Tampereen yliopisto

Viliam Sälli

SOFTWARE DEVELOPMENT FOR PROCESSING OF IMAGES IN CORRELATIVE MICROSCOPY

Bachelor thesis
Faculty of Engineering and Natural Sciences
Examiners: Turkka Salminen, Lucio Azzari
April 2023

ABSTRACT

Viliam Sälli: Software development for processing of images in correlative microscopy
Bachelors' thesis
Tampere University
Bachelor's degree in Materials science
April 2023

Correlative microscopy is a powerful method for materials science and life science researchers to obtain information from a sample, such as morphology and topography. Using multiple microscopy methods to inspect the same sample yields more data, which requires a versatile tool to analyze accurately. Multiple software solutions exist that can achieve this task, but they are not fully specialized to correlative microscopy. The goal of this thesis is to develop correlative microscopy software with MATLAB App Designer and deliver a functional and intuitive program capable of basic microscope image correlation. The microscopy types focused on are light microscopy, scanning electron microscopy, and certain spectroscopy methods, although the program is designed as a general tool to analyze any images.

MATLAB has an extensive library of built in functions for scientific analysis and image processing, which makes it an instinctive platform to develop image analysis software. MATLAB App Designer is a software building tool that enabled the creation of this project. Visual components were trivial to arrange and tie together, but due to the limited catalogue of components and certain other limitations, some compromises had to be made. Overall, it is a potent tool for creating software for specific tasks. Combined with deep knowledge of MATLABs' intricacies, various App Designers limitations can also be bypassed.

The resulting program is capable of correlating images from different microscopy methods and can be used to analyze and present data. The app includes algorithms for image transformations, image enhancements, color transformations, and custom image blending.

Keywords: Correlative microscopy, graphical interface, software development, image analysis

TIIVISTELMÄ

Viliam Sälli: Ohjelmistokehitys kuvien käsittelyä varten korrelatiivisessa mikroskopiassa
Kandidaatintyö
Tampereen Yliopisto
Materiaalitieteiden kandidaatin tutkinto-ohjelma
Huhtikuu 2023

Korrelatiivinen mikroskopia on tehokas menetelmä materiaalitieteen ja biotieteiden tutkijoille saada tietoa näytteestä, kuten sen morfologiasta ja topologiasta. Useiden mikroskopiamenetelmien käyttäminen saman näytteen tarkasteluun tuottaa enemmän kuvia ja tietoa, jonka oikeaoppiseen tutkimiseen tarvitaan ohjelmisto, joka on erikoistunut useiden kuvien yhtäaikaiseen tutkimiseen. Useita ohjelmistoratkaisuja on olemassa, jotka kykenevät kuvien linjaamiseen, mutta ne eivät ole täysin erikoistuneita korrelatiiviseen mikroskopiaan. Tämän opinnäytetyön tarkoituksena on kehittää ohjelmisto korrelatiivisen mikroskopian käyttötarkoitukseen MATLAB App Designerilla ja luoda toimiva ja intuitiivinen sovellus, joka tarjoaa erilaisia menetelmiä mikroskooppikuvien analysoimiseen. Tämä ohjelma on suunniteltu monipuoliseksi työkaluksi, joka mahdollistaa kaikenlaisten kuvien tutkimisen.

MATLAB tarjoaa laajan valikoiman sisäänrakennettuja funktioita ja toimintoja tieteelliseen analyysiin ja kuvankäsittelyyn. Tämä tekee siitä soveltuvan ohjelmointikielen kyseiseen tehtävään. MATLAB App Designer on ohjelmistojen kehitystyökalu, joka mahdollisti tämän projektin. Visuaalisia komponentteja oli helppo järjestellä ja yhdistellä, mutta rajoitetun komponenttikatalogin ja tiettyjen rajoitusten vuoksi projektin laajuutta täytyi säätää kehityksen aikana. Se on yhteen vedettynä tehokas työkalu pienten ja erityistarkoituksiin luotujen sovellusten kehittämiseen.

Luotu ohjelmisto kykenee linjaamaan kuvia eri mikroskopiamenetelmistä ja sitä voidaan käyttää tietojen analysointiin ja esittämiseen. Sovelluksen algoritmit mahdollistavat kuvien geometristen ulottuvuuksien muunnokset, värinmuunnokset ja kuvien liittämiset mukautetusti.

Avainsanat: Korrelatiivinen mikroskopia, graafinen käyttöliittymä, ohjelmistokehitys, kuvankäsittely

TABLE OF CONTENTS

SYMBOL DESCRIPTION

GUI	Graphical user interface
UI	User interface
UX	User experience
CPU	Central processing unit
GPU	Graphics processing unit
TEM	Transmission electron microscopy
SEM	Scanning electron microscopy
CCD	Charge coupled device
EDS	Energy dispersive spectroscopy

1. INTRODUCTION

Microscopic analysis plays a vital role in experimental research, especially in fields such as materials science, providing essential information that cannot be seen by naked eye. Numerous microscopy methods exist, that specialize in various aspects of material inspection, such as morphology, topography, composition, or structure. To conduct a thorough analysis, multiple techniques must be used, but achieving a completely comprehensive analysis is not practical. For correlative microscopy, overlaying images is a logical approach, although manual alignment can be a tedious task. While there are software solutions available, they tend to be tailored to specific needs. Therefore, we aim to develop an intuitive and efficient software solution that can modify microscope images and overlay them with ease. It is also our objective to make it universal and not constrained to a particular method of microscopy technique.

Sophisticated software, such as ZEISS ZEN and Thermo Fisher Maps 3 are designed for scientific sample analysis and hold a plethora of functionalities for inspecting images [1, 2]. Although these programs hold great potential in the field of correlative microscopy, they are expensive and require special expertise to operate. Our plan is to create software that fits our specific needs without clutter or overcomplicated functionalities that steepen the learning curve. We aim to conduct this project, in collaboration with Jyri Lehto, with the goal of building a solid foundation that is a platform for further development. The focus of this thesis is on creating the graphical user interface (GUI), with some dives into the backend development invisible to the user.

2. CORRELATIVE MICROSCOPY

Correlative microscopy is an advanced field of study, which incorporates multiple microscopy techniques and employs their strengths to extract as much information from a sample as possible. The possible microscopy techniques include but are not limited to light microscopy, electron microscopy, atomic force microscopy and X-ray fluorescence microscopy. [3] Correlative microscopy has been proven to be useful in materials science and life sciences alike, since, e.g., continuously shrinking electrical components and nanoscale sub-cellular structures are points of interest for researchers [4]. Correlative microscopy can be conducted with an arbitrary set of techniques, but most common combination are light and electron microscopy. These techniques have their own advantages and weaknesses, and by combining the data, researchers can obtain complementary information about the same sample. However, the complexity and excessive cost of instruments and software solutions can make it challenging to implement, and the interpretation of the combined data requires specialized expertise. Nonetheless, as advances in technology continue to make correlative microscopy more accessible and user-friendly, it is expected to play an increasingly vital role in a wide range of scientific fields.

