MASARYK
UNIVERSITY

FACULTY OF INFORMATICS

Grain detection and manipulation library

Bachelor's Thesis

LENKA JANÍKOVÁ

Brno, Spring 2025

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Grain detection and manipulation library

Bachelor's Thesis

LENKA JANÍKOVÁ

Advisor: doc. RNDr. Barbora Kozlíková, Ph.D.

Department of Visual Computing

Brno, Spring 2025


FACULTAS ARTIS INFORMATICÆ
UNIVERSITAS MASARYKIANA

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

I declare that I used Grammarly during the writing of this thesis for grammar checking and writing suggestions. I verified its outcome and take full responsibility for it.

Lenka Janíková

Advisor: doc. RNDr. Barbora Kozlíková, Ph.D.

Acknowledgements

I want to thank my advisor, doc. RNDr. Barbora Kozlíková, Ph.D., for helping me with the structure of the thesis and for valuable advice and reviews throughout the entire writing process.

I also want to thank the team from Edhouse s.r.o. and Thermo Fisher Scientific Brno s.r.o., namely Ing. Michal Křen, Jakub Holzer, Ph.D., Austin Day, Ph.D., Ing. Jiří Jánoš, Ing. Martin Petřek, and Mgr. Ondřej Hlouša, for helping me understand the basic crystallographic concepts and for advice regarding the quality of the code.

Abstract

This thesis aims to implement a C# library for crystallographic grain detection and manipulation and integrate it into the xTalView application developed by the Edhouse s.r.o. company for their customer Thermo Fisher Scientific Brno s.r.o.

The library provides methods for grain calculation based on a multiphase map previously acquired via the electron backscatter diffraction (EBSD) method, different means of grain visualization, and support for their filtering and manipulation.

The assignment was driven by the need to rework these functionalities and add new ones to the application, as the previous implementation was not optimal and needed significant improvements.

Keywords

crystallography, C#, grain, map coloring

Contents

List of Figures

Introduction

Grains play an important role in material science. Their shape, orientation, size, count, and other attributes influence the properties of crystalline specimens, such as plasticity or toughness. By observing them, scientists can determine the optimal way of processing a material in order to reach a certain inner structure with specified properties. Therefore, it is very useful to have software that can determine all of the previously named grain attributes and present them to a user in an understandable way with all the necessary details. Also, since some of the materials are more challenging to measure with electron backscatter diffraction (EBSD) than others, it is necessary to provide the material scientists with a way of improving the collected data.

The Thermo Fisher Scientific Brno s.r.o.[1] company produces electron microscopes of many different types, including those that are specialized to gather EBSD data, which contains the information necessary for grain detection. The software for the microscopes is developed by the Edhouse s.r.o.[2] company under the name xTalView. As a part of the Edhouse s.r.o. company, I have been tasked to rewrite the existing implementation of grain detection and subsequent manipulation of grains in xTalView and add new features that were previously not present in the application. Among these belong, e.g., map coloring by grain properties as well as their respective legends generation. The aim of this thesis is to implement and integrate a library with these functionalities into xTalView.

In the thesis, I first define the basic crystallographic terms that I use consistently throughout the text. Then, a numbered list of requirements is given in Chapter 2. Chapter 3 firstly describes the solutions that were previously present in xTalView. Secondly, two of the most widely used applications for grain analysis, AZtecCrystal[3] and EDAX OIM Analysis,[4] are investigated in the context of the thesis requirements. In Chapter 4, I give a detailed description of the requirements fulfilling algorithms writing process, their integration into xTalView,

1. `https://thermofisher.jobs.cz`
2. `https://www.edhouse.eu/cz/`
3. `https://nano.oxinst.com/azteccrystal`
4. `https://www.edax.com/products/ebsd/oim-analysis`

their functionality, and the improvements they brought to the application. Chapter 5 contains an assessment of my work written by experts I consulted throughout the time I spent on this thesis. I was able to fulfill all expectations and the library was included in a released version of xTalView.

1 Introduction to crystallography

Crystallography is a natural science field that studies the structure of crystalline materials. By observing these materials, the crystallographers can ascertain the specimen's properties, such as plasticity or toughness [1]. Based on their visualization, it is possible to determine an optimal method of material processing for a specific purpose or to reveal faults in the material's microstructure [2]. In this part of the thesis, I will describe the basic concepts in crystallography that motivated the assignment of this thesis.

1.1 Grain

In crystallography, grain or crystallite refers to a part of a crystalline material with uniform phase and orientation [3]. Polycrystalline silicon with visible grains is shown in Figure 1.1.

Studying grains plays an important role in material science since the microstructure of a material may substantially impact most of its qualities [4]. It is, therefore, desirable to have the ability to affect the shapes and sizes of grains in a material. Some examples of these processes include annealing or cold working.

Annealing consists of heating a metal above its recrystallization point and slowly cooling afterward, which makes the grains regrow, changing the material's microstructure [5]. Cold working is a process of manually deforming a quenched metal, leading to grain elongation [1]. Materials that have smaller and more misshaped grains are tougher, as opposed to those with bigger and more regular-shaped grains, which are more ductile [1].

1.1.1 Grain boundaries

Grain boundaries describe the border between two neighboring grains. Boundaries are manifested by two neighboring points in a map having different phases or orientations [7].

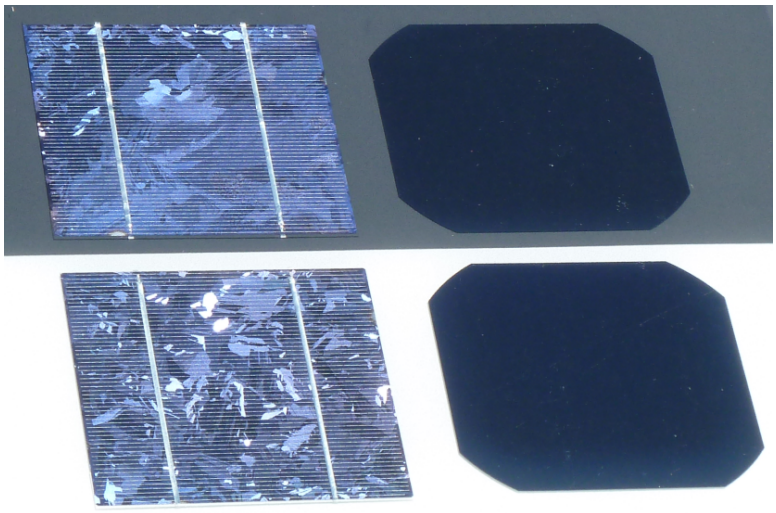

Figure 1.1: Comparison of solar cells made from polycrystalline (left) and monocrystalline (right) silicon. Adapted from [6].

1.1.2 ASTM E112

ASTM E112 is a standard for average grain area determination [8]. Because the scanned area does not usually contain the whole specimen, some grains have incomplete data, and therefore, their total surface area is practically unknown. For the purpose of this thesis, the ASTM E112 standard indicates that the area of grains touching exactly one edge of the scanned region should be multiplied by the factor of 2, whereas those touching two or more edges should have their area multiplied by 4.

1.2 Electron Backscatter Diffraction

Electron backscatter diffraction (EBSD) is a technique used for crystallographic pattern acquisition [9]. It primarily uses scanning electron microscopy (SEM) to acquire a pattern that describes the orientation and phase of a grain in all of the scanned points [10].

During the pattern acquisition via SEM microscope, a high-energy electron beam is sent at the observed specimen at an angle usually

close to 70°. Some electrons of the beam get absorbed, while others are scattered [11, 12]. Part of the backscattered electrons are reinforced and subsequently captured on a screen usually made of phosphor, forming a diffraction pattern consisting of Kikuchi bands [10, 13]. It is possible to determine the crystal's phase and orientation using this pattern [10, 14].

EBSD is generally quite effective at obtaining grain-related information; therefore, it is important for EBSD software to be able to process and manipulate it. This fact led to the assignment of this thesis.

1.3 xTalView

xTalView is the name of the software that necessitated the creation of the library described by this thesis. The application allows the user to collect data using an SEM microscope and an EBSD detector and to further analyze and manipulate it. The intended use case of the software is similar to that of applications such as AZtecCrystal [15] or EDAX OIM [16], which are concerned with EBSD data analysis. At the same time, xTalView is offering the ability to acquire EBSD data, similarly to AZtecHKL [17] or EDAX APEX [18]. Since these applications are not free to use for trial purposes and the Thermo Fisher Scientific Brno s.r.o. company does not have any agreement with the authors of the software, I was not able to explore the applications any further than through their official promotional materials, which are my sole sources of information regarding these.

2 List of requirements

The following requirements were set by Ing. Michal Křen, team leader and branch manager at Edhouse s.r.o., Ing. Jakub Holzer, Ph.D., application scientist at Thermo Fisher Scientific Brno s.r.o., and Austin Day, Ph.D., EBSD technical expert and consultant at Aunt Daisy Scientific Ltd. at the beginning of the thesis assignment, with minor changes being implemented throughout the coding process. It was agreed that the library should be developed in C# since the rest of the application is written in WPF [19], which uses C# for code implementation. Also, it is necessary that the implementation is written in a way that makes it easy to add new functionalities in the future.

