Optimal fitting of protein structures to electron microscopy maps

Dissertation with the aim of achieving a doctoral degree at the
Faculty of Mathematics, Informatics and Natural Sciences

submitted by

Kai Karius

born in Torgau

European Molecular Biology Laboratory Hamburg
2021

Disputation Date:	26.09.2022
Reviewers:	Prof. Dr. Tim Gilberger
	Dr. Jan Kosinski

Contents

Chapter 1

Preface

This section serves as a guide to the thesis and gives a short summary of the content of each of the following sections and its intended purpose. This thesis was written in the wider field of computational molecular biology and the narrower area of structural integrative modeling. The latter is a loose set of computational methods that tries to combine information from different experimental sources to predict a set of possible models of protein complexes or proteins (see 2). The authors original training is mathematics and physics and this has its bearing on how questions were addressed and which questions were addressed. This thesis was written to enable someone with a modern education in computational biology to gain access as easily as possible to the problems that are discussed. It seeks to augment the existing literature and present a coherent and detailed perspective on a limited set of problems. Hopefully, the reader will be able to follow, criticize and advance the authors work if he or she is so inclined.
The *introduction* contains a short survey of the biological notions and the experimental methods that are used to obtain data that is used in integrative structural modeling to predict models. The methods are described in coarse detail with an emphasis on the aspects that might have an effect on the modeling procedure. On the other hand it describes the *Assembline* pipeline, which is the computational and conceptual context of the modeling procedure. Although it is a specific pipeline, many of the concepts discussed are typical for the field, such as the method of systematic fitting, which is described more in-depth.
Systematic fitting provides the list of positions and orientations for the "building blocks" that will be used during the actual modeling procedure. Together with this list, a score is calculated and associated with each specific placement. Subsequently in the Monte-Carlo modeling procedure the positions and orientations are "drawn" from the systematic fitting results according to their score. Therefore, the success of the modeling itself depends to some extend on the systematic fitting. Because of this close connection, the modeling

procedure is described, too.
The present a broad perspective on the essential issues in structural integrative modeling, there is a section on system definition and data curation. (2.5.2).
The discussion of methods has been split into *computational methods* and *mathematical methods* for the sake of readability, although both aspects are heavily intertwined. The former contains an introduction into data structures and algorithms that are important in the field and tries to elucidate the practical aspects. The latter is a high-level introduction into some of the utilized mathematical concepts. An attempt was made to strike a balance between completeness and intuitive accessibility. In both sections, where ever possible, graphical aides have been used to illustrate.
The *results* section tries to account for and contextualize the authors efforts. A lot of the results are software implementations, the results are therefore partially written like a developer documentation, with stretches of user documentation. The actual code basis can be accessed via git at `https://git.embl.de/karius/fitter.git`.
The *discussion* sections expands on the results section and tries to critically assess the results, contextualize them and speculate about possible further developments.

Chapter 2

Introduction

2.1 Systematic fitting

The term "systematic fitting" describes a systematic exploration of possible "fits" of molecular structures in an experimentally obtained electron density map. The term first introduced in [1] and describes a processing step within the *Assembline* software, which enables the modeling of macromolecular complexes using varying experimental data sources. In *Assembline*, systematic fitting was used to precalculate a set of likely positions and orientations, which are later used as a basis to generate an ensemble of models. This highlights the two main aspects of systematic fitting: The *systematic* coverage of all possible positions and orientations of a structure within a density map and the *fitting* score that calculates the "quality" of each fit. For both aspects there are multiple options of realization. Depending on the nature of the experimental data, some of these options perform better than the other. It is in the interest of the modeler to choose a set of methods for systematic fitting that reveals as much information as possible. This is because in *integrative structural modeling* projects information is typically scarce, a circumstance that creates the need to integrate experimental data from different sources in the first place. This thesis explores the role of systematic fitting in this context and tries to provide methods to optimize the information gain of a systematic fitting protocol in the context of an integrative structural modeling project. Since both the fitting and the modeling aspect are deeply intertwined, both of them and their relation to each other will be described in the following sections.

2.2 Protein complexes

The primary target for integrative structural modeling within the *Assembline* pipeline are macromolecular complexes, often protein complexes. Protein complexes can be described in the broadest sense as associations of multiple polypeptide chains. The ambiguity in this description is mirrored in the ambiguity of experimental detection of protein complexes. Depending on the stability and strength of association of the constituent proteins (subsequently referred to as “subunits”) a protein complex may appear to the experimenter as a well-resolved atomic structure with a clear quaternary shape or as a transiently associated group of proteins that are shown to be functionally linked in a cellular environment. In the context of integrative structural modeling mainly those protein complexes that are amenable to structure-resolving experimental methods are of current interest. These complexes need to be stable enough to maintain a discrete number of states of “shape” (subsequently referred to as “conformational states”) as a necessary precondition to determine their structure (see [2]).

2.3 The notion of a protein structure

Since the advent of molecular biology a clear link between structure and function of biomolecules has emerged. Basic phenomena such as binding, enzymatic specificity or allostery are routinely shown to be influenced by the position of single residues or the quaternary structure of the holocomplex in question. The general lack of a predictive theoretical description of molecular interactions and the limitations of experimental data acquisition however demand some concessions to our idea of “structure”. The first clear definition of a notion of protein structure originates in crystallography and describes it as the average position of atoms in the protein crystal. The assumption that a protein structure within the crystal resembles the protein in vivo to some degree then bridges the results of crystallography to biological reasoning. If, for instance, two protein subunits exhibit specific binding sites or an active site can be predicted from the crystal structure, one can test these hypotheses by further experiments such as genetic mutation of single residues within that sites. Beyond topological classification it is possible to focus on the physical and chemical properties of domains. Well-studied examples are distributions of electrical and of hydrophobic patches on protein surfaces. In conjecture with the laws of physical chemistry it is possible to draw conclusions on the binding affinity and specificity of two molecules.

2.4 Experimental methods of structure determination

The very term “structural integrative modeling” presupposes at least two sources of information to integrate. There is a variety of “input data” coming from very different experimental sources and yielding different amounts of information. Each data resource comes with its own idiosyncrasies that may or may not play a role in the modeling procedure. Thus a short introduction is in order.

2.4.1 Crystallography

A glance at figure 2.1, which depicts a recent statistic of the number of structures stored within the ProteinDataBase database (see [3]) that were acquired via crystallographic methods, reveals the one reason for the importance of crystallographic data in integrative structural modeling. Not only does it provide a library of structures of often atomic resolution, but indirectly inform prediction methods that rely on training data. Crystallographic structures will form the bulk of the basic “building blocks” of the structural models obtained. Protein crystallography started out in 1960 (see [4], [5]) and its results have been made available in a central database from 1976 onwards (see [3]). The massive growth shown in figure 2.1 was made possible by advances in theory, acquisition, computational power and user interfaces. The structure determination via crystallography is a multifaceted and prolonged process, a simplified and abbreviated discussion is thus inevitable. Introductions to protein crystallography can be found in [6] and [7].

The experimental setup and principle of acquisition

Crystallography, as the name suggests, requires crystals of biomolecules. These include proteins, amino acids and larger molecular assemblies. While the existence of protein crystals was already a known fact in the 19th century, only with the emergence of the study of inorganic crystals in the 1930s (see [8] for a short historical introduction) via diffraction techniques, crystals of biomolecules became a source of structural knowledge about biomolecular structure. Much like in the case of cryo-electron microscopy (cryo-EM), modern (bio-) crystallography is the result of incremental improvements of different aspects of the crystallographic pipeline. One of the primary challenges is the growing of an actual crystal. Organic crystals differ from their inorganic counterparts in some important aspects: They can hold between 25 % and 90 % solvent, they grow rather slow

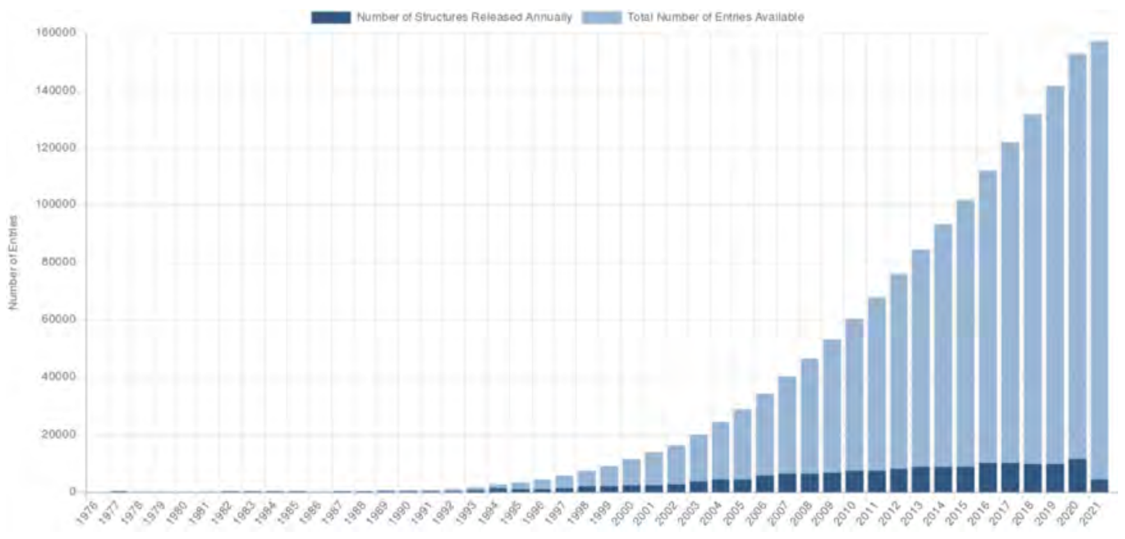


Figure 2.1: A chart representation of the number of structures with the PDB database as of June, '21.

and are comparatively small, they are susceptible to dehydration and mechanical stress and are temperature and pH-sensitive. This fact also increases the empirical value of a crystallographic structure: One may assume that the molecule has a similar structure in crystal form and in its native solution, such as the cytosol. Since smaller compounds may be able to diffuse within the crystal, it is even possible to observe biochemical processes within a crystal, such as enzymatic activity. There is no accepted predictive theory of crystallization of biomolecules as of the date of writing this and the conditions of crystallization have to be found by onerous empirical research, most often specific to each case. One major limitation until the 1980s was the abundance of the specimen needed to grow a crystal. Without genetic engineering only biomolecules with a great natural abundance could be crystallized. Genetics and biochemistry have created a extensive set of tools that can help to obtain a crystal.
If a crystal of sufficient size and quality can be obtained, the researcher can try to record a diffraction image at an x-ray source (see figure (2.2)). The wavelength of the probing radiation must chosen to match the level of detail to be resolved and therefore is in the wavelength range of x-ray radiation. The theory of diffraction holds true in the case of biomolecules, too ([9]): The order of the crystal results in a regular grid of so called unit cells wherein the average position of the atoms constituting the specimen is constant. Hence the atom form a global grid which results in a selective reflection of the incoming radiation. These show up as “reflections” - spots of varying intensity that encode information about the structure of the unit cell. Due to the fact that one measures only the intensity of the reflections on the screen and not the phase (a radiation wave in approximation being completely determined by the amplitude and the phase), there is the so called phase problem. A number of ways to solve the phase problems have been devised, nowadays almost automatizing the process. The acquired images record so called

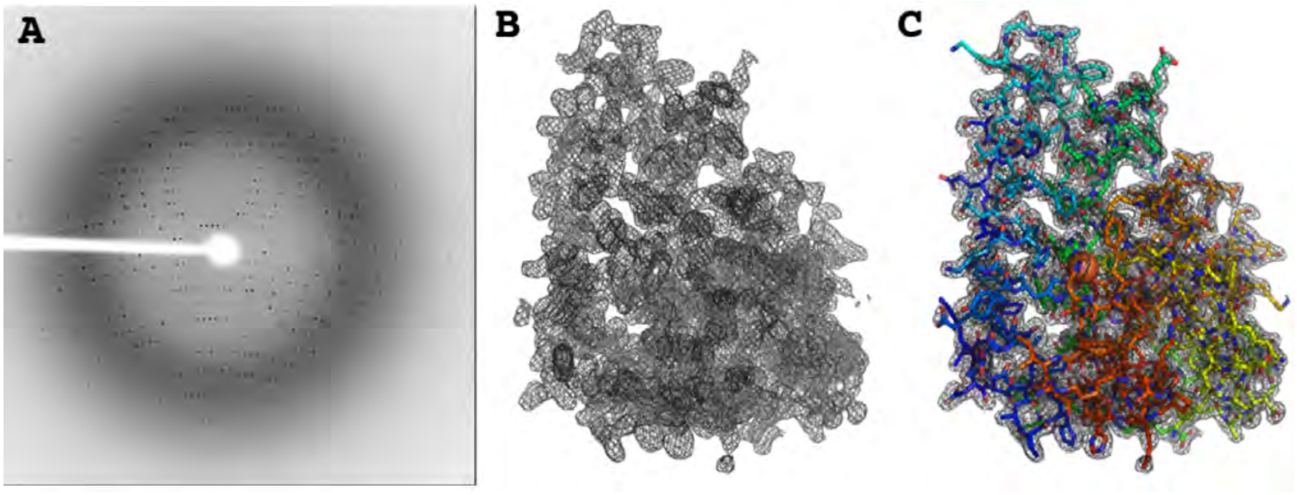


Figure 2.2: An example of a diffraction image, the associated density and the resulting structure. Taken from [10].

refractions, the “dots” that can be seen in figure 2.2. These refractions encode information about the electron density. Once a sufficient set of refractions have been recorded, and iterative process of prediction and refinement can yield a high resolution structure of the specimen.

The Crystallography workflow

Even a preliminary understanding of the crystallographic workflow can be valuable to the modeler, to assess the validity and character of the data resources provided by a structure obtained via crystallography. A brief discussion is therefore in order. The following subdivision of the procedure lends itself: Obtaining a crystal, acquiring diffraction images, and constructing and refining a structure.
The growing of a high-quality crystal can be the most challenging step in the procedure. First of all, a sufficient number of specimen have to be obtained. This is done with the tools of genetic engineering available nowadays and also limited by them. Not every specimen will by easy to express in a large quantity or to purify, often the biological function of the biomolecule in question itself might complicate this process, such as disordered regions or highly charged domains. Optionally, genetic engineering is employed to modify the specimen to make it more amenable to crystallization. The goal of the next step is to find the conditions of crystallization, which may not exist at all. Several parameters, such as the temperature, the pH-value, concentrations or precipitation agents might be varied to find a set of conditions that enables crystallization. Once this is done, there is the need for optimization since a high quality crystal is more likely to yield a low resolution structure. If these conditions are optimized, a supersaturated solution is created to first nucleate and then grow crystals. A supersaturated solution is in a non-equilibrium

Physical	Chemical	Biochemical
1. Temperature/temperature variation	1. pH	1. Purity of the macromolecule/nature of impurities
2. Surfaces/heterogeneous nucleants	2. Precipitant type	2. Ligands, inhibitors, effectors
3. Methodology/approach to equilibrium	3. Final precipitant concentration	3. Aggregation state of the macromolecule
4. Mother-liquor volume	4. Ionic strength	4. Post-translational modifications
5. Geometry of chamber or capillary	5. Cation type and concentration	5. Source of macromolecule
6. Gravity	6. Anion type and concentration	6. Proteolysis/hydrolysis
7. Pressure	7. Degree of supersaturation	7. Chemical modifications
8. Time	8. Reductive/oxidative environment	8. Genetic modifications
9. Vibrations/sound/mechanical perturbations	9. Concentration of the macromolecule	9. Inherent symmetry of the macromolecule
10. Electrostatic/magnetic fields	10. Metal ions	10. Degree of denaturation
11. Dielectric properties of the medium	11. Initial precipitant concentration	11. Isoelectric point
12. Viscosity of the medium	12. Cross-linkers/polyions	12. Unstructured regions
13. Rate of equilibration	13. Detergents/surfactants/amphophiles	13. His tags, purification tags
	14. Non-macromolecular impurities	14. α-Helix content
	15. Chaotropes	15. Conformational states
		16. Thermal stability
		17. Allowable pH range
		18. History of the sample

Table 2.1: A list of possible factors in the process of crystallization of a biomolecule, taken from [8].

state, where the solvent is present beyond its own solubility limits. This then leads to precipitation, ideally in form of a crystal. There are several methods to introduce a supersaturated state, all of them change the physico-chemical properties of the solution in some way. Figure (2.1) offers an overview of the multitude of factors bearing on the crystallization process.

Once a crystal is grown, a data acquisition at an x-ray source can be attempted. One seeks to record as many reflections as possible, the information contained in the reflections contributes directly to structural information. A skilled practitioner may read a wealth of information just from the pattern, though there are accessible software solutions for many given tasks in this step. Should the unit cell exhibit a symmetry, this will also show up in the diffraction patterns. Although this information is strictly speaking redundant, it can be used to attenuate the systematic error in the intensity measurements. Before the advent of molecular replacement methods to solve the phase problem, isomorphous replacement methods, where the crystal is soaked with a heavy atom solution and the difference between the refraction patterns is used to obtain the phase information, can be attempted in this step. The final images are then normalized and processed to yield a dataset which consists of the reflections (identified by their Miller indexes ([9])) and their associated intensities. Given this information, a reconstruction of the structure can be attempted.

The last step is an iterative process that refines the structural model. The wealth of already existing structures can be leveraged to solve the phasing problem by predicting an initial structure, calculating the corresponding the so called patterson map and subsequently minimizing the difference between the predicted and the measured map (molecular replacement). If the resulting density map allows, a structure can gradually be built into it, starting with the backbone and continuing with the sidechains. This, then, can be used as the basis for another iteration. An important measure of quality is the so called R-factor, which is calculated by assessing the difference between the diffraction image of the experimental and the predicted structure. Given the nature of biomolecular crystals not every region of the resulting density might be ordered, stable or occupied by the molecule itself. The so called B-factor can be associated with every predicted atom, indicating the degree of its “rigidity” (the B-factor is also referred to as “Temperature factor”).

Crystallographic density maps as modeling resource

Crystallographic structures are near indispensable in integrative structural modeling. They constitute in a way the “base-line” of structural information, without their information about the the positions of atoms the search space of possible conformations would be too big entirely to effectively search through. The idiosyncrasies of crystallographic maps therefore have a direct bearing on the modeling procedure.

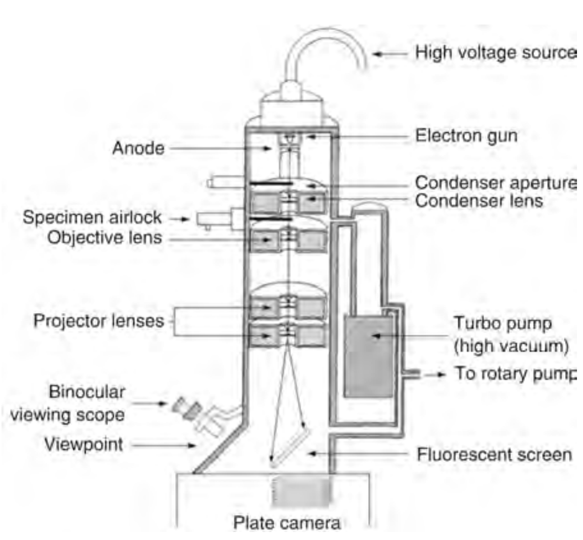
While nowadays a quality assurance at the end of the refinement procedure is a given, it might still pay to check the basic metrics of quality, such as steric clashes and Ramachandran outliers. The crystallographic map also does not show the protein per se, but the protein within the unit cell. Especially in the case of protein complexes there might be a difference in conformation between the protein in crystal and in the complex. Also, naturally disordered regions, such as loops, that are stabilized within the crystal might not necessarily be ordered within the proteins native state. To assess such questions the B-Factors can be used as a valuable resource. Regions with high-temperature B-Factors can be interpreted as being flexible and hence provide an idea of possible conformational flexibility of the protein structure. Lastly, any crystallographic structure should be assessed for possibly present water molecules, ligands and other remnants of the acquisition method and if they should be part of the modeling effort.

2.4.2 Single particle cryo-EM

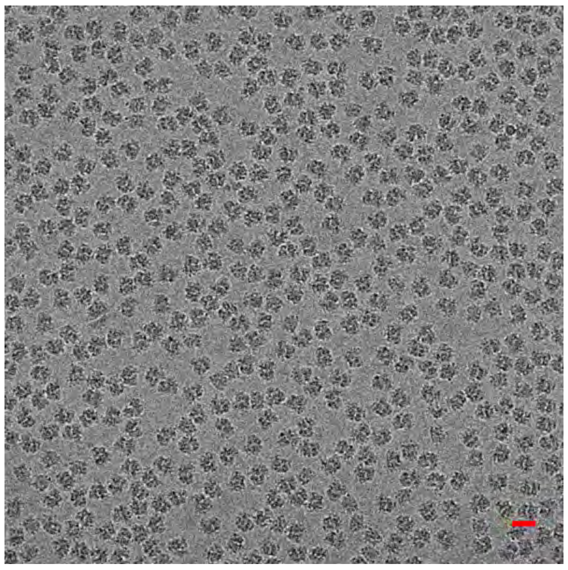
Density maps obtained by single particle cryo-EM can overall be said to be an important, information-rich data source for integrative modeling. This discussion will highlight the basic principle, provide a rough sketch of the work flow and offer a perspective from the view of integrative structural modeling.

The experimental setup and principle of acquisition

Figure (2.3a) shows a schematic of a cryo-EM microscope. As opposed to x-ray crystallography the probing particles are not photons of the x-ray wavelength but a beam of electrons. This difference has far reaching consequences in questions of methodology



(a) A sketch of a cryo-EM microscope, taken from [2]



(b) Single particle raw image, taken from [11], the red line signifies 300 Å.

Figure 2.3: Experimental setup and raw data of an cryo-EM single particle experiment.

and experimental opportunity. This description will exclusively focus on single particle cryo-EM. Arguably the most interesting aspect of this kind of cryo-EM microscopy is the possibility of structure determination without the need of the sample to crystallize. Every imaging method must somehow account for averaging out noise to reveal the immutable structure underlying the image. In crystallographic acquisitions this averaging process is largely achieved by the crystal itself, resulting in a diffraction image which in favorable circumstances contains all the information needed to reconstruct an atomic structure. In single particle cryo-EM this averaging is achieved by recording projections of a large number of particles of the specimen and applying a computational pipeline to reconstruct a three dimensional density map. Figure (2.3b) shows an example of raw cryo-EM data for the case of eukaryotic ribosomes, the images of the particles oriented in varying orientations will be the basis for the reconstruction process.
Due to the fact that electrons are scattered by air the inside of the microscope has to be kept in a vacuum state. This poses a problem for biomolecular samples, since they occur intrinsically hydrated and evaporation would distort their natural state. Two of the most prominent techniques to deal with this are the embedding in vitreous ice (hence $\underline{\text{cryo}}$-EM) and the staining of the specimen in a heavy metal salt, known as *negative stain.* The average free path length of electrons in vitreous ice of 2800 Å for elastic scattering and 750 Å ([2]), which requires the sample to be delivered in a thin film. The resolution is limited by the setup of the microscope itself and is far beyond the de-Broglie wavelength of the electron, the highest amount of details that can be resolved is around $1 - 2$ Å for the case of bio-molecular complexes (ibid.) The electron beam itself has its source at the top in the electron gun and is modulated in angle of convergence and magnitude by

the condensers. The sample itself is right on top of the objective lens, which forms the image that is then projected onto a fluorescent screen or, for data acquisition onto a CCD camera, by the projector lenses. The lenses are magnetic since they need to bend the path of electrons.
The interactions within an EM-microscope are reported to be complex and beyond this discussion; a brief descriptive attempt for illustration will be made. One is mainly interested in elastic scattering events for high resolution reconstructions. Elastically scattered electrons, in the weak-phase-object approximation (which includes thin films), have their phase shifted in an amount proportional to the integral of the electrostatic potential (corresponding to the density which one is interested in). Therefore one is interested in this phase shift. The phase shift itself has an effect on the intensity finally recorded by the camera. The so-called Contrast Transfer Function (CFT) theory details the contribution to this phase shift resulting from the experimental setup. If one is able to account for this contribution one can reconstruct what essentially can be understood as a projection of the three-dimensional density to the two dimensional image plane.
Given a collection of these projections and the standard computational pipeline ([12]) one is ideally able to construct a three-dimensional reconstruction of the electrostatic potential.

The single particle cryo-EM workflow

The single particle cryo-EM workflow has been object to several optimizations of both the experimental setup and computational procedures. Virtually all of the steps have an impact on the final result and sometimes the interpretation of the resulting density map. A brief discussion is therefore in order (for a more complete perspective, see [12]).

- Sample preparation is paramount for cryo-EM structure determinations. The more structurally homogeneous the sample is, the better. A biophysical characterization of the sample can be beneficial to assess composition and conformation. In the case of protein complexes it might not be a given that the tentative complex actually forms a stable holocomplex. There might be different conformational states, which is no problem in itself, if their number is not too big and every state is represented by a good number of particles. Negative stain EM is also a possible way to assess this, more so since it requires less particles. Several methods exist to stabilize complexes: crosslinking, a change in buffer conditions or adding a stabilizing ligand to the sample. A large number of particles is beneficial for the reconstruction, too.
- The specimen sample is commonly prepared on a carbon film which is supported by the grid. As mentioned, to prevent dehydration the specimen has to be either

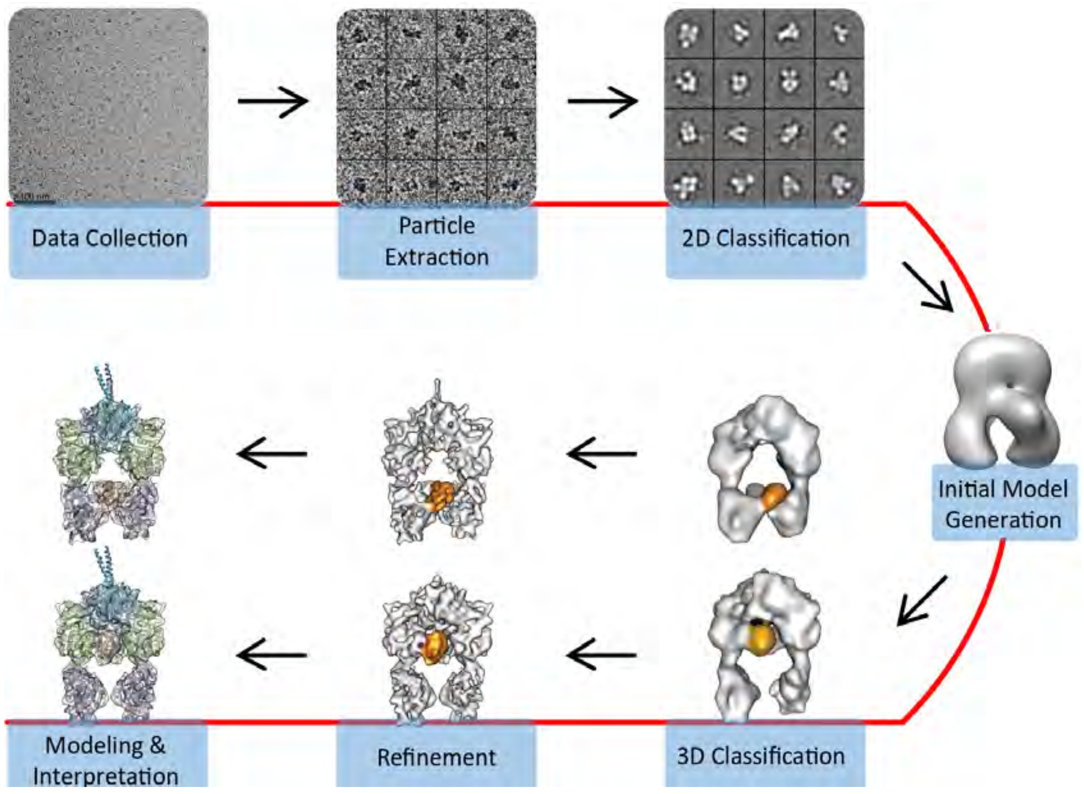


Figure 2.4: An illustration of the image processing workflow of the single particle cryo-EM procedure, taken from [13]

vitrified or stained in a heavy salt. In the case of vitrified ice the ice layer needs to be as thin as possible but not too thin to exclude the particles from the middle or induce a preferred orientation. The latter phenomenon is more prevalent in some species than others and might lead to a bias in the projections. Automatized plungers can reliably reproduce vitrified specimen without impurities within the ice.

- The image acquisition is a step which, although many intermediate steps have been automatized, holds a large number of parameters and details that need optimization. For high resolution results the microscope needs to be aligned well and a number of parameters, such as spot size or the appropriate defocus settings, can be adjusted to improve contrast. A lower defocus will result in more high-resolution information due to the mathematical properties of the contrast transfer function theory, which also holds the cause for the need to acquire at a number of different defocus settings. When adjusting the electron dose there is a trade of involved between higher contrast with a higher dose and more radiation damage, which one seeks to minimize. A lot of optimization is taken care of for the user by modern solutions, such as the correction for movements of the specimen within the ice or weighting later acquisitions with a higher "B-factor" to correct for radiation damage which will accumulate in later acquisitions.

- Given the final set of images, the image processing part of the pipeline can be undertaken. Figure (2.4) illustrates some of the most important steps. The picking of particles out of the raw images can be initiated by hand and helped along my software to automatically find them. The next step is the optional two-dimensional image classification, based on the standard technique of k-means clustering. Once a suitable set of classes is found, aligned and averaged, the process of three-dimensional reconstruction can begin. One may choose an initial structure in this iterative process, such a previously obtained negative stain density map. It must be said that it is dangerous to introduce bias either at the 2D classification or 3D reconstruction, see [14], since overfitting might occur quite easily. There is the option of reconstructing more than one conformation from the same data set, resulting in multiple distinct density maps. An important measure of quality is the Fourier Shell Correlation (FSC), which is calculated by splitting the same data set into two and subsequently reconstructing and comparing two versions of the resulting density using this metric.

Single particle cryo-EM maps as modeling resource

Finally it is important for the modeler to know the pitfalls in this procedure. If a density map is provided to the modeler, for instance, it is worth checking the experimental procedure that have led to this result. For it is easy to assume that a density map, by giving the impression of finality that the results of long and arduous procedures often have, is a signal of an actual existing structure. This does not need to be the case. A good biochemical and biophysical characterization can prevent such fallacies: Was the original sample structurally homogeneous? Is there proof of the protein complex actually stabilizing in a small number of conformations? Also, if two different conformations are the end result of the Single particle cryo-EM procedure, the modeler might want to ask: Are these conformations real or an artifact of the reconstruction process?
An additional point is to be made about the difference between maps obtained via negative stain and vitrified ice. Negative stain does not, by experimental design, reveal information about the internal density of the specimen. Therefore, in later stages of the modeling, it might be worth addressing this difference, for example by using different scoring functions during the systematic fitting (see 2.5.5) step.

2.4.3 Electron cryotomography

Electron cryotomography (cryo-ET) is an experimental technique that is closely related to single particle cryo-EM (see 2.4.2) yet exhibits some key differences that result in different

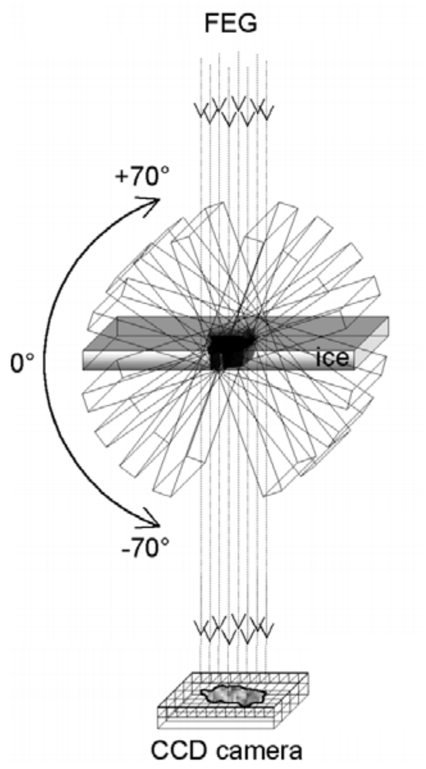


Figure 2.5: Schematic illustration of a tilt-series, taken from [15].

challenges and opportunities compared to cryo-EM. Like cryo-EM, cryo-ET underwent a long historical development of continuous, successive improvements in both its experimental and computational aspects. In conjunction with integrative structural modeling it has been used to solve some key challenges (e.g. [1]) in modern structural biology. This warrants a short introduction to the technique.

The experimental setup and principle of acquisition

As it is for single particle cryo-EM acquisition, an electron microscope (see 2.3a) is necessary for cryo-ET acquisitions. There are however fundamental differences in sample preparation, image acquisition and processing and density reconstruction. Sample is not purified into single particles and exposed by use of a grid. In recent years, it has become possible to expose entire sections of tissue in the form of thin slices. Using the technique known as focused ion beam (FIB) milling, frozen tissue samples or cells are carefully thinned by said ion beam, until they are less than ca 500 nm thick, so that the electron beam can penetrate them. A pre-screened and suitable sample is mounted in the vacuum of the electron microscope and subsequently a so called tilt-series is recorded.
This series of acquisitions is central to the technique and illustrated in figure (2.5). The sample is tilted in a series of angles typically ranging from $-70°$ to $70°$ and an image is recorded for every orientation. A lot of the idiosyncrasies of cryo-ET stem from this very

procedure. In contrast to cryo-EM the same area is exposed to multiple radiation dosages so that the intensity needs to be lower to not degenerate the sample. The tilting causes a stage drift for which one must account with tracking and autofocusing procedures. Different tilt angles have different thicknesses and a different geometric relation to the electron beam. Together with the low dosage this makes defocus determination more challenging and worsens the signal to noise ratio. The tilt images are recorded in sequence, so the radiation damage accumulates in later images so that a tilt scheme must be selected with care to preserve as much high resolution information as possible. E.g. multiscale mapping is used to focus on promising regions while at the same time keeping the radiation dosage low when finding these regions. This is a small impression of the considerations involved in the acquisition process. A more exhaustive list can be found in [16].
The image processing, similar to single particle cryo-EM and possibly more so, relies on additional information, such as the tilt scheme and the acquisition settings, to reconstruct the resulting densities. While there are again similar features, such as the sample preparation and gauging of the microscope, there is an outstanding difference. There is an intermediate 3D-density that is a first reconstruction of the tilted images. This density has the important property that it is constructed from an incomplete coverage of angles (see above) and is to be considered as an anisotropic representation of 3 dimensional space. With this proviso, more familiar computational techniques such as particle picking, alignment and density classification are employed. This section is a terse description to contrast cryo-ET with single particle cryo-EM, a more faithful descriptions of the workflow will be given in the next section.

The cryo-ET workflow

The workflow will be sketched closely based on the excellent discussion in [16]. The focus will be on the differences to the already discussed methods in single particle cryo-EM (see 2.4.2). Figure (2.6) shows a possible representation of the workflow.

- The sample preparation, microscopic image acquisition and the image processing have many common aspects with their counter parts in single particle cryo-EM. The differences result from the use of a tilt series instead of the (often) naturally different orientations in single particle cryo-EM. An advantage is that the samples do not need to be single particles, therefore, a broader range of objects of study is accessible to cryo-ET, such as protein complexes in situ ([17]). A disadvantage stem from the fact that within a tilt-series the same area is hit by an electron beam and therefore the radiation damage will accumulate in the later tilts of a series. In general, image processing, the design of the tilt series and the acquisition

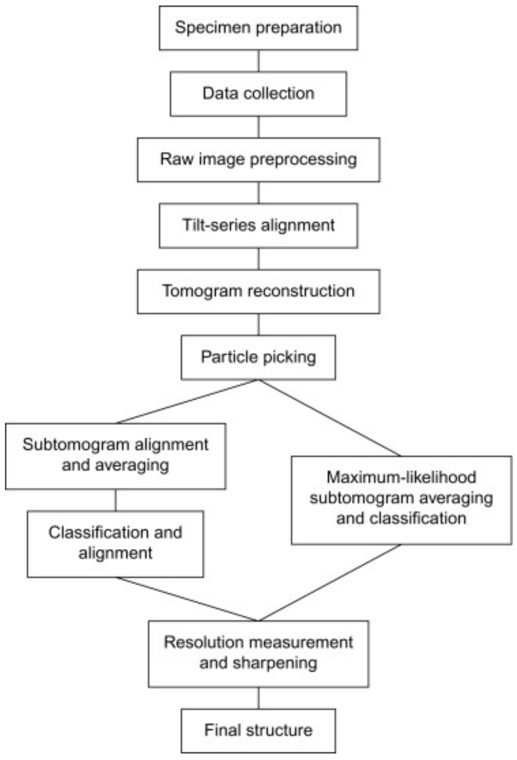

Specimen preparation
Data collection
Raw image preprocessing
Tilt-series alignment
Tomogram reconstruction
Particle picking
Subtomogram alignment and averaging
Classification and alignment
Maximum-likelihood subtomogram averaging and classification
Resolution measurement and sharpening
Final structure

Figure 2.6: A schematic of a cryo-ET workflow, taken from [16].

parameters have to take into account the geometrically different situation of each tilt ([18]). After the acquisition of a tilt series, the tomogram reconstruction can be achieved by methods such as the Fourier synthesis.

- The actual, 3D space reconstruction is obtained during subtomogram averaging. This process is very similar to its single particle cryo-EM pendant, a notable difference is the three-dimensional nature of the tomogram. The missing orientational information caused by the angular wedge must be taken into special consideration. This is easier to meet if the sample is isotropically oriented (when statistical averaging might help) than in the anisotropic case. The rotational alignment bears close resemblance to the systematic fitting counterpart and has similar properties, such as being computationally expansive and being best approached in an iterative, refinable manner.

Cryo-ET maps as modeling resource

Cryo-ET maps are not easy to obtain due to the complex pipeline and the relative novelty of the method. Yet they bear great promise for integrative structural modeling for they have two properties that play into its strengths: One is that they have a intrinsically lower signal to noise ratio and will more likely produce low resolution results, a demand that integrative structural modeling was practically created to meet. The second is the opportunity to gain information about complexes in situ environments and bio-molecules. It might happen that the resolution of a density representing a more extended cellular environment is less than what is necessary to determine atomic details ab initio. This will lead to a great number of alternative hypothesis about the precise biochemical interactions and arrangements, for example represented by an ensemble of models. Integrative structural modeling is since conception prepared to deal with many alternative hypothesis and is accustomed to dealing with the degree uncertainty arising from a broad ensemble of models.

2.4.4 Concepts of resolution

Due to the varied nature of data sources in integrative structural modeling, the modeler is confronted with a variety of measures of data quality. Most prominently among these measures is the concept of resolution, because it is connected to crystal structures (2.4.1), single particle cryo-EM density maps (2.4.2), cryo-ET maps (2.4.3) and the simulation of molecular densities (3.2.3). The concepts of "resolution" connected to these densities are closely tied to their respective idiosyncrasies. A comprehensive elucidation of their interrelation is an ongoing task over the last decades and will in all likelihood not be possible too soon. Due to the frequent encounters the modeler has with these concepts, it seems however a good idea to have the briefest mention and some literature references.
Crystallographic resolution has more than one instance as a concept ([19]). The most simple definition is d_{high} as the smallest distance (in $\mathring{A}$) that is resolved in the diffraction pattern. This diffraction pattern is already a result of a statistical clean up in the standard pipeline, it gives however a rough idea about the *global* quality of the data. The range of $0\ \mathring{A} - 1.5\ \mathring{A}$ is called "atomic" and is of great interest to the structural integrative modeler, since his or her models will often derive structures from maps of this range.
The *cryo-EM/ET resolution concept* is seemingly universally mentioned as being topic of an "ongoing discussion". There are multiple measures and perspectives ([20],[21]). The most used concept is that of the FSC (Fourier Shell Correlation) at an agreed upon cutoff value. This procedure roughly measures the consistency of the reconstruction process by calculating the correlation of different spatial frequencies k (defining Fourier shells) of two independent reconstructions from the same dataset (the data set is halved). Where this consistency, measured via the correlation, falls below the agreed upon cutoff value, the spatial frequency parameter k is read and its inverse is defined as the resolution. The dominant issues are the exact cutoff value, the role of noise and the influence of different processing steps. The modeler needs to be aware that the "resolution" might vary locally and that the number given to him has to be assessed critically.

The *local resolution after ResMap* ([22]) has been developed for this very reason. It

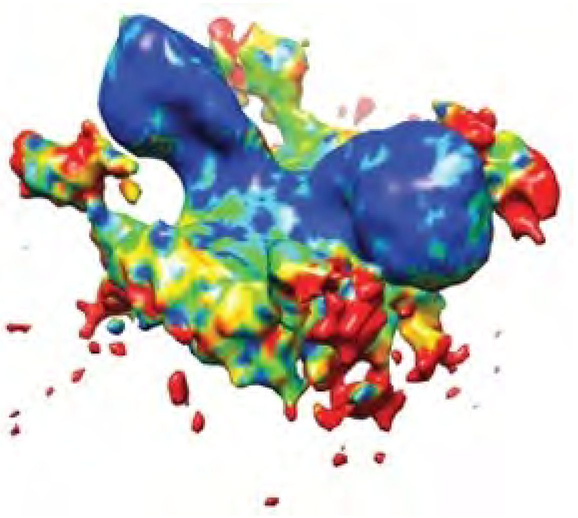


Figure 2.7: Example of a map processed with ResMap. Taken from [22]. The blue regions correspond to ca. 30 $\mathring{A}$ and the red regions to ca. ca. 58 $\mathring{A}$.

assess the resolution of *each voxel* using a statistical procedure to test for the presence of *features* of different wavelengths. This helps the modeler to assess different regions of the map displaying different levels of details and consider the modeling decisions accordingly. Figure (2.7) shows an example.
Lastly, the *resolution factor* of the density map simulation mimics the effect of high and low resolutions. Often it relates to the width σ of a Gaussian, there are different conventions how the nominal resolution of the cryo-EM/ET density map should relate to this σ. To the authors knowledge, none of them have been proven intrinsically superior to the other, either analytically or empirically. Some of these conventions can be found at [23].

2.4.5 Crosslinks

Crosslinks are widely used in integrative structural modeling, for they are acquired in a wholly different manner than crystallographic or cryo-EM data. This makes them an orthogonal data source and a valuable tool for cross validation. A cross-linker is a class chemicals whose members consist of two parts: Reactive end groups and a spacer. During the crosslinking chemical reaction the reactive end group may react with residues to form covalent bond with them, so that these residues are now connected by a spacer of known length (e.g. thebis(sulfosuccinimidyl)suberate (BS3) or disuccinimidyl suberate (DSS) cross-linker of length 11.4 $\mathring{A}$). This information can be used in integrative structural modeling procedures.

Although there is a growing number of variations of the crosslinking technique, the general procedure can be outlined in a few steps (Figure 2.8). Crosslinking can be attempted

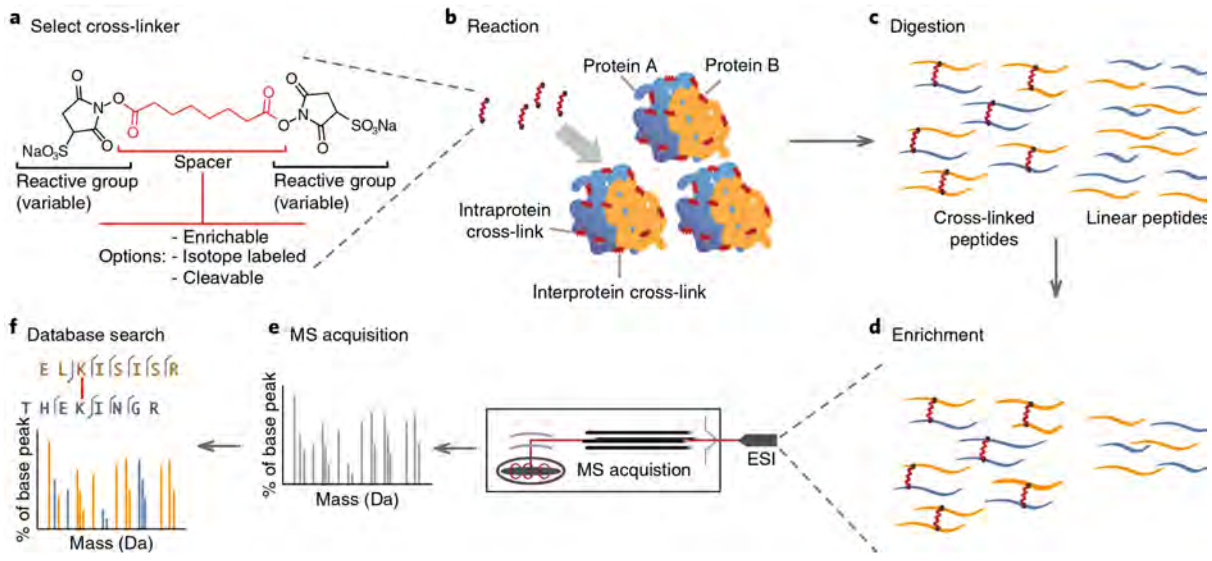


Figure 2.8: An illustration of the crosslinking technique from a general perspective, taken from [24].

in a purified protein sample or in vivo, depending on the intentions of the experimenter. If solely the isolated structure of the protein complex is of concern, one would gravitate towards the first option. The crosslinking reaction itself needs to be optimized depending on the crosslinker and the protein and the intended results, parameters to be optimized are, amongst others, buffer and reaction time. Cross-linkers typically attach to a specific subset of peptide residues, depending on the type of cross-linker. After digestion of the crosslinked protein sample the resulting mixture of crosslinked and linear peptides is enriched for the former. After mass spectrometry data acquisition using the enriched peptide sample, database search and statistical evaluation based on the concept of FDR (False Discovery Rate, see [25]) is performed and the list of crosslinked residues associated with a measure of certainty is made available to the modeler.

A number of concerns are of interest to the modeler. First, crosslinks in the context of protein complexes can be classified into three distinct categories: *monolinks* - crosslinks that are attached to just one residue, *intralinks* - crosslinks between two residues of the same subunit of the protein complex and *interlinks* - crosslinks between two residues of different subunits of the complex. While the information content of the different types of crosslink varies, all of them are potentially useful within a modeling project and shouldn't be disregarded carelessly. Besides the already mentioned fact that crosslinks constitute orthogonal information with regard to the other data resource types, there is the fact that crosslinks are sensitive to conformational changes. In one sample of crosslinks there might be encoded two or more different conformational states. Trivially, crosslinks that are satisfied in one state might be violated in another. Lastly, the geometry of the crosslinker determines the "resolution". A short crosslinker will offer more precise structural information than a longer crosslinker but is also less likely to actually crosslink any residue.

2.5 Structural integrative modeling

Structural integrative modeling comprises a multitude of computational methods that are employed to combine multiple differing data sources to produce a single final *model* or an ensemble thereof that agree with the data. A model can be represented as either a set of atomic positions that constitute a biomolecular structure, a sequence of conformations ("snapshots") if the goal is to model a kinetic process, or an ensemble of equally likely conformations that can alternatively be represented as a probability density. The data resources used as an input are rather diverse (see 2.4). In principle any data resource that has some bearing on the spatial conformation or conformational changes of a biomolecular complex can be used as an input. Both the intended form of representation of the resulting model/models and the specific nature of the experimental information provided by the data resources have a massive influence on the methods and tools used in structural integrative modeling. In this thesis, I will limit myself to the techniques used within the *Assembline* protocol ([26]) and expand on specific aspects such as the task of *fitting*, which is a collection of methods that attempt to correlate crystallographic data with electron microscopy data. To situate this thesis within a broader scientific context, a brief survey of existing work is appropriate.
Assembline itself relies chiefly on the IMP (Integrative Modeling Platform, [27]) and UCSF Chimera ([28]). IMP itself does not define a single protocol but offers a framework for integrative modeling and a wide range of tools to this end. It comes with an exhaustive online documentation and rich set of examples for widely varying situations a modeler might encounter. Its programming interface is available in `C++` and `Python` languages ([29]), which is actively maintained as of writing this. There are other software tools for integrative modeling. For example, the HADDOCK software protocol (see [30]) focuses on identifying interfaces within protein complexes ("docking"). They offer the possibility to integrate chemical shift data from NMR experiments and mutagenesis data to predict possible binding sites. The relative position of subunits, subcomplexes or domains within the protein complex is rich in information and valuable for modeling. HADDOCK offers a server, where docking jobs may be submitted also by inexperienced users. PyRy3D (see [31]) bears conceptual resemblance the *Assembline*, its main objective being to use Monte-Carlo based methods to predict the structure of macromolecular assemblies. It is accessible as a user friendly (their claim) web server. The Rosetta software suite (see [32]) offers a wider array of protocols, including docking, ab initio structure prediction, structure prediction of macromolecular complexes and local optimization of structures. Its interfaces include a command line interface, Python interfaces and publicly accessible servers.