2.1 Light microscopy

Light microscopy utilizes visible light and its interactions with a sample to capture information. These interactions include reflections, absorptions and scattering of photons, which can be inspected by eye using special lenses or recorded with camera systems. [5] Light microscopy is divided into subcategories, with bright field microscopy as the most basic one. Bright field microscopy involves illuminating the sample with a light source from below and detecting light that passes through the sample. Since areas of the sample that absorb more light appear as darker regions in the virtual microscope image, structure and topology of the sample can be examined. This technique was originally developed to inspect micro-organisms and cells, but it can be used to inspect other partially transparent samples, such as thin materials. [5, 6] The sample can also be illuminated from above, where reflected photons form the virtual image. This method is more practical when inspecting thicker and opaque samples, which is why it is used widely in material inspection. A basic bright field microscope is pictured in Fig. 1.

Dark field microscopy examines only scattered light, and can be used to detect features, such as defects, that bright field microscopy cannot detect. Illuminating light is blocked by using a toroidal reflection mirror, which illuminates the sample without hitting the eyepiece [7]. This way only the light that scatters from the sample will be captured and displayed in the virtual image. As with bright field microscopy, the sample can be illuminated from below and above.

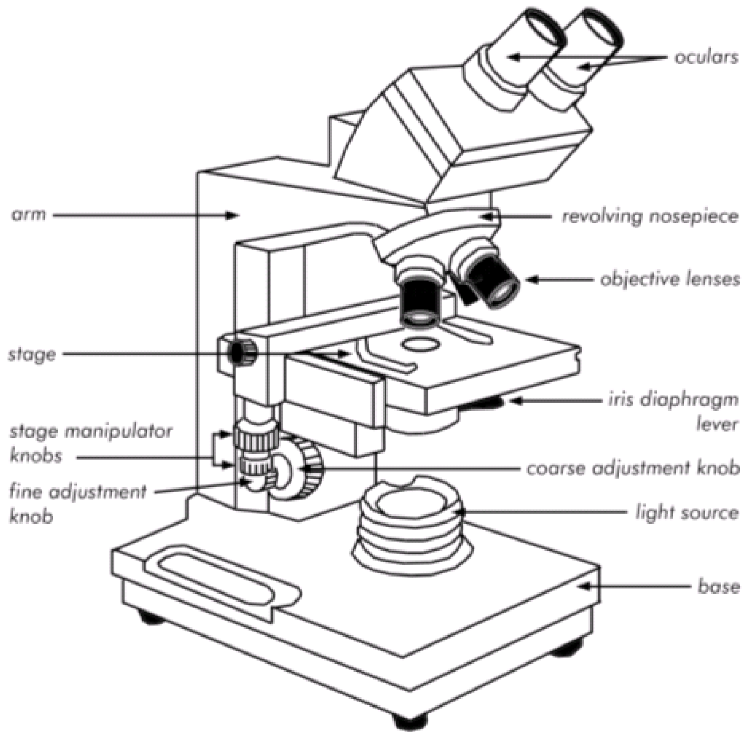


Figure 1: *A bright field microscope and its components [8]*

Light microscopy is a cheap and versatile technique, which can be utilized on a vast scale of samples. Samples are easy to manufacture, and results can be produced swiftly. However, the resolution of the microscopes is limited by the wavelength of light, and smallest details that can be resolved are around 200nm [9]. Depth of field is also an issue, since the samples surface roughness causes blurriness.

2.2 Electron microscopy

Unconstrained by the wavelength of light, electron microscopy is capable of high spatial resolution. Electron microscopes utilize a beam of electrons instead of light to produce a magnified image of a sample.

Transmission electron microscopy (TEM) fires a beam of electrons using an electron gun, which passes through a very thin sample. Depending on the transparency to the electrons, the beam passes through the sample, and irradiates a phosphorescent screen. This process generates photons, which are then captured by charge coupled devices (CCD). [11, 12] Electrons have a wavelength 100 000 times shorter than visible light photons, but due to certain technical limitations, such as lenses, TEM can reach resolutions of 50-200 picometers [9, 10, 14]. The sample must be thin enough to allow electrons to pass through it, typically thinner than 100 nanometers, which makes manufacturing such samples difficult and arduous. [9] TEM is an older method of electron microscopy, and serves a purpose in life sciences and materials science, furnishing information regarding the internal configurations of the sample, such as crystal arrangements and stress state data [13]. In correlative microscopy, TEM images are challenging to align with light microscope images due to the great magnification difference and limited field of view.

Scanning electron microscopy (SEM) also utilizes electrons, but it detects various information by analyzing the electrons' impacts on the surface. A beam of electrons is scanned across the surface of the sample, and emit X-rays, auger electrons, primary backscattered electrons, and secondary electrons upon impact. These are collected with detectors and converted to a visual representation. [15] This means that the samples thickness is not significant, and the produced image reveals information about the surface and composition of the sample [14].

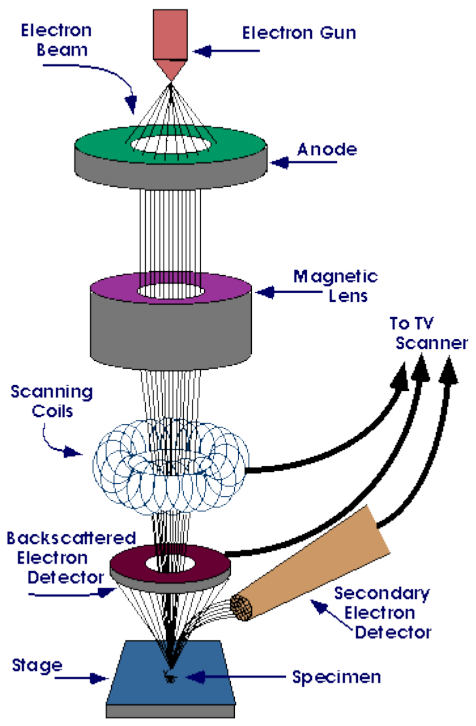


Figure 2: *Individual components of a SEM microscope [15]*

SEM is less challenging to combine with lower resolution microscopy techniques in correlative microscopy, as the sample can be prepared more freely. Dimensions of the sample can be much larger, which means that the sample is easier to inspect using light microscopes and other microscope methods. The resulting image depicts the topography and morphology of the samples surface, which is beneficial to combine with light microscopy. [14]

3. DEVELOPMENT OF THE SOFTWARE

The scope of the project was to create a program capable of presenting and analyzing correlated microscopy data. After loading a dataset of microscope images, their individual location, rotation, and scale can be manipulated in a convenient way, so the images can be precisely overlayed on top of each other. The image brightness, contrast, hue and transparency can be also changed to improve visibility, and image borders can be modified by cropping. These images can then be categorized under tags to classify them e.g., based on the microscopy technique. MATLAB App Designer was chosen as the tool to create the program. With it, a standalone program can be created with minimum time spent on creating the framework, and an extensive archive of MATLAB functions made by our instructors for these specific tasks can be utilized. The following functionalities discussed are the main components of the software, but do not encompass all the tasks required to integrate all the elements together and create a cohesive system.

