A Novel Method for Finding Overlap Depth: Development of ArtiaX, a ChimeraX plugin

Department of Mathematics, Linköping University

Gunnar Arctaedius

LiTH-MAT-EX–2023/07–SE

Credits:	**30 hp**
Level:	**A**
Supervisor:	**Fredrik Berntsson**, Department of Mathematics, Linköping University
Examiner:	**Andrew Ross Winters**, Department of Mathematics, Linköping University
Linköping:	**June 2023**

Abstract

Visualization and image filtering are important parts of cryo-electron tomography analysis. ArtiaX, a plugin developed for UCSF ChimeraX, has been extended to improve the functionality of these two parts. For the visualization, a method of moving 3D surfaces to remove overlap between them has been developed and implemented. To accommodate this, a Monte Carlo approach using Poisson disc sampling for approximating volume of overlap between 3D surfaces is used, and a novel method for measuring overlap has been invented, called the Normal Projection Method, useful for measuring the depth of overlap between surfaces. For the image filtering, tomogram averaging and frequency filters have been added to the ArtiaX toolbox.

Keywords:
Cryo-electron tomography, image processing, surface visualization, overlap measurement, normal projection method.

URL for electronic version:
`http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-195970`

Sammanfattning

Visualisering och bildfiltrering är viktiga delar inom kryoelektrontomografianalys. ArtiaX, ett plugin utvecklat för UCSF ChimeraX, har utökats för att förbättra funktionaliteten inom dessa två områden. För visualiseringen har en metod för att flytta 3D ytor så att de inte överlappar utvecklats och implmenterats. För att skapa denna funktion används en Monte Carlo metod med Poisson disk sampling för att uppskatta volymen av 3D ytor, och *normalprojektionsmetoden*, en ny metod för att mäta överlappsdjup har skapts och implementerats. För att förbättra bildfiltreringen har tomogram-snittning och frekvensfilter lagts till bland verktygen i ArtiaX.

Nyckelord:
Kryoelektrontomografi, bildbehandling, ytvisualisering, överlapsmätning, normalprojektionsmetoden.

URL för elektronisk version:
`http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-195970`

Acknowledgements

The work was done in collaboration with the Frangakis Group at the Buchmann Institute for Molecular Life Sciences at Goethe University Frankfurt and the Department of Mathematics at Linköping University. The development is based on the ArtiaX plugin, developed by the Frangakis group. Thanks to head of the group, Achilleas Frangakis, for the opportunity and help with focusing the work. Special thanks go to Utz Ermel of the group for much mentoring and help understanding the code, as well as help guiding the development. Thanks also go out to my supervisor at the Department of Mathematics, Fredrik Berntsson, for helping me develop some of the methods, and Andrew Ross Winters, my examiner, for helping focus the work. The plugin is based on UCSF ChimeraX, developed by the Resource for Biocomputing, Visualization, and Informatics at the University of California, San Francisco.

Contents

Chapter 1

Introduction

ArtiaX is an open source plugin available for the molecular visualization software ChimeraX, and is intended for the visualization and processing of cryo-electron tomography data[1], two important aspects for research in structural biology [8]. For the intentions of this thesis, the functionality of ArtiaX has been extended. This work embodies two distinct yet interconnected parts, each contributing to the holistic analysis and interpretation of macromolecular structures and cellular organization. By addressing limitations in the processing and visualization of the plugin, the work of this thesis enhances the functionality and applicability of ArtiaX, and by extension, ChimeraX, facilitating more comprehensive investigations in structural biology.

The first part of the thesis focuses on improving the visualization functionality. ArtiaX can be used to render biological scenes on the nano-meter scale by placing and orienting realistic 3D surface models in a larger scene. It has many tools that allow the user to manually place and orient surfaces in their scenes, or automatically generate surfaces using geometrical primitives, such as along a curve in a helical fashion. However, these scenes often contain large quantities of complex surface models in close proximity, as they appear in the real world. Since there is no builtin collision detection in ArtiaX, these surfaces might overlap, creating an unrealistic scene. The goal of this part of the thesis was therefore to create a function that can move selected surfaces the shortest amount possible such that nearby surfaces no longer overlap. The user can place or generate surfaces at will, and then use this new functionality to remove any potential overlap that has occurred, resulting in a more realistic scene. In order to achieve this goal, a novel method for measuring overlap between triangular

[1] ArtiaX is available at https://github.com/FrangakisLab/ArtiaX.

meshes has been developed, and subsequently dubbed the Normal Projection Method.

In the second part, tools to improve the signal-to-noise ratio of cryo-electron tomography data have been developed. These tools focus on tomograms, large 3D images collected from an electron microscope. Tomograms can contain high and low frequency noise, which reduces the resolution of models generated from them [9]. To combat this, the goal for this part of the thesis was to implement filters and averaging tools that can be used to increase the signal-to-noise ratio in the tomograms.

By implementing both of these functionalities, ArtiaX offers an integrated platform for working with tomograms in a simple and efficient way, while also functioning as a tool for rendering state-of-the-art biological scenes. These scenes are also viewable through ChimeraX's virtual reality capabilities.

All the code written for the thesis is in scientific python[2], using the ChimeraX development tools, amongst which there are some functions implemented in C++ that have been utilized as often as possible due to their increased computation speed. The code also builds heavily on the classes and features already present in ArtiaX, using the surface and tomogram frameworks extensively. In addition to the underlying mechanics of the tools added, both a graphical and a command-line user interface are part of the plugin. However, the description of these implementation details is outside the scope of this thesis, and can be found in the ArtiaX documentation.

The development of both parts are completely independent, and as such the thesis is divided up in two parts; one about removing overlap between surfaces in a 3D-scene, and one about filtering tomograms. Both these parts will include sections describing the mathematical background, the implementation, and the results of the implemented tools. After this, a discussion about the limitations and improvements of the added features is presented, along with other potential features to add to ArtiaX. Finally, a short conclusion summarizes the work done and the conclusions drawn from the thesis.

[2]Both numpy [11] and scipy [18] have been frequently used.

Chapter 2

Removing Overlap

When rendering realistic biological environments in 3D, it is important that no two rendered surfaces overlap. ChimeraX, especially with ArtiaX, is a convenient and useful tool for creating naturalistic scenes on a nano-meter scale. But, as there is no builtin collision detection in ChimeraX, a suite of functions have been developed for ArtiaX to move surfaces so that they no longer overlap. To create these functions, two methods have been implemented that calculate overlap between pairs of surfaces. One method generates semi-random points to calculate the volume of overlap, whereas the other method calculates the depth of overlap, two notions that are defined in Section 2.1.1. To completely remove the overlap between all surfaces, an iterative approach is used, where the overlap between all surfaces is first calculated using one of these methods. Using the size of the overlaps, the surfaces are then moved away from each other. This process of measuring overlap and moving the surfaces is then repeated until no two surfaces overlap anymore. In addition to this, an extension to the functions has been developed, that moves surfaces on a larger attachment-surface.

In this chapter, the representation of the surfaces and the definition of overlap between surfaces is first presented in Section 2.1. The theoretical background of the methods employed to remove overlap between surfaces is then presented in Section 2.2, followed by Section 2.3 on the implementation of these functions in ArtiaX. Lastly, the results of the implementation is presented along with comparisons between different methods in Section 2.4.

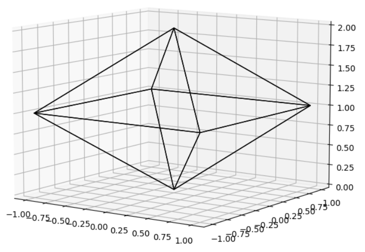


Figure 2.1: A triangle mesh octahedron.

2.1 Representation of Surfaces

Surfaces in ChimeraX are stored and represented as triangular meshes, i.e., a list of vertices and a list of triangles explaining how the vertices are connected. A simple example of this would be an octahedron, which could be described using eight triangles with the following lists:

```
vertices  = [[0,0,0], [1,1,1], [1,-1,1], [-1,1,1], [-1,-1,1],
            [0,0,2]].
triangles = [[0,1,2], [0,1,3], [0,2,4], [0,3,4], [1,2,5],
            [1,3,5], [2,4,5], [3,4,5]],
```

where the vertices list describes six points in 3D space and the triangles list describes eight triangles, connecting the vertices using their indices in the list of vertices. Plotting this produces Figure 2.1. ChimeraX uses this format to store and represent surfaces to the user.

2.1.1 Definition of Overlap

Two surfaces are considered overlapping if any of the triangles from one surface intercepts any of the triangles of the other surface. In this thesis, two measures of overlap are considered, namely, *overlap volume*, and *depth of overlap* (or, *overlap depth*). Depending on the implementation, both measures could be used for any surface geometry, but the theoretical foundation of the measures

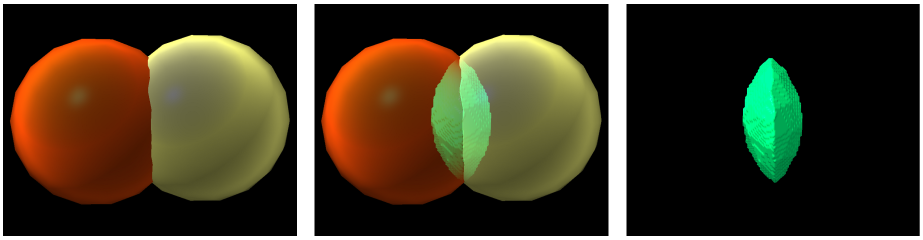


Figure 2.2: Two spheres overlapping, with the intersection highlighted in green. The volume of this intersection is the overlap volume of the two spheres.

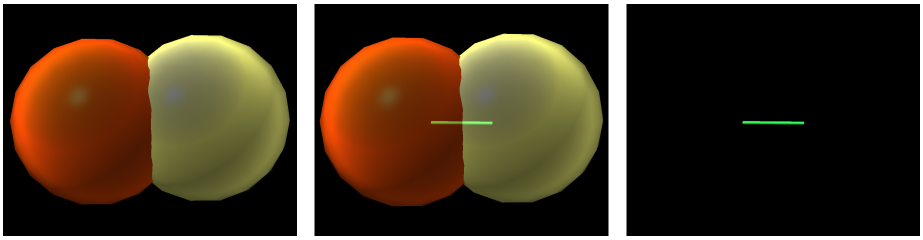


Figure 2.3: Two spheres overlapping, with a line whose direction and length indicate the shortest distance the spheres could be moved so that they no longer overlap. This distance and direction compose the depth of overlap between the two spheres.

rely on continuous, hole-less, surfaces at the point of overlap. The definition, as used in this thesis, of both measures is as follows:

Definition 1. *The **overlap volume**, V_O, between two surfaces A and B, is defined as*

$$V_O = V_{A \cap B}, \tag{2.1}$$

where $V_{A \cap B}$ is the enclosed volume of the intersection of A and B.

