small data sets have too little data to create an adequate rendering, but large data sets create very realistic renderings and so each are utilised to accommodate their strengths and negate their weaknesses.

Testing:

What I am testing	Data Entered	Data Type (erroneous, boundary, normal)	Result	Was this expected
Creating a model with 2 images.		boundary		Yes
Creating the correct xyzrgb array				Yes

Creating a model from 3 or more images.	  	normal		No, I assumed incorrectly that I would have received a better rendering however the reason I didn't was because I didn't give it enough data points.

Here is the rending class' final code:

```
#Rendering method-----------------------------------------------------------------------------------------------
class RenderedimageData():
    def __init__(self, images, f, maxValue, mindisp, uniquenessRatio):
        homogenized = np.array([[1,0,0],
                    [0,1,0],
                    [0,0,-f]])

        distortion = np.array([[0.],
                      [0.],
                      [0.],
                      [0.]])
        if len(images) > 2:
            totalImages = (len(images) - 1)
            currentLeftImage = 0
            while totalImages > currentLeftImage:
                pathL = images[currentLeftImage]
                pathR = images[(currentLeftImage + 1)]
                left_image = cv2.imread(pathL)
                right_image = cv2.imread(pathR)
        #Image~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                height, width, _ = left_image.shape
                RGB = right_image.reshape(-1,3)
        #Matrices~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                camreaMatrix = np.array([[f,0,width/2],
                                [0,f,height/2],
                                [0,0,1]])
                transformation = np.float32([[1,0,0,-width/2],
                                    [0,1,0,-height/2],
                                    [0,0,0,-f],
                                    [0,0,-1/100,0]])

                camreaMatrix = self.OptimalMatrix(camreaMatrix, distortion, right_image)

                rVector, tVector = self.rt_Vectors(homogenized, camreaMatrix)

                stereo = self.depthMap(left_image, right_image)

                points = self.pointsTo3D(stereo, transformation)

                deleteNoise = ((stereo.reshape(-1)) > (stereo.reshape(-1)).min())
                points = points[deleteNoise]
                RGB = RGB[deleteNoise]

                c = [ [i] for i in points]
                d = [ [i] for i in RGB]
                a = 0
                largeArray = []
                for i in points:
                    pointsArray = np.array(c[a])

                    coloursArray = np.array(d[a])

                    mainArray = np.hstack((pointsArray, coloursArray))
                    if a == 0:
                        largeArray = mainArray
                    else:
                        largeArray = np.vstack((largeArray, mainArray))
                    a +=1

            points = largeArray[abs(largeArray[:,2]-(np.mean(largeArray,axis=0)[2]))<2]
```

```
            axis, RGB = points[:,:3], points[:,3:]
            project = plt.axes(projection='3d')

            project.scatter(axis[:,0], axis[:,1], axis[:,2], c = RGB/255, s = 0.01)
            plt.show()

        else:
            pathL = images[0]
            pathR = images[1]
            left_image = cv2.imread(pathL)
            right_image = cv2.imread(pathR)
            height, width, _ = left_image.shape
            RGB = right_image.reshape(-1,3)

            camreaMatrix = np.array([[f,0,width/2],
                                     [0,f,height/2],
                                     [0,0,1]])
            transformation = np.float32([[1,0,0,-width/2],
                            [0,1,0,-height/2],
                            [0,0,0,-f],
                            [0,0,-1/100,0]])

            camreaMatrix = self.OptimalMatrix(camreaMatrix, distortion, right_image)

            rVector, tVector = self.rt_Vectors(homogenized, camreaMatrix)

            stereo = self.depthMap(left_image, right_image, maxValue, mindisp, uniquenessRatio)

            points = self.pointsTo3D(stereo, transformation)

            deleteNoise = ((stereo.reshape(-1)) > (stereo.reshape(-1)).min())
            points = points[deleteNoise]
            RGB = RGB[deleteNoise]

            projected = self.projectPoints(points, rVector, tVector, camreaMatrix, distortion)

            xPoints, yPoints = projected[:, 0], projected[:, 1]

            projected, RGB = self.filterNoise(projected, RGB, xPoints, yPoints, right_image)

            self.createModel(height, width, projected, RGB)

    def OptimalMatrix(self, camreaMatrix, distortion, right_image):
        camreaMatrix, _ = cv2.getOptimalNewCameraMatrix(camreaMatrix, distortion, (right_image.shape[:2]), 1, (right_image.shape[:2]))
        return camreaMatrix

    def rt_Vectors(self, homogenized, camreaMatrix):
        _, RotationMatrix1, _, transitionVectors = cv2.decomposeHomographyMat(homogenized, camreaMatrix)
        return RotationMatrix1[3], transitionVectors[1]

    def depthMap(self, left_image, right_image, maxValue, mindisp, uniquenessRatio):
        stereo = cv2.StereoSGBM_create(minDisparity = mindisp,
                                       numDisparities = maxValue - mindisp,
                                       blockSize = 7  ,
                                       uniquenessRatio = uniquenessRatio,
                                       speckleWindowSize = 200,
                                       speckleRange = 2,
                                       disp12MaxDiff = 10,
                                       P1 = 8*3*1**2,
                                       P2 =32*3*1**2)
        return stereo.compute(left_image, right_image)

    def pointsTo3D(self, stereo, transformation):
        points = cv2.reprojectImageTo3D(stereo, transformation)
        points = points.reshape(-1,3)
        return points

    def projectPoints(self, points, rVector, tVector, camreaMatrix, distortion):
        projected, _ = cv2.projectPoints(points, rVector, tVector, camreaMatrix, distortion)
        projected = projected.reshape(-1, 2)
        return projected.astype(int)

    def filterNoise(self, projected, RGB, xPoints, yPoints, right_image):
        height, width = right_image.shape[:2]
        deleteNoise = ((0 <= xPoints & yPoints)&(xPoints < width)&(yPoints < height))
        return projected[deleteNoise], RGB[deleteNoise]

    def createModel(self, height, width, projected, RGB):
        model = np.zeros((height, width, 3), np.uint8)
        model[projected[:, 1],projected[:, 0]] = RGB
        global renderedimage
        renderedimage = model
        cv2.imshow("Rendered Data", model)
```

Phase 3 review

What has been done?

Inside of the main python file, I have appended the class for the rendering of images, I have then amended this class to be able to accommodate the rendering of multiple images and have implemented a new rendering technique to help assist OpenCV in the areas it was sub-optimal, namely in rendering more than two multiple images. I have also enhanced the code I developed during Phase 1 building on the requirements of my design as well as feedback on the testing that I have completed.

How has it been tested?

Each time I added a new section to the rendering algorithm, I executed the program to see if it worked and see there were any errors generated. The final rendering algorithm was also tested with normal and boundary data (as laid out in the testing table). For each test I provided what I was testing, the data that I entered, the result and if that result was expected.

How it meets the success criteria and user expectations

It has met the following criteria:

- Render models.
    - The program should be able to render an image from two still photos.
        - The program should be able to take images as input data.
    - The program should be able to render an image from more than two images.
    - The program should be able to render an image from a video.
        - The program should be able to take a video as input data.
            - The program should be able to segment this video into its frames.
    - From the input data the program should be able to create:
        - A stereo image that also is influenced by the calibrations.
        - The x y z r g b values of each pixel.
        - A point cloud made from these values.
- A non-technical calibration.
    - The user should be able to calibrate:
        - minimum possible disparity.
        - maximum possible disparity.
        - uniqueness ratio.
        - focal length.

Changes in the design that have resulted from this section.

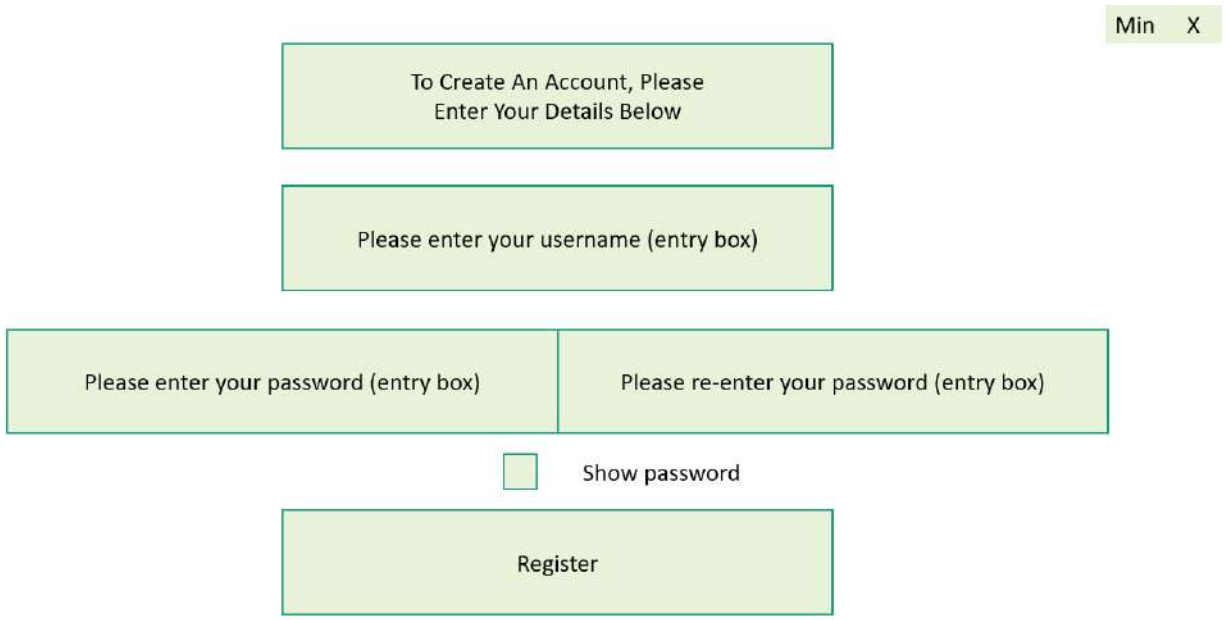
This section has slightly deviated from the design section, as instead of just using just OpenCV, I have also utilised matplotlib. Other than this change, the functionality of the program has stayed the exact same.

Summary of the whole project at this phase

The rendering class has now been amended and placed into the main file; the program is now able to open the main window and the user can now interact with the UI to select two or more images / a video, and the user can then select to render the images or the video. My testing showed a limitation that the point clouds of three images have empty space and have little to no similarity to the images.

Phase 4, Registration User Interface

This is design I created in the design section:



Inside of the main file I will be using Tkinter (a built-in python function) to create the user interface for the registration window. However, before I code all the widgets to create the UI, I will first need to append the already created class windows:

```
class windows(tk.Tk):

    def __init__(self):
        tk.Tk.__init__(self)
        container = tk.Frame(self)

        container.pack(side = "top", fill = "both", expand = False )
        container.grid_rowconfigure(0, weight = 1)
        container.grid_columnconfigure(0, weight = 1)

        self.farmes = {}

        for i in signinwindow, mainwindow:
            frame = i(container, self)

            self.farmes[i] = frame

            frame.grid(row = 15, column = 15, sticky = "nsew")

        self.show_frame(signinwindow)

```

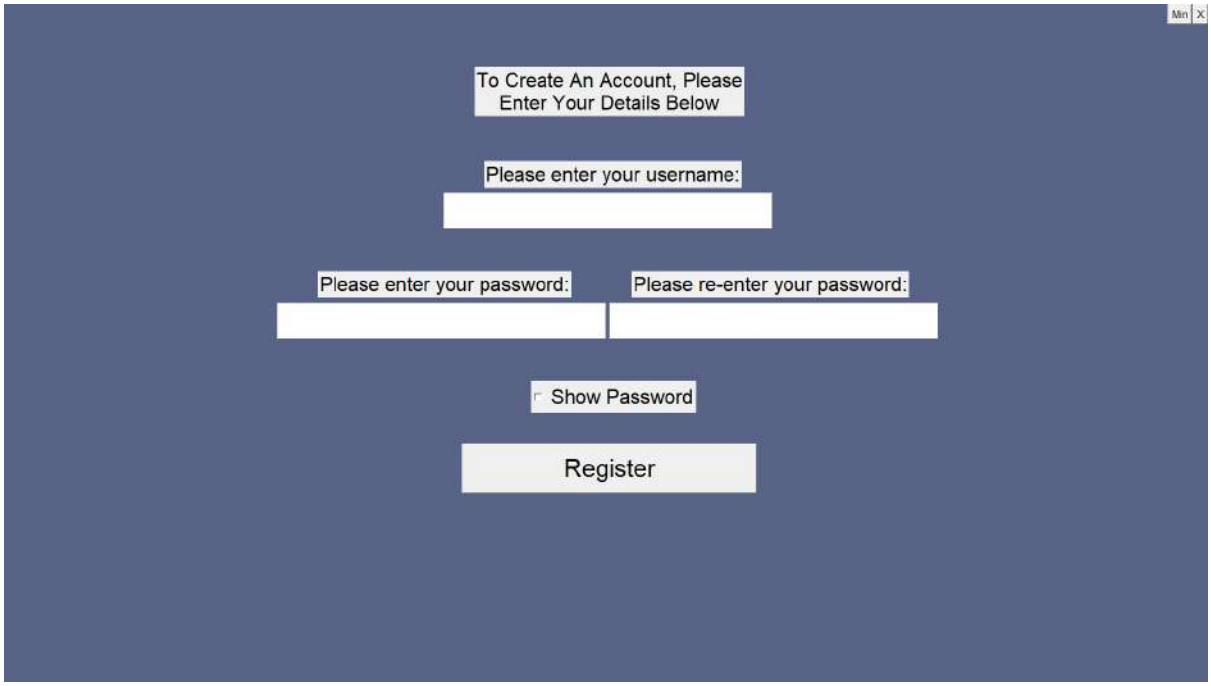
Inside of the class windows, I set up a for loop. What this loop does is for every frame I define; it creates the required attributes and assigns it to that frame. This is essential, as without this there is no way of showing all the frames. Next, I need to create a class named “sigininwindow” and create the UI inside of its initialization.

```
class signinwindow(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent, bg = "#596387")
        ButtonFont = font.Font(size = 30, family="Helvetica")
        LabelFont = font.Font(size = 24, family = "Helvetica")
        EntryFont = font.Font(size = 35, family = "Helvetica")
        #Buttons-----------------------------------
        self.minmaxbutton = tk.Button(self,
                                      command = minmax_frame, text = "Min", font = (font.Font(size = 12, family="Helvetica")))
        self.minmaxbutton.place(x = 1855, y = 0)

        self.exitbutton = tk.Button(self,
                                    command = quitWindow, text = "X",  font = (font.Font(size = 12, family="Helvetica"))).place(x =1895 , y = 0)
        self.signupbutton = tk.Button(self,
                                      text = "Register", font = ButtonFont, width = 20)
        self.signupbutton.place(x =730 , y =700 )
        #Entry Boxs-------------------------------------------------
        self.usernameBox = tk.Entry(self,
                                    font = EntryFont)
        self.usernameBox.place(x = 700, y = 300)
        self.passwordBox = tk.Entry(self,
                                    show = "*", font = EntryFont)
        self.passwordBox.place(x = 965, y = 475)
        self.checkpasswordBox = tk.Entry(self,
                                    show = "*", font = EntryFont)
        self.checkpasswordBox.place(x = 435, y = 475)
        #Labels-----------------------------------------------------
        infomationLabel = tk.Label(self,
                                   text = """To Create An Account, Please
Enter Your Details Below"", font = LabelFont).place(x = 780, y = 100)
        usernameLabel = tk.Label(self,
                                 text = "Please enter your username:" , font = LabelFont).place(x = 765, y = 250)
        passwordLabel = tk.Label(self,
                                 text = "Please enter your password:" , font = LabelFont).place(x = 500, y = 425)
        repasswordLabel = tk.Label(self,
                                 text = "Please re-enter your password:" , font = LabelFont).place(x = 1000, y = 425)
        #CheckBox---------------------------------------------------
        self.showpassBox = tk.Checkbutton(self,
                                          text = "Show Password", font = LabelFont)
        self.showpassBox.place(x = 840, y = 600)

```

The code above produces this window below:



As you can see, I created a class for the registration window called sigininwindow, and in the initialization of this class the layout is created. Each item in this window has at least two lines dedicated to it; one is used to define what it is (for example self.signupbutton = tk.Button(self, text = "Register", font = ButtonFont, width = 20) and the second states where it will be placed in the window (for example self.signupbutton.place(x =730 , y =700))). For this window, I have used tkinters .place system which means each item is given a x and y co-ordinate which dictates where it is on the page. The reason I used .place instead of the grid system, I used within the main window, is because .place offers a more exact way of placing each component, this made it easier for me to create the design.

I next assigned commands to these buttons:

```
46        self.minmaxbutton = tk.Button(self,
47                                      command = minmax_frame, text = "Min", font = (font.Font(size = 12, family="Helvetica")))
48        self.minmaxbutton.place(x = 1855, y = 0)
49
50        self.exitbutton = tk.Button(self,
51                                    command = quitWindow, text = "X",  font = (font.Font(size = 12, family="Helvetica"))).place(x =1895 , y = 0)
```

The two commands that have been assigned to the buttons are minmax_frame (for the minmax button), and quitWindow (for the exit button). These two commands when called run their respective classes. Both of these class' were created and explained further in "Phase 2, Main Window".

The next part I will be addressing is the show password check box. This can be found on lines 76-78 of the program, for this I will need to append the class sigininwindow and create a sub-routine inside of it. This will allow the user to see their password as they type it.

```
39 class signinwindow(tk.Frame):
40     def __init__(self, parent, controller):
41         tk.Frame.__init__(self, parent, bg = "#596387")
42         self.checkVar = tk.IntVar()
```

First I assigned a value to the variable checkVar, the value assigned was tk.IntVar().

```
77        self.showpassBox = tk.Checkbutton(self,
78                                          text = "Show Password",variable = self.checkVar, font = LabelFont)
```

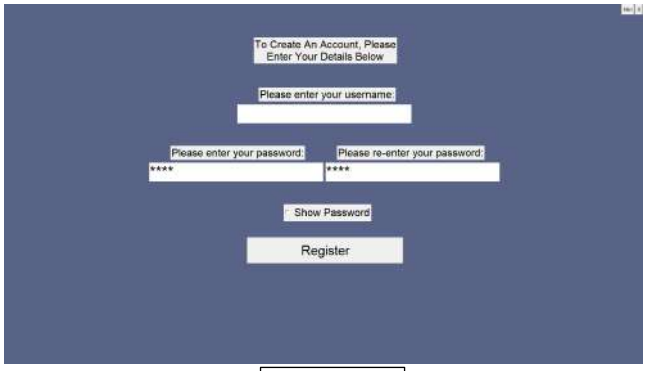
Next, I appended the widget check button to hold the variable self.checkVar, what this means, is when the check button is pressed the variable will change, and in this case because the variable is tk.IntVar() when pressed the check buttons variable will be 1 or 0 depending if the check button is on or off.

```
80
81     def showpassword(self):
82         if self.checkVar.get() == 1:
83             self.passwordBox.config(show = "")
84             self.checkpasswordBox.config(show = "")
85         else:
86             self.passwordBox.config(show = "*")
87             self.checkpasswordBox.config(show = "*")
88
```

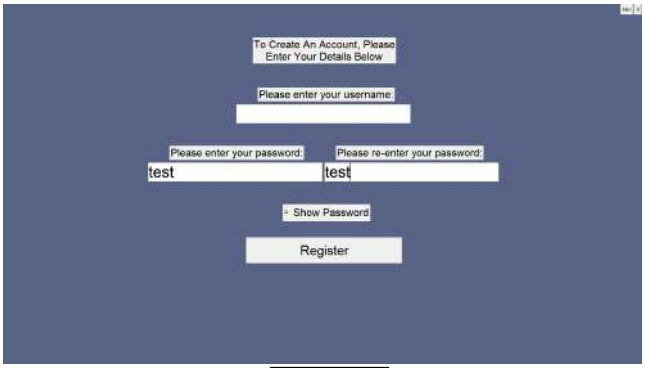
I then created this sub-routine showpassword and appended it to the class sigininwindow. This sub-routine checks the value of checkVar and uses that data to determine if the password should be hidden or not.

```
77        self.showpassBox = tk.Checkbutton(self,
78                                          text = "Show Password",variable = self.checkVar, command = self.showpassword, font = LabelFont)
```

I then append the check button to add the command show password. This should mean whenever the check button is clicked, the subroutine showpassword should be executed. I will need to now test to see if this properly functioning, to do this I will enter the word "test" into both password boxes and then click the show password box. The correct result would be if the two boxes showed the word "test" after it has been clicked; here are the results.