3.1 MATLAB App Designer

App Designer utilizes a drag and drop style visual element graphical user interface tool, where the program can be fleshed out with ease, and different solutions can be tested for problems. There is a large catalogue of visual elements, such as buttons, sliders, lists and tabs. [16]

These elements have a set of different event callbacks when the user interacts with the elements. These callbacks can be programmed to manage different tasks. However, as it became evident, the set of elements are cemented into the App Designer, with little options to modify them. These limitations were not known before starting the development, so some of the ideas for the project had to be scrapped and the scope of the project adjusted.

3.2 The graphical interface

The graphical user interface consists of a large viewport, where the selected images will get drawn. A viewport is a section of screen that displays the rendered images on top of each other. Since the viewport is the point of interest in the program, it takes most of the space on the GUI. To achieve a user-friendly interface, the tools are arranged along the screen's edges. The rest of the GUI is designed as follows:

- Data loading/saving functions and rendering settings on the left.
- Image hierarchy and categorization on the right.
- Image modifications on the bottom.

Firstly, after fleshing out the initial GUI, the load button was added. Upon clicking this button, a callback function is called, which opens the file selection dialogue. Multiple files can be selected, which returns the file names and file paths inside a string vector. The file names are then combined with the path string, and the image files are read with the `imread()` function. It returns a three-dimensional array of width-by-height-by-3. The first and second dimension correspond to the spatial coordinate of the image's pixel, while the third dimension corresponds to the color coordinates of a pixel, or the intensities of the colors red, green, and blue. The images' arrays are saved inside the program alongside with the file name into class objects called `ImageStruct:s`. Additionally, an active pixel map is created, which keeps track of visible pixels, and oversees transparency of the image. Each pixel of the image corresponds to a number between zero and one in the active pixel map, where zero represents fully transparent pixel, and one fully opaque pixel. Images are also assigned an alpha value, which is multiplied to the active pixel map. This alpha value modifies the opacity of the image and can be controlled by the user. These `ImageStruct:s` are saved inside the apps private cell array `Image_List`.

3.2.1 Image selection and hierarchy panel

On the right side of the app, the user is shown a list of all the images' names loaded into the workspace. Files can be selected, and options changed for the selected files. Choosing the visual element for this task had to be done carefully, since this visual representation of the images is the heart of the program. The user will make most of the user interactions in this window, and the user experience will depend on the functionality of it. This is why several options were considered:

- List box is a list of strings, where elements can be selected. There is an option for multiselection, where multiple strings could be selected. This would be beneficial for manipulating the properties of multiple images at the same time. On the other hand, there are no clear ways to organize the images.
- Table is a two-dimensional array of data. It features multiselection and cell editability, which could enhance the functionality of the software. The columns could display any data about the images, but the hierarchy system would be hard to visualize.
- Node trees are hierarchy systems, where individual elements are children of either the tree element or other nodes. The tag structure conveys the relationships of tags and images effectively, and it features a tick box for each node, which could be used as a hide/unhide button. Then again, it does not have multiselection, and nodes can be nontrivial to rearrange and delete.

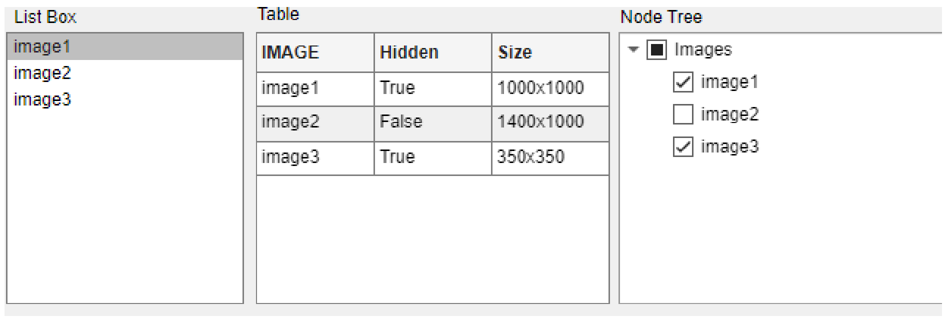


IMAGE	Hidden	Size
image1	True	1000x1000
image2	False	1400x1000
image3	True	350x350

Figure 3: *Examples of the image list alternatives.*

The node tree element was chosen for its visual clarity and the ability to use the tick box. The table could have been promising if it ever were fully realized due to the limitlessness of data it could display.

Since the nodes can be declared from anywhere, and they need to be tied with corresponding images, the nodes are declared alongside the image class structure. The node is saved as a property, and it can be used to identify the selected or checked nodes by looping through the cell array of `ImageStruct`:s and comparing the node to the images saved node.

Since multiple elements cannot be selected from the node tree, a way to transport image properties from image to another is needed, because individually setting the same values for each image by hand is time consuming and tedious. This problem was solved by adding context menus, which are dialog boxes that open when right clicking an element. Each node has an individual menu with a button to apply specific values from a selected image.

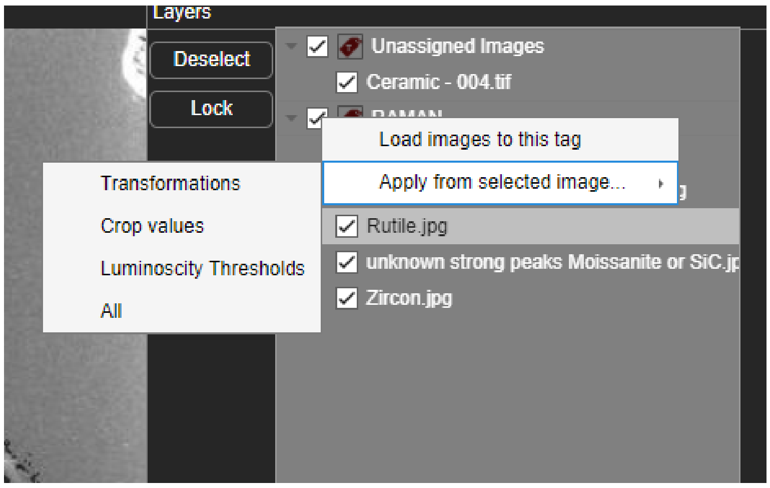
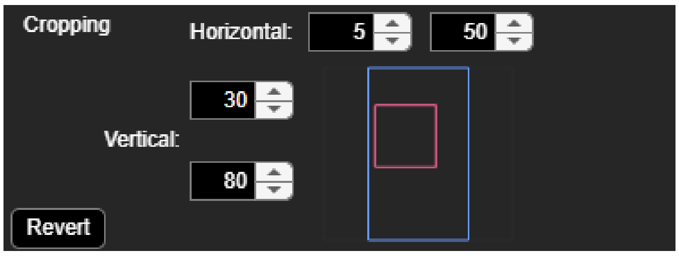


Figure 4: *A context menu that opens when right clicking a tag.*

The “Apply from selected image” button is common for all images and tags, and it opens a submenu, where the selected values will be copied over to either the tag and the images under it, or the image clicked. The user can select to copy over transformations, crop values, luminosity thresholds or everything. However, the button will be non-selectable when nothing is selected. Upon clicking the “Load images to this tag” button, a loading dialogue will open, and the selected images will be loaded under the tag. This button is tag specific, and it will not display when right clicking an image.