1. The library shall implement an algorithm for "grain calculation." This entails the discovery of grains based on a multiphase map, in which every point bears information about a phase and orientation. The user should be able to cancel the calculation. In case of interruption, the library should be able to restore itself to the state it was in prior to the start of the algorithm.

   The algorithm should connect pixels of a multiphase map into grains based on their phase and disorientation (least difference of misorientation between two pixels). If two neighboring pixels have the same crystallographic phase and their disorientation fits in a user-defined tolerance, they are considered as a part of the same grain.

   1.1. The library shall calculate a set of properties for each grain. Namely, these properties are area, misorientation, $\phi 1$, Φ, $\phi 2$, number of neighbors, equivalent circle diameter, center of mass X, center of mass Y, and, in most cases, also minor axis, major axis, axes ratio, and ellipse angle.

   If a grain has an area smaller than four pixels, its minor axis, major axis, axes ratio, and ellipse angle are not to be calculated since the grain is not big enough to be significant. These parameters are also not calculated for edge grains in case the user selects to use the ASTM E112 standard for reasons described in the Introduction chapter, Section 1.1.2.

2. The library shall visualize a map of grains based on specific parameters: type ("Standard"/"By Area"/"By Aspect Ratio"), scale ("Blue"/"Rainbow"), and method ("Binned"/"Continuous"). It should also support the creation of their respective legends.

    2.1. "Standard" coloring type: The library shall assign a color to every single grain while ensuring that no two grains sharing a common boundary are colored the same. The algorithm should be written in a way that makes it effortless to add a possibility for the user to select a custom color scheme in the future.

    2.2. The library shall assign a color to every single grain according to values of input fields type (other than "Standard"), scale, and method. If the "By Aspect Ratio" coloring type along with the ASTM E112 standard are chosen, the algorithm should differentiate between unindexed points, grains without aspect ratio, and grains with aspect ratio, and color these distinctly.

    2.3. Any combination of type, scale, and method, in which the type is not set to "Standard," is assigned a legend containing the key for the coloring mechanism. If the "Standard" coloring type is chosen, all existing legends for grain map should be deleted; otherwise, four legends are to be created. These are combinations of minimum/maximum values in either pixels or micrometers and the minimum/-maximum string, along with the title of the selected coloring type in black or white letters.

3. The library shall support highlighting a single grain based on the position of a cursor on the map. The color used for highlights should be visibly different from those already present in the map.

    3.1. The library shall implement means to figure out which grain a selected point in a map belongs to.

4. The library shall specify and implement filters applicable to grains based on their properties. It should allow the initialization of multiple filters on the same property. Once the filters

are created, it should be possible to change or delete them. The filtered data should be reflected by changes in the grains table, grains map, and grains statistics.

5. The library shall support the import and export of binary files containing information needed to reconstruct all calculated grains in a project. It is necessary to maintain backward compatibility in order to be able to use data created during the structure formation; therefore, the exported files should have an easily recognizable header that indicates its format.

6. The library shall implement algorithms for the manipulation of the original multiphase map. It should also support statistics reports based on the number of changed pixels on the map. Multiphase map points should bear information regarding the origin of their data.

    6.1. The library shall enable the user to remove grains that are smaller than a user-defined threshold. The input parameters are the same as with grain calculation, along with a parameter that sets the minimum number of pixels a grain has to consist of in order to be kept. The displayed information should describe the number of newly unindexed pixels.

    6.2. The library shall provide the user with the ability to add unindexed points to neighboring grains based on the number of adjacent pixels from a single grain. The input parameters are the same as with grain calculation, along with a parameter that describes the "aggressiveness" setting, which sets the minimum number of pixels that belong to a single grain and that are adjacent to an unindexed pixel in order to include the pixel in the examined grain. The parameter's value must be between 2 and 7 pixels. The displayed information should describe the number of newly indexed pixels.

    6.3. The library shall enable the user to apply a bilateral filter on grains. The input parameters are exactly the same as

those for grain calculation. The scope of the bilateral filter should be a single grain, not the entire map.

7. The library shall support statistics calculation on grain properties, namely minimum, maximum, average, median, and standard deviation on the following properties: area, misorientation, neighbor count, ellipse angle, ellipse major axis, ellipse minor axis, ellipse axes ratio, and equivalent circle diameter.

   The statistics should be calculated only from currently visible grains. In the case of properties that are not found in certain grains, such as the ellipse axes ratio, the grains that do not bear these should not be taken into consideration for these calculations. If there is no grain with non-null values of a certain property, the result visualization for that property should remain empty.

3 Existing Solutions

This chapter explores the existing solutions. The first section is concerned with the implementation formerly used in xTalView. The following section describes well-known applications used in EBSD data analysis, EDAX OIM [16], and AZtecCrystal [15].

3.1 xTalView

Before the assignment of this thesis, a lot of the required functionality existed but needed reworking, primarily because of faults in the algorithms as well as new requirements. On a few occasions, it happened that a single piece of code was called repeatedly, which slowed the code down, or that the naming of methods and fields was no longer matching their desired meaning. I will summarize the way the algorithms were previously implemented and explain why it was necessary to update them.

3.1.1 Grain calculation

The previous implementation of Requirement 1 used a variation of flood fill to find grains. Its results were correct, but the algorithm was quite lengthy and could not be parallelized with map acquisition. Also, there was no way of interrupting the algorithm once the calculation started.

3.1.2 Grain property calculation

This algorithm is connected to Requirement 1.1. Since the property calculation relies on set mathematical formulas, there was not much I could do to improve the calculation process itself.

However, the ASTM E112 standard was previously not required; therefore, the algorithm used to calculate all properties of every grain, even those that did not make sense, namely the minor axis, major axis, axes ratio, and ellipse angle. The mentioned properties are used to describe the grain's shape for ellipse fitting. Since the ASTM E112 creates artificial data about an edge grain's area and, therefore, about

its shape, it is impossible to calculate these properties in a meaningful way.

3.1.3 Grain map visualization

Before the assignment of this library, there was only one method of grain coloring, which did not work properly, as will be described in the following paragraph. This led to the formulation of Requirement 2 and its sub-requirements.

3.1.4 Standard color assignment

The "Standard color assignment" algorithm refers to Requirement 2.1. Previously, the application used to assign a grain's color based on its ID. The modulo operation was applied to it, and the resulting number served as an index to a list of colors. This was very quick, but it had one serious flaw: there was no guarantee that two neighboring grains would not be colored the same. This led to multiple grains getting connected visually since, by default, the grain boundaries visibility is turned off.

3.1.5 Coloring by properties

At the beginning of my work, there was no existing implementation for coloring by specific parameters as described by Requirement 2.2; the only implementation of color assignment was the one described in the "Standard color assignment" paragraph above.

3.1.6 Legends generation

The only grain map coloring algorithm used before the creation of this library was the one described in Section 3.1.4, which did not use any meaningful metrics to determine a grain's color; therefore, no legends as per Requirement 2.3 for grain map were needed.

However, other maps in the application do use similarly specified legends to describe the key used for the map's color determination, such as index quality (IQ) maps. I took these as a visual guide and updated them to fit the specifications defined for grain legends.

3.1.7 Highlighting grains

The original algorithm used for grain highlighting, as described by Requirement 3, darkened the color by subtracting an RGB constant from the color of the grain. This approach worked well until the implementation of highlighting by specific properties since the range of used colors was much larger than before. In some cases, the highlighted grain could not be well recognizable or could even appear unindexed (painted in black).

Another flaw of the previous implementation was that if a grain was completely surrounded by a highlighted grain, then the inner grain was darkened as well.

The grain discovery based on a position in the map was done through a GrainsMap object, a map that held multiple information for each of its pixels, with one of these being the ID of a grain it belongs to. This resolved Requirement 3.1 well, but the solution was not as straightforward as it could have been.

3.1.8 Grain filtering

Previously, the application did not support multiple filters of a single property. Also, a higher number of decimal numeric errors was reported. The process of adding, removing, or updating a filter, along with the updates of the table, map, ellipse, and other, were written in a way that made the grains be filtered multiple times, which was redundant and could be quite lengthy. Instead of small, specified classes for filters with unique fields, one class with many null values was used for filter settings definition.

3.1.9 Binary import and export

The original binary export was missing the misorientation information, leading to the corresponding column in the grains table being empty after loading an already calculated project. As the structure changed quite significantly since the beginning of the library implementation, the need for new binary import and export arose in the form of Requirement 5. It was thus also necessary to maintain some level of backward compatibility.

3.1.10 Data manipulation

Previously, the application did not keep track of the statistics of the reindexed points. Also, prior to my rework, there was no demand to keep the data origin for each pixel. As per Requirement 6, the points of the multiphase map should now contain a field set to contain the following values:

1. Indexed: the point has valid crystallographic data that was acquired during map acquisition.
2. Unindexed: the point does not have valid crystallographic data acquired during map acquisition.
3. Filled in: the point has valid crystallographic data, but this data was copied from a neighbor during the "Fill simple gaps" algorithm.
4. Cleaned up: the point does not have valid crystallographic data, because it was set as unindexed during the "Denoise" algorithm.