Before



After

The final part of Phase 4 will be creating the function which allows the user to register as a user. For this, I need to append to the sigininwindow class and define a new sub-routine to allow the user to do this.

```
6 import csv
```

First, I needed import in the necessary library, csv. I will have to utilise csv for all the communication between my program and the external databases.

```
def register(self):
    if self.passwordBox.get() == self.checkpasswordBox.get() and len(self.passwordBox.get()) > 0 and len(self.usernameBox.get()) > 0 :
        try:
            with open("UserData.csv", "r") as file:
                fieldnames = ["Username", "Password", "UserKey"]
                reader = csv.DictReader(file, fieldnames = fieldnames)
                for row in reader:
                    if row["Username"] == self.usernameBox.get():
                        arg
                with open("UserData.csv", "a") as file:
                    writer = csv.DictWriter(file, fieldnames = fieldnames)
                    writer.writerow({"Username": "%s" % self.usernameBox.get()
                                     ,"Password": "%s" % self.passwordBox.get()
                                     , "UserKey": "%s" % ("".join(str(ord(l)) for l in self.usernameBox.get())) })
```

Next, I created the subroutine register (located inside of siginwindow). On line 90, I created an if statement, this is to check if the user has; entered two different passwords, has not entered a username, or has not entered a password. If they have met the requirements of the if statement, a new try statement is invoked (line 91-102). Inside of the try statement is where all the external database file dialogs will happen. The start the of the program opens the file UserData as a file to be read (r), then reads each row in column Username in the file to see if the username the user chose is already taken. If the username is not taken, then the sub routine re-opens the file UserData as a file it can append (a) to. It then appends the user data to the file. The UserKey is a unique string of numbers each user gets based on their usernames, this will be used to locate the user saved files and pre-sets later. However, if the username is taken an argument is passed through which will invoke the except statement.

```
103             except:
104                 errorPopup = tk.Toplevel()
105 
106                 errorMessage = tk.Label(master=errorPopup,
107                                         text= """Error: That Username has
108 already been taken""").grid(row=1, column=1, columnspan=2)
109                 cancelButton = tk.Button(master= errorPopup,
110                                         text="Ok",
111                                      command = lambda: errorPopup.destroy()).grid(row=2, column=2)
112         else:
113             errorPopup = tk.Toplevel()
114 
115             errorMessage = tk.Label(master=errorPopup,
116                                     text= """Error: Please check you have both a
117 username and matching passwords""").grid(row=1, column=1, columnspan=2)
118             cancelButton = tk.Button(master= errorPopup,
119                                     text="Ok",
120                                  command = lambda: errorPopup.destroy()).grid(row=2, column=2)
121 
```

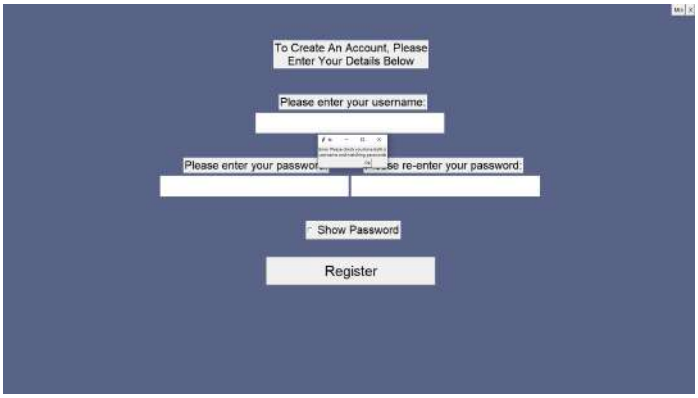
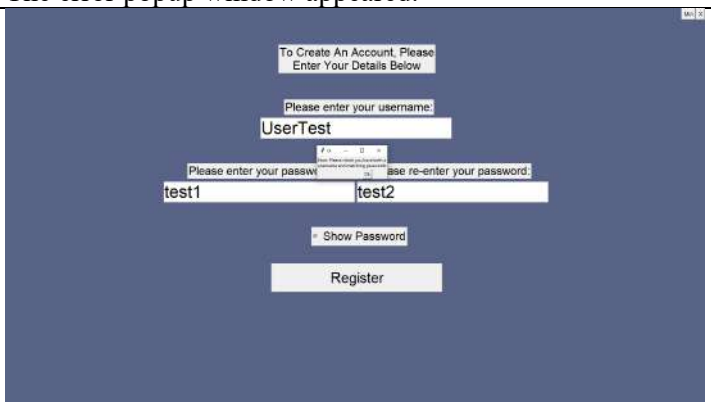
The except statement when invoked will create an error window, another error window will be created if the if the if statement requirements are not met. I discussed in length the creation of these error type windows previously, so I will not go into any detail now.

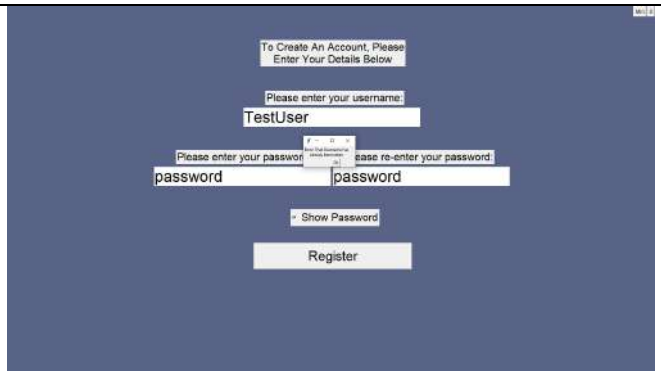
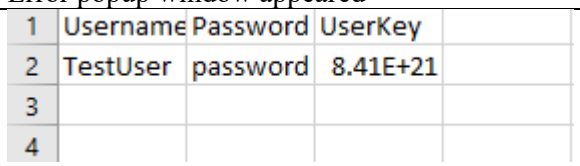
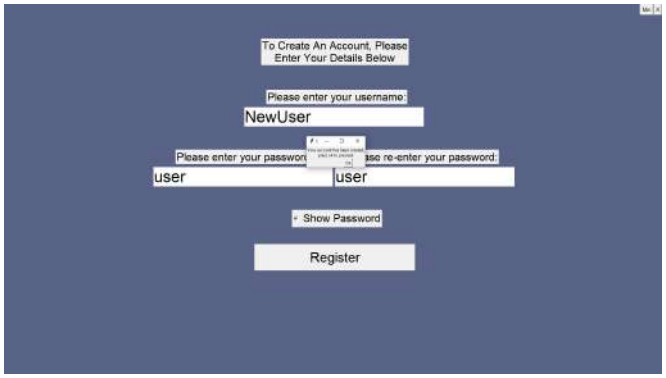
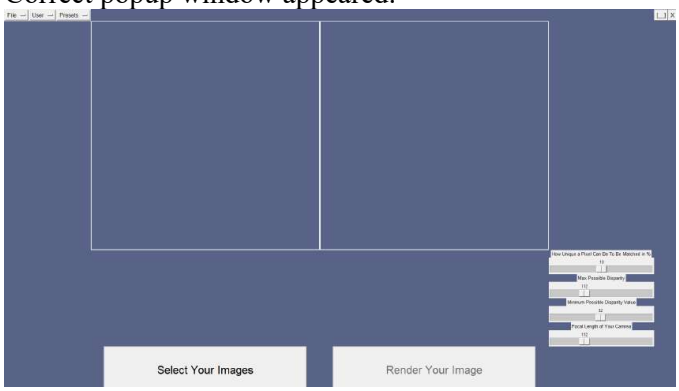
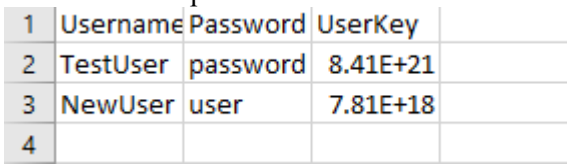
```
43          self.nextframe = lambda: controller.show_frame(mainwindow)
104             Popup = tk.Toplevel()
105
106             Message = tk.Label(master=Popup,
107                                text= """Your account has been created,
108 press ok to proceed""").grid(row=1, column=1, columnspan=2)
109             Button = tk.Button(master= Popup,
110                                text="Ok",
111                                command =self.nextframe).grid(row=2, column=2)
112
```

These final additions are what transfers the user from the signup window to the main window. I originally wanted to have the user transferred to the main window as soon as they created a user, however no matter how many different methods I tried, I could not get it to work satisfactorily. This compromise however won't affect the use or functionality of my software. Now to make sure the system functions properly and as designed; I will need to test it.

Testing:

Here is how I will be testing the system:

What I am testing	Data Entered	Data Type (erroneous, boundary, normal)	Result	Was this expected
1. Registering with no data	null	erroneous	 The error popup window appeared.	Yes
2. Registering with different passwords	"UserTest" "test1", "test2"	erroneous	 The error popup window appeared.	Yes

3. Registering with a taken username	"TestUser" "password" "password"	erroneous	 Error popup window appeared	Yes
4. Registering with correct data.	"NewUser", "user" and "user"	normal	 Database before test.  Correct popup window appeared.  Main window opened.  Database after test.	Yes

Phase 4 code listing:

File Edit Format Run Options Window Help

```
import tkinter as tk
import tkinter.font as font
from tkinter import filedialog
from PIL import ImageTk, Image
import cv2
import csv


class windows(tk.Tk):

    def __init__(self):
        tk.Tk.__init__(self)
        container = tk.Frame(self)

        container.pack(side = "top", fill = "both", expand = False )
        container.grid_rowconfigure(0, weight = 1)
        container.grid_columnconfigure(0, weight = 1)

        self.farmes = {}

        for i in signinwindow, mainwindow:
            frame = i(container, self)

            self.farmes[i] = frame

            frame.grid(row = 15, column = 15, sticky = "nesw")

        self.show_frame(signinwindow)

        global maxScreen
        maxScreen = True


    def show_frame(self,cont):
        frame = self.farmes[cont]
        frame.tkraise()


class signinwindow(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent, bg = "#556387")
        self.checkVar = tk.IntVar()
        self.nextframe = lambda: controller.show_frame(mainwindow)
        ButtonFont = font.Font(size = 30, family="Helvetica")
        LabelFont = font.Font(size = 24, family = "Helvetica")
        EntryFont = font.Font(size = 35, family = "Helvetica")
        #Buttons-----------------------------------
        self.minmaxbutton = tk.Button(self,
                                      command = minmax_frame, text = "Min", font = (font.Font(size = 12, family="Helvetica")))
        self.minmaxbutton.place(x = 1855, y = 0)

        self.exitbutton = tk.Button(self,
                                    command = quitWindow, text = "X",  font = (font.Font(size = 12, family="Helvetica"))).place(x =1895 , y = 0)
        self.signupbutton = tk.Button(self,
                                      text = "Register", font = ButtonFont, width = 20, command = self.register)
        self.signupbutton.place(x =730 , y =700 )
        #Entry Boxs-------------------------------------------------------
        self.usernameBox = tk.Entry(self,
                                    font = EntryFont)
        self.usernameBox.place(x = 700, y = 300)
        self.passwordBox = tk.Entry(self,
                                    show = "*", font = EntryFont)
        self.passwordBox.place(x = 965, y = 475)
        self.checkpasswordBox = tk.Entry(self,
                                         show = "*", font = EntryFont)
        self.checkpasswordBox.place(x = 435, y = 475)
        #Labels-----------------------------------------------------------
        infomationLabel = tk.Label(self,
                                   text = """To Create An Account, Please
Enter Your Details Below""", font = LabelFont).place(x = 750, y = 100)
        usernameLabel = tk.Label(self,
                                 text = "Please enter your username:" , font = LabelFont).place(x = 765, y = 250)
        passwordLabel = tk.Label(self,
                                 text = "Please enter your password:" , font = LabelFont).place(x = 500, y = 425)
        repasswordLabel = tk.Label(self,
                                   text = "Please re-enter your password:" , font = LabelFont).place(x = 1000, y = 425)
        #CheckBox---------------------------------------------------------
        self.showpassBox = tk.Checkbutton(self,
                                          text = "Show Password",variable = self.checkVar, command = self.showpassword, font = LabelFont)
        self.showpassBox.place(x = 840, y = 600)

    def showpassword(self):
        if self.checkVar.get() == 1:
            self.passwordBox.config(show = "")
            self.checkpasswordBox.config(show = "")
        else:
            self.passwordBox.config(show = "*")
            self.checkpasswordBox.config(show = "*")

    def register(self):
        if self.passwordBox.get() == self.checkpasswordBox.get() and len(self.passwordBox.get()) > 0 and len(self.usernameBox.get()) :
            try:
                with open("UserDate.csv", "r") as file:
                    fieldnames = ["Username", "Password", "UserKey"]
                    reader = csv.DictReader(file, fieldnames = fieldnames)
                    for row in reader:
                        if row["Username"] == self.usernameBox.get():
                            arg
                    with open("UserData.csv", "a", newline = "") as file:
                        writer = csv.DictWriter(file, fieldnames = fieldnames)
                        writer.writerow({"Username": "%s" % self.usernameBox.get()
                                         ,"Password": "%s" % self.passwordBox.get()
                                         , "UserKey": "%s" % (''.join(str(ord(i)) for i in self.usernameBox.get())) })
                Popup = tk.Toplevel()

                Message = tk.Label(master=Popup,
                                   text= """Your account has been created,
press ok to proceed""").grid(row=1, column=1, columnspan=2)
                Button = tk.Button(master= Popup,
                                   text="Ok",
                                   command =self.nextframe).grid(row=2, column=2)


            except:
                errorPopup = tk.Toplevel()

                errorMessage = tk.Label(master=errorPopup,
                                        text= """Error: That Username has
already been taken""").grid(row=1, column=1, columnspan=2)
                cancelButton = tk.Button(master= errorPopup,
                                         text="Ok",
                                         command = lambda: errorPopup.destroy()).grid(row=2, column=2)
        else:
            errorPopup = tk.Toplevel()

            errorMessage = tk.Label(master=errorPopup,
                                    text= """Error: Please check you have both a
username and matching passwords""").grid(row=1, column=1, columnspan=2)
            cancelButton = tk.Button(master= errorPopup,
                                     text="Ok",
                                     command = lambda: errorPopup.destroy()).grid(row=2, column=2)
```

Phase 4 review

What has been done?

Using Tkinter, I appended the main program file to add a user interface for the registration system. The registration interface is now populated with, buttons, text, entry boxes, and a check button as per my design. The main button (for registration) is linked to the function which processes the user data and either allows a user to proceed or make the user enter new details. All data that is being processed is stored in an external database.

How has it been tested?

Each time I have added a new part to the registration interface, I have executed the program to see if it still works and functions correctly and if it the GUI is aligned correctly. The registration functions have also been tested, with both erroneous and normal data (as specified in the testing table). For each test, I provide what I was testing, the data that I entered, the result and if that result was expected.

How it meets the success criteria and user expectations

The registration window has a simple design and allows the users to create an account and access the main window. It has met the following criteria:

- Clear front end
    - The registration window should provide the following features:
        - An entry box for; their username, password, and a check password.
        - A button for; registering their data and showing their password.
- A functioning account system
    - A registration window which allows the user to enter their details to create an account.
        - The details need to be saved to an external database, with a unique user key.

Changes in the design that have resulted from this section

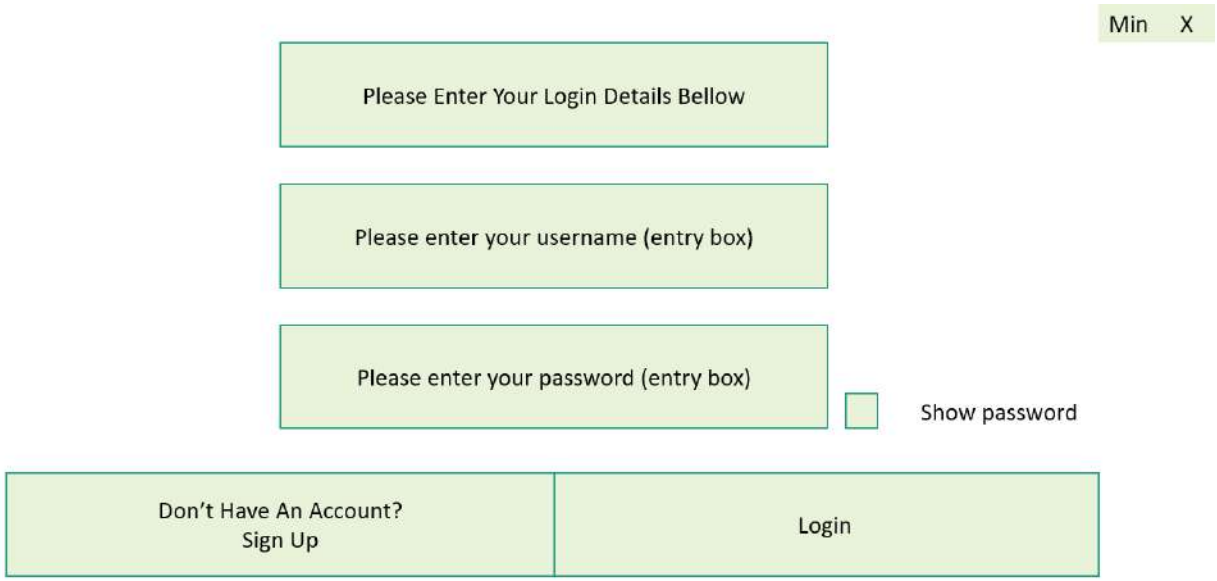
This section has almost followed the design, with the only change being that of an extra pop-up window.

Summary of the whole project at this phase

The main file now contains both the user interfaces for the main and registration window, it also contains the functions for creating a new user and rendering a video or selection of images. The only window left to create is the login window, and then the program will almost be finished.

Phase 5, Login User Interface

This is design I created in the design section:



Inside of the main file, I will be using Tkinter to create the user interface for the login window. However, before I code all the widgets to create the UI, I will first need to append the already created class windows:

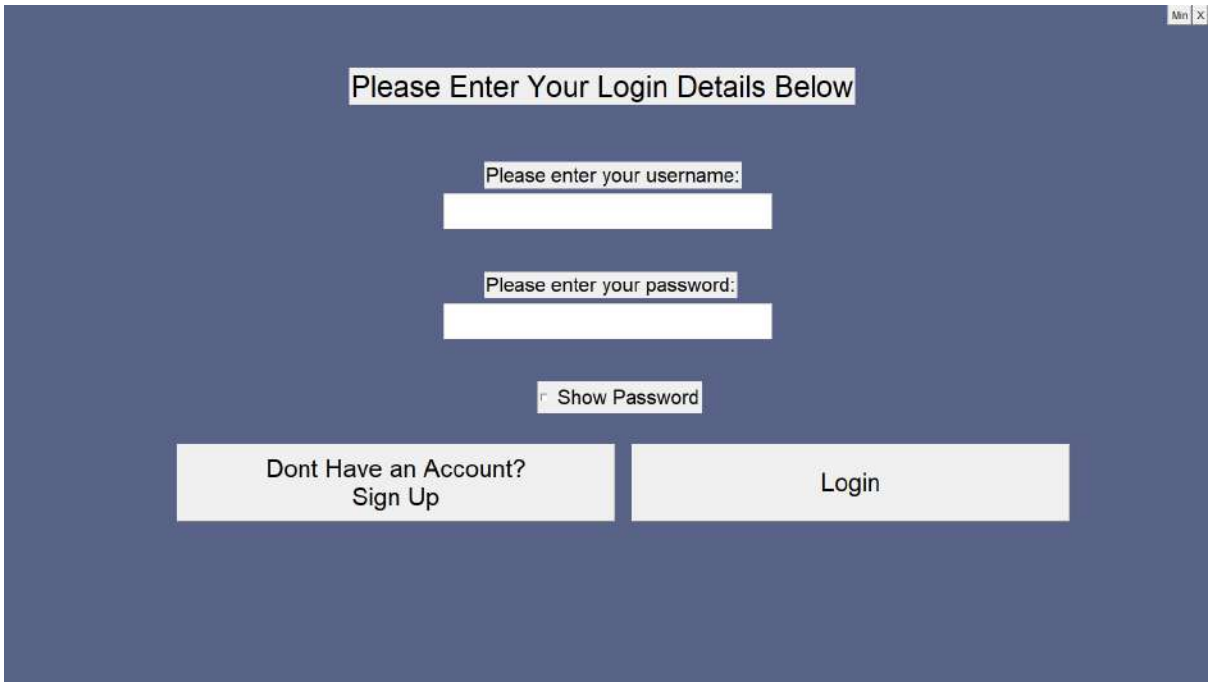
```
8
9  class windows(tk.Tk):
10
11     def __init__(self):
12         tk.Tk.__init__(self)
13         container = tk.Frame(self)
14
15         container.pack(side = "top", fill = "both", expand = False )
16         container.grid_rowconfigure(0, weight = 1)
17         container.grid_columnconfigure(0, weight = 1)
18
19         self.farmes = {}
20
21         for i in signinwindow, loginwindow, mainwindow:
22             frame = i(container, self)
23
24             self.farmes[i] = frame
25
26             frame.grid(row = 15, column = 15, sticky = "nesw")
27
28         self.show_frame(loginwindow)
29
30         global maxScreen
31         maxScreen = True
32
33
34     def show_frame(self,cont):
35         frame = self.farmes[cont]
36         frame.tkraise()
```

Inside of the class windows, I have appended a for loop that I created in Phase 4. I first added the new window name into the list of window names, then I changed which widow should be displayed first. Now I will need to create the class loginwindow.

```
class loginwindow(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent, bg = "#596387")
        self.checkVar = tk.IntVar()
        self.nextframe = lambda: controller.show_frame(mainwindow)
        ButtonFont = font.Font(size = 30, family="Helvetica")
        LabelFont = font.Font(size = 24, family = "Helvetica")
        EntryFont = font.Font(size = 35, family = "Helvetica")
        #Buttons-----------------------------------
        self.minmaxbutton = tk.Button(self,
                                      command = minmax_frame, text = "Min", font = (font.Font(size = 12, family="Helvetica")))
        self.minmaxbutton.place(x = 1855, y = 0)

        self.exitbutton = tk.Button(self,
                                    command = quitWindow, text = "X",  font = (font.Font(size = 12, family="Helvetica"))).place(x =1895 , y = 0)
        self.loginbutton = tk.Button(self,
                                     text = "Login", font = ButtonFont, width = 30, height = 2)
        self.loginbutton.place(x =1000 , y =700 )
        self.newuserbutton = tk.Button(self,
                                       text = """Dont Have an Account?
Sign Up""", font = ButtonFont , width = 30)
        self.newuserbutton.place(x =275 , y =700 )
        #Entry Boxs-------------------------------------------------
        self.usernameBox = tk.Entry(self,
                                    font = EntryFont)
        self.usernameBox.place(x = 700, y = 300)
        self.passwordBox = tk.Entry(self,
                                    show = "*", font = EntryFont)
        self.passwordBox.place(x = 700, y = 475)
        #Labels-----------------------------------------------------
        infomationLabel = tk.Label(self,
                                   text = """Please Enter Your Login Details Below""", font = font.Font(size = 35, family = "Helvetica")).place(x = 550, y = 100)
        usernameLabel = tk.Label(self,
                                 text = "Please enter your username:" , font = LabelFont).place(x = 765, y = 250)
        passwordLabel = tk.Label(self,
                                 text = "Please enter your password:" , font = LabelFont).place(x = 765, y = 425)
        #CheckBox---------------------------------------------------
        self.showpassBox = tk.Checkbutton(self,
                                          text = "Show Password",variable = self.checkVar,  font = LabelFont)
        self.showpassBox.place(x = 850, y = 600)
```

This code listing above creates this window:



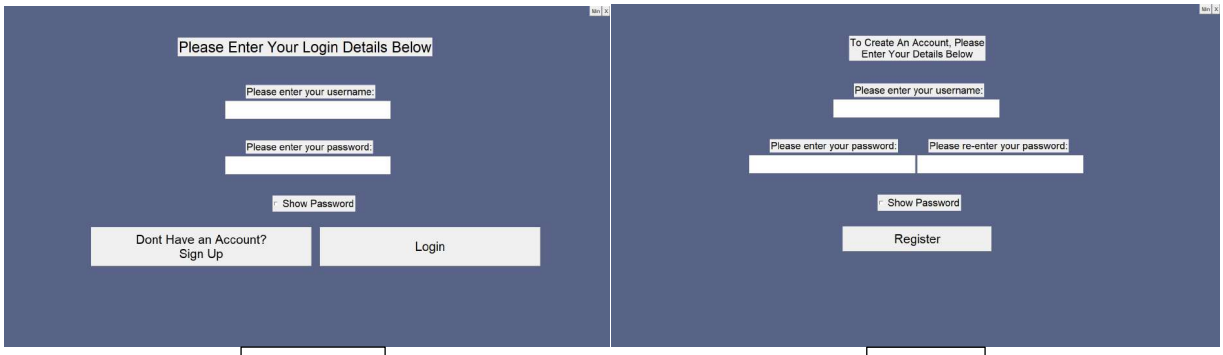
As you can see, I created a class for the login window called loginwindow, and in the initialization of this class the layout is created. Each item in this window has at least two lines dedicated to it; one is used to define what it is (for example self.showpassBox = tk.Checkbutton(self, text = "Show Password", variable = self.checkVar, font = LabelFont)) and the second states where it will be placed in the window (for example self.showpassBox.place(x = 850, y = 600)). For this window, I have used tkinters .place system which means each item is given a x and y co-ordinate which dictates where it is on the page, just like I did for the previous phase.