3.2.2 Unnecessary data removal

Microscopy images often contain non-essential elements such as microscope information and scale bars that do not play a significant role in examining the pictures. Highlighting only the important parts also speeds up each drawing cycle, since reducing the number of pixels results in fewer calculations required. This is why cropping is a useful tool to implement early on. Cropping can be implemented with array slicing, which is built into MATLAB, but can only produce rectangular crop sections. This is primitive, but often sufficient, to cut off unnecessary parts of the image. To achieve this, four new number input boxes are needed to define the start and end points of the crop, two for horizontal and two for vertical dimensions. These number boxes need to dynamically set their number limits according to their counterparts, for example as the crop start points gets increased, the end point number box should change its minimum value to the new start point. For instant feedback, a visualization of the current crop section is displayed. By utilizing the axis element, it was possible to draw rectangles that correspond to the image shape and the desired cropping section.


Cropping

Horizontal: 5 50

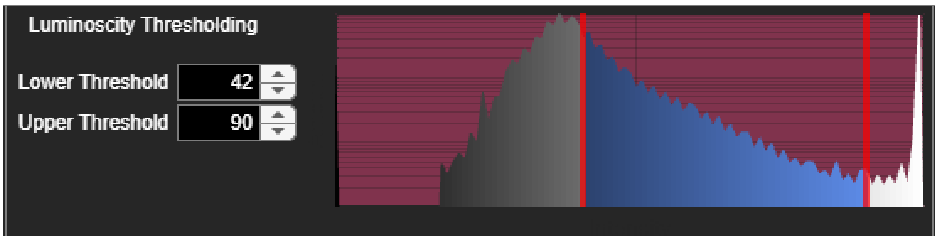
Vertical: 30 80

Figure 5: *The cropping tool*

Even though some of the unnecessary information can be removed with the crop tool, often brighter sections of the image need to be highlighted. Higher color luminosity in microscope images often represent stronger signal, and by thresholding it, the user can adjust the visualization of the signals that are visible.

Images are a collection of pixels, which often represent color with RGB values, but it is not the only way to portray color. HSV is abbreviation from hue, saturation, and value, where hue represents the dominant color wavelength in a circular pattern, saturation depicts the intensity of the color, and value represents the brightness of the color. In the HSV color space, one can adjust any component independently without affecting the others. RGB color space exhibits strong color correlation, so it is advantageous to convert the input image from the RGB color space to the HSV color space. [19] As the third element, value, is a point of interest, it can be extracted from the HSV image, and saved to each image as a luminosity map.

Each image must have a lower and upper threshold values to filter the pixels based on their luminosity during rendering. Each threshold has an input box which determines the luminosity threshold range.


Luminoscity Thresholding

Lower Threshold 42

Upper Threshold 90

Figure 6: *The luminosity thresholding tool.*

Additionally, a histogram of the appearance rates of luminance values in the image is displayed in the tool. This is calculated from a luminance map, which represents each pixels' luminance value in a two-dimensional grid. To filter out the pixels, which are within the luminosity thresholds, the active pixel map can be utilized, where each pixel has a value between zero or one. Since pixels that correspond to a zero in the active pixel map are fully transparent, all values in the luminance map are examined. If a value falls outside the thresholds, it will be marked as a zero in the active pixel map.

3.3 Drawing the viewport

The process of combining images into a single viewport is a crucial and resource-intensive task. When the user is adjusting the images, the speed of image rendering should not affect the workflow. Given that microscopy images often possess a high resolution, the quantity of calculations will quickly accumulate. MATLAB is a dynamically typed language and is generally around ten times slower than C++ [17]. This is why the program's performance must be evaluated thoroughly, to decide if further optimization is needed.

The initial challenge is to determine the final resolution of the viewport. It is important to ensure that each image can fit within the viewport resolution even after applying scaling, rotation, cropping and offsets. Since computing the scale and rotation values can be computationally intensive, it is beneficial to only recalculate the image and save the output when there is a change in the scale and/or rotation values. The resolution of the viewport could be set to the size of the largest image, but offsetting must be first considered. When rendering content, each image is centered in the viewport using the user defined x and y offsets. If the largest image is displaced, some pixels are drawn outside the boundaries of the viewport, resulting in an error. To guarantee that each pixel is drawn inside the viewport, image sizes are sampled and padded with the absolute value of their respective offsets in each direction. The maximum values obtained from these padded sizes are used to determine the size of the viewport. This way the offsetting origin stays in the middle, but it might create much larger images than needed.

The rotation algorithm uses `interp2()` to interpolate the rotated pixels. It additionally generates black pixels when there is no data to interpolate, or outside the image's boundaries. To address this, the already created active pixel map can utilized again. By rotating both the image and the map, any black pixels that fall outside the boundaries of the rotated image will correspond to a zero on in the map. This is because the rotation algorithm produces zeros outside the rotated area in the active pixel map.

After all image transformations have been applied to the original image's data, the final viewport can be drawn. An array of the viewport's resolution is declared with only zeros inside. The hierarchy of images in the node tree is then looped over from bottom to top, so the images higher in the hierarchy will be drawn on top. Each unticked image in the node tree is skipped. The top-left corner position of the image on the final image canvas is calculated by adding the differences of the viewports and the images centers to the x and y offsets. The pixels are then overridden with the images individual alpha value by adding the images pixels multiplied by the alpha value and the current final images pixels multiplied by the difference of one and the alpha value. The alpha value mixes the images RGB values.

The function is fast enough on its own, but some optimization ideas could be to save the unfinished viewport to each image before it overrides the pixels, so when an image is modified, the viewport could be rendered starting from that image. The background could be taken from the image's stored data and the drawing process continued from there.

3.3.1 Zooming

Despite seeming like a simple functionality in a GUI, achieving zooming in MATLAB App Designer required a lot of planning. This is due to many reasons, such as the way that the visual element responsible for displaying the render works and MATLABs performance. Since zooming is a crucial task which needs to be conducted quickly, it was decided that instead of scaling up the viewport render, it should be cropped into incrementally smaller slices, since scaling is a performance intensive task.

The main zooming technique is an adapted version of the technique that Jazz and Piccolo framework uses [18]. When the user points the mouse pointer to a pixel, and as the user zooms in or out with the scroll wheel, the pixel will be visually in the same spot in the viewport. This zooming technique is frequently found in image manipulation software, such as Photoshop and Gimp 2, and it fits into a fast workflow.

The visual element in charge of displaying the viewport render scales up the displayed image to fit inside its borders. For the purposes of this discussion, this element will be referred as `UI.Image`, as it appears in MATLAB App Designer. This means that either the vertical or horizontal dimension is always the limiting scale factor in the images.