Definition 2. *The **depth of overlap** between two surfaces A and B, is defined as the shortest distance, d, either surface can be moved so that they no longer overlap.*

Figures 2.2 and 2.3 shows an example of overlap volume and depth of overlap, respectively. When working with the depth of overlap, it is often helpful to measure the direction along which the shortest overlap occurs.

2.2 Theoretical Background

To remove overlap between surfaces, two methods of measuring the size of the overlap have been implemented. One method to measuring overlap volume and one to measure the depth of overlap, named the *Normal Projection Method*. The mathematical background to these two methods, and a pseudocode algorithm describing the normal projection method is presented in the forthcoming sections.

In addition, a method of iteratively moving overlapping surfaces is presented, along with a mathematical theorem, providing an argument that this method will indeed remove all overlap with enough iterations. Finally, a mathematical description of the method used to move surfaces along an attachment-surface is presented.

2.2.1 Measuring Volume

There are many algorithms available for estimating the enclosed volume of triangular meshes [21, 20, 17]. However, these mostly rely on either voxelization of the mesh (changing the representation from a triangle mesh to a voxel map), or knowing the vertices and triangles of the volume to measure. When measuring the volume of the overlap of two surface meshes, neither of these approaches are preferable, as the vertices and triangles that make up the intersection surface are not known, and finding these is expensive.

Monte Carlo Approach for Measuring Volume

One method that can approximate the volume of the overlap between two surfaces in 3D-space, without needing a full mesh or voxel description of the overlap, is a Monte Carlo approach [16]. It, using some sort of random process, generates points within the bounding box of the surface and then calculates the proportion of the generated points that are inside the surface. From this proportion and the volume of the bounding box, an estimate of the volume can be calculated. For a scenario where the volume of the overlap between two surfaces is to be calculated, either of the bounding boxes of the surfaces can be chosen, and the proportion of the generated points that are inside of both surfaces can instead be used to estimate the overlap volume. Described mathematically, the volume of the overlap, V_O, can be calculated by

$$V_O = \frac{n_{A\cap B}}{n} V_{bbox}, \tag{2.2}$$

where $n_{A\cap B}$ is the number of points inside both surfaces, n is the total number of points generated, and V_{bbox} is the volume of the chosen bounding box. To calculate $n_{A \cap B}$, a method is needed for checking whether a point is inside of both surfaces.

The problem of testing whether a point is inside or outside of a triangulated surface mesh is common in computer graphics, so there exists a plethora of methods available for this task [14]. One of the simplest and quickest methods is the *Jordan Curve Theorem-based method* [14], where a ray is cast from the point being tested, to some point outside the volume, such as outside the edge of the bounding box. If the ray intercepts the surface of the volume an even number of times, the tested point is outside the volume. If it intercepts an odd number, it is inside.

With the Monte Carlo approach, where the points are generated using a uniform random distribution, the error obtained from the simulation is proportional to $\frac{c}{\sqrt{n}}$, where $c > 0$ and n is the number of points generated [5]. However, instead of uniformly generated points, the points can be generated using some other scheme, such as Poisson disc sampling [4]. This is a method that produces random points in a sample domain that are never within a set distance r from one another [4], creating points that are more evenly spread than point generated from a uniform distribution. It works by generating points that are within r and $2r$ distance away from set points, and discarding generated points that are too close to existing points, and adding the ones that are not [2]. Using some quasi-random sampling method like Poisson disc sampling could decrease the number of points needed to get the same measurement compared to uniform sampling.

2.2.2 Normal Projection Method for Measuring Depth of Overlap

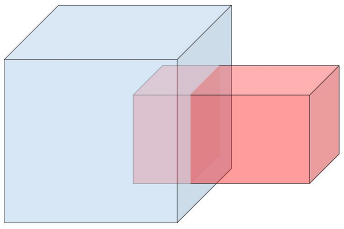
The Normal Projection Method is a computationally efficient method for calculating the depth of overlap between two surfaces in 3D space, developed for the purposes of this thesis. Its goal is to calculate the depth of overlap (see Definition 2) between a pair of surfaces. It does this by first finding the intercept between the surfaces, and then finding a fit for the plane described by the interception points of both surfaces. The normal of this plane is assumed to be the line along which the surfaces can be moved the shortest distance apart to remove overlap between them. All points of the surfaces are projected onto the normal. The furthest distance between two projected points is then measured and returned along with the normal. Algorithm 1 shows a pseudocode interpolation of the method. Figure 2.4 shows the steps of the method graphically, using two overlapping boxes as an example.

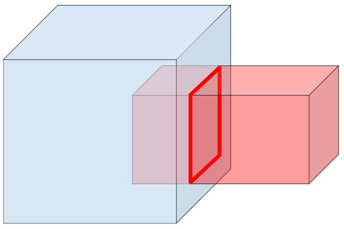
```
input  : Two surfaces, s1 and s2.
output: The depth of penetration of s1 into s2, s1_depth, the depth of
        s2 into s1, s2_depth, as well as the normal along which the
        penetration is measured, normal.
intercept_points ← GetInterceptionPoints(s1, s2);
normal, middle ← GetPlaneGeometry(intercept_points);
Turn the normal so that it faces from the first surface's center to the
  plane;
s1_in_s2 ← GetPointsOnOtherSideOfPlane(s1, normal, middle);
s2_in_s1 ← GetPointsOnOtherSideOfPlane(s2, −normal, middle);
s1_depth ← Max(Norm(middle − ProjectToLine(s1_in_s2, normal)));
s2_depth ← Max(Norm(middle − ProjectToLine(s2_in_s1, normal)));
```

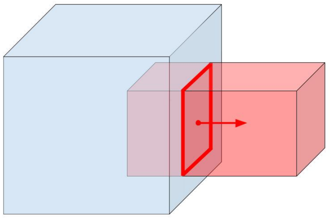
Algorithm 1: A pseudo code implementation of the Normal Projection Method. It relies on many functions to perform geometrical tasks, such as finding interception points between surfaces, fitting a plane to a group of points, sorting points on one side of a plane, and projecting points to a line. The surfaces can be of any representation, such as a triangular mesh or voxels. The function *GetPointsOnOtherSideOfPlane* returns a list of coordinates that can be projected onto the normal.

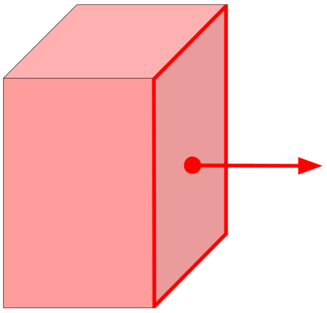
The method doesn't require any certain representation of the surfaces, but has been designed to work well with triangular meshes. It requires a function that can find common points between the surfaces to find the intersection points, which can be done in a number of ways, such as a nearest neighbor search. It also relies on a function for fitting a plane to a group of points, such as a least squares method. Finally, it needs a function that can find points from one surface that are on a specified side of a plane. If the surface is a triangular mesh, the vertices on the selected side of the plane can be the points returned. In addition to these specialized functions, it also relies on basic linear algebra calculations, such as projecting points to a line and measuring the length of vectors.

Algorithm 1 is applicable for many shapes, but is not guaranteed to give a correct measure for complex geometries. For spheres it is guaranteed to give a perfect measurement, since the intercept of two spheres is always a circle that lies on a plane which is orthogonal to the line through the centers of the spheres, i.e., the line along which the overlap of the spheres is the shortest [10]. The normal projection method tries to measure the depth of overlap, but it does not always return the shortest distance the two surfaces can be moved to avoid overlap. This is because it assumes that the shortest distance to move the surfaces will be along the normal of the interception plane, meaning that there might be a shorter distance to move the object. However, the normal


(a) Two overlapping boxes.


(b) The intersection points between the boxes is highlighted in red.


(c) The normal of the plane and a point on it has been identified and is presented as the arrow in red.


(d) The portion of the boxes that are not in the other box have been removed. Notice how the blue box is completely removed, since no part of it is present in the red box.

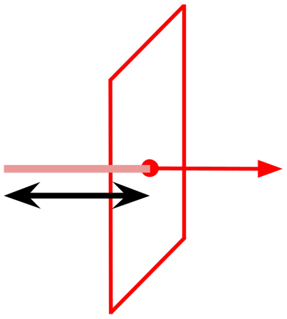

(e) The remaining parts of the boxes have been projected onto the normal (the red arrow). The black arrows indicate the length of the protrusion of the boxes into each other, which, together with the normal, compose the depth of overlap between the boxes.

Figure 2.4: A pictured example of the Normal Projection Method being used to calculate the depth of overlap between two overlapping boxes.

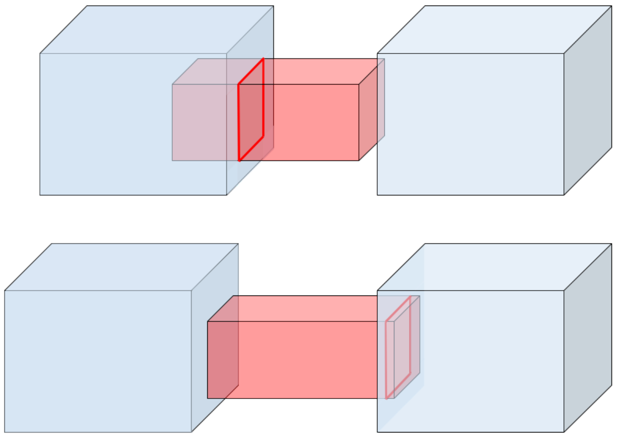


Figure 2.5: An example of how new overlap can occur when moving two surfaces that are overlapping. In the upper picture the left blue box and the red box overlap, and in the bottom picture the left blue box has moved left and the red box has moved right so that they no longer overlap. The red box has however moved into the right blue box, so that they now overlap. The red outline shows where the surfaces intercept.

projection method always finds a length to return, even though it is easy to find examples of concave geometries where the method will fail. For most convex and even many concave geometries the normal of the plane approximated by the interception points will yield a result that moves the surfaces apart.

2.2.3 Iterative Movement

When moving overlapping surfaces, they could be moved into other surfaces, generating new overlaps. Figure 2.5 shows how this can happen when two overlapping surfaces are moved in the presence of a third surface. To ensure no surface pairs overlap, an iterative approach is employed where all surfaces are moved so that their current overlaps are removed, and then moved again to remove any new overlap that has occurred. This process continues until no two surfaces overlap. Algorithm 2 shows a pseudocode implementation of the algorithm which relies on a function that calculates the movements of all surfaces to remove the current overlaps. This function could, for example, use the normal projection method presented in Section 2.2.2 or a function that measures the overlap volume between surfaces, see Section 2.2.1.

```
input: Surfaces, a list of all surfaces in some representation. The
       surfaces need to be movable by the function MoveSurface()
Movements ← CalculateMovements(Surfaces);
while Movements is not all zeros do
    for Surface and corresponding Movement in Surfaces and
     Movements do
        MoveSurface(Surface, Movement)
    end
    Movements ← CalculateMovements(Surfaces);
end
```

Algorithm 2: An algorithm for iteratively moving surfaces until no pair overlaps. It relies on a function *CalculateMovements* that can measure overlap between all surface pairs and calculate movements for each surface that removes the current overlaps.