Phase 4, and 5 have very many similar functions inside of them, they both possess the show password check button, minimize / maximize screen button and the exit button, because of this I will not be explaining how I implemented them as it was covered in the previous section.

The first new widget I will be addressing is the Don't Have an Account button. I will need to assign a command to it which will take the user to the registration page.

```
58          self.newuserbutton = tk.Button(self,
59                                         text = """Dont Have an Account?
60  Sign Up""", font = ButtonFont , width = 30, command = lambda: controller.show_frame(signinwindow))
```

This is the same command I used on the registration page to redirect the user; however, I did not have to create a whole new pop-up window to accomplish this. This is because when using the controller, it will only be defined inside of the initializations of one of the children of a parent class and because I can assign the button command directly inside the initialization, I did not require any work around. See below showing the testing performed to see if the button functions correctly.



Before After

The final part of Phase 5 will be the creation of the function which allows the user to login to their account. For this, I need to append to the loginwindow class and define a new sub-routine to allow the user to do this. Like with the Phase 4, I will have to utilise csv for all the communication between my program and the external databases.

```
87      def login(self):
88          if len(self.passwordbox.get()) > 0 and len(self.usernamebox.get()) :
89              match = False
90              try:
91                  with open("UserData.csv", "r") as file:
92                      fieldnames = ["Username", "Password", "UserKey"]
93                      reader = csv.DictReader(file, fieldnames = fieldnames)
94                      for row in reader:
95                          if row["Username"] == self.usernamebox.get():
96                              match = True
97                              if row["Password"] == self.passwordbox.get():
98                                  Popup = tk.Toplevel()
99
100                                 Message = tk.Label(master=Popup,
101                                                    text= """Your logged in,
102 press ok to proceed""").grid(row=1, column=1, columnspan=2)
103                                 Button = tk.Button(master= Popup,
104                                                    text="Ok",
105                                                 command =self.nextframe).grid(row=2, column=2)
106                             else:
107                                 arg
108                     if match == False:
109                         arg
110             except:
111                 errorPopup = tk.Toplevel()
112
113                 errorMessage = tk.Label(master=errorPopup,
114                                         text= """Error: Ethier the username or
115 password is incorrect""").grid(row=1, column=1, columnspan=2)
116                 cancelButton = tk.Button(master= errorPopup,
117                                         text="Ok",
118                                      command = lambda: errorPopup.destroy()).grid(row=2, column=2)
119         else:
120             errorPopup = tk.Toplevel()
121
122             errorMessage = tk.Label(master=errorPopup,
123                                     text= """Error: Enter a username or
124 password is incorrect""").grid(row=1, column=1, columnspan=2)
125             cancelButton = tk.Button(master= errorPopup,
126                                     text="Ok",
127                                  command = lambda: errorPopup.destroy()).grid(row=2, column=2)
128
```

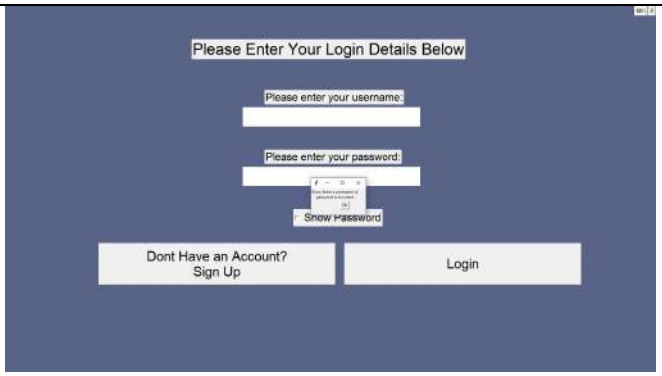
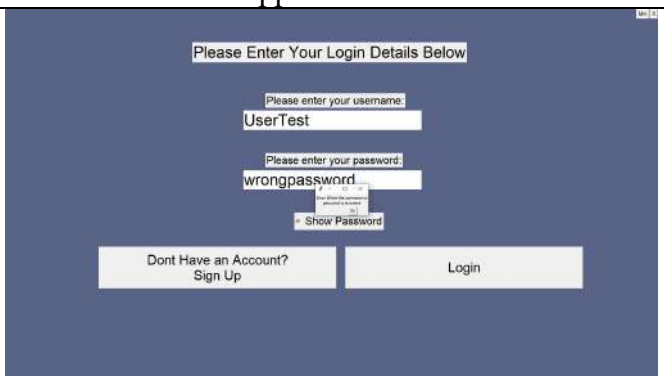
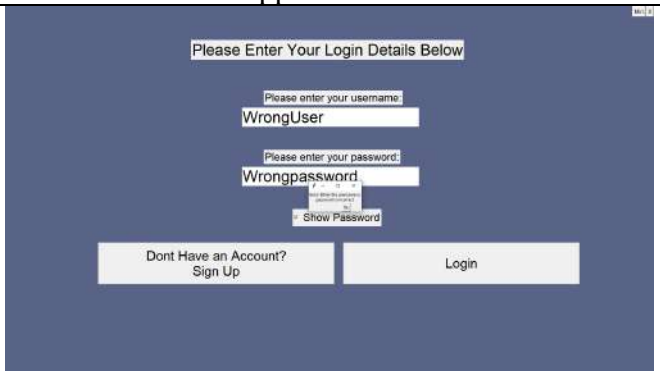
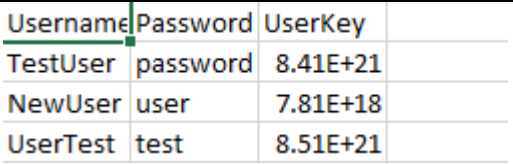
I then created the subroutine login (located inside of loginwindow). On line 88, I created an if statement, this is to check if the user has; entered a password and username. If they have met the requirements of the if statement, a new try statement is invoked (line 89-103). Inside of the try statement is where all the external database dialog will happen. At the start, the program opens the file UserData as a file to be read (r), then reads each row in the column Username to see if the username the user entered even exists. If the username does exist, then a new if statement is created to check that the password the user entered is linked to the username they entered. If both the username and password are linked a new pop-up window occurs, this window is identical in purpose and code to the pop-up window in the registration window. However, if the password does not link to the username or the username is not in the database an argument is passed through which will invoke the except statement and an error message will occur.

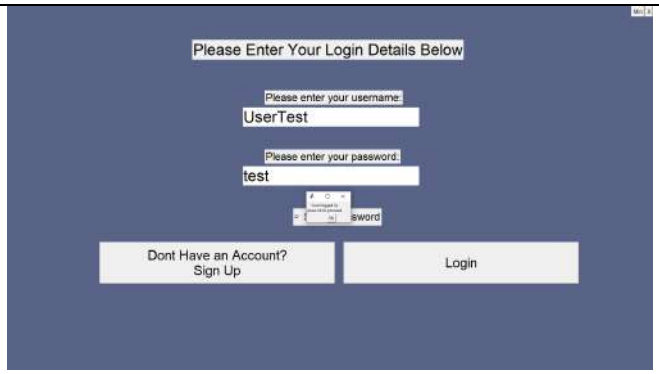
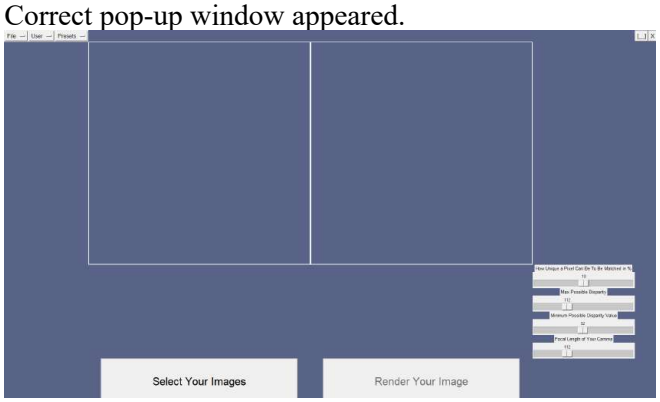
```
55          self.loginbutton = tk.Button(self,
56                                       text = "Login", font = ButtonFont, width = 30, height = 2, command = self.login)
```

I have now added the function login to the command of the login button. Now to make sure the system functions properly I will need to test it.

Testing:

Here is how I will be testing the system:

What I am testing	Data Entered	Data Type (erroneous, boundary, normal)	Result	Was this expected
1. Logging in with no data entered	null	erroneous	 The error window appered	Yes
2. Logging in with correct username but incorrect password	"UserTest" "wrongpassword"	erroneous	 The error window appered	Yes
3. Logging in with wrong username and incorrect password	"WrongUser" "wrongpassword"	erroneous	 The error window appeared	Yes
4. Logging in with correct data	"UserTest" "test"	normal	 Database during test.	Yes

			 Correct pop-up window appeared.  Main window correctly opened.	

Here is the code listing for Phase 5:

```
class loginwindow(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent, bg = "#596387")
        self.checkVar = tk.IntVar()
        self.nextframe = lambda: controller.show_frame(mainwindow)
        ButtonFont = font.Font(size = 30, family="Helvetica")
        LabelFont = font.Font(size = 24, family = "Helvetica")
        EntryFont = font.Font(size = 35, family = "Helvetica")
        #Buttons-----------------------------------
        self.minmaxbutton = tk.Button(self,
                                      command = minmax_frame, text = "Min", font = (font.Font(size = 12, family="Helvetica")))
        self.minmaxbutton.place(x = 1855, y = 0)

        self.exitbutton = tk.Button(self,
                                    command = quitWindow, text = "X",  font = (font.Font(size = 12, family="Helvetica"))).place(x =1895 , y = 0)
        self.loginbutton = tk.Button(self,
                                     text = "Login", font = ButtonFont, width = 30, height = 2, command = self.login)
        self.loginbutton.place(x =1000 , y =700 )
        self.newuserbutton = tk.Button(self,
                                       text = """Dont Have an Account?
Sign Up""", font = ButtonFont , width = 30, command = lambda: controller.show_frame(signinwindow))
        self.newuserbutton.place(x =275 , y =700 )
        #Entry Boxs-------------------------------------------------------
        self.usernamebox = tk.Entry(self,
                                    font = EntryFont)
        self.usernamebox.place(x = 700, y = 300)
        self.passwordbox = tk.Entry(self,
                                    show = "*", font = EntryFont)
        self.passwordbox.place(x = 700, y = 475)
        #Labels-----------------------------------------------------------
        infomationLabel = tk.Label(self,
                                   text = """Please Enter Your Login Details Below""", font = font.Font(size = 35, family = "Helvetica")).place(x = 550, y = 100)
        usernameLabel = tk.Label(self,
                                 text = "Please enter your username:" , font = LabelFont).place(x = 765, y = 250)
        passwordLabel = tk.Label(self,
                                 text = "Please enter your password:" , font = LabelFont).place(x = 765, y = 425)
        #CheckBox---------------------------------------------------------
        self.showpassBox = tk.Checkbutton(self,
                                          text = "Show Password",variable = self.checkVar, command = self.showpassword,  font = LabelFont)
        self.showpassBox.place(x = 850, y = 600)

    def showpassword(self):
        if self.checkVar.get() == 1:
            self.passwordbox.config(show = "")
        else:
            self.passwordbox.config(show = "*")

    def login(self):
        if len(self.passwordbox.get()) > 0 and len(self.usernamebox.get()) :
            match = False
            try:
                with open("UserData.csv", "r") as file:
                    fieldnames = ["Username", "Password", "UserKey"]
                    reader = csv.DictReader(file, fieldnames = fieldnames)
                    for row in reader:
                        if row["Username"] == self.usernamebox.get():
                            match = True
                            if row["Password"] == self.passwordbox.get():
                                Popup = tk.Toplevel()

                                Message = tk.Label(master=Popup,
                                                   text= """Your logged in,
press ok to proceed""").grid(row=1, column=1, columnspan=2)
                                Button = tk.Button(master= Popup,
                                                   text="Ok",
                                                   command =self.nextframe).grid(row=2, column=2)
                            else:
                                arg
                    if match == False:
                        arg
            except:
                errorPopup = tk.Toplevel()

                errorMessage = tk.Label(master=errorPopup,
                                        text= """Error: Ethier the username or
password is incorrect""").grid(row=1, column=1, columnspan=2)
                cancelButton = tk.Button(master= errorPopup,
                                         text="Ok",
                                         command = lambda: errorPopup.destroy()).grid(row=2, column=2)
        else:
            errorPopup = tk.Toplevel()

            errorMessage = tk.Label(master=errorPopup,
                                    text= """Error: Enter a username or
password is incorrect""").grid(row=1, column=1, columnspan=2)
            cancelButton = tk.Button(master= errorPopup,
                                     text="Ok",
                                     command = lambda: errorPopup.destroy()).grid(row=2, column=2)

```

Phase 5 review

What has been done?

Using Tkinter, I appended the main program file to add a user interface for the login system. The login interface is populated with, buttons, text, entry boxes, and a check button as per my design. The main button (for logging in) is linked to the function which processes the user data and either allows a user to proceed or make the user enter new details, all data that is being processed here links to an external database. The other main button (for going to the registration section) is only linked to a command which redirects the user.

How has it been tested?

Each time I have added a new part to the login interface, I executed the program to see if it worked, and if it the UI was aligned correctly. The registration functions have also been tested, with both erroneous and normal data (as specified in the testing table), each test I provide what I was testing, the data that I entered, the result and if that result was expected.

How it meets the success criteria and user expectations

The login window has a simple design and allows the users to log into an account and access the main window. It has met the following criteria:

- Clear front end
    - The login window should provide the following features:
        - An entry box for; their username, password.
        - A button for; logging in, creating an account, and for showing their password.

Changes in the design that have resulted from this section

This section has almost followed the design with the only change being that extra pop-up window has been added, the same as the registration window.

Summary of the whole project at this phase

The main file now contains both the user interfaces for the main, registration and login window, it also contains the functions for creating a new user, logging into a user, and rendering a video or selection of images. The program is almost finished, as all I have left to do is refine the program and add the last functions which required the other windows to be finished.

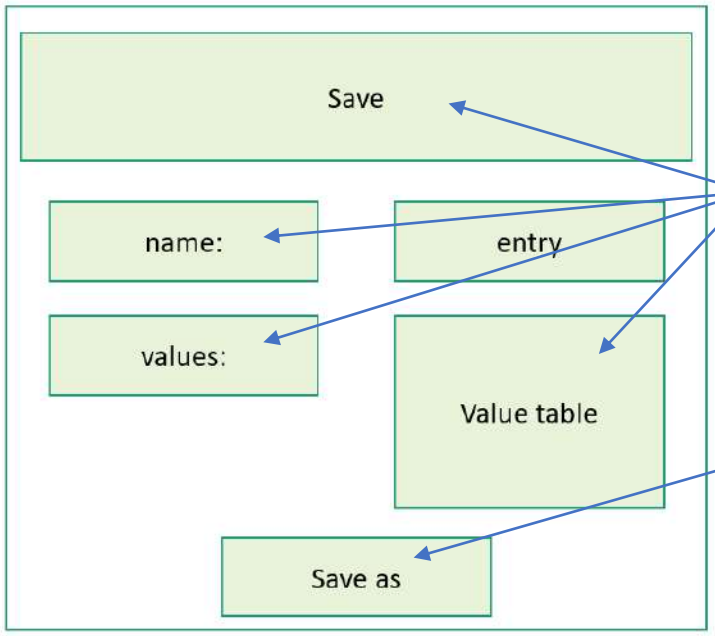
Phase 6, Saving and opening function

Pre-sets:

The first thing to address is the saving and opening of pre-sets inside of the main window. For this I will need to create two separate functions: one for saving, and one for opening. I will start with the saving function.

```
371     def save(self):
372         with open("PresetData.csv", "a") as file:
373             fieldnames = ("UserKey", "Name", "Values")
374             writer = csv.DictWriter(file, fieldnames = fieldnames)
```

To start with, I have created the function save (located inside of mainwindow). The function first opens the file PresetData to append it, however this is where I ran into a problem, I quickly realised that I would need a new pop-up window for this function (which I have not yet designed). Because of this I will now both design and create the save pop up window. This is the design I have created:



The design for the save pop up window contains labels, a button and an entry box. The labels text will consist of, “name:”, “values:”, “Save” and the label Value table will contain what the calibration options currently are.

The button inside this pop-up window will save the users data after being pressed, once saved the window will also terminate.

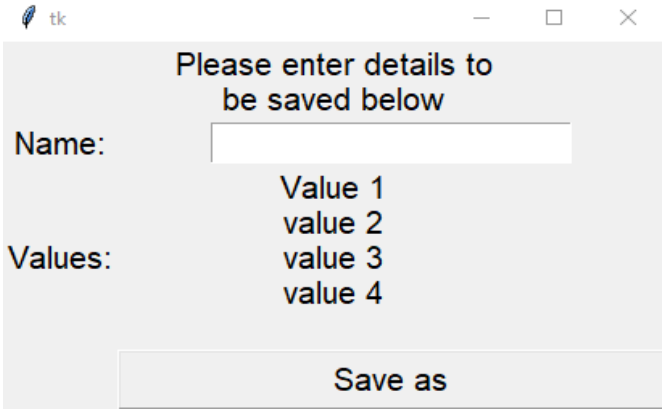
For the above design, I have included it in a separate file. This is the code I created:

```
import tkinter as tk
import tkinter.font as font

class saveWindow():
    def __init__(self):
        Popup = tk.Toplevel()
        font = tk.font.Font(size = 15, family="Helvetica")
        #Labels--------------------------------------------------------------------
        saveLabel = tk.Label(master = Popup,
                             text= """Please enter details to
be saved below""", font = font).grid(row=1, column=2, columnspan=3)
        nameLabel = tk.Label(master = Popup,
                             text= "Name:", font = font).grid(row=4, column=2)
        valueLabel = tk.Label(master = Popup,
                             text= "Values:", font = font).grid(row=5, column=2)
        tableLabel = tk.Label(master = Popup,
                             text= """Value 1
value 2
value 3
value 4
""", font = font).grid(row=5, column=1, columnspan=3)
        #Button--------------------------------------------------------------------
        saveButton = tk.Button(master= Popup,
                             text="Save as", width = 30, font = font).grid(row=6, column=3)
        #EntryBox------------------------------------------------------------------
        entryBox = tk.Entry(master = Popup, font = font).grid(row=4, column=3)

```

I have created a class called saveWindow, and in the initialisation of this class, the layout is created. On line 6, the pop-up window is created and is raised to the top level, then on lines 9-21 all the labels are created, these labels include saveLabel: which tells the user to enter the details they want to assign the pre-set below, nameLabel: which says name, tableLabel: which will contain the pre-sets values but currently has the place holder value 1- 4, and valueLabel: which says Values. Then on lines 23, a button is defined, this button has the text Save as, however it does not have a command. Lastly on line 26, I created an entry box named entryBox, this is where the user will input the name that they want this pre-set to be called. Once executed this code creates this window:



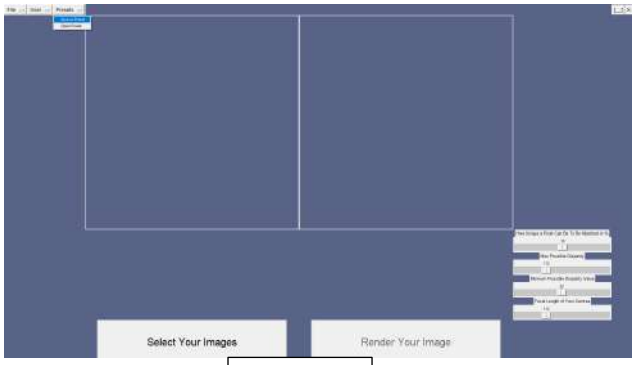
I then appended the main file and added this class. I now will need to find a way of calling this class to show the pop-up window.

```
368     def OptionChange(self, event):
369         if self.Fileclicked.get() == "Change File Type":self.Fileclicked.set("File"), self.changeFileType()
370         elif self.Presetsclicked.get()  == "Save as Preset":self.Presetsclicked.set("Presets"), self.save()
371
372     def save(self):
373         saveWindow()
```