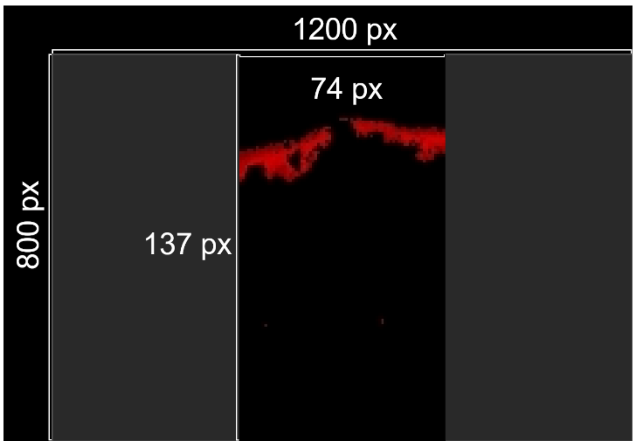


Figure 7: *The pixel dimensions of the `UI.Image` with an example viewport render, where the vertical dimension constrains the scale of the image.*

Fig. 7 exhibits a vertical limiting dimension where the vertical space of the viewport render is fully utilized while the displayed image has empty space on either side, thereby indicating the vertical dimension as the constraining factor. Computing the new dimensions of the zoomed viewport render is not straightforward since the zoomed render should fill the whole `UI.Image` window. Scaling down the raw render dimensions would yield the same shape image as the original one, resulting in unspent space in the `UI.Image` window. This is why the zoomed render shape must be fitted to the shape of the `UI.Image`. There are three scenarios for calculating each of the dimensions, which, for the vertical dimension, are as follows:

1. When the constraining dimension is the horizontal dimension, and the vertical space of the `UI.Image` is greater than the height of the viewport render, there is surplus space to draw the image. Whole vertical dimension of the render can be drawn.
2. The constraining dimension is still horizontal, but the viewport render is already zoomed to fit inside the `UI.Image`. Overflowed parts of the render must be cropped to maintain the correct aspect ratio.
3. Constraining dimension is vertical, so the new height is simply the height of the viewport image multiplied by the zoom factor.

After the new zoomed dimensions have been computed, the position from which the raw viewport render gets cropped must be defined. To figure out the pixel that the mouse pointer is pointing to, a pixel ratio is calculated, which is a measure of the scaling relationship between the viewport render and `UI.Image`. The distance from the currently displayed images top-left corner to the mouse pointer is calculated and multiplied with the pixel ratio, which yields the pixel that is under the mouse pointer.

After computing the mouse position on the viewport render, the zoom crop start position can be calculated. The render should zoom in a manner that the pixel under the mouse pointer is in the same position after zooming. This can be achieved by maintaining the ratio between the distance to the pixel the mouse pointer is currently pointing to, and the total dimension length. This only works if the user zooms from zoom level 1 to 2, so data from last zoom cycle must be used. The function used to calculate this simplifies into the equation 1, where z_{pos} is the zoom crop start position, X is the pixel under the mouse, z is a zoom factor, σ is the size of the viewport render, ζ is the zoom crop start position from last zoom cycle and ρ is the size of the zoomed crop section from last zoom cycle:

$$z_{pos} = X - \frac{\sigma * z * (X - \zeta)}{\rho} \qquad (1)$$

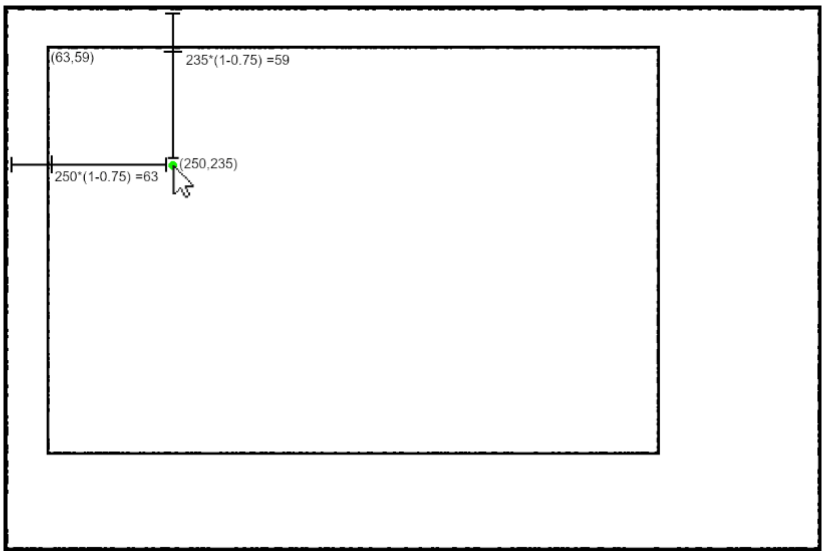


Figure 8: *Zoom example, where the render size is 1200 by 800 pixels, and gets zoomed in by a factor of 0.75. Since last zoom factor was 1, equation 1 simplifies to the displayed calculation.*

After all values have been calculated, the zoomed image can be cropped from the raw render, and calculated data can be saved for the next cycle. The resulting zooming functionality is deemed intuitive enough, and no other way of moving the zoomed image is currently needed.

3.4 The backend development

Even though the focus of this thesis was on the graphical elements of the program, some backend development was crucial to support the graphical user interface. These methods do not affect the graphical interface directly, but rather are necessary to maintain an effective workflow.

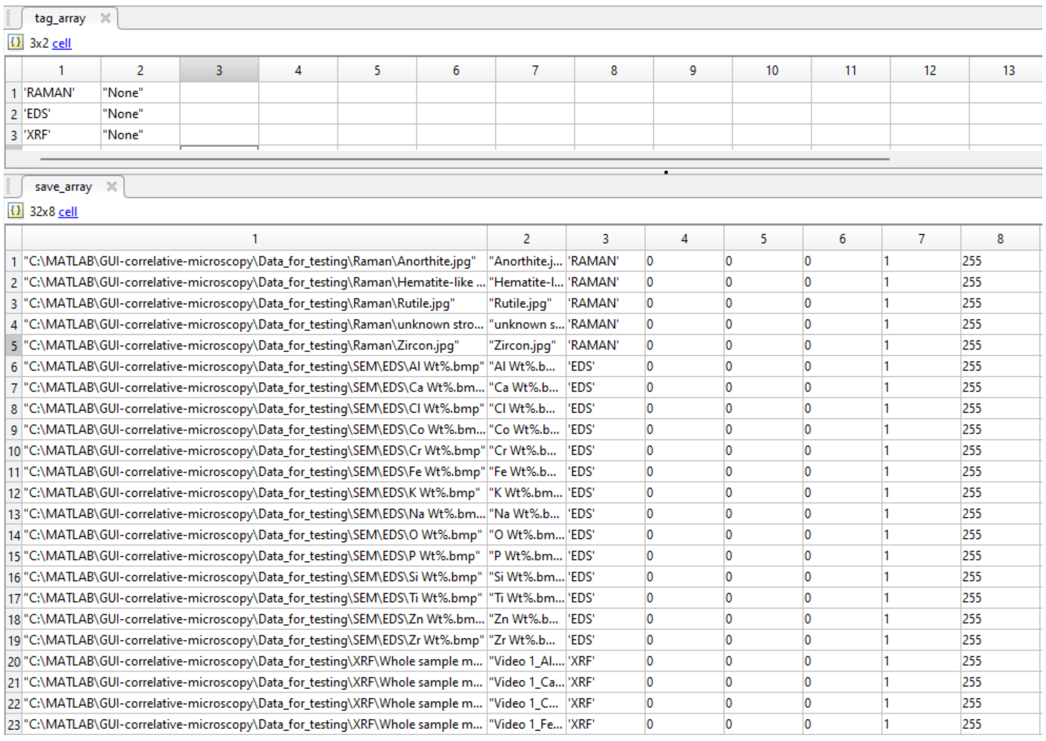
3.4.1 Saving and loading projects

As the program inherently serves as a presentation tool, users ought to have the capability of saving an ongoing project, which can then be accessed when necessary. MATLAB has an integrated feature, where the workspace variables will be saved in a `.mat` file, which can be opened again to access the saved variables. One may select specific variables from the workspace to save within the `.mat` file, or alternatively, opt to save the entirety of the workspace variables. Saving all variables is not a desirable or even viable way to save the session, since MATLAB App Designer objects cannot be saved with the `save()` function. The `ImageStruct`:s could be saved, but the uncompressed image data variables could take dozens of megabytes of space. The least space consuming way of saving the data is to save the paths to the images, load them again, apply the transformations and list them under desired tags. The saved file will work only on the same computer used to save the project, since the paths to the images are not guaranteed to match on different computers. The need for this feature would be rare, but it would be an idea to implement in further development.

