Real-time Diagnostic Tools for the Scanning Electron Microscope

Liuchuyao Xu
Robinson College

***Abstract*—**

I. Introduction

The scanning electron microscope (SEM) is a type of microscope that produces images using signals generated from the interaction between electrons and the surface under observation. Higher resolution can be achieved compared to the traditional optical microscope, since electrons have much lower wavelength than light. An SEM can have resolution lower than one nanometre, whereas that of an optical microscope is often limited to a few hundred nanometres [1]. This has benefited a variety of fields. For example, scientists have been using the SEM to analyse the doping density in semiconductor [2]. The goal of the project is to develop software tools in Python to support diagnosis of SEM images for the purpose of assisting the operator or automating procedures.

Fig. 1 shows the basic construction of an SEM. The electron gun generates an electron beam, which is transformed into an electron probe after passing through the condenser lens and objective lens. It is scanned across the specimen under the effect of the scanning coil. As a result of the interaction between the incident electrons and the specimen, some electrons are emitted from the specimen. These are called secondary electrons and are collected by the detector, which generates signals whose magnitude depend on the strength of the secondary electrons. The display unit produces one frame after each complete scan of the specimen.

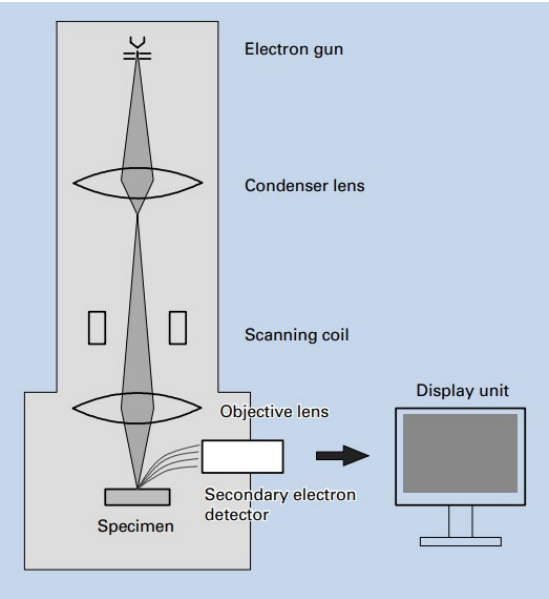


Fig. 1: Basic construction of an SEM [3].

The quality of an SEM image is affected by aberrations. While some exist because of the fundamental properties of the microscope and are difficult to get rid of, some can be completely eliminated by adjusting relevant settings. Two important ones are focus and stigmation, which directly affect the resolution and astigmatism of the image, respectively. Fig. 2 illustrates the effect of wrong focus and stigmation settings.

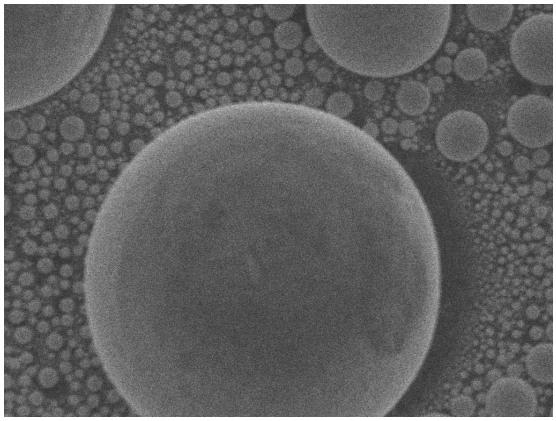
Focus determines the focal point of the electron probe. When the focal point is far from the surface of the specimen, the incident electrons interact with the specimen in a larger area. As a result, spots near each other produce signals of closer magnitude. This makes the image appear blurry, as shown in Fig. 2b.

Stigmation controls stigmators in the SEM, which are used to compensate for astigmatism. Astigmatism arises due to imperfections in components of the SEM, and describes uneven focus in the electron probe, as shown in Fig. 3. When the electron probe is out of focus, astigmatism makes the incident electrons interact with the specimen in an elliptical area, and thus makes the image appear stretched. When the electron probe is in focus, astigmatism makes the image appear blurry. Fig. 4 gives some examples of distorted images.