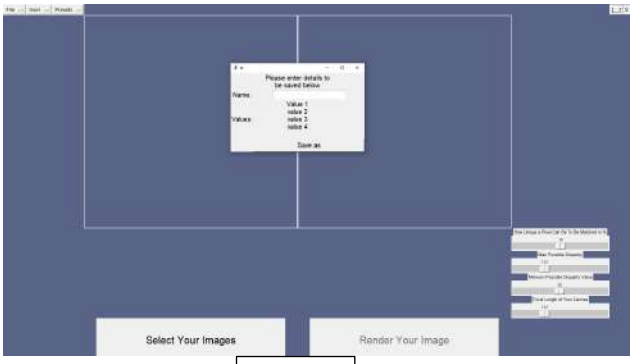
Inside of the class mainwindow, I appended the subroutine OptionChange. All that this routine does is when the option is selected it executes a command. In this case, when Save as Preset is selected inside of menu Presets, the Presets menu icon is set to remain as Presets, and the subroutine save is executed.

```
247         presetsdrop = tk.OptionMenu(self,
248                                     self.Presetsclicked, *Presetsoptions, command = self.OptionChange)
```

Next, in the initialization of mainwindow, I appended the option menu presetdrop to have a command. The command I assigned was OptionChange, and now if the option save as preset is selected the function save should execute. Here is the result when I select that menu option:



Before



After

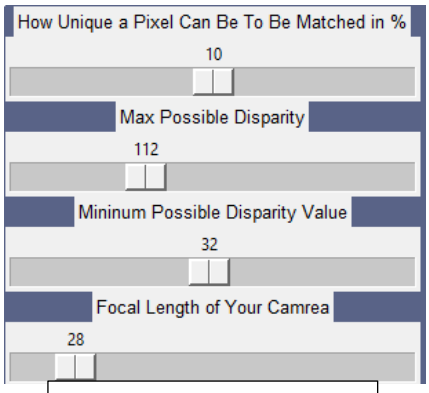
The test of calling the function worked perfectly. Next, I will need to pass data into the class saveWindow that will later use the data.

```
372     def save(self):
373         x = [self.minDispSlider.get(), self.maxDispSlider.get(), self.focalSlider.get(), self.uniqueRatioSlider.get()]
374         saveWindow(x)
```

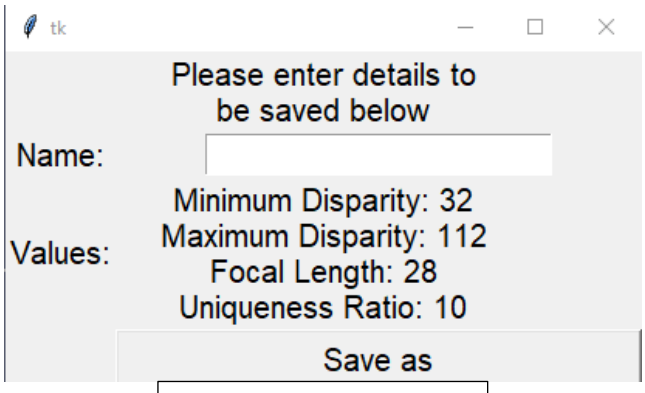
I have created a variable x which contains the values of each calibration option in the form of a string. x is then passed into saveWindow.

```
418 class saveWindow():
419     def __init__(self, x):
420         Popup = tk.Toplevel()
421         font = tk.font.Font(size = 15, family="Helvetica")
422         #Labels-------------------------------------------------------------------
423         saveLabel = tk.Label(master = Popup,
424                              text= """Please enter details to
425 be saved below""", font = font).grid(row=1, column=2, columnspan=3)
426         nameLabel = tk.Label(master = Popup,
427                              text= "Name:", font = font).grid(row=4, column=2)
428         valueLabel = tk.Label(master = Popup,
429                              text= "Values:", font = font).grid(row=5, column=2)
430         tableLabel = tk.Label(master = Popup,
431                              text= ("""Minimum Disparity: %s"""% x[0] + """
432 Maximum Disparity: %s"""% x[1] + """
433 Focal Length: %s"""% x[2] + """
434 Uniqueness Ratio: %s"""% x[3]), font = font).grid(row=5, column=1, columnspan=3)
435         #Button-------------------------------------------------------------------
436         saveButton = tk.Button(master= Popup,
437                              text="Save as", width = 30, font = font).grid(row=6, column=3)
438         #EntryBox-----------------------------------------------------------------
439         entryBox = tk.Entry(master = Popup, font = font).grid(row=4, column=3)
```

I then pass x into the initialization of the saveWindow, x is then used within the text of tableLabel. Each value of x is then assigned to their corresponding title (example Minimum Disparity and x[0] which holds the mindispslider value). When executed, the popup window should now contain all the calibration data. When tested these were the results:



Calibration options



Pop up window

As the test was successful, the final part to saving pre-sets is saving that data to a file. For this, I will utilise import csv to manage all the file handling.

```
class windows(tk.Tk):

    def __init__(self):
        tk.Tk.__init__(self)
        container = tk.Frame(self)

        container.pack(side = "top", fill = "both", expand = False )
        container.grid_rowconfigure(0, weight = 1)
        container.grid_columnconfigure(0, weight = 1)

        self.farmes = {}

        for i in signinwindow, loginwindow, mainwindow:
            frame = i(container, self)

            self.farmes[i] = frame

            frame.grid(row = 15, column = 15, sticky = "nesw")

        self.show_frame(loginwindow)

        global maxScreen
        global key
        key = ""
```

```
    def login(self):
        if len(self.passwordbox.get()) > 0 and len(self.usernamebox.get()) > 0 :
            match = False
            try:
                with open("UserData.csv", "r") as file:
                    fieldnames = ["Username", "Password", "UserKey"]
                    reader = csv.DictReader(file, fieldnames = fieldnames)
                    for row in reader:
                        if row["Username"] == self.usernamebox.get():
                            match = True
                            if row["Password"] == self.passwordbox.get():
                                global key
                                key = (''.join(str(ord(l)) for l in self.usernamebox.get()))
```

To allow the users key to be used throughout the saving and opening processes, I appended the class windows and added key as a global variable, next I assigned key with the UserKey of the user who just logged in (the same line defining key can also be found inside of registration).

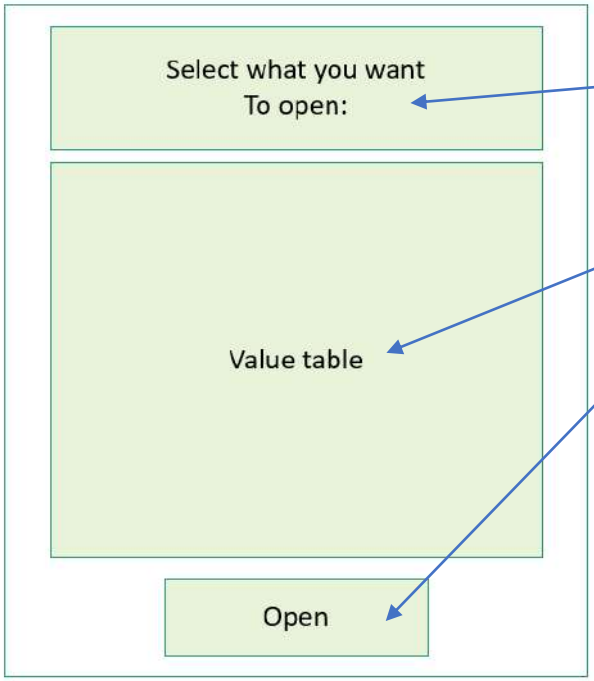
```
445     def save(self):
446         if len(self.entryBox.get()) > 0:
447             try:
448                 with open("PresetData.csv", "r") as file:
449                     fieldnames = ["UserKey", "Name", "Data"]
450                     reader = csv.DictReader(file, fieldnames = fieldnames)
451                     for row in reader:
452                         if row["UserKey"] == key:
453                             if row["Name"] == self.entryBox.get():
454                                 arg
455                 with open("PresetData.csv", "a") as file:
456                     fieldnames = ["UserKey","Name","Data"]
457                     writer = csv.DictWriter(file, fieldnames = fieldnames)
458                     writer.writerow({"UserKey": "%s" % key,
459                                      "Name": "%s" % self.entryBox.get(),
460                                      "Data": "%s" % self.x})
```

I next created a function save inside of saveWindow. On line 446 inside of this function, I have created an if statement, this is to check if the user has entered a username. If they have met the requirements of the if statement, a try statement is invoked (lines 447-460). Inside of the try statement is where all the external database dialog will happen. At the start, the program opens the file PresetData as a file to be read (r), then reads each row in column UserKey in the file to find if that user had any previously made pre-sets, if they have it then checks if the pre-sets, they previously made are named the same as what they are trying to call this pre-set. If the names are different, then the sub routine re-opens the file PresetData as a file it can append (a) to. It then appends the pre-sets data to the file. However, if the username is taken, an argument is passed through which will invoke the except statement.

```
460         except:
461             ErrorPopup = tk.Toplevel()
462 
463             Message = tk.Label(master = ErrorPopup,
464                                text= """Youve already used that name for
465 a preset choose another""").grid(row = 1, column = 1, columnspan =2 )
466             Button = tk.Button(master = ErrorPopup,
467                                text = "Ok", command = lambda: ErrorPopup.destroy()).grid(row=2, column=2)
468 
```

The except statement, when invoked, will create an error window. A similar error window is created if the if statement is not fulfilled. I discussed these error type windows previously, so I will not go into any detail here.

Now I have finished the saving of pre-sets, I will now start on opening pre-sets. The design and creation of the opening pop-up window is shown below.



The design for the open pop-up window contains a label, a button, and a list box. The labels text will contain “Select what you want to open”.

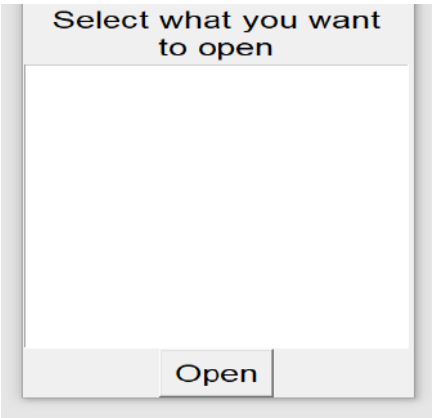
The list box will contain all the names of previously created pre-sets and will allow the user to select one.

The button inside this pop-up window will open the user’s data after being selected, once opened the window will also terminate.

For the above design, I have included it in a separate file. This is the code I created:

```
1  import tkinter as tk
2  import tkinter.font as font
3  
4  class openWindow():
5      def __init__(self):
6          Popup = tk.Toplevel()
7          font = tk.font.Font(size = 20, family="Helvetica")
8          #Label-------------------------------------------------
9          openLabel = tk.Label(master = Popup,
10                               text =  """Select what you want
11 to open""", font = font).grid(row = 1, column = 1)
12         #Listbox ----------------------------------------------
13         selectionBox = tk.Listbox(master = Popup,
14                                   font = font).grid(row = 2, column = 1)
15         #Button------------------------------------------------
16         openButton = tk.Button(master = Popup,
17                                text = "Open", font = font).grid(row = 3, column = 1)
18 
19 
20 openWindow()
```

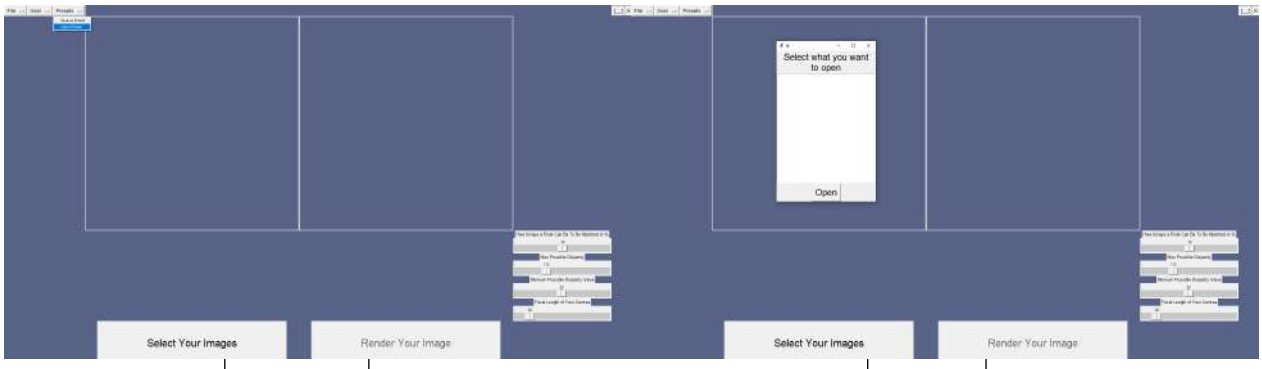
I created a class called openWindow, and in the initialisation of this class the layout is created and the raised. On line 6, the pop-up window is created and is raised to the top level, then on lines 9-11 the label is created. Next on lines 13-14, a list box is created, this list box does not yet have any options to be selected. This will be addressed later. Then on lines 16-18, a button is defined, this button has the text Open, however it does not have a command. Once executed this code creates the following window:



I have appended the main file and added this class, next I will need to find a way of calling this class to show the pop-up window.

```
371     def OptionChange(self, event):
372         if self.Fileclicked.get() == "Change File Type":self.Fileclicked.set("File"), self.changeFileType()
373         elif self.Presetsclicked.get()  == "Save as Preset":self.Presetsclicked.set("Presets"), self.save()
374         elif self.Presetsclicked.get()  == "Open Preset":self.Presetsclicked.set("Presets"), self.open()
```

Next, inside of the class mainwindow, I have appended the subroutine OptionChange. All that this routine does is when the option is selected it executes a command. In this case, when Open Preset is selected inside of menu Presets, the Presets menu icon is set to remain as Presets and the subroutine open is executed. Here is the result, when I click that menu option:



Before After

As the test of calling the function worked correctly, I will now need to find a way of showing the user all the pre-sets they have previously made.

```
486
487     def ListOptions(self):
488             with open("PresetData.csv", "r") as file:
489                 fieldnames = ["UserKey", "Name", "Data"]
490                 reader = csv.DictReader(file, fieldnames = fieldnames)
491                 for row in reader:
492                     if row["UserKey"] == key:
493                         self.selectionBox.insert("end", row["Name"])
494
495
```

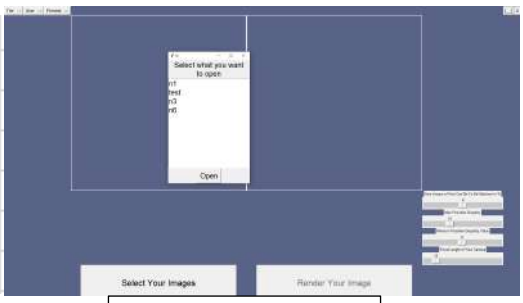
I have created a function named ListOptions, inside this function the database PresetData is opened. The function then searches through the database looking for any pre-sets that a user has created. If they have created a pre-set then the function inserts the name of their pre-sets into the selection box.

```
469 class openWindow():
470     def __init__(self):
471         Popup = tk.Toplevel()
472         font = tk.font.Font(size = 20, family="Helvetica")
473         #Label-------------------------------------------------
474         openLabel = tk.Label(master = Popup,
475                              text =  """Select what you want
476 to open""", font = font).grid(row = 1, column = 1)
477         #Listbox -----------------------------------------
478         self.selectionBox = tk.Listbox(master = Popup,
479                                        font = font)
480         self.selectionBox.grid(row = 2, column = 1)
481         self.ListOptions()
```

Inside of the initialization of openWindow, I called the function ListOptions. This should now insert all the pre-sets a user account has created. To test this I logged into the account UserTest (UserKey = 8.51E+21) and selected Open Presets.

UserKey	Name	Data
8.51E+21	n1	[32, 112, 28, 10]
8.41E+21	n2	[5, 192, 41, 9]
8.51E+21	test	[32, 112, 28, 10]
8.51E+21	n3	[32, 112, 28, 10]
8.51E+18	n4	[32, 112, 28, 10]
8.51E+21	n6	[32, 112, 28, 10]

Pre-sets Database



Pop-up window

As the test was successful, the final part to opening pre-sets, is changing the files calibration to fit the pre-set selected.

```
500     def open(self):
501         if len(self.selectionBox.get("anchor")) > 0:
502             with open("PresetData.csv", "r") as file:
503                     fieldnames = ["UserKey", "Name", "Data"]
504                     reader = csv.DictReader(file, fieldnames = fieldnames)
505                     for row in reader:
506                         if row["UserKey"] == key:
507                             if row["Name"] == self.selectionBox.get("anchor"):
508                                 data = row["Data"]
509                                 data = data.replace("[", "")
510                                 data = data.replace("]", "")
511                                 data = data.replace(",", "")
512                                 data = data.split(" ")
513                                 dataList = list(map(int, data))
```

Inside of the class openWindow, I have created a new function open. When called this function will first check if the user has selected an option (line 501), if they have, the program then opens the database PresetData. The function then searches each line of the database until it finds the users key, and if that line holds the name of the pre-set they chose the function then reads the column data of that line. However, the data that is read is in the form a string is not a list of integers, so we must reformat this string into a list. To do this the function first replaces both the square brackets and commas with empty string values, then the variable dataList is created and assigned the value of the list of the integers in data (line 513).

```
285         global uniqueRatioSlider, maxDispSlider, minDispSlider, focalSlider
286         uniqueRatioSlider = self.uniqueRatioSlider
287         maxDispSlider = self.maxDispSlider
288         minDispSlider = self.minDispSlider
289         focalSlider = self.focalSlider
```

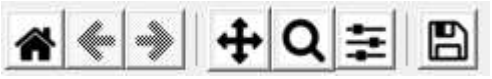
To allow the calibration options to be altered, I have made each slider a global variable, this allows the sliders to be accessed within any part of the program.

```
513                             minDispSlider.set(data[0])
514                             maxDispSlider.set(data[1])
515                             focalSlider.set(data[2])
516                             uniqueRatioSlider.set(data[3])
517
518         else:
519             ErrorPopup = tk.Toplevel()
520
521             Message = tk.Label(master = ErrorPopup,
522                                text= "Please select a pre-set").grid(row = 1, column = 1, columnspan =2 )
523             Button = tk.Button(master = ErrorPopup,
524                                text = "Ok", command = lambda: ErrorPopup.destroy()).grid(row=2, column=2)
525
```

The function now sets the sliders to their corresponding values contained within the pre-set. However, if the user hasn't chosen a pre-set and the if statement is failed an error popup is created with the text "Please select a pre-set".

Renderings:

The next thing to address is the saving and opening of renderings inside of the main window. For this I will need to create two separate functions: one for saving, and one for opening. I will start with the saving function.



Matplotlib (what I have used for some of the renderings) has a built-in save function as you can see above, which eliminates the need for me to create a save function. However, I will still need to find a way to save the renderings created by OpenCV.

```
    def showimage(self, points, colours, rotation, translation, camreaMatrix, distortion, width, height):
        global renderedimage
        renderedimage = self.projectThePoints(points, colours, rotation, translation, camreaMatrix, distortion, width, height)
        cv2.imshow("Your Rendered Image", self.projectThePoints(points, colours, rotation, translation, camreaMatrix, distortion, width, height))
```

Inside of the showimage function inside RenderedimageData, I have created a new global variable renderedimage, this allows me to access this variable anywhere in the code. I then assigned the information which holds the data, which is used for displaying the rendering, to the rendered image data.

```
class saveWindow():
    def __init__(self, x):
        Popup = tk.Toplevel()
        font = tk.font.Font(size = 15, family="Helvetica")
        self.x = x
        #Labels-------------------------------------------------------------------
        saveLabel = tk.Label(master = Popup,
                             text= """Please enter details to
be saved below""", font = font).grid(row=1, column=2, columnspan=3)
        nameLabel = tk.Label(master = Popup,
                             text= "Name:", font = font).grid(row=4, column=2)
        valueLabel = tk.Label(master = Popup,
                             text= "Values:", font = font).grid(row=5, column=2)
        if len(self.x) == 4:
            tableLabel = tk.Label(master = Popup,
                              font = font, text= ("""Minimum Disparity: %s"""% x[0] + """
Maximum Disparity: %s"""% x[1] + """
Focal Length: %s"""% x[2] + """
Uniqueness Ratio: %s"""% x[3])).grid(row=5, column=1, columnspan=3)
        else:
            tableLabel = tk.Label(master = Popup,
                              font = font, text= ("""This will save your
last rendering of two images""")).grid(row=5, column=1, columnspan=3)
```

I then appended the class saveWindow and inside its initialisation, I created an if statement. This is to see if the user is saving a pre-set or a rendering. It checks this by seeing the length of the value x, if its length is four, it will be a pre-set as that is the only length the value x can be if a pre-set is saved. If a pre-set is being saved, it sets the text of the tableLabel to the values being saved, however if it is a rendering being saved, it sets it to “This will save your last rendering of two images”. The reason for this approach, rather than making a new class for saving renderings, is to not write code and make the system more compact and of course less to test.

```
    def save(self):
        if len(self.entryBox.get()) > 0:
            try:
                if len(self.x) == 4:
                    with open("PresetData.csv", "r") as file:
                        fieldnames = ["UserKey", "Name", "Data"]
                        reader = csv.DictReader(file, fieldnames = fieldnames)
                        for row in reader:
                            if row["UserKey"] == key:
                                if row["Name"] == self.entryBox.get():
                                    arg
                    with open("PresetData.csv", "a") as file:
                        fieldnames = ["UserKey","Name","Data"]
                        writer = csv.DictWriter(file, fieldnames = fieldnames)
                        writer.writerow({"UserKey": "%s" % key,
                                         "Name": "%s" % self.entryBox.get(),
                                         "Data": self.x})
                else:
                    try:
                        if len(renderedimage) != 0:
                            try:
                                with open("RenderedData.csv", "r") as file:
                                    fieldnames = ["UserKey", "Name", "Data"]
                                    reader = csv.DictReader(file, fieldnames = fieldnames)
                                    for row in reader:
                                        if row["UserKey"] == key:
                                            if row["Name"] == self.entryBox.get():
                                                arg
                                with open("RenderedData.csv", "a") as file:
                                    fieldnames = ["UserKey","Name","Data"]
                                    writer = csv.DictWriter(file, fieldnames = fieldnames)
                                    cv2.ppf_match_3d.writePLY(renderedimage, self.entryBox.get())
                                    writer.writerow({"UserKey": "%s" % key,
                                                     "Name": "%s" % self.entryBox.get()})
                            except:
                                ErrorPopup = tk.Toplevel()

                                Message = tk.Label(master = ErrorPopup,
                                                   text= """Youve already used that name
please choose another""").grid(row = 1, column = 1)
                                Button = tk.Button(master = ErrorPopup,
                                                   text = "Ok", command = lambda: ErrorPopup.destroy()).grid(row=2, column=1)
                    except:
                        ErrorPopup = tk.Toplevel()

                        Message = tk.Label(master = ErrorPopup,
                                           text= "     Please render a image first     ").grid(row = 1, column = 1)
                        Button = tk.Button(master = ErrorPopup,
                                           text = "Ok", command = lambda: ErrorPopup.destroy()).grid(row=2, column=1)
```

I then appended an if statement to the function save, this will then query, if the length of x is equal to four if so, it will execute the save function for pre-sets, however if not it creates a new try except statement. The statement is the same, as the one for saving pre-sets with the only differences being where it is saved (RenderedData.csv), what is saved (Name of the file), how its saved (a built in OpenCV function) and the extra try/except statement which is there to ensure that the user has previously rendered an image.

```
380     def OptionChange(self, event):
381         if self.Fileclicked.get() == "Change File Type":self.Fileclicked.set("File"), self.changeFileType()
382         elif self.Fileclicked.get() == "Save":self.Fileclicked.set("File"), saveWindow([])
```

Inside of the mainwindow class, I have appended the OptionChange function to bind the command saveWindow to the menu option Save.