Applying the "Bilateral filter" algorithm does not change the data origin value since the method slightly changes the data of almost all indexed points, but does not newly index or unindex any points. Also, the data is usually changed by quite a small value.

3.1.11 Denoising

The previous implementation used to first calculate the grains, then remove the small ones from the multiphase map, and afterward calculate the grains again. Based on the comments I found in the code it seems that sometimes in the past, there used to be an initiative to calculate anti-grains. Anti-grains are parts of the map with unindexed pixels, which are connected in a similar way to grains. These can then provide useful information, such as the amount of porosity in the material, the shape of the pores, and more. This was never fully accomplished and the idea was suppressed. Therefore, the second grain calculation was redundant and only had the effect of updating the GrainsMap.

3.1.12 Fill simple gaps

In the past, the UI did not allow for changes in the "aggressiveness" setting of the algorithm used to fulfill Requirement 6.2. Its value was set to 5, which was too restrictive. Otherwise, the basic structure of the algorithm worked well; therefore, I used parts of it in my solution.

3.1.13 Bilateral filter per grain

The formerly used implementation of bilateral filter worked well and there were no complaints regarding its performance.

3.1.14 Statistics

Originally, the only statistics that were available to the user were median grain area, average grain area, standard deviation of grain area and the total number of grains. These did not correspond to the subset currently filtered by the application and always referred to the full dataset.

3.2 Other existing crystallographic software

The other applications that are most used in the crystallographic field, namely AZtecCrystal [15] and EDAX OIM [16], are not free to use for trial purposes; therefore, I was not able to observe them more closely than from official promotional materials. To analyze them, I used mostly official training videos showing the use cases of these applications that are related to the desired functionality of xTalView.

3.2.1 Grain calculation

A video [20] published by EDAX states that the EDAX OIM software reconstructs grains using orientations of points of the map. No further specifications of the algorithm are given. I was not able to find any information about the grain calculation process in AZtecCrystal.

3.2.2 Grain property calculation

EDAX OIM calculates many properties for all grains, including, e.g., size and misorientation [21]. Some of these statistics are calculated on anti-grains as well, such as size, aspect ratio, etc.

According to one of the training videos [22] published by Oxford Instruments, AZtecCrystal offers a table containing information about a currently displayed map. The user may also choose to display "Grain sizing results," which bear information about singular grains. The information present is phase, area, perimeter, equivalent circle diameter, and more.

3.2.3 Grain map visualization

Based on a video [23] I found on the EDAX YouTube channel, it seems that the software can color each pixel according to its properties, which may or may not indicate grains. Also, grains can be visualized by drawing boundaries between points with certain misorientations. Lastly, the application offers the ability to color grains in a way that no two adjacent grains are similar [23]: see Requirement 2.1. It is not entirely clear how many colors are used. Based on the aforementioned video I was able to discern approximately 15 different colors used in one of the examples, though, because of the YouTube video compression and the fact that there are no explicitly drawn grain boundaries, this number might not be definitive.

I did not find any example in which AZtecCrystal would color grains in random colors so that no two neighboring grains looked the same; instead, to visually divide two grains, the presenter in YouTube videos on the Oxford Instruments channel visualized grain boundaries with sought-after properties.

3.2.4 Coloring by properties

A video [16] published by EDAX mentions the possibility of coloring individual pixels by their properties. Coloring by the orientation of a crystal lattice axis is demonstrated with the coloring key being an inverse pole figure (IPF) map legend based on a predefined sample axis. This process suggests grains as areas that are colored with similar shades. In xTalView, IPF maps are also available [24]. This is not

an entirely reliable way of visualizing grains, though, since the key only considers the orientation of the crystal lattice at the given point, meaning that additional information for grain detection, such as phase or rotation angle, is omitted. For example, it might happen that two grains with the same or very similar rotation axis lie directly next to each other, in which case they will be colored with the same shade. This will visually merge them and suggest that they are both parts of a single grain, which is incorrect because of the phase difference. Similar caveats happen with most ways of coloring by properties, though, and can be easily resolved by adding grain boundaries to the map.

In a video [25] I found on the Oxford Instruments YouTube channel, a way of coloring a map by grain properties is shown. The inputs that the user may select are phases, color scheme, range type, and grain parameter.

3.2.5 Highlighting grains

According to a video [21], it is possible to click on a grain in a map in order to highlight and separate it. It is possible to highlight many grains simultaneously.

The AZtecCrystal software offers the ability to highlight a grain using the "Select grain" tool. This allows the user to click on a point in a map to select a grain to which the point belongs. The whole grain is then outlined in white [22]. It is revealed that the algorithm used for grain boundary discovery is flood fill [25]. Also, if a grain is selected in the "Grain Sizing Results" table, it is highlighted in the map. The same functionality is also offered by xTalView.

3.2.6 Data filtering

Data filtering in EDAX OIM is called "partitioning" [21]. The dialog window used for partitioning definition allows the user to filter grains as well as singular points. Grains can be filtered based on their size, aspect ratio, rotation angle, orientation spread, etc. Points can be filtered based on their confidence index, image quality, video signal, crystal orientation, phase, and more.

A video [23] I found on the EDAX YouTube channel states that it is possible to filter grains and anti-grains by setting the minimum size

in pixels they have to reach in order to be considered valid. Also, once the grain statistics are calculated, the user can select a group of grains to be visualized by selecting a range in the grain size graph.

A training video [26] published on the Oxford Instruments channel on YouTube shows that grain filtering is done by clicking on a property column title in the grain table, which brings up filter options for the specified property. For properties with numeric values, the user may select two values, the logical operator applied on these, and whether each value is an inclusive or exclusive border value. For other properties, a list of values with checkboxes is displayed. The user may choose to create a subset that stores the currently filtered data. Based on another video [25], it seems that it is impossible to create multiple filters on a single property in one subset. Also, a video that concentrates on working with maps [22] states that it is possible to create a subset by selecting grains directly in a map.

3.2.7 Data export

In a video [21] published on the EDAX YouTube channel, a text import of map data is mentioned. AZtecCrystal supports the manipulation of data in HDF5 format, specifically .h5oina files [15], but the software can read .cpr, .ctf, and .ang files [27] as well. It is also possible to export pixel map data as a .csv file [22].

3.2.8 Data improvement methods

EDAX OIM uses a process called "NPAR," which takes six neighboring patterns for every pattern in the map (EDAX OIM uses a hexagonal grid) and creates a moving average of all seven points [21]. This improves the ability of the software to index points drastically.

AZtecCrystal offers "Wild Spike Removal," which unindexes points that have wildly different orientations compared to their neighbors, "Zero Solution Removal," which tries to add unindexed points to neighboring grains similarly to Requirement 6.2, and "Auto Clean Up," which runs one cycle of "Wild Spike Removal" and two cycles of "Zero Solution Removal." This action can be performed on specific phases and subsets. All cleanup actions are stored and displayed in a

list. The user may change the order of these, in which case they are all performed again in the new order on the original data.

3.2.9 Statistics

A video [23] published by EDAX shows how grain statistics are calculated and visualized in the EDAX OIM software. In the video, an example is given of a graph presenting the distribution of grain size diameter versus area fraction and number fraction. Alongside this graph is the data arranged in a table. The user can also see a summary of statistics, which contains fields such as number of grains, average diameter, etc. The user may choose whether edge grains are included or not. It is possible to create graphs on different properties, such as grain shape, grain average IQ, and more.

In AZtecCrystal, minimum, maximum, mean, and standard deviation are calculated for grain properties of grains that are present in the table at the moment. Some of the properties include area, roundness, equivalent circle diameter, and fitted ellipse angle.

4 Implementation and Integration

In this chapter I will describe the implementation process of both improved and completely new algorithms, along with their integration into xTalView. The algorithms have been tested by the QA team and used in a released version [24] of the application. Throughout the implementation, I was able to use three datasets with various properties. In this section, they will be referred to as Project 1 (map resolution 300x200 pixels, 3,826 grains), Project 2 (map resolution 800x600 px, 3,903 grains), and Project 3 (map size 2000x2000, 108,149 grains). I will use these to present the effects of the algorithms introduced in this chapter.

During the writing of the library functions, I have created a small testing application, which served as a playground for drawing the visual results of my algorithms. Using this application, I wrote the grain calculation algorithm, the coloring algorithm and the methods necessary for grain filtering. This application served as a proof of concept. The algorithms were then integrated to xTalView, which made it possible to expand the library's functionality and see its compatibility with the entire xTalView application.

4.1 Grain calculation

The algorithm resolves all specifications of Requirement 1. The "Grain calculation" algorithm creates the basis for all of the other algorithms, since without grains, other operations specified by the requirements could not be performed.

As mentioned in Chapter 2, whether two neighboring points belong to a single grain or not is decided based on two properties: their phase and disorientation. To group these points, it is necessary that their phase is the same and that their disorientation in degrees fits into a limit set by the user input "Boundary misalign".

As opposed to the previously used flood fill, the algorithm I wrote examines the map in the reading direction. A single pixel is compared to its neighbors to the left and upwards. If none of these are evaluated as being a part of a single grain, a new grain is created containing only the focused point. If one of the two neighboring pixels fulfills

the requirements set by the previous paragraph, the examined pixel is evaluated as a part of the neighboring pixel's grain. Lastly, if both neighboring pixels fit the connecting requirements, the points' grains are either merged, or, if they are already a part of a single grain, the examined point is simply added to the grain.