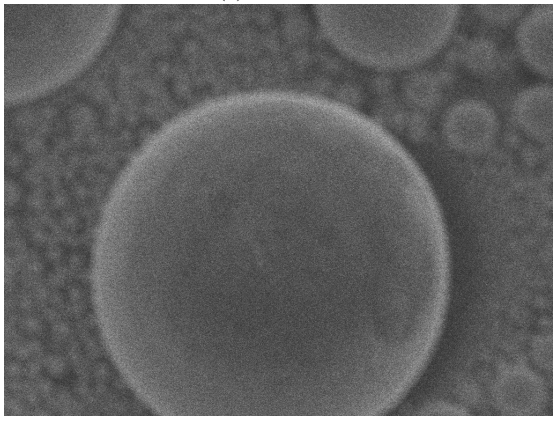
Although experienced SEM operators can often find the right settings for focus and stigmation in a short time, it may not be as straightforward for new users. Sometimes, the surface being observed may have a complex structure and makes adjusting even harder. The complexity arises because any judgement of an image is based on what the operators see through their eyes, which is rather subjective. Intensive training and practical experience are often required for an operator to become efficient in using the SEM.

Fast computing can aid the operators in a few ways. Firstly, a numerical evaluation of the quality of the image may be provided, which eliminates the subjectivity in using human eyes. Numbers are also easier to note down if any record is required. Two operators may have different views on the same image, but the numbers will not be different. Therefore, cooperation and communication between operators can be enhanced. The use of numbers also enable automatic procedures for adjusting settings of the SEM, which save time and may produce better results than doing it manually.

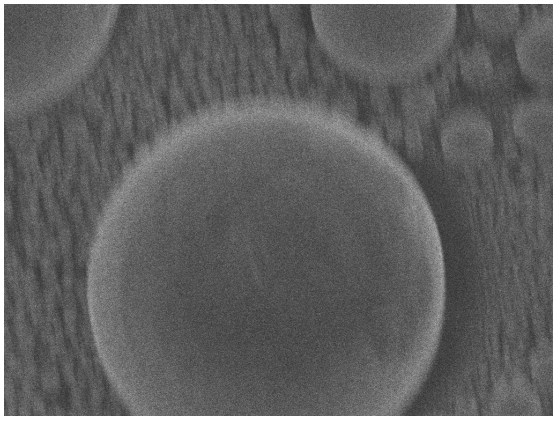
The project focuses on developing tools that use Fast Fourier Transform (FFT) to help operators evaluate the focusing and



(a) In focus.



(b) Out of focus.



(c) In focus with astigmatism.

Fig. 2: Sample SEM images.

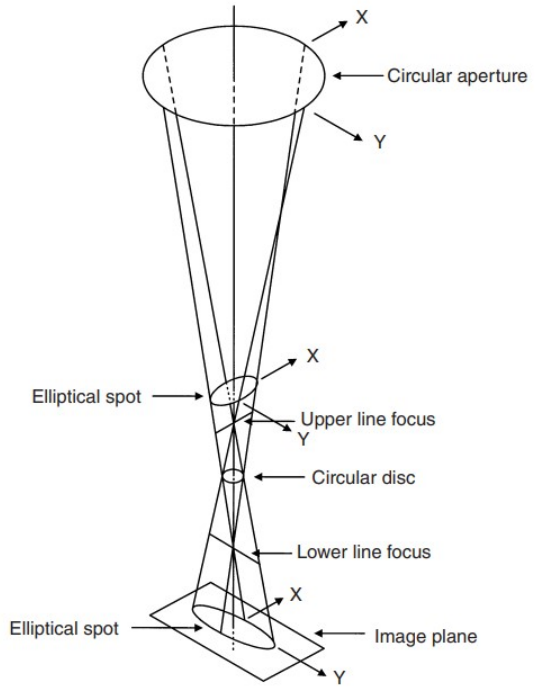
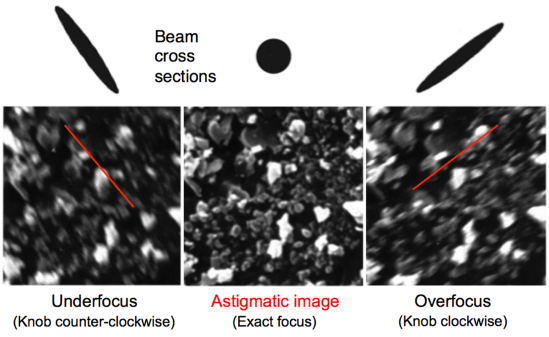
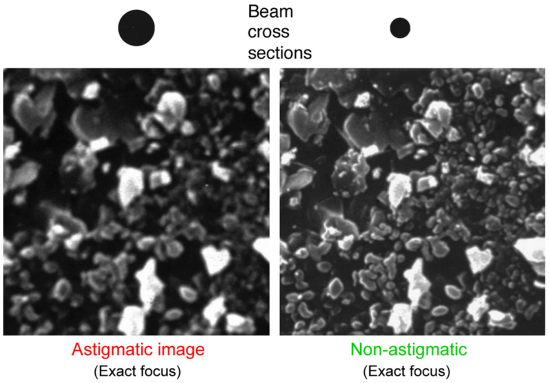


Fig. 3: Uneven focus of the SEM.