Now I have finished the saving of renderings I will now start on opening renderings.

```
650 class openWindow():
651     def __init__(self, _type):
652         self.type = _type
```

I first pass a new variable _type into the class openWindow, this is to be used to distinguish if the user is trying to open a pre-set or a rendering.

```
669     def ListOptions(self):
670         if self.type == "pre-set":
671             with open("PresetData.csv", "r") as file:
672                 fieldnames = ["UserKey", "Name", "Data"]
673                 reader = csv.DictReader(file, fieldnames = fieldnames)
674                 for row in reader:
675                     if row["UserKey"] == key:
676                         self.selectionBox.insert("end", row["Name"])
677         else:
678             with open("RenderedData.csv", "r") as file:
679                 fieldnames = ["UserKey", "Name", "Data"]
680                 reader = csv.DictReader(file, fieldnames = fieldnames)
681                 for row in reader:
682                     if row["UserKey"] == key:
683                         self.selectionBox.insert("end", row["Name"])
```

I have then appended the function ListOptions inside of the class openWindow. Inside of it, I have created an if statement which checks the type of file the user is trying to open. Then inside of the else statement, the file RenderedData is opened and all the saved renderings the user has created are displayed and can be chosen from.

```
684     def open(self):
685         if len(self.selectionBox.get("anchor")) > 0:
686             if self.type == "pre-set":
687                 with open("PresetData.csv", "r") as file:
688                         fieldnames = ["UserKey", "Name", "Data"]
689                         reader = csv.DictReader(file, fieldnames = fieldnames)
690                         for row in reader:
691                             if row["UserKey"] == key:
692                                 if row["Name"] == self.selectionBox.get("anchor"):
693                                     data = row["Data"]
694                                     data = data.replace("[", "")
695                                     data = data.replace("]", "")
696                                     data = data.replace(",", "")
697                                     data = data.split(" ")
698                                     dataList = list(map(int, data))
699                                     minDispSlider.set(data[0])
700                                     maxDispSlider.set(data[1])
701                                     focalSlider.set(data[2])
702                                     uniqueRatioSlider.set(data[3])
703             else:
704                 with open("RenderedData.csv", "r") as file:
705                         fieldnames = ["UserKey", "Name", "Data"]
706                         reader = csv.DictReader(file, fieldnames = fieldnames)
707                         for row in reader:
708                             if row["UserKey"] == key:
709                                 if row["Name"] == self.selectionBox.get("anchor"):
710                                     data = row["Name"]
711                                     file = cv2.ppf_match_3d.loadPLYSimple(data)
712                                     cv2.imshow("Your Rendered Image", file)
```

The next step in allowing the user to open their saved renderings is opening and displaying the selected renderings. The method of opening the selected data is essentially the same as it is for pre-sets, however with the main difference being what happens to the data selected. Whatever file the user chooses, its matrix will be loaded by the built in OpenCV (function seen on line 711), and the rendering is displayed as shown on line 712.

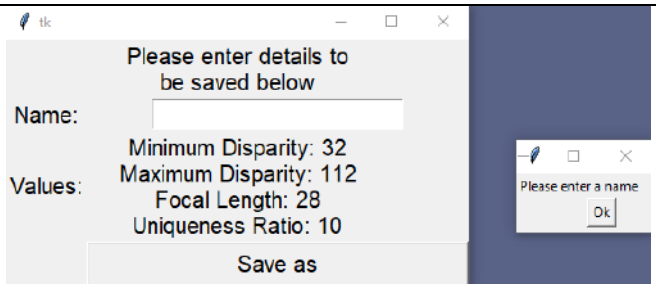
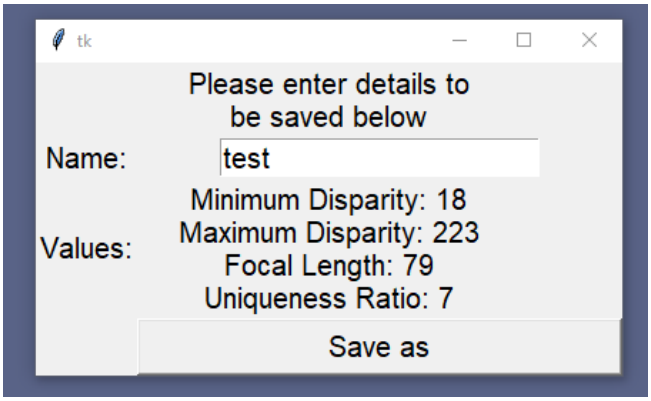
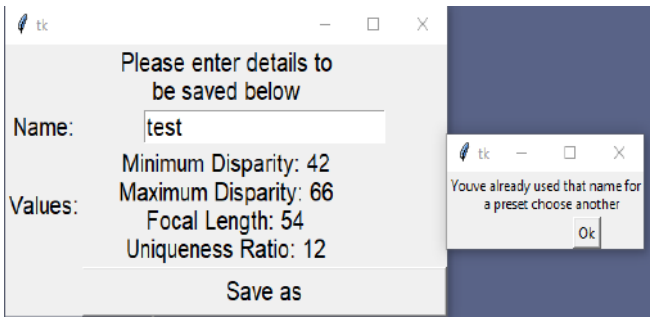
```
380     def OptionChange(self, event):
381         if self.Fileclicked.get() == "Change File Type":self.Fileclicked.set("File"), self.changeFileType()
382         elif self.Fileclicked.get() == "Save":self.Fileclicked.set("File"), saveWindow([])
383         elif self.Fileclicked.get() == "Open":self.Fileclicked.set("File"), openWindow("rendering")
384         elif self.Presetsclicked.get()  == "Save as Preset":self.Presetsclicked.set("Presets"), self.save()
385         elif self.Presetsclicked.get()  == "Open Preset":self.Presetsclicked.set("Presets"), openWindow("pre-set")
386
```

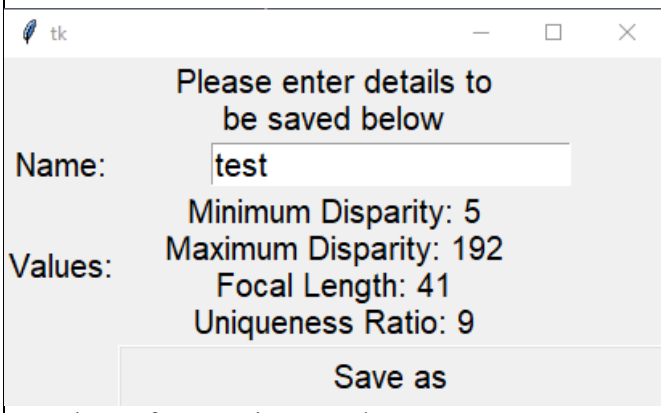
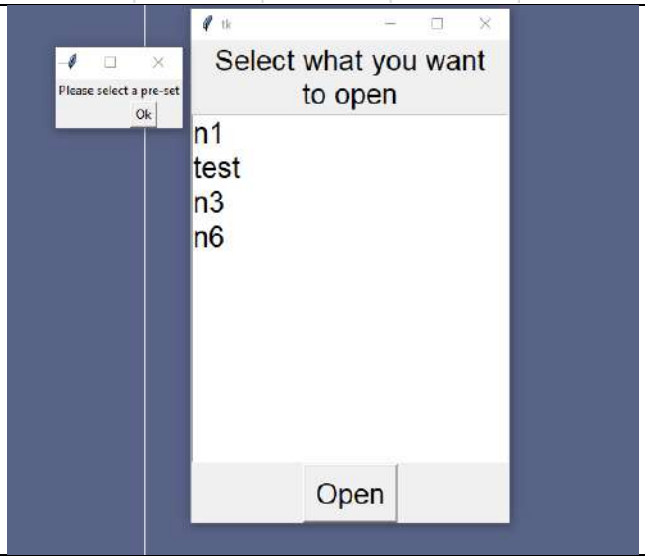
I have appended the function OptionChange located inside the class mainwindow. In this function (on line 383) I have added the command openWindow to the open option inside of file, I also pass through the _type rendering, and on line 385 passed through the _type pre-set into the open pre-set option.

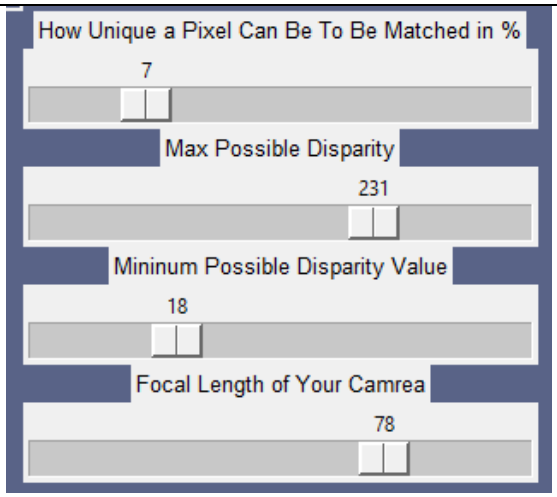
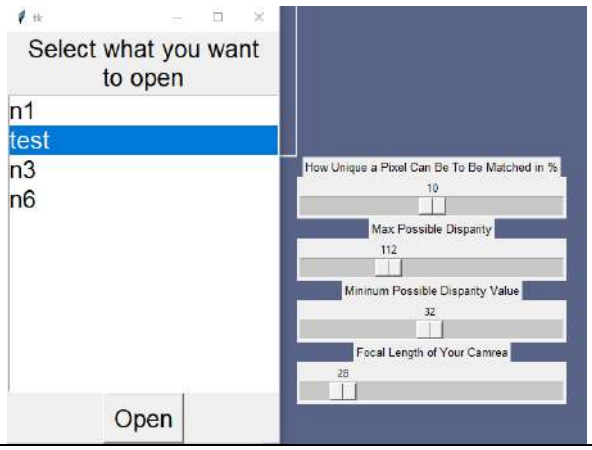
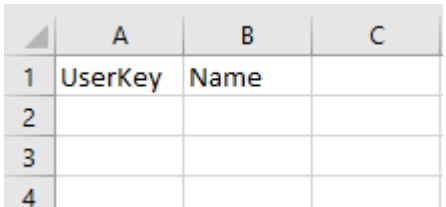
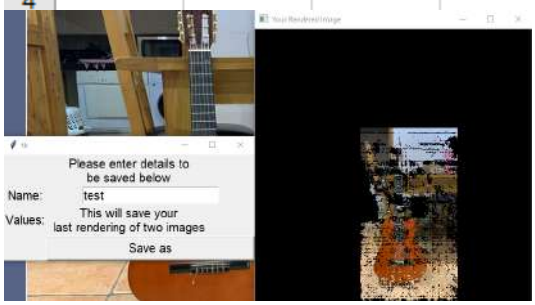
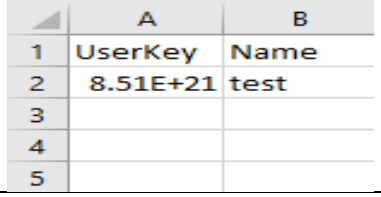
You can now both save and open pre-sets and renderings.

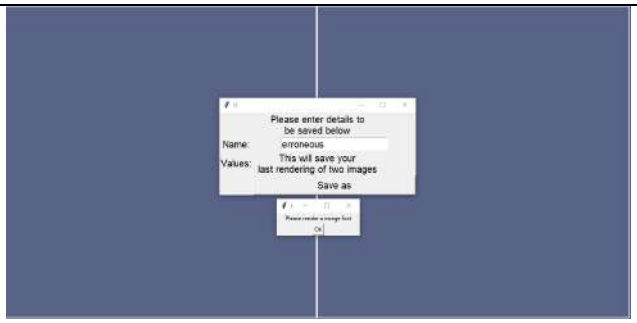
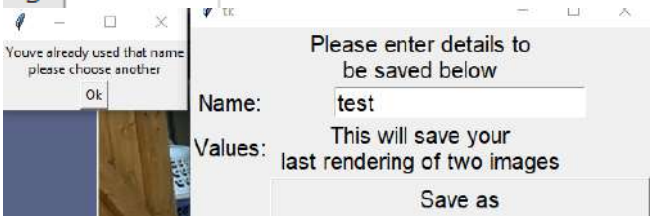
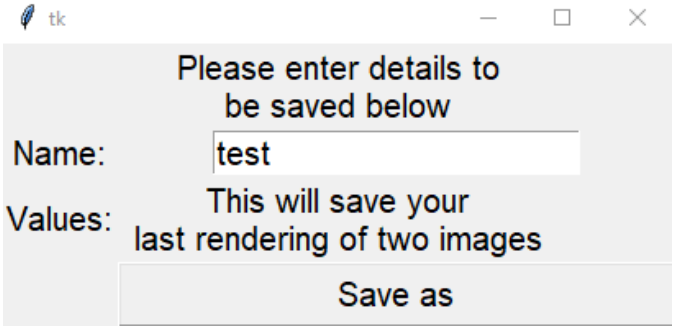
Testing:

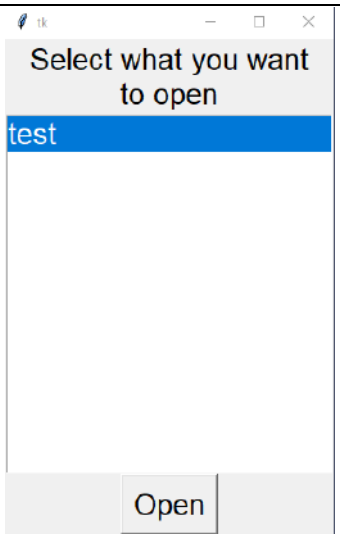
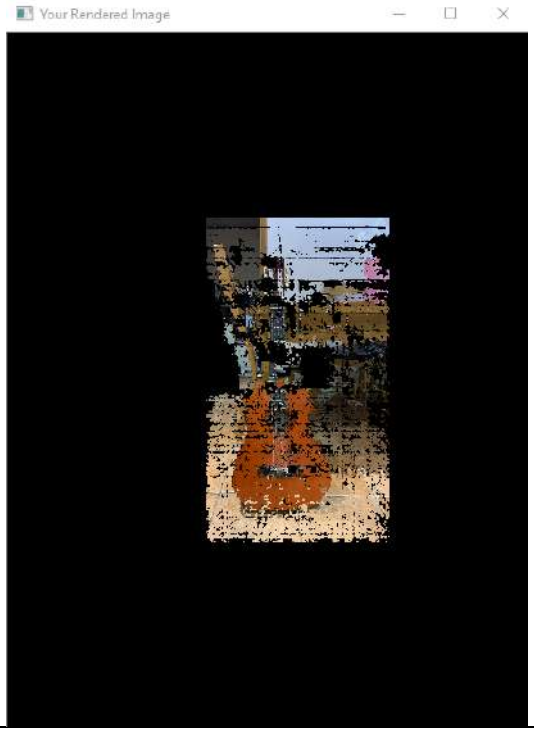
Here is how I will be testing these functions:

What I am testing	Data Entered	Data Type (erroneous, boundary, normal)	Result	Was this expected
1. trying to save with no data entered	null	erroneous	 The error window appered	Yes
2. Saving calibration data as a pre-set	User = UserTest, Name = “test”	normal	Database before the test: UserKey | Name | Data  Database after save is pressed: UserKey | Name | Data 8.51151E+21 | test | [18, 223, 79, 7]	Yes
3. Saving with already taken name	User = UserTest, Name = “test”	erroneous	 The error window appeared	Yes

4. Saving a pre-set with the same name as another users pre-set.	User = TestUser Name = “test”	normal	This was the database before test. UserKey | Name | Data 8.51151E+21 | test | [18, 223, 79, 7]  Database after save is pressed: UserKey | Name | Data 8.51E+21 | test | [18, 223, 79, 7] 8.41E+21 | test | [5, 192, 41, 9]	Yes
5. pressing open without selecting a pre-set	User = UserTest	erroneous		Yes
6. Opening a pre-set	User = UserTest Pre-set = test	normal	Pre-sets data: 8.51E+21 | test | [32, 112, 28, 10] Calibration settings before:	Yes

			 After: 	
7. Saving a rendering.	User = UserTest name = test	normal	Database before test:   Database after saved was pressed: 	Yes

8. Trying to save a rendering without first creating one.	User = UserTest name = erroneous	erroneous	 Correct Pop-up window created	Yes
9. Saving with already taken name	User = UserTest Name = test	erroneous	Datebase bfore test: A B 1 UserKey Name 2 8.51E+21 test 3 4 5  Correct po-up created	Yes
10. Saving a rendering with the same name as another users rendering.	User = TestUser Name = test	normal	Datebase bfore test: A B 1 UserKey Name 2 8.51E+21 test 3 4 5  Database after test: A B C 1 UserKey Name 2 8.51E+21 test 3 8.41E+21 test 4	Yes

11. Opening a rendering	User = UserTest Rendering = test	normal	 After open is pressed: 	Rendering selected to open.	Yes

Phase 6 review

What has been done?

Using Tkinter, I have appended the main program file to add a user interface for the saving and opening files and pre-sets. The saving interface is populated with, a button, text, and an entry box, and a check button as per design, while the opening interface only has a button and a list box as per my design. The main button (for saving) is linked to the function which processes the user data and either allows them to save their data or make the user enter new details, all data that is being processed here links to an external database. The other main button (for opening data) is also linked to a function to process the data they selected either allowing them to open it or enter new details.

How has it been tested?

Each time I have added a new part to the saving and opening interface, I executed the program to see if it worked, and if the UI was aligned correctly. The saving and opening functions have also been tested, with both erroneous and normal data (as specified in the testing table), each test I provide what I was testing, the data that I entered, the result and if that result was expected.

How it meets the success criteria and user expectations

The save and open window has a simple design and allows the users to save their calibration options to later be re-opened. It has met the following criteria:

- A functioning account system:
    - The user should be able to access the saved data they have previously made.
        - They should be able to save pre-sets and renderings against their account.
        - They should be able to open pre-sets and renderings against their account.
        - They should only be able to access the data linked to their user key.

Changes in the design that have resulted from this section.

I did not design these windows in my original design section, as I did not anticipate their need. The overall design of my project has required updating as a result of these new editions.

Summary of the whole project at this phase

The main file now contains all user interfaces for the main, registration and login window, it also contains the functions for creating a new user, logging into a user, rendering a video or selection of images, saving and opening files and pre-sets. The program is almost finished as all I have left to do is refine the program.

Phase 7, refining and finalizing the program.

This phase will be for; reviewing, revising and updating of my code: to remove any repeated functions or lines which I can revise to become more optimal, and the creation of functions: which required the creation of the framework / body of the main program before they could be attempted and could not be done in a previous phase.

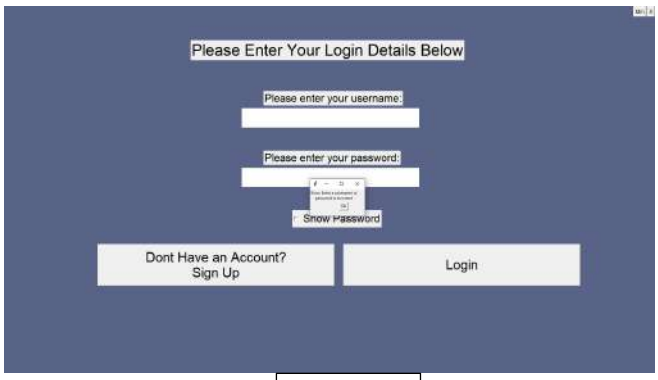
The first issue I have identified that requires updating, is the sub-optimal misuse of lines for each individual pop-up windows. I will remove every instance of an error window and create a class for these windows instead.

```
396 class Errorpopup():
397     def __init__(self, error):
398         ErrorPopup = tk.Toplevel()
399 
400         Message = tk.Label(master = ErrorPopup,
401                            text= error).grid(row = 1, column = 1, columnspan =2 )
402         Button = tk.Button(master = ErrorPopup,
403                            text = "Ok", command = lambda: ErrorPopup.destroy()).grid(row=2, column=2)
404 
```

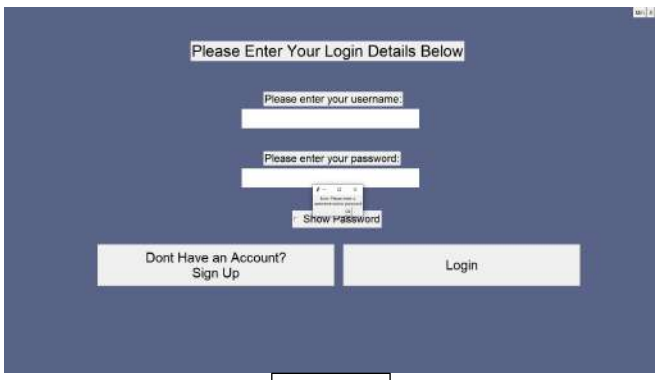
I have created the class Error popup, this when called creates an error window, each error window will still have their own respective label text, this is possible because of the value error we pass through the class.

```
111                     Errorpopup("""Error: Ethier the username or
112 password is incorrect""")
```

This line was taken from the login class, however wherever I have created an error window, I have replaced with this class call. As you can see, when the class is called the text that the error needs to display is passed through with it. All combined this has saved me fifty-six lines of unwanted code, making the code just that much more efficient and less to test / validate its operation. To test this function works correctly, I will re-test entering no data into the login window:



Before



After

As you can see both windows are identical, which show that this was successful as nothing was affected by altering this code.