The tags must be saved alongside the images inside the save file. This is done by creating a cell array with two dimensions. The width will be two and height is the same as the number of custom tags appended to the project. On each row, a tags name and parents name are present. The tags are added into the cell array from top to bottom, so the parent is always added before the children's nodes.

The images are saved in a comparable way, by creating a cell array and retaining crucial information of the images, such as:

- The path to the image file
- The filename of the image
- The tags name
- Transformations
- Etc.



3x2 cell

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	'RAMAN'	"None"											
2	'EDS'	"None"											
3	'XRF'	"None"											

32x8 cell

	1	2	3	4	5	6	7	8
1	"C:\MATLAB\GUI-correlative-microscopy\Data_for_testing\Raman\Anorthite.jpg"	"Anorthite.j...	'RAMAN'	0	0	0	1	255
2	"C:\MATLAB\GUI-correlative-microscopy\Data_for_testing\Raman\Hematite-like ...	"Hematite-l...	'RAMAN'	0	0	0	1	255
3	"C:\MATLAB\GUI-correlative-microscopy\Data_for_testing\Raman\Rutile.jpg"	"Rutile.jpg"	'RAMAN'	0	0	0	1	255
4	"C:\MATLAB\GUI-correlative-microscopy\Data_for_testing\Raman\unknown stro...	"unknown s...	'RAMAN'	0	0	0	1	255
5	"C:\MATLAB\GUI-correlative-microscopy\Data_for_testing\Raman\Zircon.jpg"	"Zircon.jpg"	'RAMAN'	0	0	0	1	255
6	"C:\MATLAB\GUI-correlative-microscopy\Data_for_testing\SEM\EDS\Al Wt%.bmp"	"Al Wt%.b...	'EDS'	0	0	0	1	255
7	"C:\MATLAB\GUI-correlative-microscopy\Data_for_testing\SEM\EDS\Ca Wt%.bm...	"Ca Wt%.b...	'EDS'	0	0	0	1	255
8	"C:\MATLAB\GUI-correlative-microscopy\Data_for_testing\SEM\EDS\Cl Wt%.bmp"	"Cl Wt%.b...	'EDS'	0	0	0	1	255
9	"C:\MATLAB\GUI-correlative-microscopy\Data_for_testing\SEM\EDS\Co Wt%.bm...	"Co Wt%.b...	'EDS'	0	0	0	1	255
10	"C:\MATLAB\GUI-correlative-microscopy\Data_for_testing\SEM\EDS\Cr Wt%.bmp"	"Cr Wt%.b...	'EDS'	0	0	0	1	255
11	"C:\MATLAB\GUI-correlative-microscopy\Data_for_testing\SEM\EDS\Fe Wt%.bmp"	"Fe Wt%.b...	'EDS'	0	0	0	1	255
12	"C:\MATLAB\GUI-correlative-microscopy\Data_for_testing\SEM\EDS\K Wt%.bmp"	"K Wt%.bm...	'EDS'	0	0	0	1	255
13	"C:\MATLAB\GUI-correlative-microscopy\Data_for_testing\SEM\EDS\Na Wt%.bm...	"Na Wt%.b...	'EDS'	0	0	0	1	255
14	"C:\MATLAB\GUI-correlative-microscopy\Data_for_testing\SEM\EDS\O Wt%.bmp"	"O Wt%.bm...	'EDS'	0	0	0	1	255
15	"C:\MATLAB\GUI-correlative-microscopy\Data_for_testing\SEM\EDS\P Wt%.bmp"	"P Wt%.bm...	'EDS'	0	0	0	1	255
16	"C:\MATLAB\GUI-correlative-microscopy\Data_for_testing\SEM\EDS\Si Wt%.bmp"	"Si Wt%.bm...	'EDS'	0	0	0	1	255
17	"C:\MATLAB\GUI-correlative-microscopy\Data_for_testing\SEM\EDS\Ti Wt%.bmp"	"Ti Wt%.bm...	'EDS'	0	0	0	1	255
18	"C:\MATLAB\GUI-correlative-microscopy\Data_for_testing\SEM\EDS\Zn Wt%.bm...	"Zn Wt%.b...	'EDS'	0	0	0	1	255
19	"C:\MATLAB\GUI-correlative-microscopy\Data_for_testing\SEM\EDS\Zr Wt%.bmp"	"Zr Wt%.b...	'EDS'	0	0	0	1	255
20	"C:\MATLAB\GUI-correlative-microscopy\Data_for_testing\XRF\Whole sample m...	"Video 1_Al....	'XRF'	0	0	0	1	255
21	"C:\MATLAB\GUI-correlative-microscopy\Data_for_testing\XRF\Whole sample m...	"Video 1_Ca...	'XRF'	0	0	0	1	255
22	"C:\MATLAB\GUI-correlative-microscopy\Data_for_testing\XRF\Whole sample m...	"Video 1_C...	'XRF'	0	0	0	1	255
23	"C:\MATLAB\GUI-correlative-microscopy\Data_for_testing\XRF\Whole sample m...	"Video 1_Fe...	'XRF'	0	0	0	1	255

Figure 9: *Visualization of the save file, where the upper array is the saved tag array, and the lower array is the saved image array.*

After the two cell arrays have been declared, a file selection dialog will be opened. The user is asked to select a destination for the saved data. After successfully selecting the path to the save file, a `.mat` file is created, where the two cell arrays values are held. The resulting file is no larger than two kilobytes, which more than acceptable for this project.

Loading this data is not strictly straightforward since the saved data is essentially a blue-print for a project. After pressing the load button and selecting a compatible `.mat` file, the loading process begins. The two cell arrays are extracted from the data, and starting with the tags, they are turned into UI elements. Each row of the tag array is inspected, and a new tag is created. To locate the parent tag, the app searches through the existing tag array and matches the tag's name to the saved parent string. The tags are declared first, so the images can be designated straight away to correct tags.

Loading the images is similar in nature to loading the tags. On each row of the saved images cell array a new `ImageStruct` is declared, the image data is fetched from the saved image file path, and the saved transformations are applied to it. It is then appended to the `Image_List` array, and a new node is created into the tree.