With Algorithm 2, many surfaces can be moved to produce an end result of a 3D scene with no overlapping surfaces, or at least such a result can be expected, as discussed in the next section.

Iterative movement convergence

When moving surfaces using the iterative movement described in Section 2.2.3, the question arises whether the algorithm is guaranteed to finish, that is, whether after enough iterations, no two surfaces overlap. To provide an argument for why convergence is expected in a simplified version of the problem, Theorem 1 has been developed and proved. Before getting to the theorem, some restrictions need to be described, the function that moves surfaces defined, and a lemma presented.

Let X be a collection of $1 \leq n < \infty$ spheres s_i in $\mathbb{R}^3$ with radii $0 < r_i < \infty$ and center point c_i for any $i = 1, 2, ..., n$. Two spheres are said to overlap if they intercept at more than one point.

Definition 3. *Let $f(X)$ be a function that moves spheres in X based on their overlap. If multiple spheres overlap, $f(\cdot)$ moves the spheres that overlap the most. If more than one pair of spheres overlap the same amount, one of the pairs is chosen. The spheres are moved away from each other along the line that goes through the center of both spheres. Both spheres are moved the same distance so that they only touch in exactly one point. That is, if sphere s_i and s_j overlap, they are moved so that $||c_i - c_j|| = r_i + r_j$.*

Definition 4. *Let $S(X)$ be the sum of the distances of the centers of all spheres* *in X, that is, $S(X) = \sum_{i \neq j} ||c_i - c_j||, \forall i, j = 1, 2, ..., n$.*

Lemma 1. *If $X = \{s_1, s_2, s_3\}$, where s_1, s_2, s_3 are spheres with radii $r_1, r_2, r_3 < \infty$ and s_1 and s_2 overlap, but neither overlap with s_3, then the distance from the center of s_3 to the other spheres will never decrease after $f(\cdot)$ is applied to X.*

Proof of Lemma 1. The spheres can, without loss of generality, be assumed to have centers $c_1 = (0, 0, 0)$, $c_2 = (x_2, 0, 0)$, where $x_2 > 0$, and $c_3 = (x_3, y_3, 0)$ for s_1, s_2, and s_3 respectively. The overlap between s_1 and s_2 is $\omega = r_1 - (x_2 - r_2) = r_1 + r_2 - x_2$, meaning that after $f(\cdot)$ is applied, s_1 and s_2 will have their centers in $(-\omega/2, 0)$ and $(x_2 + \omega/2, 0)$ respectively. There are three possible cases depending on where s_3 is placed:

(i) $x_3 = -\omega/4$ or $x_3 = x_2 + \omega/4$.

(ii) $-\omega/4 < x_3 < x_2 + \omega/4$.

(iii) $x_3 < -\omega/4$ or $x_3 > x_2 + \omega/4$.

In Case (i), the distance between s_3 and one of the spheres doesn't change, whereas the difference increases to the other sphere. In Case (ii) the distance to both other spheres increase. However, in Case (iii), the distance to one of the spheres increases, but to the other it decreases. Assuming, without loss of generality, that $x_3 < -\omega/4$, even in this case, the total distance always increases, except in the case where $x_3 < -\omega/2$ and $y_3 = 0$, i.e., all three spheres are on a line, in which the total distance remains the same. This is because the angle $((x_3, 0), (x_3, y_3), (x_2 + \omega/2, 0))$ is larger than the angle $(x_3, 0), (x_3, y_3), (-\omega/2, 0)$. □

Theorem 1. *If X contains $2 \leq n < \infty$ spheres with finite radii and two or more of them overlap, then*

$$S(X) < S(f(X)).$$

Proof of Theorem 1. Applying Lemma 1 for all of the non-overlapping spheres gives that the total distance between them and the overlapping spheres never decreases, while the distance between the two overlapping spheres must increase. This means that the difference in total distance between all spheres must increase. □

Theorem 1 provides an argument as to why it can be expected that $f(\cdot)$ applied to X enough times will lead to a scene without any spheres overlapping. Since $f(\cdot)$ is defined such that it only moves two spheres, Theorem 1 shows that

S will always increase as long as there are still overlaps. This in turn shows that the spheres will spread out more and more for every iteration. While not a full proof that the scene converges to a scenario without overlap even in this simplified version of the problem, it is taken as enough motivation for assuming that a function that moves surfaces based on the overlap between them can, with enough iterations, remove the overlap between all surfaces.

2.2.4 Moving Surfaces Along an Attachment-Surface

When moving surfaces to remove the overlap between them, different boundary conditions could apply to the movement. One such boundary condition is when the surfaces are to be attached to some other, usually larger, attachment-surface. This could, for example, represent proteins embedded in a cell membrane. In this case, the location of the surfaces can be noted, after which they can be moved using some method for moving surfaces to remove overlap. The surfaces can then be moved back onto the attachment-surface. An algorithm to perform this task has been developed and works by first identifying the attachment-surface of every surface. After moving the surfaces to remove overlap in the usual manner, the attachment-surface can be found for each surface by looking for it along the normal of the attachment-surface at the starting position of the surface. The surface can then be moved to the attachment-surface at the point where the normal, extended from the new position of the surface, intercepts the attachment-surface. Figure 2.6 shows a 2D example of the algorithm. If the line extended from the new position of the surface does not intersect the attachment-surface, the surface can be placed on the plane defined by the original location of the surface and normal of the attachment-surface at the original point of contact.

This algorithm does not work if the curvature of the attachment-surface is too large, as it might not be found after the movement of the surfaces. How large the curvature can be depends on how far the surfaces get moved by the algorithm to remove overlap. As this algorithm moves the surface away from where it was originally moved, new overlap might occur. Using an iterative approach, as described in Section 2.2.3, to remove said overlap can solve this problem. It cannot, however, solve the problem that it might be impossible to fit all surfaces on the attachment-surface, if the attachment-surface is not sufficiently large.

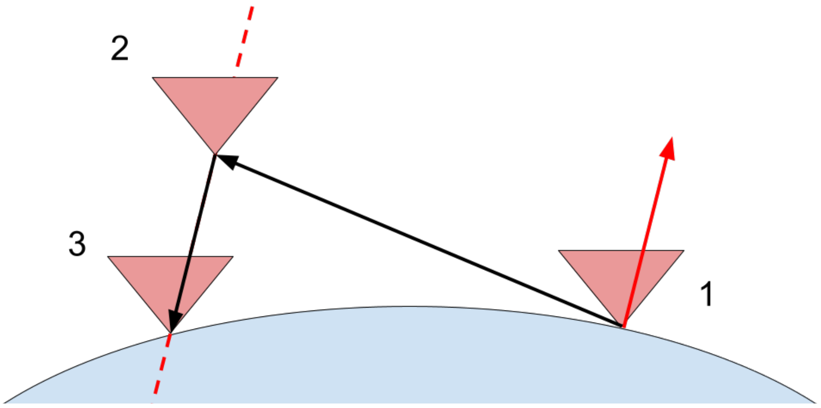


Figure 2.6: A red triangle being moved along a blue attachment-surface. The red triangle starts of at position 1, and is moved along the black arrow to position 2. The red arrow indicates the normal of the attachment-surface at position 1, and the dotted red line shows the direction of this normal. The triangle is then moved along the dotted red line to the attachment-surface at position 3.

2.3 Implementation

In ArtiaX, particles with a location and rotation can be placed in 3D-space. Triangular mesh surface models can be attached to these particles, which are used to render a scene with realistic biological features displayed through particles with attached surfaces. However, the surfaces simply render at the position of the particles they are attached to without any regard for collision between the surfaces. This can lead to an unrealistic scene where surfaces are overlapping. To solve this, two methods for removing overlap between surfaces have been implemented.

Both methods operate by pairwise measuring the overlap between surfaces, applying a force proportional to the overlap, and moving all surfaces in a iterative process until no surfaces overlap. The difference in the methods is their way of measuring the overlap and application of the force. One method uses a probabilistic method of determining the volume of the overlap, described in Section 2.2.1, whereas the other method attempts to measure the depth of the overlap, described in Section 2.2.2. This section deals with the implementation of these methods, by first looking at the implementation of the methods for measuring overlap in Section 2.3.1, and then the function that moves surfaces iteratively in Section 2.3.2. To illustrate the implemented methods, the surfaces used in the images of this section are recreations of a transmembrane adhesion complex[1], representing a realistic example of the types of surfaces that might be used in ChimeraX [1].

2.3.1 Measuring Overlap

The two methods for measuring overlap discussed in Sections 2.2.1 and 2.2.2, namely the Monte Carlo approach to measuring volume and the normal projection method for measuring depth of overlap respectively, have been implemented into ArtiaX. From the user's perspective, there is a function available that moves the selected surfaces so that they no longer overlap, with the option of choosing the normal projection method or the Monte Carlo approach as the method for calculating the overlap between surfaces. The normal projection method is the default choice available to users but since it is not guaranteed to work for all geometries, the Monte Carlo approach exists as an alternative.

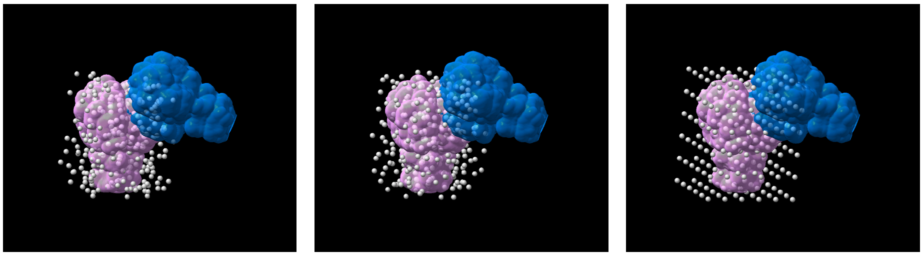
The following sections outline and describe the implementation of both methods, describing in detail the algorithms used and showing how the methods work in ArtiaX.

Calculating Overlap Volume

Multiple approaches have been tested to find the quickest and most reliable method for measuring the volume of two overlapping surfaces. They all rely on generating points within a given bounding box of the surface and calculating the proportion of points inside the surface. This proportion can then be multiplied with the volume of the bounding box for an estimate of the volume of the surface. The basic principles are these described in Section 2.2.1, however, different methods of generating the points inside the bounding box have been tried out.