2.5.1 Structural Integrative Modeling as implemented in Assembline

The Structural integrative modeling pipeline *Assembline* as outlined in [33], [26] and [34] consists of the following steps:

1. Data collection and curation. In this step the data is obtained from an experimenter or online databases and is critically evaluated, cleaned of unnecessary content and artifacts. After this, the data must be registered in a central file or data structure, so that it becomes clear how each of the different data files containing density maps, atomic structures or crosslinks relates to the actual protein complex in question.

2. The first computational step is called "Systematic Fitting". For every structure that has been obtained in the last step an exploration of its possible places within the EM map of the complex in performed. This can happen in a number of ways: To determine a set of possible positions one can either choose the Monte Carlo method or a regular grid. The "quality" of each positioning can be assessed with a number of computational functions (see 2.5.5).

3. The second computational step involves setting up the model sampling procedure. This procedure can be a kind of Monte Carlo procedure, the Metropolis-Hastings algorithm, a gradient descent algorithm or a mixture of similar tools. At the beginning of this step, the model objects are encoded as computational representations, as are their relations between each other. A total score expresses the quality of each sampled model. The list of positions and orientations of the rigid bodies that will be drawn from to sample different models are obtained in the systematic fitting step. Additionally the scores associated with the positions and orientations are used as input for the Monte-Carlo scoring function.

4. In the last step, the top models are searched out and examined. Additionally one wants to make sure that within the model sampling step, an exhaustive number of models have been sampled, to not miss out on significant portions of possible models.

I will outline the details of each one of these steps and remark on their significance to each other and the procedure as a whole.

2.5.2 System definition, data collection and data curation

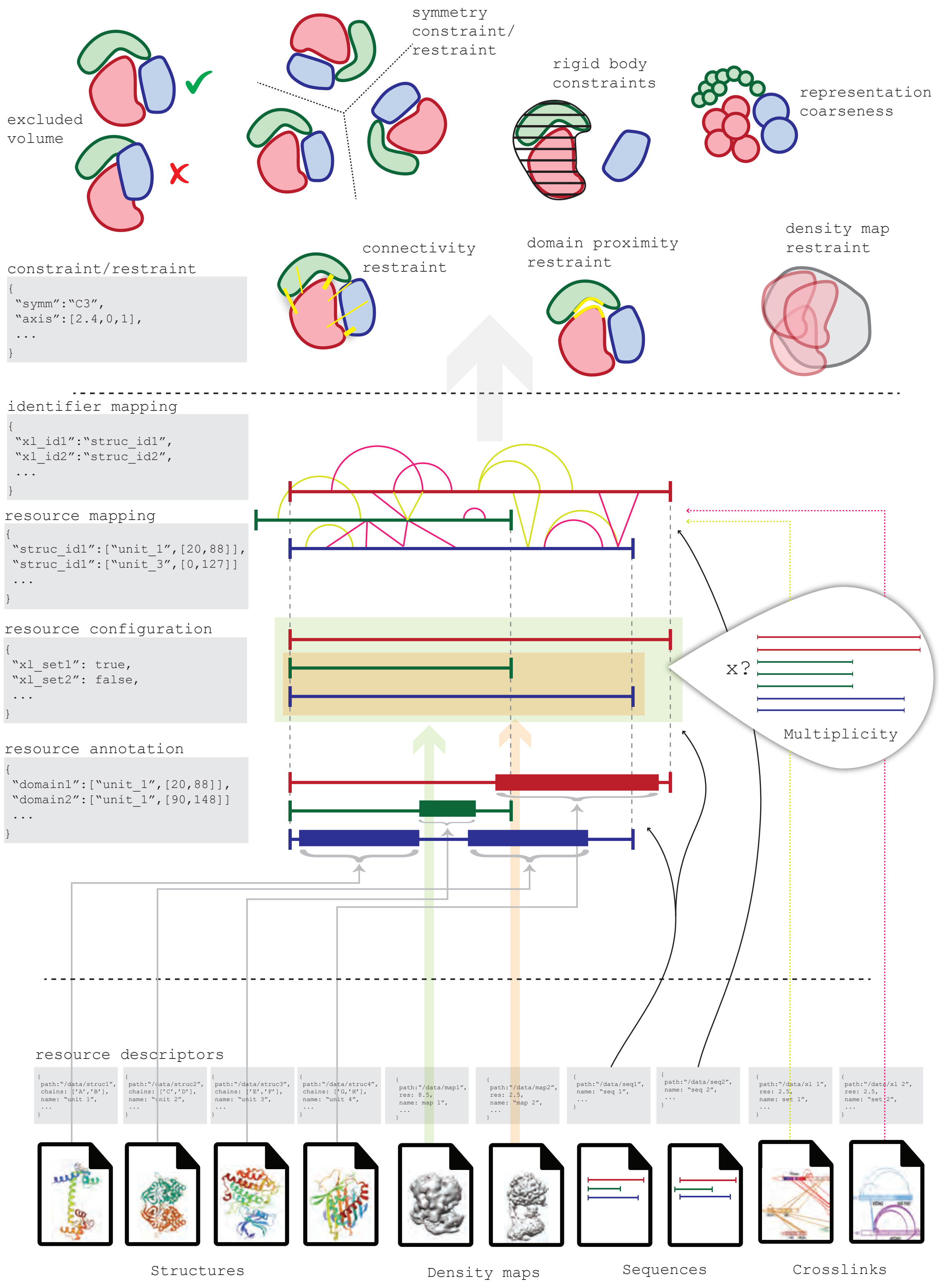
This step is an essential and laborious procedure whose successful execution determines the possibility success of the integrative modeling process. The central challenges are:

- The definition of the system in terms that are both intuitive to the biologist performing the modeling procedure and the computational system that executes the necessary steps is non trivial problem. While the biologist employs visual intuition that is very amenable to human imagination, this does not translate into computer logic, which is distinctly sequential. Further complication arises from the fact that different data sources that are to be integrated during the modeling procedure might follow different conventions that must carefully be translated into each other.
- Data collection includes interfacing with online databases, which might hold information that might contribute to the overall picture of the model. This includes e.g. crystal structures, cross-links, homology models and even information about protein-protein interfaces. This aspect of preparation is heuristic, meaning one might want to come back to it and reiterate it after initial modeling attempts.
- Data curation includes creating a clean register of the data sources and evaluating it critically. The former aspect of it is necessary due to the heuristic nature of integrative modeling processes and the likely need to return to this step and for reevaluation. The critical evaluation is needed to counteract possible faulty data sources, contradicting data sources and data sources that come with the need of initial interpretation.

In the following I will discuss and illustrate these tasks.

System definition

The definition of the system and its relation to the resources and the results of the modeling procedure can be quite demanding and may undergo change during multiple iterations of the same modeling project and can be very diverse depending on the nature of the specific modeling project. The aspects of this step are, while related, of a varying nature, I will refer to their sum as the “model system definition”. One possible perspective on this model system definition is to look at it as a tripartite structure, as outlined in (Figure 2.9), which illustrates the guiding principle of this discussion.
The three parts outlined can be called the *resource description layer* (bottom), the *model consistency* layer (middle) and the *geometry specification* layer (top). I shall discuss them



constraint/restraint
```
{
 "symm":"C3",
 "axis":[2.4,0,1],
 ...
}
```

identifier mapping
```
{
 "xl_id1":"struc_id1",
 "xl_id2":"struc_id2",
 ...
}
```

resource mapping
```
{
 "struc_id1":["unit_1",[20,88]],
 "struc_id1":["unit_3",[0,127]]
 ...
}
```

resource configuration
```
{
 "xl_set1": true,
 "xl_set2": false,
 ...
}
```

resource annotation
```
{
 "domain1":["unit_1",[20,88]],
 "domain2":["unit_1",[90,148]]
 ...
}
```

resource descriptors
```
{
 path:"/data/struc1",
 chains: ['A','B'],
 name: "unit 1",
 ...
}
```
```
{
 path:"/data/struc2",
 chains: ['C','D'],
 name: "unit 2",
 ...
}
```
```
{
 path:"/data/struc3",
 chains: ['E','F'],
 name: "unit 3",
 ...
}
```
```
{
 path:"/data/struc4",
 chains: ['G,'H'],
 name: "unit 4",
 ...
}
```
```
{
 path:"/data/map1",
 res: 8.5,
 name: map 1",
 ...
}
```
```
{
 path:"/data/map2",
 res: 2.5,
 name: "map 2",
 ...
}
```
```
{
 path:"/data/seq1",
 name: "seq 1",
 ...
}
```
```
{
 path:"/data/seq2",
 name: "seq 2",
 ...
}
```
```
{
 path:"/data/xl 1",
 res: 2.5,
 name: set 1",
 ...
}
```
```
{
 path:"/data/xl 2",
 res: 2.5,
 name: "set 2",
 ...
}
```

Figure 2.9: A tripartite perspective on the system definition in integrative structural modeling (*Assembline*)

in this order.

The *resource description layer* gathers and describes the raw data instances which hold the information that is to be combined into a number of possible final models. Although it is in principle not restricted to the data types that will be outlined, they constitute the most common data sources. The FASTA files (`.fasta` or `.fsa`) contain pairs of sequence identifiers and amino acid sequences, which typically constitute the subunits of the protein complex. They are provided by the experimenter. Different kinds of experimental data sources might demand different sequence instances of the same underlying protein complex, e.g. there might be differing sequences for the complex that was used in a cryo-EM data acquisition and reconstruction and the crosslinking dataset (a HIS-tag for the pulldown). This difference cannot be neglected, since the crosslinking dataset might refer to a different sequence register and can only be correctly mapped in this register. Therefore it is possible to have multiple data sources for the sequences, which might otherwise be thought of as immutable. Still the sequences are the most stable (unlikely to be changed by the modeler) entity, which is why they can be used to relate all other data sources to each other. This data source does usually not require a further specification of the modeler.

The files (or database identifiers) containing the structural information might the most variable source of information. There are two common formats, the `PDB` ([3]) and the `CIF` (Crystallographic Information File). Both contain information about the average position of residues and atoms. They may hold several non-covalently linked *chains*, water molecules, ligands and particles of a more exotic type. Their count register and the chain identifier specifies the covalently linked subunits. They may also contain a wealth of further information, such as crystallographic B-Factors and information about chemical bonds. Though this is already a data source rich in information, the sum of all structure files adds a layer of complexity. The modeler might for example wish to only address a subset of data in a structure file, use multiple different versions of a structure represent by several files, concatenate different structure files into one structure or break up an existing structure represented by one file into parts to simulate flexibility. To enable these operations, the structure files can be described and annotated in resource descriptors, which are either `JSON` (JavaScript Object Notation) or data objects instantiated using the `Python` programming language (the gray boxes in Figure 2.9) that can hold identifiers, ranges, chain identifiers etc. These descriptors form the binding element that ties the raw structural data into the model system definition.

Of similar importance yet different structure are the density map files, typically found (in the *Assembline* project) in the `MRC` and `CCP4`. They are of a binary format and describe the geometry of the density that is supposed to represent the holocomplex or subcomplexes in question. These files may contain crystallographic information, in our context we usually focus on the geometry of the densities, i.e. the size and shape. For example,

the modeler might want to address only a part of a given density or crop areas of densities characterized as noise. These steps are undertaken either later within the pipeline or as a preprocessing step by the modeler. Additional information about the density is again specified in a resource descriptor. This information includes a parameter called "resolution", which roughly correlates with our basic understanding of "resolution" but does not need to be identical to experimental resolution-like parameters (e.g. the optical resolution in crystallography or the Fourier shell correlation in cryo-EM and cryo-ET) or computational estimates, such as that of the `ResMap` software ([22]). This parameter plays a pivotal role in the systematic fitting procedure. Another information that needs to be specified is the density threshold, which specifies the minimum `float` value that still constitutes the signal of the protein complex. This is a non trivial aspect since including or excluding different subdensities might misrepresent the signal as noise or vice versa. Yet another aspect is the assignment of subunits or subcomplexes of the holocomplex to different density maps, as not every structure may appear in every map. On top of that there might be different versions of a density representing the holocomplex or a subcomplex. The method of experimental acquisition of the density map can play a role, too, since it can change how the map is computationally and geometrically interpreted, as is the case with negative stain EM maps, where the density map is to be interpreted as the signal of the surface of the protein complex, as opposed to the three-dimensional density interpretation. All this information is needed upstream of the *Assembline* pipeline at one point or another and needs to be gathered and specified by the modeler.
The last commonly used data source are files that contain information about crosslinks. These are usually `csv` (Comma Separated Values) files that list different types of crosslinks and associated information. The most essential information is the type of crosslink, the indexes one or two residues that it is linked to, and the degree of confidence that this particular crosslink constitutes a real signal. Similarly to density maps crosslinks might represent the structure of subunits, subcomplexes and the holocomplex and must be annotated as such in the resource descriptor. The specific type of the chemical used as a crosslinker is also important, as it encodes information about the minimum and maximum distance between two residues that the crosslinker can signify. Since crosslinking assay might work with different protein sequences than those utilized for the cryo-EM density map, a reference to the appropriate sequence resource is needed, too.
If this deceptively simple step is conducted carefully it can build a productive foundation for the modeling project, if not, it can cripple the modelers efforts. It ties in directly to the next layer, the *data consistency* layer. Here the data is related to each other and the bridge between different data sources and their respective idiosyncratic conventions is established. While this is the main purpose of this layer, there are a number of aspects to consider.
The entities used as central organizational unit (as indicated in figure 2.9) are the se-

quences. Therefore possibly different sequence identifiers (such as those of the sequences used in crosslinking and those used in the cryo-EM density map) need to be mapped to each other. Concurrently the sequence registers need to be aligned. Since the sequence identities used in `FASTA` files might contain information non essential to the modeling and might be quite verbose, a user defined identifier and the appropriate mappings are common. Once this mapping is accomplished, the structural resources can be mapped to the sequences. This usually happens by mapping the resource specification (file names and chains) to the corresponding sequence. This step is fairly straightforward since the sequences of the structure files (if they are contained within the structure files) have to match the sequence ranges that the structure is supposed to represent and can be used to do an additional consistency check. The mapping of the density map resources to sequences that represent the holocomplexes and subcomplexes can be achieved in a similar way, with the additional complication that different density map resources might correspond to different conformational states of the complex in question. The same considerations and mappings need to be undertaken for the different crosslinking data sources that might be given. Often, the modeler might be willing to test the modeling results of different sets of assumptions, test different structures, crosslinking data sets or density maps. In these cases it is convenient to have the option to "activate" or "deactivate" different resources for a given modeling iteration. This is also done in this layer. Additionally resources can be annotated, e.g. to highlight different domains or subcomplexes, which might be of use in the subsequent interpretation of the resulting models. A last aspect to review that influences the next layer significantly is the multiplicity of the given subunits. The multiplicity of every subunit, i.e. the number of instances of every subunit or subcomplex in the final model, encodes the stoichiometric information that is obtained in experiments. This plays a constitutive role for complexes that exhibit symmetries.

The final organizational layer of the model system definition is the *geometry specification* layer, which is the jumping-off point towards the actual simulation of different models. It exclusively contains assumptions about the relation of the different constituents of the modeling procedure in three-dimensional space. It has a massive impact on the final models and draws on the information already prepared within the other two layers.

The geometric information results both from general knowledge about the structure of biomolecules and experimental results specific to the modeling project and is encoded in terms of restraints and constraints. This is due to the fact that within the *Assembline* pipeline the final models are generated on basis of a Monte-Carlo ([35],[36]) procedure, which operates using a central score which "good" models minimize and "bad" models don't. Different aspects of geometrical nature might be encoded as restraints, which are allowed to express a degree of uncertainty, and as constraints (as in the case of symmetries), which have a bearing on how the subunits are moved during the simulation, but not the score that measures the "quality" of the model. This distinction is ultimately

of importance in the context of the IMP modeling platform. The fact that the same molecules cannot occupy the same space is encoded within the excluded volume restraint, which can be more lenient or strict depending on the modelers choice. The symmetry condition is an example for the constraints. It is specified by noting the symmetry axis, the kind of symmetry or the transformation matrix and the structures that adhere to the specified symmetry group.
While symmetries can be described as constraints and strong restraints (to model approximate symmetries), another similar concept, that of rigid bodies, is modeled as a constraint. Members of a rigid body, subunits, domains or whole subcomplexes, do not change their position relative to each other. Both crosslinks and covalently linked regions of structures can be encoded as connectivity restraints. Depending on the degree of confidence and the length of the link these restraints can be given different weights. The modeler might for example wish to model the links representing covalent bonds with fairly high weights and the low-confidence crosslinks with lower weights. This is an example of how the modeler can encode degrees of certainty regarding different information sources. Of similar nature are proximity restraints which might arise from binding studies or point mutations at specific interfaces between structures. The density map restraint encodes the likely position of a given structure within one or multiple density maps. It is one of the most significant sources of information and the only one that has an inherent fixed coordinate system, since mere distances and relations among constituent structures are not sufficient to describe a fixed system of reference. It is significantly influenced by the method of experimental acquisition, the degree of detail and the relation to structures of the density map in question. Together with crosslinks it is a signal that can, as mentioned earlier, encode possible different conformational states of the holocomplex. It is commonly weighted fairly high.
A last issue is the coarseness of representation. It is not always advantageous to represent structures in the most fine grained fashion possible, since smaller details might have a negligible influence on the final set of models but be computationally demanding, therefore impeding effective generation of a set of models. The IMP platform offers the possibility to encode different degrees of coarseness for different restraints, the modeler can use a residue-grained representation for the crossliking restraints and a larger bead representation for the density map restraint of a low resolution compared to a single residue.
It is to be stressed that the outlined perspective of the task of model system definition is only a heuristic to illustrate the process. The *Assembline* pipeline enables a configuration of these aspects in as little as two files, which constitute one possible implementation of the model system definition.

Data curation

A small yet decisive aspect of modeling is the curation of the data that is available to the modeler. This entails preparing the data, evaluating its quality and consistency and extracting configuration parameters for the subsequent modeling procedure. Often, the modeler might want to return to this step to reevaluate the role and importance of a particular data resource.

Arguably the most significant contribution of structural information originates in the electron density maps. In the case of cryo-EM and cryo-ET, the maps that the modeler faces are three-dimensional reconstructions of two dimensional images, which are already result of an extensive processing pipeline (see 2.4.2). This allows for a great deal of variability, in some cases the modeler is given multiple different reconstructions of supposedly the same protein complexes. Therefore, a critical assessment of this data source is essential.

A property of great interest to the modeler is the *local resolution* parameter. Opposed to the FCR (2.4.4), which is a measure of the global consistency of the dataset, the *local resolution* defined in [22] reflects the spatially variable feature resolution, i.e. different areas of the map might have different degrees of "detail resolution". These two resolution parameters give valuable hints for the resolution parameter that is used later on to generate densities from the atomic structures (see 3.2.3) that will be the input for the systematic fitting procedure. It is advisable to try a range of parameter instances and evaluate the results. There might be cases where "details" are obviously noise, such in the case of density maps obtained from a negative stain experiment, when the density map is the signal of a surface and it becomes possible to clearly distinguish noise (not forming a surface) from signal (forming a surface). In such cases a low-pass-filter or a Gaussian blur filter can be of help to remove the noise.

A second parameter is the density threshold that separates the "noise" of a density map from what is considered a signal of the structure of the protein complex. This might be difficult, since it might be difficult to distinguish noise from real features at the surface of the map. A guiding heuristic can be the experimentally estimated average volume of globular proteins according to [37], which states that they occupy $d = 1.21 \frac{\mathring{A}^3}{\mathrm{kDa}}$. This would suggest to choose the threshold so that the resulting density occupies the volume $V = d \cdot$ [protein mass in kDa].

If the model project involves density maps of subcomplexes and the holocomplex at the same time, they may need to be aligned, so that the subcomplex is oriented and positioned with reference to the coordinate system implicit in the density map of the holocomplex. This might either be done by hand or by automated fitting procedures. Similarly the given density map of the holocomplex might contain unwanted densities that would obscure the systematic fitting procedure (such as membranes that are not to be modeled). Here it might be advisable to crop the insignificant parts of the density map, either by hand or computationally. This will also have a positive side effect on computational performance, which universally scales with the pixel volume of the input densities.

Another way to improve the computational performance is to *downsample* the input, which is convenient when a density map low on details has an unnecessarily high pixel dimension. If this is true or not is within the subjective judgement of the modeler, a hint however can be obtained by comparing the *local resolution* with the pixel size ($= \frac{\text{coordinate volume}}{\text{pixel volume}}$). If the latter is much smaller than the former, a downsampling might be advisable.

The sequence resources, if there are multiple different FASTA files, might need to be aligned. If they are not, crosslink start and stop residues might refer to the wrong register, therefore giving an erroneous representation of the data.

Consequently the sequence registers of the structure resources needs to be checked. Additionally the modeler might want to "break up" a given structure to account for flexibility. Vice versa several independent structures might need to be concatenated into one larger structure. Furthermore it is helpful to check the structure resources for molecules not needed for modeling, such as water molecules or unexpected ligands. If a region of a protein structure is annotated as disordered, the modeler might want to consider to remove this region, since it literally holds no information about its structure and might unnecessarily bias the modeling procedure.

Lastly, the crosslinks warrant an inspection. Yet again, it is essential to ascertain the sequence register. Crosslinks come in three flavors, monolinks, interlinks and intralinks (see 2.4.5). While every one of those types can carry valuable information, there is an obvious advantage of, for example, having a lot of crosslinks between structures representing ordered regions as opposed to having only monolinks in an disordered regions. It can help to interpret the behavior of the Monte-Carlo model sampling procedure later on to have a good idea of the crosslink dataset. Additionally the modeler might want to consider the coarseness of the system representation at this point, since two crosslinks 5 residues apart might be obscured but a 10 residue bead size representation. Additionally crosslink data sets are associated with a measure of significance for each crosslink, this again might help in interpreting final models. A violated significant crosslink can be viewed as more severe than a violated insignificant crosslink. If the structure resources define a subunit which contains various intralinks, this can be used to assess to coherence of both data resources before doing any modeling at all. If the crosslinks are mapped to a rigid body and are violated, this is a hint that either the structure is not representing the conformation that it takes in the complex that is modeled, or that the crosslink dataset is faulty. A useful tool to visualize the overall population of crosslinks, xiNet, is found in [38].

Data collection

Typically, the main source of data resources in an integrative modeling project will be either the modeler or one or multiple collaborators. Nevertheless there are more ways to obtain data resources, in case the existing ones are not sufficient.
Commonly the primary concern are the structure resources. These can be obtained from online databases, most notably the PDB ([3]). Whole subunits or domains might be obtained this way. Another option is to try a structure prediction approach using dedicated servers, such as SWISS-MODEL ([39]), Rosetta ([32]), PSIPRED server ([40], for prediction of secondary structure) or the recently established AlphaFold database ([41]).
The same can be attempted for the density map resources. The EMDB (Electron Microscopy Data Bank, [**EMDB**]) is a prominent dedicated database.
Crosslinking is a relatively recent innovation, dedicated databases are only now being established. Two examples are the database provided on the website of [42] and the PRIDE database [43]. Datasets might also be attached to single publications.

2.5.3 Model sampling

To understand the process of model sampling, i.e. the generation of a set of models conforming to the restraints and constraints defined in the manner outlined above (2.5.2), it is convenient to first consider the objective function, within *Assembline* commonly called *scoring function*. The scoring function is meant to encode the "goodness" of a model, its output value is therefore a scalar. The lower this number is, the better the model is said to be. The task is therefore to find an ensemble of low scoring models. The input of the scoring function are the numerical representations of the position and orientation of all structures, be it atomic models of domains or coarse-grained bead-like representations thereof, in the form of transformations. These transformations tell the algorithm where to place and how to orient each single structure. The scoring function $\mathcal{S}$ therefore has the general form

$$\mathcal{S} : \{\mathbf{T}_0, \mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_{N-1}\} \longrightarrow s \in \mathbb{R} \tag{2.1}$$

With the $\mathbf{T}_i, i \in \{0 \ldots N-1\}$ denoting the transformations, which can be encoded numerically in different manners, and N denoting the number of structures within the simulation. What follows immediately from this first specification is the massive size of the search space. Since every $\mathbf{T}_i$ needs at least 6 independent parameters (3 for the position, 3 for the orientation), it is a 6-dimensional object. For every further independent $\mathbf{T}_j$ holds the same, resulting in a $6N$ dimensional space in total. This size of the search space of possible configurations warrants the use of the *Monte-Carlo-Simulation* method. Also, the difference between restraints and constraints (as used within the *Assembline* framework) emerges here. Restraints can be understood as limiting the relative position and orientation of the structures with some degree of uncertainty. For example, while every crosslink restraint has its best possible state (corresponding to its lowest score) in the average length of the linker, deviations are allowed and can be modeled using different distributions. The more severe the deviation, the higher the score associated to the restraint and the larger and more unfavorable its impact on the total score will be. This contrasts with constraints. They define relative positions and orientations of structures with absolute certainty and therefore do not need to turn up as a factor within the scoring function, since they only would add a term which would be constant for all possible configurations. In fact, constraints effectively reduce the dimension of the search space - given a rigid body constraint, which fixes the relative position and orientation of two or more structures towards each other, one only needs 6 degrees of freedom to describe the entire rigid body. As suggested above the restraints are added as terms in a sum to construct the scoring function. It takes the form:

$$\mathcal{S} = \sum_{1}^{R} w_i r_i \left(\mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3, \ldots, \mathbf{T}_N\right) \tag{2.2}$$

with $w_i \in \mathbf{R}$, the r_i being functions that map the $\mathbf{T}_i$ to $\mathbb{R}^+$ and R the total number of restraints. Every r_i would encode one given restraint and might potentially depend on all $\mathbf{T}_i$. The w_i are weights that can be used by the modeler to change the relative contribution that each restraint makes to the total score, a lower weight corresponds to a lower influence over the total score. So although both constraints and restraints encode geometrical information, their role in the model sampling procedure in fundamentally different.

Although the restraints might depend on the position and orientation of all possible structures (as an energy term might), in practice it often depends on less. The restraints implemented in *Assembline* that are used most commonly in modeling projects are the following. In this description I will take the transformations $\mathbf{T}_i$ as a stand in for "position and orientation of a structure within the simulation" for brevities sake.

- The *excluded volume restraint* is an example of a restraint which requires all $\mathbf{T}_i$ to be included in the calculation. It encodes the fact that no two bodies can occupy the same position in space, or, equivalently, that the mass density of biomolecules is roughly constant[CITATION]. It is implemented as a number of checks of the distance between the representations (e.g. a structure as a rigid body consisting of spheres of some radius) of the structures. This restraint can be hard or soft, the latter option allowing for a specified amount of overlap of the volumes. It will score high if an overlap occurs, otherwise it will not contribute to the total score, therefore sanctioning model configurations with extensive overlaps. Due to the computationally intense cost of calculation of the restraint it can pay off to define it for a more coarse-grained representation of the system, to diminish the number of necessary checks.

- The *discrete restraints* demands only one $\mathbf{T}_i$, since it encodes the likelihood of a specific structure to be positioned at a given point in a given orientation. The possible values of this restraint are precalculated (see 2.5.5). In fact, this precalculation has a double role within the *Assembline* pipeline, as the possible positions $\mathbf{T}_i$ for a given structure often results from systematic fitting procedure. Therefore, these restraints are only applicable to structures that have undergone the said procedure. The precise value of this restraint depends on a number of parameters such as the resolution parameters, chosen density thresholds or the type of score used during the systematic fitting. This restraint encodes a significant amount of information, so that it might pay off to vary the listed parameters.

- *Connectivity restraints* can model crosslinks, covalent bonds or proximate surface restraints, since they all build on the notion of the distance between two points. Depending on the bond it describes it can be modeled as a distribution whose minimum is at the average bond length and whose steepness can reflect the strength of the bond. For example the modeler might want to enforce a strict restraint on the covalent bonds of the peptide backbone of a subunit and a comparatively looser bond on a low confidence crosslink. It can be extended to model more complex notions of closeness, e.g. to keep close interfaces between subunits or domains that have been shown to interact in biochemical experiments. Another use case might be a symmetry restraint. Not modeled as a constrained this restraint would allow for deviations from perfect symmetry. For further refinement, the cross-correlation value can be used as a restraint.

Having outlined the construction of and influences on the scoring function the task that remains is to find the global minimum and the local minima over the domain of all possible positions and orientations of the structures of the modeled complex. As mentioned, this domain is quite voluminous and the "compass" to navigate it, the scoring function, also contains a lot of parameters and might be expensive to calculate. The method of choice in this kind of situation is often a Monte-Carlo method.
The number of variations of the Monte-Carlo method is vast and will not be elaborated here, a possible introduction is [35]. A common feature however is the use of random samples from the search domain. In our case the search domain is the maximally $6N$ dimensional real vector space, a representation of the possible positions and orientations of

N structures. Another common feature of Monte-Carlo methods is that this exploration of the search domain is informed by random variables but not entirely guided by them. In terms of the scoring function this means that for a given random position and orientation we need a notion of other random positions and orientations that are close, meaning their respective scores do not differ too much from another. This gives us the possibility of a gradual descent to configurations with lower score values and therefore better models. Hence we are faced with two requirements: A method to randomly sample the search domain and a way to transport our system from one point in the search domain to the other. *Assembline* offers two options for the first task.
The most commonly used option is the use of systematic fitting libraries, which are calculated beforehand. A random sampling of $\mathbf{T}_i$ is created together with a score, which might be any possible score that the modeler sees fit to be used in a given modeling scenario. This set of pairs of transformations $\mathbf{T}_i$ and associated scores s_i forms a representation of the search domain. *Assembline* then samples from this basic set using the scores of s_i or a function thereof as weights to sample better scoring transformations more often than the others. The actual movements are facilitated facilitated by *Mover* objects, which are able to propose a new configuration, which is then either accepted or rejected by a criterion that is based on the scoring function. If the criterion is met, the accepted configuration will be the jumping-off point for the next iteration. This process is illustrated in (figure 2.10). Underlying these manipulations of positions and orientations are also the previously defined constraints, which have the effect that e.g. a symmetric configuration of some kind moves in a manner that preserves the symmetry. The other option, which can be used in the case of absence of density maps, are random transformation drawn from a uniform distribution.
The criterion mentioned above is the so called Metropolis criterion [36]. It is calculated using the score difference between two successive steps $\Delta s = s_{i+1} - s_i$ in the Monte-Carlo sampling procedure. First, the number

$$m = e^{\frac{-\Delta s}{T}} \tag{2.3}$$

is calculated, with T being a parameter called temperature. This is compared to a random number u drawn from a uniform distribution in $[0, 1]$:

$$m > u \Rightarrow \text{move is accepted}$$
$$m \leq u \Rightarrow \text{move is rejected}$$

If there is a decrease of the scoring function, Δs will be negative and consequently $m >= 1$. This will always lead to an accepted move. If however there is an increase in s, the move will be rejected with some likelihood, which is larger for a big increase. There is always a

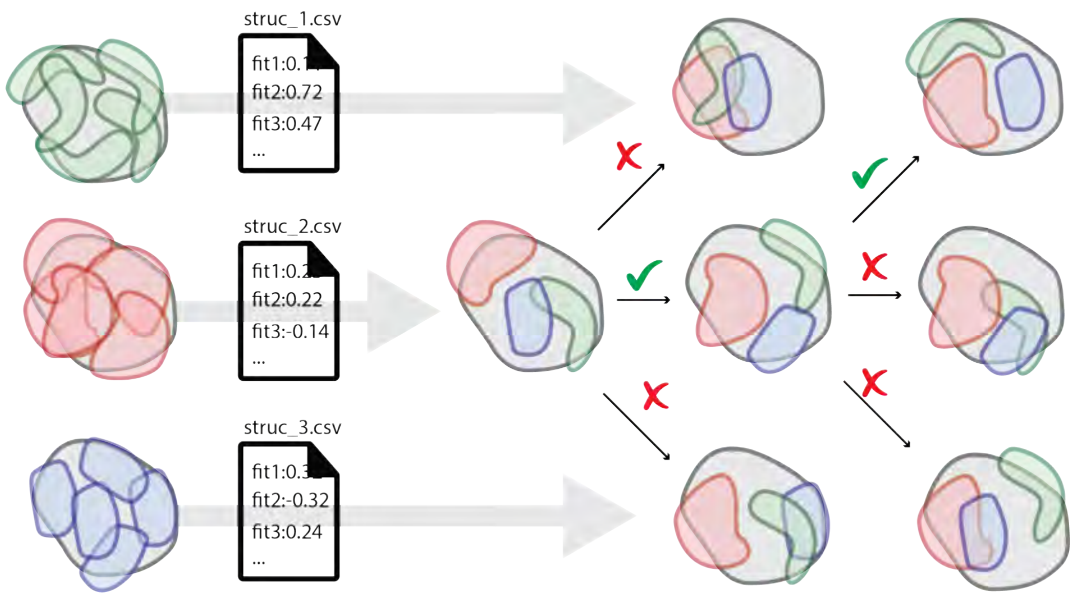


Figure 2.10: An illustration of the sampling of configurations using a library of systematic fits.

non-zero probability that an unfavorable move will be accepted. This enables the Monte-Carlo run to escape local minima. The temperature factor T relaxes the strictness of the rejections with a growing positive value. This results in a movement with greater leaps, comparable to the Brownian motion of particles of higher temperature. *Assembline* employs the technique of *simulated annealing*, which changes the temperature parameter T after a number of steps according to a predefined schedule. The hoped for effect of this procedure is a "cooling down" of the simulation towards the end so that the configuration can settle in a local minimum of the scoring function. In the beginning the temperature term is higher to enable greater jumps and a more exploratory movement in the global landscape of the scoring functions domain.
In some cases the restriction of the $\mathbf{T}_i$ to the results of the systematic fitting procedure might obscure and prevent further descent towards good-scoring models due to the coarse grained representation of the configuration space. For this eventually *Assembline* offers a *refinement* mode, which relaxes this restriction and enables a local, free exploration of configuration space that might further improve the models. It is to be noted that this refinement protocol cannot be seen as a replacement of the protocol outlined above, since it would converge much slower and prevent to effective exploration of the global configuration space.
The inherent probabilistic nature of the Monte-Carlo sampling makes a number of runs necessary. For every run the last configuration in the sampling procedure can be considered the result of this run. Alternatively, every configuration in a run can be considered a resulting model since it is not impossible for the sampling procedure to produce a good (low scoring) model early in the run.

2.5.4 Analysis of models

Structural integrative modeling aims to produce a small number of feasible models, according to the terms set out in the model system definition (see 2.5.2). Given enough non-contradictory information about the protein complex in question, this is an achievable goal. There is always a remainder of uncertainty due to the inherent error in data and the systematic error, which is why the result of the procedure outline above is always a number of differing model configurations (possibly corresponding to different biological conformations of the protein complex). This uncertainty can be assessed by measuring the variation in particle positions between different model configurations. The standard measure to achieve this is the RMSD (Root Mean Square Deviation)-measure (see 3.8). The ideal scenario would therefore be a number of models which are very similar to each other, their RMSD-variation being small compared to the size of the model. One might however end up with a number of equally feasible models that differ to a larger extent. This might be for a number of reasons: conformational changes, insufficient information from the data resources or an incorrectly constructed scoring function. In any case one needs to consider a number of questions once the model sample population is created.

1. *Restraint satisfaction* is equivalent to model quality, since violated restraints will be equivalent to contradiction to the data the restraints encode. Different restraints might require different degrees of strictness. While it might be permissible that only 90% of the crosslink restraints are satisfied (especially if the pertaining crosslinks come with a low degree of significance), a violation of the excluded volume restraints is inadmissible because it contradicts basic biochemical knowledge. In *Assembline* the scores of single restraints are accessible to the modeler. This information can be used to propose a change of weight within the scoring function and to rerun the Monte-Carlo simulation to reevaluate the model with stronger or weaker emphasis on one or more restraints. Especially in multistate modeling it is advisable to look for contradictory sets of restraints since these might signal to different conformational states of the protein complex in question. If all restraints are sufficiently satisfied and the model sample population does not consist of a few well defined clusters (see below) this might be a hint that more information is needed to generate more definite models. It might also be enlightening to rerun the Monte-Carlo sampling with *less* restraints to evaluate the influence of a specific restraint/restraint weight combination on the models. The restraint satisfaction is the most important aspect to evaluate, it constitutes both the primary quality measure and the primary set of parameters that the modeler can influence on a rather intuitive level.

2. *Convergence* of the scoring function over (simulation-) time and ideally the single

restraint functions is strictly not required for a good model but a sign that the scoring function has settled on a global or local minimum. Since these minima are interpreted as feasible models the modeler has an active interest in finding the minima and therefore in the question of convergence. Convergence can be checked directly by visual inspection of the score graph (see 2.11 for a visualization). The modeler has to decide for a trade-off between two alternatives. If the Monte-Carlo run would be allowed to have infinite length, convergence would be apparent (since convergence is defined with respect to infinite series). Due to the necessarily limited amount of steps there is the danger of stopping the Monte-Carlo run too early and miss out on minima. One cannot rule out this possibility categorically, therefore convergence estimates are of great convenience. *Assembline* offers the mentioned score graphs as an output for visual inspection. This method will be sufficient in many cases. Its downside is that it requires the subjective judgement of the modeler. While the question of convergence itself is reliably answered by a human being (as long as the modeler is able to recognize a straight line), it might be inconvenient if the number of parallel Monte-Carlo runs is large. Additionally it can pay off to check the convergence *during* the simulation to prevent unnecessarily long runs. For this reason *Assembline* offers an implementation of the Geweke-Convergence criterion for finite time series [44]. In brief, this criterion evaluates the statistical properties of mean and variance in different ranges of steps and assumes that they should be similar if convergence is achieved. This is because the “thermal fluctuation” of the score function then only depends on the noise created by the stochastic movement of the structures and not a change in the scoring function due to contributions by restraints.

Exhaustiveness addresses the question how well the volume of the space of all possible model configurations is sampled. This aspect arises due to the stochastic nature of the modeling procedure. It might be that even in a large number of modeling runs some “corners” of the the configuration landscape are not covered, creating a bias within the modeling procedure. For this reason *Assembline* offers an exhaustiveness test based on [45]. This tiered approach uses the score population and the RMSD (cf 3.8) as numerical representation of the results of the modeling procedure. It works by selecting a population of well-scoring models, splitting them into two populations and subjecting to a series of increasingly stringent tests of similarity. The underlying assumption is that a sufficiently well sampled population should minimize these differences. The last and most thorough tier of the test, overlaying the densities of the single instances of model samples and comparing the resulting density between the split populations of models, can have further significance to the modeler. If a modeling procedure does not yield sharply defined differing instances of models, this overlay of densities can be interpreted as a probability density that

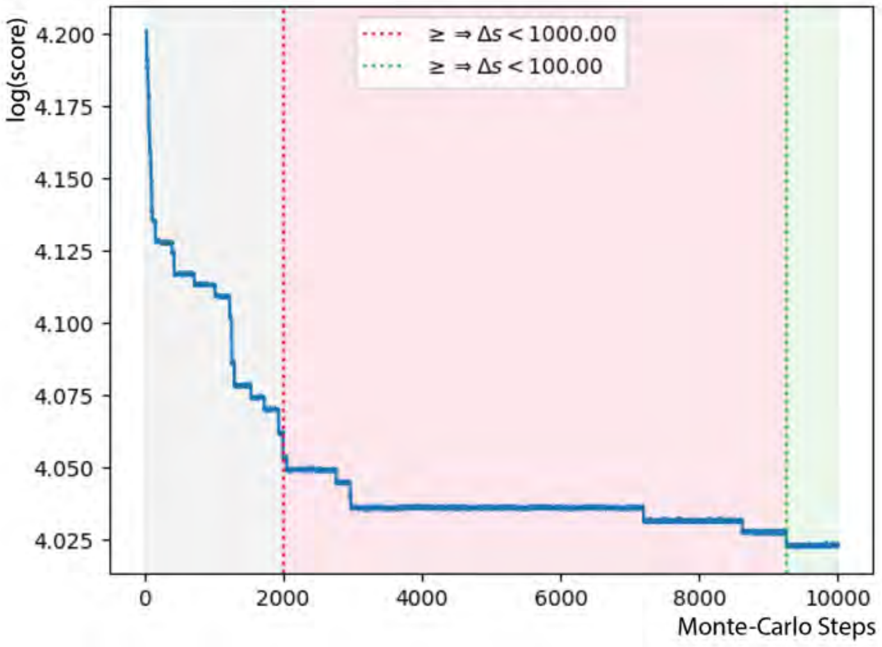


Figure 2.11: An example of a convergence plot. Differently colored areas highlight stretches where the oscillations between scores Δs according to the Cauchy-convergence criterion (see 4.2) are bounded by the amount indicated in the legend.

reflects the probable positions of the structures. In some cases, this might still be a sufficient result to reflect a gain in knowledge about the structure of the protein complex.

3. Lastly, the *crossvalidation* with unused data and existing hypothesis serves both as an embedding within the broader biological context and a consistency check for the modeler. There might be cases that data cannot be encoded as computational restraints. This data can be a valuable resource for cross-validation or give hints to what is missing within the system setup in case of failure to reach conclusive models according the criteria laid out above. A good model might also allow for interpretations regarding the evolutionary, functional or structural properties of a protein complex. This step serves as a tie-in of the modeling procedure into the broader landscape of experimental and hypothetical exploration of the system in question.

2.5.5 Systematic Fitting

Systematic fitting is arguably the most significant source of structural information used in integrative structural modeling with *Assembline*. As outlined above, the fitting of a struc-

ture or a map into the density map of the holocomplex provides both a list of positions and orientations and a score that is associated with each entry of that list. Every subunit or subcomplex is to be assigned a position and an orientation in the final model or models that relates to its position and ortientation within the density map of the holocomplex. Additionally, during the Monte-Carlo procedure the possible placements of the structures are drawn from this list. This constitutes the importance of this step within the modeling procedure as a whole.
Since the density map often has a resolution too low for an unambiguous assignment of aminoacid residues, one necessarily has to cover a representative part of all possible possible orientations and positions of each subunit within the density map of the holocomplex. I will refer to this as *spatial sampling.* The second aspect of systematic fitting is the question: What constitutes a good fit? How can I know that a specific position and orientation is a good place for a particular subunit? This is answered by calculating scores that relate different possibilities to place a subunit within the density map of the holocomplex. I will refer to this as *scoring.* I will now introduce both aspects of systematic fitting. In course of this discussion I will refer to the density map of the holocomplex as *target* and to the atomic structure or density map to be fitted as the *query*, for brevity. In many cases the the query needs to be converted to a *simulated density map.* This needs to happen if the query is given as a structure and the chosen scoring method exclusively accepts density maps as input, such as the overlap score (2.4), the cc-score (2.5) and the cam-score (2.7). There are multiple methods to do this, one frequently used method is Gaussian blurring of point masses.

Spatial sampling

There is an infinite number of possible positions and orientations (for brevity: transformation) of the query within the target. The scoring method selected by the modeler will assign a value to each possible transformation. In this way the underlying "score landscape", or score space can be "explored". Much the same way that a density map defines a geometrical object (representing a biomolecule) by assigning a density value to each three-dimensional pixel, the score space can also be understood as a 6 dimensional geometric object. By *systematic fitting* I refer to any representation of the score space that represents it *as a whole.* In *Assembline*, this representation is one of the basic resources for model sampling, see 2.5.3. There are two main approaches, random sampling and sampling by regular grid.
Random sampling is the method employed in the *FitMap* tool, implemented in *UCSF Chimera* ([28]), which is used in the systematic fitting step within *Assembline.* The tool offers a range of options, *Assembline* exposes a subset of these options. A common proce-

dure would be the following (parameters and aspects not essential to the issue of spatial sampling are omitted):

1. Specify a set of parameters: The total search volume, the number of random transformations `n` to be scored, the coarseness of angle tolerance and shift tolerance. The latter options dictate which transformations are to be seen as "the same". If two transformations are close with regard to the set tolerances, they will be registered as identical.

2. Generate a random transformation by generating a random translation and a random rotation. A method `random_direction()` is used in both tasks to supply a normalized vector of a random direction. This method is called once for the random translation, the result is then scaled to the dimensions of the `bounding box` of the search volume. The method is called twice to generate two normal vectors of a random rotation. This specifies a orientation uniquely up to a sign, which is determined by the canonical use of the vector product.

3. Apply the transformation to all pixels of the simulated map of the structure, score according to the selected scoring method. If the option is enabled, a gradient descent can attempt a subsequent optimization of the score.

4. Bin the result according the the defined angle and shift tolerance.

This can be called a systematic representation in so far as that by the law of large numbers with a sufficient size of `n` the sampling *should* approximate the uniform distribution, therefore covering the search space. The limiting factors are the angular and translational bin size, effectively discretizing the 6-dimensional search space, and the optimization step, favoring the better scoring solutions (which might be welcome). As hinted at in 2.5.3, *Assembline* offers a number of statistical post processing options, replacing the actual score for a given translation with a measure of its statistical significance. Most notable is the option of calculating p-values by assuming the zero-hypothesis that the scores are normally distributed. The scores are treated as a statistical population that way. This differs from the perspective promoted in this thesis, where the "scoring space" is treated as a 6-dimensional geometric object.
Regular grids are the alternative to achieve systematic fitting as defined above. They also allow for a specification of the degree of coarseness, yet are not generated by a random sampling. Since they are integral to the conceptual results of this thesis, they will be discussed in detail in the results section (see 5.5.1) and their application to measure global metrics (*numerical quadrature*) is described in 4.2.4.

Scoring in general

Once a placement is sampled, a score can be calculated. Typically the query needs to be "brought" to that placement, this is accomplished by transforming the data structures underlying the query. On account of being more volumous, the target can rest, its transformation would be more costly than the alternatives. Let the particular placement be specified by a translation and rotation transformation T and R, then any function s that takes the transformed query and target data as input and outputs a number r is called a score:

$$s\colon\ (\text{target}, \text{query}, \mathrm{T}, \mathrm{R}) \longrightarrow r$$

r itself can be a real number or a whole number, the only condition that must be met is that different score values can be put into an order, so that the scores can be compared to rank "worst" to "best" fit. This definition is intentionally broad. There is more than one way to calculate a score in the described situation and a number of scores have been proposed and studied [46]. In the context of structural integrative modeling the question arises naturally: What is the "best" score? What could tell a good score from a bad score? This work will outline an attempt to advance on these questions. It is instructive to study the underlying ideas of the most used scores.
Most scores are functions that are constructed with help of a basic geometric intuition. Understanding these intuitions can help to decide which type of score is most appropriate to use in a particular case and how costly a score might be computationally. For ease of view I will illustrate the prinicple of a score in two dimensions, although in real cases the scores operate on a three-dimensional space. For details regarding the mathematics employed see 4.

The overlap score

Figure (2.12) shows a two dimensional grid with a query and a target density (for an introduction of densities see 3.2.2). They overlap to some extent. The geometric idea is that if this overlap is high the fit, specified by a position and orientation of the query density relative to the target density, is a better fit. Figure (2.12) is simplified since it shows the densities as binary, where query and target are either "present" in a pixel (value 1.0) or "absent" (value 0.0). In a real case, the values would be more diverse and the densities might not be necessarily topologically connected ("in one lump"). For this illustration the simplification is however suitable. Here the overlap consists of pixels where both target and query density have value 1.0. This is exploited in the calculation by assigning target and query a list of values $\mathbf{t}$ and $\mathbf{q}$, one for each pixel. Subsequently

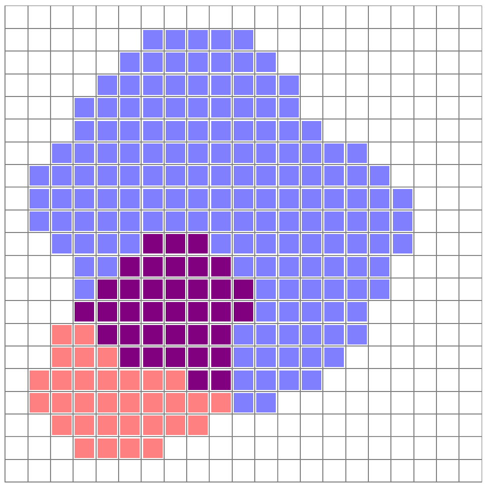


Figure 2.12: A simplified rendition of a two dimensional density, the target is rendered in blue, the query in red and their overlap in purple.

the product of the values of **t** and **q** per pixel is calculated and summed up,

$$\mathrm{ov}\left(\mathbf{t},\mathbf{q}\right) = \sum_{i=0}^{P-1} t_i \cdot q_i = \langle \mathbf{t} | \mathbf{q} \rangle \tag{2.4}$$

with P denoting the number of pixels. This sum will be zero if there is no overlap and it will be maximal if the overlap equals the pixel volume of the query. Mathematically, the densities have been modeled as vectors of dimension P and the sum (2.4) is their scalar product (see 4.2.1). The values 0.0 and 1.0 have been chosen for illustrative purposes. If the values of the vectors where not simply 0.0 or 1.0, as it is the case in a real density maps, the argument still would still hold. If for a given pixel both target and query density have high values, the product will be a more significant part of the sum as if one or both of them had two values.