3.4.2 Optimizations

The viewport rendering function is the bottleneck of the whole program, since it is executed every time any of the loaded images get manipulated in any way. Fine-tuning an image often requires multiple manipulations in quick succession of each other, so the function needs to be made as fast as possible by eradicating all unnecessary calculations to shave off milliseconds from each cycle.

As previously stated, C++ and C are generally ten times as fast as MATLAB, but most built-in functions are optimized to run as fast as possible, and the team behind MATLAB at MathWorks have put a significant effort into making built-in functions to run as efficiently as possible [20]. This is why these functions should be favored over self-made functions as often as possible. However, these functions are often written in C or Fortran, and MATLAB possesses the ability to compile the code to a format that can be utilized in interpreted script. Converting MATLAB code to C could be a way to optimize crucial code that cannot be replaced with built in functions.

After profiling the function and testing out which parts take the most time, the conclusion was that 90% of the time spent rendering the viewport came from three sequential lines, which were executed once for every visible image:

1. The array that contains the images’ active pixel map gets multiplied by the images’ alpha value, which is a real value in [0,1].
2. Repeating the array three times for each color channel of the RGB image.
3. Replacing the pixels in the final viewport render to the sum of the image data multiplied by the alpha value array and the current viewport render data multiplied by the complement of the alpha value array.

Since image resolutions tend to be high in microscope images, rendering times range from milliseconds to even minutes. Optimizing these three lines could have the greatest impact on the performance, so several attempts were made. Firstly, the alpha array repeating is obsolete, so it can be removed entirely. Secondly parallelization was tried instead of vectorized calculations, by parallelizing each row calculation to run at the same time. Both were then automatically converted to C using MATLAB Coder, and then compiled to MEX, so the functions can be utilized from MATLAB. Lastly a custom script was created in C++, which accomplishes the same objective, and compiled to MEX. Here are the optimization attempts tested by rendering 12 images with a resolution of 1024 by 808 pixels 100 times.

Table 1: Optimization attempts on viewport rendering.

	Total time for 100 renders (seconds)	Average time per render (milliseconds)
Original code	21.1019956	211
Removed alpha array repeat and transferred the code to an individual function.	14.0972052	141
Removed alpha array repeat function auto converted to C and compiled to MEX.	62.452083	625
Parallelized individual pixel replacement.	115.253684	1152
Parallelized individual pixel replacement function auto converted to C and compiled to MEX.	112.899848	1129
Custom written C++ code compiled to MEX.	n/a	>10000

Optimization attempts were mostly fruitless since compiling to MEX seemed to negatively impact performance. The custom written C++ code was promising, since writing nuanced tasks in a more efficient language intuitively should raise performance, but inexplicably performed distinctly worse than other solutions. Insufficient time spent researching the cause of the poor performance yielded no answers to this. Removing the alpha array repeating however yielded a 35 % improvement in performance. As of now, it will remain as the solution. The custom written C++ code will remain as a good starting point for future optimizations. Another potential optimization method could be using GPU arrays, which run code on graphical processing units. GPUs excel on calculations that require heavy parallel processing, such as matrix calculations and image processing. However, utilizing GPU arrays can be non-trivial, and requires supported NVIDIA hardware, which restricts the availability of computers capable of running the code. [22]

4. RETROSPECTIVE TO DEVELOPMENT

The choice to use MATLAB as the language for the software had both advantages and disadvantages. On the other hand, programming in MATLAB was fast due to it being a high-level language, with a vast catalogue of built-in functions optimized for mathematical and scientific calculations. It was often convenient that built in functions existed to solve extremely specific problems that occurred during development. MATLAB does not have a steep learning curve since the syntax is easy to comprehend and variable types do not need to be declared due to MATLAB being an interpreted language. App Designer is extremely easy to use, and the drag and drop style visual element placement provided a quick way of testing out graphical user interfaces. However, the main drawbacks of the software, such as slowness and clunkiness of using it, and no viable ways to modify the visual elements, constrain the software from its true potential.

MATLAB is very RAM heavy, as running the software often uses up to 3 gigabytes of RAM, which can affect performance in some computers. The program also rarely uses more than 30 % of CPU power due to poor optimization. QtCreator features an app creator, which runs on C++ [21]. Due to C++ memory management, the software created with it could potentially consume less RAM and be more CPU efficient. However, with limited experience working with C++ the development would have been slower, and QtCreators visual elements are as constrained as MATLABs components. Python tkinter is a GUI framework, and would have been much more flexible to develop, but Python is notoriously slow [17].

In conclusion, there are no perfect solutions for entry level software development since more efficiency often requires more development time and skill. For the purposes of this project, MATLAB was deemed to be more than sufficient.

5. CONCLUSION

The completed program met its expectations and was delivered within the deadlines. The program can overlay the images in an intuitive manner and features a set of tools to enhance displaying data. Most features that were proposed in the beginning of the project are featured in the finished program, but it still has a long way to go before it can be considered finished. An example usage case can be seen in Fig. 10.

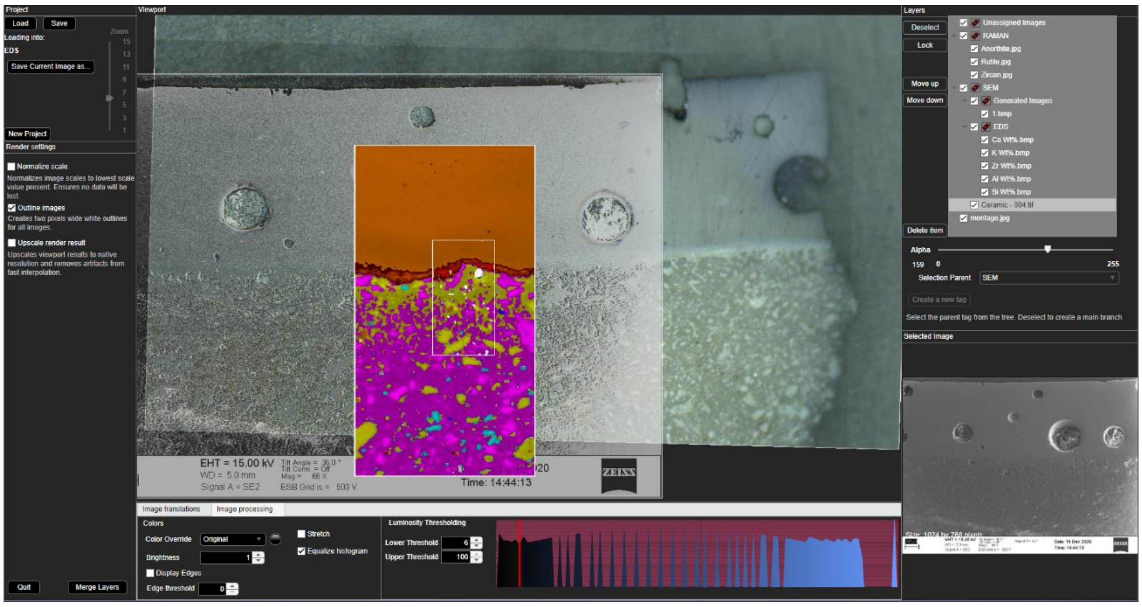


Figure 10: *The finished program with aligned data*

In Fig. 10, SEM, Raman microscopy and energy dispersive spectroscopy images were aligned with a bright field microscope image. This overlaying was conducted in roughly ten minutes, which was accelerated by knowing roughly where the images were taken from the sample. Spectroscopy images are color-coded to the corresponding elements. The sample in question was a ceramic fragment attributed to Chai kiln that was active in the Five dynasties period in China (approx. 950 to 1000 AD). The top part of the image is the glazing, and the bottom is the ceramic. The data indicates that a calcium-rich interface layer has formed. The ceramic is rich in aluminum, oxygen, potassium, and silicon. The ceramic consists of grains of different compounds, whereas the glazing is mostly homogeneous. Raman microscope detected anorthite or calcium aluminosilicate in the border, which explains the overlapping of aluminum and calcium and dominance of calcium in the border.