(a) Astigmatic images



(b) Astigmatic and non-astigmatic image

Fig. 4: Sample astigmatic SEM images [4].

astigmatism of SEM images, and also looks into an algorithm for the automatic correction of them.

II. Algorithms

A. *Histogram Equalisation*

There are many types of histograms in image processing. The grey level (brightness level) pixel intensity histogram, which plots the number of pixels of each grey value in the image, is the most relevant one for SEM images. This is because an SEM translates the energy of the secondary

electrons directly into a grey level, colours do not exist in SEM images.

The algorithm for obtaining the histogram of an SEM image is given by

$$n_l = \frac{1}{P} \sum_p I(l_p = l), \tag{1}$$

where n_l is the normalised number of pixels of grey level l, P is the total number of pixels in the image and l_p is the grey level of the p_{th} pixel.

Fig. 5a shows an 8-bit grey-scale image, i.e. its depth of digitisation is 8-bit and it has 256 grey levels. Fig. 5b shows the histogram of the image and its integral. As can be seen in the histogram, most pixels are concentrated in the middle of the grey scale. This is reflected by the fact that the image is missing its highlights and shadows.

Histogram equalisation is a method for adjusting the distribution of pixel intensities of an image, in order to improve its overall contrast. Effectively, it is achieved by spreading out the more frequent intensity values. To perform histogram equalisation, first obtain the integral of the histogram using

$$s_l = \sum_{i=1}^{l} n_i,$$

where s_l is the value of the integral at grey level l. The integral spans from 0 to 1 as the histogram is normalised. Scale the integral by the maximum grey level and preform rounding to create the transform function

$$f_l = \lfloor s_l L \rfloor, \tag{2}$$

If a pixel in the original image has grey level l, it will have grey level f_l in the transformed image. For example, if all pixels in the original image are concentrated between grey level 100 to 200, the pixels of level 100 will become 0 in the new image while the pixels of level 200 will become 255 (assuming 8-bit image).

The result is more noticeable when the image has a low contrast, such as the one shown in Fig. 5a. The histogram-equalised version of it is given in Figure 5c and the new histogram in Fig. 5d. More details are now visible as the contrast has been enhanced.

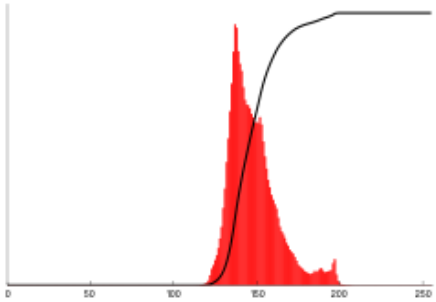
B. Fast Fourier Transform

The Fourier transform (FT) decomposes a function into its constituent frequencies, as illustrated in Fig. 6. The function in red is can be represented by a superposition of all the sinusoidal functions shown, each having a unique frequency and with magnitudes as indicated on the right panel. A function that changes more abruptly will have stronger high-frequency components than a smoother one.

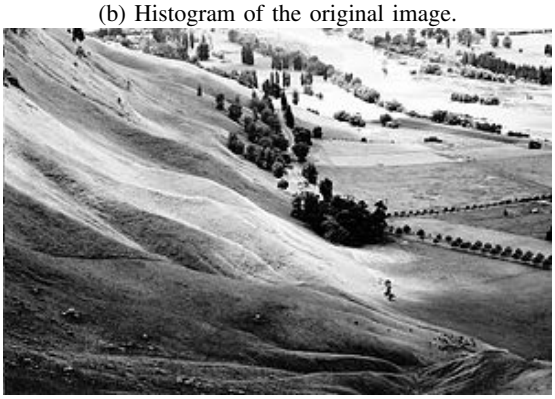
In image processing, the discrete Fourier transform (DFT) is often preferred, which takes a finite sequence of equally-spaced values as input and is thus better suited than the FT. It is defined by

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N}, \quad 0 \leq k \leq N-1, \tag{3}$$

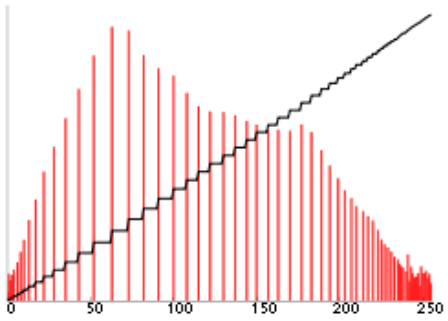
(a) Original image.



(b) Histogram of the original image.



(c) Histogram-equalised image.



(d) Histogram of the histogram-equalised image.

Fig. 5: Histogram and histogram equalisation [5].