Once the grains are constructed, the GrainsMap object is created and filled with the grain information. Then, neighboring grains are found, based on whom the coloring algorithm decides what colors can be used for specific grains. The next step uses algorithms that were previously present in the application for grain boundary tracing. These algorithms have shown great results and they work with the new Grain structure definition; therefore, I decided to keep using them. Afterwards, the grain coloring algorithm is called, which is described in one of the following sections.

The algorithm periodically checks whether the user asked for the calculation to be aborted. If the cancellation call is registered before the application moves to the map coloring, the calculation is successfully interrupted and no data is overwritten on the disk. Because of the time constraint before release, the interruption makes the program to be stuck in a state in which the original table and map are displayed, but they are not connected, meaning that grains cannot be highlighted. Also, the menu for displaying grain ellipses, boundaries, etc. and filters are disabled. This was, however, deemed as satisfactory, since the data is not overwritten and the user may simply reload the project by switching between project or sites, by running the calculation again and letting it finish, or by launching and exiting the "Denoise" mode. Upon performing any of these, the project is loaded from the disk, and thus the application is fully functional again. The issue was deemed minor and is planned to be fixed in a future version of xTalView.

The biggest advantage of this algorithm is that the direction in which pixels are examined is the same as the direction in which the map acquisition is performed. Therefore, it is possible to use the algorithm parallel to map acquisition. Both of these actions are quite time-consuming on big maps, thus not having to wait with the start of one until after the other ends is very beneficial.

4.2 Grain property calculation

"Grain property calculation" implements a solution for Requirement 1.1. As foreshadowed in Section 3.1.2, the calculation process consists of firmly set formulas; therefore, I did not change it further than to accommodate the improvements I made to the structure of the Grain class.

My contribution is that the minor axis, major axis, axes ratio, and ellipse angle are only calculated in grains that are big enough and, if the ASTM E112 standard is selected, are not touching the edges of the scanned area. For this purpose, I added a multiplier to the structure of the Grain class that describes how the grain is affected by the standard. The algorithm then looks at the grains' area and multiplier. If the area is smaller than four pixels or the multiplier is not 1, the algorithm skips the ellipse-connected properties calculation and the values of these fields are set to null.

Figure 4.1 shows that when ASTM E112 is selected, the grains that touch the edge of the picture do not have aspect ratio, meaning that they are colored gray and their ellipses are not calculated, thus they are not drawn. This is also true for small grains regardless of whether the correction of size is selected, visualized by singular gray points in the figures.

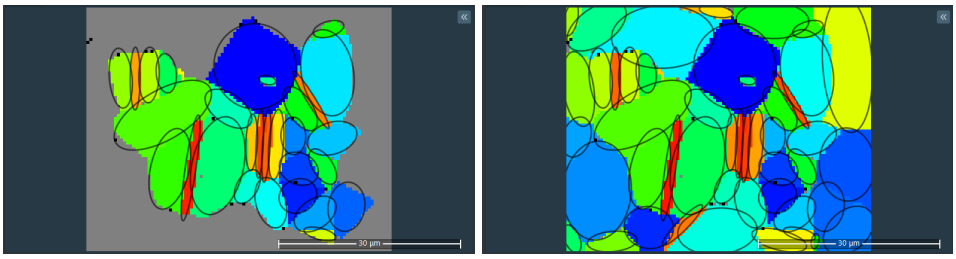

Figure 4.1: Submaps created from Project 2. Calculated with boundary misalign set to 5 °, grain coloring type to "By Aspect Ratio," grain coloring scale to "Rainbow" and grain coloring method to "Binned." The map on the left was calculated with ASTM E112 applied, whereas the one on the right used no correction. Ellipses are drawn in the pictures. Slight changes in color might be observed.

4.3 Standard color assignment

"Standard color assignment" fulfills the specifications of Requirement 2.1. Once the grains are constructed, it is possible to assign to them a standard color that is not truly representative of any property, but instead is chosen so that no two neighboring grains are colored the same.

At the beginning of the assignment, it seemed that this might serve as motivation for the implementation of the four-color theorem:

"The four-color theorem states that any map in a plane can be colored using four-colors in such a way that regions sharing a common boundary (other than a single point) do not share the same color." [28]

Since both Requirement 2.1 and Theorem 4.3 concentrate on coloring a map in a way such that no two adjacent objects are colored the same, my supervisor suggested that I try to implement a solution for the theorem and limit the number of colors to 4.

The first algorithm that I implemented was similar to a BFS algorithm: a grain in the list was assigned a color based on the first available color index of all its neighbors. Afterward, all uncolored neighbors were added to a list, and the index pointing at the examined grain was increased. If there were no more available colors (meaning that all of the grain's neighbors have already been assigned every available color), a repairing algorithm was called: the index was decreased, and the algorithm tried to assign the grain a different color, still trying to use the next lowest color index. This would continue until the repairing algorithm was successful. If the index moved to the end of the list while there were still uncolored grains in the map, an uncolored grain was taken from the grains set and placed into the list. The algorithm ended when there were no more uncolored grains.

I tested the algorithm provided with a list of 21 colors to choose from on the named testing projects, with the biggest number of colors used being 6. To test the repairing functionality, I tried decreasing the number of colors to 5, which significantly slowed down the algorithm because of the need for more frequent calls of the repairing algorithm, and further removal of another color option would make the program run unacceptably slow. However, this was not an issue, since the application specialist's feedback pointed to the preference for more colors rather than less, eliminating the need for further limitations

on the number of colors. After solving a problem closely described in the "Highlighting grains" algorithm section, the number of colors has stabilized at 20. The colors are defined in a way that makes it very easy to add a different color set in the future.

The algorithm had one major flaw, though, because the application seemed to crash during the color assignment on Project 3, with the cause of the crash pointing towards insufficient RAM. This pushed me to make the algorithm more straightforward.

My attempts at simplifying the process led to a second algorithm that would no longer need to keep a list of grains. The new implementation chooses a grain, determines its color based on its neighbors in the same way as previously, and then proceeds to pick the next grain to be colored. This is achieved by observing the newly colored grain's uncolored neighbors. From these, the one with the highest number of neighbors is picked and colored next. This process lowers the chance of encountering a situation where a grain with a large number of neighbors cannot be colored because of the number of different colors its neighbors use. If no grain can be picked (the grain has no uncolored neighbors), another uncolored grain is picked from the set of all grains. The algorithm stops when no uncolored grains remain.

Figures 4.2 and 4.3 demonstrate the process of coloring grains. The numbers in each grain represent the number of the grain's neighbors. The example first tries to use blue color. If at least one of the grain's neighbors is blue, then the example tries green, then yellow, and lastly red.

The testing of the new algorithm has shown that the maximum number of displayed colors does not exceed the limit of 7, even on the least favorable datasets, such as Project 1. Both the spatial and time complexity of the algorithm have improved as well.

An example of Projects 1 and 2 colored using this algorithm can be observed in Figure 4.4. Project 3 data is very noisy, which leads to huge amounts of grains that do not touch each other. This results in a map where most of the grains are colored using the first color in the list, light blue (R: 1E, G: 90, B: FF). In order to better distinguish grains in similar maps, meaning those with many unindexed points found at grain boundaries, I recommend coloring by either area or aspect ratio, which leads to better results since more colors are used. This comparison is clearly visible in Figure 4.5. Also, a few applications of

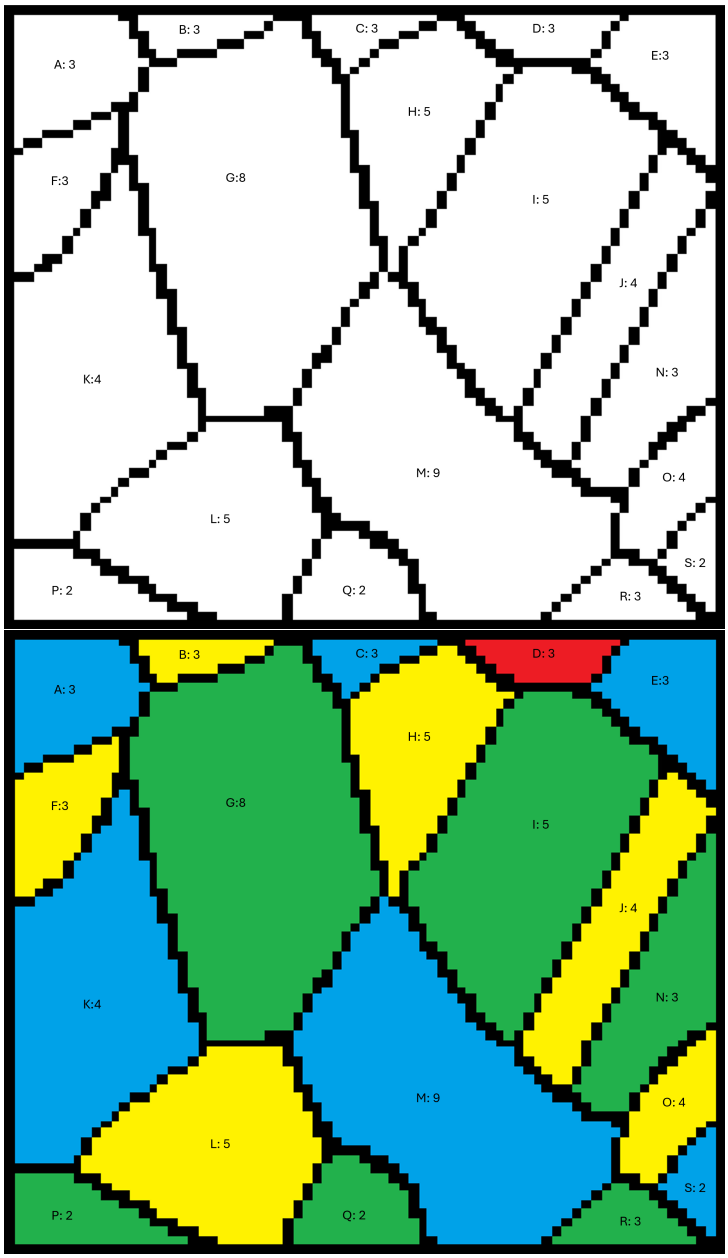

Figure 4.2: A simplified cutout of grain shapes obtained from Project 2 before and after the application of the "Standard color assignment" algorithm.