The final new function I need to create is the method for logging out of the main page.

```
214 class mainwindow(tk.Frame):
215     def __init__(self, parent, controller):
216         tk.Frame.__init__(self, parent, bg = "#596387")
217         self.ImageFile = True
218         self.nextframe = lambda: controller.show_frame(loginwindow)
```

I have created a new variable inside mainwindows initialisation, this variable named nextframe is what will be called as the command to logout of the main window. This command is the same process as what was used in the login window to swap to this window.

```
388
389     def logout(self):
390         logoutWindow = tk.Toplevel()
391 
392         Message = tk.Label(master = logoutWindow,
393                                 text= """Are you sure that you want to log out,
394 all unsaved data will be lost?""").grid(row=1, column=1, columnspan=2)
395         quitButton = tk.Button(master= logoutWindow ,
396                                 text="Yes",
397                                 command = self.nextframe).grid(row=2, column=1)
398         cancelButton = tk.Button(master= logoutWindow,
399                                 text="No",
400                                  command = lambda: logoutWindow.destroy()).grid(row=2, column=2)
401 
```

Next, I defined a new function called logout, this will (when called) open a new popup window with the text “Are you sure you want to log out, all unsaved data will be lost?”, the user then has the choice between clicking the button labelled “No”: which will close only the popup window, or clicking the button labelled “Yes”: which will raise the frame loginwindow to the top.

```
381     def OptionChange(self, event):
382         if self.Fileclicked.get() == "Change File Type":self.Fileclicked.set("File"), self.changeFileType()
383         elif self.Fileclicked.get() == "Save":self.Fileclicked.set("File"), saveWindow([])
384         elif self.Fileclicked.get() == "Open":self.Fileclicked.set("File"), openWindow("rendering")
385         elif self.Presetsclicked.get()  == "Save as Preset":self.Presetsclicked.set("Presets"), self.save()
386         elif self.Presetsclicked.get()  == "Open Preset":self.Presetsclicked.set("Presets"), openWindow("pre-set")
387         elif self.Userclicked.get() == "Log Out": self.Userclicked.set("User"), self.logout()
```

I have appended to the function OptionChange to add the final menu option log out. This option when clicked will run the function logout as discussed previously.

```
252         userdrop = tk.OptionMenu(self,
253                                  self.Userclicked, *Useroptions, command = self.OptionChange)
254         userdrop.config(font = MenuFont)
```

Now I need to add the command OptionChange to the option menu section userdrop.

Phase 7 review

What has been done?

In this section I created the function to allow the user to logout of their account and reviewed / optimised the system as a whole, removing unnecessary code developed during the previous phases.

How has it been tested?

Each time I added a new part or changed an existing section of the main program, I executed my suite of tests, to see if it worked and functioned correctly, as per my design.

Changes in the design that have resulted from this section.

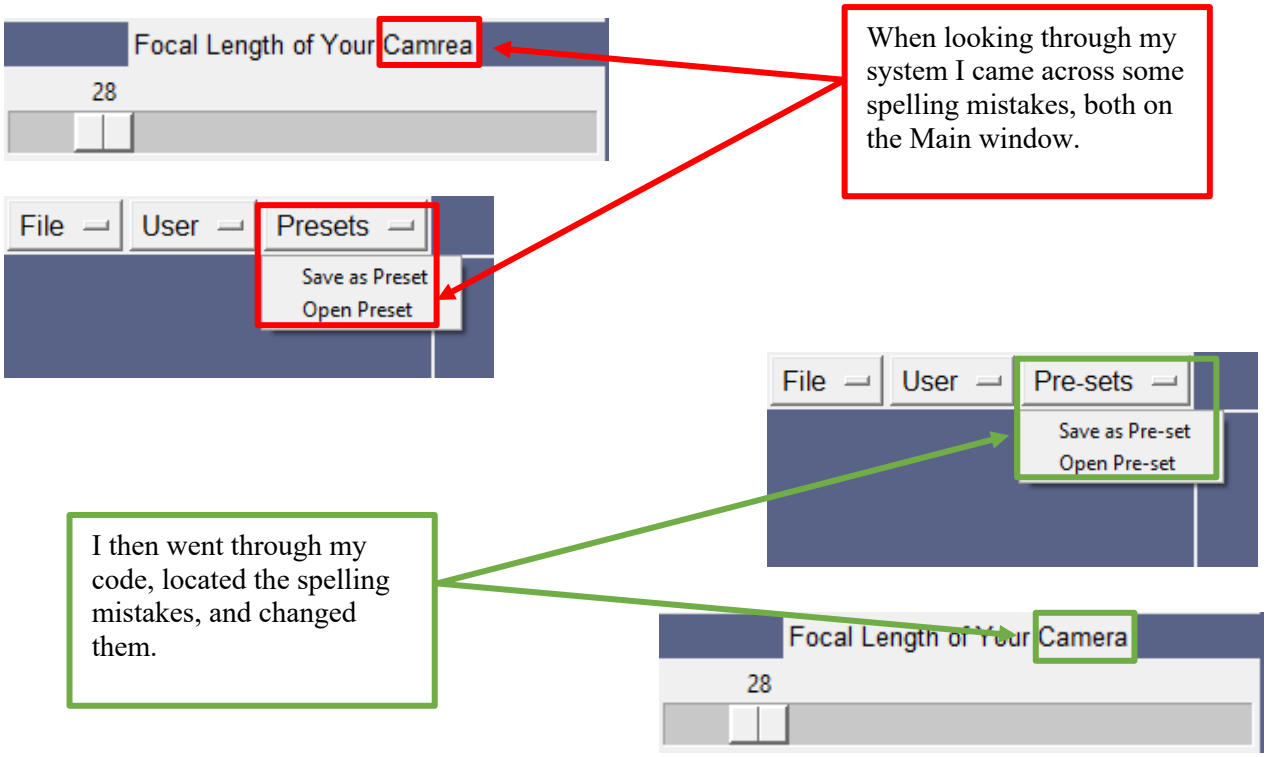
No design changes have occurred from this section.

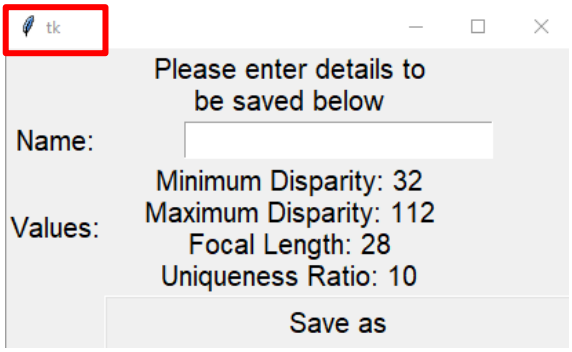
Summary of the whole project at this phase

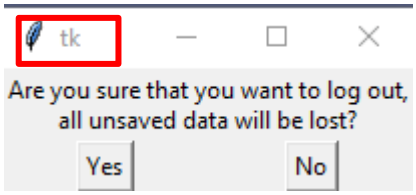
The main file now contains both the user interfaces for the main, registration and login window, it also contains the functions for creating a new user, logging into a user, logging out a user, saving and opening pre-sets and renderings and rendering a video or selection of images. The program is now finished and all that is required is to test the program in its entirety to make sure it meets the criteria I specified in the analysis section.

Phase 8, final testing

In this section, I will review and validate each element of my UI's and inspect for any errors or parts which are not to my design. I will then move on to the last three main stages of testing performed by me: system, acceptance, and usability testing as laid out in my design section.



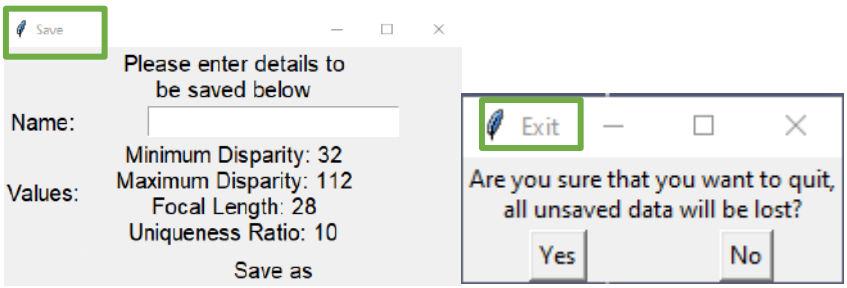




I also saw that all my pop-up windows have the label tk displayed at the top, this to me looks unprofessional and incomplete.

```
596 class saveWindow():
597     def __init__(self, x):
598         Popup = tk.Toplevel()
599         Popup.title("Save")
```

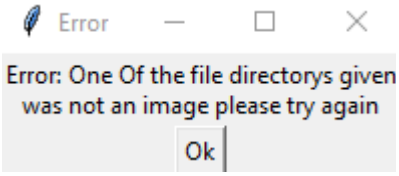
I appended every popup window and added a title which fitted its function ("Save for the save window etc..").



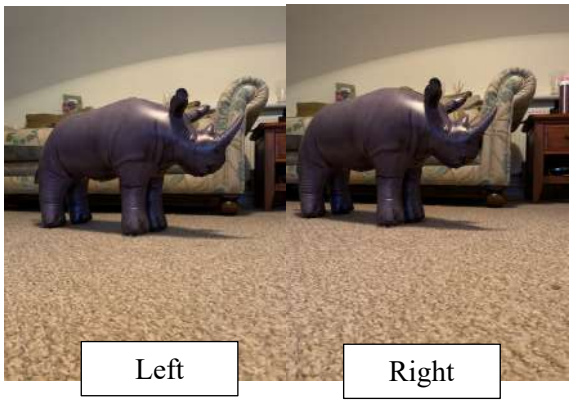
Rendering:

The function which this prototype is built around is the rendering function. This is where the majority of any errors could occur here as it is code developed and integrated primarily by me. To make sure I test this thoroughly, I will improve and add different forms of data input and variable input to remove any errors that might still exist.

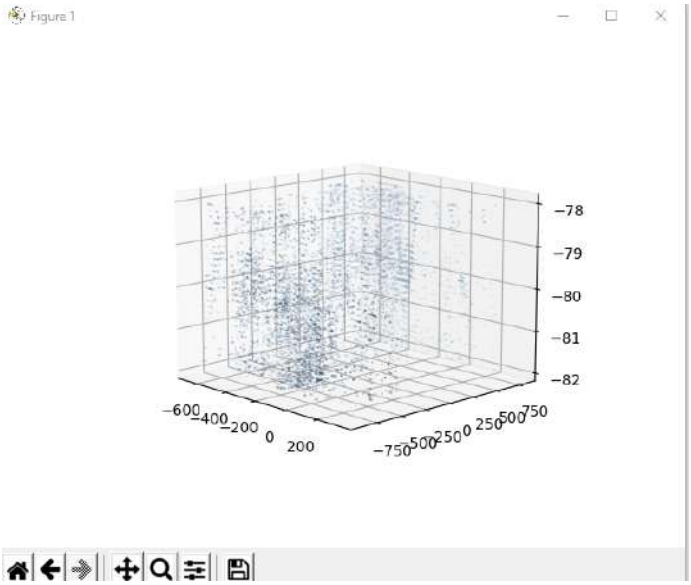
The first test I will perform is selecting data that is not an image or video for both video data input and image data input:



This test came back successful, which means that the system will never try to render things that are not images. The next test will be the rendering of two images; however, the test data will not be the same as the data I have been using throughout the development of the prototype. This is to make sure that the system does not just work with those two images, the object that I used to take an images of is an ornament and will help simulate what a user would have in their own home and possibly use as an object:



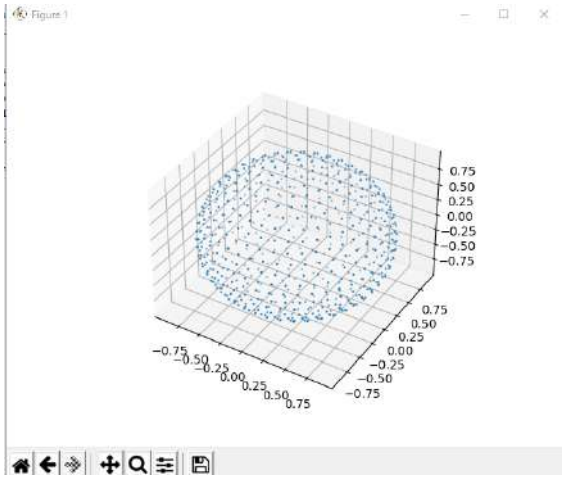
As you can see this test came back successful, because of this I can move on to the next test; this test will be to see the functionality of rendering 3 or more images, for this I will use a new set of visual inputs and pass each through my system:



This result, while not what was wanted, was expected, this is due to the large difference in colours, poor lighting and very few data points. This also does highlight a limitation in my solution; however, this will be mentioned in the limitations section of my evaluation.

The final test for the rendering section will be the rendering of a video. Before I do this however, I will need to stipulate some criteria that I will follow in the accumulation of the video, this set of criteria will be in place to assure that the video I attain will provide the best set of data points for my system. Firstly, I will be using a different camera: this is to provide better quality images than my phone does, I will create a backdrop out of white paper: to eliminate background noise, I will use a smooth object: to eliminate texture problems, and finally I will use proper lightning. While yes this may not replicate what users will do, this test is to showcase what its potential, not what it might do. Here is the testing data which I acquired:

This frame of the video showcases what I took a video of. The object I used was small glass orb, it fits the set criteria of being smooth and a bonus of being a simplistic object. However, it still has a few drawbacks, mainly that it being glass will affect the number of points identified by the system, this is due to the disparity between two images / frames being harder to identify on a transparent object. These drawbacks however do not outweigh the positive and so I passed the video through my program and this was the result:



Due to the restrictions, this rendering came out amazing, while yes, the colours are not apparent, which is likely due to the fact it is glass and there are not too many data points projected on to the graph it still is a perfect proof of concept, this test to me has proven successful and shows the capacity my prototype holds.

Interface inputs:

I performed a test on all the buttons, entry boxes, menu options and sliders inside of my program to make sure that my user interface was performing as it should. I first started with the registration window, the check button: to show the password they are entering works as expected, the register button when pressed with incorrect data performs as expected and finally with correct data it opens the main window as expected. This page also contains three entry boxes, inside the design section I specified how I will be testing these:

Entry Box Input	**Normal/Erroneous**	**Result**
Null (no data).	Erroneous	Error pop-up window created.
Data that has already been taken.	Erroneous	Error pop-up window created.
Passwords that do not match.	Erroneous	Error pop-up window created.
Correct data.	Normal	Account is created and opens main window.

I next preformed the save tests on the login window, this page also contained a check button which works as expected, the login button when pressed with incorrect data performs as expected and finally with correct data it opens the main window as expected. This page also contains two entry boxes, inside the design section I specified how I will be testing these:

Entry Box Input	**Normal/Erroneous**	**Result**
Null (no data).	Erroneous	Error pop-up window created.
Data that does not link to an account.	Erroneous	Error pop-up window created.
Correct data.	Normal	Account is created and opens main window.

Finally, I will check my main window. This page contains; the button for selecting images which works as expected, the button for selecting videos also worked as expected and finally the render button which works as expected. The main window also has a set of sliders which control the renderings created, here is the test I designed labelled out in the design section for them for them:

Slider 1	**Slider 2**	**Slider 3**	**Slider 4**	**Result**
Minimum	Minimum	Minimum	Minimum	The program generates fewer data points.
Maximum	Maximum	Maximum	Maximum	The program generates more data points.
Random Value	Random Value	Random Value	Random Value	The program creates a less realistic rendering.

The main window also contains three drop down menus file, user, and pre-sets. Each one contains its own unique options I will need to test. First off, I will test file, its options are change file type: which works as intended when selected, save: which creates a pop-up window that contains a button for saving when pressed with incorrect data creates an error window and when pressed with correct data saves the file. This pop-up contained an entry box and here is the test for it:

Entry Box Input	**Normal/Erroneous**	**Result**
Null (no data).	Erroneous	Error pop-up window created.
Data that has already been taken.	Erroneous	Error pop-up window created.
Correct data.	Normal	The file is saved.

The option menu file has one last option, open. This when selected opens a new pop-up window which contains both a button and a selection menu. The selection menu works as intended, allowing users to select previously created files, the button also worked as expected allowing users to open a selected file or creating an error pop up if they have not selected a file. The next option menu is user which only contains one option, logout. When selected this option creates a pop-up window asking if they are sure they want to exit, which is what it is meant to do. This pop up contains two buttons yes and no, if "yes" is clicked the user is logged out and if "no" is pressed the pop-up window is closed, as intended. The final option menu is pre-sets, this contains the two last options, save pre-sets and open pre-sets. Both these functions use the same windows as previously discussed, so no further testing is necessary to validate use.

Hardware:

As per my specification this program will need to function on different operating systems, for this test I used both a windows and mac OS's. The test proved that my prototype would work on multiple systems, however the systems in question do have to be quite powerful to process the volumes of image data. I also said in my specification that the user should be able to use any "off the shelf camera" to gather image data. To test this, I used a phone camera and a DSLR (the only two I had at my disposal). This test showed me that while both will create rendering, the DSLR however created better ones, this is due to its higher resolution.

Checklist:

I performed a full system / functionality test. In this test, I utilised the testing table I created inside of my design section rigorously testing every single "Action to test" to establish the operation and seeing if they worked, as per my design. Each item was marked "Yes" or "No" depending on if they have worked or not:

Action to test	**Working?**
The "minimize/maximise" button minimizes or maximises the screen.	**Yes**
The "exit" button closes all windows.	**Yes**
The "show password" check button shows the user their password.	**Yes**
The "login" button checks if the user entered correct details then either logs them in and opens main window or asks them to enter new details.	**Yes**
The "Don't have an account" button opens registration window.	**Yes**
The "Register" button opens UserData and saves the users account information to the database if the username is not already taken.	**Yes**
The menu option "Change file type" changes the data the user can upload from images to video or vice versa.	**Yes**

The menu option “save” saves the rendering that the user just created to the database RenderedData.	**Yes**
The menu option “open” opens a previously saved rendering that the user created.	**Yes**
The menu option “logout” opens the login window	**Yes**
The menu option “Open Pre-set” opens a previously saved set of calibration options.	**Yes**
The menu option “Save Pre-set” saves a set of calibration options to the database PresetData.	**Yes**
Any errors create a pop-up window instead of resulting in a python error.	**Yes**
The “Select your images” button opens the users file directory to select images	**Yes**
The “Select your video” button opens the user file directory to select a video.	**Yes**
If a video is selected it should be separated into frames.	**Yes**
You should be able to render an image from as few as 2 images.	**Yes**
Selecting images displays both the first and last image selected on the canvases.	**Yes**
Selecting a video displays both the first and last frame of the video on the canvases.	**Yes**
The “Render image” button will run the rendering algorithm and return the rendered image.	**Yes**
The program can produce a stereo image/images from the data selected.	**Yes**
The program can produce a point cloud from the stereo images.	**Yes**
The program can combine the point clouds	**Yes**
The program can display the created 3D model/Image.	**Yes**

Stakeholder Testing:

Now that I have tested my system in its entirety, making sure that there are no errors. I now feel confident in passing my prototype onto my stakeholders. I will be providing them all with a copy of my software along with a selection of questions.

Here are the questions I asked:

- How did you find the account system?
- Was the software easy to use?
- What are your thoughts on the rendering of two images?
- What are your thoughts on the rendering of more than two images?
- What are your thoughts on the rendering of a video?
- How accurate were the renderings?

Here were their responses:

How did you find the account system?

Martyn Farrelly:

"I personally liked it. It was very simple and easy to use, like any good registration account system. I found having an account system like yours to be very beneficial, having the option to save both renderings and calibration options was incredibly useful, and I did find myself re-opening saved pre-sets almost every rendering."

Jonathan Ruffles:

"I found it useful being able to save and open personalised pre-sets, it was also great that I could open renderings I've already created, as it does take quite a while to render multiple images."

James Heely:

"Both registering and logging in were very easy to do. I especially liked how you could save pre-sets to your account; this is because I found a configuration which worked great with my camera and it would've been a problematic to have to redo it every time."

Was the software easy to use?

Martyn Farrelly:

"It was easy for me; however I could see how some could get confused. There isn't much on the side of additional help but if you know about renderings it is all very self-explanatory. The layout of the system was also very simplistic which does help, in comparison to others which are overly crowded."

Jonathan Ruffles:

"I did find it particularly easy to use, there wasn't any hidden features that need prior knowledge to figure out, which did make it very simple but easy to use."

James Heely:

"I didn't find any of it hard to grasp, however it did take me a while to find how to render a video as I couldn't find where the option was located. But other than that, the system was very easy to use and easy to understand."

What are your thoughts on the rendering of two images?

Martyn Farrelly:

"I was very surprised on how well it rendered two images. Normally it would take a lot more images to gain proper information on an object, however your system seems to accomplish it with ease. This is not saying that the renderings were amazing however were a lot better than expected."

Jonathan Ruffles:

"They were great! I didn't expect there to be such a complete image however I was wrong. With other software I have used they struggle to render a model from two images however yours didn't seem to struggle. The main concern I have however is when you rotate the image there is a lot of blank space in between the layers."

James Heely:

“I found them quite impressive. The overall quality of the rendering from just two images was better than I expected, however the renderings were still looked incomplete, but they were recognisable. I also liked how quickly the system took to create the rendering of these images.”

What are your thoughts on the rendering of more than two images?

Martyn Farrelly:

“This is the only fault I can see in the program. The renderings were not only incomplete but took very long to render, while I only used about 10 different images, I thought due to the accuracy of the 2-image rendering it would have been even better.”

Jonathan Ruffles:

“They were quite hit and miss, I tried both a lot and very few images and the difference between both was alarming. The more images I inputted the better the rendering was, but the less and less I inputted the more unrealistic and incomplete it got. I also noticed that the renderings took a long time to complete when I started adding more and more images.”