In examples such as this, the practicality of correlative microscopy becomes evident. Observations and theories can be reinforced with different microscopy techniques, and new findings can be synthetized by combining pre-existing data. The program fulfilled its objectives while compiling this data. Overall, the program serves its purpose, and can be used effectively to correlate microscopy data. With its potential for further development and improvement, this application holds promise for future use in professional laboratory settings.

SOURCES

[1] Thermofisher.com. (2019). *Thermo-Fisher-MAPS-V3*. [online] Available at: https://www.thermofisher.com/fi/en/home/electron-microscopy/products/software-em-3d-vis/maps-software.html [Accessed 3 Apr. 2023].

[2] Zeiss.com. (2023). *ZEISS ZEN Microscopy Software*. [online] Available at: https://www.zeiss.com/microscopy/en/products/software/zeiss-zen.html [Accessed 3 Apr. 2023].

[3] Loussert Fonta, C. and Humbel, B.M. (2015). Correlative microscopy. *Archives of Biochemistry and Biophysics*, [online] 581, pp.98–110. doi:https://doi.org/10.1016/j.abb.2015.05.017.

[4] Eswara, S., Pshenova, A., Yedra, L., Hoang, Q.H., Lovric, J., Philipp, P. and Wirtz, T. (2019). Correlative microscopy combining transmission electron microscopy and secondary ion mass spectrometry: A general review on the state-of-the-art, recent developments, and prospects. *Applied Physics Reviews*, [online] 6(2), p.021312. doi:https://doi.org/10.1063/1.5064768.

[5] Technology Networks (2021). *An Introduction to the Light Microscope, Light Microscopy Techniques and Applications*. [online] Analysis & Separations from Technology Networks. Available at: https://www.technologynetworks.com/analysis/articles/an-introduction-to-the-light-microscope-light-microscopy-techniques-and-applications-351924 [Accessed 6 Apr. 2023].

[6] Ward, B. (2021). *What Is Bright-field Microscopy?* [online] Microscopeclarity.com. Available at: https://microscopeclarity.com/bright-field-microscopy/ [Accessed 8 Apr. 2023].

[7] Nikon's MicroscopyU. (2023). *Darkfield Illumination*. [online] Available at: https://www.microscopyu.com/techniques/stereomicroscopy/darkfield-illumination [Accessed 8 Apr. 2023].

[8] Austincc.edu. (2023). *What are the parts of the brightfield microscope? PreLab 3.8*. [online] Available at: https://accmultimedia.austincc.edu/biocr/1406/labm/ex3/prelab_3_8.htm [Accessed 17 Apr. 2023].

[9] Czech, M. (2015). *Limit of resolution of optical microscope - WikiLectures*. [online] Wikilectures.eu. Available at: https://www.wikilectures.eu/w/Limit_of_resolution_of_optical_microscope#:~:text=The%20resolution%20of%20the%20light,than%20400%20nm%20is%20needed [Accessed 17 Apr. 2023].

[10] Molecular Biology of the Cell. (2017). *Conventional transmission electron microscopy*. [online] Available at: https://www.molbiolcell.org/doi/full/10.1091/mbc.e12-12-0863 [Accessed 13 Apr. 2023].

[11] Elert, G. (2023). *Glenn Elert*. [online] Hypertextbook.com. Available at: https://hypertextbook.com/facts/2000/IlyaSher-

man.shtml#:~:text=%22The%20smallest%20distance%20that%20can,a%20nm%20can%20be%20achieved.%22 [Accessed 13 Apr. 2023].

[12] Henderson, R., Cattermole, D., McMullan, G., Scotcher, S., Fordham, M., Amos, W.B. and Faruqi, A.R. (2007). Digitisation of electron microscope films: Six useful tests applied to three film scanners. *Ultramicroscopy*, [online] 107(2-3), pp.73–80. doi:https://doi.org/10.1016/j.ultramic.2006.05.003.

[13] News-Medical (2016). *What is Transmission Electron Microscopy?* [online] News-Medical.net. Available at: https://www.news-medical.net/life-sciences/What-is-Transmission-Electron-Microscopy.aspx#:~:text=Transmission%20electron%20microscopy%20(TEM)%20is,such%20as%20structure%20and%20morphology. [Accessed 13 Apr. 2023].

[14] Thermofisher.com. (2020). *Electron Microscopy | TEM vs SEM | Thermo Fisher Scientific - FI*. [online] Available at: https://www.thermofisher.com/fi/en/home/materials-science/learning-center/applications/sem-tem-difference.html#:~:text=The%20difference%20between%20SEM%20and,sample)%20to%20create%20an%20image. [Accessed 13 Apr. 2023].

[15] Purdue.edu. (2019). *Scanning Electron Microscope - Radiological and Environmental Management - Purdue University*. [online] Available at: https://www.purdue.edu/ehps/rem/laboratory/equipment%20safety/Research%20Equipment/sem.html#:~:text=SEM%20stands%20for%20scanning%20electron,medical%20and%20physical%20science%20communities. [Accessed 13 Apr. 2023].

[16] MATLAB ® Release Notes. (n.d.). Available at: https://www.mathworks.com/help/pdf_doc/matlab/rn.pdf. [Accessed 27.3.2023]

[17] Jonathan (2018). *A Comparison of Programming Languages*. [online] QUANTITATIVE RESEARCH AND TRADING. Available at: https://jonathankinlay.com/2018/10/comparison-programming-languages/ [Accessed 27 Feb. 2023].

[18] Reiterer, H. and Büring, T. (2009). Zooming Techniques. Encyclopedia of Database Systems, [online] pp.3684–3689. Available at: https://link.springer.com/referenceworkentry/10.1007/978-0-387-39940-9_1128 [Accessed 23 Mar. 2023].

[19] Entropy in Image Analysis III. (2022). [online] MDPI. doi:https://doi.org/10.3390/books978-3-0365-3516-.

[20] Mathworks.com. (2023). *MATLAB - MathWorks*. [online] Available at: https://se.mathworks.com/products/matlab.html?s_tid=hp_products_matlab [Accessed 3 Apr. 2023].

[21] Www.qt.io. (2023). *Qt Creator*. [online] Available at: https://www.qt.io/product/development-tools [Accessed 4 Apr. 2023]

[22] Mathworks.com. (2023). *Transfer Data to and from the GPU*. [online] Available at: https://se.mathworks.com/help/parallel-computing/gpuarray.html [Accessed 12 Apr. 2023].