The three methods tested were: A classic Monte Carlo approach with points generated with uniform randomness within the bounding box, a Poisson disc approach with points generated using Poisson disc sampling (see Section 2.2.1), and a regular grid approach, where the points were placed on a 3D regular grid. The accuracy and time taken for the different approaches can be found in Section 2.4.1. The number of points used in a particular method is specified by the user, as the number of points needed for a good approximation of the volume of overlap depends on the shape of the surfaces. More specifically, it depends on the proportion of the volume of the surface and the volume of the bounding box that encloses the surface. Figure 2.7 shows an example of the three different methods for generating points.

[1] Available at https://www.emdataresource.org/emdlist=DOI:10.1038/s41467-020-16511-2



(a) Points generated using uniform randomness.

(b) Points generated using Poisson disc sampling.

(c) Points placed on a regular grid.

Figure 2.7: Two overlapping surfaces, with three different methods of generating 260 points in the bounding box of the pink surfaces. Note how the points are more spread in (b), whereas the points have a tendency to clump together, and there are holes without any points in (a).

To calculate overlap between a group of surfaces, they are all compared pairwise. The bounding boxes of the surfaces are compared, and the surfaces that have overlapping bounding boxes are noted. Then the surfaces are processed, starting with the surface overlapping with the most other surfaces. A set of points is generated inside of the currently processed surface's boundary box using one of the methods for generating points described above. These points are then checked to determine if they are contained within the currently processed surface, and all other generated points are discarded. The remaining points are then checked to find how many are contained in each of the other surfaces that have overlapping bounding boxes with the currently processed one. This produces a list, with the number of points that are contained in both the currently processed surface and the surfaces whose bounding boxes overlap with it. Calculating the fraction of all generated points that are in both surfaces and the volume of the bounding box of the currently processed surface, an estimate of the overlapping volume between the two surfaces can be calculated. The currently selected surface is completely processed and is removed from the calculations so as not to be processed again. The next surface with the most overlapping bounding boxes is selected to be processed. The processing continues until there are no more surfaces whose bounding box overlaps another that is not yet processed. Figure 2.8 shows a small 2D example of the process, highlighting how the selection of which surface to process works, as well as how the points are generated for that surface.

This algorithm relies on a function for finding whether a point is inside or outside of a surface, which has been implemented using the Jordan Curve Theorem-

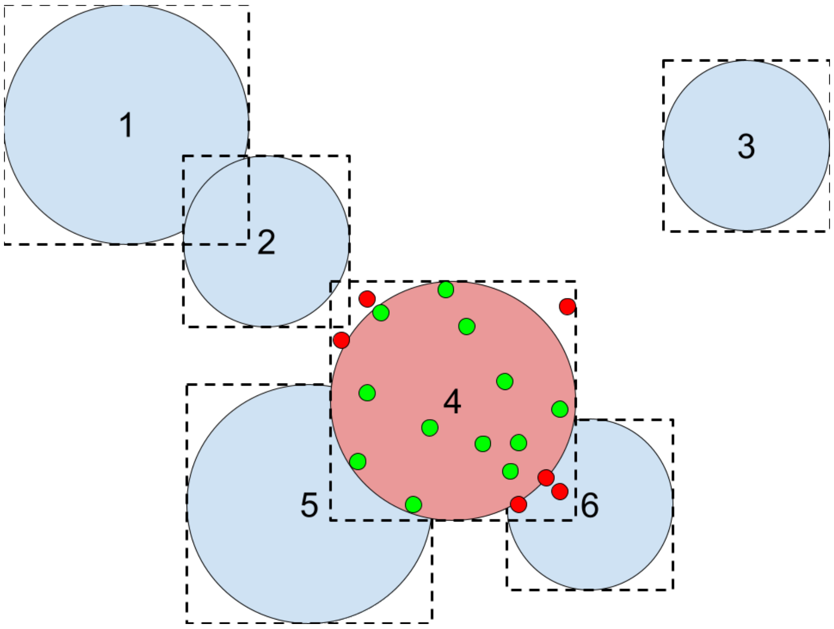


Figure 2.8: A simplified 2D example of the process used for measuring the pairwise overlap volume between surfaces. In this example, the circles represent the surfaces, and the dotted lines around them represent their bounding boxes. The first step of the process is to find the surface whose bounding box overlaps with the most other surfaces, which in this case is surface four, overlapping with three other surfaces. Points are then generated inside of the bounding box belonging to the fourth surface, and the ones that are outside of surface four are discarded (the red points). The remaining points (the green points), are then checked to see how many are contained in the surface whose bounding boxes overlap with surface four, i.e., surfaces two, five, and six. Surface four is then considered processed, and so are surfaces three, five, and six, as none of them have overlapping bounding boxes with any more unprocessed surfaces. The next surface to process would therefore be surface one or two, after which the process will finish, as the overlap between all surfaces is known. Using this method, only two sets of points need to be generated, which accelerates processing compared to generating points for every pair of surfaces.

based method mentioned in Section 2.2.1. A function exists in ChimeraX for finding the first intersection between a line and a triangle in a triangular mesh, implemented in C++, making it faster than the code implemented in python. This function has been used to implement the Jordan Curve Theorem-based method.

Calculating Overlap Depth

The normal projection method was created to solve the problem of calculating the depth of overlap between two surfaces, and as such has been implemented in ArtiaX (see Section 2.2.2 for the theoretical background to the method). It is the standard method used in the function that moves surfaces until they no longer overlap.

The implementation of the normal projection method relies on a ChimeraX function implemented in C++ that quickly finds points close to one another between two lists of points. As input it takes two lists of coordinates and returns the points in both lists that are within a specified distance from each other. This function is used to find the intersection between two surfaces, using the triangle vertices in the mesh representation of the surfaces as the points given to the function. With the intersection points found, a least squares fit is used to find the normal of a plane approximated by the intersection points, and the mean of them is defined as a point on the plane. Using the normal and this point on the plane a mathematical description of the plane is calculated, and the surfaces are filtered to only leave the vertices on the other side of the plane compared to the center of the surfaces. These remaining points are then projected onto the normal, and the maximum distances of a vertex from the plane for both surfaces is calculated. The sum of these two distances are returned along with the normal of the plane. An example of how this might look using two surfaces in ArtiaX is shown in Figure 2.9

2.3.2 Moving Surfaces

When using the normal projection method for measuring overlap of a surface pair, a distance and a direction is returned. These values are used to move both surfaces half the returned distance each along the returned direction but opposite ways. When a surface overlaps with more than one other surface, the average of the movements is used to move the surface. Using the Monte Carlo approach returns an approximation of the overlap volume, which is used to calculate a distance to move the surfaces, which are both moved the same amount away from their respective centers. The distance is calculated by simply taking the third root of the overlap volume, multiplied by a constant specified

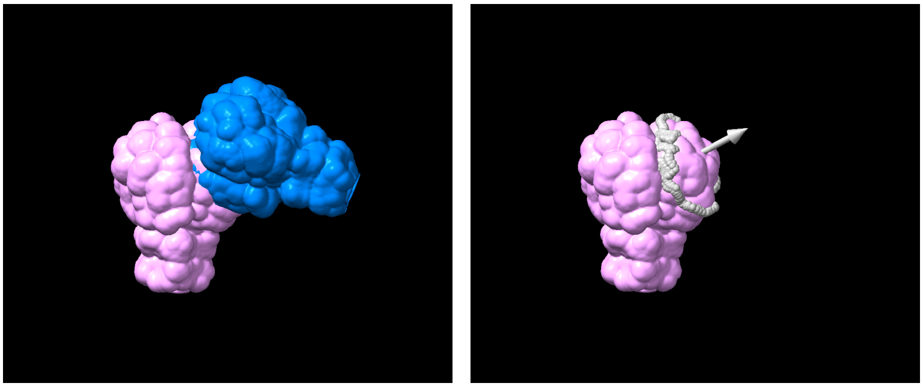


Figure 2.9: Two surfaces overlapping, with the intersection points and the normal fit to those points shown in white in the image to the right.

by the user, meaning the overlap is assumed to be cubic. The overlap is almost certainly not cubic, but it gives a fair estimate of the overlap distance, often only needing one or two iterations to move the surfaces a good distance apart.

2.4 Results

The functions described in Section 2.3 have been implemented and work for moving surfaces to remove overlap between their attached surfaces. Figure 2.10 shows a simple case of two surfaces overlapping, and how they look after being moved to remove overlap. Figure 2.11 shows a more complex scene with several overlapping surfaces, and what that scene looks like after using the implemented functions to move the surfaces.

2.4.1 Generating Points for the Monte Carlo Approach

The three strategies to generate points for the Monte Carlo approach discussed in Section 2.3.1 were tested for speed and accuracy, by using them to approximate the volume of an enclosed surface with known volume. The full results can be found in Table 2.1. As can be noted from the table, using a regular grid gives a poor approximation of the volume, no matter how many points are generated. In the experiment, the number of points generated with Poisson disc sampling is not actually the values stated. Instead of generating a set number of points, a radius was set within which no other points could be created. The stated value actually represents the number of points possible using the set radius, if

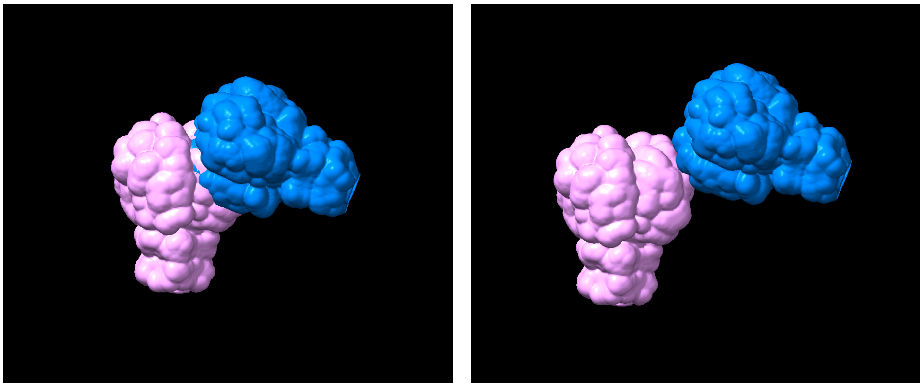
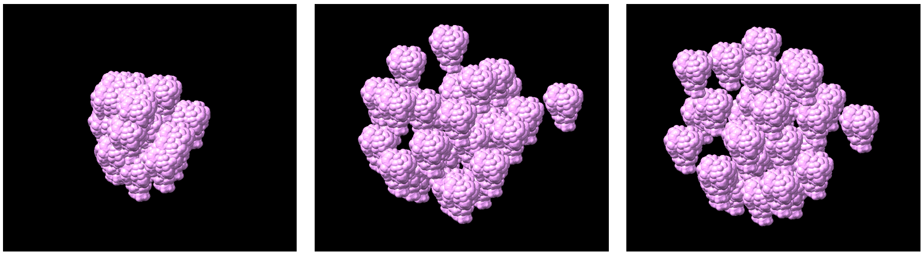


Figure 2.10: Two surfaces, one blue and one pink, being moved to remove the overlap between them.