James Heely:

“I found them quite lacklustre, the renderings the program produced were not completely satisfactory the main problem being how incomplete they were. The renderings seemed worse than the renderings of just two images, however the way the new models were displayed I did prefer to how they were prior.”

What are your thoughts on the rendering of a video?

Martyn Farrelly:

“Just like the two image renderings it was quite amazing, the model the program created was incredibly realistic and in comparison, to the multiple image rendering was a massive step up, however I did also find that it took quite a while again for the program to create the renderings, but overall it was very good.”

Jonathan Ruffles:

“The renderings they produced were great. They were very life-like, however some videos created better renderings than others. The renderings that did turn out well and would be something I would use in a VR game due to how realistic they were.”

James Heely:

“I absolutely loved these, the renderings it produced were amazing, especially in regard to how the system rendered multiple images. The program did take a while to render the model and there was a bit of lagged response when I tried to move the model, but other than that they were perfect.”

How accurate were the renderings?

Martyn Farrelly:

“Two out of the three methods of rendering a model were brilliant, however as I said before the multiple image renderings were sub optimal. However, I do believe on the whole the renderings were fairly accurate.”

Jonathan Ruffles:

“As I said earlier, they were quite hit and miss, the rendering of two images was great, the rendering of more than two had its own set of problems being good and bad depending on how many images you inputted and finally the rendering of a video was amazing, however on the whole they were very good.”

James Heely:

“The only renderings I had a problem with were the models generated by multiple images, however the rest were amazing. The accuracy was quite impressive especially the rendering from a video which created an almost perfect model of what I wanted. The rendering from two images was also amazing being a lot more precise than I would ever of thought, overall, I thought they were brilliant.”

Synopsis of stakeholder testing:

The stakeholders all were able to use the program, and all seemed to be content with its functionality and options. They all mentioned how the simplistic design was useful and easy to navigate which was part of my specification. They also seemed to like functionality of prototype, especially as mentioned the ability to open and save both renderings and pre-sets. They all also all seemed to like how accurate the renderings were for both two image and video rendering, which was again part of my specification.

However, there was also some problems and concerns raised. Firstly, all the stakeholders had at least some problem in generating accurate models with multiple images, this was concern was both identified and explained inside of the final testing phase. A solution to this might be to implement a new form of rendering. The stakeholders also raised a concern for the length of time it took to render videos and multiple images. This is from how much data processing my program must go through to allocate a x, y, and z coordinate to a pixel. A solution to this could be to tighten the restrictions on what pixels get assigned values, however this would also mean less complete picture and so without testing the trade-off cannot be assessed. Finally, Martyn Farrelly, my help / guidance system stakeholder raised the concern about guidance and help throughout the system, however I do not believe this to be a problem. This is due to the target audience being people already accustomed to rendering models. This belief is backed up by all three of my stakeholders (who are a representatives of the target audience) all understood and grasped how to use my application.

Evaluation

Criteria Met

Criteria	Has it been met?	Pages where criteria were met
Clear front end		
The main user interface should provide the following features:		
Select Images, Select Video, Render Image, close program and minimize/maximize program buttons.	Yes	73, 75, 76
Calibration options: minimum possible disparity, maximum possible disparity, uniqueness ratio and focal length.	Yes	73
Display the images/frames the user has chosen to enter.	Yes	78, 82
Menu options to save and open changes in the program's calibration, save and open renderings and log out.	Yes	73
The registration window should provide the following features:		
An entry box for; their username, password, and a check password.	Yes	95
A button for; registering their data and showing their password.	Yes	95
The login window should provide the following features:		
An entry box for; their username, password.	Yes	103
A button for; logging in, creating an account, and for showing their password.	Yes	103
All windows must have large buttons and clear text.	Yes	73, 95, 103
A functioning account system		
A registration window which allows the user to enter their details to create an account.	Yes	95
The details need to be saved to an external database, with a unique user key.	Yes	99
A login window which allows the user to login to their account.	Yes	106-107

The user should be able to access the saved data they have previously made.		
They should be able to save pre-sets and renderings against their account.	Yes	120, 122
They should be able to open pre-sets and renderings against their account.	Yes	121-122, 124
They should only be able to access the data linked to their user key	Yes	121, 123
A non-technical calibration.		
The calibration needs to be appropriate for new users.		
Each option needs to be clear on the main page.	Yes	73
They all need text which describe their function.	Yes	73
This text needs to be clear.	Yes	73
Only a few options to eliminate the chance of confusion.	Yes	73
The user should be able to calibrate:		
Minimum possible disparity	Yes	86, 132
Maximum possible disparity	Yes	86, 132
Uniqueness ratio	Yes	86, 132
Focal length	Yes	86, 132
Render models.		
The program should be able to render an image from two still photos.	Yes	64, 65, 69, 74, 90
The program should be able to take images as input data.	Yes	86
The program should be able to render an image from more than two images.	Yes	91
The program should be able to render an image from a video.	Yes	132
The program should be able to take a video as input data.	Yes	81-82
The program should be able to segment this video into its frames.	Yes	81
From the input data the program should be able to create:		
A stereo image that also is influenced by the calibrations.	Yes	56-57
The x y z r g b values of each pixel.	Yes	61, 88
A point cloud made from these values.	Yes	90, 91, 130, 131, 132

The renderings created must be clear and recognisable.		
The stakeholders have to say that the renderings created are recognisable.	**No**	
The program needs to be easy to use.		
Each stakeholder needs to not have a problem with how to use the software.	Yes	136-138

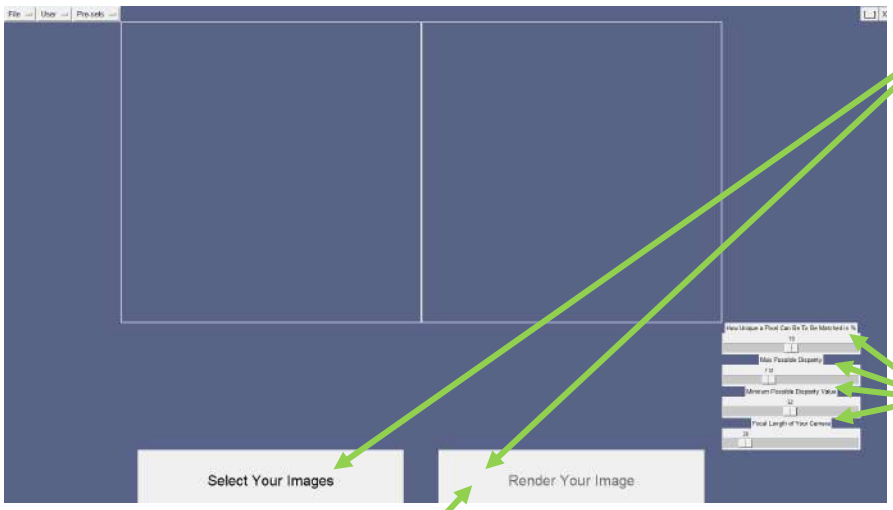
Criteria not met:

The rendered image will have to be clear and recognisable:

This was the only criteria point I did not fulfil, while I did receive positive feedback on two out of three methods of rendering for the accuracy and realistic nature of them, the second way to render: with more than two images did not receive any positive feedback. While I do believe that it would not have been wrong to say that I accomplished this task, due to almost sixty-seven percent of all renders done will be “clear and recognisable”, however I would only have counted this as successful if all renders would have been “clear and recognisable” not just two thirds. The only solution to be able to overcome this problem, is to in a later version of the software look for new ways to create better more precise point clouds for low data input like using ICP (Iterative closest point) in conjunction with the two-image rendering system I already have, in order to help create better models. However due to my initial plan for rendering and not knowing the constraints I would run into, I did not have time to properly research and develop a method including ICP.

Usability Features

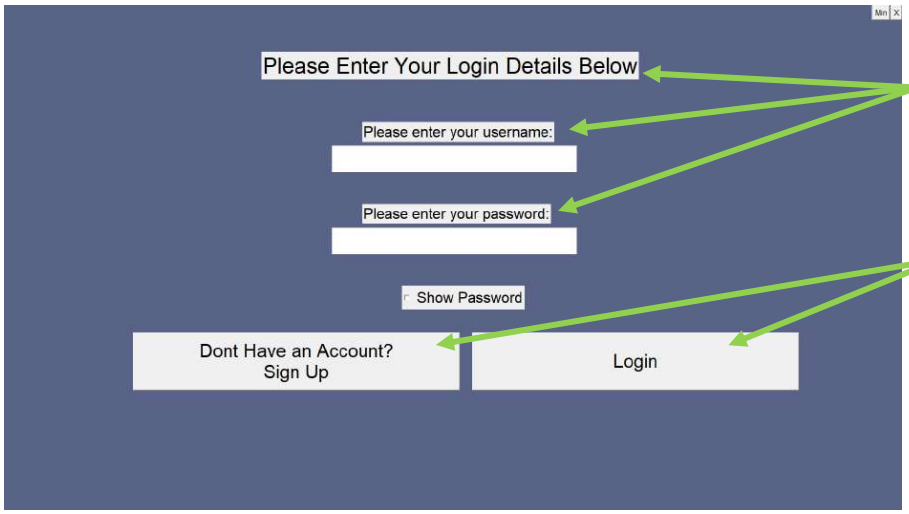
In my design section (specifically inside the user experience features sub-section) I specified a set of criteria to help create a better users experience, here is how I fulfilled them:



Both buttons are large and have clear text, to both help eliminate miss clicks and make the overall use of the software easier.

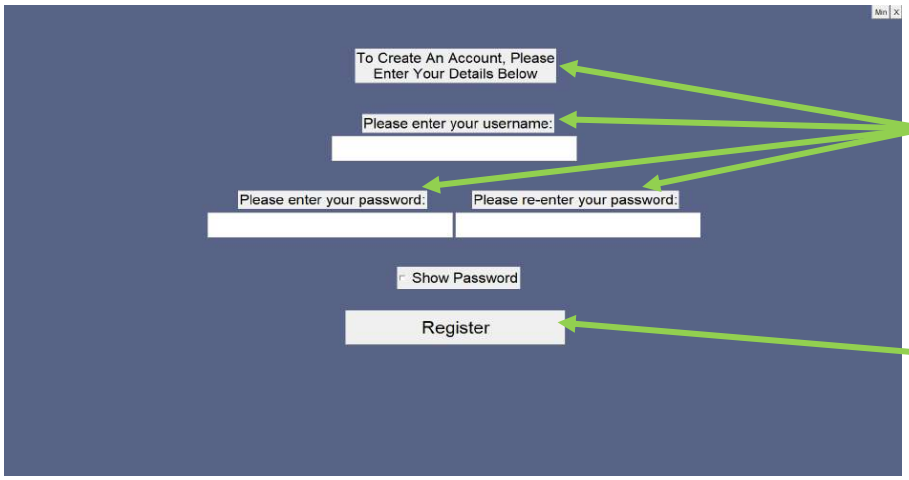
All the text is large enough to be apparent to even people with visual impairment. It may look small here but, on the prototype, but it is not.

The button is disabled until the user has selected data to be rendered this is to make sure that the user does not try to render without any data and create errors.



All the text is large enough to be apparent to even people with visual impairment. It may look small here but, on the prototype, but it is not.

Both buttons are large and have clear text, to both help eliminate miss clicks and make the overall use of the software easier.



All the text is large enough to be apparent to even people with visual impairment. It may look small here but, on the prototype, but it is not.

The button is large and has clear text, to both help eliminate miss clicks and make the overall use of the software easier.

Limitations

There are two main types of limitations with my solution that I have encountered, software and hardware. Both having their own unique limitations on my solution. First off, I will discuss the hardware limitations:

The main hardware limitation I ran into is the computers processing power. Each time an image is processed through my system each pixel is looked at and its x, y, and z values are determined. This alone is a lot of work for a processor to handle. For example, when a thirty second video (running at sixty frames per second) is uploaded from a phone to be rendered, that would mean my program would have to look at 1.1x10^10 pixels. This for any system is a lot of processing time.

The next hardware limitation is the cameras itself. Rendering images which are skewed from each other are more difficult to properly analyse, and so create sub optimal renderings. This is because working out the disparity between two images which do not align well, will create false positives, in terms of pixel matching.

The next set of limitations I will discuss are the software limitations:

The main software limitation is the inability to create realistic renderings of two or more images (excluding videos), this is the main limitation because the main function of the program was to create renderings. The fact that this cannot be accomplished one hundred percent of the time is a significant drawback, this was all identified in both my final testing and the stakeholders testing.

The next software limitation I have identified is the ability to export the files to another system for further use, this while not identified by my stakeholders, I believe it is extremely important. The main use of this prototype was to be used by my stakeholders (e.g. VR developers) and without the ability to save these rendering to more file types than just a PLY file, there is a chance they will not be able to use their renderings.

How to overcome these limitations:

To overcome the main hardware limitation, in later versions of my software I can create harder restrictions on what needs to be processed, this will undoubtedly lead to worse quality in renderings, however until further investigation is made into this topic I cannot say for certain that it would.

The next hardware limitation could be overcome by creating a stereoscopic camera. In the development of my idea, I did research into creating my own stereoscopic camera, however that endeavour was quickly swept aside after I discovered OpenCV, but a stereoscopic camera would eliminate the worry of the changes in axis, as no matter what rotation or pitch the images will be taken from equal distances apart and at the same angle.

To overcome the main software limitation, I would have needed to have more development time. The methods that I have used already can work in conjuncture to ICP (Iterative Closest Point). ICP is the method of identifying two closest points in a point cloud and using them to align the clouds together, different to the x, y, z method I have used. By utilising ICP it would be extremely more difficult, it would also provide seamless renderings. To over-come the last limitation, I would need to research the different software's in use by my stakeholder group and gather information on what file types they require. If necessary, I would then need to possibly create a converter to change the file types I can create to the files they would need. This would require a lot of validation and testing.

In conclusion if I had more development time, I believe in later versions all the limitations could be overcome. However even in light of these limitations, I consider the whole investigation a success with not only reaching 95% of my goals but creating a working prototype for "The Creation of 3 Dimensional Models From Image Sequencing".