The cross correlation score

The cross correlation score is similar to the overlap score. Its functional principle is similar, which is reflected in the rule for its calculation:

$$\mathrm{cc}\left(\mathbf{t},\mathbf{q}\right) = \frac{\langle \mathbf{t} | \mathbf{q} | \rangle}{\|\mathbf{t}\| \cdot \|\mathbf{q}\|} \tag{2.5}$$

$$= \frac{\mathrm{ov}\left(\mathbf{t},\mathbf{q}\right)}{\|\mathbf{t}\| \cdot \|\mathbf{q}\|} \tag{2.6}$$

Meaning it is a normalization of the overlap score. The geometrical intuition behind this comes into play if one compares the overlap score of queries of different sizes. Figure (2.13) illustrates this. The right example is clearly the better fit, yet the $\mathrm{ov}\left(\mathbf{t},\mathbf{q}\right)$ score

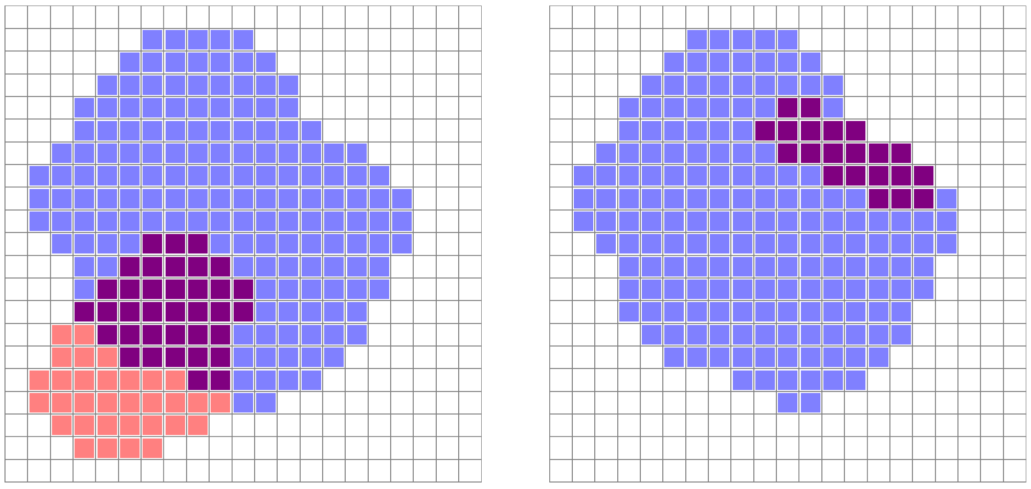

(a) A larger query density partially overlapping. (b) A smaller query density completely overlapping.

Figure 2.13: The cross correlation score in different contexts.

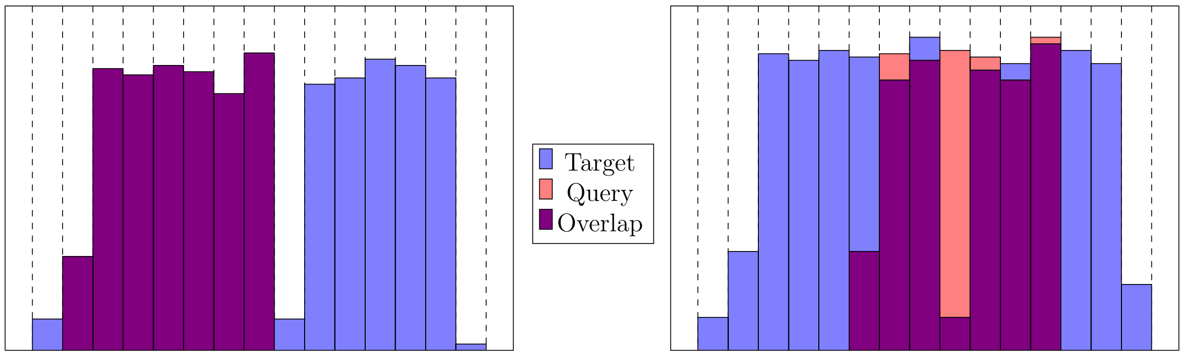
would be smaller due to the overall smaller volume of the query density. If one would compare these two scores one might be mislead to think that the left case constitutes the better fit. The cross correlation score then enables the comparison of queries of different sizes. If one considers a perfect fit, meaning the two densities have the exact same shape and the placement is such that their shapes are absolutely congruent,

$$\begin{aligned} \mathrm{cc}\left(\mathbf{t},\mathbf{t}\right) &= \frac{\langle \mathbf{t}|\mathbf{t}|\rangle}{\|\mathbf{t}\| \cdot \|\mathbf{q}\|} \\ &= \frac{\|\mathbf{t}\|^2}{\|\mathbf{t}\| \cdot \|\mathbf{t}\|} = 1 \end{aligned}$$

If one allows only positive densities (and densities signify in general the average location of particles (electrons), so they will be positive) the cross correlation score will be in the interval $[0, 1]$. A further development in this line of reasoning is the correlation-about-mean score.

The correlation-about-mean score

So far the scores presented are based on the assumption that on a global scale the geometric overlap of query and target density should be maximal, meaning areas of high density should coincide with other areas of high density. This is useful for very coarse maps without much detail. In maps with high detail resolution this criterion might be useless, because the overall overlap might be close to constant and therefore unhelpful to distinguish good fits from bad fits. The details are often lost in this process, because



(a) A complete overlap.

(b) A partial overlap.

Figure 2.14: Overlap of a query density at two different placements.

only the geometric overlap is important and not the "shape" of the densities. This is a direct consequence of the sums in (2.4) and (2.5). The correlation-about-mean score tries to rectify this shortcoming. To illustrate how it works I will discuss an example of a one dimensional density. This is required by the fact that the underlying argument refers to the functional form of the density values in a more pronounced sense than the hitherto described scores. This however is difficult to illustrate in a two dimensional grid on paper. Figure (2.14) shows a one dimensional density at two different placements. (a) illustrates a complete and (b) a partial overlap. It is intuitively clear that (a) is a better fit, due to the match in "shape" of the two densities. A way to codify this intuition mathematically is provided by the correlation-about-mean score:

$$\operatorname{cam}(\mathbf{t},\mathbf{q}) = \frac{\langle \mathbf{t}-\overline{\mathbf{t}}\cdot\mathbf{1}|\mathbf{q}-\overline{\mathbf{q}}\cdot\mathbf{1}|\rangle}{\|\mathbf{t}-\overline{\mathbf{t}}\cdot\mathbf{1}\|\cdot\|\mathbf{q}-\overline{\mathbf{q}}\cdot\mathbf{1}\|} \tag{2.7}$$

where $\overline{\mathbf{t}}$ and $\overline{\mathbf{q}}$ are the respective averages of the vector components and $\mathbf{1}$ denotes a vector with only "1"s as entries: $\begin{pmatrix}1 & 1 & \dots & 1\end{pmatrix}^T$, its length equals the length of $\overline{\mathbf{t}}$ and $\overline{\mathbf{q}}$. If we define $\mathbf{q}$ to be a linear function of $\mathbf{t}$, so $\mathbf{q} = s\mathbf{t} + c\mathbf{1}$ with s and c arbitrary scalars, we find

$$\begin{aligned}\mathbf{q}-\overline{\mathbf{q}}\cdot\mathbf{1} &= s\mathbf{t}+c\mathbf{1}-\overline{s\mathbf{t}+c\mathbf{1}}\cdot\mathbf{1}\\ &= s\mathbf{t}+c\mathbf{1}-s\overline{\mathbf{t}}\cdot\mathbf{1}-c\overline{\mathbf{1}}\cdot\mathbf{1}\\ &= s\left(\mathbf{t}-\overline{\mathbf{t}}\cdot\mathbf{1}\right)\end{aligned} \tag{2.8}$$

where the fact that the mean is a linear function was used in (2.8). Therefore we have

$$\operatorname{cam}(\mathbf{t},\mathbf{q}) = \frac{s\langle \mathbf{t}-\overline{\mathbf{t}}\cdot\mathbf{1}|\mathbf{t}-\overline{\mathbf{t}}\cdot\mathbf{1}|\rangle}{|s|\|\mathbf{t}-\overline{\mathbf{t}}\cdot\mathbf{1}\|\cdot\|\mathbf{t}-\overline{\mathbf{t}}\cdot\mathbf{1}\|} = \pm 1$$

depending on s being positive or negative. This means if the query is a scaled or uniformly shifted version of the target or vice versa the $\operatorname{cam}(\mathbf{t},\mathbf{q})$ score will be at its maximum, 1 (assuming s positive). Indeed, if we calculate the cross correlation score and the correlation-about-mean score for both cases, we find

$$\begin{aligned}
\mathrm{cc}\left(\mathbf{t}, \mathbf{q}_{complete}\right) &= 0.7414 \\
\mathrm{cc}\left(\mathbf{t}, \mathbf{q}_{partial}\right) &= 0.6524 \\
\mathrm{cam}\left(\mathbf{t}, \mathbf{q}_{complete}\right) &= 0.4492 \\
\mathrm{cam}\left(\mathbf{t}, \mathbf{q}_{partial}\right) &= 0.1646
\end{aligned}$$

and a pronounced difference in the different score functions: $\Delta\mathrm{cc} = 0.089$ and $\Delta\mathrm{cam} = 0.285$. Thus the correlation-about-mean score in better at distinguishing shape, it registers the "dent" in 2.14 (b) to a higher degree than the cross correlation score. This example is only a very simple instance, in an intricate three-dimensional density this effect will be more pronounced. It also shows the importance of the functional form, i.e. the "shape" of densities for certain scores. This is close to human intuition and a desirable quantity. In the same manner another intuitively clear idea is to be discussed, the idea of masking.

Masked scores

Masking is nothing that is connected to any specific score but to the preprocessing of the input densities for each score. It is of particular interest in structural integrative modeling, because per definition the query density, which represents a part of the holocomplex that is to be modeled, will be smaller than the target density, which represents the holocomplex. Masking offers the possibility of ignoring the parts of the target density, which are not coinciding with the current placement of the query density. Consider the placement of a query density in figure (2.15,a). It could be described as one of the best possible fits. The cam-score of this fit however is only 0.4367. If we mask out the all the pixels that are not "within the contour" of the query, the cam-score becomes 1.0 in this simplified example. This is not surprising, since the cam-score has terms in its definition (see 2.7) that refer to densities as global entities, meaning information will be taken into account that is not important locally. In that sense, masking will change any given fitting score. It remains to discuss what exactly constitutes a good mask, it depends on the mask what is considered "global" or "local". Here the situation is similar to the question of what constitutes a good score. Masks can be defined by choosing a parameter, $p_{density}$, and selecting a subset of pixels where the corresponding density value $\mathbf{t}$ is greater than this parameter:

$$\mathbf{q}_{p_{density}} = \{q_i \in \mathbf{q} \mid q_i \leq p_{density}\}$$

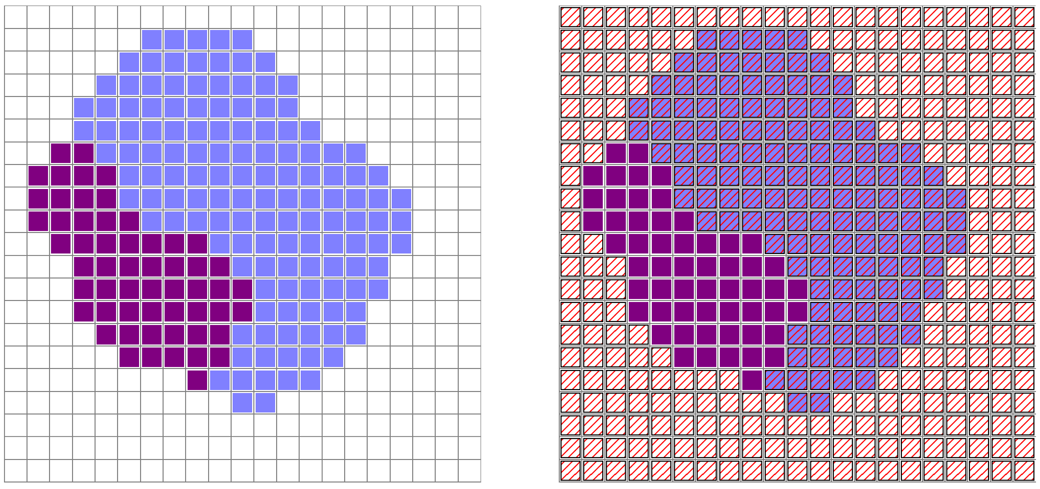

(a) A query density fitting well into the target density. (b) The same scenario with a mask indicated.

Figure 2.15: Illustration of the application of a mask.

There are other ways to pick masks. This is a common way in structural integrative modeling, since it allows to ignore "weaker" densities. These parts of a density of lower value might or might not be part of systematic or random noise and it might pay off to ignore certain parts of a given target density. The example indicated in figure (2.15) works in a similar way, only that the mask is *defined* with respect to the query density and *applied* with respect to the target density.

$$\mathbf{t}_{p_{density}} = \{t_i \in \mathbf{t} \mid q_i \leq p_{density}\}$$

Masks defined like this can have a significant impact on the result of the fitting procedure, and as with scores itself, there is no definitive *a priori* method to tell if and which masks should be applied in a given systematic fitting procedure.

The envelope score

As we can see these scores are derived from simple geometric intuitions. In the case of the experimental method of negative stain EM the density that is measured is a signal of the surface of the protein complex, since the staining reagent only coheres at the surface and it is the source of the signal. We can adapt our geometric intuition and come up with scores that take this specific experimental situation into account. The envelope score and the chamfer distance (see 2.5.5) are such scores and will be discussed in the following. On a more general level it has to be said that while geometric intuition is a valuable tool it cannot account for all peculiarities of a given experimental method. For every method and case it is not always obvious which score might be the best choice.

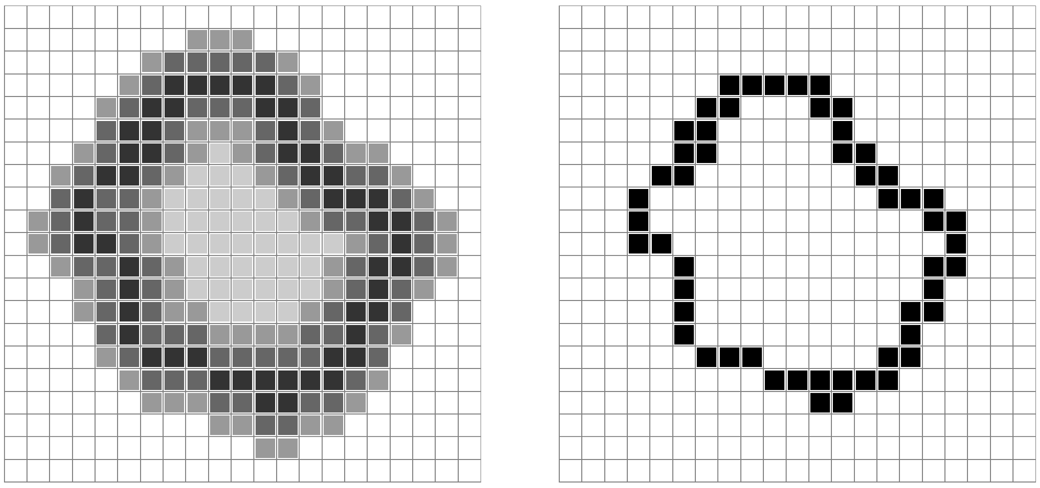

(a) A full map denser at the edges. (b) A possible envelope of the same map.

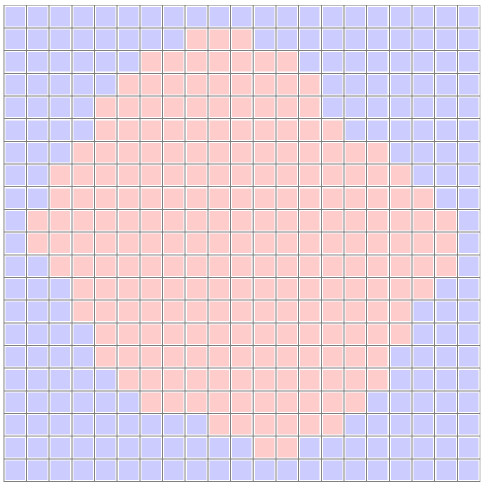
Figure 2.16: Illustration of the generation of an envelope.

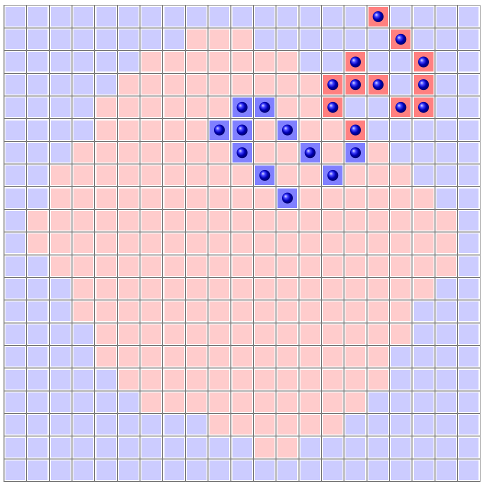
The envelope score needs, as the name suggests, an envelope to work. An envelope with respect to a density is a subset of pixels that is defined by two thresholds, t_{low} and t_{high}. An envelope $\mathbf{e}$ is then

$$\mathbf{e}_{t_{low},t_{high}} = \{t_i \in \mathbf{t} \mid t_{low} \leq t_i \leq t_{high}\}$$

Figure 2.16 shows the process for a two dimensional instance of a density. While there is freedom in the choice of the threshold t_{low}, it is unfeasible to choose it in a way that the resulting envelope envelops too small a volume. This would happen if there are only a few pixels left that fulfill the above condition and no envelope forms at all or the resulting envelope is patchy and has holes. As a rough approximation we can assume that the average density of matter in protein complexes is constant and furthermore assume that the volume that is demarcated by the chosen envelope should roughly correspond to the volume we would expect from the mass of the protein complex. This can be approximated by use of the empirical result that the average density of globular proteins is about $1.21\frac{\text{Å}^3}{\text{kDa}}$ [47]. A possible convention has been established [46] and states that the pixel volume of $\mathbf{e}_{t_{low},t_{high}}$ should be chosen so that it corresponds to of the total volume.
The calculation of the envelope score is straightforward, once the thresholds are defined. A feature that distinguishes it from the scores mentioned so far is that the query structure is not represented as a density but as a list of particles (see 3.2.1). For the scoring itself it is convenient to define a auxiliary density $\mathbf{s}$.

1. Use the pixel grid $\mathbf{e}$ of the original map for $\mathbf{s}$, set the values for each pixel such that the “inner” pixel values are -1 and the “outer” pixel values are 0:


(a) The density $\mathbf{s}$ and its redefined entries.


(b) Particles placed within the map.

$$s_i = \begin{cases} -1 & \text{if } t_i \geq t_{low} \\ 0 & \text{if } t_i < t_{low} \end{cases} \tag{2.9}$$

2. Place the query particles within the density $\mathbf{s}$ and, if the coordinates of a particle coincide with the volume of a pixel, change its value according to the rule

$$s_i = \begin{cases} 2 & \text{if } s_i = -1 \\ -2 & \text{if } s_i = 0 \end{cases} \tag{2.10}$$

   This step rewards placements where particles fall within the envelope and punishes placements where particles are outside of the envelope.

3. Form the sum

$$s = \sum_{i=0}^{P-1} s_i \tag{2.11}$$

   with P being the number of pixels. The result will be an integer. The bigger the integer, the better the placement.

Figure (2.17b) illustrates this scoring scheme.

The chamfer distance score

The chamfer distance constitutes a similar method. Unlike the envelope score however the query must be transformed into a density and subsequently into an envelope in the manner of 2.5.5. For this scoring method we have therefore two envelopes: $\mathbf{e}_t$ and $\mathbf{e}_q$. To calculate the chamfer distance we follow this procedure:

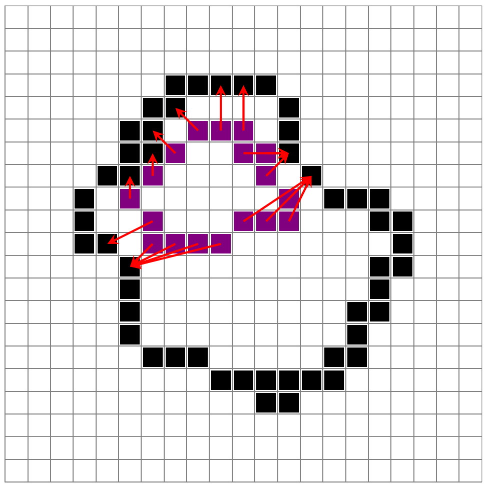


Figure 2.18: Illustration of the chamfer distance.

1. Place $\mathbf{e}_t$ and $\mathbf{e}_q$ in the same density pixel grid.
2. For every pixel of the query envelope, q_i, determine the closest pixel in the target envelope and calculate the distance d_i.

   $$\mathrm{d}_i = \min\{|q_i - t_i| \mid i \in \{0 \ldots P_t - 1\}\}$$

   P_t being the number of target pixels. The pixel position can be calculated according to 3.2.2
3. Lastly, form the sum

   $$s = \sum_{0}^{P_q-1} d_i$$

   with P_q being the number of pixels of the query envelope.

The smaller the score, the better the placement of the query density is considered to be. Figure (2.18) illustrates the described process. The envelope score and the chamfer distance share the property that they work best in cases where the query structure is identical to the target density. Queries that are partial structures relative to the holo-complex, as in subcomplexes, subunits or domains, will suffer from the fact that there is a great deal of freedom in placement. Many possible placements will produce similar scores. The smaller the query, the greater this freedom and the harder varying placements are distinguishable from each other.

2.5.6 Other systematic fitting software options

The arrival of the "resolution revolution" (see [48]) in cryo-EM associated methodology means that there had been an inherent need to associate crystallographic structures with cryo-EM maps, whose resolutions are not sufficient for unambiguous assignment of structures. This resulted in several software packages that facilitate systematic fitting as defined above (see 2.5.5). In this section, some prominent solutions will be briefly introduced.

The situs software library

The well established SITUS library (see [49]) offers two command line programs that enable systematic fitting: *colores* (see [50]) and *collage* (see [51]).
colores is reported to offer Fourier-accelerated fitting (see 3.1.1) via the cross-correlation score (see 2.5). The information implicit in shape-contours is highlighted by application of a Laplace filter (see 3.1.2). The *colores* program can be regarded as using regular grids for spatial sampling. For the translational dimension this is implicit through the use of Fourier acceleration and the rotational dimension is covered by a regular grid of homogeneously spaced Euler angles (see 4.2.3). The range of applicability with regard to the resolution is claimed to extend to 30 $\AA$.
The *collage* program is a conjugate-gradient optimization tool which enables exchange of information between multiple simultaneously fitted query structures. This extends the definition of systematic modeling as above and overlaps with modeling steps that would appear later in the *Assembline* protocol. However, it can be seen as an conceptual extension since the steric clashes between the fitted rigid bodies can also be associated with a $6D$ score distribution.

The PowerFit software

The newer PowerFit software (see [52], [53]) similarly offers a cross-correlation score based, Fourier accelerated and Laplace filtered fitting approach. A possible utilization of GPU devices and multicore CPU is claimed. An additional feature, the core-weighted score, is introduced which aims at weighting score contributions from the protein complex core higher than contributions from the surface. This library can also be classified as implementing a regular grid spatial sampling approach. It introduces some useful features such as resampling and cropping.

The gEMfitter software

The gEMfitter software (see [54]) expands computationally on PowerFit by utilizing the GPU-specific hardware resource of texture memory (see 3.3.3). Also, multi-GPU support is claimed.

Areas of advancement

In the further thesis, several aspects of existing software are attempted to be improved upon. Other features are entirely novel. They will be briefly discussed here:

- Currently, there are tentative uses of GPU features to harness the possibilities of modern parallel computation in published fitting software. There are however still areas left unexplored, such as different scoring methods, the exhaustive use of GPU hardware features and the use of multiple GPUs to solve problems more effectively.
- Some aspects of numerical mathematics are present in modern software, such as the `FFT`-acceleration (3.1.1, [52]) or the use of Wigner-polynomials ([55]) for certain types of scoring methods. However, there are modern numerical methods that are already applied in other fields such planetary science ([56],[57]), geophysics ([58]) or biomedical imaging ([59]), which might benefit the methods of structural integrative modeling, e.g. by enabling the calculation of useful statistics or the practical representation of systematic fitting results.
- One presently discussed problem in integrative structural modeling is the question of exhaustiveness (see 2.5.4, [45]). This question attempts to determine, if a given modeling project has indeed found a representative set of models confirming to the data. This thesis attempts to introduce the methods of information theory into the field in the hope to limit and quantify this representative set.
- Finally, current fitting software is often bound to specific search schemes and output formats. These sometimes limiting features are relaxed to enhance the interface with other software.

2.6 The TFIIIC protein complex

The transcription factor IIIC (TFIIIC) is a protein complex that is involved in recruitment of the RNA polymerase III, which in turn plays a pivotal role in the transcription of genes

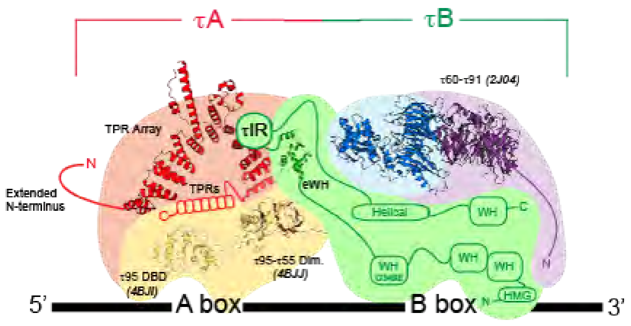


Figure 2.19: Illustration of the tentative structure of the TFIIIC complex.

transcribing tRNA. Figure (2.19) shows a rendering of the structure of the complex. It consists of 2 subcomplexes, τ_A and τ_B with 3 subunits each, $\tau_{131}, \tau_{95}, \tau_{55}$ and $\tau_{138}, \tau_{91}, \tau_{60}$, respectively. It has been shown to bind to intragenic promoters that encode genes of lengths from 31 to 93 nucleotides, which is why the linking region τ_{IR} is thought to be very flexible.

2.7 Objectives

This section outlines the objectives that have formed themselves during the project time associated with the EMBL-PhD program. Partially they were determined from the beginning, partially they have crystallized themselves out of initial projects as it is so often in scientific work. These are condensed perspectives to give the reader some help to frame what follows in the methods sections and the result section. There will be an elaboration of these points in the discussion section.

(A) Initial projects were two integrative structural modeling projects: the PFC protein complex and the TFIIIC protein complex. Both had different initial data resources, which changed over time, and both had different results.

(B) These differences in conditions and results led me to considerations concerning the question if they can be quantified somehow. After getting an impulse from modern crystallographical research I tried to develop an information theoretical perspective on the most essential data sources and processing step: crystal structures systematically fitted to cryo-EM/ET density maps. To enable to calculation of a information theoretical quantity (entropy) I needed to address the 6-dimensional space of orientations and translations that is typical for systematic fitting.

(C) This led me to the numerical methods of quadrature via function approximation and the attempt, to establish these methods in the 6-dimensional translational and rotational sampling space.

(D) In parallel, the massive computational demands became clear and I tried to harness modern computational resources (GPUs). I tried to assess if the data structures and algorithms typical for integrative structural modeling are amenable to parallel computation and attempted an implementation in a language specific to parallel computation (CUDA).

(E) The great variety of the scoring functions used in systematic fitting, together with the already established goal of an information theoretical perspective, led me to an attempt to unify these different scoring methods in one conceptual framework. In this line of questioning I also tried to implement my own type of score, the partial surface score.

Chapter 3

Computational methods

3.1 Algorithmic methods

Integrative structural modeling addresses a variety of data structures and data processing methods by nature. This section tries to elucidate some of the algorithmic techniques in a self-contained and brief manner.

3.1.1 Fourier accelerated fitting

The definition of the cross-correlation, which is, in some form or another, used in a lot of fitting or template matching procedures, is

$$\begin{aligned} \mathrm{cc}\left(\mathbf{u},\mathbf{v}\right) &= \langle \mathbf{u}|\mathbf{v}\rangle && (3.1)\\ &= \sum_{i=0}^{n} u_i \cdot v_i && (3.2) \end{aligned}$$

For the sake of the argument **u** and **v** can be seen as functions $u(x)$ and $v(x)$ (in computer memory functions can represented as value-vectors in any case). Then, the cc score would read

$$\mathrm{cc}\left(u(x), v(x)\right)(0) = \int_{\mathbb{R}} u(x) \cdot v(x)\,\mathrm{d}x \tag{3.3}$$

"$\cdot$" denoting the standard point wise multiplication among functions. One immediately sees the analogy to the sum employed in 3.1. In fitting or template matching procedures one often seeks to test different positions τ, shifting one signal against the other. In this case, the score cc becomes a function of that shift $\mathrm{cc}(\tau)$.

$$\mathrm{cc}(u(x), v(x))(\tau) = \mathrm{cc}(\tau) = \int_{\mathbb{R}} u(x) \cdot v(x - \tau)\,\mathrm{d}x \tag{3.4}$$

The last term in 3.4 is known as a convolution operation and the following theorem is know about it:

$$\int_{\mathbb{R}} u(x) \cdot v(\tau - x)\,\mathrm{d}x = \mathcal{F}^{-1}\{\mathcal{F}\{u(x)\} \cdot \mathcal{F}\{v(x)\}\} \tag{3.5}$$

where $\mathcal{F}\{u(x)\}$ and $\mathcal{F}\{v(x)\}$ denote the Fourier transformations of $u(x)$ and $v(x)$ and $\mathcal{F}^{-1}\{\ldots\}$ the inverse Fourier transformation. This can be used to calculate the convolution 3.4 if $v(x)$ is inverted to $v(-x)$. Then, the signs in 3.5 and 3.4 match. As 3.5 shows, the integral operation has been replaced by Fourier transformations, an inverse Fourier transformation and one point wise multiplication. This in itself would not cause a gain of computational efficiency, but the `FFT` (F̲ast F̲ourier T̲ransform) algorithm ([60]) runs with complexity $O(n \log n)$ instead of $O(n^2)$, which is the case when one uses the naive implementation. This enables a speedup, which is why almost every modern fitting library uses this feature. This process is easily generalizable to higher euclidean spaces such as $\mathbb{R}^3$, and some more work can be generalized to more complex spaces, such as $SO(3)$ (see [55]).

3.1.2 Filters

Filters are ubiquitous in image processing. And since density map (see 3.2.2) can be seen as three dimensional images, filters find their application in integrative structural modeling, too. Discrete filters of the type discussed here are usually applied by the operation of convolution. Figure (3.1) illustrates the application of one specific filter. The process is fairly intuitive and so is the generalization to densities. The result $\tilde{p}_{[i,j,k]}$ of a filter *kernel* $f_{[l,m,n]}$ of size $(2L+1, 2M+1, 2N+1)$ applied to the neighborhood of a given pixel $p_{[i,j,k]}$ can be written as

$$\tilde{p}_{[i,j,k]} = \sum_{l=-L}^{L} \sum_{m=-M}^{M} \sum_{n=-N}^{N} f_{[l,m,n]} \cdot p_{[i+l,j+m,k+n]} \tag{3.6}$$

Note that filters always take into account a given environment of a pixel and that any programmatic implementation must take care of the boundaries of the respective pixels. Two widely used filters are the gaussian and the laplace filter. The gaussian filter, as a matrix $f_{[l,m,n]}$ in the above sense, can be realized as a standard Gaussian with a given parameter σ. It has a smoothing effect, the greater σ, the "smoother" the effect is. The

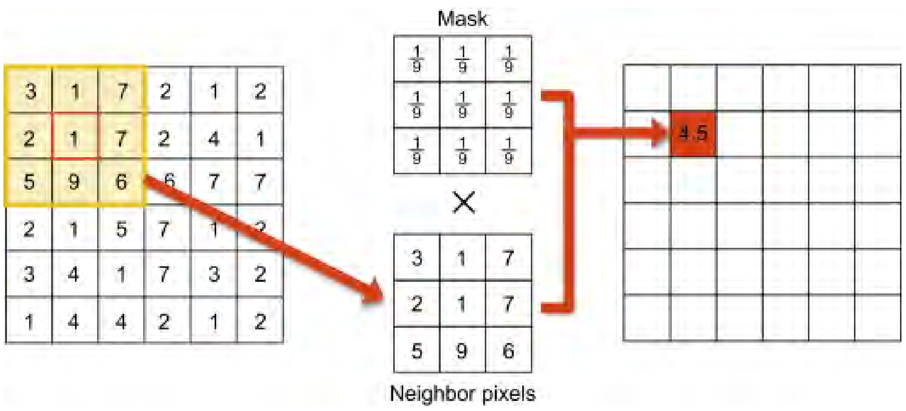
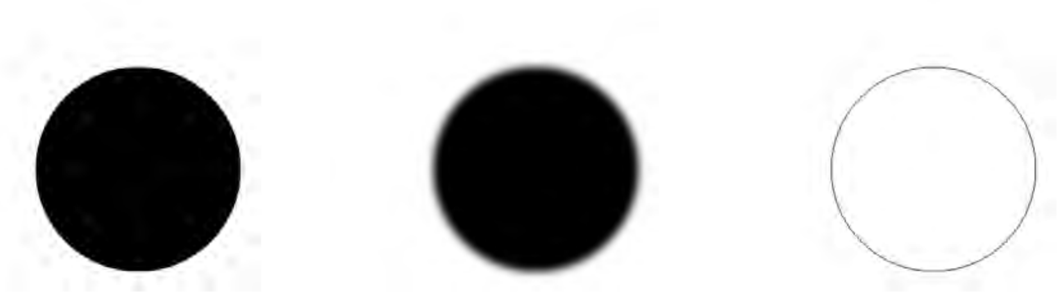

Mask
$\frac{1}{9}$ $\frac{1}{9}$ $\frac{1}{9}$
$\frac{1}{9}$ $\frac{1}{9}$ $\frac{1}{9}$
$\frac{1}{9}$ $\frac{1}{9}$ $\frac{1}{9}$
×
3 1 7 2 1 2
2 1 7 2 4 1
5 9 6 6 7 7
2 1 5 7 1 2
3 4 1 7 3 2
1 4 4 2 1 2
3 1 7
2 1 7
5 9 6
Neighbor pixels
4.5

Figure 3.1: An illustration of the convolution operation used to apply image filters. The array labeled “mask” is what I refer to as “filter”. Taken from [61].



(a) Original image of a black circle. (b) An application of a gaussian filter with $\sigma = 5$. (c) An application of a Laplace filter.

Figure 3.2: Illustration of the effects of the Gauss filter and the Laplace filter.

greater the dimensions of the filter, the more values get taken into account and the more blurred the image becomes. Figure 3.2b gives a visual impression of this effect. An application is density simulation of atomic structures (see 3.2.3), which is accomplished by applying a gaussian filter to point densities whose location and total volume is defined by the atoms. Figure 3.3a shows the two dimensional kernel of size $(3, 3)$ of a Gauss filter with $\sigma = 5$.

Laplace filters are used for edge detection. Figure (3.2c) shows the application of a Laplace filter to figure (3.2) (it is actually the negative image, the adaption was made for visualization purposes). The kernel of the Laplace filter is a discretization of the Laplace operator

$$\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \tag{3.7}$$

and is used to detect “edges”. This is accomplished by measuring the absolute value of the local curvature. An application is the enhancement of contours in single particle

0.1467	0.2419	0.1467
0.2419	0.3989	0.2419
0.1467	0.2419	0.1467

(a) A two dimensional Gauss filter kernel.

0.25	0.5	0.25
0.5	−3	0.5
0.25	0.5	0.25

(b) A two dimensional Laplace filter kernel.

Figure 3.3: Examples of numerical kernels of different filters.

cryo-EM or cryo-ET density maps, e.g. in [49]. Figure (3.3b) shows an example of a two dimensional kernel of a Laplace filter of size $(3, 3)$.

3.2 Data Structures and Algorithms

Computer programs are largely defined by the data that are written in their memory and the operations that are executed on these data. The hardware of a computer constrains what kind of data can be represented and what processes can be realistically run on a given setup. How the user interacts with a program puts additional constraints on the specifics of data structures that are employed. A third set of constraints can originate from the requirements of an algorithm that processes the given data structure. The concrete layout of data structures is therefore a vital matter.
Furthermore it simplifies the exposition of this chapter to start with a clear illustration of the employed data structures and algorithms independently from a concrete hardware setup or file type conventions. This approach highlights the necessity of certain choices as consequences of a computational environment. I will therefore first describe the data structures and algorithm from a purely computational perspective and subsequently discuss the concrete implementations and conventions.
The two most prevalent data structures in integrative structural modeling are *Particle sets* and *Densities.*

3.2.1 Particle sets

Particle sets occur in several instances within structural integrative modeling, most prominently as sets of atoms, as beads in modeling or as the center positions of a set of gaussian densities. Typically particles come in a set of more than one member and every one of these members is at least characterized by a position in a three-dimensional space. This position $\mathbf{p}$ is characterized as a set of coordinates:

$$\mathbf{p} = \begin{pmatrix} x_0 \\ x_1 \\ x_3 \end{pmatrix} \text{ with } x_0, x_1, x_2 \in \mathbb{R}$$

The particle is cast into a standard vector form to later apply well known methods the manipulate it. As the name suggests the data structure Particles rarely appears as set of a single particle, different particles are then differentiated by an integer index: $\{\mathbf{p_0}, \mathbf{p_1}, \dots, \mathbf{p_{P-1}}\}$ with $P \in \mathbb{N}$ being the total number of particles.

Representation in computer memory

While the $x_i \in \mathbb{R}$ for all i are mathematically represented as elements of the real numbers $\mathbb{R}$, it is neither necessary nor possible to represent any real number within computer memory. It is not possible since this would require infinite precision and therefore infinite memory space and it is not necessary since the experimental source of the data instances has its own limit of precision. This limit is usually for lower than what standard data types offer.
The representation of a real number of limited precision in classical memory is so important that a convention has been put in place. I am going to employ mostly the so called single precision floating point format (`float32`) which has been standardized in `IEEE 754`[62]. It has the memory size 32 bit or 4 bytes, so that one particle as characterized above would correspond to 12 bytes. 1 megabyte could then hold $\left\lfloor \frac{1024^2 \text{ B}}{12 \text{ B}} \right\rfloor \widehat{=} 87381$ particles, which is lower than the average number of atoms of protein structures in the PDB. The memory layout is quite natural:

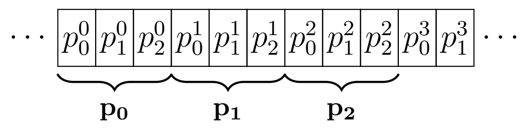


Figure 3.4: Memory layout of a particle set. Each cell corresponds to 4 B of memory.

One set of coordinates would then correspond to an array of size 3. Given this layout the ith coordinate of the nth particle is retrieved by addressing the index j as

$$j = 3n + i$$

Given an index j of a memory cell one gets the particle and coordinate indices:

$$\begin{aligned} n &= j/3 \\ i &= j \bmod 3 \end{aligned}$$

The / of the upper equation signifies integer division.
Often there is a need for a further classification of particles beyond their coordinates, e.g. if the particles signify atoms it can be necessary to remember what type of atom. This can be achieved by use of 4 `float32` instances instead of 3 and adaption of the formulae above. Another practical necessity is the ability to address subsets of particles, which can be achieved as thinking of subsets of particles as subsets of indices, e.g. $\{\mathbf{p_4}, \mathbf{p_8}, \mathbf{p_{12}}\} \mathrel{\widehat{=}} \{4, 8, 12\}$.

The bounding box of a particle set

Given a particle set one is often interested in the "box-shaped"-volume it occupies. This can help to determine the relation of two particle sets, if one is contained in the other, if they overlap, or the simply get a rough estimate of the space they occupy. The bounding box can be defined of the smallest possible cuboid whose axis are parallel to the axis of the coordinate system that contains the coordinates of all particles within the set. Figure 3.5 illustrates this for a small number of particles. A bounding box can be defined as a pair of coordinates $\mathbf{p}$ and $\mathbf{q}$ and a coordinate $\mathbf{r}$ is said to be contained within the bounding box if

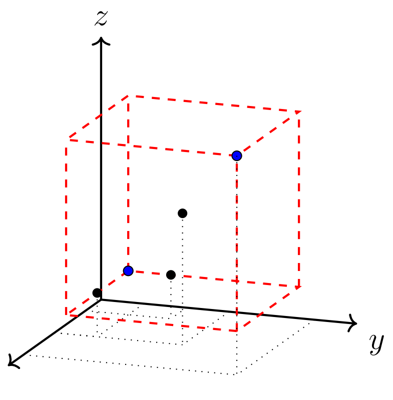


Figure 3.5: Illustration of the bounding box of 4 particles. The blue dots denote the defining coordinates of the bounding box.

$$\begin{aligned} p_0 \leq r_0 \leq q_0 \\ p_1 \leq r_1 \leq q_1 \\ p_2 \leq r_2 \leq q_2 \end{aligned}$$

hold true. This definition immediately provides a method to calculate the bounding box of a particle set. Firstly the inequalities are independent and secondly the minimum and maximum of the set of each single coordinate of all particles fulfill the required definition.

$$\begin{aligned}
\min\{r_0^i \mid i = 0 \dots P-1\} \leq r_0^j \leq \max\{r_0^i \mid i = 0 \dots P-1\} \qquad j = 1 \dots P-1 \\
\min\{r_1^i \mid i = 0 \dots P-1\} \leq r_1^j \leq \max\{r_1^i \mid i = 0 \dots P-1\} \qquad j = 1 \dots P-1 \\
\min\{r_2^i \mid i = 0 \dots P-1\} \leq r_2^j \leq \max\{r_2^i \mid i = 0 \dots P-1\} \qquad j = 1 \dots P-1
\end{aligned}$$

Furthermore the facts that

$$\begin{aligned}
\min(\min(a,b), \min(c,d)) &= \min(a,b,c,d) \\
\max(\max(a,b), \max(c,d)) &= \max(a,b,c,d)
\end{aligned}$$

for any numbers a, b, c, d identify the calculation of a bounding box as a reduction type operation (see section 3.3.6) and therefore suitable for parallel computation methods. Given two bounding boxes A and B there are a number of useful operations that can help to understand the spatial relations between A and B. Relevant are the *intersection*, the *union* and the *contains* operation, which I will outline briefly. The intersection of A and B is defined as the bounding box of all points whose coordinates are both within A and B (see Figure 3.6) and can be calculated like so:

$$\begin{aligned}
\mathbf{p_{AB}} &= \max(\mathbf{p_A}, \mathbf{p_B}) \\
\mathbf{q_{AB}} &= \min(\mathbf{q_A}, \mathbf{q_B})
\end{aligned}$$

The min and max operation are to be executed component-wise, e.g.

$$\min(\mathbf{p}, \mathbf{q}) = \begin{pmatrix} \min(p_0, q_0) & \min(p_1, q_1) & \min(p_2, q_2) \end{pmatrix}^T$$

If any of the components of $\mathbf{p_{AB}}$ is greater or equal to any of the components of $\mathbf{q_{AB}}$, the intersection can be considered to be empty.

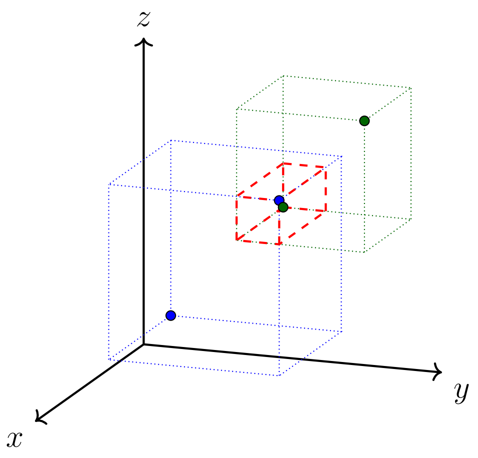


Figure 3.6: Intersection of two bounding boxes.

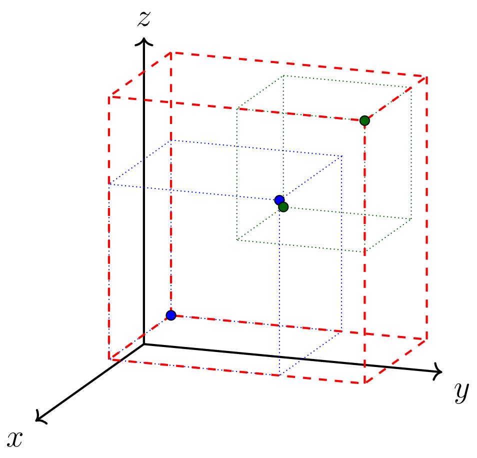


Figure 3.7: Union of two bounding boxes.

The union of A and B is defined in an analog manner (see figure 3.7):

$$\mathbf{p_{AB}} = \min(\mathbf{p_A}, \mathbf{p_B})$$
$$\mathbf{q_{AB}} = \max(\mathbf{q_A}, \mathbf{q_B})$$

Lastly, if A is contained within B can be decided by checking the following two conditions:

$$\mathbf{p_A} \geq \mathbf{p_B}$$
$$\mathbf{q_A} \leq \mathbf{q_B}$$

The operations $\leq$ and $\geq$ again have to be carried out component-wise.

Configurations of particle sets and the RMSD measure

In integrative structural modeling one makes use of particle sets to model rigid bodies. Often the question of the optimal model is reduced to the optimal position of some set of rigid bodies. Since the number of possible configurations is usually exceedingly numerous one ends up with an ensemble of configurations, realized by different orientations and positions of the constituting rigid bodies of a model. The memory layout of a set of configurations can be cast into the form of a set of particle sets (see figure 3.8).

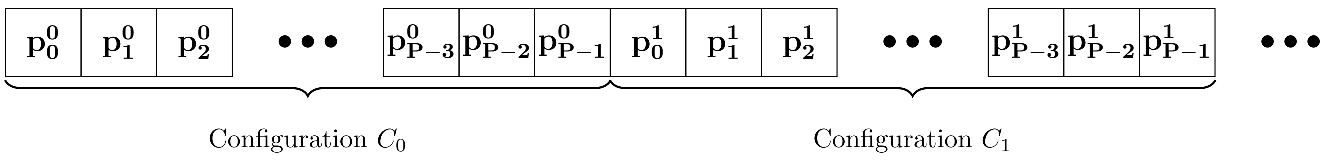


Figure 3.8: Memory layout of a particle set. Each cell corresponds to one set of coordinates, e.g. 12 B of memory for 3 `float32` instances per particle.

For C configurations, P particles, N coordinates the jth `float32` in this memory layout, the cth configuration, the pth particle and the ith coordinate are connected by

$$\begin{aligned} j &= c \cdot P + p \cdot N + i \\ c &= j/(P \cdot N) \\ p &= (j - c \cdot P)/N \\ i &= j - c \cdot P - p \cdot N \end{aligned}$$

For the following reasons one might want to compare different configurations:

- Clustering. Often the result of a modeling procedure is a big number of configurations. It is not practical to review every configuration by visual inspection. Given a measure of similarity (see 3.8) a clustering of configurations becomes possible. Depending on the specifics of the clustering procedure this can provide the modeler with an overview over the ensemble of configurations.
- Alignment. If the given set of configuration is not oriented with reference to some coordinate system, it is not possible to compare different configurations. One way to solve this situation is to reorient the configurations so that they are in some way (see 3.8) optimally aligned.

A common measure of closeness is the so called *RMSD*, the *root mean square deviation*. Its definition spells

$$\mathrm{RMSD}(\mathbf{v}, \mathbf{w}) = \sqrt{\frac{1}{P} \sum_{i=0}^{P-1} \|\mathbf{v}_i - \mathbf{w}_i\|^2} \tag{3.8}$$

where the vectors in the sum are corresponding to the coordinates of single particles. So $\mathbf{v}$ and $\mathbf{w}$ correspond to two configurations and the index i refers to the individual particles. The RMSD is a generalization of the euclidean distance between two particles. Figure (3.9) illustrates the principle. The distance between corresponding pairs is summed up and then, analogously to the euclidean distance, the square root is calculated. If the

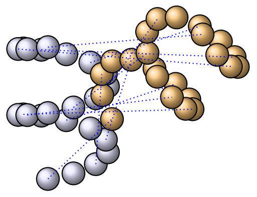


Figure 3.9: Illustration of RSMD of two configurations. Every second pair of corresponding particles is connected by a line denoting the distance between that pair.

configurations coincide perfectly, the RMSD is 0. There are two causes why the RMSD might not be 0. Either the two configurations describe different confirmations of the same particle set. In that case the RMSD can be read as a measure of difference between two configurations. The other cause might be that the two configurations occupy different orientations and positions in space. Figure (3.9) for instance shows an example of this case. Judging from the RMSD alone it is not possible to distinguish these two causes. To do this one uses a RMSD minimization alignment algorithm as described in the next section.