(a) 31 copies of the same surface, many of which overlap.

(b) The surfaces moved using the normal projection method.

(c) The surfaces moved using the Monte Carlo approach.

Figure 2.11: A group of overlapping surfaces, being moved to remove overlap. All images are from the same angle.

all points were places on a grid. This means that the total number of points for the Poisson disc sampling is different in every run, and usually about half of the stated value. This explains why the time taken for the Poisson disc sampling method is usually shorter than the time taken for uniform sampling, as fewer points had to be tested to determine if they were contained in the surface.

Table 2.1: Data collected from approximating the volume of a surface with known volume using different methods for generating points. All values are averages from ten rounds of approximation, except for the grid approach, which generates the same points every time and therefore gives the same result every time. All the relative values are related to the value in the uniform row with the same number of generated points. So, for example, the average error using 500 points generated with Poisson disc sampling has 44% of the error, 71% of the standard deviation, and takes 73% of the time needed for using 500 points with uniform sampling.

Method	N.o. points	Error [%]	Rel. Error	Rel. std. dev.	Time [s]	Rel. time
Uniform	10	46.0	1.00	1.00	0.0114	1.00
	100	11.9	1.00	1.00	0.0878	1.00
	500	5.3	1.00	1.00	0.4398	1.00
	1000	4.6	1.00	1.00	0.8699	1.00
Poisson	10	64.6	1.40	2.05	0.0242	2.12
	100	9.1	0.77	1.15	0.0763	0.87
	500	2.3	0.44	0.71	0.3216	0.73
	1000	2.6	0.56	0.76	0.6183	0.71
Grid	10	100.0	2.17	-	0.0035	0.30
	100	55.5	4.67	-	0.0627	0.71
	500	32.8	6.20	-	0.3134	0.71
	1000	26.1	5.65	-	0.6693	0.77

While the results are not conclusive, the Poisson disc sampling method was selected for its high accuracy and low standard deviation, leading to a more stable result. As calculating whether a point is in a surface or not is time consuming, the ability of the Poisson disc sampling method to produce a set of spread points and thus a good approximation of the volume even at low point-counts makes it superior to the other two methods.

2.4.2 Normal Projection Method and Monte Carlo Time Comparisons

Because of the different nature of the methods, it is hard to directly compare the normal projection method and Monte Carlo approach for removing overlap. The reason is twofold; Firstly, for the Monte Carlo approach more points can always be generated, and the distance moved based on the overlap volume can always be decreased (an option given to the user), leading to a slower but more precise method, as the risk of surfaces being moved to far away from each other decreases, but the risk that two surfaces don't move far enough away increases, leading to more iterations being needed to move the surfaces so that they no longer overlap. Secondly, as no measure for the success of the methods has been defined, it is hard to tell which function did a better job of moving the surfaces as little as possible while still removing the overlap.

However, using settings for the Monte Carlo approach that give a result that looks similar to the normal projection method, the time taken for each method to move 31 overlapping surfaces was measured. After performing the movements ten times for each method, the average time taken in seconds for the normal projection method was 48.22958, and for the Monte Carlo approach 58.03436. So, for this test case, the Monte Carlo approach took almost 10 seconds, or 20%, longer. It should be mentioned that the measured time also includes the time taken to do all the surrounding work for the function, such as updating the location of the surfaces, redrawing the scene, and organizing all the data. This work is the same for both functions, but takes many seconds each time the function is called, meaning that the normal projection method can be expected to work more than 20% faster than the Monte Carlo approach.

Chapter 3

Tomogram Processing

Tomograms are large three-dimensional images, reconstructed from a series of 2D images taken from different angles with an electron microscope [19]. In ArtiaX, tomograms can be displayed and analyzed. To increase the signal-to-noise ratio of the tomograms, Fourier-space filters and slice-averaging has been implemented. Frequency filters can be useful as both high-frequency and low-frequency noise appear in the tomograms. Slice-averaging, or simply averaging, refers to averaging voxel-values of the 3D image, which can also reduce the noise, as it is is assumed to be Gaussian. This chapter will examine the theory behind the image processing in Section 3.1, the tested methods in Section 3.2, and the results of the testing and the filters in Section 3.3. Since tomograms can take up many Gigabytes of storage space, many methods and algorithms have been tested to find a space and time efficient implementation.

3.1 Theoretical Background

Image processing, specifically frequency filters, can be efficiently done in Fourier-space. This section gives an explanation of how cross-correlation can be used to average a 3D image, and presents the mathematical background to how the *Fast Fourier Transform* (FFT) can be used to filter large 3D images.

3.1.1 Discrete Cross-Correlation

The discrete Cross-Correlation is an operation that has many uses, but is most commonly used for measuring similarity between signals [15]. The one dimensional cross-correlation of two real discrete signals $x[l]$ and $h[l]$ can be defined as [15]

$$z[l] = \sum_i x[i]h[l-i], \tag{3.1}$$

or, for three dimensional signals $x[l,m,n]$ and $h[l,m,n]$,

$$z[l,m,n] = \sum_i \sum_j \sum_k x[i,j,k]h[l-i,j-m,n-k]. \tag{3.2}$$

In Figure 3.1, an example of multidimensional cross-correlation using this definition is presented using a small 6x6 size image and a 3x3 kernel. The kernel is moved over the entire input image, multiplying the values in the kernel with the aligned values in the input image, to produce the output value at the position where the kernel is currently centered. The same principle of calculation applies for 3D images as well. What happens at the border of the signal depends on the type of cross-correlation used, usually with the input signal expanded some way and the kernel only being moved over the original size of the signal. In the example in Figure 3.1, the signal has been padded with zeros. Another common technique is to assume the signal is surrounded by copies of itself, or that the signal is mirrored in the edges.

Averaging Through Cross-Correlation

Cross-correlating an image with the right kernel can be used to average the image in a desired direction. The values of the kernel decide the weight to put to different parts of the original image when computing the average. To ensure the image is averaged and does not increase the total size of the signal, all the values in the kernel should sum to 1. Figure 3.2 shows a simple example of how cross-correlation can be used to average an image in a set direction. This can be done to a signal in an arbitrary number of dimensions, by simply using a kernel of the same number of dimensions. The number of pixels that are considered for the averaging depends on the size and the values of the kernel.

Fourier-Space Cross-Correlation

Cross-correlation can also be performed in Fourier-space, with (3.3) describing the relationship between the cross-correlation of two time-discrete functions $h[l]$ and $x[l]$, and the discrete Fourier transform of the cross-correlation as

$$z[l] = \sum_{i=0}^{N-1} x[i]h[l+i] \Longleftrightarrow Z[n] = X^*[n]H[n], \tag{3.3}$$

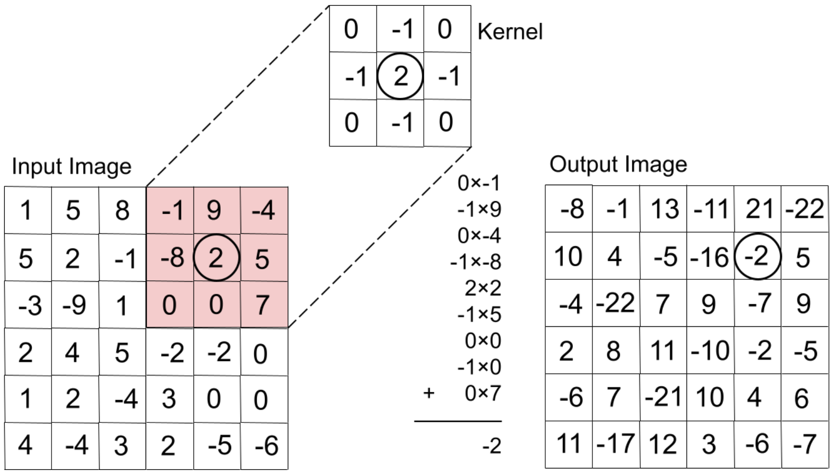


Figure 3.1: A 6x6 signal being cross-correlated with a 3x3 kernel, using zero padding. The image highlights how the kernel matches the signal when centered at position (5,2), and it shows the sum that defines the value of the output at position (5,2). The entire output image is produced by moving the kernel across the input image and calculating the matching sum.

where $H[n]$ and $X[n]$ are the discrete Fourier transforms of $h[l]$ and $x[l]$, and $X^*[n]$ refers to the complex conjugate of $X[n]$ [3]. Using the FFT (described in Section 3.1.2) can potentially speed up the calculation for computing a discrete cross-correlation [13].

3.1.2 Fourier-Space Filters

To reduce noise in an image, high and low-pass filters can be applied. The simplest filters are brick-wall filters, where all frequency components above or below a threshold value are simply set to zero. This design is simple to implement, but can cause ringing effects. To minimize these effects, more advanced transitions between the filtered and non-filtered frequencies can be applied. A raised cosine function can be used to smooth the transition, or the more common Gaussian cut-off [15].

To filter a discrete signal, the FFT can be utilized. The FFT is a sweep of efficient algorithms all implementing the discreet Fourier transform [3], allowing a digital image to be transformed into Fourier-space. After transforming the signal to Fourier-space using the FFT, a mask can be applied to reduce or increase certain frequencies in the signal, after which the signal can be transformed back

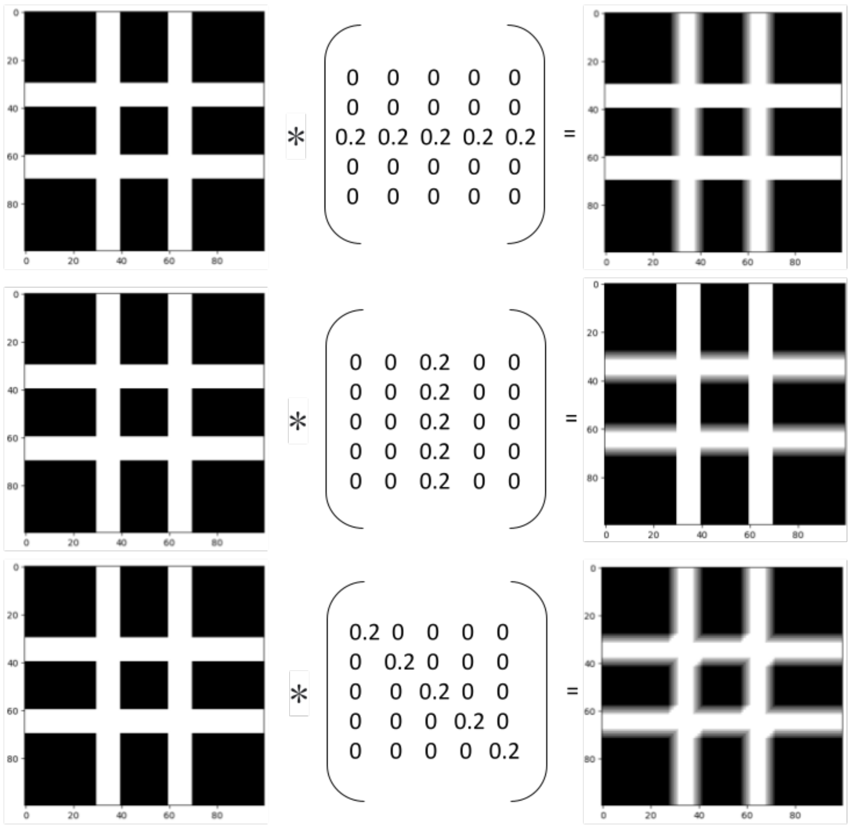


Figure 3.2: An image averaged in different directions using cross-correlation. All three images to the left are the same input signal, and the * symbol denotes cross-correlation with the kernel. The three different kernels produce three different outputs; the first image is averaged in a horizontal direction, the second a vertical, and the third a diagonal. In all three examples a 5x5 kernel is used, and 5 pixel values are used to calculate every pixel value in the output. The input signal is padded using mirroring.