Final Code Listing:

```
import tkinter as tk
import tkinter.font as font
from tkinter import filedialog
from PIL import ImageTk, Image
import numpy as np
import cv2
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import csv


class windows(tk.Tk):

    def __init__(self):
        tk.Tk.__init__(self)
        container = tk.Frame(self)

        container.pack(side = "top", fill = "both", expand = False )
        container.grid_rowconfigure(0, weight = 1)
        container.grid_columnconfigure(0, weight = 1)

        self.frames = {}

        for i in (signinwindow, loginwindow, mainwindow):
            frame = i(container, self)

            self.frames[i] = frame

            frame.grid(row = 16, column = 16, sticky = "news")

        self.show_frame(loginwindow)

        global maxScreen
        global key
        key = None
        maxScreen = True


    def show_frame(self,cont):
        frame = self.frames[cont]
        frame.tkraise()


#Login Window-----------------------------------------------------------------------------------------------------------------------------------------------------------
class loginwindow(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent, bg = "#848387")
        self.checkVar = tk.IntVar()
        self.nextframe = lambda: controller.show_frame(mainwindow)
        ButtonFont = font.Font(size = 30, family="Helvetica")
        LabelFont = font.Font(size = 24, family = "Helvetica")
        EntryFont = font.Font(size = 35, family = "Helvetica")
        #Buttons-----------------------------------
        self.minmaxbutton = tk.Button(self,
                              command = minmax_frame, text = "Min", font = (font.Font(size = 12, family="Helvetica")))
        self.minmaxbutton.place(x = 1885, y = 0)

        self.exitbutton = tk.Button(self,
                              command = quitWindow, text = "X",  font = (font.Font(size = 12, family="Helvetica"))).place(x =1895 , y = 0)
        self.loginbutton = tk.Button(self,
                              text = "Login", font = ButtonFont, width = 30, height = 2, command = self.login)
        self.loginbutton.place(x =1000 , y =700 )
        self.newuserbutton = tk.Button(self,
                              text = """Dont Have an Account?
Sign Up""", font = ButtonFont , width = 30, command = lambda: controller.show_frame(signinwindow))
        self.newuserbutton.place(x =975 , y =700 )
        #Entry Boxs-------------------------------------
        self.usernamebox = tk.Entry(self,
                              font = EntryFont)
        self.usernamebox.place(x = 700, y = 300)
        self.passwordbox = tk.Entry(self,
                              show = "*", font = EntryFont)
        self.passwordbox.place(x = 700, y = 475)
        #Labels-----------------------------------------
        infomationLabel = tk.Label(self,
                              text = """Please Enter Your Login Details Below""", font = font.Font(size = 35, family = "Helvetica")).place(x = 550, y = 100)
        usernameLabel = tk.Label(self,
                              text = "Please enter your username:" , font = LabelFont).place(x = 765, y = 250)
        passwordLabel = tk.Label(self,
                              text = "Please enter your password:" , font = LabelFont).place(x = 765, y = 425)
        #Checkbox---------------------------------------
        self.showpassBox = tk.Checkbutton(self,
                                  text = "Show Password",variable = self.checkVar, command = self.showpassword,  font = LabelFont)
        self.showpassBox.place(x = 850, y = 600)

    def showpassword(self):
        if self.checkVar.get() == 1:
            self.passwordbox.config(show = "")
        else:
            self.passwordbox.config(show = "*")

    def login(self):
        if len(self.passwordbox.get()) > 0 and len(self.usernamebox.get()) > 0 :
            match = False
            try:
                with open("UserData.csv", "r") as file:
                    fieldnames = ["Username", "Password", "UserKey"]
                    reader = csv.DictReader(file, fieldnames = fieldnames)
                    for row in reader:
                        if row["Username"] == self.usernamebox.get():
                            match = True
                            if row["Password"] == self.passwordbox.get():
                                global key
                                key = ("".join(str(chr(i)) for i in self.usernamebox.get()))
                                Popup = tk.Toplevel()

                                Message = tk.Label(master=Popup,
                                                   text= """Your logged in,
press ok to proceed""").grid(row=1, column=1, columnspan=2)
                                Button = tk.Button(master= Popup,
                                                   text="Ok",
                                                   command =self.nextframe).grid(row=2, column=2)
                            else:
                                arg
                    if match == False:
                        arg
            except:
                Errorpopup("""Error: Either the username or
password is incorrect""")
            else:
                Errorpopup("""Error: Please enter a
```

```
username and or password""")


#Registration Window
class signinwindow(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent, bg = "#596387")
        self.checkVar = tk.IntVar()
        self.nextframe = lambda: controller.show_frame(mainwindow)
        ButtonFont = font.Font(size = 30, family="Helvetica")
        LabelFont = font.Font(size = 24, family = "Helvetica")
        EntryFont = font.Font(size = 35, family = "Helvetica")
        #Buttons-----------------------------------------
        self.minmaxbutton = tk.Button(self,
                                      command = minmax_frame, text = "Min", font = (font.Font(size = 12, family="Helvetica")))
        self.minmaxbutton.place(x = 1855, y = 0)

        self.exitbutton = tk.Button(self,
                                    command = quitWindow, text = "X",  font = (font.Font(size = 12, family="Helvetica"))).place(x =1885 , y = 0)
        self.signupbutton = tk.Button(self,
                                      text = "Register", font = ButtonFont, width = 20, command = self.register)
        self.signupbutton.place(x =730 , y =700 )
        #Entry Boxs--------------------------------------------
        self.usernameBox = tk.Entry(self,
                                    font = EntryFont)
        self.usernameBox.place(x = 700, y = 300)
        self.passwordBox = tk.Entry(self,
                                    show = "*", font = EntryFont)
        self.passwordBox.place(x = 965, y = 475)
        self.checkpasswordBox = tk.Entry(self,
                                         show = "*", font = EntryFont)
        self.checkpasswordBox.place(x = 435, y = 475)
        #Labels------------------------------------------------
        infomationLabel = tk.Label(self,
                                   text = """To Create An Account, Please
Enter Your Details Below""", font = LabelFont).place(x = 750, y = 100)
        usernameLabel = tk.Label(self,
                                 text = "Please enter your username:" , font = LabelFont).place(x = 745, y = 250)
        passwordLabel = tk.Label(self,
                                 text = "Please enter your password:" , font = LabelFont).place(x = 300, y = 425)
        repasswordLabel = tk.Label(self,
                                   text = "Please re-enter your password:" , font = LabelFont).place(x = 1000, y = 425)
        #CheckBox----------------------------------------------
        self.showpassBox = tk.Checkbutton(self,
                                          text = "Show Password",variable = self.checkVar, command = self.showpassword, font = LabelFont)
        self.showpassBox.place(x = 840, y = 600)

    def showpassword(self):
        if self.checkVar.get() == 1:
            self.passwordBox.config(show = "")
            self.checkpasswordBox.config(show = "")
        else:
            self.passwordBox.config(show = "*")
            self.checkpasswordBox.config(show = "*")

    def register(self):
        if self.passwordBox.get() == self.checkpasswordBox.get() and len(self.passwordBox.get()) > 0 and len(self.usernameBox.get()) :
            try:
                with open("UserData.csv", "r") as file:
                    fieldnames = ["Username", "Password", "UserKey"]
                    reader = csv.DictReader(file, fieldnames = fieldnames)
                    for row in reader:
                        if row["Username"] == self.usernameBox.get():
                            raise
                    with open("UserData.csv", "a") as file:
                        writer = csv.DictWriter(file, fieldnames = fieldnames)
                        writer.writerow({"Username": "%s" % self.usernameBox.get()
                                         ,"Password": "%s" % self.passwordBox.get()
                                         , "UserKey": "%s" % ("".join(str(ord(i)) for i in self.usernameBox.get())) })
                        global kwy
                        key = ("".join(str(ord(i)) for i in self.usernameBox.get()))
                    Popup = tk.Toplevel()

                    Message = tk.Label(master=Popup,
                                       text= """Your account has been created,
press ok to proceed""").grid(row=1, column=1, columnspan=2)
                    Button = tk.Button(master= Popup,
                                       text="Ok",
                                   command =self.nextframe().grid(row=2, column=2)


            except:
                Errorpopup("""Error: That Username has
already been taken""")
        else:
            Errorpopup("""Error: Please check you have both a
username and matching passwords""")


#Main Window-------------------------------------------------------------------------------------------------------------------------------------
class mainwindow(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent, bg = "#596387")
        self.ImageFile = True
        self.nextframe = lambda: controller.show_frame(loginwindow)
        #Fonts-------------------
        ButtonFont = font.Font(size = 24, family="Helvetica")
        MenuFont = font.Font(size = 12, family="Helvetica")
        LabelFont = font.Font(size = 10, family = "Helvetica")
        #Buttons-----------
        self.minmaxbutton = tk.Button(self,
                                      command = minmax_frame, text = "(__)", font = MenuFont)
        self.minmaxbutton.grid(row = 0, column= 12)

        self.exitbutton = tk.Button(self,
                                    command = quitWindow, text = "X",  font = MenuFont).grid(row = 0, column = 14)
        self.RenderImageBtn = tk.Button(self,
                                        text="Render Your Image", command = self.RenderImage, state = "disabled", font = ButtonFont, height = 3, width = 30 )
        self.RenderImageBtn.grid(row = 13, column = 6 )
        self.SelectImageBtn = tk.Button(self,
                                        command = self.imageSelector, text = "Select Your Images", font = ButtonFont, height = 3, width = 30 )
        self.SelectImageBtn.grid(row = 13, column = 5)
        #Dropdown selections-----------
        Fileoptions = ["Change File Type", "Save", "Open"]
        Useroptions = ["Log Out"]
        Presetsoptions = ["Save as Pre-set", "Open Pre-set"]

        self.Fileclicked = tk.StringVar()
        self.Userclicked = tk.StringVar()
        self.Presetsclicked = tk.StringVar()

        self.Fileclicked.set("File")
        self.Userclicked.set("User")
        self.Presetsclicked.set("Pre-sets")

        self.filedrop = tk.OptionMenu(self,
                                      self.Fileclicked, *Fileoptions, command = self.OptionChange)
        self.filedrop.config(font = MenuFont)
        userdrop = tk.OptionMenu(self,
                                 self.Userclicked, *Useroptions, command = self.OptionChange)
        userdrop.config(font = MenuFont)
        presetsdrop = tk.OptionMenu(self,
                                    self.Presetsclicked, *Presetsoptions, command = self.OptionChange)
        presetsdrop.config(font = MenuFont)

        self.filedrop.grid(row = 0, column= 0)
        userdrop.grid(row = 0, column= 2)
        presetsdrop.grid(row = 0, column= 4)
        #Canvas------------------
        self.LCanvas = tk.Canvas(self,
                                 bg = "#596387", height = 650, width = 650)
        self.LCanvas.grid(row = 1, column = 5 )
        self.RCanvas = tk.Canvas(self,
                                 bg = "#596387", height = 650, width = 650)
        self.RCanvas.grid(row = 1, column = 6)
        #Sliders-----------------
        self.minDispSlider = tk.Scale(self,
                                      from_=1, to = 64, orient = "horizontal", length = 295, width = 20)
        self.minDispSlider.set(32)
        self.minDispSlider.grid(row = 10, column = 10)
        self.maxDispSlider = tk.Scale(self,
                                      from_=16, to = 320, orient = "horizontal", length = 295, width = 20)
        self.maxDispSlider.set(112)
        self.maxDispSlider.grid(row = 8, column = 10)
        self.focalSlider = tk.Scale(self,
                                    from_=28, to = 100, orient = "horizontal", length = 295, width = 20)
        self.focalSlider.set(28)
        self.focalSlider.grid(row = 12, column = 10)
        self.uniqueRatioSlider = tk.Scale(self,
                                          from_=1, to = 15, orient = "horizontal", length = 295, width = 20)
        self.uniqueRatioSlider.set(10)
        self.uniqueRatioSlider.grid(row = 6, column = 10)

        global uniqueRatioSlider, maxDispSlider, minDispSlider, focalSlider
        uniqueRatioSlider = self.uniqueRatioSlider
        maxDispSlider = self.maxDispSlider
        minDispSlider = self.minDispSlider
        focalSlider = self.focalSlider
        #Labels------------------
        self.minDispLabel = tk.Label(self,
                                     text = "Minimum Possible Disparity Value", font = LabelFont).grid(row = 9, column = 10)
        self.maxDispLabel = tk.Label(self,
                                     text = "Max Possible Disparity", font = LabelFont).grid(row = 7, column = 10)
        self.focalLabel = tk.Label(self,
                                   text = "Focal Length of Your Camera", font = LabelFont).grid(row = 11, column = 10)
        self.uniqueRatioLabel = tk.Label(self,
                                         text = "How Unique a Pixel Can Be To Be Matched in %", font = LabelFont).grid(row = 5, column = 10)

    def imageSelector(self):
        self.imagePathStr = []
        imagePath = filedialog.askopenfilename()
        while len(imagePath) > 0 or len(self.imagePathStr) < 2:
```

```
            self.imagePathStr.append(imagePath)
            imagePath = filedialog.askopenfilename()
        length = len(self.imagePathStr) - 1)
        try:
            while length >= 0:
                imageCheck = Image.open((self.imagePathStr[length]))
                length -= 1
            pathL = Image.open((self.imagePathStr[0]))
            pathR = Image.open(self.imagePathStr[(len(self.imagePathStr)-1)])

            NewPathR = pathR.resize((650,650))
            NewPathL = pathL.resize((650,650))

            self.RCanvas.image = ImageTk.PhotoImage(NewPathR)
            self.RCanvas.create_image(0,0, image = self.RCanvas.image, anchor = "nw")

            self.LCanvas.image = ImageTk.PhotoImage(NewPathL)
            self.LCanvas.create_image(0,0, image = self.LCanvas.image, anchor = "nw")

            self.RenderImageBtn.configure(state = "active", bg = "light green")
        except:
            Errorpopup("""Error: One Of the file directorys given
was not an image please try again""")


    def videoSelector(self):
        self.video = filedialog.askopenfilename()
        if len(self.video) > 0:
            try:
                self.imagePathStr = []
                self.createframes(self.video)

                pathL = Image.open((self.imagePathStr[0]))
                pathR = Image.open(self.imagePathStr[(len(self.imagePathStr)-1)])

                NewPathR = pathR.resize((650,650))
                NewPathL = pathL.resize((650,650))

                self.RCanvas.image = ImageTk.PhotoImage(NewPathR)
                self.RCanvas.create_image(0,0, image = self.RCanvas.image, anchor = "nw")

                self.LCanvas.image = ImageTk.PhotoImage(NewPathL)
                self.LCanvas.create_image(0,0, image = self.LCanvas.image, anchor = "nw")

                self.RenderImageBtn.configure(state = "active", bg = "light green")
                print (self.imagePathStr)
            except:
                Errorpopup("""Error: The file directory given
was not a video please try again""")

    def createframes(self, video):
        Video = cv2.VideoCapture(self.video)
        maxFrames = int(Video.get(cv2.CAP_PROP_FRAME_COUNT))
        success, image = Video.read()
        count = 0

        while success:
            image = cv2.resize(image,(650,650))
            cv2.imwrite("images/frame%d.jpg" % count, image)
            self.imagePathStr.append("images/frame%d.jpg" % count)
            count += 1
            success, image = Video.read()


    def changeFileType(self):
        if self.ImageFile == True:
            self.ImageFile = False
            self.SelectImageBtn.config(command = self.videoSelector,
                                       text = "Select Your Video")
        else:
            self.ImageFile = True
            self.SelectImageBtn.config(command = self.imageSelector,
                                       text = "Select Your Images")


    def OptionChange(self, event):
        if self.FileClicked.get() == "Change File Type":self.FileClicked.set("File"), self.changeFileType()
        elif self.FileClicked.get() == "Save":self.FileClicked.set("File"), saveWindow([])
        elif self.FileClicked.get() == "Open":self.FileClicked.set("File"), openWindow("rendering")
        elif self.PresetsClicked.get()  == "Save as Pre-set":self.PresetsClicked.set("Pre-sets"), self.save()
        elif self.PresetsClicked.get()  == "Open Pre-set":self.PresetsClicked.set("Pre-sets"), openWindow("pre-set")
        elif self.UserClicked.get() == "Log Out": self.UserClicked.set("User"), self.logout()

    def logout(self):
        logoutWindow = tk.Toplevel()

        Message = tk.Label(master = logoutWindow,
                           text= """Are you sure that you want to log out.
all unsaved data will be lost?""").grid(row=1, column=1, columnspan=2)
        quitButton = tk.Button(master= logoutWindow ,
                               text="Yes",
                               command = self.controller().grid(row=2, column=1)
        cancelButton = tk.Button(master= logoutWindow,
                               text="No",
                               command = lambda: logoutWindow.destroy()).grid(row=2, column=2)


    def save(self):
        x = [self.minDispSlider.get(), self.maxDispSlider.get(), self.focalSlider.get(), self.uniqueRatioSlider.get()]
        saveWindow(x)

    def RenderImage(self):
        images = self.imagePathStr
        mindisp = self.minDispSlider.get()
        maxValue = self.maxDispSlider.get()
        focal_length = self.focalSlider.get()
        uniquenessRatio = self.uniqueRatioSlider.get()
        RenderedimageData(images, focal_length, maxValue, mindisp, uniquenessRatio)

#Rendering method-------------------------------------------------------------------------------------------------------------------------------------------------------
class RenderedimageData():
    def __init__(self, images, f, maxValue, mindisp, uniquenessRatio):
        homogenized = np.array([[1,0,0],
                                [0,1,0],
                                [0,0,-f]])

        distortion = np.array([[0.],
                               [0.],
                               [0.],
                               [0.]])
        if len(images) > 2:
            totalImages = (len(images) - 1)
            currentLeftImage = 0
            while totalImages > currentLeftImage:
                pathL = images[currentLeftImage]
                pathR = images[(currentLeftImage + 1)]
                left_image = cv2.imread(pathL)
                right_image = cv2.imread(pathR)
        #Image-----------------------------------------------------------------------
                height, width, _ = left_image.shape
                RGB = right_image.reshape(-1,3)
        #Matricies-------------------------------------------------------------------
                camreaMatrix = np.array([[f,0,width/2],
                                         [0,f,height/2],
                                         [0,0,1]])
                transformation = np.float32([[1,0,0,-width/2],
                                             [0,1,0,-height/2],
                                             [0,0,0,-f],
                                             [0,0,-1/100,0]])

                camreaMatrix = self.OptimalMatrix(camreaMatrix, distortion, right_image)

                rVector, tVector = self.ri_Vectors(homogenized, camreaMatrix)

                stereo = self.depthMap(left_image, right_image)

                points = self.pointsTo3D(stereo, transformation)

                deleteNoise = ((stereo.reshape(-1)) > (stereo.reshape(-1)).min())
                points = points[deleteNoise]
                RGB = RGB[deleteNoise]

                c = [ [i] for i in points]
                d = [ [i] for i in RGB]
                a = 0
                largeArray = []
                for i in points:
                    pointsArray = np.array(c[a])

                    colourArray = np.array(d[a])

                    mainArray = np.hstack((pointsArray, colourArray))
                    if a == 0:
                        largeArray = mainArray
                    else:
                        largeArray = np.vstack((largeArray, mainArray))
                    a +=1

            points = largeArray[abs(largeArray[:,2]-(np.mean(largeArray,axis=0)[2]))<2]

            axis, RGB = points[:,:3], points[:,3:]
            project = plt.axes(projection='3d')

            project.scatter(axis[:,0], axis[:,1], axis[:,2], c = RGB/255, s = 0.01)
            plt.show()

        else:
            pathL = images[0]
            pathR = images[1]
            left_image = cv2.imread(pathL)
            right_image = cv2.imread(pathR)
            height, width, _ = left_image.shape
            RGB = right_image.reshape(-1,3)
```

```
            camreaMatrix = np.array([[f,0,width/2],
                                     [0,f,height/2],
                                     [0,0,1]])
            transformation = np.float32([[1,0,0,-width/2],
                         [0,1,0,-height/2],
                         [0,0,0,-f],
                         [0,0,-1/100,0]])

            camreaMatrix = self.OptimalMatrix(camreaMatrix, distortion, right_image)

            rVector, tVector = self.rt_Vectors(homogenized, camreaMatrix)

            stereo = self.depthMap(left_image, right_image, maxValue, mindisp, uniquenessRatio)

            points = self.pointsTo3D(stereo, transformation)

            deleteNoise = ~(stereo.reshape(-1)) > (stereo.reshape(-1)).min()
            points = points[deleteNoise]
            RGB = RGB[deleteNoise]

            projected = self.projectPoints(points, rVector, tVector, camreaMatrix, distortion)

            xPoints, yPoints = projected[:, 0], projected[:, 1]

            projected, RGB = self.filterNoise(projected, RGB, xPoints, yPoints, right_image)

            self.createModel(height, width, projected, RGB)

    def OptimalMatrix(self, camreaMatrix, distortion, right_image):
        camreaMatrix, _ = cv2.getOptimalNewCameraMatrix(camreaMatrix, distortion, (right_image.shape[:2]), 1, (right_image.shape[:2]))
        return camreaMatrix

    def rt_Vectors(self, homogenized, camreaMatrix):
        _, RotationMatrix1, _, translationVectors = cv2.decomposeHomographyMat(homogenized, camreaMatrix)
        return RotationMatrix1[3], translationVectors[3]

    def depthMap(self, left_image, right_image, maxValue, mindisp, uniquenessRatio):
        stereo = cv2.StereoSGBM_create(minDisparity = mindisp,
                                       numDisparities = maxValue - mindisp,
                                       blockSize = 7  ,
                                       uniquenessRatio = uniquenessRatio,
                                       speckleWindowSize = 300,
                                       speckleRange = 2,
                                       disp12MaxDiff = 10,
                                       P1 = 8*3*1**2,
                                       P2 =32*3*1**2)
        return stereo.compute(left_image, right_image)

    def pointsTo3D(self, stereo, transformation):
        points = cv2.reprojectImageTo3D(stereo, transformation)
        points = points.reshape(-1,3)
        return points

    def projectPoints(self, points, rVector, tVector, camreaMatrix, distortion):
        projected, _ = cv2.projectPoints(points, rVector, tVector, camreaMatrix, distortion)
        projected = projected.reshape(-1, 2)
        return projected.astype(int)

    def filterNoise(self, projected, RGB, xPoints, yPoints, right_image):
        height, width = right_image.shape[:2]
        deleteNoise = ((0 <= xPoints) & (xPoints < width) & (0 <= yPoints) & (yPoints < height))
        return projected[deleteNoise], RGB[deleteNoise]

    def createModel(self, height, width, projected, RGB):
        model = np.zeros((height, width, 3), np.uint8)
        model[projected[:, 1],projected[:, 0]] = RGB
        global renderedimage
        renderedimage = model
        cv2.imshow("Rendered Data", model)

#Quitting the windows-------------------------------------------------------------------------------------------------------------------------------------
class quitWindow():

    def __init__(self):

        exitWindow = tk.Toplevel()

        master = exitWindow

        master.title("Exit")

        Message = tk.Label(exitWindow,
                           text= """Are you sure that you want to quit,
all unsaved data will be lost!""").grid(row=1, column=1, columnspan=2)
        quitButton = tk.Button(master= exitWindow ,
                               text="Yes",
                               command = self.quitALL).grid(row=2, column=1)
        cancelButton = tk.Button(master= exitWindow,
                               text="No",
                               command = lambda: exitWindow.destroy()).grid(row=2, column=2)

    def quitALL(self):
        app.destroy()


#Minimizing/Maximizing the windows-----------------------------------------------------------------------------------------------------------------------
class minmax_frame():
        def __init__(self):
            global maxScreen
            if maxScreen == True:
                app.attributes("-fullscreen", False)
                maxScreen = False
            elif maxScreen == False:
                maxScreen = True
                app.attributes("-fullscreen", True)
#Error popup window--------------------------------------------------------------------------------------------------------------------------------------
class Errorpopup():
    def __init__(self, error):
        ErrorPopup = tk.Toplevel()

        master = ErrorPopup

        master.title("Error")

        Message = tk.Label(ErrorPopup,
                           text= error).grid(row = 1, column = 1,)
        Button = tk.Button(master = ErrorPopup,
                           text = "Ok", command = lambda: ErrorPopup.destroy()).grid(row=2, column = 1)
#Save files/Presets--------------------------------------------------------------------------------------------------------------------------------------
class saveWindow():
    def __init__(self, x):
        Popup = tk.Toplevel()
        Popup.title("Save")
        font = tk.font.Font(size = 16, family="Helvetica")
        self.x = x
        #Labels--------------------------------------------------------------
        saveLabel = tk.Label(master = Popup,
                             text= """Please enter details to
be saved below""", font = font).grid(row=1, column=2, columnspan=3)
        nameLabel = tk.Label(master = Popup,
                             text= "Name:", font = font).grid(row=4, column=2)
        valueLabel = tk.Label(master = Popup,
                             text= "Values:", font = font).grid(row=5, column=2)
        if len(self.x) == 4:
            tableLabel = tk.Label(master = Popup,
                                  font = font, text= ("""Minimum Disparity: %s"""% x[0] + """
Maximum Disparity: %s"""% x[1] + """
Focal Length: %s"""% x[2] + """
Uniqueness Ratio: %s"""% x[3])).grid(row=3, column=1, columnspan=3)
        else:
            tableLabel = tk.Label(master = Popup,
                                  font = font, text= ("""This will save your
last rendering of two images""")).grid(row=3, column=1, columnspan=3)
        #Button--------------------------------------------------------------
        saveButton = tk.Button(master= Popup,
                               text="Save as", width = 30, font = font, command = self.save).grid(row=6, column=3)
        #EntryBox------------------------------------------------------------
        self.entryBox = tk.Entry(master = Popup, font = font)
        self.entryBox.grid(row=4, column=3)

    def save(self):
        if len(self.entryBox.get()) > 0:
            try:
                if len(self.x) == 4:
                    with open("PresetData.csv", "r") as file:
                        fieldnames = ["UserKey", "Name", "Data"]
                        reader = csv.DictReader(file, fieldnames = fieldnames)
                        for row in reader:
                            if row["UserKey"] == key:
                                if row["Name"] == self.entryBox.get():
                                    exp
                    with open("PresetData.csv", "a") as file:
                        fieldnames = ["UserKey","Name","Data"]
                        writer = csv.DictWriter(file, fieldnames = fieldnames)
                        writer.writerow({"UserKey": "%s" % key,
                                         "Name": "%s" % self.entryBox.get(),
                                         "Data": self.x})
                else:
                    try:
                        if len(renderedimage) != 0:
                            try:
                                with open("RenderedData.csv", "r") as file:
                                    fieldnames = ["UserKey", "Name", "Data"]
                                    reader = csv.DictReader(file, fieldnames = fieldnames)
                                    for row in reader:
                                        if row["UserKey"] == key:
                                            if row["Name"] == self.entryBox.get():
                                                exp
                                with open("RenderedData.csv", "a") as file:
                                    fieldnames = ["UserKey","Name","Data"]
                                    writer = csv.DictWriter(file, fieldnames = fieldnames)
```

```
                                ov2.ppf_match_3d.writePLY(renderedimage, self.entryBox.get())
                                writer.writerow({"UserKey": "%s" % key,
                                                 "Name": "%s" % self.entryBox.get()})
                        except:
                            Errorpopup( """Youve already used that name
please choose another""")
                else:
                    Errorpopup("    Please render a image first     ")
            except:
                Errorpopup("""Youve already used that name
please choose another""")
        else:
            Errorpopup("Please enter a name")


#Open files/Presets-------------------------------------------------------------------------------------------------------------------
class openWindow():
    def __init__(self, _type):
        self.type = _type
        Popup = tk.Toplevel()
        Popup.title("Open")
        font = tk.font.Font(size = 20, family="Helvetica")
        #Label-------------------------------------------
        openLabel = tk.Label(master = Popup,
                             text = """Select what you want
to open""", font = font).grid(row = 1, column = 1)
        #Listbox -----------------------------------------
        self.selectionBox = tk.Listbox(master = Popup,
                                       font = font)
        self.selectionBox.grid(row = 2, column = 1)
        self.ListOptions()
        #Button------------------------------------------
        openButton = tk.Button(master = Popup,
                               text = "Open", font = font, command = self.open)
        openButton.grid(row = 3, column = 1)

    def ListOptions(self):
        if self.type == "pre-set":
            with open("PresetData.csv", "r") as file:
                fieldnames = ["UserKey", "Name", "Data"]
                reader = csv.DictReader(file, fieldnames = fieldnames)
                for row in reader:
                    if row["UserKey"] == key:
                        self.selectionBox.insert("end", row["Name"])
        else:
            with open("RenderedData.csv", "r") as file:
                fieldnames = ["UserKey", "Name", "Data"]
                reader = csv.DictReader(file, fieldnames = fieldnames)
                for row in reader:
                    if row["UserKey"] == key:
                        self.selectionBox.insert("end", row["Name"])

    def open(self):
        if len(self.selectionBox.get("anchor")) > 0:
            if self.type == "pre-set":
                with open("PresetData.csv", "r") as file:
                        fieldnames = ["UserKey", "Name", "Data"]
                        reader = csv.DictReader(file, fieldnames = fieldnames)
                        for row in reader:
                            if row["UserKey"] == key:
                                if row["Name"] == self.selectionBox.get("anchor"):
                                    data = row["Data"]
                                    data = data.replace("[", "")
                                    data = data.replace("]", "")
                                    data = data.replace(",", "")
                                    data = data.split(" ")
                                    dataList = list(map(int, data))
                                    minDispSlider.set(data[0])
                                    maxDispSlider.set(data[1])
                                    focalSlider.set(data[2])
                                    uniqueRatioSlider.set(data[3])
            else:
                with open("RenderedData.csv", "r") as file:
                        fieldnames = ["UserKey", "Name", "Data"]
                        reader = csv.DictReader(file, fieldnames = fieldnames)
                        for row in reader:
                            if row["UserKey"] == key:
                                if row["Name"] == self.selectionBox.get("anchor"):
                                    data = row["Name"]
                                    file = ov2.ppf_match_3d.loadPLYSimple(data)
                                    ov2.imshow("Your Rendered Image", file)

        else:
            Errorpopup("Please select a file")




app = windows()
app.attributes("-fullscreen", True)
app.mainloop()
```