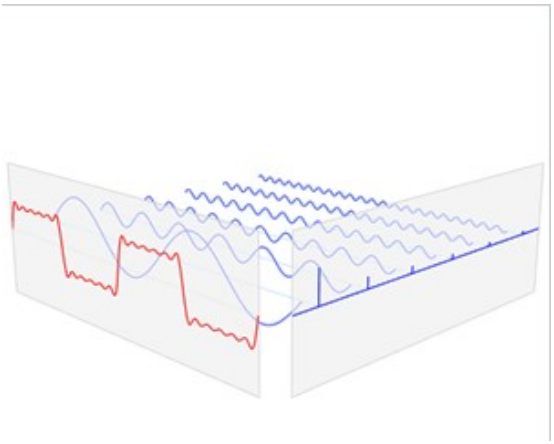


Fig. 6: Fourier transform [6].

where $x_0, x_1, ..., x_{N-1}$ is the input sequence and $X_0, X_1, ..., X_{N-1}$ is the output sequence. X_k is effectively the result of the dot product between the vector $[x_0, x_1, ..., x_{N-1}]$ and $[e^0, e^{-i2\pi k/N}, ..., , e^{-i2\pi k(N-1)/N}]$, and the latter is a finite sequence of frequency $2\pi k$. Therefore, the DFT computes the magnitude of the component of the input sequence with frequency $2\pi k$, for $0 \leq k \leq N-1$.

Fig. 7 gives an example of FFT applied on a real image. The top row shows a set of images and the bottom row shows the corresponding transforms (amplitudes only). Points near the origin represent lower-frequency components and are cut off when a high-pass filter is applied to the image.

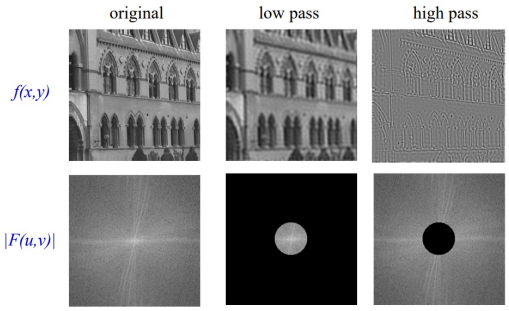


Fig. 7: FFT and filtering on images [7].

A drawback of using (3) is that it has a time complexity of $O(N^2)$. In image processing, where the inputs are 2D sequences, the time complexity quickly explodes and makes the equation practically impossible to use. A fast Fourier transform (FFT) is an algorithm that computes the DFT with smaller timer complexity. There are many feasible algorithms and all known ones have complexity $O(N \log N)$ [8]. Detailed discussion on FFT algorithms is beyond the scope of this report.

C. Focusing and Astigmatism Correction

The focusing and astigmatism of an SEM image can be evaluated using its FFT. Fig. 8 provides a set of sample images that illustrate how different degrees of defocus and astigmatism affect the image and its FFT. An in-focus image contains more details, and its FFT will thus have stronger high-frequency components. As an astigmatic image goes from under-focus from over-focus, the elliptical incidence area of the electron probe rotates by 90 degrees and makes the image appear stretched in the new direction. As a result, its FFT also rotates by 90 degrees. K.H. Ong, J.C.H. Phang and J.T.L. Thong proposed an algorithm for automatic focusing and astigmatism correction [9] using these properties.

The FFT of an image can be converted into a binary image by applying a threshold to the magnitudes, and segmented into 8 regions as shown in Fig. 9. A pixel has value 1 if the magnitude is above the threshold and 0 otherwise. Let I be a matrix representing the current image with focal length set to F, obtain an under-focused image I_{uf} and over-focused image I_{of} by setting the focal length to $F - \Delta F = F_{uf}$ and $F + \Delta F = F_{of}$, respectively. Let FI, FI_{uf} and FI_{of} be matrices representing the binary FFT of I, I_{uf} and I_{of}, respectively.

The focal length can be adjusted by comparing the sum of pixel values in FI, FI_{uf} and FI_{of}. Let the perfect focal length for the image be $\hat{F}$, then

$$\begin{cases} \text{sum}(FI_{of}) < \text{sum}(FI) < \text{sum}(FI_{uf}), & \hat{F} < F_{uf} \\ \text{sum}(FI_{of}) < \text{sum}(FI_{uf}), & F_{uf} < \hat{F} < F \\ \text{sum}(FI_{of}) = \text{sum}(FI_{uf}) < \text{sum}(FI), & F = \hat{F} \\ \text{sum}(FI_{uf}) < \text{sum}(FI_{of}), & F_{of} > \hat{F} > F \\ \text{sum}(FI_{uf}) < \text{sum}(FI) < \text{sum}(FI_{of}), & \hat{F} > F_{of} \end{cases}.$$

Let

$$P = \text{sum}(FI_{of}) - \text{sum}(FI_{uf}). \tag{4}$$

Then, the rules for adjusting focal length are

- when $P > 0$, the focal length should be decreased.
- when $P < 0$, the focal length should be increased.