with the inverse FFT. When using real-valued signals, the computations can be simplified and sped up using the real-valued fast Fourier transform, the RFFT, and its inverse, the IRFFT [12]. The FFT can be used to transform multidimensional signals by transforming it along one axis at a time. However, since the result of a RFFT is itself complex, the RFFT can only be used along one axis in the case of a multidimensional real signal, and the remaining directions must be transformed using the normal FFT. As such, when a multidimensional signal is said to be transformed using the RFFT, the RFFT is only used along one axis.

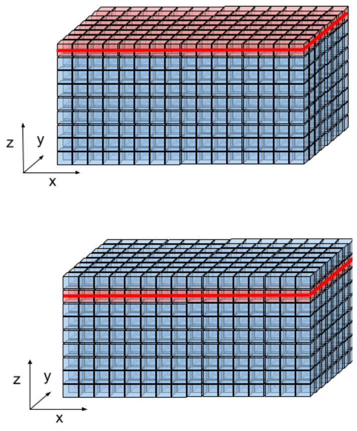
3.2 Implementation

In ArtiaX, tomograms are three-dimensional density maps, usually constructed using data collected from a cryo-electron microscope. These tomograms are represented by a 3D-matrix, and can be visualized as rectangular prisms, where each density-value is rendered as the brightness of the corresponding voxel. However, tomograms are also often rendered as a series of 2D-slices. These 2D-slices (or slabs, used interchangeably) are then studied to find features in the collected data. To simplify data collection and improve the resolution of models created using the data, multiple methods are employed to improve the signal-to-noise ratio in the tomograms. Two such methods have been implemented into ArtiaX as part of this thesis, namely, averaging of multiple tomogram slices and frequency filters.

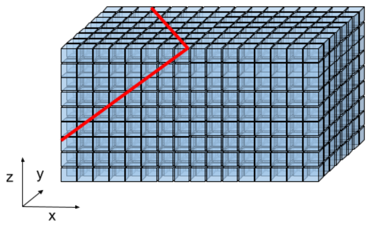
To understand the challenges with the implementations of these methods, it is necessary to first give a quick explanation of the way that tomograms are stored and represented in ChimeraX.

3.2.1 Data Format

As previously noted, the tomograms are stored as 3D-matrices in ChimeraX, where each voxel value can be reached by three indices, noting the distance in each of the Cartesian axes. So, if the tomogram is stored in matrix T, the voxel value at position (x, y, z) is reached as $T[x, y, z]$. Using this, the 3D version of the tomogram can be rendered. However, as mentioned, a more common way of viewing the tomograms is as a series of 2D slabs that cut through the tomogram, where the user can choose the orientation of the slab. The user can then move through the entire tomogram, by keeping the orientation of the slab stable, but moving the slab along its normal. The size of the tomogram and the orientation of the slab decides the total number of slabs needed to view the entire tomogram. The slab is always moved one unit along its normal when



(a) The normal of the slab is along the z-axis.



(b) The normal of the slab is not along one of the Cartesian axes.

Figure 3.3: Examples of how tomograms are stored in ChimeraX, where each cube is one data point in the 3D-matrix. In (a), the slab goes perfectly through the center of each data point, but in (b) the slab does not. Because of this, in (a), the pixel values in the slab can be read directly from the matrix, whereas some interpolation method is needed in (b) to calculate the pixel values of the slab.

moved, where one unit is the distance between two neighboring voxels. Figure 3.3a shows an example of the slab being moved through the tomogram, where the slab is oriented such that its normal is along the z-axis. In a case where the normal of the slab is along a Cartesian axis, the data describing the slab is simply a 2D-matrix taken directly from the original 3D-matrix. However, when the slab is oriented such that its normal is not along a Cartesian axis, the situation is more complicated. As illustrated in Figure 3.3b, the size of the slab shifts as it moves through the tomogram, and the data describing the slab is not simply a 2D-matrix extracted from the original 3D-matrix. Since the slab doesn't necessarily go though the center of each voxel of tomogram, some interpolation method is required to calculate the pixel values for the slab. In ChimeraX, this interpolation is done on the GPU using shaders implemented in OpenGL, resulting in quick calculations, so that the user can change the orientation of the slab and move the slab through the tomogram in a quick manner. This real time interpolation data is not available to ArtiaX, but is only used for drawing the scene.

3.2.2 Tomogram Slice Averaging

A simple way to increase the signal-to-noise ratio of a tomogram is to average the data from multiple slices. That is, instead of only showing the data corresponding to the slice currently being viewed, showing a slice composed of the average data of the current slice and n slices above and below the current one, effectively acting as a low-pass filter focused in one spacial direction. If the noise is randomly distributed, and the signal is spread over a number of slabs, the noise should be reduced and the signal improved. This operation is henceforth called *tomogram averaging*.

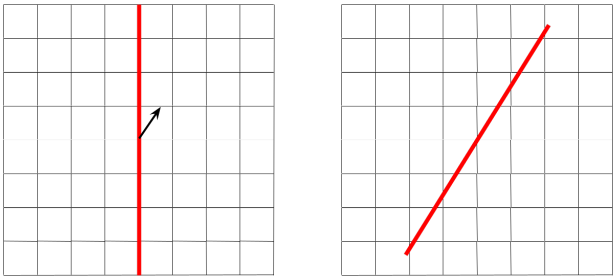
To average tomograms, two methods have been tested. The first one relies on calculating the pixel values for each slab and creating a new tomogram where the matrix defining the tomogram is changed so that each slab is an average of a specified number of slabs above and below it. To do this, interpolation is needed, because, as discussed in Section 3.2.1, computing the pixel values of a slab is easy when the normal of the slab is one of the Cartesian axes, but not when the normal is an arbitrary axis. The second method uses cross-correlation to average the tomograms, avoiding the problem of interpolation.

Interpolation method

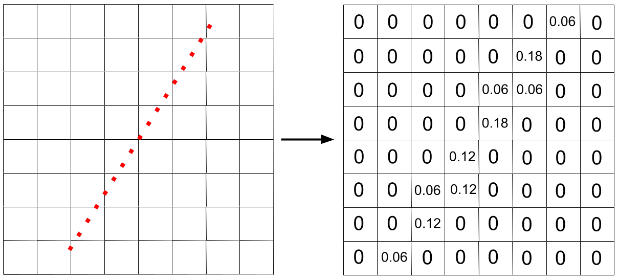
To solve the problem of finding the values describing a tomogram slice that is not along one of the Cartesian axes, interpolation is needed, as explained in Section 3.2.1. This interpolation is done on the GPU in ChimeraX, but in ArtiaX all computation takes place on the CPU. As tomograms often consist of tens of millions of floating point values, two interpolation implementations[1] were tested to find a method that could compute the required values in a sufficiently short time. Both implementations utilize that the data to be interpolated is stored on a structured rectilinear grid, and both are able to use linear and nearest neighbor interpolation. Section 3.3.1 compares the runtime needed between the four different methods on a large tomogram data set.

Using these interpolation methods, the pixel values of an arbitrary slab through the tomogram can be calculated. By calculating the values for all the slabs in the tomogram, theses slabs can then be averaged. A new tomogram can then be created by interpolating from the averaged slabs back to the matrix representation of the tomogram, after which ChimeraX can display the new, averaged, tomogram to the user.

[1]Both from scipy [18], one from the image processing tools, scipy.ndimage.map_coordinates and one from the interpolation tools, scipy.interpolate.RegularGridInterpolator.



(a) Creating and rotating the line (red line) according to the given axis (black arrow).



0	0	0	0	0	0	0.06	0
0	0	0	0	0	0.18	0	0
0	0	0	0	0.06	0.06	0	0
0	0	0	0	0.18	0	0	0
0	0	0	0.12	0	0	0	0
0	0	0.06	0.12	0	0	0	0
0	0	0.12	0	0	0	0	0
0	0.06	0	0	0	0	0	0

(b) Sampling the line and creating a kernel using the samples.

Figure 3.4: The sampling method for creating a 2D kernel that can be used for averaging an image in the direction dictated by an arbitrary axis using cross-correlation. As is clearly seen in this example where only 17 sample points have been used, a high sample rate is necessary to obtain a kernel that accurately represents the given axis.

Cross-Correlation method

Another way to create averaged tomograms is to use cross-correlation. By cross-correlating the tomogram with an appropriate kernel, it can be averaged over an arbitrary axis without the need of interpolation, as described in Section 3.1.1. To create the correct kernel, a 3D-matrix is created of the same size in all directions, which is specified by the user and determines over how many slabs the tomogram is to be averaged. A line is then created along the axis with the same length as the size of the matrix, and this line is then sampled at a high rate. The cells of the kernel matrix are assigned values equal to the proportion of the samples of the line that are within the cell. Figure 3.4 provides a simple 2D example of this method for creating a kernel.

The operation of cross-correlating a kernel with an image can be done in Fourier-space, potentially speeding up calculations, as described in Section 3.1.1. This, however, has not been implemented, but instead the cross-correlation is performed directly on the matrices, without Fourier transformation.[2] Another way of speeding up calculations is to not use a cubic kernel, but instead create the smallest, rectangular matrix that covers all the non-zero values. As implemented currently, many of the cells on the kernel matrix end up being zero, causing unnecessary computations.

The result from a cross-correlation with this kernel and the matrix that defines the tomogram yields a matrix that is used to create a new, averaged, tomogram. Section 3.3.1 shows the time taken to perform this operation on a large tomogram, and Section 3.3.1 provides a comparisons between this method of averaging a tomogram and the interpolation method described in Section 3.2.2.