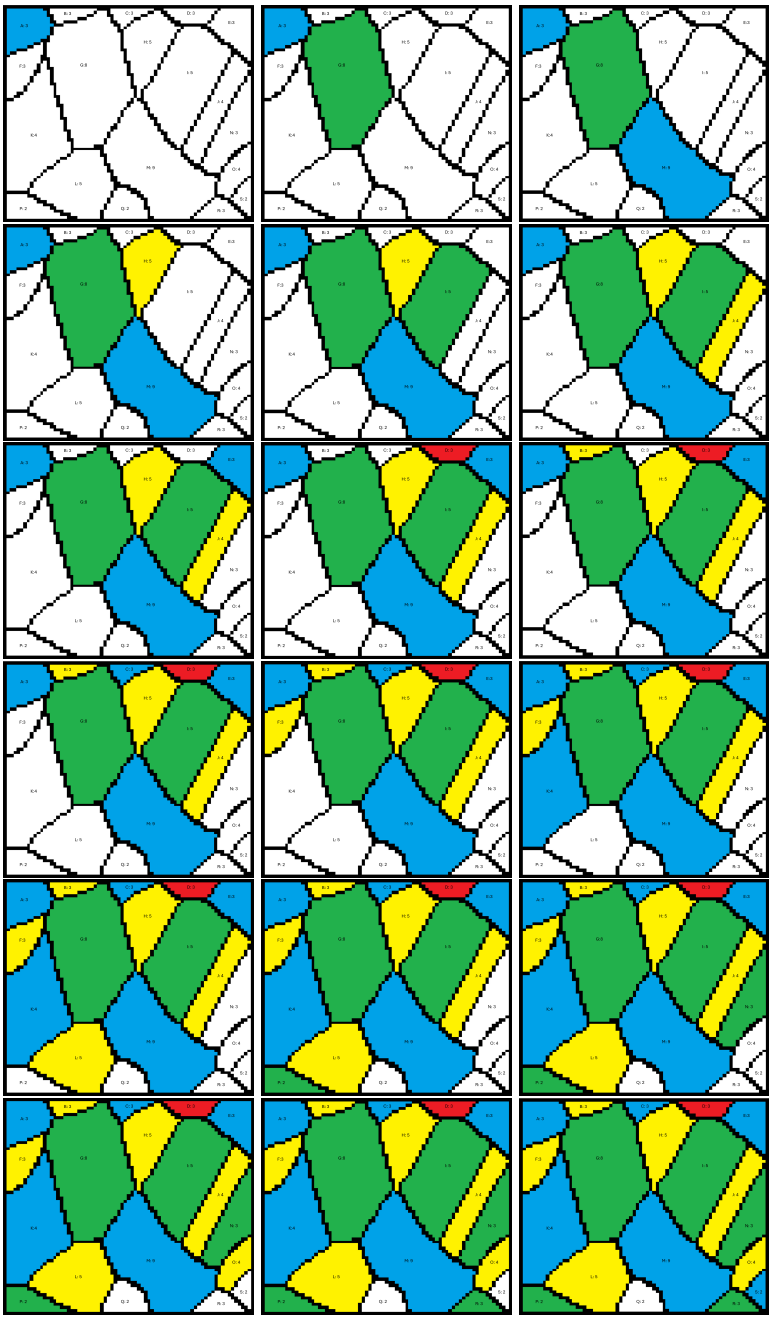

Figure 4.3: The process of coloring grains demonstrated on a simplified cutout obtained from Project 2. The images are sorted from left to right and from top to bottom.

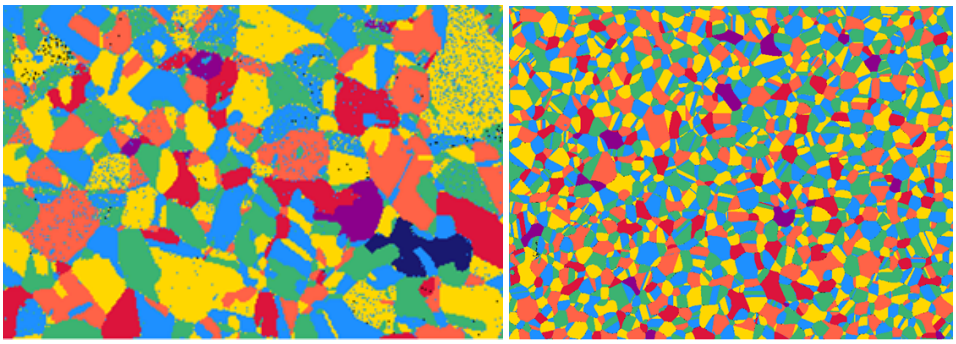

Figure 4.4: Maps of Projects 1 and 2 as calculated using "Standard" coloring type and no grain area correction. Boundary misalign was set to 15 ° for Project 1 (left) and to 5 ° for Project 2 (right).

the "Fill simple gaps" algorithm with carefully selected parameters might help by removing some of the unindexed points between grains.

4.4 Coloring by properties

The "Coloring by properties" algorithm's specification is set by Requirement 2.2. The user may select different grain coloring types, scales, and methods. The combination of these determines the algorithm that is chosen for the map visualization.

Firstly, if the "Standard" coloring type is selected, the comboboxes for scale and method are disabled, and their values are not taken into account during the map visualization. The colors used in the bitmap creation are those assigned by the "Standard color assignment" algorithm.

Secondly, if the "By Area" or "By Aspect Ratio" type is selected, the other comboboxes become available to the user, and based on their settings, a scale and a method are passed onto the coloring algorithm.

If the coloring method is set to "Binned," the algorithm creates bins of grains that have the same value of the set parameter (area or aspect ratio). These bins are then sorted by the value of the parameter, which makes it possible to get a number x for each bin that represents a value on a scale from 0 to 1 like so:

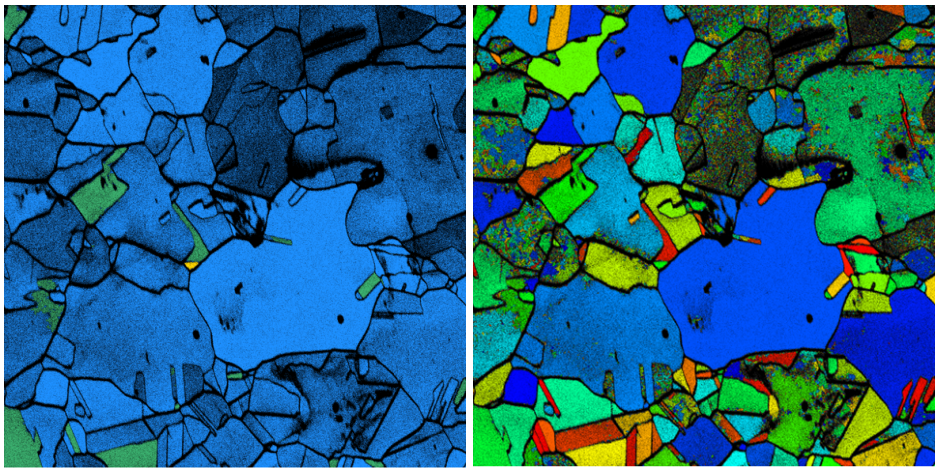

Figure 4.5: Comparison of various coloring settings on Project 3. The project on the left is calculated with boundary misalign set to 5 °, coloring type set to "Standard" and no grain area correction applied. On the right the same map with same parameters is calculated, but the coloring type is set to "By Aspect Ratio," coloring scale to "Rainbow" and coloring method to "Binned."

$$x = \frac{i}{n-1}$$

where *i* is the index of the bin and *n - 1* is the number of all bins.

The coloring method "Continuous" skips the creation of bins and instead finds minimum and maximum values out of all grains. Then, for each grain, the resulting number *x* is calculated using this formula:

$$x = \frac{v-min}{max-min}$$

where *v* is the value of the parameter for the specific grain, *min* is the minimum present value, and *max* is the maximum present value.