Particle set alignment by RMSD minimization

The value of the RMSD function is bounded by 0. Equation (3.8) is continuous and smooth, it has a minimum value for all given relative configurations between two particle sets. The alignment procedure seeks to orient two configurations in such a way that the RMSD becomes minimal. The alignment algorithm described below achieves this by providing a rotation that, if applied, transforms one of the particle set configurations so that the RMSD between the two will be minimal. Before applying this algorithm however one has to take care that the centers of mass of the two configurations coincide. The *center of mass* is defined as

$$\mathbf{R}_{com} = \frac{1}{P}\sum_{i=1}^{P-1} m_i \mathbf{v}_i,$$

The numbers m_i can be interpreted as weights, so that "heavier" particles contribute more to the calculation. Once calculation of the center of mass is a reduction type operation and therefore amenable to parallelization. Once the center of mass is calculated, the particle set can be translated

$$\mathbf{v}_i \rightarrow \mathbf{v}_i - \mathbf{R}_{com}$$

so that the center of mass coincides with the origin.

Now the structural alignment can be executed. I will follow the approach of [63], but only describe the actual steps needed to actually perform the calculation. Given two configurations $\{\mathbf{v}_{\mathbf{0},0}, \mathbf{v}_{\mathbf{0},1}, \ldots, \mathbf{v}_{\mathbf{0},P-1}\}$ (the reference configuration) and $\{\mathbf{v}_{\mathbf{j},0}, \mathbf{v}_{\mathbf{j},1}, \ldots, \mathbf{v}_{\mathbf{j},P-1}\}$ (any other configuration $\mathbf{j}$ of the same particle set). We define an error term

$$\boldsymbol{\varepsilon}(\mathbf{q}) \coloneqq \mathbf{D}(\mathbf{q})\mathbf{v}_{\mathbf{0}} - \mathbf{v}_{\mathbf{j}}$$

which expresses the deviation between a transformed reference configuration and the other configuration. The transformation $\mathbf{D}(\mathbf{q})$ is parametrized by the quaternion $\mathbf{q}$ and that be translated into a rotation matrix (see 4.2.3). The algorithm consists of four steps. First, calculate for all particle pairs

$$(\mathbf{v}_{\mathbf{0},i}, \mathbf{v}_{\mathbf{j},i}) = \left(\begin{pmatrix} x_{\mathbf{0},i} & y_{\mathbf{0},i} & z_{\mathbf{0},i} \end{pmatrix}^T, \begin{pmatrix} x_{\mathbf{j},i} & y_{\mathbf{j},i} & z_{\mathbf{j},i} \end{pmatrix}^T \right)$$

the symmetric real matrices $\mathbf{M}^{\mathbf{j}}_i$

$$\begin{aligned}
\mathbf{M}^{\mathbf{j}}_{i,00} &= x_{\mathbf{j},i}^2 + y_{\mathbf{j},i}^2 + z_{\mathbf{j},i}^2 + x_{\mathbf{0},i}^2 + y_{\mathbf{0},i}^2 + z_{\mathbf{0},i}^2 - 2x_{\mathbf{j},i}x_{\mathbf{0},i} - 2y_{\mathbf{j},i}y_{\mathbf{0},i} - 2z_{\mathbf{j},i}z_{\mathbf{0},i} \\
\mathbf{M}^{\mathbf{j}}_{i,01} &= 2(y_{\mathbf{j},i}z_{\mathbf{0},i} - z_{\mathbf{j},i}y_{\mathbf{0},i}) \\
\mathbf{M}^{\mathbf{j}}_{i,02} &= 2(-x_{\mathbf{j},i}z_{\mathbf{0},i} + z_{\mathbf{j},i}x_{\mathbf{0},i}) \\
\mathbf{M}^{\mathbf{j}}_{i,03} &= 2(x_{\mathbf{j},i}y_{\mathbf{0},i} - y_{\mathbf{j},i}x_{\mathbf{0},i}) \\
\mathbf{M}^{\mathbf{j}}_{i,11} &= x_{\mathbf{j},i}^2 + y_{\mathbf{j},i}^2 + z_{\mathbf{j},i}^2 + x_{\mathbf{0},i}^2 + y_{\mathbf{0},i}^2 + z_{\mathbf{0},i}^2 - 2x_{\mathbf{j},i}x_{\mathbf{0},i} + 2y_{\mathbf{j},i}y_{\mathbf{0},i} + 2z_{\mathbf{j},i}z_{\mathbf{0},i} \\
\mathbf{M}^{\mathbf{j}}_{i,12} &= -2(x_{\mathbf{j},i}y_{\mathbf{0},i} + y_{\mathbf{j},i}x_{\mathbf{0},i}) \\
\mathbf{M}^{\mathbf{j}}_{i,13} &= -2(x_{\mathbf{j},i}z_{\mathbf{0},i} - z_{\mathbf{j},i}x_{\mathbf{0},i}) \\
\mathbf{M}^{\mathbf{j}}_{i,22} &= x_{\mathbf{j},i}^2 + y_{\mathbf{j},i}^2 + z_{\mathbf{j},i}^2 + x_{\mathbf{0},i}^2 + y_{\mathbf{0},i}^2 + z_{\mathbf{0},i}^2 + 2x_{\mathbf{j},i}x_{\mathbf{0},i} - 2y_{\mathbf{j},i}y_{\mathbf{0},i} + 2z_{\mathbf{j},i}z_{\mathbf{0},i} \\
\mathbf{M}^{\mathbf{j}}_{i,23} &= -2(y_{\mathbf{j},i}z_{\mathbf{0},i} - z_{\mathbf{j},i}y_{\mathbf{0},i}) \\
\mathbf{M}^{\mathbf{j}}_{i,33} &= x_{\mathbf{j},i}^2 + y_{\mathbf{j},i}^2 + z_{\mathbf{j},i}^2 + x_{\mathbf{0},i}^2 + y_{\mathbf{0},i}^2 + z_{\mathbf{0},i}^2 + 2x_{\mathbf{j},i}x_{\mathbf{0},i} + 2y_{\mathbf{j},i}y_{\mathbf{0},i} - 2z_{\mathbf{j},i}z_{\mathbf{0},i}
\end{aligned}$$

The remaining indices of these 4×4 matrices have been omitted due to its symmetry. Second, the matrices $\mathbf{M}^{\mathbf{j}}_i$ are summed up in the manner of (4.12)

$$\mathbf{M}^{\mathbf{j}} = \sum_{i=0}^{P-1} \mathbf{M}^{\mathbf{j}}_i$$

Third, determine the eigenvalues and eigenvectors $\lambda^{\mathbf{j}}_i$ and $\mathbf{v}^{\mathbf{j}}_{\lambda_i}$ of $\mathbf{M}^{\mathbf{j}}$, where the index i now refers to the ith eigenvalue. Lastly, select the eigenvector $\mathbf{v}^{\mathbf{j}}_{\text{min}}$ corresponding to the lowest eigenvalue $\lambda^{\mathbf{j}}_{\text{min}}$ (which are all real and therefore well-ordered due to the symmetry of the matrix) and interpret it as a quaternion:

$$\mathbf{v}^{\mathbf{j}}_{\text{min}} \hat{=} \left[{q_0}^{\mathbf{j}}, {q_1}^{\mathbf{j}}, {q_2}^{\mathbf{j}}, {q_3}^{\mathbf{j}}\right]$$

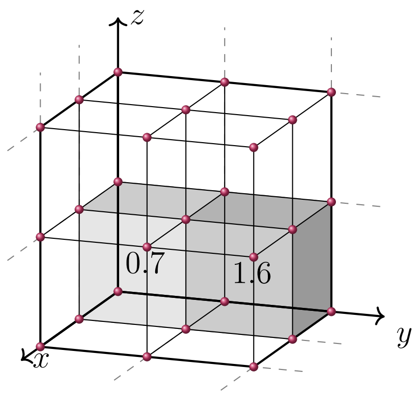


Figure 3.10: The three-dimensional intuition of a density. Shown are two example pixels of different values and the three-dimensional grid underlying a density data structure.

The corresponding matrix $\mathbf{D}(\mathbf{q}^{\mathbf{j}})$ will be the transformation aligning the two configurations so that their RMSD is minimal. The whole process can be extended for the alignment of any number C of configurations by varying the index $\mathbf{j}$ and can be used in conjunction with the memory layout and address scheme (3.8). Especially the first, the second and the third step are amenable for parallelization.

The PDB file format

The PDB file format has been the standard file format to convey the results of crystallographic, cryo-EM, NMR or modeling based structure determinations for a long time and to a great extend still is. It is a line-and-column based file format that holds information about positions of particles, chemical bonds, secondary protein structures or experimental information. [64] holds the current specifications. For modeling purposes the “Coordinate section” is the most interesting.

3.2.2 Densities

Densities are the second prominent data type in structural integrative modeling. They can represent the final result of a x-ray diffraction experiment (see 2.4.1), a cryo-electron microscopy (see 2.4.2) or cryo-tomography (see 2.4.3) reconstruction or a simulated electron density map. Densities are always saved in discrete chunks, commonly called *voxels* or *pixels* that add up to a cuboid. These pixels (remark: I will call them pixels, regardless their spatial dimension, 2- or 3D) are associated with two data fields: their *value*, which can be a real number, an integer or a complex number and their *position* in three-dimensional space,arranged in a grid-like fashion. For our purposes the value will be assumed to be a real number. The pixels are also assumed to have the shape of a cube of

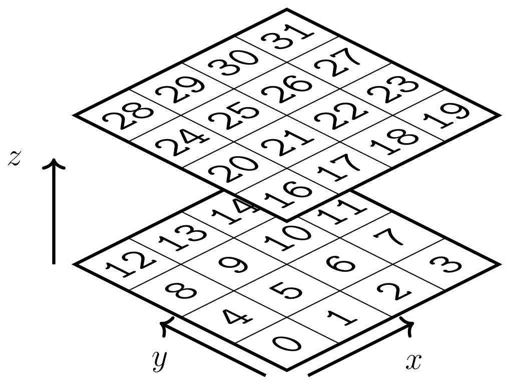


Figure 3.11: The numbering scheme as related to the three dimension grid.

constant extensions. Figure 3.10 illustrates this.

Given the size of one pixel a density could then be modeled as a list of pixels, specified by their positions and values. This is wasteful and can be done in a more parsimonious manner, if one is willing to adapt a convention. The key is the underlying grid of the cuboid. Figure (3.11) illustrates an example of such a convention. The basic idea is to agree on where the “jumps” are going to occur. In this illustration the count run along the x-axis and then jumps one unit along the y-axis (beginning with $0, 4, 8, \ldots$) to continue. Once one xy layer is filled, the count jumps to the lowest x, y position (`16`) on the next layer, moving along the z axis. If one now specifies the total number of pixels along each axis (e.g. `[40,40,40]`) and the size in real space that the whole density corresponds to (e.g. `[103 Å,103 Å,103 Å]`), the density is described completely. It is equivalent to specifying the pixel size (e.g. `[2.575 Å,2.575 Å,2.575 Å]`) and the number of pixels in each dimension.

Representation in computer memory

Given the convention outlined above, a density can be represented in computer memory the following way. I will denote the spatial dimensions of the density as $\mathbf{d}$ (referred to as *coordinate dimension*) and the number of pixels along each axis as $\mathbf{p}$ (referred to as *pixel dimension*). Both can be represented by 12 B of memory, the components of $\mathbf{d}$ modeled as a `float32` and the components of $\mathbf{p}$ modeled as a `int32`. The values assigned to the single pixels can take, as mentioned above, different forms, I will model them as `float32`. There are conventions for laying out the pixel values in memory differing from the above mentioned instance, I will however always choose this one. This representation is shown in figure (3.12). There is an important distinction between how pixels are related geometrically and how they are related in linearized computer memory. Often it is necessary to translate between the two representations: the value in the ith memory

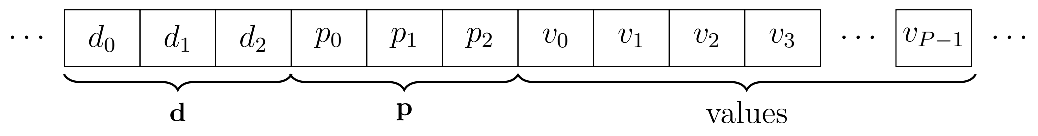


Figure 3.12: Memory layout of a density. Each cell corresponds to 4 B of memory. The value section is of $p_0p_1p_2$ cells length.

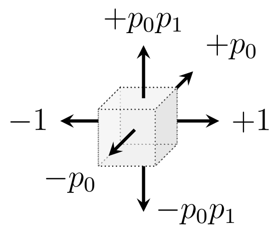


Figure 3.13: Offsets in linear memory and the movement in three-dimensional space they correspond to.

unit corresponds to the pixel $[k, l, m]$, in the layer m, the row l and the position k (z, y, x), as indicated in figure (3.11). The formulae of conversion are:

$$i = m \cdot p_0p_1 + l \cdot p_0 + k \tag{3.9}$$
$$m = i/(p_0p_1) \tag{3.10}$$
$$l = (i - m \cdot p_0p_1)/p_0 \tag{3.11}$$
$$k = i - m \cdot p_0p_1 - l \cdot p_0 \tag{3.12}$$

Here the division used is integer division. Given $\mathbf{p}$, $\mathbf{d}$ and an index, one can calculate the pixel size and the position $\mathbf{v}$ of a given pixel $[k, l, m]$ as follows.

$$v_0 = \left(\frac{d_0}{p_0} + 0.5\right) \cdot k$$
$$v_1 = \left(\frac{d_1}{p_1} + 0.5\right) \cdot l$$
$$v_2 = \left(\frac{d_2}{p_2} + 0.5\right) \cdot m$$

It will be necessary to relate pixels to their three-dimensional neighborhood, e.g. finding pixels that surround a given pixel. While in three-dimensional space "above" and "below" are clear notions, it is different in the linear representation mentioned above. It is, however, always the same offset that separates two adjacent pixels in a given direction. In figure (3.11) for instance, the pixel one level above another always has an index increment of 16. So one can move in the linearized representation of a density as one does in three-dimensional space using this fact. Figure (3.13) summarizes the offsets.

3.2.3 Molecular density simulation

To enable the use of fitting scores that fit density maps to other density maps for the case of fitting molecular structures represented by particle sets to density maps a method to simulate a density map on the basis of a particle set is needed. A prominent instance to achieve this is the following method.

In its core the method calculates densities per particle and per pixel. Figure (3.14) illustrates this process in a two dimensional density. The three-dimensional case works completely analogical. For every particle the distance to every pixel is calculated, where the spatial position of pixels is determined according to 3.2.2. The distance is calculated the classical way

$$d = \sqrt{\sum_{i=0}^{2} (\mathbf{P}_i - \mathbf{p_i})}$$

where $\mathbf{P}$ denotes the particle coordinates and $\mathbf{p}$ the pixel coordinates. As a next step the intensity for each pixel for each particle is calculated according to a chosen function which converges to 0 towards infinity and is greater than 0 everywhere on the positive x-axis. A common choice is a version of the Gaussian function. Since this simulation of densities is used in the context of fitting crystal structures to density maps obtained by single particle cryo-EM microscopy the Gaussian is chosen in a way to mimic the limitation put up by the resolution achieved in experiment. It is assumed that the resolution R is related to the Gaussian ρ in such a way that

$$R = 2d \Leftrightarrow \rho \text{ is at half the value of its maximum}$$

A short calculation shows that this is the case when

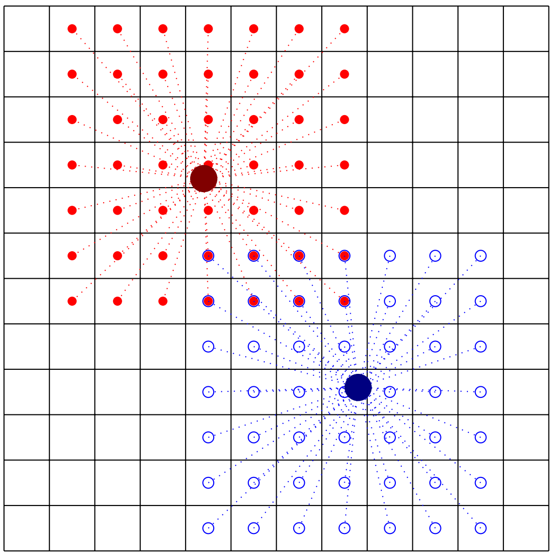
$$\sigma_R = \frac{1}{2\sqrt{2\ln 2}} \cdot R$$

with σ_R signifying the standard deviation of the Gaussian fulfilling the condition above. The Gaussian is then

$$\rho_R(d) = \frac{Z}{\sqrt{2\pi}\sigma_R} \exp\left[-\frac{1}{2}\left(\frac{d}{\sigma_R}\right)^2\right]$$

where Z denotes the atomic number of the the atom that is represented by the particle in question. Since the integral of the standard Gaussian is 1, this particular version of the Gaussian function has the integral $\int_{\mathbb{R}} \rho_R(d)\,\mathrm{d}d = Z$ encoding the intuition that atoms of higher atomic number Z should contribute more to the density.

In practice it is computationally expensive to calculate this for every pixel in a given density, so a cutoff is put in place that prevents the calculation of function values once the pixels are too far away from the particle in question. Once this is achieved for every


(a) Pixel distance calculation.

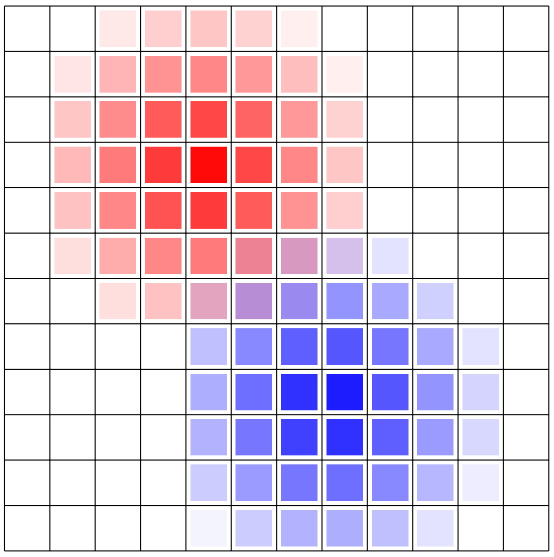

(b) Subdensities and overlap.

Figure 3.14: Illustration of density generation

particle, the densities are summed up for each pixel, every particle potentially contributing to a given pixel.

$$\rho_{\text{total}}\,(\text{pixel}) = \sum_{i=0}^{P-1} \rho_R\,(P_i, \text{pixel})$$

with P_i being the ith particle. This is illustrated in figure (3.14b).

The MRC/CCP4 file format

The density data type is often saved in the `MRC/CCP4` file format. This file format has a header of `1024 bytes` which holds the basic geometric parameters necessary to describe a volume and additional information, which are insubstantial to this discussion. After the header the density is saved as binary information according to the specifications within the header. Typically any reader will evaluate the header first and then construct a data representation of the density from the binary data. The format of the header is a convention and can be assessed at [65].

3.2.4 The connected components labling algorithm

Basic description of the algorithm

The *C̲onnected C̲omponents L̲abeling* algorithm (henceforth “CCL”) is a well-known and important algorithm in image processing (see [66]). The data structure it operates on

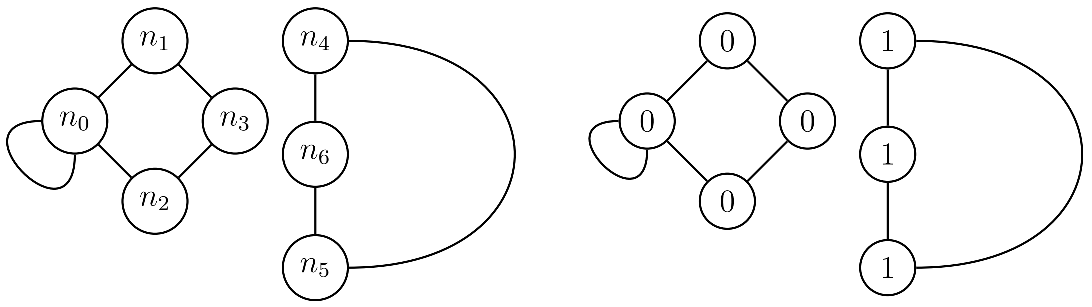


Figure 3.15: Illustration of the CCL algorithm: Left shows an input graph, right shows the labeling.

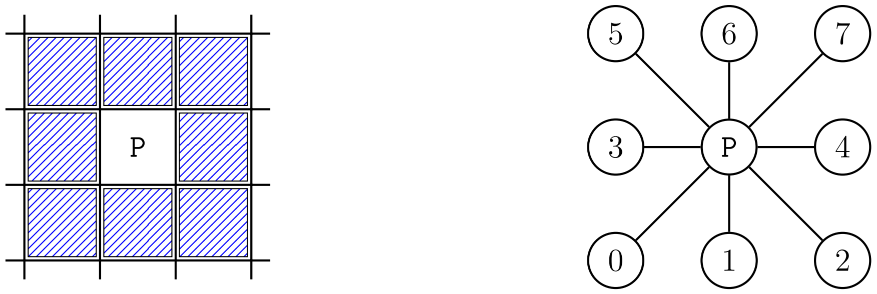


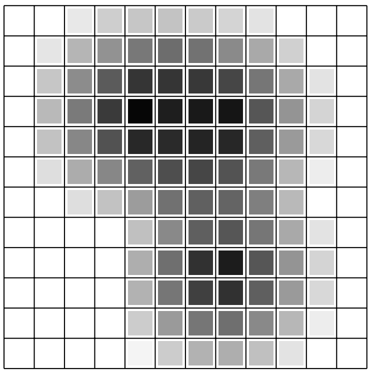
Figure 3.16: Depiction of a 8-neighborhood in a 2-dimensional grid of pixels.

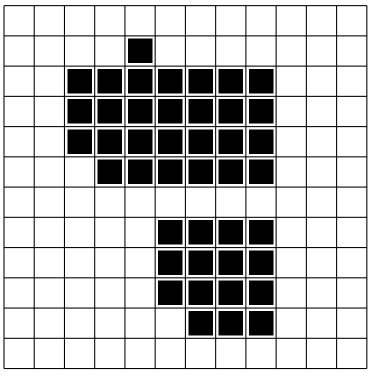
is an (undirected) graph data structure, yet in image processing it is simply a two or three-dimensional array of pixels or voxels, respectively. Pixels are then understood as the nodes of the graph and the edges are assigned to neighboring pixels. Its purpose, as the name suggests, is to label connected components. A connected component of a graph can be defined a set of set of nodes that are mutually impossible to reach by moving along the edges. Figure (3.15) shows a simple graph which has two connected components, $\{\{n_0, n_1, n_2, n_3\}, \{n_4, n_5, n_6\}\}$ labeled by "0" and "1". The labels do not have to be natural numbers, but are often chosen to be represented as such.

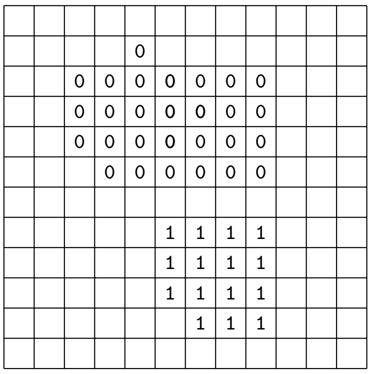
In an image processing environment, two pixels are seen as "connected" if they are part of a predefined neighborhood, a common choice for two dimensional images is the so called "8-neighborhood, illustrated in figure (3.16). This effectively gives the pixels constituting the image a graph data structure and makes the image data structure accessible to the operation of the CCL algorithm. In practice, a binary mask is applied to an image effectively segmenting it into different areas by some chosen criteria. This segmented image is then the input to the CCL-algorithm. The masking criterion might be a great number of things, such as a threshold-criterion for a pixel value or a binary function of the neighboring pixels, such as a threshold on a gradient norm or the norm of a Hessian or Laplacian.

Figure (3.17) illustrates this by use of an example.

A possible implementation of the CCL-algorithm is the so called *UnionFind*-algorithm. Figure (3.18) elucidates the following description: The *UnionFind* algorithm can be un-

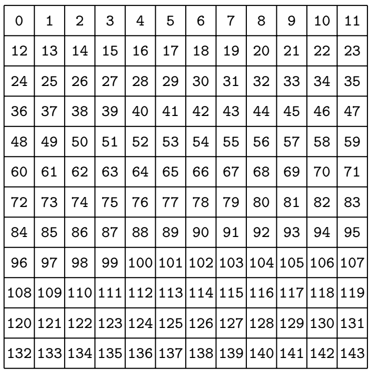

(a) The raw image.


(b) An applied density threshold.

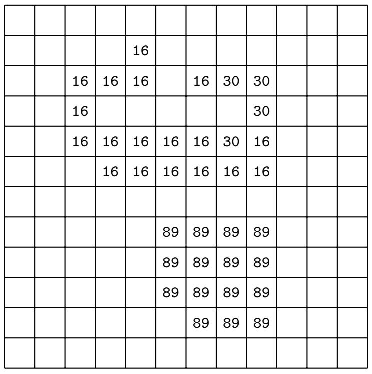


(c) A labeling of the thresholded image.

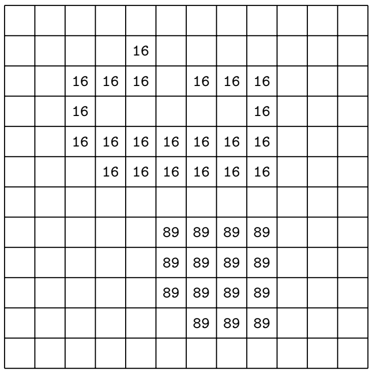
Figure 3.17: An illustration of the application of the CCL algorithm in image processing.



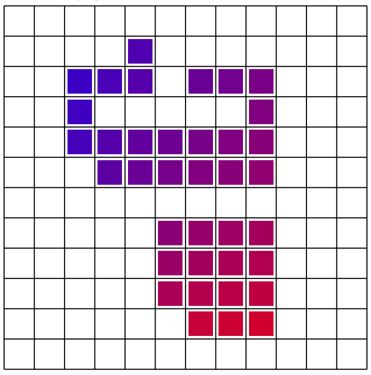
(a) A `labels` array.

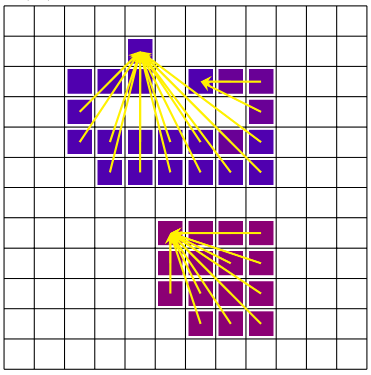


(b) `labels` after the first pass.



(c) `labels` after the second pass.


(d) A `segmentation` array.


(e) `segmentation` after the first pass.

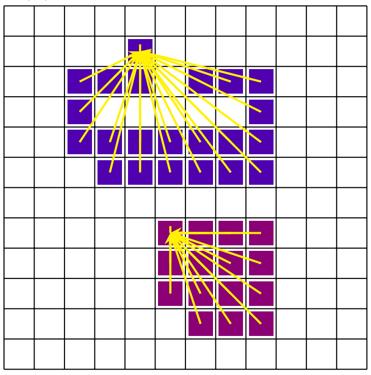

(f) `segmentation` after the second pass.

Figure 3.18: An illustration to aid the explanation of the *UnionFind* algorithm. The colors in the `segmentation` array symbolize individual labels.

derstood in two steps and by looking at how these effect the underlying data structures:

- The input of the algorithm are two arrays, the `segmentation` (see figure 3.18d) and the `labels` array (see figure 3.18a). The `segmentation` array must hold binary or multi class segments, which can be saved in the form of integers. If the `segmentation` is a binary mask, it can be created by applying a `threshold` to an existing density, e.g. setting all `segmentation` values below the `threshold` to `0` and otherwise to `1`. There is no limit on the number of segments except the pixel number. The `labels` array holds initially one distinct label for each pixel.

- In the first pass every label `l` is accessed and in turn all neighbors `{n0,n1,...}` of the 8-neighborhood (see 3.16) that have been iterated already are considered. For each pair `l,n` one checks if the pixels belong to the same segment:

  ```
  if segmentation[l] == segmentation[n]:
          Union(labels,l,n)
  ```

  If that is the case, the `Union` operation is performed. This operation seeks the `root` of both pixels, selects the minimum and writes it to both the addresses of the pixels in the `labels` array.

  ```
  def Union(labels,l,n):
          r0 = find(labels,l)
          r1 = find(labels,n)
          if r0 < r1:
                  labels[r1] = r0
          if r1 < r0:
                  labels[r0] = r1
  ```

  where `Find` seeks out the `root` of each label.

  ```
  def Find(labels,l):
          while (labels[l] != l)
                  l = labels[l]
          return l
  ```

  The labels play a double role as labels on the one hand and addresses on the other hand. If a label `l` is identical with its address, it is called a `root`. Otherwise it will refer to the address of yet another label, which might again be either a `root` or refer to yet another label. The nature of the `Union` operation ensures that this chain terminates at a `root`. Effectively the operations `Union` and `Find` create a tree-structure, one of which will in the end cover every connected component. Figure

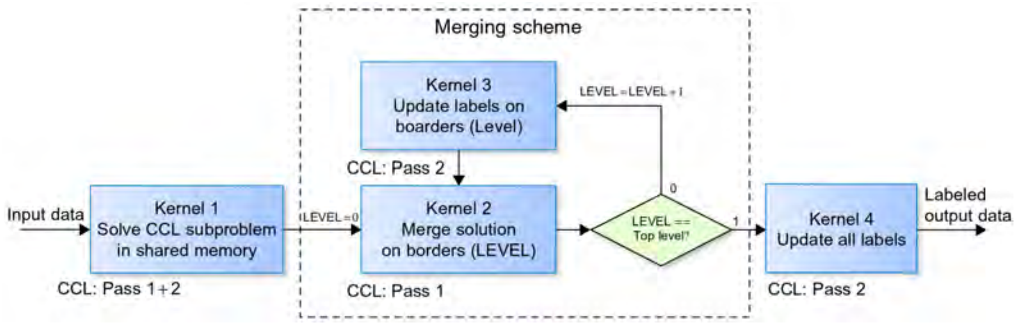


Figure 3.19: The kernel structure of a parallel CCL-implementation. Taken from [66].

(3.18b and 3.18d) show the results of these steps in our example. The arrows indicate the above mentioned chains.

- The second pass “unifies” the label chains, one also speaks of “flattening” the trees. To every node (pixel) of a tree will be assigned the label of the `root` node, again using the `Find` operation.

```
for l in range(#LABELS):
        labels[l] = find(labels,l)
```

Figure (3.18c and 3.18f) show the result of the second pass.

The parallel version of the algorithm

The solution present above is sequential in nature and therefore not amenable to implementation in a parallel processing environment. However, [66] have offered a solution for this problem for the case of 2-dimensional images. In their publication they offer the pseudocode of an implementation structured as indicated in figure (3.19). Their innovation lies in the use of the `shared` memory and an a solution to the issue of non-sequentiality of parallel processing. `Kernel1` in figure (3.19) assigns one `thread` to check the neighbors of each `pixel` and conduct the `Union` and the `Find` operation. Because there is no guarantee which `thread` will finish first and the labels will most likely not be propagated in sequence and therefore might not have an unbroken chain to their `root` label, they implemented an internal loop, which runs until no label is changing its value anymore. They accomplish this by checking a flag which is accessed using atomic operations (see 3.3.4) and serves as a thread-safe communication device for each `block`.
Due to the size limitation of `shared` memory, they are left with a number of equally-sized subvolumes, which are consistently labeled in themselves but need to be merged into the complete volume. This is accomplished by `Kernel2` and `Kernel3`. `Kernel2` merges the labels on the surfaces between the already labeled subvolumes on growing interfaces (see figure 3.20). Each iteration the merging interfaces grow to accommodate larger and larger

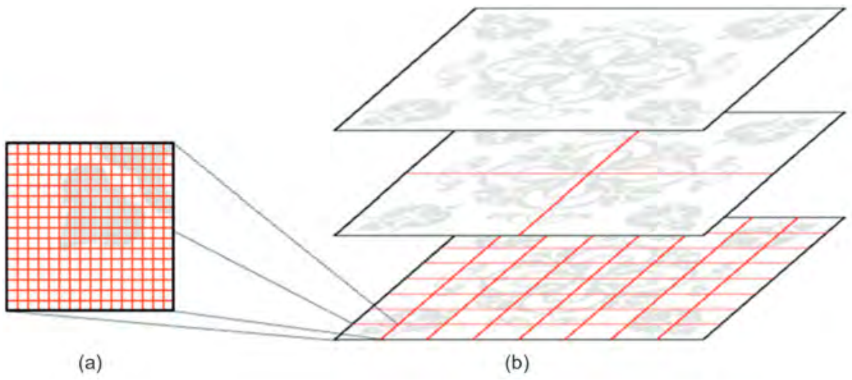


Figure 3.20: The merging scheme used in `Kernel2`, taken from [66].

subvolumes. `Kernel3` flattens the tree data structures in the sense elaborated above for a gain in performance. The final, global flattening of the tree structure is facilitated by `Kernel4`. All kernels are called from one kernel, utilizing dynamic parallelism (see 3.3.5), to minimize host interaction during execution.

3.3 Parallel programming

Parallel programming addresses the task of conceptualizing and implementing computational procedures on parallel computation devices. The existence of such devices is owed to the existence of a class problems that historically arose in the field of computer graphics (hence the common name GPU). In recent years however, scientific computing has massively expanded due to innovations in both parallel computation hardware and programming interfaces for this hardware. A comprehensive introduction to the most advanced programming interface, CUDA, can be found at [67]. The data types and connected algorithms that are tractable in a parallel setting have to exhibit certain quality, which is commonly abbreviated SIMD (Single Instruction, Multiple Data). This property signifies data that can be described as similar chunks of the same type and algorithms that operate on these data chunks independently at the same time without interfering with each other. The data types and algorithms used in many problem settings within structural biology exhibit this property. This part of the method section describes the central concepts used within the source code attached to this thesis and serves to make it accessible to change and improvement.

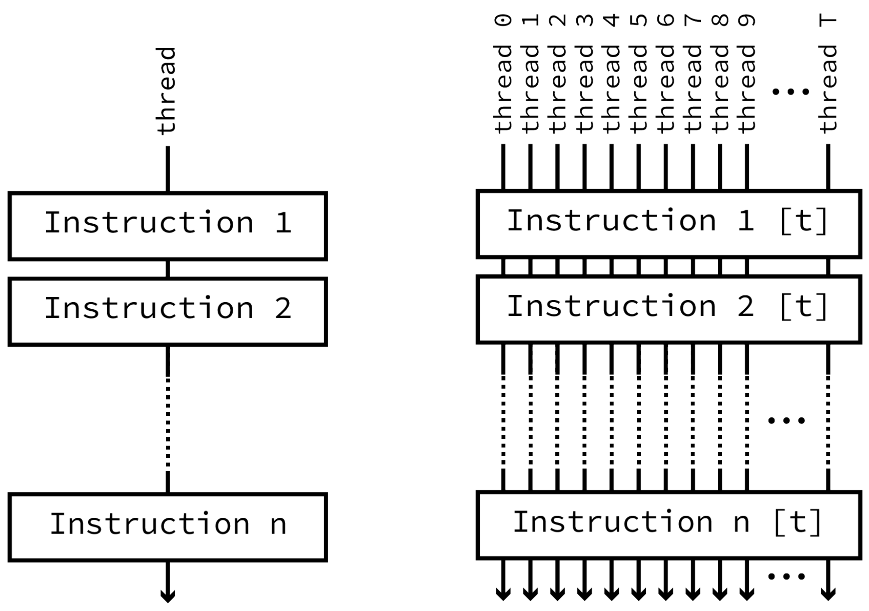


Figure 3.21: Contrasting the difference between sequential (*left*) and parallel (*right*) processing.

3.3.1 The memory hierarchy and the processing model

The central ideas of parallel programming are closely tied to the architecture of the parallel processing device, in short *device*, and to introduce them it is convenient to describe this architecture.

Figure (3.21) contrasts the basic difference between sequential and parallel programming. The `thread` can be viewed as the "string" that runs through all instructions and which defines the call order of these instructions. Sequential programming is characterized by employing only one thread, if this thread needs to perform the same instructions on a massive amount of data, the programming idiom of a "loop" is used, which simply repeats the same set of statements within the same thread. In parallel programming on the other hand one needs to coordinate a number `T` of `threads`. Each of these `threads` follows the same set of instructions, with the slight difference that each instruction targets and processes a different chunk of data. To do this, every `thread` is associated with a `thread id`, which can be use to index the data chunk that is to be processed by the associated `thread`. These instructions are specified within a so called `kernel`, which is the source code that will be submitted to the parallel processing device. This might look like this:

```
__global__
void square_kernel(int * d_input, int * d_ouput){
        d_ouput[threadIdx.x] = d_input[threadIdx.x]*d_input[threadIdx.x];
}
```

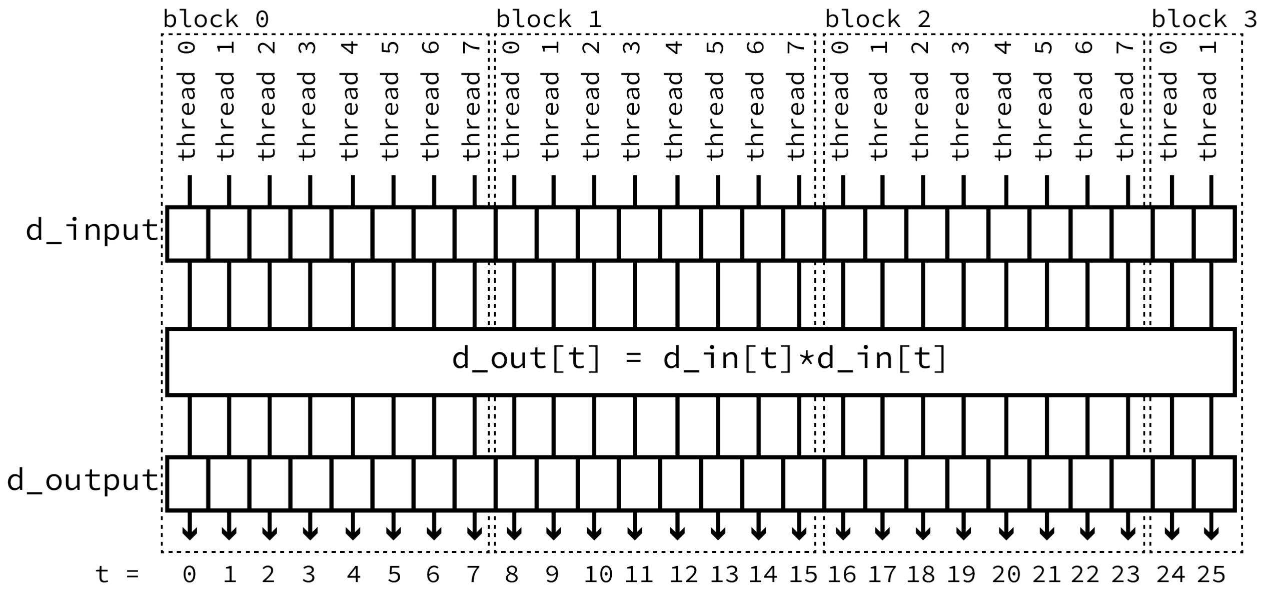


Figure 3.22: blocks divide the workload, every block is associated with its own group of threads.

Several remarks are in order: `threadIdx.x` is a canonical variable unique to each `thread`. The `.x` part mirrors that the `threads` are modeled as a one dimensional grid. It is also possible to model a kernel with two or three-dimensional grids using `threadIdx.y,threadIdx.z`. The `__global__` macro identifies the function as a `kernel`. A `kernel` does not have a return type, so the output memory location has to given as a parameter of the function. While this example kernel will run in this form, it is a simplification. Another layer of structure is needed to complete the picture: `blocks`. A `block` represents a chunk of the computational work to be done and is assigned a number of `threads T`. Figure (3.22) illustrates this. Every `block` has its associated id, `blockIdx.x`. `blocks` can also be organized in different layouts using the canonical variables `blockIdx.x,blockIdx.y,blockIdx.z`. A more realistic version of the above kernel would then be:

```
__global__
void square_kernel(int * d_input, int * d_ouput, int N){
        int t = blockDim.x*blockIdx.x + threadIdx.x;
        if (t < N)
                d_ouput[t] = d_input[t]*d_input[t];
}
```

As indicated in figure (3.22), the variable `t` can now be used to uniquely identify a input data chunk. This snippet also highlights the use of the `blockDim.x` canonical variable, whose value is the number of `threads` per `block`. In the above example this is `8`. Since all `blocks` have the same number of `threads`, an `if` statement is used to prevent overreach by checking if the index `t` is greater than the size `N` of the `d_input` and `d_output` arrays. This means that the redundant `threads` will not perform any operation.Each of the `blocks` is run independently. The user can submit a work load, given the resources have been copied to device memory, by calling the kernel function and specifying the number of `threads` per `block` and the number of `blocks` like so:

```
square_kernel<<<gridDim,blockDim>>>(d_input,d_output,N);
```

with `gridDim` and `blockDim` denoting the number of `blocks` and `threads`, respectively. Both the `gridDim` and `blockDim` are limited by the hardware specifications. The totality of blocks is called a `grid`. As already indicated by the canonical variables `threadIdx.x`, `blockIdx.x` and `blockDim.x`, there is a special set of language conventions: The canonical variables are device code, the above call pattern (`kernel<<<gridDim,blockDim>>>(...arugments)`) is host code. After this host call, the underlying device resource management system will distribute the `blocks` of workload to different streaming multiprocessors (SM) of the device. Every GPU has a number of this SMs, each SM can run the workload of one or multiple blocks simultaneously.
The `threads` of different `blocks` cannot directly communicate with each other. Within one `block` however, `threads` can communicate in a number of ways. One of these is the use of so called `shared` memory. `shared` memory is one of the memory types available on a device. In fact, it is the third type discussed, since `d_input` and `d_output` are written to `global` memory and the variable `t` the `register` memory. The memory hierarchy on a parallel computation device is an important feature and requires discussion. Figure (3.23) visualizes the different memory types. The types of memory differ by volume, life time and accessibility. To harness them in a optimal way, one needs to incorporate these properties into the implementation of the parallel computation procedure:

- `register` memory. This memory type is the fastest one and the "closest" to the SMs. The variable `t` in the kernel above is an example. Each `thread` has its own instance of variables saved in the `register`. One often aims to run as many `blocks` as possible simultaneously on a SM. The `register` memory effectively limits this number, since its memory volume is limited per SM, meaning all `blocks` that run simultaneously have to share this limited resource. For instance, the byte volume `register` memory on an `GeForce GTX 1060` is `65536`, corresponding to 2^{14} `float` instances. If a kernel is defined in such a way that its requirement for `register` memory exceeds this capacity, it cannot be run on the device. The lifetime of this memory type is the lifetime of the `block`, meaning, once the computations associated with the `block` are finished, the information saved in the `registers` is lost.

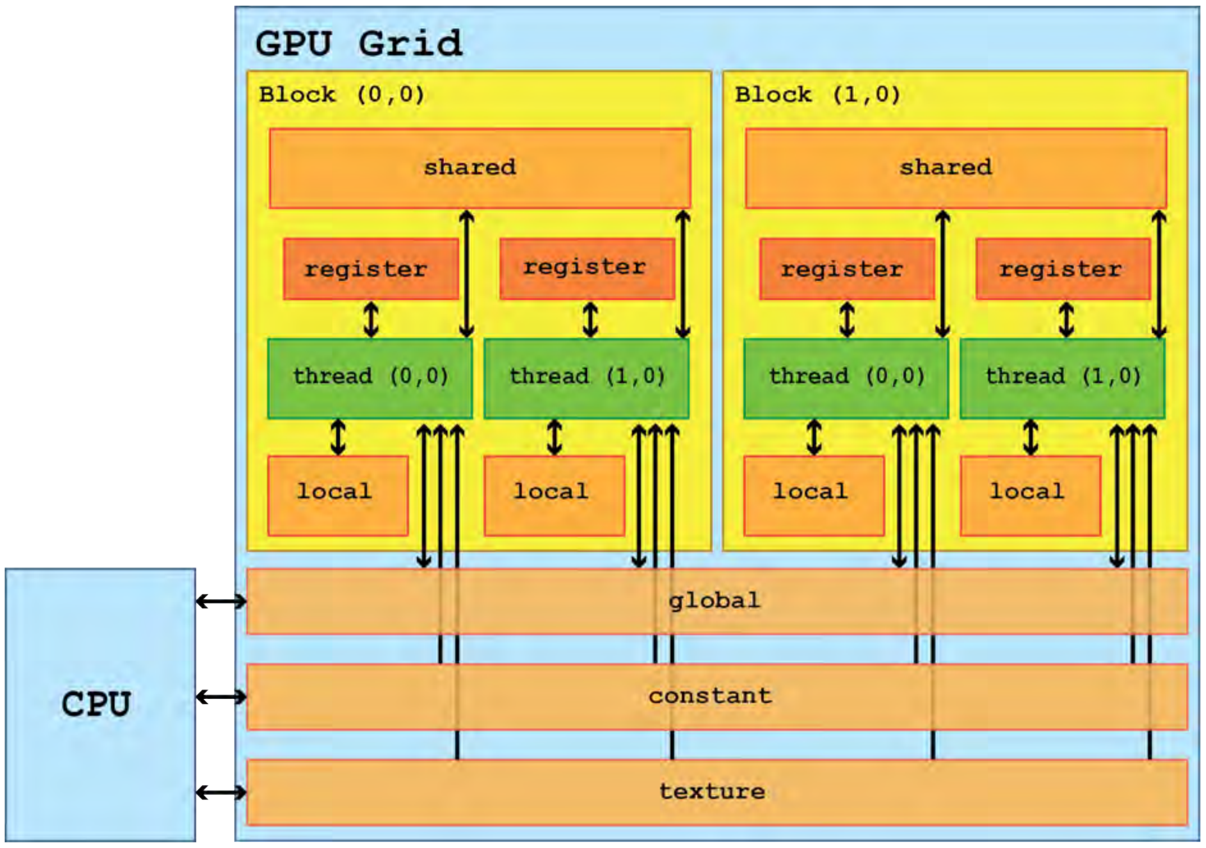


Figure 3.23: An illustration of a GPU memory hierarchy as exposed in the CUDA programming interface.

- `shared` memory is the second fastest memory type. It also is limited by the hardware specifications, for the `GeForce GTX 1060` its volume is `48 kB`. Its usefulness lies in the fact that it is accessible to all `threads` within a block. This can be used to achieve fast cooperation between the `threads`. Every `thread` can write to and read from the `shared` at any time point during the `block` execution. It has to be declared with the `__shared__` macro and its total volume can be specified both within the kernel source code (device code) and in the kernel call (host code). Its lifetime is that of the `block`.

- `global` memory is roughly equivalent to the working memory of a CPU and has by far the most volume (`6 GB` for the `GeForce GTX 1060`). Every `thread` and every `block` can access it at any time. It is the slowest of the discussed memory types. Its lifetime is that of the program. It has to be allocated, freed and initiated within host code using the special functions `cudaMalloc`, `cudaFree` and `cudaMemcpy`. The latter can copy memory ranges from and to host and device.

- `local` memory is used when the `register` or `shared` memory "spills over", e.g. when too big of an array is declared within kernel code. It is as slow as the `global` memory. Its use should be avoided.

- The `texture` memory is a special type of memory that excels, as the name suggests, in handling textures. Since one needs to deal with densities in structural integrative modeling, which can be interpreted as a 3 dimensional texture, it is highly useful. It comes with a number of conventions and will be discussed later on (see 3.3.3).