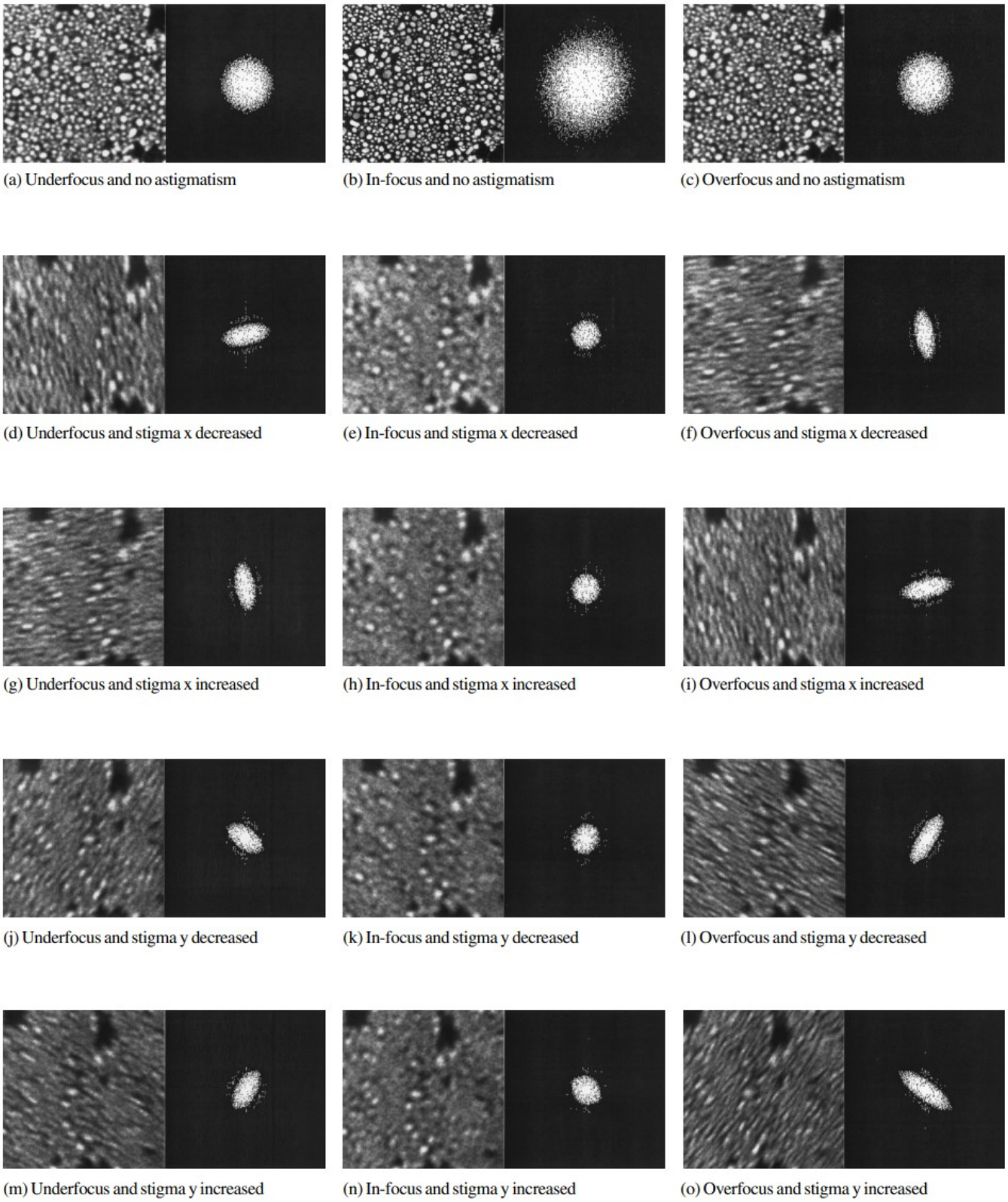
After the focal length has been set to the perfect value, i.e. when $F = \hat{F}$, the stigmator settings can be determined by comparing the sum of pixel values in different regions of FI, FI_{uf} and FI_{of}. The rules for adjustment can be determined using Fig. 8. Let

$$P_{R12} = \text{sum}(FI_{of,R12}) - \text{sum}(FI_{uf,R12}), \tag{5}$$
$$P_{R34} = \text{sum}(FI_{of,R34}) - \text{sum}(FI_{uf,R34}), \tag{6}$$
$$P_{S12} = \text{sum}(FI_{of,S12}) - \text{sum}(FI_{uf,S12}), \tag{7}$$
$$P_{S34} = \text{sum}(FI_{of,S34}) - \text{sum}(FI_{uf,S34}). \tag{8}$$

Then the rules for adjusting stigmator settings are

- when $P_{R12} > 0$ and $P_{R34} < 0$, stigma x should be decreased.
- when $P_{R12} < 0$ and $P_{R34} > 0$, stigma x should be increased.
- when $P_{S12} > 0$ and $P_{S34} < 0$, stigma y should be increased.
- when $P_{S12} < 0$ and $P_{S34} > 0$, stigma y should be decreased.