3.2.3 Tomogram Frequency Filters

The noise apparent in tomograms captured using cryo-electron tomography is of both higher and lower frequency than the signal [9]. As such, the ability to high-, low-, and band-pass filter tomograms has been added to ArtiaX. The user can specify the cutoff frequencies and whether to use a sharp, Gaussian, or raised cosine cutoff for the filters. To filter a tomogram, the matrix data defining the tomogram is transformed into Fourier-space, and multiplied point-wise with the chosen filter created from user-specified values. The result of the multiplication is transformed back using the inverse Fourier transform, and then used to create a new, filtered, tomogram, as per the method described in Section 3.1.2. All transformations are implemented using the RFFT, as this decreases the amount of ram required by a factor of two, compared to using the normal FFT, as also mentioned in Section 3.1.2.

3.3 Results

In this section, the results of the implemented image-processing tools are presented, along with pictures showing the application of the tools used on tomograms [3]. The different methods of averaging tomograms are compared, and one of the methods is implemented.

[2]The cross-correlation is performed using the scipy function scipy.ndimage.correlate [18].

[3]The tomogram used to visualize the effects of the tools implemented is taken from the public tilt series in the Electron Microscopy Public Image Archive (EMPIAR accession code: EMPIAR-10304) https://www.ebi.ac.uk/empiar/EMPIAR-10304/ [7]

3.3.1 Tomogram Slice Averaging

The first set of results are tests performed to compare the speed of the interpolation method and the cross-correlation method of averaging described in Section 3.2.2. In all experiments, the same tomogram with a size of $1024 \times 1700 \times 200$ floating point values is used. In addition to the results of the experiments used to compare the methods, a small discussion is given regarding which methods has been implemented into ArtiaX.

Interpolation

To compare the four implementations of interpolation described in Section 3.2.2, they were all used to interpolate many values on a tomogram.

For every interpolation method, three experiments were carried out, the results of which are presented in Table 3.1. All experiments interpolated n slabs of size 1024×1700 in the center of the tomogram, with n set to 3, 21, and 41. The different values of n represents different amounts of slabs used to average a tomogram, with $n = 1$ representing the scenario where 1 slab above and below the current slab are used to calculate the average of the current slab, $n = 21$ a scenario where 10 slabs above and below the current slab are used, and $n = 41$ a scenario where 20 slabs above and below the current slice are used. Each experiment was repeated ten times, with the average representing the time it would take to calculate the values of one slice in the averaged tomogram, where 200 slices are needed to create an averaged tomogram the size of the original. This average time was therefore multiplied with 200, to give en estimate of the time needed to interpolate the values to create an entire averaged tomogram, assuming that the interpolated values used to create one averaged slab cannot be used in creating any others. Even assuming that the interpolated values can be used to calculate the average of many slabs, the time taken to interpolate a sufficient number of values is too large as to still be desirable to users.

In addition to the interpretation needed, there are many more steps needed to create a complete averaged tomogram, as described in Section 3.2.2. This means that for a large tomogram, with a correspondingly large data set, such as the one used in the experiment, it would take many minutes on most computers to average.

Cross-Correlation

The time taken for the cross-correlation method to average the tomogram was also measured, for comparison purposes with the interpolation methods. Table 3.2 contains the time taken for different numbers of slabs to include in the averaging. As in the experiments with the interpolation methods, the number

Table 3.1: Different interpolation methods and the time it takes to interpolate over a specified number of slabs of size 1024x1700 for a regular 3D grid. Map stands for the scipy method scipy.ndimage.map_coordinates, and RGI stands for scipy.interpolate.RegularGridInterpolator. Further, N.o. Slabs is how many slabs were interpolated above and below the current slab, and time/tomogram is the estimated time it would take to create an average slab for a full tomogram made up out of 200 slabs. The relative time shows the time for the different methods as compared to a nearest neighbor interpolation using Map. All times are the average times from 10 runs with the same settings.

Method	Order	N.o. Slabs	Time/Slab [s]	Time/Tomo. [s]	Rel. Time
Map	Nearest	1	0.18	36.48	1.00
		10	1.28	255.06	1.00
		20	2.45	489.55	1.00
	Linear	1	0.31	62.92	1.72
		10	2.15	429.33	1.68
		20	4.14	828.98	1.69
RGI	Nearest	1	0.48	96.88	2.66
		10	3.48	696.32	2.73
		20	6.29	1257.66	2.57
	Linear	1	1.19	238.90	6.55
		10	8.17	1634.72	6.41
		20	15.54	3109.00	6.35

of slabs refers to the number of slabs above and below the current slab to weight into the average. Again, all experiments were carried out ten times, with the averaged displayed in the results. Unlike the other experiments however, this time the data comprising the entire tomogram is calculated, meaning the resulting data from the cross-correlation can be used to directly create a new, averaged, tomogram. This is in contrast to the interpolation experiments, when only the interpolation needed to average one slab was measured, not taking into account the time it would take to actually average over all required interpolated values. Even so, the time taken is related to time per tomogram of the fastest interpolation method. One problem that becomes apparent with this method is that the number of multiplications grows on the scale of $\mathcal{O}(n^3)$.

Table 3.2: Measured time taken for the cross-correlation method of averaging tomogram data. See Table 3.1 for an explanation of the different values and a comparison with the interpolation method.

N.o Slabs	Time	Relative Time
1	2.77	0.029
10	9.38	0.013
20	28.28	0.022

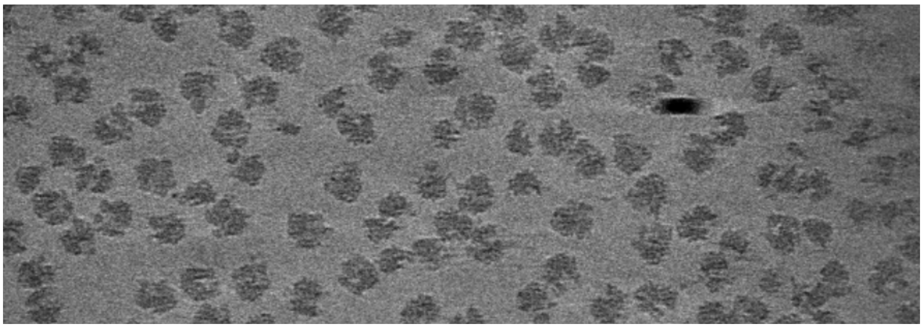
Averaging Method Discussion

When comparing the time taken to simply interpolate the required values used in the interpolation method to the time taken to produce the matrix describing the averaged tomogram using the cross-correlation method, it is clear that the cross-correlation method is superior in terms of computation speed. Because of this much faster computation speed, it was chosen as the preferable method, and has as such been introduced in ArtiaX, allowing the user to average any tomogram in an arbitrary direction. However, because of the cubic growth of the computations needed it is still not a perfect implementation for averaging over many slabs.

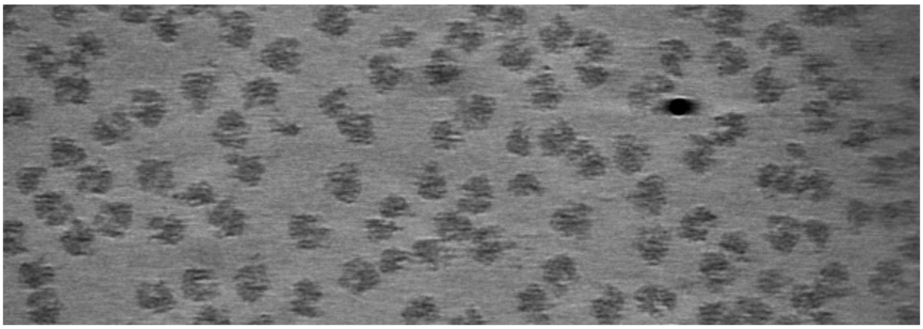
Figure 3.5 contains a tomogram, next to an image of the same tomogram averaged along the direction of the camera using the cross-correlation method. The averaged image can be seen to contain less high-frequency noise in the empty areas between particles, and the particles are more clear. This is a result of the noise being averaged out, but the signal being reinforced.

3.3.2 Tomogram Frequency Filters

The Fourier-space filters implemented in ArtiaX can successfully filter tomograms using the RFFT. Figure 3.6a shows a tomogram containing many copies of the same particle before any filter has been applied to it, and Figures 3.6b and 3.6c shows the same tomogram after being low-pass and high-pass filtered respectively. The low-pass filtered tomogram has been blurred, but contains less noise in-between particles. The high-pass filter on the other hand, has clearer particles than the original image. Both filters are applied to the tomogram in Figure 3.6d, where the particles are distinct, and the noise reduced. However, there is still noise in the image, some of which cannot be removed with frequency filters, because it is in the same frequency band as the original signal.



(a)



(b)

Figure 3.5: A portion of a tomogram in 2D representation (a), and the same tomogram but with the slab shown being the average of the 10 slabs above and below it (b). The dark regions are the particles being examined in the tomogram, and the black oval in the top right corner is an artifact from the tomogram acquisition process. In (a) there is high-frequency noise in-between particles, creating a grainy image. In (b), this noise has been reduced, while the particles remain clear.

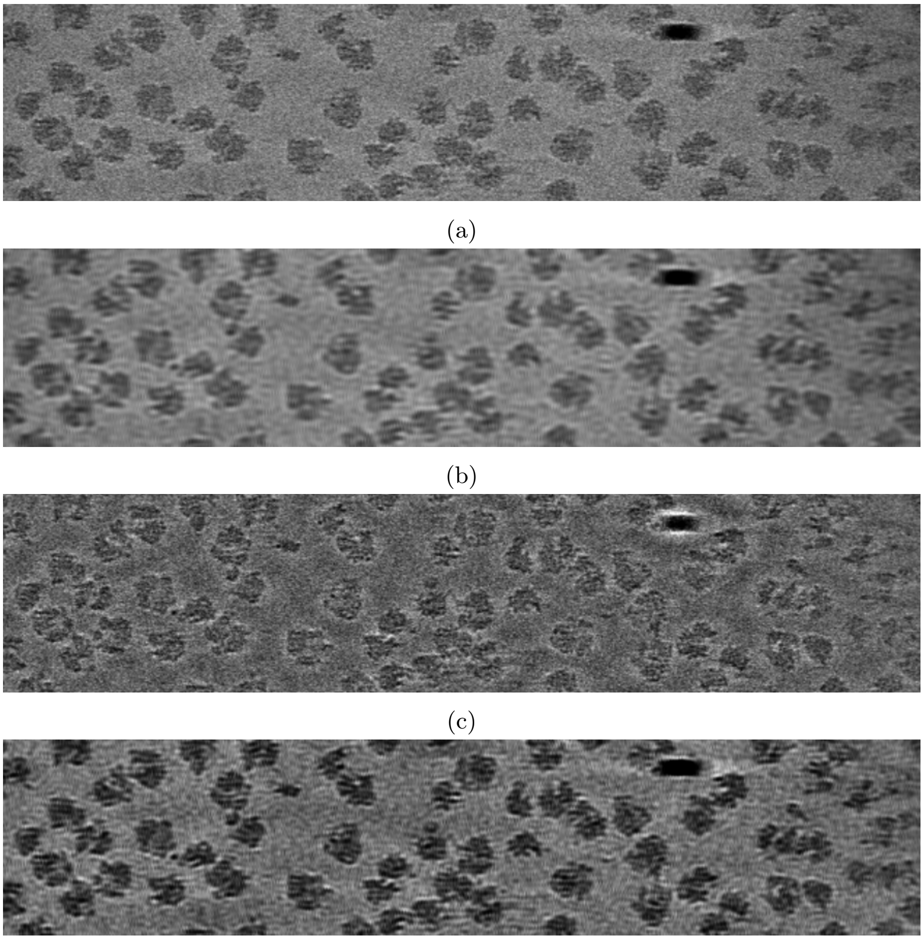


Figure 3.6: A part of a tomogram in 2D representation (a), with the same tomogram low-pass filtered (b), high-pass filtered (c), and band-pass filtered (d). The dark regions are the particles being examined in the tomogram, and the black oval in the top right corner is an artifact from the tomogram acquisition process. In the original image there is both high and low frequency noise, as can be seen in (b), where the grainy noise in between particles is lessened, and in (c) where the particles are made clearer. In (d) the noise is reduced while the particles are highlighted through the use of a band-pass filter.