One use of the `shared` memory, as mentioned, is to facilitate the communication between `threads`. One might for example let each `thread` process its own data chunk and then use the result among all threads. A simple example is to calculate the sum of all squares in an array:

```
__global__
void square_kernel(int * d_input, int * d_ouput, int N){
        //declare shared memory array to sum up values
        __shared__ float sum[1024];
        //the index used to address the global memory
        int t = blockDim.x*blockIdx.x + threadIdx.x;
        //the thread id
        int tid = threadIdx.x;
        //initiate the shared memory to 0
        sum[tid] = 0.0f;
        //loop over the complete array, square and add to sum. Each thread
        //in each block calculates a number of squares and adds it to its
        //dedicated portion of the shared memory. the t variable increases
        //in the total size of the grid
        while (t < N){
                sum[tid] += d_input[t]*d_input[t];
                t += gridDim.x*blockDim.x;
        }
        //all threads need to be finished to sum up the values stored in
        //shared memory
        __synchtreads();
        //the number of threads adding up is halfed in each step,
        //effectively folding the shared memory onto itself
        if (tid < 1024) sum[tid] += sum[tid + 1024]; __synchtreads();
        if (tid < 512) sum[tid]  += sum[tid + 512];  __synchtreads();
        if (tid < 256) sum[tid]  += sum[tid + 256];  __synchtreads();
        if (tid < 128) sum[tid]  += sum[tid + 128];  __synchtreads();
        if (tid < 64) sum[tid]   += sum[tid + 64];   __synchtreads();
        if (tid < 32) sum[tid]   += sum[tid + 32];   __synchtreads();
        if (tid < 16) sum[tid]   += sum[tid + 16];   __synchtreads();
        if (tid < 8) sum[tid]    += sum[tid + 8];    __synchtreads();
        if (tid < 4) sum[tid]    += sum[tid + 4];    __synchtreads();
        if (tid < 2) sum[tid]    += sum[tid + 2];    __synchtreads();
        //only one thread needs to save the result to global memory
        if (tid == 0)
                d_out[0] = sum[0] + sum[1];
}
```

This code snippet highlights the use of `shared` memory. Also, it shows the important concept of *synchronization*. In this example, it is important that all `threads` have finished their iteration of the input data array before adding up the results. This can be done by calling the `__syncthreads()` function, which ensures that all `threads` “meet up” at this point before the execution code continuous. A lack of proper synchronization can result in a common error type, the so called *race condition*. This error class is quite dangerous since it does not break the program and might be invisible to the user. It occurs when the result of an operation defined within a kernel is dependent on the order of `thread` execution. Since, when `threads` are executed in parallel, there is no guarantee of execution order, this is the case quite frequent. The `__synchthreads()` canonical function then enables control of synchronization. A race condition type error will cause different results for the

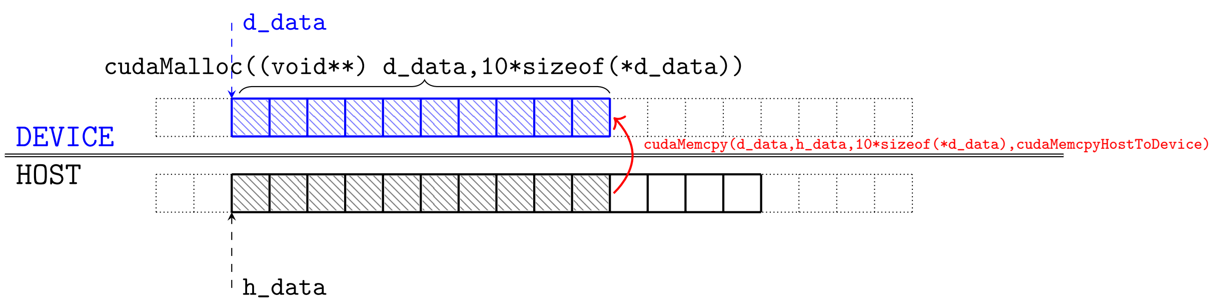

d_data
cudaMalloc((void**) d_data,10*sizeof(*d_data))
DEVICE
cudaMemcpy(d_data,h_data,10*sizeof(*d_data),cudaMemcpyHostToDevice)
HOST
h_data

Figure 3.24: Illustration of relevant function for memory management between device and host.

same computation between multiple runs of the program.

3.3.2 Communication between host and device

Communication between host and device serves two main purposes: The copying of data between host and device and the instructions that the device is to follow, which are issued from the host, except in the case of dynamic parallelism.
As indicated above, two functions are important for the allocation and copying of data: `cudaMalloc` to reserve a specified amount of memory at store the memory address `d_data` of the first element. `cudaMemcpy` can copy a specified amount of bytes from a host memory address `h_data` to a device memory address `d_data` or vice versa, depending whether the parameter `cudaMemcpyHostToDevice` or the parameter `cudaMemcpyDeviceToHost` was specified in the call. Figure (3.24) illustrates this process. It is to be noted that there is an asynchronous version of the memory copy operation, which does not block further instructions within the source code until is done. This can be used to simultaneously process and copy data, effectively hiding the time usage of the copy operation (*latency hiding*). The operation `cudaFree` can be used to free memory on the device.
The above operations are called from source code running on the host. They, by themselves, cannot issue instructions beyond memory management. This is, as mentioned above, achieved by calling so called `kernels`. The `kernels` have always return type `void`, their definition are annotated by the macro `__global__` and their call patterns consist of two distinct units: The `grid` specification enclosed in `<<<...>>>` and the function `parameters`.

```
kernel<<<gridDim,blockDim,sharedMemVolume,stream>>>
        (d_data1,d_data2,host_value);
```

The last two `grid` specifications are optional. The `sharedMemVolume` parameter can be used to dynamically allocate a volume of `__shared__` memory during runtime, within the `kernel` this memory must then be declared as e.g.

```
extern __shared__ float smem[];
```

This can for example be used to configure kernels to work on different GPUs, which have different `__shared__` memory volumes. The `stream` parameter can be used to specify a `cudaStream`, which is a technology that can help to synchronize processes on the host side. The `parameters` in the function argument will be implicitly copied to the device. They can be memory addresses pointing to device memory, or host variables. Passing a host memory address for example will be an error. Anything that can be considered input will fall into these two categories, anything that can be considered output has to be a device memory address. Both accessing device memory in host source code and vice versa will cause an error. `kernel` calls are non-block with regard to the host code. That means that the host code will continue executing and not wait for the `kernel` to finish its process. If the result of the calculation is needed directly after the `kernel` call, `cudaDeviceSynchronize()` should be called. This function will halt the host thread until all preceding device calls are finished.
Lastly, functions specified by the `__device__` macro can be called from kernels. They have access to the intrinsic variables of the kernel, `threadIdx,blockIdx,blockDim,gridDim`. A function can be specified both as `__device__` and `__host__`, in this case it can be called in both host and device code but does not have access to the intrinsic `kernel` variables.

3.3.3 Texture memory

`texture` memory encodes a ubiquitously used operation in the device hardware, which makes it computationally superior than a direct implementation. Figure (3.25) illustrates a reoccurring task in structural integrative modeling: Two densities are rotated and translated against each other. The dots (●) represent the centers of the three-dimensional pixels. Densities are supposed to model continuous objects, yet in a computational setting one has to render them as discrete pixels. The actual values of a density at a pixel are canonically interpreted being concentrated in the center of the pixel (see to 3.2.2 for further discussion). Translations and rotations are continuous operations and do not need to respect the grid underlying the density. The red dots are clearly not aligned to the blue dots. This raises the question: what value has a density between two pixels? `texture` memory provides an efficient tool to handle these situations. They intrinsically perform a linear, bilinear or trilinear interpolation of the pixel values, depending on the dimension of the `texture`. If one were to consider the value of a point directly between

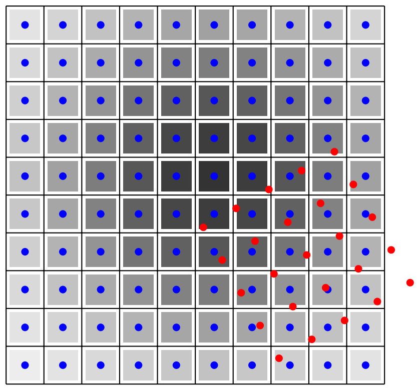


Figure 3.25: A density grid and its pixel centers(blue) and a rotated set of pixel centers(red).

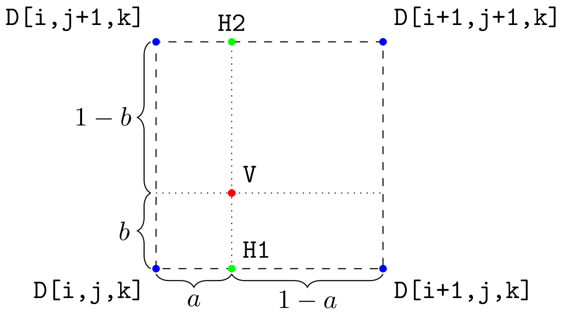


Figure 3.26: Sketch for two dimensional texture fetching.

two pixel centers, e.g. `(i,j,k)` and `(i+1,j,k)`, one would calculate the average:

$$D[i+0.5,j,k] = \frac{1}{2}\left(D[i,j,k] + D[i+1,j,k]\right)$$

If it were not halfway in between, but 20% of the way from `(i,j,k)` to `(i+1,j,k)`, one would simply change the weights in such a way that the closer pixel value contributes proportionally to $1 - \frac{\text{evaluation point distance to closest pixel}}{\text{distance between neighbouring pixels}}$. This way, a the calculated value right on top of a pixel center would be identical to the data pixel value.

$$\begin{aligned} D[i+0.2,j,k] &= \frac{4}{5}D[i,j,k] + \frac{1}{5}D[i+1,j,k] \\ &= \left(1-\frac{1}{5}\right)D[i,j,k] + \frac{1}{5}D[i+1,j,k] \\ &\text{or, more generally} \\ &= (1-a)\,D[i,j,k] + aD[i+1,j,k] \end{aligned}$$

with $a \in [0,1]$. The same kind of logic can be applied to two dimensional textures, with one intermediate step. Figure (3.26) provides a sketch for the construction. By first calculating the `texture` values at auxiliary points `H1` and `H2` using the above formula

$$H1 = (1-a) \cdot D[i,j,k] + a \cdot D[i+1,j,k]$$
$$H2 = (1-a) \cdot D[i,j+1,k] + a \cdot D[i+1,j+1,k]$$

and then applying it again to $H1$ and $H2$

$$\begin{aligned} V &= (1-b) \cdot H1 + b \cdot H2 \\ &= (1-b) \cdot ((1-a) \cdot D[i,j,k] + a \cdot D[i+1,j,k]) \\ &+ b\,((1-a) \cdot D[i,j+1,k] + a \cdot D[i+1,j+1,k]) \\ &= (1-a)(1-b) \cdot D[i,j,k] + a(1-b) \cdot D[i+1,j,k] \\ &+ (1-a)b \cdot D[i,j+1,k] + ab \cdot D[i+1,j+1,k] \end{aligned}$$

with $a, b \in [0,1]$. This is easily generalized to the third dimension, following analogous conventions:

$$\begin{aligned} V &= (1-a)(1-b)(1-c) \cdot D[i,j,k] + a(1-b)(1-c) \cdot D[i+1,j,k] \\ &+ (1-a)b(1-c) \cdot D[i,j+1,k] + (1-a)(1-b)c \cdot D[i,j,k+1] \\ &+ ab(1-c) \cdot D[i+1,j+1,k] + a(1-b)c \cdot D[i+1,j,k+1] \\ &+ (1-a)bc \cdot D[i,j+1,k+1] + abc \cdot D[i+1,j+1,k+1] \end{aligned}$$

The number of arithmetic operations in this expression, along with the fact that this has to be calculated for every single sample point (e.g. every single pixel center of a translated or rotated density), makes the use of a hardware accelerated solution such as the `texture` memory advantageous.
There are a number of features and caveats that are essential to the use of `texture` memory. To request a value from `texture` memory is referred to as *texture fetching*. `texture` memory enables the user to fetch up to 4 dimensional `float` data values, we will be interested in one dimensional data. Given the variable `density_texture`, one can request the concrete value like so

```
float v = tex3D<float>(density_texture, x, y, z);
```

where `x,y,z` are `float` instances that are bounded by the pixel dimensions of the density in question. If one wants to access the value at a specific pixel center of the pixel `i,j,k`, one calls

```
float v_pixel = tex3D<float>(density_texture,
                      (float) i + 0.5,  (float) j + 0.5, (float) k + 0.5);
```

following the convention that a pixels value is thought of residing in its geometric center. It can be useful to access the `texture` independent from the numerical value of the pixel dimensions of the density by first normalizing the values `x,y,z`:

$$\begin{aligned} x_n &= \frac{x}{\text{pixeldim.x}} \\ y_n &= \frac{y}{\text{pixeldim.y}} \\ z_n &= \frac{z}{\text{pixeldim.z}} \end{aligned}$$

with $x_n, y_n, z_n \in [0, 1]$. This has to be enabled be setting the `normalizedCoords` option and is useful in the case of scale free quadrature nodes, see **??**. The mentioned convention of pixel centers still holds. Beyond the calculation of density values within the density proper there are several options (called `addressMode`) of how to deal if one requests values that are outside of the density proper, such as $(1.4, 0.2, 0.2)$ in normalized coordinates. The most useful option for our purposes is the option `cudaAddressModeBorder`, which returns a predefined value. Since we intend to model densities this value will be set to 0.0. The discussed options represent a small sample of the possibilities, a mostly complete reference can be found at [67].
There are some idiosyncrasies of the `texture` memory that need to be taken into account and are not necessarily obvious. One of them is a swap of the y and z axes compared to the MRC file convention. The linear representation (see 3.2.2) of the three-dimensional volume in the MRC format *width, depth, height* needs to be converted the a representation of the convention *width, height, depth*. This can be achieved by mapping the linear address in one representation to the three-dimensional pixel representation following one convention, swap the indexes to change the convention, and then linearize this new pixel representation: `linear_index1` $\rightarrow$ `i,j,k` $\rightarrow$ `i,k,j` $\rightarrow$ `linear_index2`. A second important point is the inherent use of `pitched memory` in `texture memory`. `pitched memory` refers to a specific memory layout which aligns the memory to a specific pitch, which can be imagined as a register. Figure (3.27) illustrates the necessary memory layout. `pitched memory` requires more space, yet the pitch enables fast read access. This layout has to be taken into account when the density is copied to device memory. Failure to do so will result in a `texture` with does not represent the density anymore.

3.3.4 Atomic operations

The need for atomic operations arises in the context of parallel processing due to its non-sequential nature. The following example operation occurs often in practice:

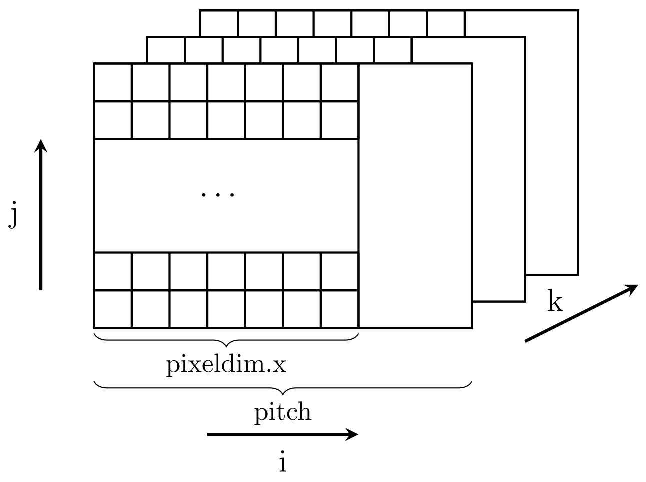
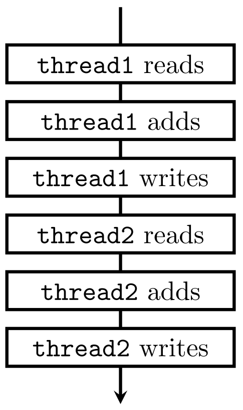
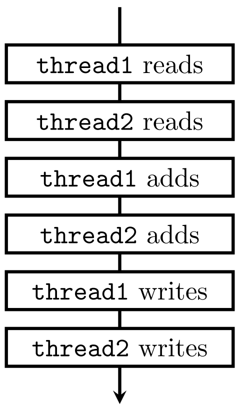


Figure 3.27: Visualization of the texture memory layout highlighting the role of the pitch.



(a) Possible process flow variant one.



(b) Possible process flow variant two.

Figure 3.28: Two different variants of a process execution flow.

- `thread1` reads the contents of `global` memory unit `array[220]`, which, for the sake of the argument, are assumed to be the integer `5`, adds some value `3` and writes the result back to `array[220]`, which should now contain `8`.

- `thread2` attempts something similar, reads from and writes to `array[220]` and in between adds the value `2`. After this operation the contents of `array[220]` should be `7`.

These operations might effect in different sequences of their constituent steps. Figure (3.28) shows two of these sequences. The result of the operation sequence described by figure (3.28a) will be `10`. The result of the sequence described by figure (3.28b) however, will be `7`, since `thread2` does not read after `thread1` has written its result. This is a typical case of a race condition.
Atomic operations provide a programming interface that internally sequentialize these operations so that the result is independent from the order of thread execution. They are rather simple in application and provided by the `CUDA` parallel programming interface.

The above example would be handled by

```
float * d_array; //pointer to device memory array
atomicAdd(d_array + 220, value0); //read out value at address 220, add value0,
                                  //write to address 220
```

Atomic operations are in principle possible for all binary operations on primitives, such as `float` or `int`, yet often only provided for `int`. They can however be extended to different data types.

3.3.5 Dynamic Parallelism

Dynamic parallelism describes the possibility of execution grids (see 3.3.1) specified by kernels to launch grids on their own, independent of the host. This can be useful if the parameters of `kernel 2` execution and therefore the size and input of the grid can change depending on the results of `kernel1`. This gives implementations of parallel processes to possibility to adapt their execution models to the properties of the data. An example is the use of adaptive spatial grid of higher degrees of refinement in regions of interest.

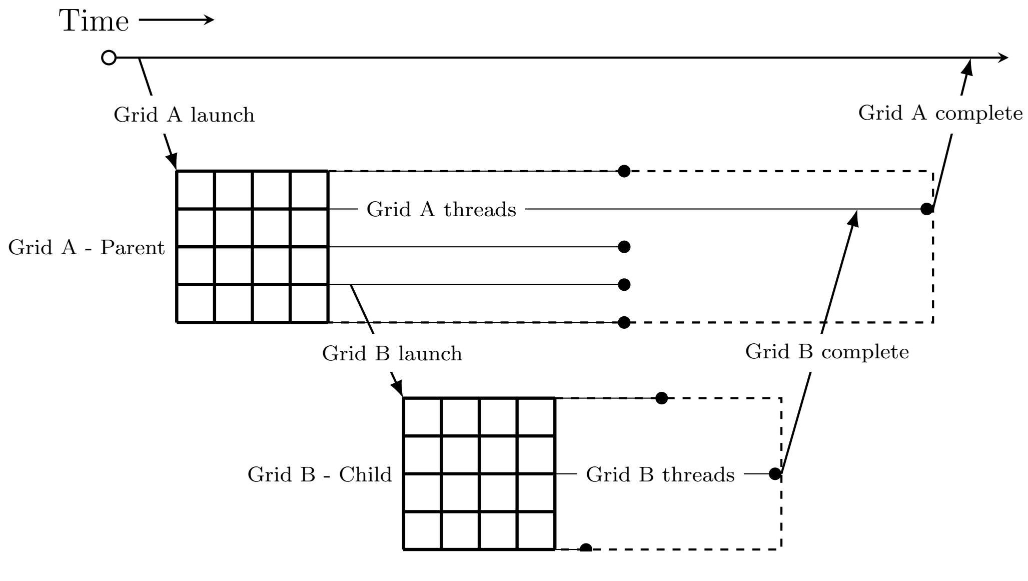


Figure 3.29: A schematic that illustrates dynamic parallelism within CUDA. Adapted from [68].

Figure (3.29) depicts a parent grid calling a child grid. The child grid always finishes before the parent grid finishes. Not every memory type can be passed to the child grid, however the view on `global` memory is consistent for parent and child grid. Kernels are launched the way described above, yet it is important to note that *every* `thread` will launch a grid if the kernel call is unguarded, which will almost certainly break the program flow. Hence dynamic kernel calls are guarded by an if-statement:

```
if (threadIdx.x == 0)
        kernel_call<<<gridDim,blockDim>>>(parameters ...);
```

Additionally special care must be taken in the case that the parent grid relies on the results of the child grid and a proper synchronization has to take place. The recommended idiom (see [68]) for this case is

```
__syncthreads();
if(threadIdx.x == 0)
        cudaDeviceSynchronize();
__syncthreads();
```

There are inherent limits to recursion depth depending on the device architecture.

3.3.6 Special operations

A number of operations occur repeatedly in varying problem contexts, such as `sort`,`find` and so on. Most of those have been implemented efficiently in standard sequential computational libraries. In parallel processing these operations often have a different, more efficient implementation. While there are cases where it is advisable to implement these operations using only the `CUDA` basic interface, this is often a less advantageous path due to economical and compatibility considerations. A solution is offered by NVIDIA in form of the `thrust` library (see [69]), which not only offers implementation of a number of common algorithms but also adapts to the specific GPU that is used. A brief introduction of the operations used later on will follow.
Every of the enumerated operations works on so called `device_vectors`, the `thrust` version of the standard `vector` data type. Contiguous ranges of memory can be cast using `thrust::device_point_cast`; that way a standard memory range can be used as an input. Often the algorithms require a unary or binary operation which has to be declared and defined before the call to the `thrust` library function is made.

```
struct sum_threshold : public thrust::binary_function<TDensity,TDensity,TDensity>
{
        TDensity threshold;
        __device__
        TDensity operator()(const TDensity& u, const TDensity& v) const {
        if (u < threshold && v < threshold)
                return (TDensity) 0.0;
        if (u < threshold)
                return v;
        if (v < threshold)
                return u;
        return u+v;
        }
};

Density_Register * instance = &_instance_registry[gpu_index];
thrust::device_ptr<TDensity> d_pixel_begin =
thrust::device_pointer_cast(&instance->d_data[0]);
thrust::device_ptr<TDensity> d_pixel_end =
thrust::device_pointer_cast(&instance->d_data[vol(*instance->h_pixel_dim)]);
sum_threshold binary_op;
binary_op.threshold = threshold;
h_result[0] = thrust::reduce(d_pixel_begin,d_pixel_end,0.0f,binary_op)
```

This code snippet exemplifies the important conventions and techniques. A `binary_function` is defined which takes two `TDensity` (mostly `float32`) data fields as input, checks if they exceed a certain threshold and returns the sum if both do, only the one that does or 0. Further down the data range between `instance->d_data` and `instance->d_data+pixel_vol` is cast into the appropriate form to be handed as input to a `reduce` call together with the aforementioned `binary_function`.

name	description
`reduce`	Reductions can be applied to a list of similar objects under a binary operations. Said binary operation accepts a pair of the objects and returns an object of the same type, which in turn will be one of the inputs for the next application of the binary operation. The sum of integers or the minimum of an array are examples. The results of reductions are independent from the order of the input sequence.
`transform`	Transforms a sequence of input elements and writes them to an output sequence, both may coincide. `transform` requires an `unary function`, which defines the specifics of the transformation as an additional input.
`transform_reduce`	Applies to a sequence first a `transform` operation specified by a `unary` function and then performs a `reduce` operation specified by a `binary` function.
`count_if`	Counts elements of a sequence that fulfill a `predicate`, which is a `unary` function that takes an element of said sequence as input and returns a boolean.
`fill`	Fills a memory range with a constant value.
`replace_if`	Replaces an element in a sequence with a constant value, if a given `predicate` is fulfilled.
`sort`	Sorts the elements given memory range. A `binary` operation which facilitates the comparison may be provided.
`unique_copy`	Copies the elements of a sequence to a specified destination, only copies the first element of consecutive identical elements.
`upper_bound`	For each element of a given sequence it returns the position where this element would be within a second sequence, specifically the last index where the element could be inserted without violating the order of the second sequence.

[70] contains exhaustive documentation.

Chapter 4

Mathematical methods

This section contains an introduction to most of the mathematical methods used in this thesis. When it it was possible, the topics have been arranged to lead into each other. At places where a graphical illustration would be a useful intuition for the reader, one was added. This section can be omitted by an informed reader and used as a mere reference.

4.1 Convergence

4.1.1 The notion of convergence

Convergence is an intuitively clear concept and commonly refers to two entities approaching each other increasingly. In the context of Monte Carlo procedures in integrative

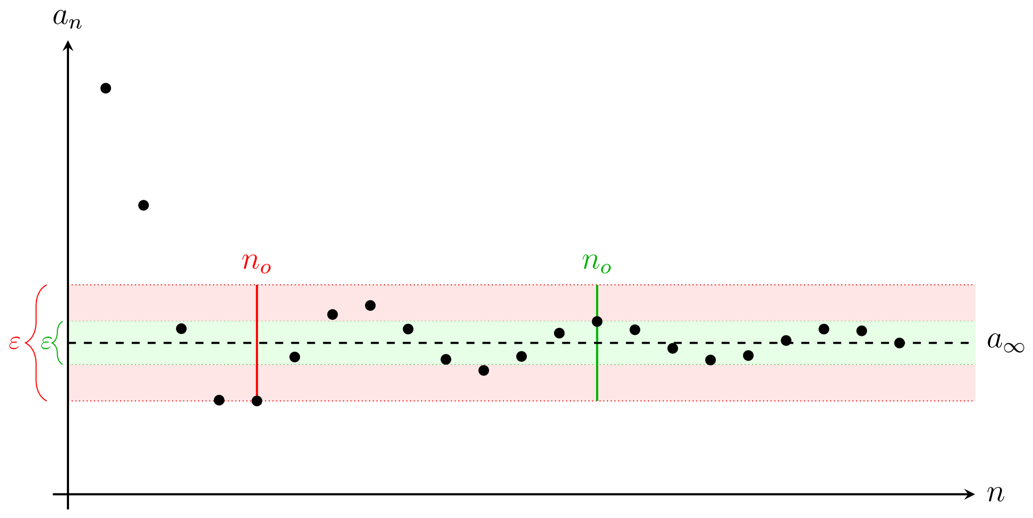


Figure 4.1: Illustration of ε-corridors of a converging series.

structural modeling it refers to a score sequence approaching a minimum. While convergence is easy to show it is harder to quantify computationally. The logically strict definition for a convergence of a series a_n ($\in \mathbb{R}$ indexed by n) to the value a_∞ is

$$\forall\ \varepsilon > 0\ \exists\ n_0 \in \mathbb{N}\ \forall\ n > n_0 \in \mathbb{N} : |a_n - a_\infty| < \varepsilon \tag{4.1}$$

This precise yet opaque definition is exemplified in figure (4.1). In plain words the definition says that for any ε-corridor (two examples in green and red) that is situated around the limiting value a_∞, there must be one n_0 at least, so that all following elements of the convergent sequence ($n > n_0$) are within the chosen ε-corridor. Since ε can be chosen as small as one likes, this test guarantees that the sequence gets "infinitely close" by first getting close (at n_0) and then never diverging anymore. This basic definition is impractical in a Monte-Carlo setting for two reasons:

- The limting value a_∞ is known in this definition, which in practical situations it never is.
- The criterium requires knowledge of infinitely many sequence values, which would correspond to an infinite runtime, which is impractical for obvious reasons.

4.1.2 The Cauchy convergence test

A test that addresses the first issue raised in the last section is the Cauchy convergence test. It is a modification of the standard test (4.1):

$$\forall\ \varepsilon > 0\ \exists\ n_0 \in \mathbb{N}\ \forall\ m, n > n_0 \in \mathbb{N} : |a_n - a_m| < \varepsilon \tag{4.2}$$

As it can be seen from the definition the test does not rely on knowledge of the limit a_∞. It can be understood most easily by considering the fact that the test requires that every possible partial sum after n_0 is bounded by ε:

$$|a_{n+1} + a_{n+2} + \ldots + a_{n+p}| < \varepsilon\ \forall\ n, p > n_0 \tag{4.3}$$

This ensures that the series will not escape an ε corridor, while is also does not state where exactly that corridor is. It is possible the construct a convergence test from the Cauchy test amenable to Monte-Carlo procedures by selecting a sequence of $\varepsilon^0, \varepsilon^1, \varepsilon^2, \ldots$ and finding the corresponding indexes $n_0^0, n_0^1, n_0^2, \ldots$, where from onwards the sample sequence a_n does not exceed the ε-corridors defined by the first sequence. If the n_0^i are all close and on average close to 0, one can speak of a fast convergence. If they are spread within the sample sequence a_n, one can assume a slow convergence that is still ongoing.

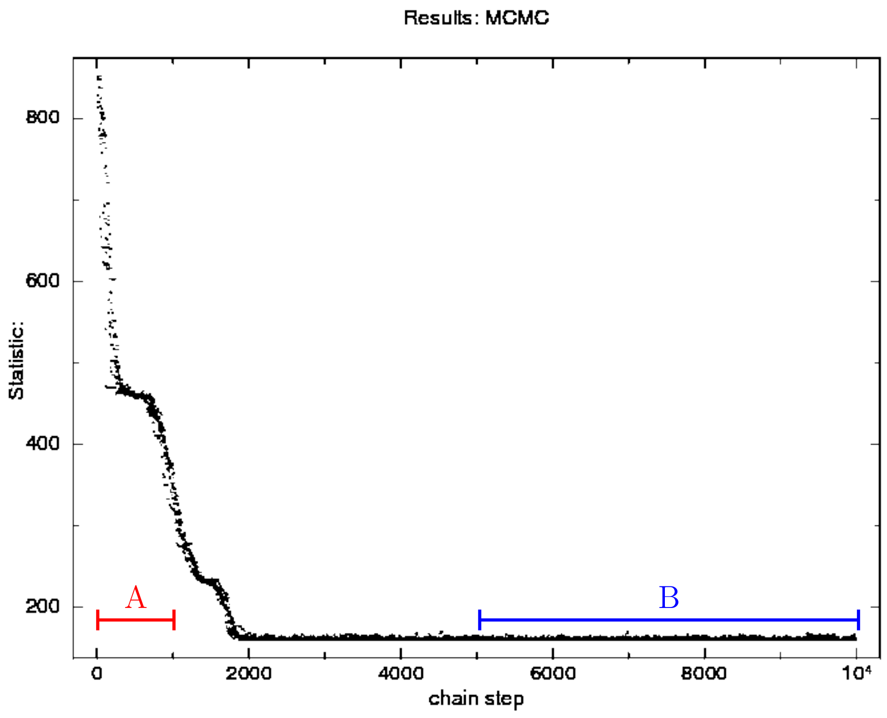


Figure 4.2: Example of a Monte-Carlo score sequence and illustration of burn-in. Taken from [71]. The typical ranges that are tested are shown as A and B.

This approach is computationally expensive since it requires exact calculations of all the differences $|a_m - a_n|$ and their respective sorting. It somewhat ignores the inherently statistical nature of a Monte-Carlo simulation. Another convergence test has to be adopted to accommodate for the practical needs of Monte-Carlo runs.

4.1.3 The Geweke convergence test

The Geweke test (see [44]) is designed to address issues that arise in a Monte-Carlo simulation. One of these issues is the so called “burn in” phase of a Monte-Carlo simulation, which describes a steep decline of the score function in a very short window in the initial phase of a run. Figure (4.2) shows an example a typical sequence. Another issue is the fact that even if a simulation runs for a very long time, the score will always exhibit some “thermal” fluctuations, and therefore is by definition 4.1 not converging, yet for all practical purposes is converging sufficiently to sample the posterior distribution. One would like to pass through the burn in phase and as little as possible of the thermally fluctuating tail. The Geweke convergence criterium measures the intuitive assumption that the *means* of different segments of Monte-Carlo walks should be close if the two segments are part of the longer tail. More to the point if two segments are part of the tail the distribution they sample is stationary and the difference of their means should be normally distributed for greater sample sizes of compared ranges. The statistic

$$z = \frac{\mu_A - \mu_B}{\sqrt{Var(s_A) - Var(s_B)}} \tag{4.4}$$

The variance needed to compare this statistic of two possible ranges, usually taken to be the first 10% of the run and the last 50% of the run (see figure 4.2) needs to be calculated on basis of the supposed spectral distribution estimation of the respective ranges. The rests on the assumption the ranges mirror a continuous time series with a non-singular value at 0. Then the variance can be estimated as

$$Var(s_A) = \frac{S_a(0)}{n} \tag{4.5}$$

For this however the spectral distributions value at 0 must be estimated estimated. The `pymc` package offers a solution (see [72]) for this problem. It measures in its standard configuration the difference in means according to 4.4 between 20 possible subintervals in the first 10% of a given sequence and the last 50% and checks if all values lie within the four standard deviations around 0 of $\mathcal{N}(0, 1)$. If this is so, the given sequence is judged to be convergent.

4.2 Vector spaces

Integrative structural modeling concerns itself with molecular structures that are modeled as a set of rigid bodies in a common, three-dimensional coordinate system. This invites the use of well-tested tools from the field of linear algebra. The features that are used depend on the context and utility they provide. I will give an overview of what will be used later on, omitting mathematical rigor when possible to favor completeness and intuitive appeal. The use of computers and the inherent uncertainty in model building also require methods from numerics and statistics.

$$V = \{\mathbf{v_0}, \mathbf{v_1}, \mathbf{v_2}, ...\}$$
$$S = \{s_0, s_1, s_2, ...\}$$

In the cases that concern us both sets have an infinite number of elements. If two vectors are added or a scalar is multiplied with a vector, the resulting vector is guaranteed to be an element of the vector space again.

$$\mathbf{v_0}, \mathbf{v_1} \in V \Rightarrow \mathbf{v_0} + \mathbf{v_1} \in V$$
$$\mathbf{v} \in V, s \in S \Rightarrow s\mathbf{v} \in V$$

The operations are denoted in the same way as the operations called multiplication and addition in the real numbers. Similar algebraical properties of both operations apply (distributivity, associativity, commutativity), making their use fairly intuitive. Addition is defined for elements V and the existence of a neutral additive element $\mathbf{0}$ ("zero-vector") which does not change a vector if added to it and an inverse additive element for each vector $-\mathbf{v}$ ("negative vector") the addition of which results in the zero-vector is guaranteed.

$$\mathbf{v} + \mathbf{0} = \mathbf{v}$$
$$\mathbf{v} + -\mathbf{v} = \mathbf{0}$$

This gives the set V the algebraical structure of a *commutative group*. The algebraical structure of the set of scalars S is that of a *field*, intuitively one can imagine them simply behaving like elements of the real numbers $\mathbb{R}$. This is but a short review, a more complete treatment can be found at [73]. While these definitions may seem cumbersome, their generality enables statements about a diverse set of objects. Consider, as a non-standard example of a vector space, the set of real-valued polynomials in the interval between $[-1, 1]$ as elements of V and the real numbers $\mathbb{R}$ as S:

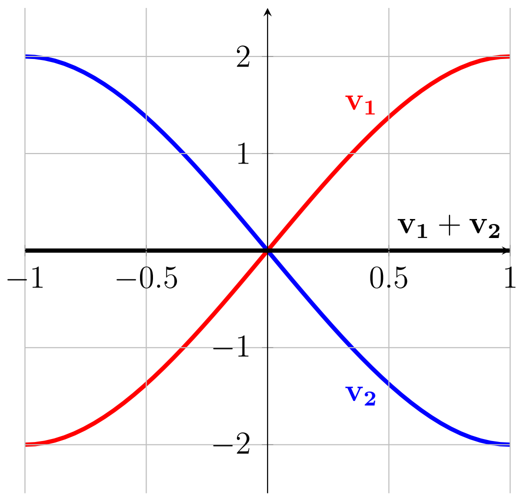


Figure 4.3: Vector addition.

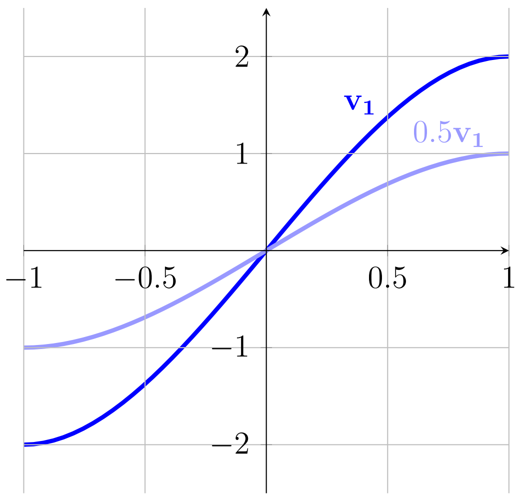


Figure 4.4: Scalar multiplication.

Figure 4.3 illustrates vector addition for this specific vector space. The red polynomial is the additive inverse of the blue polynomial, their addition resulting in the black additive neutral polynomial. Figure 4.4 shows an example of scalar multiplication. This vector space will be important for the purpose of approximating functions.

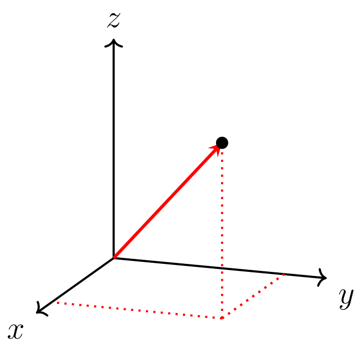


Figure 4.5: Visualization of a vector in the 3D real vector space. The black dot and the red arrow express the same point in space.

4.2.1 The three-dimensional real vector space

The most familiar example of vector spaces is the three-dimensional real vector space. It is also extensively used in structural integrative modeling, which warrants a more detailed treatment. It is also helpful to introduce some of the basic concepts in this vector space since it corresponds closely to the common geometric intuition. Figure 4.5 portrays the standard visualization method.

The basis and dimension of a vector space

A *basis* of a vector space is a subset of vectors which has the property that every other element of the vector space can be expressed a sum of scaled elements of this subset (a so called linear combination). There might be multiple different bases for the same vector space. From the above definition follows that every basis can be expressed in every other basis. Vector spaces can be quite complex and cumbersome. The concept of basis is powerful because it enables us to imagine a complex vector as a linear combination of simpler basis vectors. In the three-dimensional real vector space one would write:

$$\mathbf{v} = s_0\mathbf{v_0} + s_1\mathbf{v_1} + s_2\mathbf{v_2} = \sum_{i=0}^{2} s_i\mathbf{v_i}$$

It is apparent now what the word “dimension” means in relation to vector spaces: The dimension is the number of the elements of a basis. An additional property of the basis it is has to have the least possible number of elements within it, so that the notion of dimension is well defined. Figure 4.6 shows an example of a vector expressed in different bases.

The coefficients s_0, s_1 and s_2 of a linear combination with respect to a chosen basis are

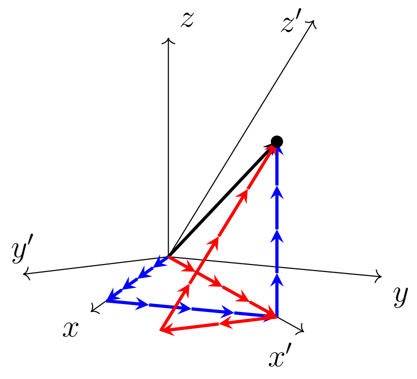


Figure 4.6: The same vector expressed in different bases.

often written in the form $\begin{pmatrix} s_0 \\ s_1 \\ s_2 \end{pmatrix}$ or $\begin{pmatrix} s_0 & s_1 & s_2 \end{pmatrix}^T$ (*tranposed*). The vector in figure 4.6 would be written as $\begin{pmatrix} 4 & 4 & 4 \end{pmatrix}^T$ and $\begin{pmatrix} 4 & 2 & 4 \end{pmatrix}^T$ in their respective bases. Notice that the same vector object can have different coefficients in different bases. It is therefore essential to always specify a basis when using this notation. It is referred to as *representation* of a vector with respect to a basis.

The scalar product

So far vectors could only be *added*. The *scalar product* introduces a kind of product of vectors whose result is a scalar. It enables us to make statements about "how much of a given vector is contained within another vector". In the three-dimensional real vector space, for $\mathbf{v_0} = \begin{pmatrix} s_0^0 & s_0^1 & s_0^2 \end{pmatrix}^T$ and $\mathbf{v_0} = \begin{pmatrix} s_1^0 & s_1^1 & s_1^2 \end{pmatrix}^T$ it is defined as

$$\langle \mathbf{v_0} | \mathbf{v_1} | = \rangle s_0^0 \cdot s_1^0 + s_0^1 \cdot s_1^1 + s_0^2 \cdot s_1^2$$

Important applications in three-dimensional space are the geometric notions of the "length" of a vector

$$\|\mathbf{v}\| = \sqrt{\langle \mathbf{v} | \mathbf{w} \rangle}$$

and the angle between two vectors

$$\measuredangle (\mathbf{v}, \mathbf{w}) = \arccos \frac{\langle \mathbf{v} | \mathbf{v} | \rangle}{\|\mathbf{v}\| \|\mathbf{w}\|}.$$

This "length" is called *norm* due to the fact that vectors can be quite abstract objects and it might not be clear what the "length" of a vector means. One important use case is the *normalization* of a vector, which scales the vector so that its norm is one:

$$\hat{v} = \frac{\mathbf{v}}{\|\mathbf{v}\|}$$

It is often referred to as *unit vector.* These concepts have a clear intuition and will be used accordingly in integrative structural modeling.
The scalar product has the important property that it is *linear* in both of its arguments, which means that

$$\begin{aligned}
\langle \mathbf{u} + \mathbf{v} | \mathbf{w} \rangle &= \langle \mathbf{u} | \mathbf{w} \rangle + \langle \mathbf{v} | \mathbf{w} \rangle \\
\langle \mathbf{u} | \mathbf{v} + \mathbf{w} \rangle &= \langle \mathbf{u} | \mathbf{v} \rangle + \langle \mathbf{u} | \mathbf{w} \rangle \\
\langle s\mathbf{v} | \mathbf{w} \rangle &= s\langle \mathbf{v} | \mathbf{w} \rangle \\
\langle \mathbf{v} | s\mathbf{w} \rangle &= s\langle \mathbf{v} | \mathbf{w} \rangle
\end{aligned}$$

A very useful application arises in conjunction with so called *orthonormal bases.* A basis $\mathcal{B} = \{\mathbf{b_0}, \mathbf{b_1}, \mathbf{b_2}, \ldots\}$ is called *orthonormal* if

$$\begin{aligned}
\langle \mathbf{b_i} | \mathbf{b_j} \rangle &= 0 \text{ when } i \neq j \\
\langle \mathbf{b_i} | \mathbf{b_j} \rangle &= 1 \text{ when } i = j
\end{aligned}$$

The significance of this definition becomes clear if one considers the scalar product of a vector $\mathbf{v}$ written as a linear combination and a basis vector $\mathbf{b_i}$ of an orthonormal basis:

$$\begin{aligned}
\langle \mathbf{v} | \mathbf{b_i} \rangle &= \langle c_0\mathbf{b_0} + c_1\mathbf{b_1} + c_2\mathbf{b_2} + \ldots + c_i\mathbf{b_i} + \ldots | \mathbf{b_i} \rangle \\
&= c_0 \underbrace{\langle \mathbf{b_0} | \mathbf{b_i} \rangle}_{=0} + c_1 \underbrace{\langle \mathbf{b_1} | \mathbf{b_i} \rangle}_{=0} + c_2 \underbrace{\langle \mathbf{b_2} | \mathbf{b_i} \rangle}_{=0} + \ldots + c_i \underbrace{\langle \mathbf{b_i} | \mathbf{b_i} \rangle}_{=1} + \ldots \quad \text{(linearity, orthonormality)} \\
&= c_i
\end{aligned}$$

Which means that using the scalar product one may calculate the coefficient of a representation with respect to an orthonormal basis, i.e. “disassemble” the vector into basis vectors.

$$\mathbf{v} = \sum_{i=1}^{d} \underbrace{\langle \mathbf{v} | \mathbf{b_i} \rangle}_{c_i} \mathbf{b_i} \tag{4.6}$$

d is the dimension of the corresponding vector space.

Linear maps in the context of the three-dimensional real vector space

Vector spaces come to life once linear maps between them are illustrated. The most important maps for our purposes in integrative structural modeling are rotations and translations. A *map* is simply a way to assign one object to another object. I will talk exclusively of these two objects as vectors within the same vector space. For a counterclockwise rotation by 90° around the z-axis in the standard basis of a vector $\mathbf{v} = \begin{pmatrix} 1 & 1 & 0 \end{pmatrix}^T$ for instance would simply be a map

$$\begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \overset{\mathcal{R}}{\rightarrow} \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix}$$

Mostly the notation takes the form of the following syntax, which can be thought of as "applied to".

$$\mathcal{R} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix}$$

An important property of the above mentioned maps is that they are *linear*. This means that one can interchange the application of the map with the operations of sums of vectors and multiplication by scalar factors, e.g.

$$\begin{aligned} \mathcal{R} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} &= \mathcal{R} \left(\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \right) = \mathcal{R} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + \mathcal{R} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix} \end{aligned}$$

This becomes significant if a vector is expanded in a basis $\mathcal{B}$ to which we apply the rotation $\mathcal{R}$:

$$\mathcal{R}\mathbf{v} = \mathcal{R} \sum_{i=1}^{d} c_i \mathbf{b_i} = \sum_{i=1}^{d} c_i \mathcal{R} \mathbf{b_i}$$

So the application of a linear map can be reduced to its application to the vectors of a given basis. Since the result is again a vector in the respective vector space it can be written as a linear combination of basis vectors. Returning to our above example and looking at its application to each vector of the so called *standard basis* of the three-dimensional real vector space we get the following result:

$$\begin{aligned}
\mathcal{R}\begin{pmatrix}1\\0\\0\end{pmatrix} &= \begin{pmatrix}0\\1\\0\end{pmatrix} &= 0\cdot\begin{pmatrix}1\\0\\0\end{pmatrix} + 1\cdot\begin{pmatrix}0\\1\\0\end{pmatrix} + 0\cdot\begin{pmatrix}0\\0\\1\end{pmatrix}\\
\mathcal{R}\begin{pmatrix}0\\1\\0\end{pmatrix} &= \begin{pmatrix}-1\\0\\0\end{pmatrix} &= -1\cdot\begin{pmatrix}1\\0\\0\end{pmatrix} + 0\cdot\begin{pmatrix}0\\1\\0\end{pmatrix} + 0\cdot\begin{pmatrix}0\\0\\1\end{pmatrix}\\
\mathcal{R}\begin{pmatrix}0\\0\\1\end{pmatrix} &= \begin{pmatrix}0\\0\\1\end{pmatrix} &= 0\cdot\begin{pmatrix}1\\0\\0\end{pmatrix} + 0\cdot\begin{pmatrix}0\\1\\0\end{pmatrix} + 1\cdot\begin{pmatrix}0\\0\\1\end{pmatrix}
\end{aligned} \tag{4.7}$$

The first = applies the rotation, the second rewrites the result in the standard basis. The coefficients therefore characterize the rotation completely with respect to the standard basis. They are usually written

$$M_{\mathcal{R}} = \begin{pmatrix}0 & 1 & 0\\ -1 & 0 & 0\\ 0 & 0 & 1\end{pmatrix} \tag{4.8}$$

The entries of this matrix have two indices and their are applied to the representation of a vector in the following manner

$$M_{\mathcal{R}}\mathbf{v} = \begin{pmatrix}m_{00} & m_{01} & m_{02}\\ m_{10} & m_{11} & m_{12}\\ m_{20} & m_{21} & m_{22}\end{pmatrix}\begin{pmatrix}v_0\\v_1\\v_2\end{pmatrix} = \begin{pmatrix}m_{00}v_0 + m_{01}v_1 + m_{02}v_2\\ m_{10}v_0 + m_{11}v_1 + m_{12}v_2\\ m_{20}v_0 + m_{21}v_1 + m_{22}v_2\end{pmatrix} \tag{4.9}$$

This is the effective computational form that I will use to compute rotations, translations are trivially achievable by simple vector addition. There are however different ways to represent rotations in three-dimensional space. They have different advantages and disadvantages, are used in varying practical computational contexts and will be discussed in 4.2.3. Linear maps and their representation in the form of matrices are a powerful tool with many applications. They will be discussed separately in the next section.

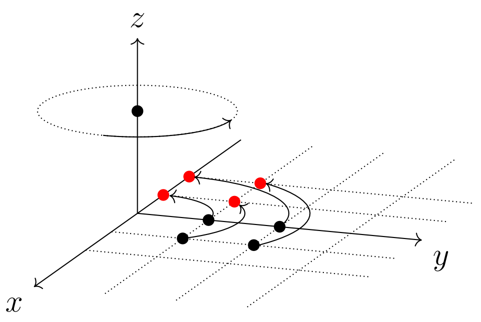


Figure 4.7: Illustration of a linear map, a counter clockwise rotation around the z-axis by 90 degree.

4.2.2 Linear maps between general vector spaces

Successive application of linear maps

It will be important to string different linear maps together, for instance to apply two rotations in succession. The successive application of two linear maps with matrix representations A, B can be thought of as another linear map with matrix representation C (see (4.10)). We only consider maps from vectors of a d-dimensional vector space into itself, so that our matrices will have d rows and d columns. The convention for denoting the position of the coefficients is $c_{n_{row}n_{column}}$

$$\begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1d} \\ c_{21} & c_{22} & \cdots & c_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ c_{d1} & c_{d2} & \cdots & c_{dd} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1d} \\ a_{21} & a_{22} & \cdots & a_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{d1} & a_{d2} & \cdots & a_{dd} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1d} \\ b_{21} & b_{22} & \cdots & b_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ b_{d1} & b_{d2} & \cdots & b_{dd} \end{pmatrix} \tag{4.10}$$

The components of C can be calculated by applying A and B to a dummy vector in the manner of (4.9) and rewriting the result in the form of (4.7). One ends up with the coefficients c_{ij} for C

$$c_{ij} = \sum_{k=0}^{d} a_{ik} b_{kj},$$

where a_{ik} and b_{kj} refer to the coefficients of A and B, respectively. This is the classic formula for matrix multiplication.