In both cases, the resulting *x* number is then passed to a method selected by the grains map coloring scale parameter and processed into an RGB value. This value sets the color used for the grain.

Lastly, I created two coloring scales that the user can use: "Blue" and "Rainbow." During the scale design, I first prototyped the "Rainbow" scale since the pictures accompanying the assignment notes used one. After presenting the coloring scale and method drafts to crystallography experts, the "Rainbow" scale was approved, and a request was made to add another monochromatic scale, resulting in the formation of the "Blue" scale. The "Blue" scale is represented by a gradient from blue (defined as RGB (0, 0, 255)) for the minimum value to white (RGB (255, 255, 255)) being the highest. In contrast, the "Rainbow" scale's gradient starts at blue (RGB (0, 0, 255)), which slowly morphs into green (RGB (0, 255, 0)), then to yellow (RGB (255, 255, 0)) and ends in red (RGB (255, 0, 0)) as the maximum value.

The "By Aspect Ratio" has one exception to the coloring: some grains do not have an aspect ratio. These are both the grains that are either too small or affected by the ASTM E112 standard, leading to them being omitted from the wider property calculation. For the purpose of differentiation, these grains are drawn in gray.

4.5 Legends generation

The generation of legends resolves Requirement 2.3. The "By Area" and "By Aspect Ratio" coloring types use a property to determine a color for each grain, meaning that they should be supported by a legend. The legend can be divided into two parts: color bar and text.

The creation of a color bar is simple. If the "Blue" coloring scale is selected, the color bar is created as a bitmap containing a gradient from blue to white, with blue being at the bottom. In the case of the "Rainbow" coloring scale, the size specifications stay the same. However, the gradient starts at blue, then morphs into green, then into yellow, and ends in red.

The text is more complicated since it is affected by the coloring type and selected units. Additionally, both black and white variations must be created.

All legends use the same formula. The title of the visualized property is placed on top, followed by the label containing the maximum value found in all grains. Afterward, the color bar is displayed, and at the very bottom lies the minimum value. The maximum and minimum values are always rounded to at most two decimal places.

The four legends are all variations of the previously described formula; what changes is the color of the text and the values themselves. The four variants are as follows:

a) Legend for UI in pixels: the text is white, and the values are displayed as kept by the library.

b) Legend for report in pixels: the text is black, and the values are displayed as kept by the library.

c) Legend for UI in micrometers: the text is white, and the values are recalculated if they are dependent on units (applies to area) or kept the same (applies to aspect ratio).

d) Legend for report in micrometers: the text is black, and the values are recalculated if they are dependent on units or kept the same.

All legends are saved to files specified by the input parameters. The difference between specific legends is shown by Figure 4.6.

4.6 Highlighting grains

This section describes the solution for Requirements 3 and 3.1. This algorithm is used when the user selects a grain in the table or clicks

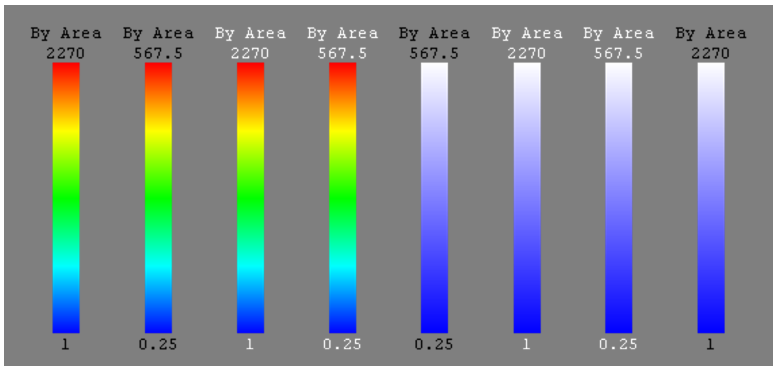

Figure 4.6: Legends generated for Project 2 displayed over gray background. Boundary misalign was set to 5 °, grain coloring type to "By Area," grain coloring method to "Binned" and no correction was used.

on a point in the grain map. In the latter case, the grain must be identified first. For this purpose, the application maintains an object called GrainsMap, in which all points bear a reference to a grain that they belong to. This structure used to be in the code even before the rework, but instead of a reference to the grain itself, the points only kept an ID of the grain, which was then used as a key to search in a dictionary of grains kept by the GrainsMap. This felt redundant and unnecessary, so I decided to rework the class.

Once a point has been selected, the grain that should be highlighted is obtained via the GrainsMap structure. Afterwards, the algorithm checks whether another grain was highlighted right before this action and, if so, repaints its area in the map to its original color. Lastly, all of the newly selected grain's pixels are redrawn in the highlight color.

The choice of a highlight color was not trivial. I could not use black since black pixels indicate unindexed points. White was also not an option since it is used as a color of valid grains when coloring with the "Blue" coloring scale. Gray signifies valid grains that do not have aspect ratio data if the "By Aspect Ratio" coloring type is selected.

Since the coloring scales used for any other coloring method than "Standard" are computed using RGB values, adding or removing a constant RGB value is not applicable as it could lead to a highlighted

grain having a color that is visually close to its neighbor, making it less noticeable.

The solution that I finally chose was to use a deep red color (RGB (80, 0, 0)) to signal that a grain is highlighted. Since it was one of the colors I originally chose for the "Standard" grain coloring type, it is visually discernible from other colors that are used in this scenario.

If the "By Aspect Ratio" or "By Area" options are selected, two coloring scales might be used. The "Blue" scale has a constant blue factor of 255, and both red and green factors are always increased by the same amount, ensuring that the resulting color is either white or always the same shade of blue that only varies in brightness, making the occasional deep red shade stand out. The values of the "Rainbow" scale are calculated in a way that ensures that at least one of the RGB factors is always equal to 255; therefore, the highlight shade is never close enough to any of the scale values to visually blend. The difference in colors is shown in Figure 4.7.

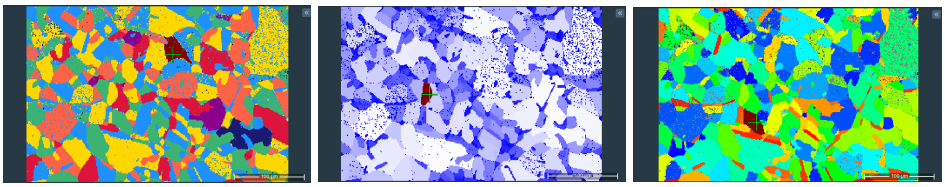

Figure 4.7: Project 1 calculated with different coloring settings. Screenshots from the application. Left to right: "Standard" coloring type, "Blue" coloring scale, "Rainbow" coloring scale.

4.7 Grain filtering

Demands connected to filters are set in Requirement 4. Once the grains are calculated and visualized, filtering is unlocked for the user. The first step in filtering is the selection of the property the user wants to filter on. This is realized using a combobox containing every property that is shown in the grain table. Once a property is selected, the filter settings with default values appear and the user can change them. If

the user clicks on the "Add filter" button, a filter is constructed with the current values of the input fields and added to a list.

Every time a filter is added, changed, or deleted, all grains are filtered, leading to the visualization of filtered-out grains in black, while those that satisfy all conditions are colored with respect to the grain coloring settings.

The existing filters are kept in a list below the module for filter creation and can be deleted by clicking on the trashcan icon beside them or updated by clicking directly on them. After a filter is selected to be updated, the same filter settings appear below the list, but this time the property selection combobox is missing. The user can manipulate the data in the settings. The filter update is finalized when the user clicks the "Update filter" button.

Since the app should support multiple filters on the same property, it was necessary to assert the Boolean algebra formula that would resolve them. After consulting the topic with the application specialist we came to an agreement that multiple filters of the same property should be resolved using logical disjunction, whereas filters of different properties should be evaluated using logical conjunction.

Figure 4.8 shows an example of multiple filters that are applied to the map, table, and statistics. The first two numeric filters are both of the same property, area, so logical disjunction is applied to them. The result of this disjunction is then evaluated in conjunction with the last filter, the exact fit filter on the property phase. The outcome of this evaluation is that only the grains that have an area between 20 to 50 pixels or 500 to 1,000 pixels and at the same time have the phase "Iron (Im3m)" are kept, while the other grains are omitted from the map, table, and statistics. In the case of this map, this effectively filters out the grains that have either an area below 20 pixels, between 51 and 499 pixels, or above 1000 pixels, as well as those that have the Phase "Iron (Fm3m)."

4.7.1 Numeric filters

Numeric filter settings consist of a double range slider and two numeric inclusivity comboboxes. The values of the sliders are reflected in the two numeric inputs and are linked; therefore, the user may affect both the low and high boundary values of the filter by pulling the

Figure 4.8: Screenshot of Project 2 with multiple filters applied.

sliders or by manually writing numbers into the input fields. The inclusivity settings are presented in the form of a combobox containing only the values "$<$" and "$\leq$." The resulting filter then clearly describes the exact requirements that a grain has to meet in order to be kept in the table, map, and statistics.

Since most of the properties use float values, it was necessary to minimize the appearance of rounding errors. This proved to be quite difficult when the switching of units was applied to filter visualization since some of the values are integers in pixels but decimals in micrometers.