Chapter 4

Discussion

The implementation of both parts of the thesis have their limitations, and there is potential for improving the new features, as well as adding other features to ArtiaX. A discussion on the limitations and possible improvements of the implemented features is presented in this chapter, which finishes with a section on other potential future additions to the plugin.

4.1 Removing Overlap Limitations and Improvements

The methods for removing overlap work in every scenario on which it has been tested. For both methods of measuring overlap the function finds overlapping surfaces and moves them accordingly. However, both methods can take a significant amount of compute time to finish when there are many surfaces in the scene. As such, the implementations of both methods could be improved, by potentially moving some of the calculations to the GPU or by implementing them in C++. The bottleneck in the current implementation of the normal projection method is identifying the vertices on the either side of the intercept plane.

The normal projection method developed for this thesis has been described mathematically in Section 2.2.2, but as is seen there, the mathematical description for the robustness of this new methodology is lacking. The extent of the shapes for which the method will give a good approximation of the depth of overlap would be useful to know for understanding the limitations of the method. Additionally, such an analysis may guide possible improvements to the method and its applicability to a broader class of cases. There are certainly many special cases that could be handled for arbitrary complex geometries.

When it comes to the Monte Carlo approach to approximating overlap volume, the Poisson disc sampling was shown to give better approximations of the volumes than a uniform distribution in the experiments discussed in Section 2.4.1, but no mathematical analysis of this has yet been made. An analysis of the convergence rate of the Poisson disc sampling approach to approximate the volume of a surface could help give these results a mathematical foundation.

As for the method of moving the surfaces, this too could benefit from a more thorough analysis. Section 2.2.3 provides a mathematical argument for why a function moving surfaces can be expected to over time remove all overlap, but even in this simplified scenario it does not prove that it always will. Extending this theorem to show that it always does (or does not) converge would give a better understanding of how a function moving surfaces could be designed to work better. Also looking at a more general scenario where not all surfaces are spheres might give insight into how the function could be improved.

That being said, for the purposes of this thesis and ArtiaX, the current methods work well and quickly in practice. In addition to the additional constraint presented in 2.2.4, another potentially useful boundary condition would be one for containing surfaces in some larger boundary-surface. A common use-case might be that some surfaces are bound by some larger boundary, for example proteins inside of a cell membrane. In this case, an extension of the method for removing overlap would be useful, where the surfaces that are moved cannot be moved outside of a selected boundary-surface.

4.2 Image Processing Limitations and Improvements

Both the averaging and the filtering of tomograms function as intended in ArtiaX. Even so, there are definite improvements that could be made to both functions. One major improvement would be to not keep the entire tomogram in RAM, but letting some of the information be stored in ROM, only accessing it when necessary. Tomograms can take up Gigabytes of space, and can therefore push the limits of the available ROM. Such a change would almost certainly increase computation time, but could be worth it for the improvements in memory availability.

For the averaging, improvements could be made by performing the cross-correlation in Fourier-space, as described in Section 3.1.1. This might speed up computation of the averaged tomogram. As is also mentioned in Section 3.1.1, another potential improvement would be to limit the size of the kernel as much as possible, to avoid unnecessary calculations. Also for the filtering there are potential improvements that could be made, such as adding more options for the filter cutoffs.

There are also more image processing tools that could be useful when working with tomograms. One important process currently unavailable in ArtiaX is *contrast transfer function* correction. The contrast transfer function introduces aberrations to the signal casued by the defocus and the spherical aberration coefficient of the objective lens used in the microscope [6]. Adding functionality into ArtiaX that could remove this noise would decrease the reliance on other tools for tomogram processing.

4.3 Addition Features

In addition to the features previously mentioned, there are others that could be implemented in ArtiaX to improve the user experience. The most pressing ones would be a feature for saving the current working session and an undo-button for undoing the last action. Being able to save an ArtiaX session would allow users to build up complex scenes, save them, and then continuing to work on them at a future time. Right now, if a session is closed, the entire scene must be rebuilt step-by-step, or recreated using a script the user has to write. As for the undo-button, such a feature could be highly useful for users, in case some element is accidentally deleted or moved. These features would not improve the processing ability of ArtiaX, but would improve usability, and potentially make the tool more accessible for new users.

Chapter 5

Conclusion

The development of the thesis has fulfilled its goals stated in the introduction, namely, to create a function that can move surfaces until they no longer overlap, and adding capabilities for filtering and averaging tomograms. Both parts encountered problems that had to be solved, and methods that needed to be tested and compared. For the removing overlap functions, a new method had to be created for measuring the depth of overlap between surfaces. This resulted in the development of the Normal Projection Method. Different methods for generating points for the Monte Carlo approach for approximating volume were also tested, with the Poisson Disc Sampling method being found superior. For the image processing part, different methods for averaging tomograms were compared, with the cross-correlation method chosen for the implementation. In addition to this, filters were successfully implemented using the RFFT.

With these two features added to ArtiaX, the plugin becomes an even more useful and relevant tool for structural biology analysis and visualization. Some of the methods developed for the purposes of this thesis can also be useful for other, unrelated goals. In particular, the normal projection method might be applicable to many other areas where 3D surfaces are represented and overlap is to be avoided, such as in animation rendering or video games.

Bibliography

[1] David Aparicio, Margot P. Scheffer, Marina Marcos-Silva, David Vizarraga, Lasse Sprankel, Mercè Ratera, Miriam S. Weber, Anja Seybert, Sergi Torres-Puig, Luis Gonzalez-Gonzalez, Julian Reitz, Enrique Querol, Jaume Piñol, Oscar Q. Pich, Ignacio Fita, and Achilleas S. Frangakis. Structure and mechanism of the nap adhesion complex from the human pathogen mycoplasma genitalium. *Nature Communications*, 11(1), June 2020.

[2] Robert Bridson. Fast Poisson disk sampling in arbitrary dimensions. *SIGGRAPH sketches*, 10(1):1, 2007.

[3] E Oran Brigham. *The fast Fourier transform.* Prentice-Hall, Inc., 1988.

[4] Robert L. Cook. Stochastic sampling in computer graphics. *ACM Trans. Graph.*, 5(1):51–72, jan 1986.

[5] Kamil Czapla and Mariusz Pleszczyński. Montecarlo methods to efficently compute volume of solids. In Rocco Fazzolari and Alaa Abdulhady Jaber, editors, *International Conference of Yearly Reports on Informatics Mathematics, and Engineering (ICYRIME)*, number 3118 in CEUR Workshop Proceedings, pages 1–9, Online, 2021.

[6] Ruben Diaz, J. Rice William, and David L. Stokes. Chapter five - Fourier–bessel reconstruction of helical assemblies. In Grant J. Jensen, editor, *Cryo-EM, Part B: 3-D Reconstruction*, volume 482 of *Methods in Enzymology*, pages 131–165. Academic Press, 2010.

[7] Fabian Eisenstein, Radostin Danev, and Martin Pilhofer. Improved applicability and robustness of fast cryo-electron tomography data acquisition. *Journal of Structural Biology*, 208(2):107–114, 2019.

[8] Utz H. Ermel, Serena M. Arghittu, and Achilleas S. Frangakis. Artiax: An electron tomography toolbox for the interactive handling of sub-tomograms in ucsf chimerax. *Protein Science*, 31(12):e4472, 2022.

[9] Achilleas S. Frangakis. It's noisy out there! a review of denoising techniques in cryo-electron tomography. *Journal of Structural Biology*, 213(4):107804, 2021.

[10] Sonja Gorjanc. Intersections of a sphere. *GeomTeh3D*, 2015.

[11] Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585:357–362, 2020.

[12] JA Hidalgo-Lopez, JC Tejero, J Fernandez, E Herruzo, and A Gago. New architecture for rfft calculation. *Electronics Letters*, 30(22):1836–1837, 1994.

[13] Douglas Lyon. The discrete Fourier transform, part 6: Cross-correlation. *J. Object Technol.*, 9(2):17–22, 2010.

[14] Carlos J. Ogayar, Rafael J. Segura, and Francisco R. Feito. Point in solid strategies. *Computers & Graphics*, 29(4):616–624, 2005.

[15] William K Pratt. *Digital image processing: PIKS Scientific inside*, volume 4. Wiley Online Library, 2007.

[16] Majd Raad, Nedal Raed, Moudar Kiwan, and Alexey Yurievich Popov. Calculate the approximate volume of complex-shape solids. *Materials Today: Proceedings*, 80:1214–1218, 2023. Second Global Conference on Recent Advances in Sustainable Materials 2022.

[17] Nico Schlömer. meshplex: Fast tools for simplex meshes.

[18] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[19] W. Wan and J.A.G. Briggs. Chapter thirteen - cryo-electron tomography and subtomogram averaging. In R.A. Crowther, editor, *The Resolution Revolution: Recent Advances In cryoEM*, volume 579 of *Methods in Enzymology*, pages 329–367. Academic Press, 2016.

[20] Shuqiang Wang. 3d volume calculation for the marching cubes algorithm in Cartesian coordinates, 2013.

[21] Cha Zhang and Tsuhan Chen. Efficient feature extraction for 2d/3d objects in mesh representation. In *Proceedings 2001 International Conference on Image Processing (Cat. No.01CH37205)*, volume 3, pages 935–938 vol.3, 2001.