Eigenvalues and Eigenvectors

Eigenvalues and eigenvectors of linear maps are a way to characterize linear maps. Consider the matrix (4.8) and its action on several points illustrated in figure 4.7. The vectors (plotted as points) in the xy-plane are rotated around the z-axis by 90°. The vector on the z-axis itself is naturally unaffected, as are all vectors on the z-axis. In the same sense, the vectors in the xy-plane cannot "escape" the xy-plane. The z-axis and the xy-plane are eigenspaces in this example and they characterize the linear map up to a scalar factor (since I can multiply vectors on the z-axis with a number and they would still lie on the z-axis, similarly for the xy-plane) that is called the eigenvalue of the eigenspace. The vectors forming the basis of an eigenspace are called eigenvectors.
Eigenvectors and eigenvalues can be calculated by formulating their defining property as an equation:

$$\mathcal{M}\mathbf{v}_e = \lambda \mathbf{v}_e \tag{4.11}$$

In our example one eigenvector $\mathbf{v}_e$ would be $\begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^T$ and its corresponding $\lambda = 1$. One eigenvalue can have one or more associated eigenvectors. The eigenvectors can, under specific conditions to the linear map, form a basis and be considered a "natural" basis in which the action of the linear map looks particularly simple.

Linear maps as vectors

In some situations it can be beneficial to view linear maps themselves as elements of a vector space. Their corresponding matrix representations can then be multiplied by scalars and added to each other in the following way:

$$s \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1d} \\ a_{21} & a_{22} & \cdots & a_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{d1} & a_{d2} & \cdots & a_{dd} \end{pmatrix} = \begin{pmatrix} sa_{11} & sa_{12} & \cdots & sa_{1d} \\ sa_{21} & a_{22} & \cdots & sa_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ sa_{d1} & sa_{d2} & \cdots & sa_{dd} \end{pmatrix} \tag{4.12}$$

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1d} \\ a_{21} & a_{22} & \cdots & a_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{d1} & a_{d2} & \cdots & a_{dd} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1d} \\ b_{21} & b_{22} & \cdots & b_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ b_{d1} & b_{d2} & \cdots & b_{dd} \end{pmatrix} = \begin{pmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \cdots & a_{1d}+b_{1d} \\ a_{22}+b_{22} & a_{22}+b_{22} & \cdots & a_{2d}+b_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{d1}+b_{d1} & a_{d2}+b_{d2} & \cdots & a_{dd}+b_{dd} \end{pmatrix} \tag{4.13}$$

4.2.3 Representations of rotations

The axis-angle representation

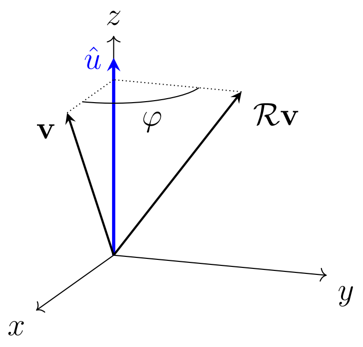


Figure 4.8: Illustration of the axis-angle convention.

The first representation that was chosen above is the intuitive axis-angle representation. It consists of the axis around which the rotation is carried out and the angle, which denotes the magnitude of the rotation (see figure 4.8). It has the advantage of a close correspondence to human visual intuition and is specified by a unit vector $\hat{u} = \begin{pmatrix} u_0 & u_1 & u_2 \end{pmatrix}^T$ and the angle φ. Its expression in matrix form is [74]

$$M_{\hat{u},\varphi} = \begin{bmatrix} \cos\varphi + u_0^2(1-\cos\varphi) & u_0u_1(1-\cos\varphi) - u_2\sin\varphi & u_0u_2(1-\cos\varphi) + u_1\sin\varphi \\ u_1u_0(1-\cos\varphi) + u_2\sin\varphi & \cos\varphi + u_1^2(1-\cos\varphi) & u_1u_2(1-\cos\varphi) - u_0\sin\varphi \\ u_2u_0(1-\cos\varphi) - u_1\sin\varphi & u_2u_1(1-\cos\varphi) + u_0\sin\varphi & \cos\varphi + u_2^2(1-\cos\varphi) \end{bmatrix}$$

The Euler angle representation

The Euler angles are a less intuitive but mathematically more practical method of specifying a rotation. Therefore they are often used to represent rotations in proofs and algorithms. Underlying the Euler angles is a procedure of successive rotations around certain axes. There are several conventions of choice of the sequence of axes. I employ the ZYZ convention, which is illustrated in figure 4.9. The important feature is that the complete coordinate system, illustrated by the axis, is rotated in each single rotation. The three successive rotations are: First, α around the original z-axis, second, β around the new y' axis and third, *gamma* around the "even newer" z''-axis. According to 4.2.3 we can write the resulting rotation as

$$\mathcal{R}(\alpha,\beta,\gamma)\mathbf{v} = \mathcal{R}_{\hat{z}'',\beta}\mathcal{R}_{\hat{y}',\beta}\mathcal{R}_{\hat{z},\alpha}\mathbf{v}$$

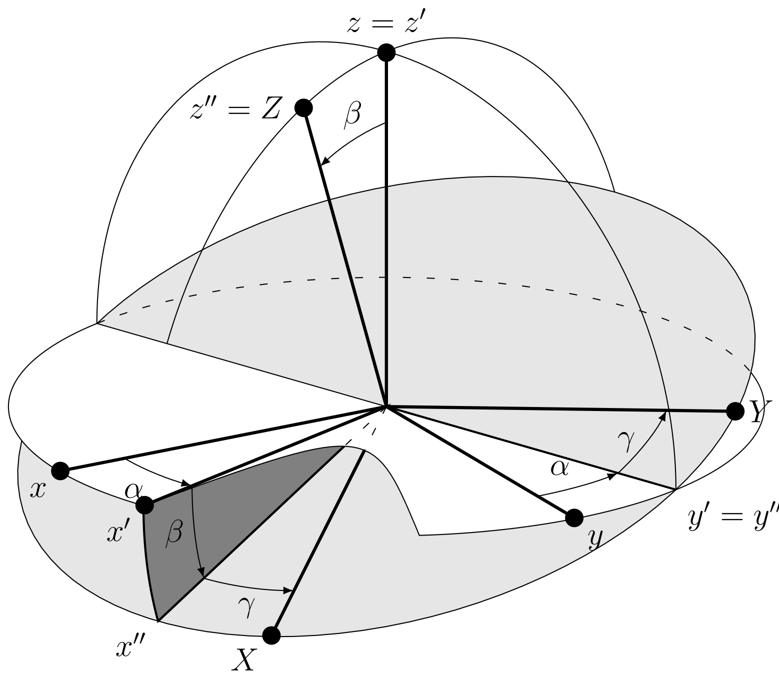


Figure 4.9: Illustration of the Euler angle procedure.

To achieve complete coverage, the angles are within the intervals $\alpha \in [0, 2\pi]$, $\beta \in [0, \pi]$ and $\gamma \in [0, 2\pi]$. By use of (4.7) one can determine the corresponding rotation matrix (in ZYZ convention). The result is

$$M_{\mathcal{R}(\alpha,\beta,\gamma)} = \begin{pmatrix} \cos\alpha\cos\beta\cos\gamma - \sin\alpha\sin\gamma & -\cos\gamma\sin\alpha - \cos\alpha\cos\beta\sin\gamma & \cos\alpha\sin\beta \\ \cos\alpha\sin\gamma + \cos\beta\cos\gamma\sin\alpha & \cos\alpha\cos\gamma - \cos\beta\sin\alpha\sin\gamma & \sin\alpha\sin\beta \\ -\cos\gamma\sin\beta & \sin\beta\sin\gamma & \cos\beta \end{pmatrix}$$

The advantages of Euler angles are apparent mostly in mathematical arguments and their property of successive application of local rotations, which promotes their use in computer graphics. Their chief disadvantage is the presence of singularities. The presence of singularities is of great practical importance in computation. The are a general feature of this description of rotations and must be carefully avoided, lest the computation bears nonsensical results. An example of a singularity can be visualized by considering the way we describe positions on the earth using longitude and latitude. While longitude ($\in [0, 360]$ - east/west) and latitude ($\in [-90, 90]$ - north/south) vary only slightly if one takes a small step in any direction near the equator, they vary considerable if one crosses the poles. The north pole itself has infinitely many coordinates, as long as the latitude is 0, any longitude can be added to it and it still describes the north pole. The same happens in three dimensions and becomes a problem since there is an inherent error in all computation the effect of which in conjunction with singularities can lead to massive numerical difficulties. The following representation of three-dimensional rotations does not have this problem.

The quaternion representation

The last form of representation discussed is that of quaternions. Quaternions are algebraical entities which are similar to complex numbers. Their properties make them uniquely convenient to describe rotations in three-dimensional space. They are widely used in computer graphics and molecular modeling. An excellent discussion of their properties and several applications can be found in [75]. Quaternions can be characterized by their numerical representation and the rules of addition and multiplication.

$$\boldsymbol{q} = q_0 + q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k}$$

with $q_0, q_1, q_2, q_3 \in \mathbb{R}$. The symbols $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$ have the property

$$\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = -1$$

When these symbols are treated like variables $(x, x^2, y, \ldots)$ in an algebraical expression the rules of addition and multiplication are obtained.

$$\begin{aligned}
\boldsymbol{p} + \boldsymbol{q} &= (p_0 + p_1\boldsymbol{i} + p_2\boldsymbol{j} + p_3\boldsymbol{k}) + (q_0 + q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k}) \\
&= (p_0 + q_0) + (p_1 + q_1)\,\boldsymbol{i} + (p_2 + q_2)\,\boldsymbol{j} + (p_3 + q_3)\,\boldsymbol{k} \\
\boldsymbol{pq} &= (p_0 + p_1\boldsymbol{i} + p_2\boldsymbol{j} + p_3\boldsymbol{k})\,(q_0 + q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k}) \\
&= (p_0q_0 - p_1q_1 - p_2q_2 - p_3q_3) \\
&+ (p_0q_1 + p_1q_0 + p_2q_3 - p_3q_2)\,\boldsymbol{i} \\
&+ (p_0q_2 - p_1q_3 + p_2q_0 + p_3q_1)\,\boldsymbol{j} \qquad (4.14)\\
&+ (p_0q_3 + p_1q_2 - p_2q_1 + p_3q_0)\,\boldsymbol{k} \qquad (4.15)
\end{aligned}$$

Again in analogy to complex numbers we define the conjugate $\overline{\boldsymbol{q}}$, the norm $|\boldsymbol{q}|^2$ and the multiplicative inverse $\boldsymbol{q}^{-1}$ of a quaternion.

$$\begin{aligned}
\overline{\boldsymbol{q}} &= q_0 - q_1\boldsymbol{i} - q_2\boldsymbol{j} - q_3\boldsymbol{k} \\
|\boldsymbol{q}|^2 = \boldsymbol{q}\overline{\boldsymbol{q}} &= q_0^2 + q_1^2 + q_2^2 + q_3^2 \\
\boldsymbol{q}^{-1} &= \frac{1}{|\boldsymbol{q}|}\boldsymbol{q}
\end{aligned}$$

Understood as vector a quaternion $\boldsymbol{q}$ can be written $[q_0, q_1, q_2, q_3]$. Consider the special quaternion $\boldsymbol{q} = [q_0, q_1, q_2, q_3] = [\sin\theta/2, v_0\cos\theta/2, v_1\cos\theta/2, v_2\cos\theta/2]$ with θ an angle and $\hat{v}$ a unit vector representation with respect to a basis. Then the operation

$$\boldsymbol{q}\,[0, x_0, x_1, x_2]\,\overline{\boldsymbol{q}}$$

is a linear map that maps $\mathbf{x} = \begin{pmatrix} x_0 & x_1 & x_2 \end{pmatrix}^T$ onto another vector. In accordance with 4.2.1 this operation can be written as a matrix.

$$M_{\hat{v},\theta} = \begin{pmatrix} 1-2(q_2^2+q_3^2) & 2(q_1q_2-q_3q_0) & 2(q_1q_3+q_2q_0) \\ 2(q_1q_2+q_3q_0) & 1-2(q_1^2+q_3^2) & 2(q_2q_3-q_1q_0) \\ 2(q_1q_3-q_2q_0) & 2(q_2q_3+q_1q_0) & 1-2(q_1^2+q_2^2) \end{pmatrix} \tag{4.16}$$

This matrix is identical with a matrix that represents the rotation around a vector $\hat{v}$ by θ yet its only "ingredients" are the coefficients of $\boldsymbol{q}$. Therefore $\boldsymbol{q}$ corresponds to a rotation operation. Successively executed rotations corresponding to the quaternions $\boldsymbol{p}, \boldsymbol{q}$ are represented by their quaternion product (see (4.14)) $\boldsymbol{pq}$. The special quaternion $\boldsymbol{q}$ defined above is also a unit-quaternion, since

$$\begin{aligned} |\boldsymbol{q}|^2 &= \sin^2\theta + v_0^2\cos^2\theta + v_1^2\cos^2\theta + v_2^2\cos^2\theta \\ &= \sin^2\theta + \cos^2\theta\ \underbrace{\left(v_0^2+v_1^2+v_2^2\right)}_{=1,\ \hat{v}\text{ is a unit vector}} \\ &= \sin^2\theta + \cos^2\theta = 1 \end{aligned}$$

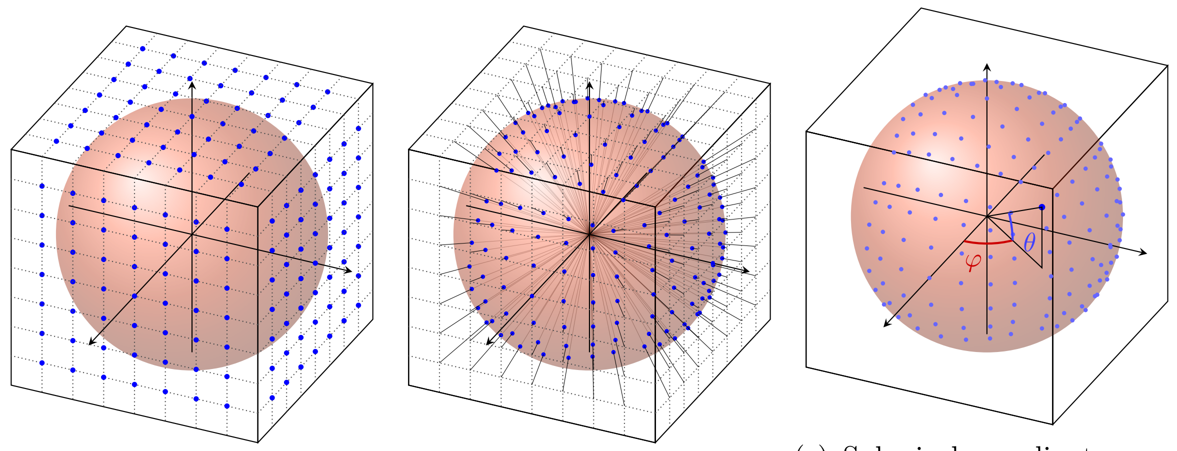
All unit quaternions correspond to rotations in this manner. There is one caveat: $\boldsymbol{q}$ and $-\boldsymbol{q}$ correspond to the same rotation.
The advantages of quaternions are their compact representation (only 4 real numbers), their independence of a specific basis (one does not need to represent them as matrices to chain rotations, only multiply them) and their freedom of singularities.

Construction of rotation grids using quaternions

The quaternion representation lends itself to an easy construction of rotational search grids, which will be illustrated following [75]. The principle of the construction is illustrated in figure (4.10). The construction can be divided into three steps. First, points on the faces of the cube enclosing the sphere are selected to define a grid (4.10a). This can be any given set of points. Then the points are projected onto the unit sphere by the operation:

$$\vec{p} = \begin{pmatrix} p_0 \\ p_1 \\ p_2 \end{pmatrix} \longrightarrow \begin{pmatrix} \frac{p_0}{\sqrt{p_0^2+p_1^2+p_2^2}} \\ \frac{p_1}{\sqrt{p_0^2+p_1^2+p_2^2}} \\ \frac{p_2}{\sqrt{p_0^2+p_1^2+p_2^2}} \end{pmatrix}. \tag{4.17}$$

Now all grid points lie on the unit sphere and can be identified with a set of spherical coordinates (φ, θ).



(a) A grid on the faces of a cube around the unit sphere. (b) The grid projected on the inscribed unit sphere (c) Spherical coordinates correspond to the projected grid points.

Figure 4.10: Illustration of principle behind the rotation grid construction.

This operation can be carried out in 4-dimensional space in an analogous manner. The cube is replaced by a 4-dimensional cube, a tesseract. The faces of the tesseract are cubes, which correspond to the square faces of the cube in three dimensions. In total, eight cubes are needed to “cover” the three-sphere S^3, which is the analogue of the two-sphere S^2 used in the illustration. The points on S^3 can be projected onto unit-quaternions, which, in turn, can be identified with rotations of rigid bodies in three dimensional space. Of the eight cubes, only four cubes are necessary because the unit-quaternions $\mathbf{q}$ and $-\mathbf{q}$ represent the same rotation. The four cubes can be defined by:

$$\{(1, p_1, p_2, p_3) \mid |p_1|, |p_2|, |p_3| \leq 1\} \tag{4.18}$$

$$\{(p_1, 1, p_2, p_3) \mid |p_0|, |p_2|, |p_3| \leq 1\} \tag{4.19}$$

$$\{(p_1, p_2, 1, p_3) \mid |p_0|, |p_1|, |p_3| \leq 1\} \tag{4.20}$$

$$\{(p_1, p_2, p_3, 1) \mid |p_0|, |p_1|, |p_2| \leq 1\} \tag{4.21}$$

Defining grids within these cubes and converting them to unit quaternions offers a computationally simple and efficient way to “move” within the space of rotations.

4.2.4 Function vector spaces

In integrative structural modeling one relates the position and orientation of experimentally obtained density maps to each other and computes metrics that are supposed to express aspects of this relation. The complexity of experimental data makes it impractical to cover every possible orientation and position, the concrete gains for model building of such an approach are questionable. In effect one makes a select number of “measure-

ments" and deems this representation sufficient.
It is often beneficial to have a smooth and differentiable representation of an object of study. The procedure is to take the above discrete measurements and try to "construct" a function out of a set of basis functions. In perfect analogy with the concepts outlined above these constructions are elements of function vector spaces. A well known example of these vector spaces is the application within the context of a Fourier transform. A given signal s is disassembled into a set of basic functions $f_1, f_2, \ldots$, where the meaning of the word "disassembled" is to be understood in the sense of a linear combination of the basis functions:

$$s \approx \sum_i c_i \cdot f_i \tag{4.22}$$

with c_i typically $\in \mathbb{R}$ or $\in \mathbb{C}$. Also, for the sake of readability the variable is omitted, meaning for instance $s(x) = s$. The set of basis functions is not arbitrary. In the case of a discrete Fourier analysis for example, it can a set of different cos-functions:

$$\{\cos(\omega_1 \cdot x + c_1), \cos(\omega_2 \cdot x + c_2), \cos(\omega_3 \cdot x + c_3), \ldots\} \tag{4.23}$$

which amounts to cos-functions with different wavelengths a phases (geometrically, shifts along the x-axis). The actual Fourier transform is merely the set of coefficients c_i with regard to that basis, mirroring the well known fact that "pure" signals have a "spiky" Fourier transform, because they are well approximated by just one or a few of the cos-basis functions.
These basis function sets can potentially be infinite, and often they are. In praxis however, our data is finite and its representation is finite too. This means that we must content ourselves with just a number n of the basis functions. Almost invariably a higher number of basis functions means a higher degree of faithfulness in representation. Due to the finite nature of our data it is also the case that *a certain size of the basis function set is actually sufficient to represent the data faithfully.* In this case a signal s might be seen as having several different but equivalent representations. If there are several ways to express one and the same thing, some of these ways might be more economical or convenient than others. From this fact comes the use of function vector spaces in computational procedures. The next section contains both an example and a discussion of the so called Chebyshev polynomials, which constitute one possible set of basis functions.

Chebyshev polynomials as an example of polynomial approximation

The Chebyshev polynomials can be defined iteratively:

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ &\cdots \\ T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x) \end{aligned}$$

So the first 5 basis polynomials are:

$$\{1, x, 2x^2 - 1, 4x^3 - 3x, 8x^4 - 8x^2 + 1\}$$

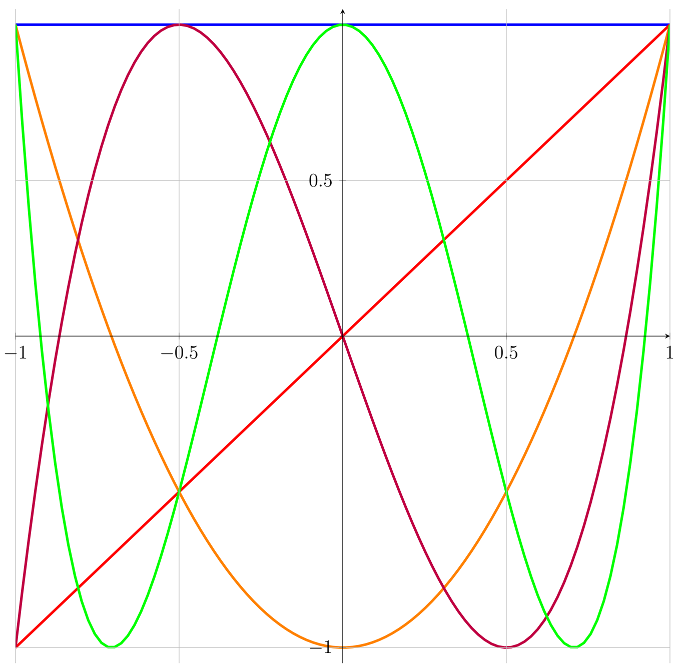


Figure 4.11: The graphs of the first 5 Chebyshev polynomials

As the plots in figure (4.11) show, the Chebyshev polynomials have a somewhat regular appearance on the interval $[-1, 1]$. This is due to their construction, they are understood to be considered only on this finite interval. The fact that they indeed constitute a basis can be proven easily, but first their scalar product must be defined:

$$\langle p(x)|q(x)\rangle = \int_{-1}^{1} p(x)q(x)\frac{1}{\sqrt{1-x^2}}\,\mathrm{d}x \tag{4.24}$$

which, as already indicated, is a measure of "how much of one vector is contained in another". Since basis vectors have the property that with regards to a scalar product none of them is contained in any of the others, the scalar product between two Chebyshev polynomials must be 0. Just as

$$\left\langle \begin{pmatrix}1\\0\\0\end{pmatrix} \middle| \begin{pmatrix}0\\1\\0\end{pmatrix} \right\rangle = 1\cdot 0 + 0\cdot 1 + 0\cdot 0 = 0;$$

is 0 because the two vectors are part of a basis with regard to the standard scalar product of the three-dimensional real vector space, the scalar product

$$\langle T_1(x)|T_2(x)\rangle = \int_{-1}^{1} x\left(2x^2-1\right)\frac{1}{\sqrt{1-x^2}}\,\mathrm{d}x = 0$$

is 0 (on account of the symmetry of the integrand, which is uneven) because $T_1(x)$ and $T_2(x)$ are orthogonal with respect to the scalar product defined in 4.24. This property is proven for the Chebyshev polynomials, one does not need to concern oneself with the details of the calculation and can just use the formalism to engage in standard operation within a vector space. As an illustrative application, consider the function $\sin(\pi x)$ on the interval $[-\pi,\pi]$. In the same way as one would calculate the contribution of each basis vector to an arbitrary vector in the more familiar three-dimensional real vector space, one can do so in a function vector space:

$$\begin{aligned}
c_0 &= \frac{\langle \sin(\pi x)|T_0(x)\rangle}{\langle T_0(x))|T_0(x)\rangle} = \frac{\int_{-1}^{1}\sin(\pi x)\cdot 1\frac{1}{\sqrt{1-x^2}}\,\mathrm{d}x}{\int_{-1}^{1}1\cdot 1\frac{1}{\sqrt{1-x^2}}\,\mathrm{d}x} = && 0\\
c_1 &= \frac{\langle \sin(\pi x)|T_1(x)\rangle}{\langle T_1(x))|T_1(x)\rangle} = \frac{\int_{-1}^{1}\sin(\pi x)\cdot x\frac{1}{\sqrt{1-x^2}}\,\mathrm{d}x}{\int_{-1}^{1}x\cdot x\frac{1}{\sqrt{1-x^2}}\,\mathrm{d}x} = && 0.5692306863596991\\
c_2 &= \frac{\langle \sin(\pi x)|T_2(x)\rangle}{\langle T_2(x))|T_2(x)\rangle} = \frac{\int_{-1}^{1}\sin(\pi x)\cdot(2x^2-1)\frac{1}{\sqrt{1-x^2}}\,\mathrm{d}x}{\int_{-1}^{1}(2x^2-1)\cdot(2x^2-1)\frac{1}{\sqrt{1-x^2}}\,\mathrm{d}x} = && 0\\
c_3 &= \frac{\langle \sin(\pi x)|T_3(x)\rangle}{\langle T_3(x))|T_3(x)\rangle} = \frac{\int_{-1}^{1}\sin(\pi x)\cdot(4x^3-3x)\frac{1}{\sqrt{1-x^2}}\,\mathrm{d}x}{\int_{-1}^{1}(4x^3-3x)\cdot(4x^3-3x)\frac{1}{\sqrt{1-x^2}}\,\mathrm{d}x} = && -0.6669166724068839
\end{aligned}$$

with $c_i = \frac{\langle v|b_i\rangle}{\langle b_i|b_i\rangle}$ being the formula to calculate basis coefficients for non-normalized basis vectors b_i. Therefore, given that basis functions above, the approximation is

$$\begin{aligned}
a(x) = {} & 0\cdot 1\\
& + 0.5692306863596991\cdot x\\
& + 0\cdot\left(2x^2-1\right)\\
& + (-0.6669166724068839)\cdot(4x^3-3x)
\end{aligned}$$

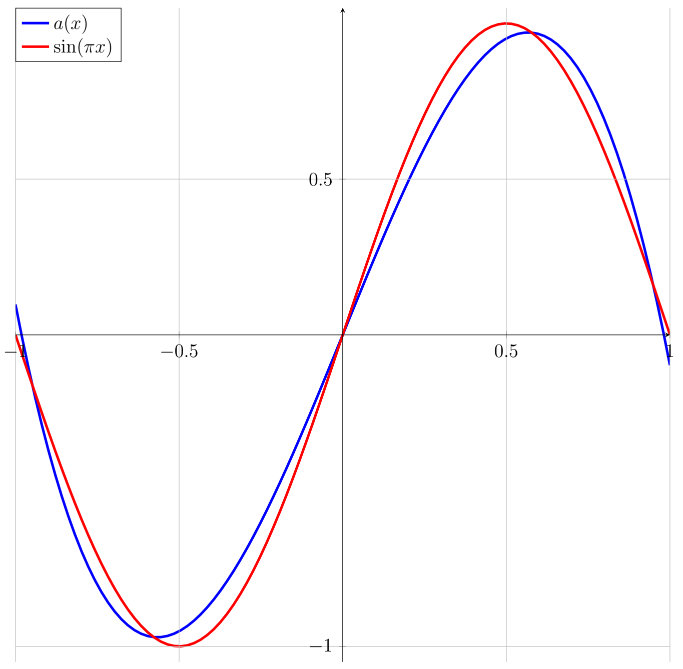


Figure 4.12: The function $\sin(\pi x)$ approximated by the first 5 Chebyshev polynomials.

Figure (4.12) shows this approximation. If a function should be approximated finite interval other than $[-1, 1]$, it can simply be linearly translated and scaled to fit this interval, be approximated and apply the inverse transformation to the approximation.

This pattern in general will be the same for all coming function approximations, the spaces wherein the approximation is performed will be more complex than the real line $\mathbb{R}$, but the concepts and the formalism will remain the same. A set of basis functions and a scalar product will still be the central tools.

The integral in the scalar product (4.24) is expensive to calculate. The Chebyshev polynomials exhibit another orthogonality condition with regard to their roots, which comes in handy for approximation purposes. This condition allows to interpolate a function $f(x)$ by evaluating them at a very limited number of points (the single prime indicates that the first term of the sum is halved):

$$f(x) \approx \sum_{i=1}^{n}{}' c_i T_i(x) \tag{4.25}$$

$$c_i = \frac{2}{n+1} \sum_{k=1}^{n+1} f(x_k) T_i(x_k) \tag{4.26}$$

where the x_k are the zeros of $T_{n+1}(x)$, namely

$$x_k = \cos\left(\frac{2k-1}{2n}\pi\right), k = 1, \ldots, n+1$$

This shows the most attractive feature of polynomial approximation of this kind: The

number function evaluations is fairly limited and the result is still a decent approximation, which also has the desirable properties of polynomials. Especially if the evaluation of $f(x)$ is very expensive this can be very advantageous. A exhaustive discussion of Chebyshev polynomials themselves can be found in [76].
Often, one needs to model higher dimensional spaces, such as functions defined on an euclidean plane, $\mathbb{R}^2$ or in euclidean space $\mathbb{R}^3$. In this case one can easily construct three-dimensional representations using the concepts and entities outlined above. For completeness' sake the procedure will be detailed verbosely, it can be generalized to any dimension:

1. The domain of the function to be approximated is $[a_x, b_x] \times [a_y, b_y] \times [a_z, b_z]$. The order of polynomial approximation is n_x, n_y, n_z, respectively.

2. $m_x \leq n_x + 1$,$m_y \leq n_y + 1$ and $m_z \leq n_z + 1$ are the numbers of the interpolation nodes. The concrete nodes of evaluation are

$$r_k^x = \cos\left(\frac{2k-1}{2m_x}\pi\right), k = 1 \dots m_x$$
$$r_l^y = \cos\left(\frac{2l-1}{2m_y}\pi\right), l = 1 \dots m_y$$
$$r_m^z = \cos\left(\frac{2k-1}{2m_z}\pi\right), m = 1 \dots m_z$$

3. The evaluation nodes adjusted to the domain defined above are

$$x_k = a_x + (1 + r_k^x)\left(\frac{b_x - a_x}{2}\right), k = 1 \dots m_x$$
$$y_l = a_y + (1 + r_l^y)\left(\frac{b_y - a_y}{2}\right), l = 1 \dots m_y$$
$$z_m = a_z + (1 + r_m^z)\left(\frac{b_z - a_z}{2}\right), m = 1 \dots m_z$$

4. The function $f(x, y, z)$ to be approximated has to be evaluated at these nodes

$$\Omega = \{\omega_{klm} = f(x_k, y_l, z_m) | k = 1 \dots m_x, l = 1 \dots m_y, m = 1 \dots m_z\}$$

5. Now, calculate the coefficients α_{ijk} with $i = 0 \dots n_x, j = 0 \dots n_y, k = 0 \dots n_z$:

$$\alpha_{ijk} = \frac{\sum_{k=1}^{m_x} \sum_{l=1}^{m_y} \sum_{m=1}^{m_z} \omega_{klm} T_i^x\left(r_k^x\right) T_j^y\left(r_l^y\right) T_k^z\left(r_m^z\right)}{\left(\sum_{k=1}^{m_x} T_i^x\left(r_k^x\right)^2\right)\left(\sum_{l=1}^{m_y} T_j^y\left(r_l^y\right)^2\right)\left(\sum_{k=1}^{m_z} T_k^z\left(r_m^z\right)^2\right)}$$

6. Finally, the approximated function $a(x, y, z)$ is

$$a(x,y,z) = \sum_{i=0}^{n_x}{}' \sum_{i=0}^{n_y}{}' \sum_{i=0}^{n_z}{}' \alpha_{ijk} T_i^x \left(2\frac{x-a_x}{b_x-a_x} - 1\right) T_j^y \left(2\frac{y-a_y}{b_y-a_y} - 1\right) T_k^z \left(2\frac{z-a_z}{b_z-a_z} - 1\right)$$

Quadrature via Chebyshev polynomials

Another important application of Chebyshev polynomials and a central concept in its own right is that of *quadrature*. Quadrature, historically, is a way to calculate the area of a square that has the same area as a given geometric shape. In modern parlance it is a way to approximate the value of an integral

$$I = \int_D f(x)\,\mathrm{d}x$$

over some domain D. Its importance in science is paramount. Many quantifiable processes in nature are imagined to be continuous, yet one usually has only a discrete sample of the quantity characterizing the process. And often one wishes to calculate characteristics such as a mean or the integral value itself. Since the functional form of $f(x)$ is often unknown, the rules of numerical quadrature are required to accomplish these goals. Quadrature rules are expressed by formulae of the type

$$I \approx \sum_i w_i f(x_i) \tag{4.27}$$

where the x_i and the w_i are the quadrature nodes and the quadrature weights respectively. Nodes and weights are intimately connected and need to be specified together. There is a wealth of different quadrature methods, some more efficient than others and many more suitable to some cases of $f(x)$ than others. The geometry of the underlying domain of integration plays a pivotal role, for the standard euclidean spaces the theory of Chebyshev polynomials provides a practical set of nodes and weights. The underlying principle is to approximate the function $f(x)$ on the integration domain using a basis of Chebyshev polynomials as detailed above and then carry out the integration of that approximation, which is a simple operation on polynomials. Consequently the quality of quadrature improves with the quality of approximation, which in turn is enhanced by a greater number of nodes. In praxis, the approximation is done “under the hood” and the nodes and weights can be precalculated. A popular method is that of Clenshaw and Curtis (see

[77]), the nodes and weights for sample size $N+1$ are

$$x_i = \cos\left(\frac{i}{N}\pi\right), i = 0\dots N \tag{4.28}$$

$$w_i = \begin{cases} \frac{2}{N}\sum_{j=0}^{\frac{N}{2}}{}'' \frac{1}{1-4j^2}\cos\frac{2ji\pi}{N} & 0 < i < N \\ \frac{4}{N}\sum_{j=0}^{\frac{N}{2}}{}'' \frac{1}{1-4j^2}\cos\frac{2ji\pi}{N} & \text{otherwise} \end{cases}$$

where the double primed sum $\sum''$ indicates that the first and the last term are to be halved.

An advantageous fact about this specific set of nodes and weights is that, if N is doubled $2N-1$, a the nodes x_i associated with N will be contained within the new set of nodes associated with $2N$. This is computationally opportune, since it allows one to double the precision without recalculating all evaluations but reuse the old ones. Figure (4.13) illustrates this feature. The result of the illustrative application using two different sets of nodes ($N_1 = 8$ and $N_2 = 15$) is

$$I_1 = \pi\sum_{i=0}^{7} w_i \sin x_i = 1.9999997146306663$$

$$I_2 = \pi\sum_{i=0}^{15} w_i \sin x_i = 1.9999999217203575$$

with the exact result $\int_0^\pi \sin(x)\,\mathrm{d}x = 2$.

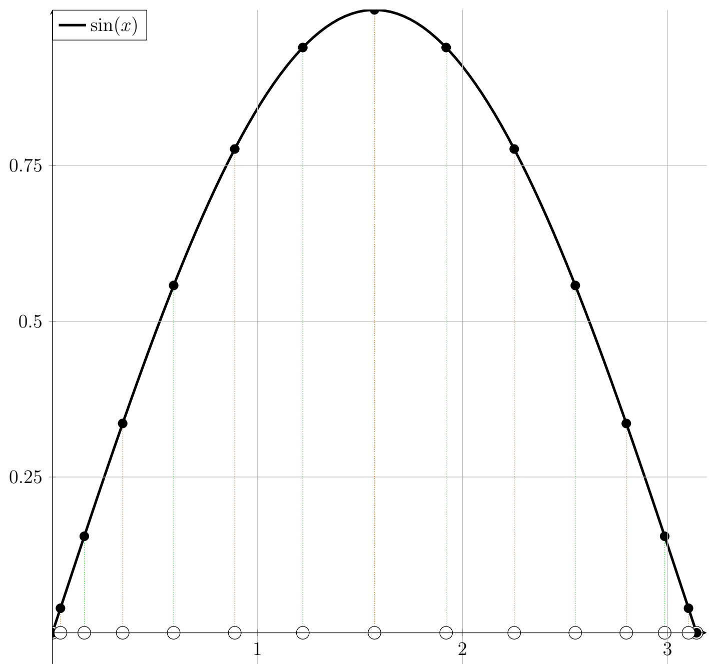


Figure 4.13: The function $\sin(x)$ with two sets of quadrature nodes, $N_1 = 8$ and $N_2 = 15$. All green nodes with green lines are also part of the set with orange lines.

It is very important to note that this definition of the Chebyshev nodes is often used with scaled nodes. The Chebyshev nodes however are defined on the integral $\int_{-1}^{1} f(x)\,\mathrm{d}x$. A scaling of the nodes requires a transformation of the integral, too. For a more general integral to be approximated one therefore has to employ

$$\int_a^b f(x)\,\mathrm{d}x \approx (b-a)\sum_{i=0}^{n} w_i f(x_i)$$

and something analogous in higher dimensions:

$$\int_V f(\mathbf{r})\,\mathrm{d}V \approx |V|\sum_V w_i f(\mathbf{r}_i) \tag{4.29}$$

4.2.5 Wigner D-functions as a basis

Introduction and definitions

The Chebyshev polynomials are one possible base set to model functions in euclidean space. Specifically in this context, they are used to model functions of the type:

$$f : \begin{pmatrix} x \\ y \\ z \end{pmatrix} \longrightarrow r, \begin{pmatrix} x \\ y \\ z \end{pmatrix} \in \mathbb{R}^3, r \in \mathbb{R}$$

This will be used to model three-dimensional densities (see 3.2.2), but can also be used to model a scoring-landscape (see 2.5.5) if one only considers the translations of the query density against the target density in $x, y \text{and} z$-directions (in this case r will correspond to the score). The advantage of doing this is the option of reconstructing a density in a computationally cheap manner and the existence of a quadrature rule. Systematic fitting (see 2.5.5). However, the translational shifts have to be amended with rotational displacement. The question then is, does something similar to the Chebyshev polynomials for the translational case exist for the rotations?
Rotations of rigid bodies in three-dimensional space (see 4.2.3) can be parametrized in many ways, a common choice are the Euler angles (α, β, γ) in the ZYZ convention, given the limits $\alpha \in [0, 2\pi), \beta \in [0, \pi], \gamma \in [0, 2\pi)$ (see4.2.3). These angles parametrize a member of $SO(3)$, the group of rotations in three dimensions. In close analogy to the three-dimensional Chebyshev polynomials $T_{ijk}(x, y, z)$ the Wigner D-functions $D^l_{mn}(\alpha, \beta, \gamma)$ form a basis set of the function vector space $\mathbf{L}^2(SO(3))$. This space consists of functions

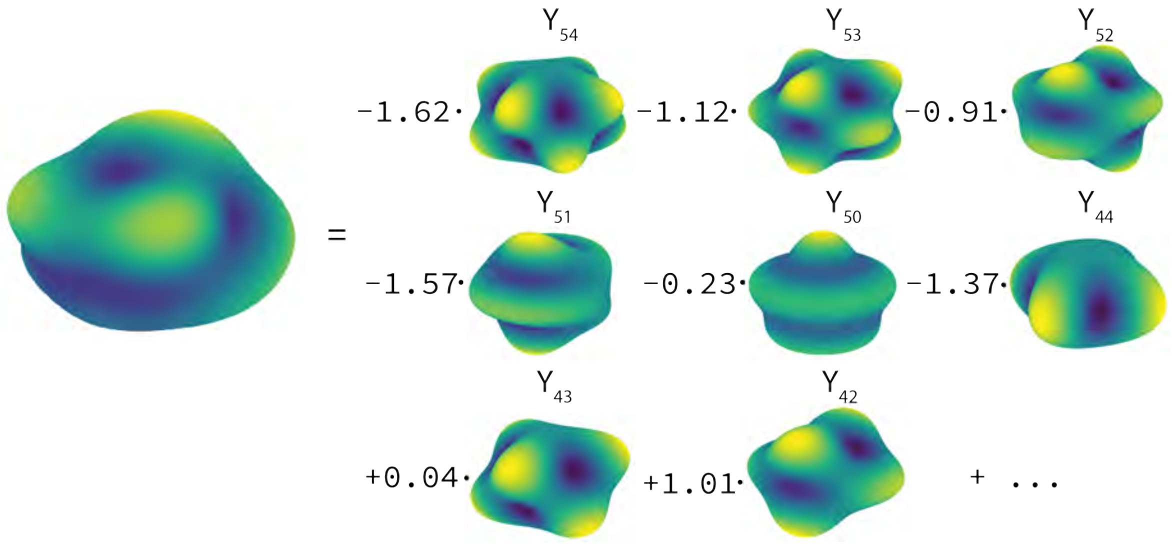


Figure 4.14: Illustration of the decomposition of a member of $\mathbf{L}^2(SO(3))$, a selection of the spherical harmonics as basis vectors is depicted. The "height profile" on the deformed sphere signifies the actual real function values.

which take members of $SO(3)$ as an input and produce values in $\mathbb{C}$

$$f\left(\alpha, \beta, \gamma\right) \longrightarrow c \in \mathbb{C}$$

and have a finite norm $\langle f|f\rangle$ under the scalar product

$$\langle f|g\rangle = \int_{SO(3)} f\left(\rho\right) g^*\left(\rho\right) \mathrm{d}\varrho(\rho), \mathrm{d}\varrho(\rho) = \sin(\beta)\, \mathrm{d}\alpha\, \mathrm{d}\beta\, \mathrm{d}\gamma \tag{4.30}$$

This includes possibly all scoring functions used in systematic fitting. The practical value of this can be seen by considering the measured distribution of a systematic fitting scoring function at a fixed translation as a member of $\mathbf{L}^2(SO(3))$. Then, the mathematical tools pertaining to function vector spaces can be applied to the data. It is worth mentioning that the intuitive notions of more familiar vector spaces such as $SO(3)$ carry over to $\mathbf{L}^2(SO(3))$, for example to idea that a vector can be "disassembled" into simpler basis vectors. As an illustration consider figure (4.14) which shows a decomposition of a member of $\mathbf{L}^2(SO(2))$, one of the possible basis sets of which are the spherical harmonics $Y_{lm}\left(\theta, \varphi\right)$. The members of $\mathbf{L}^2(SO(2))$ have the advantage of being easier to depict than the member of $\mathbf{L}^2(SO(3))$. The basic principles of the treatment of members of $\mathbf{L}^2(SO(2))$ and the basis $Y_{lm}\left(\theta, \varphi\right)$ are completely analogous to the treatment of members of $\mathbf{L}^2(SO(3))$ and the respective basis, $D^l_{mn}\left(\alpha, \beta, \gamma\right)$.

The full definition of the Wigner D-functions is

$$D^l_{mn}\left(\alpha,\beta,\gamma\right) = e^{-im\alpha}\left(-1\right)^{l-n}2^{-l}\sqrt{\frac{(2l)!}{(l+m)!\,(l-n)!}}\left(1-\cos\left(\beta\right)\right)^{-\frac{m-n}{2}}\left(1+\cos\left(\beta\right)\right)^{-\frac{m+n}{2}}$$
$$\cdot\frac{\mathrm{d}^{l-m}}{\mathrm{d}\left(\cos\left(\beta\right)\right)^{l-m}}\left(\left(1-\cos\left(\beta\right)\right)^{l-n}\left(1+\cos\left(\beta\right)\right)^{l+n}\right)e^{-in\gamma} \tag{4.31}$$

The orthogonality relation of the Wigner D-function is

$$\langle D^l_{mn}|D^{l'}_{m'n'}\rangle = \frac{8\pi^2\delta_{ll'}\delta_{mm'}\delta_{nn'}}{(2l+1)} \tag{4.32}$$

which states that under the scalar product 4.30 the functions $D^l_{mn}\left(\alpha,\beta,\gamma\right)$ are indeed orthogonal. The δ_{mn} is the Kronecker delta, which is 1 for $m = n$ and 0 otherwise. The completeness relations reads

$$\sum_{l=0}^{\infty}\sum_{m=-l}^{l}\sum_{n=-l}^{l}D^l_{mn}\left(\alpha,\beta,\gamma\right)D^{l*}_{mn}\left(\alpha',\beta',\gamma'\right) = \delta\left(\alpha-\alpha'\right)\delta\left(\cos\left(\beta\right)-\cos\left(\beta'\right)\right)\delta\left(\gamma-\gamma'\right) \tag{4.33}$$

where $\delta\left(x-x'\right)$ is the Dirac delta, which has the property $\int_{\mathbb{R}} f(x)\delta(x-x')\,\mathrm{d}x = f(x')$. $D^{l*}_{mn}\left(\alpha,\beta,\gamma\right)$ denotes the complex conjugate of $D^l_{mn}\left(\alpha,\beta,\gamma\right)$. The completeness relation tells us that every member of $\mathbf{L}^2(SO(3))$ can be "disassembled" in terms of $D^l_{mn}\left(\alpha,\beta,\gamma\right)$ and how to do it. The result of both statements are the methods to perform a Fourier expansion on $\mathbf{L}^2(SO(3))$. Every function $f\left(\varrho\right)$ can be written:

$$f\left(\varrho\right) = \sum_{l=0}^{\infty}\frac{2l+1}{8\pi^2}\sum_{m=-l}^{l}\sum_{n=-l}^{l}f^l_{mn}D^{l*}_{mn}\left(\varrho\right)\ \text{with} \tag{4.34}$$
$$f^l_{mn} = \langle f|D^{l*}_{mn}\rangle = \int_{SO(3)} f(\rho)D^l_{mn}(\rho)\,\mathrm{d}\varrho(\rho), \rho\in SO(3) \tag{4.35}$$

The coefficients $f^l_{mn}\in\mathbb{C}$ are the equivalent of Fourier-coefficients and are called the Wigner-coefficients. They encode the "building plan" for $f(\rho)$, while 4.34 provides the "assembly instructions" given f^l_{mn}.
As the formulae above suggest, $l\geq 0$ and $-l\leq m\leq l, -l\leq n\leq l$. In practical applications, l is often limited to a maximum value L. This constitutes the equivalent of a "bandwidth" know from classical Fourier-analysis. The higher L, the more detailed the approximations produced by 4.34 (the ∞ replaced by L). Since an experimental signal will, by definition, exhibit a finite amount of detail, it will be well approximated by coefficients $\{f^l_{mn}|0\leq l\leq L, -l\leq m\leq l, -l\leq n\leq l\}$, which then can serve as a compact representation of the signal.

Rotational Sampling

As in the case of translational sampling it is possible to use the functional basis set of the Wigner D-functions to sample data economically, given one uses specific sets of sampling nodes. Depending on ones needs, there is a number of choices for a rotational sampling set ([75],[78],[79]). In this work three criteria have been chosen to select an approach:

1. Grid refinement. The sampling set should expose an adaptable degree of refinement, which allows for adaptive searches.
2. Quadrature. Since the goal of this work is to calculate statistical properties of scoring functions there is a clear need for quadrature rules to calculate integrals over the rotation group $SO(3)$.
3. Exact sampling. The possibility to reconstruct a distribution to some degree of approximation is of potential use in integrative structural modeling. Therefor, the sampling set should offer this possibility in the sense of 4.34.

The approach chosen is that of McEwen and Wiaux ([79]). In their publication their offer a novel sampling theorem for bandlimited functions in $\mathbf{L}^2(SO(3))$. As indicated above, a bandlimited function with regard to a basis is defined by a constraint to its Wigner-coefficients f^l_{mn}: For a given L, M, N with $M, N < L$, $f^l_{mn} = 0 \mid \forall l \geq L, m \geq M, \geq N$, which effectively means that bandlimited functions are "built up" from a limited amount of "building blocks". The higher L, M, N, the more detailed functions can be modeled. The sampling theorem now states that there are specific nodes $(\alpha_a, \beta_b, \gamma_g)$ in $SO(3)$, the knowledge of the function value $f(\alpha_a, \beta_b, \gamma_g)$ is sufficient to faithfully reconstruct any function within the possible spectrum of functions created by a bandlimited basis set. While there is no guarantee that the experimental data falls within this specific class of functions, it is highly unlikely that it cannot be approximated to a degree that is exact for all practical purposes. In fact, given high enough L, M, N, every function can be approximated to some arbitrary degree of precision.
These sampling nodes they provide in their paper are:

$$\alpha_a = \frac{2\pi a}{2M-1}, \text{ with } a \in 0, 1, \ldots, 2M-2 \tag{4.36}$$

$$\beta_b = \frac{\pi(2b+1)}{2L-1}, \text{ with } b \in 0, 1, \ldots, L-1 \tag{4.37}$$

$$\gamma_g = \frac{2\pi g}{2N-1}, \text{ with } g \in 0, 1, \ldots, 2N-2 \tag{4.38}$$

These can be used as a sampling set to reconstruct a representation in terms of Wigner-D functions and on basis of this representation apply more familiar operations, such as a gradient or an integral. The details of this reconstruction are discussed in their paper, they also provide a software to perform the necessary calculations. The latter brings us back to a quadrature formula, which is also provided in [79].