In the case of the number of neighbors and grain ID, it is sufficient to simply round the value of the user input to an integer, as the property values are not dependent on the choice of units. These values are displayed with no decimal places.

The solution to the rounding error for area is to limit the numbers the user may input. If pixels are selected, the user may only input whole-number values. In the case of micrometers, the value is corrected to the closest multiple of the map step size. Area in pixels is rounded to zero decimal places.

A similar method can be used for the equivalent circle diameter, which is not integer in pixels, but it is calculated using a formula that takes the pixel value as input. Simply reversing the formula and applying it to the user input results in area in pixels, which is an integer. This value can then be rounded and used in the original formula, finally leading to a new, valid equivalent circle diameter value. In the case of micrometers, the value that comes from the reversed formula must be first divided by the map step size before being rounded to an integer value. In this case, it is best to display the value with three decimal places.

Other values are tricky since they do not have clear limits; therefore, I have decided to round them to two decimal places, which eliminates most of the problems.

4.7.2 Exact fit filters

The exact fit filter settings consist of two comboboxes: one contains the values that have been found throughout all grains, and the other signifies whether the value is negated using "=" or "$\neq$." Since the values are predefined, it is not necessary to make further corrections.

4.8 Binary import and export

Binary import and export follow Requirement 5. As the grain definition changed throughout the coding process, a need for compatibility with older projects was expressed. This led to the creation of four methods for four formats used during development. Based on a header found in the binary file, the application chooses how to interpret the data.

The import and export algorithms use the BitConverter class [29] to convert data to a compressed format. Most of the data kept in the objects of the Grain type is exported, with the exception of the list of neighbors since these are only needed for coloring and property

calculation. Also, BitConverter is unable to export null values, leading to issues with the export of the grain properties that are not present in all grains, such as axes ratio or ellipse angle. I have solved this issue by using double.NaN for the values that are otherwise considered null.

Once the grains, along with the bitmap, are loaded, it is simple to reconstruct the GrainsMap object by initializing it with the dimensions of the bitmap and filling it with the data obtained from the imported Grain objects.

If either the loading of the binary file or the bitmap fails, both are reset internally, and the software behaves as though the map has not yet been calculated, forbidding the user to use filters, grain ellipses etc.

There is only one version of binary export of grains since export of the older formats is no longer required. The binary grain files that are created in the current version of xTalView bear the version number 1.03.

4.9 Denoising

This algorithm follows the specifications of Requirement 6.1. It first creates an internal filter in order to sift through grains and discover those that are smaller than the size in pixels the user has provided as input. Then, the multiphase map is overwritten in all points that belong to grains that were filtered out. The new value signifies that the pixel has been unindexed by denoising.

Lastly, in the released version of xTalView, the calculation process is ran on the updated multiphase map, and new data is presented to the user. This was an oversight on my side that was caused by the time press since there is no need to run the whole calculation process as whole grains have been removed and no new information was present to the remaining valid grains; therefore, they do not need to be recalculated, as their properties remain unchanged. I did not manage to implement this change in time for this thesis, but the current algorithm generates valid data nonetheless. The improvement is planned for a future version of the library.

There are two buttons that invoke this algorithm: "Calculate from current data" and "Calculate from original data." Both of these run the

same algorithm, with the only change being that the second named first resets the multiphase map to the copy of the original one, annulling all previously finished denoise operations.

After the Denoising algorithm has finished, its result description is added to a side panel. This description contains the amount of newly unindexed pixels. If the algorithm was initialized using the "Calculate from original data" button, the side panel is reset before the description is added.

4.10 Fill simple gaps

"Fill simple gaps" is a method that resolves Requirement 6.2. The purpose of this algorithm is to allow the user to add data to the map so that the resulting map has fewer unindexed "gaps" inside of grains.

For each grain, the algorithm inspects a rectangular area that covers the whole area of the grain and a pixel-sized border from all sides. If an unindexed pixel is found, the algorithm examines its eight neighbors and checks whether they belong to the inspected grain. If the number of such pixels reaches at least the threshold specified by the "aggressiveness" setting, the pixel is then assigned the values of one of the compliant neighbors.

After the iteration of all grains has finished, the "Grain calculation" algorithm is then applied to the new multiphase map. This is necessary since the pixels have not yet been linked to their respective grains in the Grain and GrainsMap structure and with their addition, some of the grains' properties are changed; therefore, the statistics must be recalculated. In some cases, it is even possible that by indexing a single pixel, some grains might get connected because the unindexed point might have divided an originally single grain that has not been scanned properly and, therefore, also not established properly. Figure 4.9 shows a situation where the colors of the map are shifted because some grains in other parts of the map were connected.

A phenomenon has started appearing: after removing grains with an area smaller than 4 pixels and running the "Fill simple gaps" algorithm, the statistics have reported that the minimum grain area found in the map is 1 pixel. This was happening because the "Fill simple gaps" algorithm searches the 8-pixel neighbors, whereas the

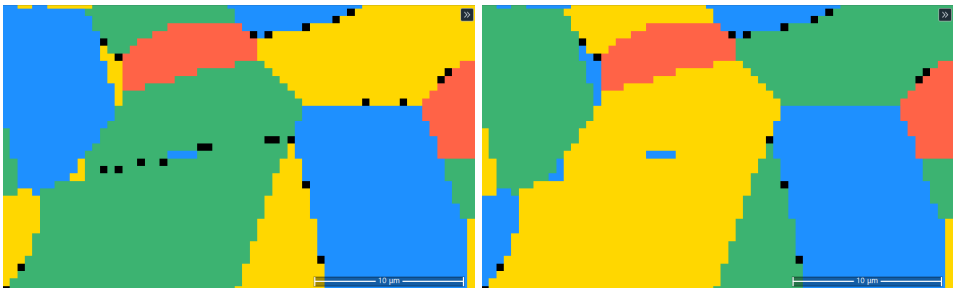

Figure 4.9: Cutouts of Project 2. On the left, the "Denoise" operation was applied with boundary misalign set to 5, coloring type to "Standard," no correction and the minimum grain size set to 4 pixels. On the right, the "Fill simple gaps" operation was applied with the same common parameters and the "aggressiveness" parameter value set to 5.

"Calculate grains" algorithm only looks at four pixels. This has led to situations in which a pixel has been assigned a value of one of its corner neighbors. The problem appears when its orientation is too different from all of the direct neighbors that appear beside, above, or below the pixel.

I figured out that the solution to this problem is to investigate the neighboring pixels further. If at least the specified amount of neighbors is found, the algorithm then looks through the neighbors that meet the conditions, and if it finds one that is directly above, below, or next to the unindexed pixel, the value is copied to the pixel. Otherwise, even though the pixel has the specified number of compliant neighbors, it is kept unindexed.

4.11 Bilateral filter per grain

The specifics of this algorithm were set in Requirement 6.3. As was previously stated, the former implementation worked well and there was no need to improve the algorithm; therefore, I only updated some of the method calls and parameters so that the method would work with the new class definitions and other changes.

4.12 Statistics calculation

The "Statistics calculation" algorithm refers to Requirement 7. Whenever the information in the grain table changes, statistics need to be recalculated so that consistent data is shown at all times. The data from the table serves as the input to the algorithm.

The calculation process itself is quite straightforward, since the statistic functions are predefined. After discovering the minimum, maximum, average, median, and standard deviation for all statistically relevant numeric properties, I had to come up with a way of dealing with grains that do not have the ellipse-connected data. The most logical solution was to omit these grains completely from the calculation of the properties in question, but to consider them for the remaining properties.

5 Feedback

I have asked the people that I consulted the most throughout the writing of the project to write an assessment of my work, namely Jakub Holzer, Ph.D., Ing. Michal Křen, and Austin Day, Ph.D. They have been kind enough to write a few paragraphs for me to include into this thesis.

"Grain detection plays a crucial role in crystallography, with its findings being extensively applied in the field of material science. The size, shape, and orientation of grains directly impact the material's properties, enabling predictions of performance under stress. Lenka revitalized an outdated codebase by refining the existing flood-fill algorithm and enhancing it with numerous new features. These included coloring based on various grain parameters and calculating grain size according to different engineering standards. A significant part of her contribution was the development and integration of filters, where she proposed the logic and seamlessly incorporated the feature into the software's framework.

Lenka collaborated closely with technical, application, and marketing teams, consistently delivering and surpassing the initial goals through several iterations. Her diligence, organization, and hard work have yielded outstanding results, making her a pleasure to work with. The outcomes of her efforts will benefit material scientists globally."

Jakub Holzer, Ph.D.
Applications Scientist, Thermo Fisher Scientific Brno s.r.o.

"As the Software Leader for the xTalView project, I am thoroughly impressed with Lenka's bachelor thesis work and its practical implementation. She successfully developed and integrated a comprehensive Grain Library into our commercial xTalView application, which demonstrates both her technical capabilities and professional approach to software development.

What stands out most is that Lenka created an entirely new algorithm for grain reconstruction from scratch, showing deep understanding of both programming principles and crystallographic concepts. The field of crystallography is notably complex, requiring extensive study and understanding of specialized concepts, which Lenka managed to master alongside her development work.