Fig. 10 shows the general flow chart of the algorithm.



(a) Underfocus and no astigmatism
(b) In-focus and no astigmatism
(c) Overfocus and no astigmatism
(d) Underfocus and stigma x decreased
(e) In-focus and stigma x decreased
(f) Overfocus and stigma x decreased
(g) Underfocus and stigma x increased
(h) In-focus and stigma x increased
(i) Overfocus and stigma x increased
(j) Underfocus and stigma y decreased
(k) In-focus and stigma y decreased
(l) Overfocus and stigma y decreased
(m) Underfocus and stigma y increased
(n) In-focus and stigma y increased
(o) Overfocus and stigma y increased

Fig. 8: Images of gold-on-carbon sample and their fast Fourier transforms (FFTs) for different degrees of defocus and astigmatism [9].

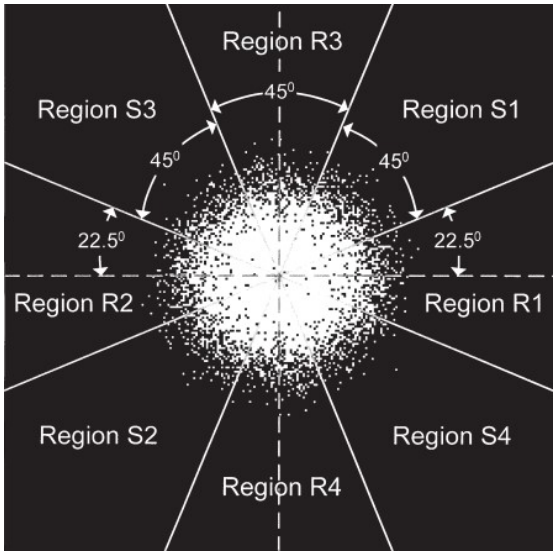


Fig. 9: Segmented fast Fourier transform [9].

III. The Software

A. Overview

The software controls the SEM and takes images from it using its application programming interface (API). The API provided by the manufacturer supports only C++. To make development easier, a Python wrapper (SEM_API.py) for the native API has been written by Luyang Han and is used by the software.

The architecture of the software is designed with a priority on maintainability. There are four modules in the software — *SemTool*, *SemImage*, *SemCorrector* and *Masker*. *SemTool* handles the graphical user interface (GUI), *SemImage* provides a class for containing an SEM image, *SemCorrector* provides the automatic focusing and astigmatism correction algorithm and uses *Masker* to efficiently segment the FFTs of images. The class diagram of the software is given in Fig. 11, showing only the major classes. The modularity improves the readability of the code and makes it easier to maintain and develop.

The Python *numpy* and *cupy* library are the main packages used for performing mathematical operations. *Numpy* provides an interface to be used in Python, but implements the functions using C code at the bottom level, which run on the CPU directly and improve performance. *Cupy* provides a similar interface as *numpy*, but runs the code on the GPU. The parallel processing power of the GPU can make *cupy* way faster than *numpy* for calculations involving matrices, such as calculating the FFT of an image. To ensure good performance, algorithms written in pure Python are avoided wherever possible.

B. SemImage

Fig. 12 shows the *SemImage* class. It encapsulates all variables and methods that are directly relevant to an image taken by the SEM. Each *SemImage* contains three *numpy.ndarray* instances — *array*, *fft* and *histogram*, which store data of the image, its FFT and histogram, respectively.