Quadrature on SO(3)

A brief glance at the definition of the Wigner-D functions $D^l_{mn}(\alpha, \beta, \gamma)$ (see 4.31) shows that the integral over $SO(3)$ will be 0, except for $m = n = l = 0$:

$$I = \int_{SO(3)} f(\rho)\,\mathrm{d}\varrho(\rho) = \int_0^{2\pi}\int_0^{\pi}\int_0^{2\pi} \sum_{l=0}^{\infty} \frac{2l+1}{8\pi^2} \sum_{m=-l}^{l}\sum_{n=-l}^{l} f^l_{mn} D^{l*}_{mn}(\alpha,\beta,\gamma)\sin(\beta)\,\mathrm{d}\alpha\,\mathrm{d}\beta\,\mathrm{d}\gamma$$
$$= \frac{2l+1}{8\pi^2} f^0_{00}$$

due to the factors $e^{-im\alpha}$ and $e^{-in\gamma}$ in the definition of $D^l_{mn}(\alpha, \beta, \gamma)$. Since only f^0_{00} is necessary to be calculated for this, one can limit the number of necessary sampling nodes and calculate weights to arrive at a quadrature rule. This rule is provided in the conscious paper (here in a slightly notationally corrected version):

$$I = \sum_{a=0}^{M-1}\sum_{b=0}^{L-1}\sum_{g=0}^{N-1} f(\alpha'_a, \beta_b, \gamma'_g) q(\beta_b) \tag{4.39}$$

with a reduced sampling domain $\alpha'_a = 2\pi a/M$ and $\gamma'_g = 2\pi g/N$. $q(\beta_b)$ are the weights and they are defined as:

$$q(\beta_b) = \frac{(2\pi)^2}{MN}\left[v(\beta_b) + (1 - \delta_{b,L-1})\, v(\beta_{2L-2-b})\right] \tag{4.40}$$

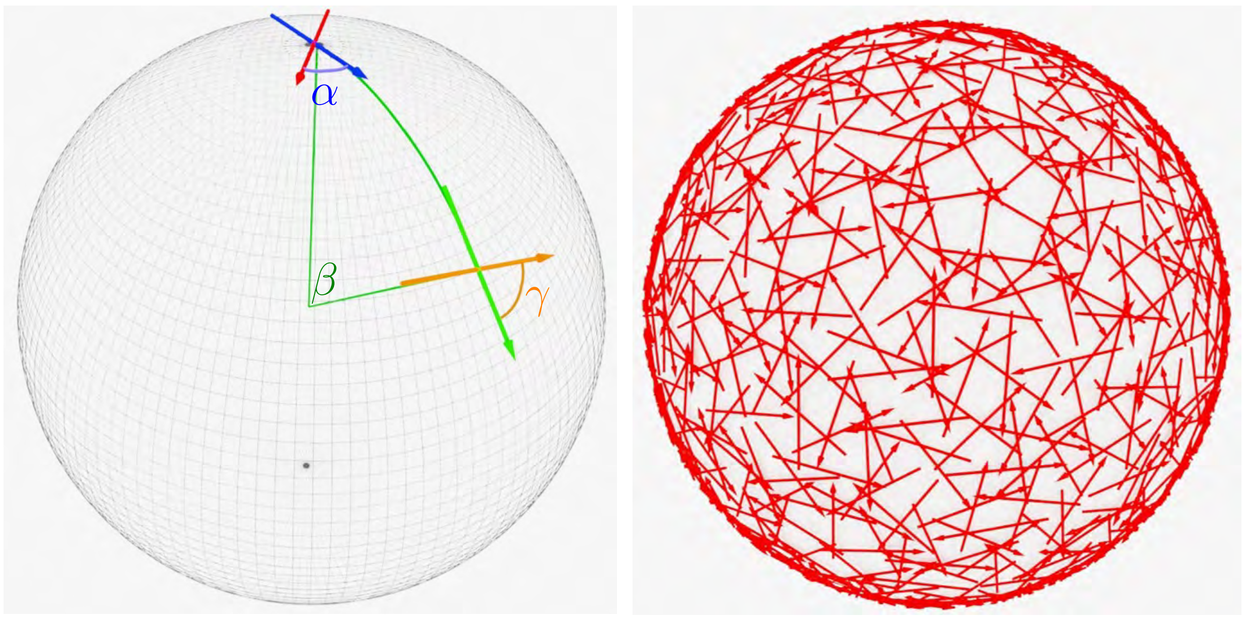
where $\delta_{b,L-1}$ is again the Kronecker-δ and $v(\beta)$ is defined as

$$v(\beta) = \frac{1}{2L-1} \sum_{m'=-(L-1)}^{L-1} w(-m')\mathrm{e}^{im'\beta} \tag{4.41}$$

with $w(-m')$ being defined as:

$$w(m') = \begin{cases} \pm i\pi/2 & m' = \pm 1 \\ 0 & m' \text{ odd and } m' \neq \pm 1 \\ 2/(1-m'^2) & m' \text{ even} \end{cases} \tag{4.42}$$

as defined in [80].



(a) The rotational node (α,β,γ). (b) An example of a sampling set.

Figure 4.15: Visualization of rotational sampling nodes.

Visualizing rotational sampling nodes

$SO(3)$ is difficult to visualize. While $SO(2)$, the space of rotations characterized by two parameters, can be represented as the surface of the unit sphere, $S0(3)$ is defined by three parameters and lacks this options. However, there are ways to get around that and these ways are useful for the purpose of visualizing a rotational sampling set. One will be used here and is adapted from the publication [78]. Figure (4.15) depicts both the construction principle of the visualization method (4.15a) and an example of a rotational sampling set visualized this way 4.15b. The former consists of a sphere and and arrow whose direction and position on the sphere encode the sampling node (α, β, γ). The construction process of said direction and positions is as follows:

1. Start at the north pole of the sphere with the arrow. Turn it counterclockwise on the spot around the axis emanating at the north pole in z-direction by an angle of α.

2. Along the meridian that is aligned to the arrow and in the direction of the arrow, slide the arrow until the angle between its new position, the center of the circle and its old position is β.

3. At that point, rotate the arrow again around an axis perpendicular to the surface of the sphere counterclockwise by an amount of γ.

Every node will be associated to a unique arrow this way.

4.3 Information Theory

Information theory was originally conceived by Shannon [81] to describe the conceptual limits of telecommunication. He drew on older concepts known from thermodynamics and statistical mechanics and put them into a more general framework. Nowadays the central notions of information theory can be found in any field that has to deal with uncertainty and ambiguity in a quantitative manner. This is the reason that it could be of use in structural integrative modeling, too. This is a minimal exposition (and mathematically not rigorous) to enable the reader to comprehend arguments at a later point. A more complete treatment can be found in [82].

Probability distributions and their interpretation

Probability distributions are the basic elements of scientific probabilistic calculus. They encode both the range of possible outcomes of a given process and the probability of a given subset of outcomes. I define "probability" in the common sense: Given a process of consistently varying outcomes the probability of an outcome is the fraction of this outcome occurring and the total number of process trials if one were to conduct infinitely many trials. There is a fundamental distinction in probability theory between the following concepts: The outcomes can be thought as a "configuration space" $\mathcal{A}$, that contains all possible events of the process in question. A random variable X is a "measurable outcome", for instance an element of $\mathbb{R}$ (a length, a quantity, etc.). X assigns a measurable value to the underlying outcome (e.g. the earnings at a game of chance, the relative expression of a gene in the cell). A probability distribution can then be described as function which assigns probabilities to possible values of X. If X denotes for example a length, meaning it can take values ≥ 0, the term

$$\mathcal{P}\left(1 \leq X \leq 5\right)$$

would describe the probability of obtaining a measurement of length between 1 and 5. Note that the length is not the same as the underlying configuration space $\mathcal{A}$, which can be quite complex and X is merely a measurable perspective on it. A good example is again games of chance, where X could signify the winnings. The underlying game can be arbitrarily complex. We will adopt a more simplistic view in the following and consider probability mass functions and probability density functions. Figure (4.16) illustrates a probability density function and figure (4.17) a probability mass function.

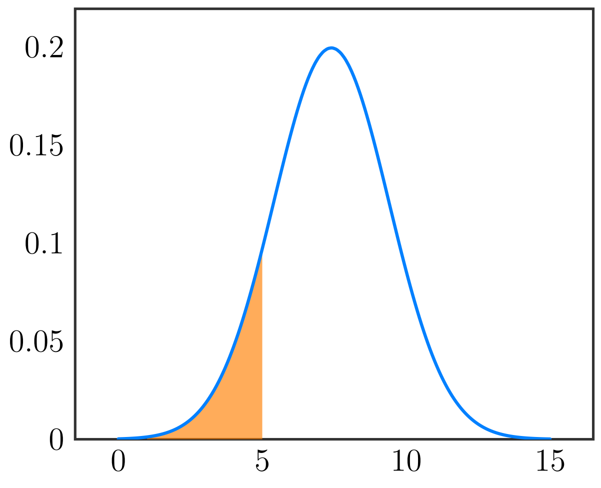


Figure 4.16: Probability density function.

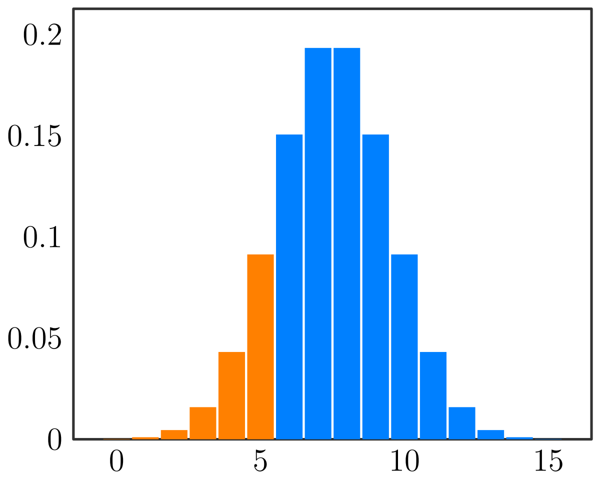


Figure 4.17: Probability mass function.

They both visualize the probability distribution of a random variable X, whose range of values are the real numbers $\mathbb{R}$. The basic intuition is the same: The more area the function covers in a certain range $[x_0, x_1]$ on the x-axis, the more probable the random variable X will fall into that range. Consequently the total area must add up to 1 (100%). The calculation of probabilities differs slightly. The probability $\mathcal{P}(1 \leq X \leq 5)$ (illustrated is the orange portion of the area) for instance would be calculated like so

$$\mathcal{P}(1 \leq X \leq 5) = \sum_{1 \leq x_i \leq 5} p(x_i) \qquad \text{mass function}$$

$$\mathcal{P}(1 \leq X \leq 5) = \int_1^5 p(x)\,\mathrm{d}x \qquad \text{density function}$$

A probability mass function is then characterized by a discrete set of values, a probability density function has to be characterized by an integrable function. In both cases they describe the probabilistic process completely and we can consider the functions themselves as a stand-in for the process. The characteristic of the “landscape” of outcomes resides in the shape of these functions. Since the shape encompasses all values any attempt to characterize the stochastic process as a whole will have to take into account all values. Thus almost all measures regarding probability distributions are sum or integral operations. The next section introduces one of these measures.

Entropy and information content

Entropy is a way to assign a number to a probability distribution as described above. In this sense it is a measure that is used to characterize such distributions and can be compared to the average value or the expected value. It is defined as

$$\mathcal{H}(X) = -\sum_{i=0}^{N-1} p(x_i)\log(p(x_i)) \tag{4.43}$$

for the discrete case. The base of the logarithm is assumed to be arbitrary for now. Entropy ranges between $0 \le \mathcal{H}(X) \le \log(N)$. One of its intuitive interpretation is as a measure of uncertainty. For a given distribution the entropy will be lower if it contains "more information" and higher if it contains "less information". To justify this intuition one can look at two extreme cases. The uniform distribution (see figure 4.19) and the delta distribution (see figure 4.18).

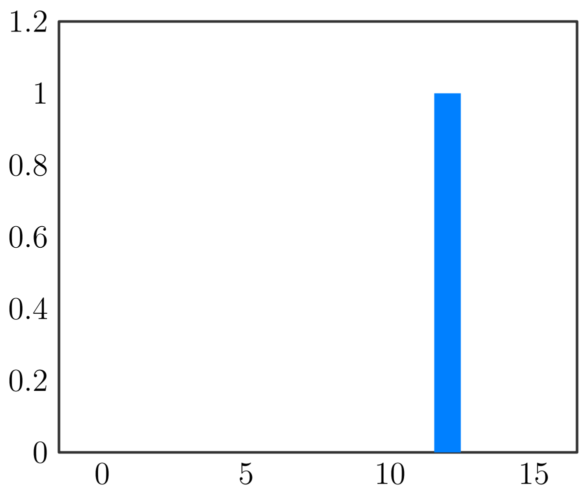


Figure 4.18: Delta distribution.

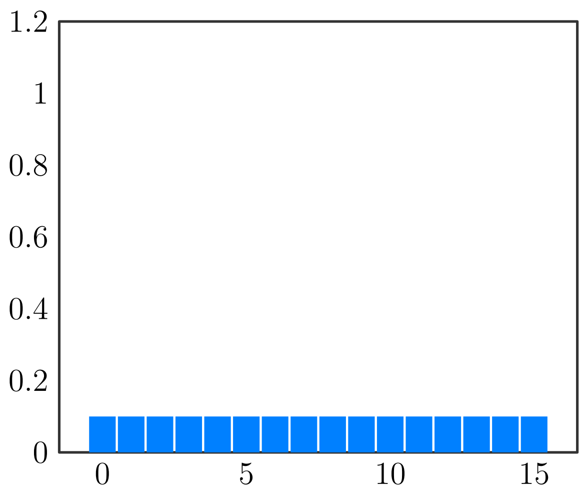


Figure 4.19: Uniform distribution.

The entropy of the delta distribution is

$$\mathcal{H}(X) = -1 \cdot \log(1) = 0 \tag{4.44}$$

using $0 \cdot \log(0) = 0$ (by use of de L'Hospital). The entropy of the uniform distribution is

$$\mathcal{H}(X) = -N \cdot \frac{1}{N}\log\left(\frac{1}{N}\right) = \log(N), \tag{4.45}$$

its maximal value. Given a probabilistic process which would be characterized by one of both distributions, we would be entirely certain of the result in the case of the delta distribution and entirely uncertain in the case of the uniform distribution. This corresponds to their respective entropy values and provides the foundation for the entropy as a measure of uncertainty. This is provided only for the case of a discrete distribution.

There is however an analogue for probability density functions, the *differential entropy*:

$$\mathcal{H}(X) = -\int\limits_{\mathbb{R}} p(x)\log(p(x))\,\mathrm{d}x \tag{4.46}$$

where possible outcomes are elements of $\mathbb{R}$. If it were not so, the integration would be over a more complex space. The immediate connection to the underlying space is one of the main differences to the entropy formula for the discrete case. It also can be negative, e.g. for the uniform distribution $p(x) = \frac{1}{L}$ we get

$$\mathcal{H}(X) = -\int\limits_{0}^{L} \frac{1}{L}\log\frac{1}{L}\,\mathrm{d}x = \log(L) \tag{4.47}$$

which will be negative for L smaller than the basis of the respective logarithm. It is however possible to retain some of the intuitive meaning of entropy in the discrete case. For that we have to define the typical set after [82, p. 245]. For later purposes we will extend it to the general case of a random variable whose values are several real numbers, $X \in \mathbb{R}^n$. This could correspond to positions in three-dimensional space (the random variables would be three Cartesian coordinates x, y, z) or in orientation space (for instance the Euler angles α, β, γ). This set of possible samples is denoted as S^n in the following. The typical set is defined as

$$A_\epsilon^{(n)} = \left\{ (x_0, x_1, \ldots, x_{n-1}) \in S^n : \left\| -\frac{1}{n}\log p\left(x_0, x_1, \ldots, x_{n-1}\right) - \mathcal{H}(X) \right\| \leq \epsilon \right\} \tag{4.48}$$

with $p\left(x_0, x_1, \ldots, x_{n-1}\right) = \prod_{i=0}^{n-1} p(x_i)$. So it is a set of sample points the probability density function of which follows a certain abstract condition involving the entropy $\mathcal{H}(X)$ and a small parameter ϵ. If we further define the *volume* of a sample set A:

$$\mathrm{Vol}\,(A) = \int_A \mathrm{d}x_0\,\mathrm{d}x_1 \ldots \mathrm{d}x_{n-1} \tag{4.49}$$

we can state the theorem that for n sufficiently large

$$(1-\epsilon)2^{n(\mathcal{H}(X)-\epsilon)} \leq \mathrm{Vol}\left(A_\epsilon^{(n)}\right) \leq 2^{n(\mathcal{H}(X)+\epsilon)} \tag{4.50}$$

holds. This puts limits on the volume of the typical set. For very small ϵ we can say that this volume is approximately equal to $2^{n\mathcal{H}(X)}$. In conjunction with the theorem that $A_\epsilon^{(n)}$ is the smallest set with probability $\mathcal{P}\left(A_\epsilon^{(n)}\right) \geq 1-\epsilon$ and the fact that an n-dimensional cube of volume a has side length $a^{\frac{1}{n}}$, so that our approximate volume is $\left(2^{n\mathcal{H}(X)}\right)^{\frac{1}{n}} = 2^{\mathcal{H}(X)}$, we can state:
Entropy is the logarithm of the sides of a cube with a volume that is equivalent to the

volume of the typical set, which holds most of the probability density. For a given random variable we therefore have the relation: The smaller the entropy, the more confined the space in which most of the probability density concentrates. In this way we recover the intuition of entropy in the discrete case.

Other information theoretic metrics

There are other concepts in information theory that can be useful in certain circumstances. The *normalized entropy* is defined as

$$\eta\left(X\right) = \frac{H}{H_{max}} = -\sum_{i=0}^{n-1} \frac{p\left(x_i\right) \log\left(p\left(x_i\right)\right)}{\log\left(n\right)} \tag{4.51}$$

As the name suggests it is useful for normalizing different entropy measurements, which can make comparisons easier. The *Kullback-Leibler-Divergence* is often called relative entropy and is defined as

$$\mathrm{D}(P \parallel Q) = \sum_{i=0}^{n-1} p\left(x_i\right) \log\left(\frac{q\left(x_i\right)}{p\left(x_i\right)}\right). \tag{4.52}$$

It takes two distributions as an input and measures a certain difference. This difference can be interpreted as the a gain of information if one would use the distribution P rather than Q. In a practical situation then, one can compare two different distributions against each other or a third base distribution, such as the uniform distribution. If $\mathrm{D}(P \parallel Q)$ is close to 0, there is no gain in information in choosing P over Q.

Chapter 5

Results

The results that will be described are of varying nature and might be hard to place. A short overview of how they fit into the general context will be provided here. There are three different kinds of results: Sections 5.1 and 5.2 describe the results obtained in the modeling project of the TFIIIC-complex (see 2.6) and additions to the *Assembline* software. The sections 5.4, 5.5, 5.7, 5.8, 5.7, 5.6 and 5.10 contain results that are partially computational and partially conceptual. They are arranged according to the sequence in which one would encounter them in a typical *Assembline*-style modeling project. The sections 5.12, 5.11 and 5.13 contain conceptual results, the results of model experiments and the results of applications of entropy measurements. The description tries to balance between detail and accessibility of exposition. More detailed descriptions are given if there is some degree of novelty involved.

5.1 TFIIIC modeling

Several systematic fitting attempts have been made using varying density maps of the PIC (Pre Initiation Complex), the TFIIIC+tDNA complex (negative stain), the TFIIIC+TFIIIB+tDNA complex as target and crystal structures and homology models of parts of TFIIIC-subunits ($\tau_{95}, \tau_{131}, \tau_{55}$) as query. The target maps had local resolutions (see 2.4.4) between 12 Å and 40 Å. No significant fits have been found.
Upon receiving a 5 Å-map that is now known as the τ_A subcomplex, $\tau_{95}, \tau_{131}, \tau_{55}$ have been fitted again using the Chimera FitInMap (see [28]) tool with the cam-score option, this time with significant fits. These fits are shown in figure (5.1). Statistical significance was achieved according to the criterion used within the *Assembline* pipeline. Figure 5.2 shows the score distributions highlighting the significant fits. Further attempts to corroborate

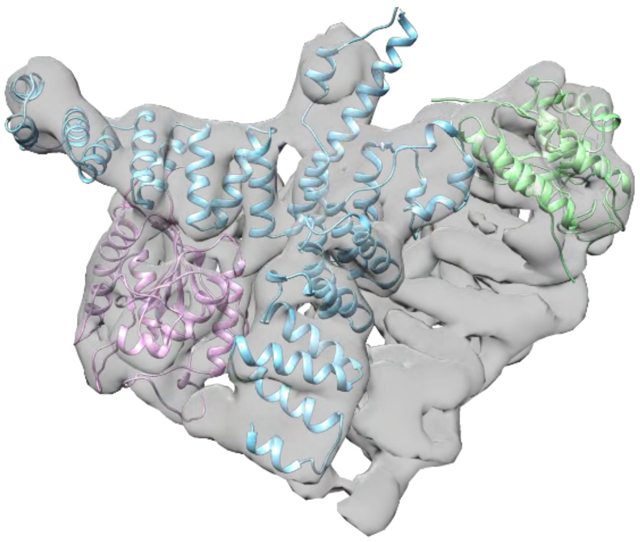
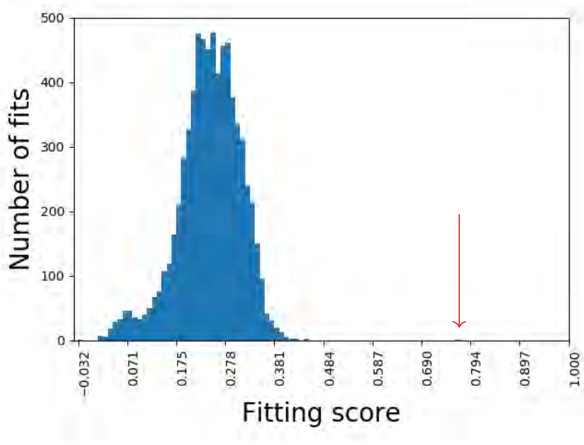
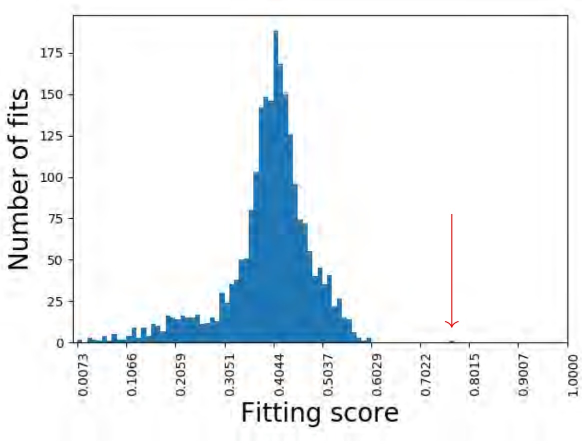


Figure 5.1: The best $\tau_{95}, \tau_{131}, \tau_{55}$-domain fits rendered within the τ_A density map.



(a) The score distribution for the τ_{131}-domain fit.



(b) The score distribution for the τ_{55}-domain fit.

Figure 5.2: Score distribution for the significant systematic fitting protocols

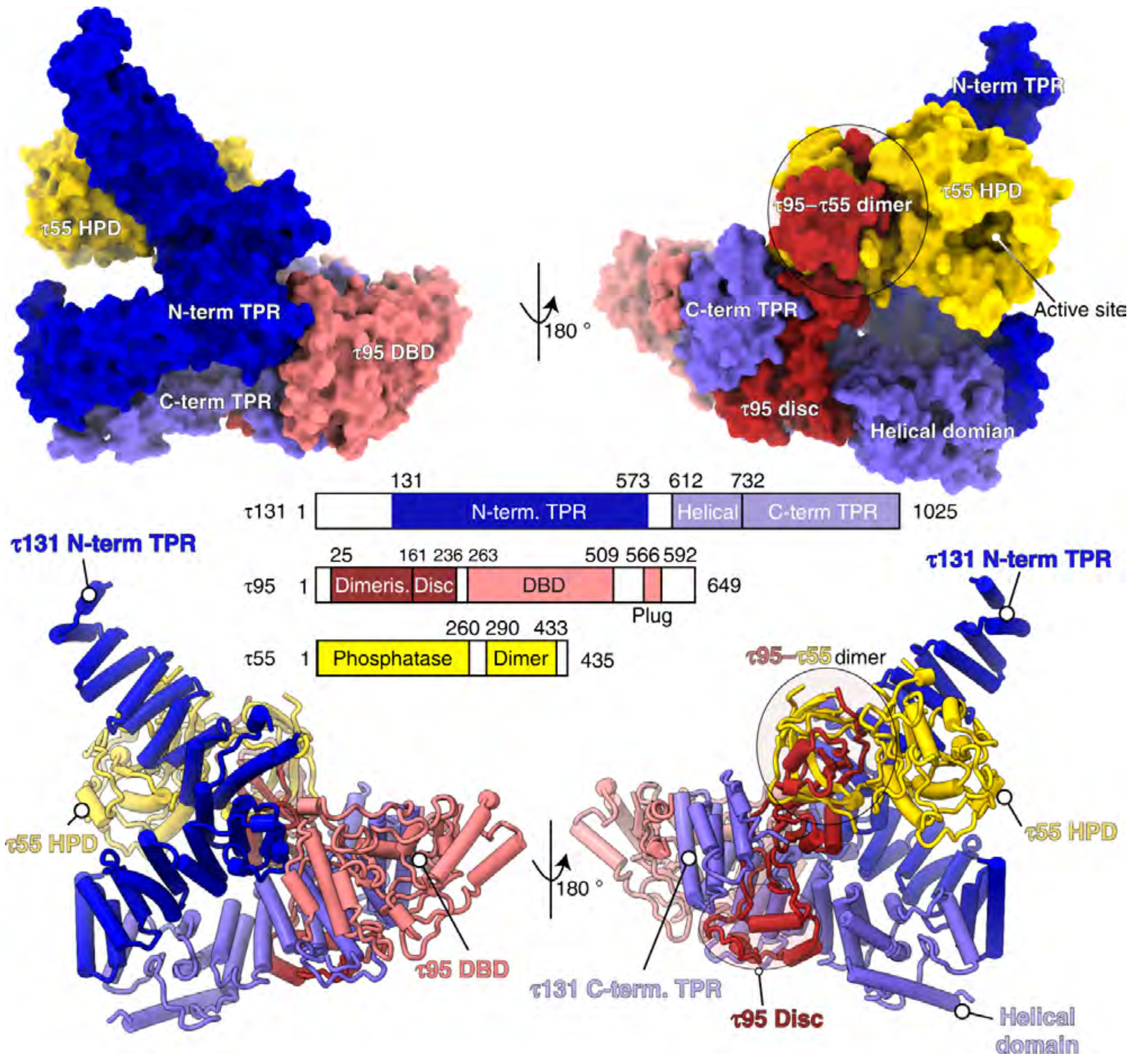


Figure 5.3: The τ_A subcomplex of the TFIIIC complex as determined in [83], image taken from there.

the fits using crosslinks provided by the collaborator were made, but yielded no conclusive results. The complete structure of the TFIIIC-τ_A subcomplex was later obtained using single particle cryo-EM methods (see [83], also see figure 5.3) In an additional effort the existing τ_A and Brf1-TBP structures were fitted to a 33.5 Å negative stain map of τ_A in complex with Brf1-TBP that was obtained earlier, to determine the handedness of the density map and tentative positions for the two structures. The scores that were used for this are the cc-score (see 2.5), the overlap-score (see 2.4), the Chamfer distance (see 2.5.5) and the envelope score (see 2.5.5). These specific scores were chosen due to their suitability to match surfaces against each other and the negative stain technique providing a signal of the protein surface. The workflow that was chosen differed slightly from the standard workflow in *Assembline*. First, the normal FitInMap tool from Chimera ([28])

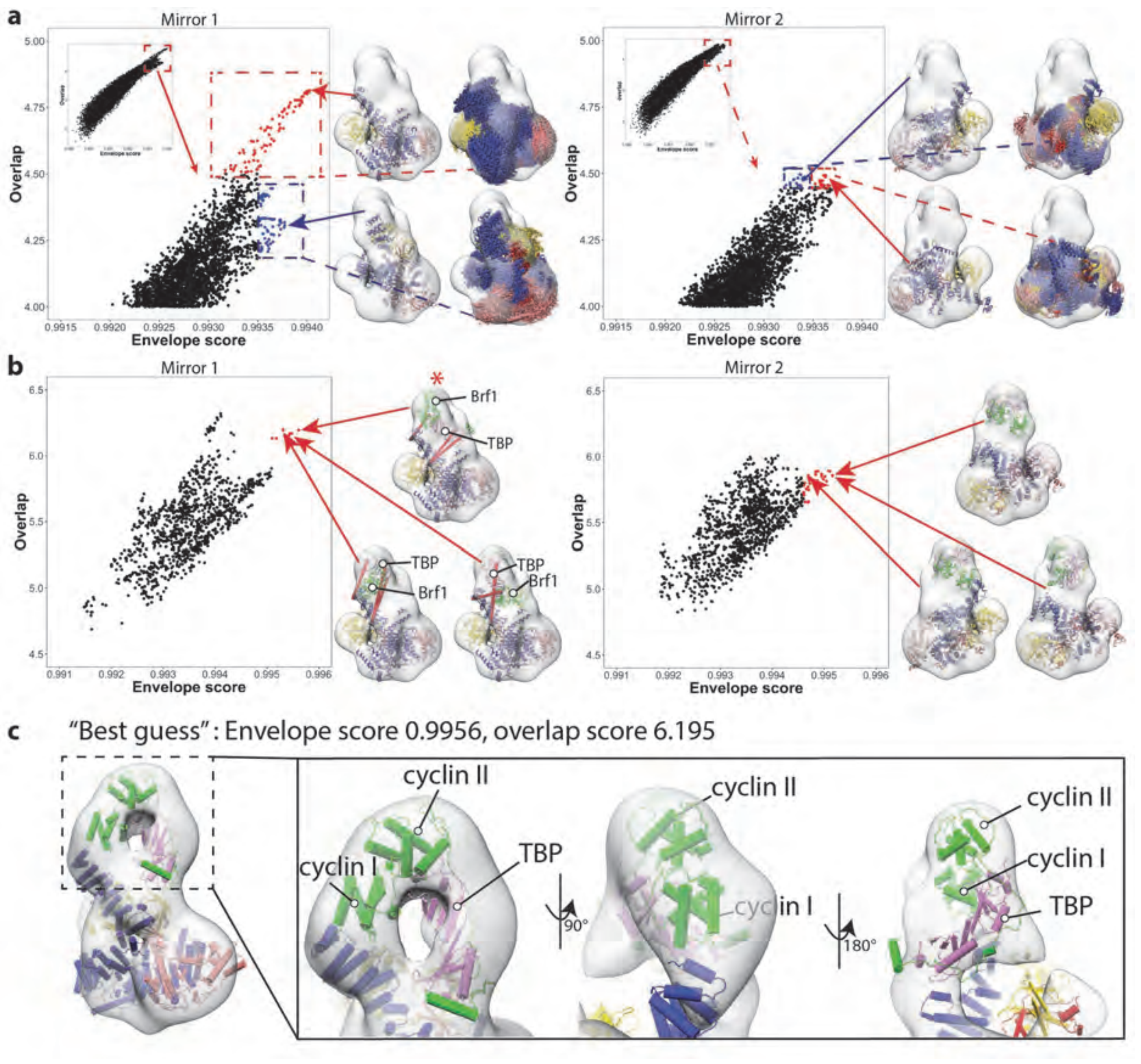


Figure 5.4: The results of a systematic fitting exploration of the τ_A-Brf1-TBP-negative stain map with the τ_A and Brf1-TBP structures using different scoring methods.

was used to sample 100.000 random initial positions using the cc-score, which were also optimized by gradient descent, which ran for 100 steps. These samples were clustered and resulted in $20.000 - 24.000$ unique transformations. These transformations were used to score the same setup for the other mentioned scores using the TEMPy library (see [84]). The resulting score populations were then correlated against each other to try and detect solutions that scored high with every scoring method. Figure (5.4) shows the results of this process. Subfigure **a** indicates that Mirror1 consistently scores better and can be interpreted as the correct orientation. Both fitting populations were used to combine them into a set of models using *Assembline*. This yielded a consistent fit for τ_A and ambiguous fits fore Brf1-TBP (see 5.4, **b**). The orientation of Brf1-TBP could not be identified due to lack of data.

5.2 Contributions to the *Assembline* structural modeling pipeline

Several contributions to the *Assembline* pipeline were made and are either published with it ([26]) or were used in other publications ([83], [85]). They take the take the form of simple helper programs or parts of the *Assembline* software. In sum, they have led to considerations that led in turn to later results.

5.2.1 Geweke convergence test implementation

As outlined in 2.5.3, *Assembline* employs a Monte Carlo procedure to generate a number of alternative models. As discussed in 4.1.3, the burn in period can be very short, leading to inefficiently long Monte Carlo runs. A Geweke convergence test (see 4.1.3 has been implemented in `Python` with the interface

```
convergent(x, first_frac = 0.1, last_frac = 0.5, intervals = 10)
```

`x` is an array of `float` values, `first_frac` denotes the fraction of the array `x` that is considered to represent the early part of a run, vice versa for `last_frac`. `intervals` is the size of the subintervals used within the test. The implementation relies heavily on [72].

5.2.2 Bash analysis tools

As described in 2.5.5, systematic fitting produces a set of folders, subfolders and files that contain the results of a fitting instance. The number of files and subfolders can become high for more elaborate projects and it may be difficult to assess the relevant information. A number of `bash` scripts have been create to enable an easier access to systematic fitting data sets. They are based on simple `bash`,`Python` and `R` scripts and are meant to be run on a console environment, such as a cluster. They will be described briefly. All scripts are used on a systematic fitting base folder.

name	description
`fitsummary`	Used on a systematic fitting base folder. Prints a summary of the queries and targets, checks if `solutions.csv` files exist (indicating that the fitting ran correctly) and if the p-values have been calculated.
`genpval`	Searches through the folder structure and invokes the p-value calculation script on every `solutions.csv`-file it encounters.
`evalpval`	Searches through the folder structure to find all `solutions_pvalues.csv`, evaluate them and print a summary that shows representatives of ranges of p-values and their respective target and query. This allows for a quick assessment of the significance of the fitting data set.
`evalstruc`	Searches through the folder structure and compiles a list of a queries that were fit in the fitting data set. Then offers the options to survey p-values for that query or to create, in the case of a `pdb`-file, a transformed version of the structure that can be placed within the target density. Also, scripts to plot the p-value distributions can be run.

An example of a `fitsummary` output:

```
                          EM map|                                Structure        sol        pval

-------------------------------------------------------------------------------------------------
       TauA_IIIB_190207_job49.mrc|                                 4BJI.pdb        yes        yes
                                |                                 4BJJ.pdb        yes        yes
                                |6F44_cut_without_helices_without_DNA.pdb        yes        yes
                                |                        T131_TPR_5AEM.pdb        yes        yes
                                |                Tau55_PGAM_2YNO.clean.pdb        yes        yes

-------------------------------------------------------------------------------------------------
TauA_IIIB_190207_job49.zmirror.mrc|                                 4BJI.pdb        yes        yes
                                |                                 4BJJ.pdb        yes        yes
                                |6F44_cut_without_helices_without_DNA.pdb        yes        yes
                                |                        T131_TPR_5AEM.pdb        yes        yes
                                |                Tau55_PGAM_2YNO.clean.pdb        yes        yes

-------------------------------------------------------------------------------------------------
```

An example of the `evalpval` output:

```
                      Structure                                  EM map        BH_adju...        cam_score        solu...

================================================0.0e+00 to 1.0e-05================================================


              T131_TPR_5AEM.pdb            TauA_IIIB_190207_job49.mrc        1.5796e-12        0.769083            0
      Tau55_PGAM_2YNO.clean.pdb            TauA_IIIB_190207_job49.mrc        6.6138e-09        0.764220            0
```

```
=============================================1.0e-05 to 1.0e-04===============================================


=============================================1.0e-04 to 1.0e-03===============================================


=============================================1.0e-03 to 1.0e-02===============================================


=============================================1.0e-02 to 1.0e-01===============================================

     Tau55_PGAM_2YN0.clean.pdb        TauA_IIIB_19020....zmirror.mrc        2.0458e-02        0.609205          0
     Tau55_PGAM_2YN0.clean.pdb        TauA_IIIB_19020....zmirror.mrc        3.3554e-02        0.598789          1
     Tau55_PGAM_2YN0.clean.pdb            TauA_IIIB_190207_job49.mrc        6.8173e-02        0.595159          1
     Tau55_PGAM_2YN0.clean.pdb            TauA_IIIB_190207_job49.mrc        7.1007e-02        0.593878          2
     Tau55_PGAM_2YN0.clean.pdb            TauA_IIIB_190207_job49.mrc        7.1727e-02        0.593004          3
```

And lastly, an instance of an `evalstruc` instance:

```
[ 1]                                        4BJI.pdb
[ 2]                                        4BJJ.pdb
[ 3]        6F44_cut_without_helices_without_DNA.pdb
[ 4]                               T131_TPR_5AEM.pdb
[ 5]                       Tau55_PGAM_2YN0.clean.pdb
[ x]                                           Exit.
?>4
T131_TPR_5AEM.pdb

                        +++ Overview for    T131_TPR_5AEM.pdb +++

  [A]            TauA_IIIB_190207_job49.mrc                     T131_TPR_5AEM.pdb        BH_adju...         cam_score
                                                                                            1.6e-12          0.769083
                                                                                            2.7e-02         -0.031849
                                                                                            1.5e-01          0.452468
                                                                                            1.5e-01          0.446771
                                                                                            1.5e-01          0.044092
  [B]        TauA_IIIB_19020....zmirror.mrc                     T131_TPR_5AEM.pdb        BH_adju...         cam_score
                                                                                            5.5e-02          0.443519
                                                                                            5.5e-02          0.443354
                                                                                            5.5e-02          0.028982
                                                                                            5.5e-02          0.028665
                                                                                            5.5e-02          0.028598
Choose action regarding to inspected structure:
[p]         create pdb ...
[d]         print pvalue distribution ...
[c]         choose different structure ...
[x]         exit ...
```

5.3 Description of the GPU implementation of the modeling classes

This section describes the structural integrative modeling parallel programming interface. This interface aims at providing tools to process the most prominent data types in integrative structural modeling and offers a number of solution for common tasks in the field. The section is subdivided into two parts: The first part will describe the general features, workflows and design principles. The second part will contain a terse description of the interface.
The features and principles described in the following result from the intention to use GPU resources as much and as convenient as possible. Both goals can only be fulfilled by adding a layer of abstraction between the raw data layer and the high level programming API. For maintenance purposes this intermediate layer was kept conceptually as simple as possible.

Memory management and data representation

The first issue that is addressed is the need for synchronization between host and device. Any data represented on the host must be mirrored on the device and vice versa (see to 3.3.2 for a brief discussion). Additionally multiple devices need to be understood as unable to communicate directly with each other, so that data cannot be spread out between devices without some considerations. Figure (5.5) shows an illustration of the implemented solution. Every data object implemented has the ability to instantiate versions of itself on any given device. These instances hold the bare minimum of memory fields needed to represent the data. Every instance is mirrored in the host memory in a one-to-one fashion, because the result of the processes deployed on different devices might change the data instances in different ways. The memory synchronization procedures are completely hidden from the high level API user. Moreover the number of devices is hidden from the user and managed in the background. The scripting interface, the layer where the actual modeling procedure is defined, is kept as free as possible from low level hardware and memory management. The following code snippet shows an example which details the generation of a density map from a particle data resource.

```
Particles pdb;
Density pdb_density;
DensityGenerator generator;
int reader_index = pdb.fromPdb(pdb_file);
```

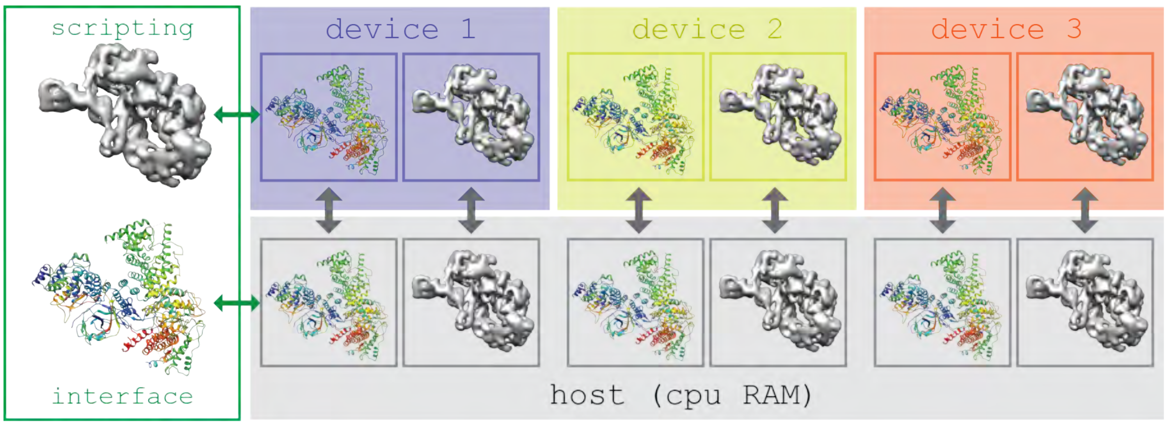


Figure 5.5: Illustration of the synchronization scheme.

```
pdb.createInstanceFromReader(gpu_index,reader_index);
generator.createInstanceFromParticles(&pdb,gpu_index);
pdb.find_bounds(gpu_index);
pdb_density.createInstanceFromBounds(pdb.volume(gpu_index),pixel_size,0,gpu_index);
Stamp stamp;
stamp.createInstance(sigma,pixel_size,gpu_index);
generator.generateMolecularDensity(&pdb_density,&pdb,&stamp,24,256,gpu_index);
pdb_density.toMrc(mrc_file,20,gpu_index);
```

As is apparent, the only reference to the presence of GPU devices is a the `gpu_index` variable, which itself is handled by the `Engine` class. While figure (5.5) only illustrates the representation of data models, entities more properly understood as processes, such as scoring methods, sampling procedures or alignment tasks, which need working memory on the device, are also designed in this way. For each of these classes a central `map<int,Register>` maps to each `gpu_index` a `Register` structure, which is simply a collection of memory addresses on the given device pertaining to the representation of the class data on that device and the host. Not always it is necessary to mirror all memory resources, for example in the case of working memory on the device for intermediate results. The following example details the `DensityGenerator_Register` structure of the `DensityGenerator` class.

```
struct DensityGenerator_Register {
        float * h_offset_shift;
        float * d_offset_shift;
        uint * d_offsets;
};
```

The `d_offsets` memory address is only present in device memory, which can be seen by acknowledging the convention that any memory address variable pointing to host memory has the prefix "h_" and address variables with the prefix "d_" point to device addresses. Given this framework any function that performs work on the data resources will look similar to this example:

```
void DensityGenerator::generateMolecularDensity(Density *const& density,
Particles *const& particles, Stamp * const& stamp, int gridDim, int blockDim, int gpu_index){
        cudaSetDevice(gpu_index);
        Density_Register * density_instance = &density->_instance_registry[gpu_index];
        Particles_Register * particles_instance = &particles->_instance_registry[gpu_index];
        DensityGenerator_Register * generator_instance = &_instance_registry[gpu_index];
        Stamp_Register * stamp_instance = &stamp->_instance_registry[gpu_index];
        ...
}
```

Here the `cudaSetDevice` function is used to make sure that the host code addresses the device of index `gpu_index` and subsequently the `Registers` are obtained to access to memory addresses that point to the memory ranges containing the respective data. After that, they can be used to perform memory management operations or parallel computation `kernel` calls.

Execution model and process management

The data representation model outlined above (see 5.3) enables the parallel use of multiple devices. While these can be run in parallel, communication between devices among each other and the host is either slow or cumbersome. The partition of the workload and the final synchronization of the results of the given processing tasks can be addressed in a number of ways, depending on the nature of the tasks itself. Two general procedures have been implemented to address this issue.

Figure (5.6) illustrates the execution model of a *dispatch queue*. This model was implemented in the `Engine` class and describes one of the classes two execution modes. Its process is the following:

- The user has to define a number of tasks using the provided parallel processing framework. These tasks must be implemented as functions with and need to have either the signature `void(int)` or `void(std::ofstream *,int)`. The last `int` signature is the `gpu_index`, which will be dynamically assigned to the task by the engine. The `std::ofstream *` variable can be used to enable access to an output file stream, e.g. to write out results or log error messages. The user need not to be

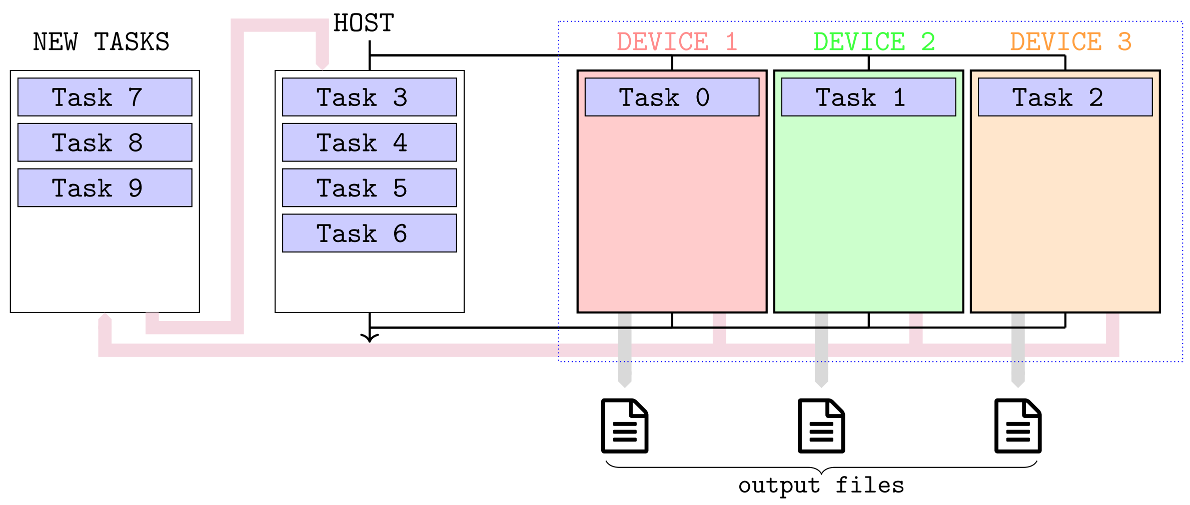


Figure 5.6: Schematics of the single-task-per-device execution mode, in this example the host distributes tasks among three devices.

concerned with the number of type of devices. These will be assessed by the engine using the `CUDA` API.

- Once the user has defined a number of tasks (any number), these can be submitted using the `Engine::dispatch(task)` function. On initiation of the program the engine will have spawned a `host thread` for each device it detects. These `host threads` will then continuously run until the program is terminated. If a task is dispatched, the engine will do one of two things: If there is an unoccupied device, the engine will assign the task to this device which will be occupied until the specific task is finished and then be ready to accept a new task. If there is no unoccupied device, the engine will hold onto the task until one device is free and the task is the first in line. There is no guaranteed order of execution for the tasks. However, tasks can be simply compounded by defining them in sequence within a new function as described above.

- While the task is running on a given device, there are two options to obtain the results. One is to write them to a `std::ofstream` (a file) to be accessed later or to define new tasks based on the results of the preceding task that will be submitted to the engine for processing.

If a given task requires the memory resources one device can supply, one can easily define a whole population of tasks, e.g. varying a given set of parameters to assess the effect on the results. This is often the case since the memory capacity of a modern GPU runs in the gigabytes, which is sufficient for many tasks in structural integrative modeling.
The second execution model can be used to distribute the workload of one massive task to multiple devices. This execution model uses the devices in a parallel manner and can be

used to effectively speed up one single task or multiple consecutive tasks using multiple devices. Yet again the number of devices and the synchronization are handled by the engine. Figure (5.7) shows a schematic of this execution model. There are several key differences to the first model:

- Since the different devices need to be synchronized after performing each of the consecutive tasks, there is the possibility that there is a general loss of performance if one device is slower or faster than the others. Therefore the tasks should be distributed as evenly as possible.
- The tasks need to be amenable to distribution in an indexable manner. Since the workload needs to be distributed over the devices, it needs to be dividable in a discrete and linear manner, so that `DEVICE 1` handles workload indexed by `i = [0,511]`, `text` the workload indexed by `i = [512,1023]` and so forth. Because the implementation of the algorithms resulting in the workload already is parallelized (since it is running on parallel computation devices), this is usually easily achieved.
- Unlike the dispatch queue, there are canonical intervals of host synchronization, which can be used to run intermediate processing steps.

For most tasks the dispatch queue model will be sufficient, the distributive execution model should just used for singular tasks. Technically, both modes could be combined if there is a task structure which is suitable for this option.

5.4 Density Sampling

The simulation of an electron density from a set of atom coordinates representing a molecular structure is a necessary part of many widely used integrative modeling protocols. It is possible to leverage parallel programming techniques to implement this procedure. Here I will describe one way to accomplish this.