The technical implementation was done using C# Framework 4.7.2 to ensure compatibility with existing systems, but the architecture was thoughtfully designed to allow future migration to .NET 8 and potential transformation into a microservice with gRPC interface. This forward-thinking approach significantly enhances the library's long-term value.

Particularly noteworthy is that her implementation resolved approximately 20 existing issues, including both bugs and missing features, which substantially improved the application's functionality and reliability. This achievement became crucial for our March 2025 release of xTalView, marking a significant milestone after six years of development. Despite the intense pressure to meet deadlines and deliver features, Lenka maintained her composure and successfully delivered a working library.

Throughout the project, Lenka demonstrated excellent teamwork skills, effectively communicating with various stakeholders including the project manager, application specialists, technical leaders, developers, and QA team. Her ability to collaborate and integrate feedback from multiple sources was instrumental in the project's success.

Given her exceptional performance and deep understanding of crystallography, combined with strong software development skills, I sincerely hope Lenka will continue her career in the crystallography domain. Her contribution to our project has been invaluable, and she has shown great potential for future growth in this specialized field."

Ing. Michal Křen
Team Leader, Branch Manager, Edhouse s.r.o.

"Grain size measurement is an important application of the EBSD technique, it has a long history and there have been many implementations.

Lenka has vastly improved the existing code & algorithms, particularly in terms of reliability, speed and versatility; as well as adding novel approaches to grain reconstruction, colouring & filtering, and to filling in missing / incorrect data.

I have been impressed by the rapidity with which she has grasped the EBSD technique and crystallography, and the clarity of her algorithm explanations & presentations. I also enjoyed reading her excellent Bachelor's thesis.

Lenka's willingness to respond to feedback and to cope with changing specifications / requirements have been exemplary, as have the speed at which changes are implemented.

The new features & improvements will be widely appreciated & used by Thermo EBSD customers."

Austin Day, Ph.D.
EBSD Technical Expert & Consultant, Aunt Daisy Scientific Ltd.

6 Conclusion

The aim of this thesis was to improve the pre-existing algorithms and implement new functionalities for grain detection and manipulation in the xTalView application in the form of a library.

I described the basic concepts connected to grains. I explored the required functionality in AZtecCrystal and EDAX OIM and I summarized the previously used implementation in xTalView. I closely specified the requirements set by crystallography and software engineering experts, whom I consulted throughout the process of writing this thesis. I described my addition to the xTalView software and compared it to the previously used methods. The library and its implementation were tested by the QA team and included in the commercially released version of xTalView.

The contribution that the library brought to the Thermo Fisher Scientific Brno s.r.o. company is evident from the reviews in Chapter 5. The library fulfilled all specified requirements and succeeded expectations since the new algorithms are not only functioning correctly, but also, for many of them, their time complexity has improved, or they can newly be ran in parallel.

There are future plans in place regarding grain analysis in xTalView, which might become a part of this library, such as improved grain boundary analysis. The library is written in a way that supports this decision.

Bibliography

[1] R. Gwyn. "Properties and grain structure," BBC. (1973), [Online]. Available: https://www.youtube.com/watch?v=uG35D_euM-0 (visited on 04/09/2025).

[2] Y. Picard *et al.*, "Future prospects for sem-based defect analysis using fast electrons," *Microscopy and Microanalysis*, vol. 16, pp. 608–609, Jul. 2010. DOI: 10.1017/S1431927610063348.

[3] S. Hassanzadeh-Tabrizi, "Precise calculation of crystallite size of nanomaterials: A review," *Journal of Alloys and Compounds*, vol. 968, p. 171 914, 2023, ISSN: 0925-8388. DOI: https://doi.org/10.1016/j.jallcom.2023.171914. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925838823032176.

[4] "What is microstructure?" (2025), [Online]. Available: https://www.ebsd.com/ebsd-for-beginners/what-is-microstructure (visited on 04/08/2025).

[5] D. Weinhandl. "What is annealing in metal?" Mead Metals, Inc. (2023), [Online]. Available: https://www.meadmetals.com/blog/what-is-annealing-in-metal (visited on 04/09/2025).

[6] K. Mueller. "File:comparison solar cell poly-si vs mono-si.png." (2014), [Online]. Available: https://commons.wikimedia.org/wiki/File:Comparison_solar_cell_poly-Si_vs_mono-Si.png (visited on 05/14/2025).

[7] I. Beyerlein, M. Demkowicz, A. Misra, and B. Uberuaga, "Defect-interface interactions," *Progress in Materials Science*, vol. 74, pp. 125–210, 2015, ISSN: 0079-6425. DOI: https://doi.org/10.1016/j.pmatsci.2015.02.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0079642515000225.

[8] D. I. C. Findeiss. "Astm e112." (2025), [Online]. Available: https://endolab.org/mt-metal-analysis/astm-e112/ (visited on 04/09/2025).

[9] "What is electron backscatter diffraction (ebsd)?" (2025), [Online]. Available: https://www.ebsd.com/ebsd-explained/what-is-ebsd (visited on 03/27/2025).

[10] A. J. Schwartz, M. Kumar, and B. L. Adams, *Electron backscatter diffraction in materials science*, 1st ed. New York: Kluwer Academic/Plenum Publishers, 2000, xvi, 339, ISBN: 0-306-46487-X.

[11] W. Massa, *Crystal structure determination*, 2nd ed. Berlin: Springer, 2000, xi, 206, ISBN: 3-540-65970-6.

[12] "How does ebsd work?" (2025), [Online]. Available: https://www.ebsd.com/ebsd-for-beginners/how-does-ebsd-work (visited on 04/08/2025).

[13] D. Prior *et al.*, "The application of electron backscatter diffraction and orientation contrast imaging in the sem to textural problems in rocks," *American Mineralogist*, vol. 84, pp. 1741–1759, Nov. 1999. DOI: 10.2138/am-1999-11-1204.

[14] S. Vespucci *et al.*, "Digital direct electron imaging of energy-filtered electron backscatter diffraction patterns," *Physical Review B*, vol. 92, Nov. 2015. DOI: 10.1103/PhysRevB.92.205301.

[15] "Azteccrystal." (2025), [Online]. Available: https://nano.oxinst.com/azteccrystal (visited on 04/08/2025).

[16] "Oim analysis." (2024), [Online]. Available: https://www.edax.com/products/ebsd/oim-analysis (visited on 05/06/2025).

[17] "Aztechkl." (2025), [Online]. Available: https://nano.oxinst.com/products/aztec/aztechkl (visited on 05/06/2025).

[18] "Apex software for ebsd." (2024), [Online]. Available: https://www.edax.com/products/ebsd/apex-software-for-ebsd (visited on 04/09/2025).

[19] "Desktop guide (wpf .net)." (2024), [Online]. Available: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-6.0 (visited on 04/26/2025).

[20] S. I. Wright and M. M. Nowell. "Microstructural characterization of additive manufactured materials via ebsd," EDAX. (2021), [Online]. Available: https://www.youtube.com/watch?v=K9QBDOf4wL4 (visited on 04/19/2025).

[21] "Advanced ebsd data processing with oim analysis - data selection, validation, and quantification," EDAX. (2021), [Online]. Available: https://www.youtube.com/watch?v=2B6JRVah5nw (visited on 04/21/2025).

[22] "Azteccrystal online training: Maps," Oxford Instruments. (2020), [Online]. Available: https://www.youtube.com/watch?v=4elj0819Bvw (visited on 04/24/2025).

[23] "Oim analysis: Grain size and grain structure information," EDAX. (2021), [Online]. Available: https://www.youtube.com/watch?v=qZm1ghqPBdA (visited on 04/19/2025).

[24] "Ebsd datasheet." (2024), [Online]. Available: https://assets.thermofisher.com/TFS-Assets/MSD/Datasheets/truepix-ebsd-detector-apreo-chemisem-ds0511-en.pdf (visited on 04/28/2025).

[25] "Azteccrystal online training: Grain analysis," Oxford Instruments. (2020), [Online]. Available: https://www.youtube.com/watch?v=qNEJhjjxWac (visited on 04/24/2025).

[26] "Azteccrystal online training: Using subsets," Oxford Instruments. (2020), [Online]. Available: https://www.youtube.com/watch?v=Xs-0cmJQl4Y (visited on 04/24/2025).

[27] "Azteccrystal online training: Cleaning and restoring projects," Oxford Instruments. (2020), [Online]. Available: https://www.youtube.com/watch?v=3tUBxgq-pYs (visited on 04/24/2025).

[28] "Four-color theorem." (1999–2025), [Online]. Available: https://mathworld.wolfram.com/Four-ColorTheorem.html (visited on 03/21/2025).

[29] "Bitconverter class." (2025), [Online]. Available: https://learn.microsoft.com/en-us/dotnet/api/system.bitconverter?view=net-9.0 (visited on 04/27/2025).

A Attached files

The ZIP archive attached to the thesis contains some of the code I wrote for this thesis. The included files are the GrainColorProvider.cs file, whose methods return colors used for grain coloring, and the Filter folder, whose files implement the two types of filters described in the thesis. These files have been selected as a representation of the code I wrote for the thesis. The other files present in the library are not published because of the trade secret.