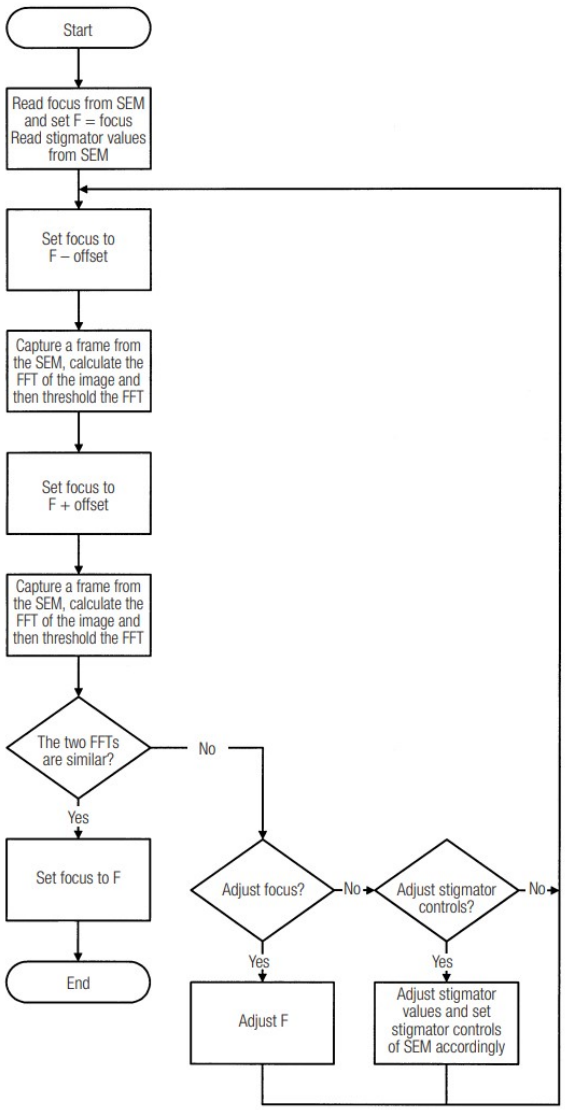


Fig. 10: General flow chart of the correction algorithm [9].

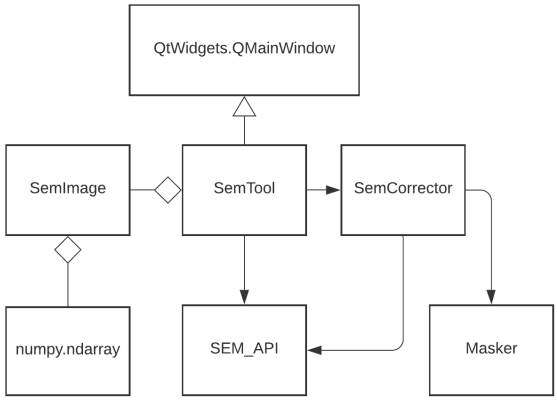


Fig. 11: Class diagram of the software in unified modelling language (UML)

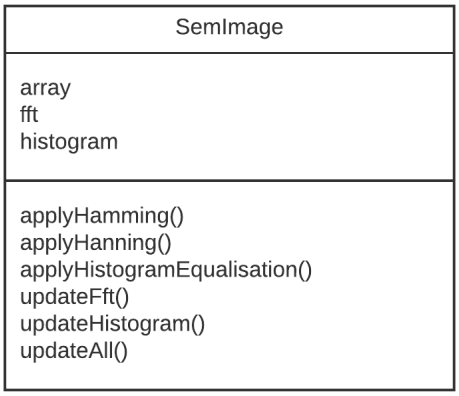


Fig. 12: *SemImage* class.

An image is effectively a finite sequence of data, and this can cause boundary effect when its FFT is calculated. The abrupt discontinuity at the edges gives the FFT some strong high-frequency components. To circumvent this, the *SemImage* class provides two methods — *applyHamming()* and *applyHanning()*, which apply a Hamming and Hanning window function to the image data, respectively. The window functions taper the edges off towards zero, and thus reduce the boundary effect. Fig. 13 illustrates how they look like, and they can be obtained by setting a_0 to 0.5 and $25/46$, respectively, in the function

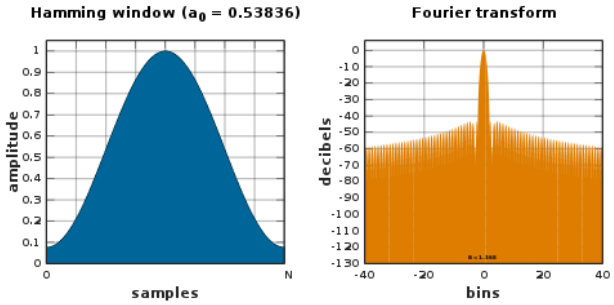
$$w[n] = a_0 - (1 - a_0) \cdot \cos \frac{2\pi n}{N}, \quad 0 \leq n \leq N. \tag{9}$$

The *applyHistogramEqualisation()* method implements the algorithm as described in II-A. The most time-consuming part of the algorithm is to map all pixels in the original image to their new values using the transform function (2). To achieve good performance, the software uses the Python *map()* function together with the *lambda* operator, which iterates through all pixels in the image and thus has time complexity of $O(N^2)$. This is not optimal and better complexity may be achieved by utilising the GPU. However, that would make the code more complex and harder to read, since the *cupy* package does not provide a straightforward solution to it. With *map()* and *lambda*, the transformation can be implemented in one line:

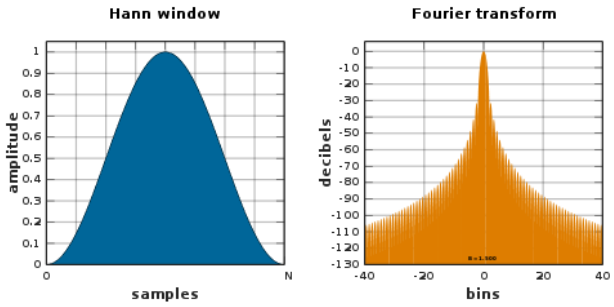
```
map(lambda x: trans[x], image)
```

trans is an array that serves as the transformation function, if the original pixel value is *x*, the new value should be *trans[x]*. The *lambda* operator is a way to create small anonymous functions, which are throw-away functions and are only needed where they are created. *lambda x: trans[x]* creates an anonymous function which returns *trans[x]* when given *x*, and *map()* uses this function to transform all pixels in *image*.

Both *updateFft()* and *updateHistogram()* use functions in *cupy* to perform the calculations. Since *cupy* may not always



(a) Hamming window.



(b) Hanning window.

Fig. 13: Window functions [10].

be available due do the hardware environment, the software automatically switches to *numpy* when *cupy* cannot be used.

The following code demonstrate how to initialise an *SemImage* instance, apply Hamming window to the image, copy its FFT to the variable *fft* and apply histogram equalisation to it.

```
image = SemImage(data)
image.applyHamming()
fft = image.fft
image.applyHistogramEqualisation()
```

C. Masker

A key factor that affects the performance of the automatic focusing and astigmatism correction algorithm is the speed of performing segmentation on the FFTs. The most straightforward way of doing it is to iterate through the whole matrix while calculating the sums in pure Python, for every FFT. This has a time complexity of $O(N^2)$. The *Masker* module provides a faster solution.

A *Masker* can be initialised with a 2D shape, and it will create eight matrices, each representing a region as shown in Fig. 9. For example, if the input is $[5, 7]$, the matrix created for region R1 will be

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

The algorithm ensures that there is no overlapping between matrices apart from at the center. After obtaining the matrix

for region R1, the sum of of values in an FFT in R1 can be calculated using the *numpy.ma* module:

```
ma.array(fft, mask=masker.r1).sum()
```

IV. Demonstrations

A. Real-time Histogram Equalisation

Screenshots of the software. Demonstration of the algorithm. Speed test of the algorithm.

B. Real-time Fast Fourier Transform

Screenshots of the software. Demonstration of the algorithm. Speed test of the algorithm.

C. Automatic Focusing and Astigmatism Correction

Screenshots of the software. Demonstration of the algorithm. Speed test of the algorithm.

V. Next Steps

Next steps.

Need to add more details on how things were made fast.

References

[1] "Scanning electron microscope." en.wikipedia.org. https://en.wikipedia.org/wiki/Scanning_electron_microscope (accessed May 9, 2020).

[2] T. Agemura and T. Sekiguchi, "Secondary electron spectroscopy for imaging semiconductor materials," 2018 International Symposium on Semiconductor Manufacturing (ISSM), Tokyo, Japan, 2018, pp. 1-3, doi: 10.1109/ISSM.2018.8651171.

[3] *Scanning Electron Microscope A to Z.* JEOL Ltd. [Online]. Available: https://www.jeol.co.jp/en/applications/pdf/sm/sem_atoz_all.pdf. Accessed: May 9, 2020.

[4] "Correcting Astigmatism in SEM Images." cambridge.org. https://www.cambridge.org/core/journals/microscopy-today/article/correcting-astigmatism-in-sem-images/7ED43987C7916AAFBE1869522546AC84/core-reader (accessed May 10, 2020).

[5] "Histogram equalization." en.wikipedia.org. https://en.wikipedia.org/wiki/Histogram_equalization (accessed May 11, 2020).

[6] "Fourier transform." en.wikipedia.org. https://en.wikipedia.org/wiki/Fourier_transform (accessed May 12, 2020).

[7] University of Oxford. (2014). 2D Fourier transforms and applications. [Online]. Available: http://www.robots.ox.ac.uk/

[8] "Fast Fourier transform." en.wikipedia.org. https://en.wikipedia.org/wiki/Fast_Fourier_transform (accessed May 12, 2020).

[9] K.H. Ong, J.C.H. Phang and J.T.L. Thong, "A robust focusing and astigmatism correction method for the scanning electron microscope," 1997, scanning 19: 553-563, doi: 10.1002/sca.4950190805.

[10] "Window function." en.wikipedia.org. https://en.wikipedia.org/wiki/Window_function (accessed May 13, 2020).