5.4.1 The procedural basis

The standard procedure with regard to density simulation off crystal structures for cryo-EM density maps is outlined in 3.2.3. The key features exploited in the parallel implementation are as follows:

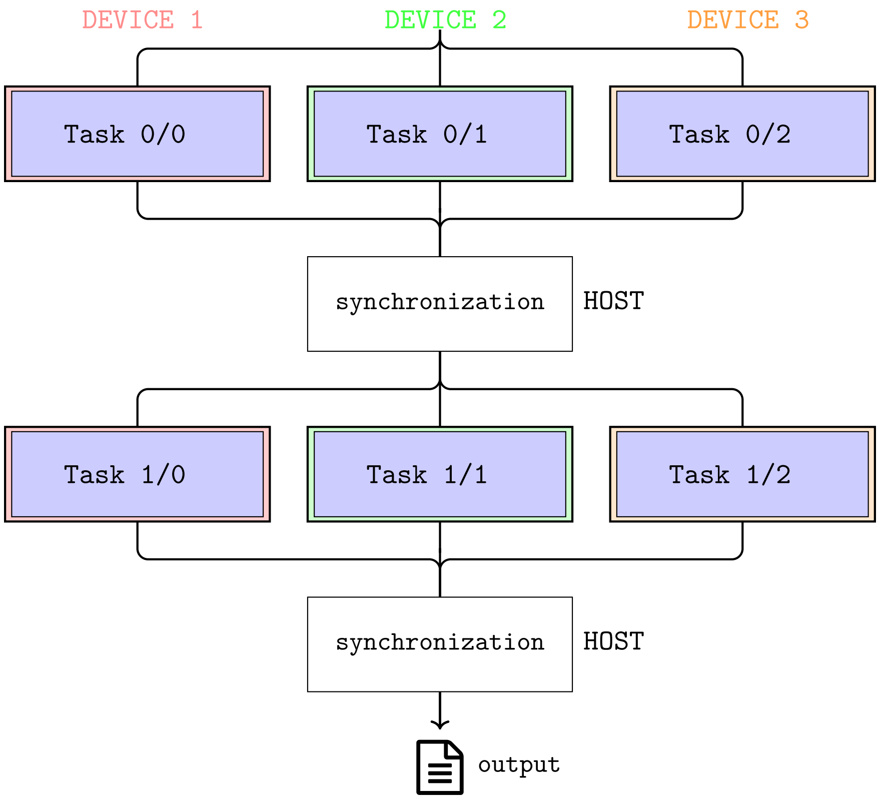


Figure 5.7: Illustration of the distributive task management with three and two consecutive tasks.

- The values of the kernel function used to simulate the density around a single particle vary very little with the particle position within a single pixel. The variation trivially approaches zero when the pixel size approaches zero.
- The simulated density for particles representing atoms of different atomic number Z are scaled versions of a base density for a given resolution factor R.
- The effective area that one particle affects is typically small since the kernel functions fall of quickly.

This means that the simulated density per atom added to the overall density does not have to be recalculated and costly operations can be saved. Through use of the memory hierarchy of a GPU device it is possible to “imprint” these scaled basic densities on the density volume in an efficient and scalable way. The imprinting is done exploiting the fact that the density values do not change and their relative positions do neither. By use of the property of the density data structure illustrated in 3.13 it is possible to circumvent the calculation of the spatial coordinates of the pixels. The arising problem of concurrency can be tackled by the use of atomic operations (see 3.3.4).

5.4.2 The procedure

The following procedure has been implemented and tested. Operations that do not contribute to the specific nature of the algorithm, such as memory management, memory initialization and certain aspects of flow control will be omitted. All described operations are conducted on the same device. The basic design idiom of the computational kernels used is the `worker` pattern. It is illustrated by figure (5.8).

1. The structure is loaded into a `particle set` data structure (see 3.2.1) with the additional information of the atom type for each `particle`. The `float4` data type is employed for that purpose. Their `bounding box` (see 3.2.1) is determined by use of the `thrust` standard library and its `reduction` operation and custom unary functions for the upper and lower "end" of the `bounding box`.

2. This information is used to create an instance of the `density` data structure. The `pixel_size` is defined by the user and since it is not guaranteed that the `coord_dimension` is a integer multiple of the pixel size, the closest value that fulfills this condition and is greater than the `particle set bounding box` will be used together with the according `pixel_dimension`.

3. Given the information about the `density` data structure, a "stamp" `density` is created with the same pixel size. This `density` represents the basic simulated density around each atom. Its size depends on the `resolution` parameter, which is set by the user. The stamp is encoded by two sets of values. One set contains the actual `values` of the `density`, the other set contains `offsets` that encode the position of the values in global, linear memory.

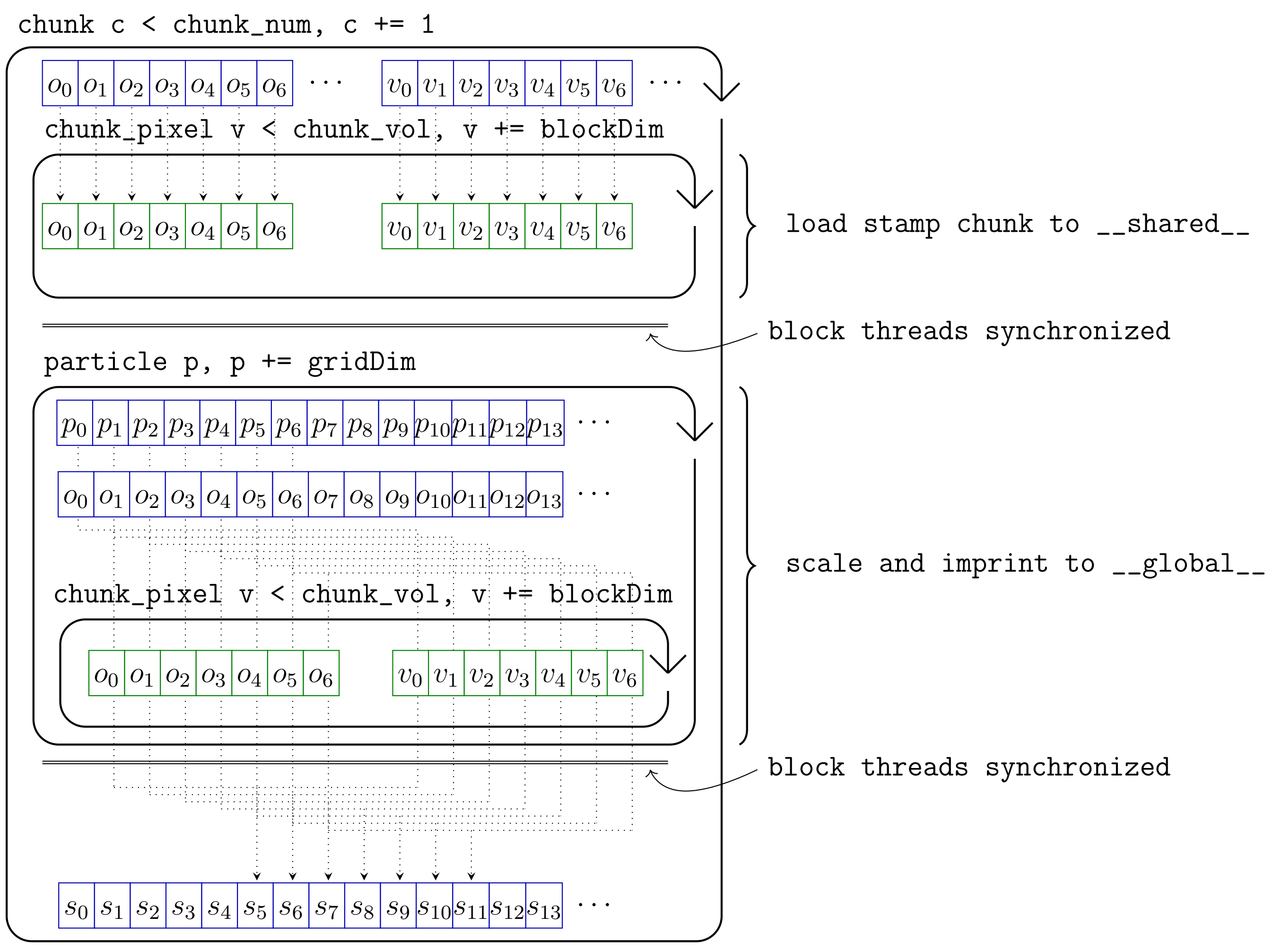
4. Depending on the available `__shared__` memory of the specific device the volume of the stamp `density` is divided into a number of chunks, both the `offsets` and the `values`.

5. Due to the fact that the stamp `density` will overstep the boundaries set by the `particle set bounding box`, the initial `density` is enlarged by a `padding`.

6. The coordinates of each of the atoms of every `particle set` are translated to the `offset` of the pixel within the target `density` that contains the coordinates in the manner of 3.2.2 and position in `__global__` memory.

7. Finally a kernel iterates over each pixel of each chunk of the stamp `density` for each `particle`, multiplies the corresponding value with the atomic number Z of the particle, calculates the address of the affected pixel within the global density by use of the above calculated `particle offsets` and the stamp `density offsets`

and writes it to the appropriate position in the `__global__` memory segment representing the overall simulated `density`. This is done via an `atomic` operation (see 3.3.4), since different `threads` might be writing to the same memory segment at the same time.



chunk c < chunk_num, c += 1
o_0 o_1 o_2 o_3 o_4 o_5 o_6 ⋯ v_0 v_1 v_2 v_3 v_4 v_5 v_6 ⋯
chunk_pixel v < chunk_vol, v += blockDim
load stamp chunk to __shared__
block threads synchronized
particle p, p += gridDim
p_0 … p_{13} ⋯
o_0 … o_{13} ⋯
chunk_pixel v < chunk_vol, v += blockDim
scale and imprint to __global__
block threads synchronized
s_0 … s_{13} ⋯

Figure 5.8: Illustration of the density sample kernel

The basic idea behind the utilization of the GPU memory hierarchy to achieve density sampling in a spatial grid is illustrated in figure (5.9). Both $\textcircled{\scriptsize I}$ and $\textcircled{\scriptsize II}$ symbolize `shared` memory volumes, which are faster to write and read from than the `global` memory volume $\textcircled{\scriptsize III}$. $\textcircled{\scriptsize I}$ holds both the actual density `values` and the `offsets` that relate the values to each other in a linear representation of the underlying grid. These `offsets` refer to the global grid. $\textcircled{\scriptsize II}$ shows that not necessarily the whole stamp density has to be held in the `shared` memory, which is usually rather small and therefore not suitable for larger stamp densities. This is dealt with by partitioning the stamp if necessary. Once all stamp partition and particles are iterated over the `global` memory $\textcircled{\scriptsize III}$ holds the complete representation of the sampled density.

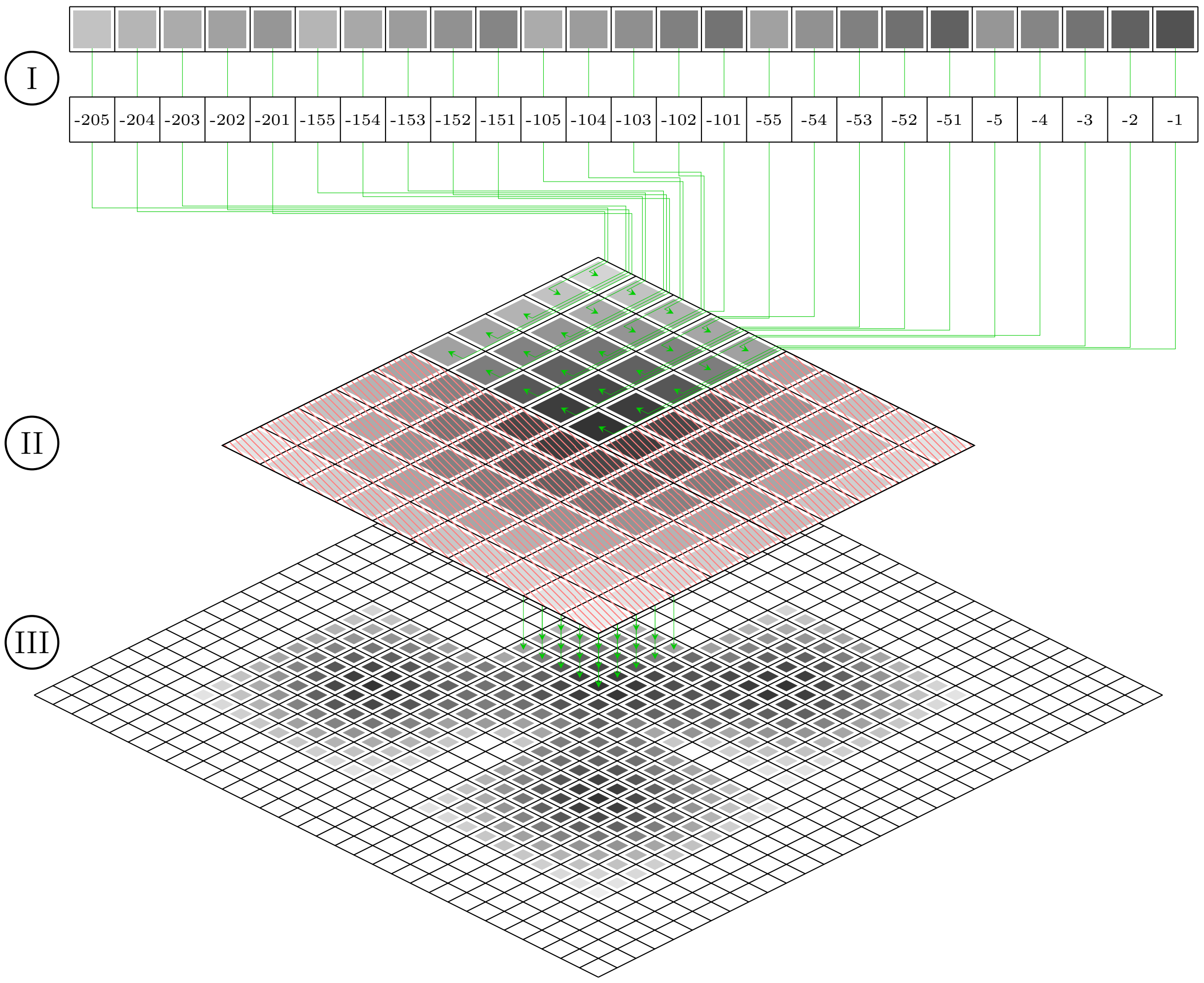


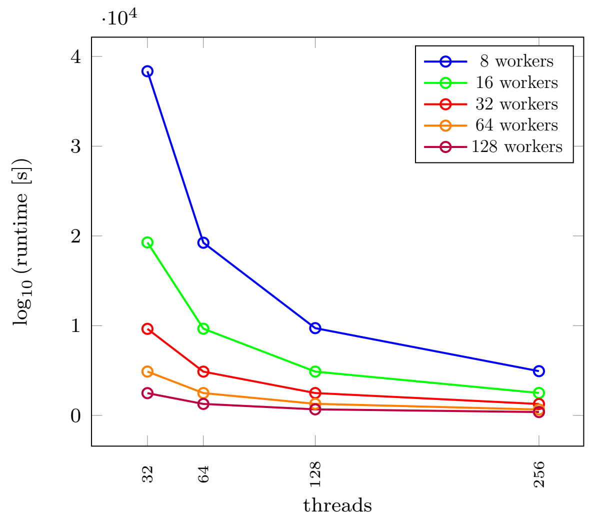
Figure 5.9: Illustration of the relation of geometric space and memory space in the density sampling kernel

Environment, Tests and Benchmarks

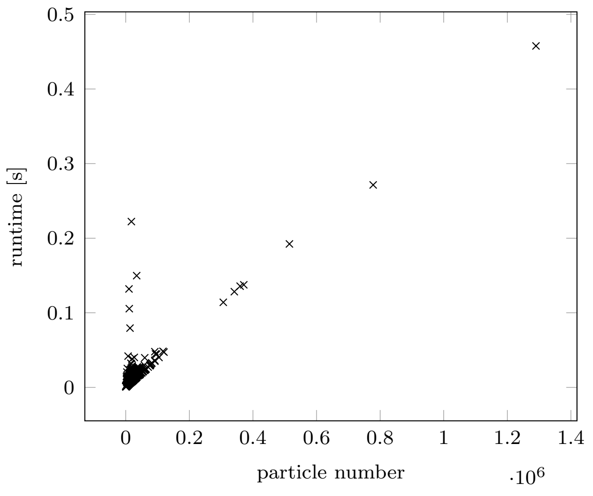
In this section I will describe how the implementation described above are embedded in a given computational environment. I will first describe the user perspective and following that the hardware embedding.
The user has again two input options, depending on the user's level of expertise and intentions:

- The command line interface. This option provides one functionality:
  `sample pdb_file.pdb density_file.mrc resolution_parameter voxel_size`
  The in inputs are specified as follows.
  - `pdb_file.pdb`
    A file of the `pdb` format (see 3.2.1). The algorithm will read out the atoms of the `ATOM` and `HETATM` field type into a `particles` data structure.
  - `density_file.mrc`
    The output file in the `mrc` file format (see 3.2.3). The resulting file will be as big as the `pixel_size` parameter, the `resolution` parameter and the structure specified by the `pdb_file.pdb` dictate, but not bigger.
  - `resolution`
    This parameter defines the "resolution", it is proportional to the standard deviation of the utilized gaussian kernels (see 3.2.3). The bigger, the more "spread' ' the particles appear.
  - `voxel_size`
    Defines the length of a cube side of one voxel that holds a specific density value (see 3.2.2). The smaller the value, the more "fine grained" the `density` will be and also the bigger the resulting file.
- The developers interface. This offers an interface for use in `C++` source code. It offers an extended functionality compared to the first interface.
  - Other kernels than the gaussian can be chosen. Since the `stamp` data structure is simply another `density`, `density` density can be chosen.
  - Multiple final `densities` maybe be sampled in rapid succession, given the memory of the GPU device suffices.
  - Any subset of the initial particle set can be sampled in rapid succession to differing final `densities`. This might be useful to sample differing domains or subunits of an input structure.

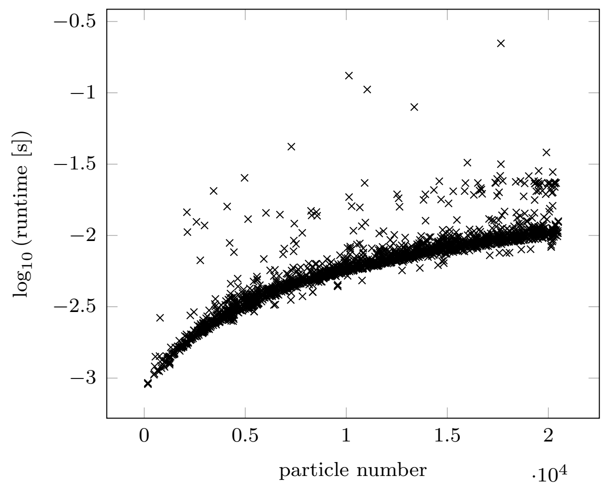
All of the described features have been tested in different circumstances. Visual inspection using the Chimera program was used that they indeed produce `densities` as expected. A more rigorous testing was conducted during the benchmarking. The benchmarking was set up in the following manner. To enable a wider coverage of cases a population of `pdb` structures has been selected and requested from the PDB database. The largest structures in the PDB database contain around 20000 atoms. For each of the atom number ranges $\{[0, 1000], [1000, 2000], \ldots, [19000, 20000]\}$ the first 100 entries have been downloaded and saved to local disk space. Every one of these 2100 structures has been sampled with three different `resolution` parameter settings: $\{5.0, 10.0, 15.0\}$. This is supposed to represent `resolutions` close to practical cases within the systematic fitting procedure. Again the optimal GPU device configuration regarding the parameters `blockDim` and `gridDim` has been sought. The measure of effectiveness is the time that the implementation takes to sample all of the 10500 different possible combinations of structures and `resolutions`. To choose an appropriate `pixel_size` the EMDB database's meta data has been consulted and the median of all pixel size has been found to be $1.35\mathring{A}$. This has been done to find the optimal configuration for a wide number of cases. The results are shown in figure (5.10a). The GPU device that performed the benchmark was a `NVIDIA Tesla V100`.
After the optimal configure of `(gridDim,blockDim) = (128,256)` was determined, further benchmarks have been conducted. A benchmark assessing the dependence of the number of `particles` has been conducted, the `resolution` parameter has been held at a constant 5.0. The results are shown in figures (5.10b) and (5.10c).
To assess the possible effects of particle clustering on the sampling speed the coefficient $\frac{\text{number of particles}}{\text{bounding box volume}}$ has been calculated for every structure in the test database. The two extremes have been found to be $5.615 \cdot 10^{-5}$ and 1.335 belonging to the PDB database entries `1Q55` and `2KU2`, respectively. The former is a rather extended structure, the latter an NMR ensemble. Both have been sampled for `resolution` parameter choices between 5.0 and 20.0.



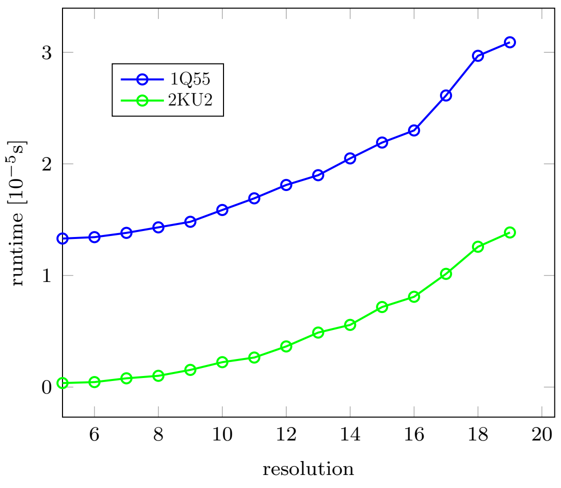
(a) Worker/threads density sampling



(b) Particle variation



(c) Particle variation, range up to 20000

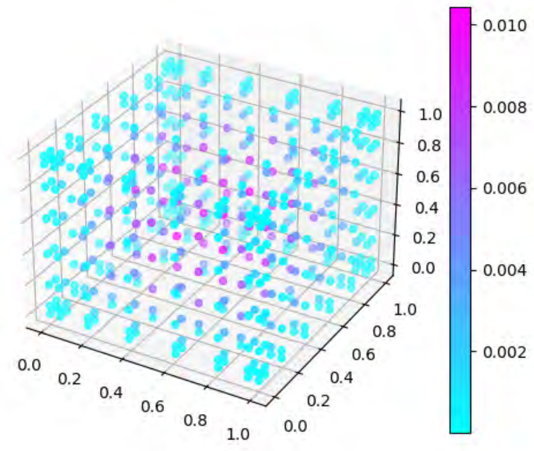


(d) Different particle distributions

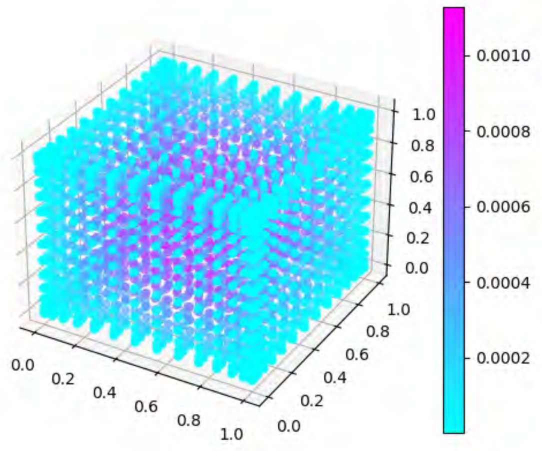
Figure 5.10: Benchmarks of the density sampling implementation

5.5 The transformational sampling scheme

This section concerns itself with the transformational sampling scheme that was developed for this thesis. As outlined in 2.5.5 one of the two options for transformational sampling is a regular grid. Together with the requirement that the scored transformations must enable quadrature (see 4.2.4 and 4.2.5) a sampling scheme has been developed and implemented. This sampling scheme is independent from any scoring function and therefore treated separately. This sampling scheme enables the calculation of 6-dimensional integrals, such as entropy, to an arbitrary level of precision.



(a) three-dimensional Chebyshev nodes for $N = 8$.



(b) three-dimensional Chebyshev nodes for $N = 16$.

Figure 5.11: Visualization of the translational nodes for two degrees of refinement. The colors encode the weights w_{ijk}.

5.5.1 The translational grid

The translational grid is constructed by applying the standard rule of construction for such cases. Given the Chebyshev nodes and weights for a degree of exactness N:

$$x_n^N = \{x_0, x_1, \ldots, x_{N-1}\}$$
$$w_n = \{w_0, w_1, \ldots, w_{N-1}\}$$

The three-dimensional grid and weights generated by these nodes and weights is then

$$x_{ijk}^N = \{(x_i, x_j, x_k) | 0 \leq i, j, k \leq N-1\}$$
$$w_{ijk}^N = \{w_i \cdot w_j \cdot w_k | 0 \leq i, j, k \leq N-1\}$$

The condition of the possibility of refining the grid can be fulfilled by choosing a sequence of Ns that are powers of 2:

$$\{N_0, N_1, \ldots\} = \{2^1, 2^2, \ldots\}$$

Since

$$x_{ijk}^N \subset x_{ijk}^{2N} \subset x_{ijk}^{2^2N} \subset \ldots$$

so that in case of a refinement the values corresponding to a less refined grid can be harnessed in a more refined grid. Figure (5.11) illustrates two example sets of translational

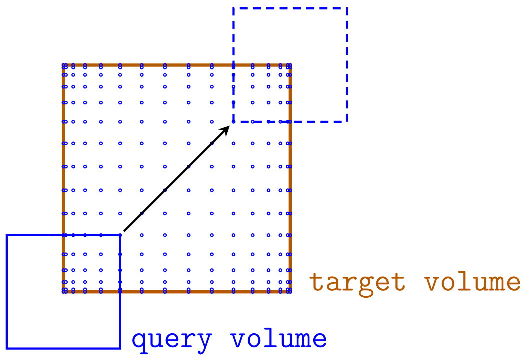


Figure 5.12: Convention for the translational sampling of shifts of query versus target volume, rendered in two dimensions. The blue dots represent different sampling nodes.

nodes. Given these rules, every further level of refinement has $2^3 = 8$ times the number of nodes of the preceding level. The nodes by themselves need to be scaled to cover a specific volume, the correct normalization factor for this case can be found in (4.29). Since these translational nodes are to be used for quadrature to measure metrics pertaining to the whole 6 dimensional sampling and these metrics in turn are to be used to compare different sampling scenarios (e.g. parameter choices, scores, structures), the scaled version of the nodes should be the same for a given target density, lest the different results cannot be compared anymore. This requires a convention regarding the translational space of target and query density that is independent from the geometric shape of the query density. Figure (5.12) illustrates this convention.

5.5.2 The rotational grid

As outlined above the rotational nodes of choice are the McEwen-Wiaux nodes, defined as

$$\alpha_a = \frac{2\pi a}{2M-1}, \text{ with } a \in 0, 1, \ldots, M-1 \tag{5.1}$$

$$\beta_b = \frac{\pi\,(2b+1)}{2L-1}, \text{ with } b \in 0, 1, \ldots, L-1 \tag{5.2}$$

$$\gamma_g = \frac{2\pi g}{2N-1}, \text{ with } g \in 0, 1, \ldots, N-1 \tag{5.3}$$

Different choices of M, L, N will result in different sets of nodes. The higher their values, the more fine-grained the rotational grid is. To enable refinement it is sufficient to double M and N, yet this is not possible for L. To guarantee that the nodes of a more refined grid contain the nodes of all less refined grid, the following refinement procedure has been

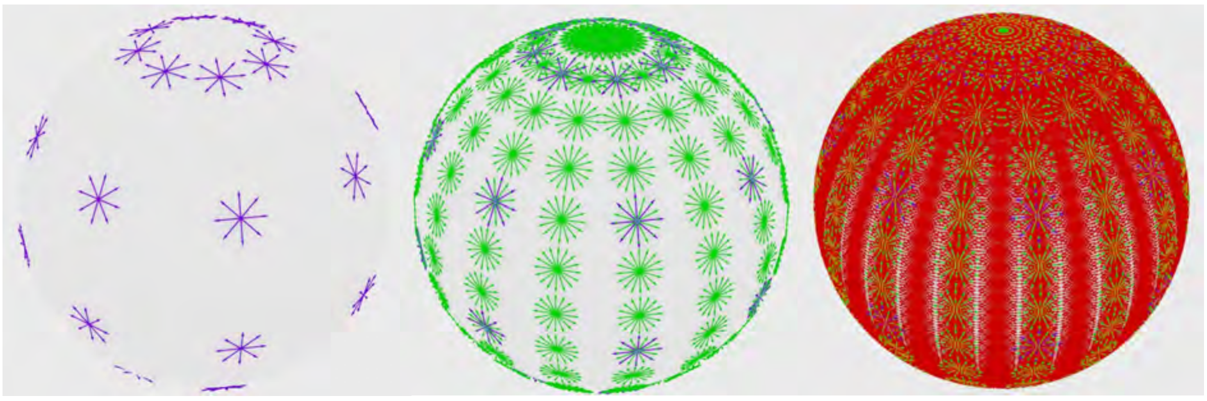


Figure 5.13: Three different degrees of refinement, all nodes in a grid left of the shown grids are contained in that grid. The subdivision tuples are, from left to right, $[8, 5, 8]$, $[16, 14, 16]$ and $[32, 41, 32]$.

defined:

$$\begin{aligned} M &\longrightarrow 2 \cdot M \\ L &\longrightarrow \frac{(3 \cdot L - 1)}{2} + 1 \\ N &\longrightarrow 2 \cdot N \end{aligned}$$

By such a procedure one can obtain the refinement sequence $[M, L, N]$ of

$$[2, 2, 2] \longrightarrow [4, 5, 4] \longrightarrow [8, 14, 8] \longrightarrow [16, 41, 16] \longrightarrow \ldots$$

Figure (4.15b) shows

5.5.3 The combined grid

A combined grid for 6-dimensional coverage has been defined. Given two grids, one translational and one rotational, and the nodes of the respective grids,

$$\{t_0, t_1, t_2, \ldots, t_T\}$$
$$\{r_0, r_1, r_2, \ldots, r_R\}$$

with T and R being the node volume of the translation and rotational grid, respectively. Using the Cartesian product of both sets, define:

$$\begin{aligned}
&\{(t_0, r_0), (t_0, r_1), (t_0, r_2), \ldots, (t_0, r_R),\\
&\ (t_1, r_0), (t_1, r_1), (t_1, r_2), \ldots, (t_1, r_R),\\
&\ \vdots\\
&\ (t_T, r_0), (t_T, r_1), (t_T, r_2), \ldots, (t_T, r_R)\}
\end{aligned}$$

as the nodes and,given the quadrature weights

$$\begin{aligned}
&\{w^t_0, w^t_1, w^t_2, \ldots, w^t_T\}\\
&\{w^r_0, w^r_1, w^r_2, \ldots, w^r_R\}
\end{aligned}$$

define the weights.

$$\begin{aligned}
&\{w^t_0 \cdot w^r_0, w^t_0 \cdot w^r_1, w^t_0 \cdot w^r_2, \ldots, w^t_0 \cdot w^r_R,\\
&\ w^t_1 \cdot w^r_0, w^t_1 \cdot w^r_1, w^t_1 \cdot w^r_2, \ldots, w^t_1 \cdot w^r_R,\\
&\ \vdots\\
&\ w^t_T \cdot w^r_0, w^t_T \cdot w^r_1, w^t_T \cdot w^r_2, \ldots, w^t_T \cdot w^r_R\}
\end{aligned}$$

This results in a 6-dimensional grid of total volume $R \cdot T$.

5.5.4 Sampling node memory layout

The mapping of a specific score s_i to a memory address m_i is a ubiquitous task and is complicated by the existence of iterative refinement. A reallocating and copying of values slows performance unnecessarily and clutters the source code. Additionally a lack of convention makes a structured reference to scores next to impossible. Such a mapping scheme has been devised and implemented. It is independent from the score type and can be extended to intermediate results, such as in the case of the cam-score (see 5.7.1). Generally, one would like two mappings

$$\begin{aligned}
\mu &: i \longrightarrow m_i\\
\tau &: i \longrightarrow (\mathbf{t}_i, \mathbf{r}_i)
\end{aligned}$$

with i ranging from 0 to `transformation_volume - 1`, so that one can consistently refer to both the scores or intermediates and the transformations that are connected to them. The implemented scheme has the following properties:

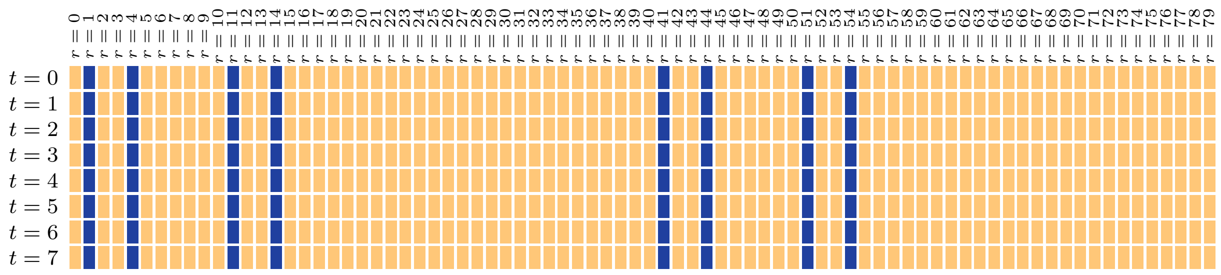


Figure 5.14: Illustration of the memory layout of a grid with a translational refinement level of 2 and the two rotational refinement levels of [2,2,2] and [4,5,4].

1. It enables arbitrary degrees of translational refinement.
2. It enables degrees of rotational refinement up to a maximum degree.
3. It is contiguous in memory and easily adopted to varying tasks.

Figure (5.14) shows an example of a small memory layout. The memory layout is to be understood as a contiguous linear memory section. The linear address of a given ith transformation can be calculated like so:

$$
\begin{aligned}
&m_i = r_i \cdot R + t_i \\
&\quad \text{with } R \text{ number of rotations} \\
&t_i = t_i^x + t_i^y \cdot T_x + t_i^z \cdot T_x \cdot T_y \\
&\quad \text{with } (t_i^x, t_i^y, t_i^z) \text{ the 3D-index of a translation} \\
&\quad \text{and } T_x, T_y \text{ the number in } x \text{ and } y \text{ direction} \\
&n = R_{ref} - s - 1 \\
&\quad \text{with } R_{ref} \text{ the total rotational refinements and } s \text{ the current level} \\
&r_i = r_i^x \cdot 2^n \cdot L_m \cdot N_m + r_i^z \cdot 2^n \cdot L_m + r_i^y \cdot 3^n - \frac{1-3^n}{2} \\
&\quad \text{with } L_m \text{ and } N_m \text{ the respective indexes of the maximal} \\
&\quad \text{rotational refinement level and } (r_i^x, r_i^y, r_i^z) \text{ the discrete index of the rotation.}
\end{aligned}
$$

The translation index (t_i^x, t_i^y, t_i^z) can be translated into a translation by accessing the respective nodes (such as Chebyshev nodes), similarly the rotational index (r_i^x, r_i^y, r_i^z) corresponds to a rotation specified by three parameters, such as the Euler angles (see 4.2.3) and a set of nodes such as the McEwen nodes (see 4.36)). The rotation and translation indexes are be derived by the standard reverse linearization procedure as outlined in 3.9. For purposes of quadrature the indexes can be mapped to the weights as given in 4.28 and 4.36.

5.6 Detection of local optimal scores

In many cases the modeler will evaluate the scoring distribution by assessing the local score maxima or minima. It is likely that the structures will placed near these optimal points during the model sampling step in the *Assembline* pipeline (see 2.5.3). For entropy measurements the optimal points are also important, since their surrounding region is likely to be information-rich. In the current implementation of *Assembline* the Chimera FitInMap tool is used, which itself uses a gradient descend method. The gradient is essential because without it one would rely on pure chance to find local optimal points and would have to ascertain their optimality in a different manner. Not all scores, such as the envelope score or the Chamfer distance (see 2.5.5) allow for the formulation of a gradient. This section introduces a so called pattern search algorithm and its implementation, which provide a possibility to find local optimal points independent of the nature of the scoring method.

5.6.1 General principle and implementation

The descend (or ascend) algorithm is based on a simple local optimization by adaptive exhaustion of the 6-dimensional search space and a quaternion based representation of the search space, which allows for an ease of movement within the search space.

The construction of the 6-dimensional search grid is based on an extension of the approach found in [75] and explained in 4.2.3. Figure (5.15) illustrates the following description. Four cubes are used to represent the rotational space and a further cube is used for the translational space. These five cubes can be easily used to construct global grids or local neighborhoods. Every pair of points consisting of one point within the translational cube and one point within either of the rotational cubes uniquely represents a translation coupled with a rotation. The value of this representation lies firstly within the fact that closeness within the cubes implies closeness within the 6-dimensional search space, secondly the easy computational representation of points in cubes and lastly the ease of conversion from the left hand side to the right hand side of figure (5.15). This conversion is detailed in 4.2.3.

The actual optimization is carried out by constructing small, adaptive, local search grids around initial positions. This is illustrated for a two dimensional example in figure (5.16). The brighter areas of the density in this figure represent more optimal values. At the start of the optimization an initial grid is constructed using the method outlined above. One of the initial grid positions is symbolized by the center blue circle in figure (5.16 **(A)**). Starting from this position, a local grid is constructed. In the 6-dimensional case the method outlined above is used again. In the illustration (5.16 **(A)**) this local grid

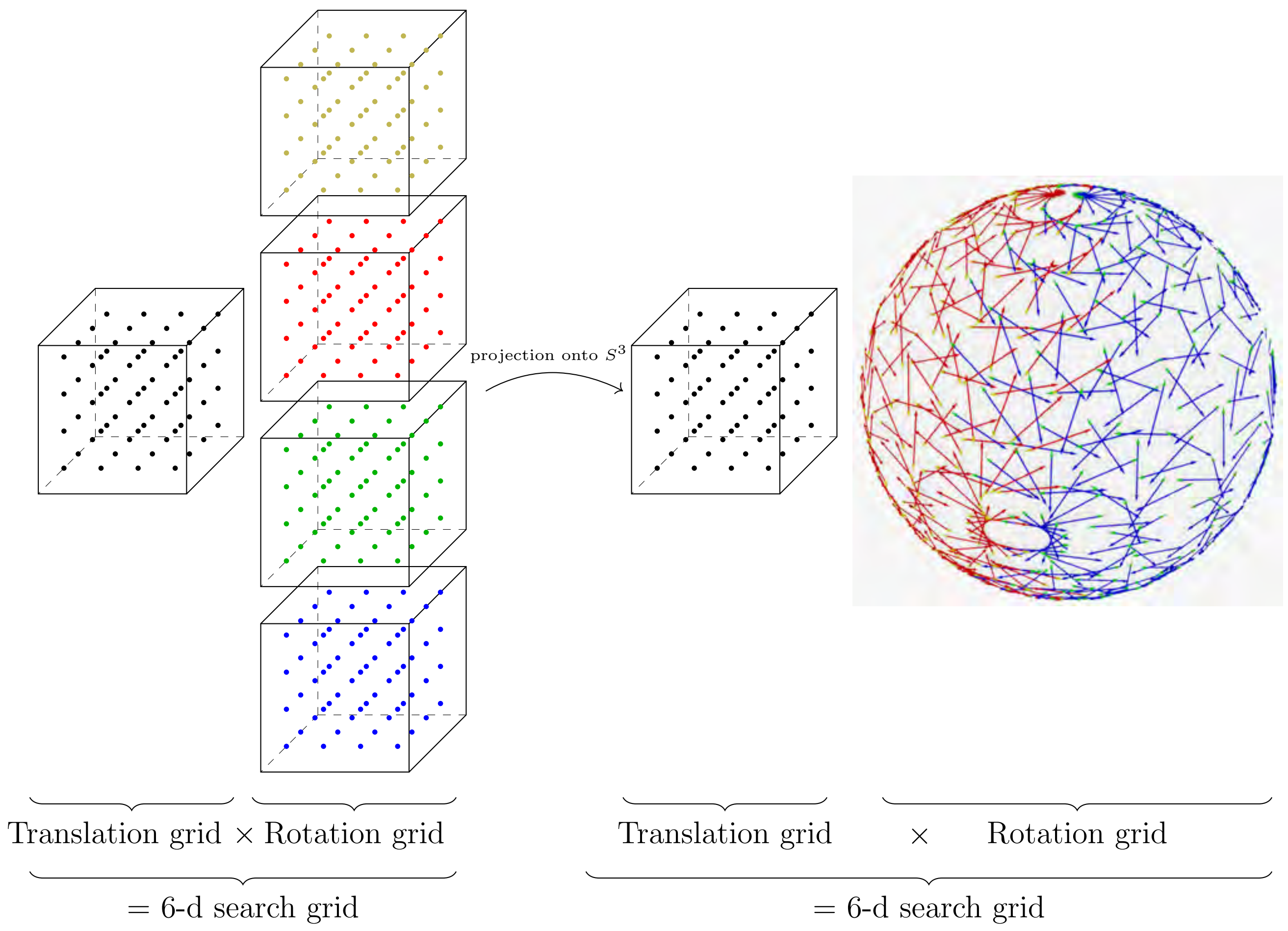

projection onto S^3
Translation grid × Rotation grid
= 6-d search grid
Translation grid × Rotation grid
= 6-d search grid

Figure 5.15: An illustration of the 6-dimensional search grid construction.

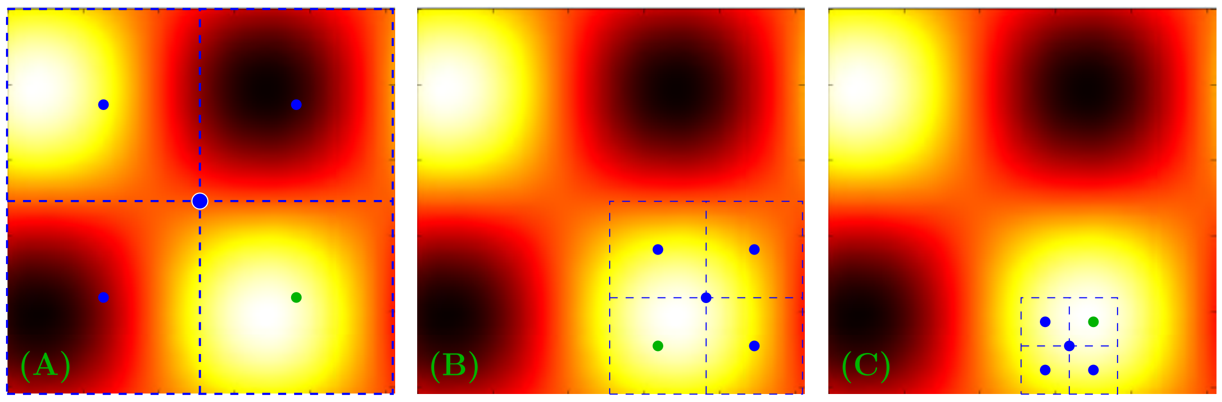


Figure 5.16: Illustration of the adaptive local grid search. Bright patches correspond to a better score.

corresponds to the 4 off-center blue points. The score is evaluated at all 5 points . The grid point with the optimal score (green) is selected to be the next initial position. In (5.16 **(B)**) another, smaller grid is constructed and the procedure is repeated. In (5.16 **(C)**), the green grid point is already quite close to one of the optimal points. The algorithm will terminate if the difference between two successive optimal approximations ist smaller than some preset value ε.
A number of observations are in order:

1. The initial position and the local adaptive grid effectively define a search volume. Its size is defined by the initial search grid. A more fine grained initial search grid will lead to smaller individual search volumes. The search grid adaption per iterative step is constructed in such a way that in principle every point within an individual search volume can be reached.

2. The algorithm will yield per initial position *either* a point at the edge of the individual search volume *or* an ε-approximation of <u>one</u> of the local optimal points. To ensure the approximation of all optimal points, one would need to choose a more fine grained initial grid.

3. The algorithm is independent from the scoring method, which can be plugged in at the appropriate place in the workflow.

4. The algorithm is independent from a predefined search grid in so far as that any given transformation (tranlsation and rotation) can in principle be reached.

5. Every dimension is halved per iteration, resulting in a rapid approximation rate. This is advantageous in a 6-dimensional space.

5.6.2 GPU implementation

Figure (5.17) depicts the workflow of the adaptive optimization algorithm. The core loop is run entirely on the GPU using dynamic parallelism (see 3.3.5). Within the loop the most important steps are the construction of local search grids as discussed above, the scoring and the evaluation of the scoring afterwards. Each of these steps is run for multiple instances simultaneously. At the end of each run, the convergence is assessed. If one particular search instance is seen to be convergent, it will not be updated anymore while the non convergent search instances are still being updated. This way a considerable speedup is achieved. If multiple GPUs are present in the system, the workload is split by dividing the set of initial search grid points.

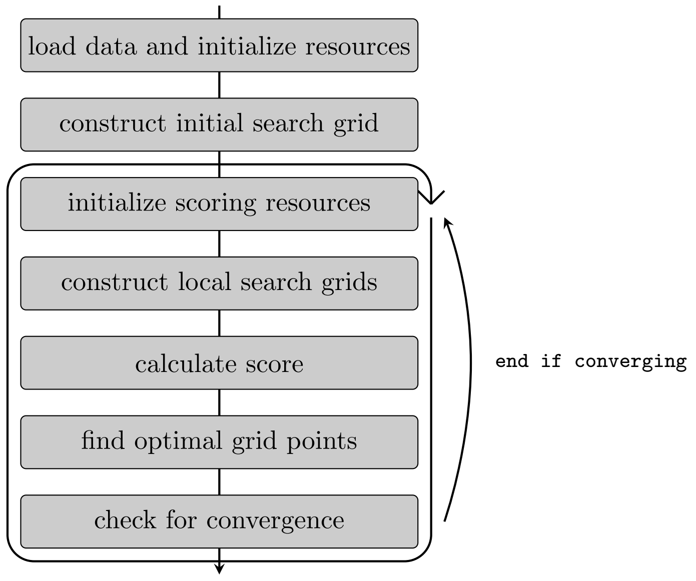

load data and initialize resources
construct initial search grid
initialize scoring resources
construct local search grids
calculate score
find optimal grid points
check for convergence
end if converging

Figure 5.17: Workflow of adaptive optimization algorithm.

5.6.3 User interface and benchmark

The implementation offers a command line interface:

```
cam_descend target.mrc query.mrc target_thresold
```

Further parameters that are currently hard coded and not exposed are:

- `query threshold` - density threshold for the query density.
- `M` - The degree of the initial rotational search grid.
- `d` - The maximal distance of grid points in the initial translational search grid.

The output of the program is a list of triples `(translation,rotation,score)`, the rotations encoded as quaternions (see 4.2.3). One of these triples will be output per initial search grid point. Each triple represents one possible local optimal point. The program automatically assess how many GPUs are present on the system and distributes the workload. Should there be multiple GPUs, multiple output files in the `csv` format will be created.
The program was tested on the TFIIIC data set (5.1). The target density map was a lower resolution version (annoted with 10 Å of the τ_A complex. A representative density map was created for the T131_TPR_5AEM using the density generator outlined above (5.4), the `resolution` parameter was set to `5.0`. The selected scoring method and implementation was the `masked cam score`, as outlined in (5.7.1). The initial parameters were set to

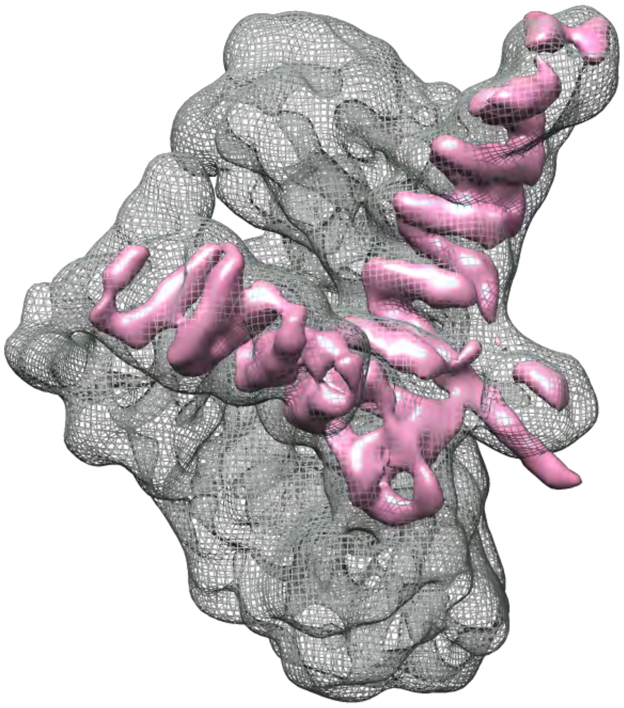


Figure 5.18: The top scoring transformation is applied to the query density and placed within the target density reference frame. The query density is shown in pink.

`M = 2, d = 30.0, eps = 0.001`, resulting in a number of `256` initial search grid points. The hardware the code was run on are 4 GPUs of the type `NVIDIA GeForce RTX 3090`. Figure (5.18) shows the top scoring solution, which is confirmed to be close to the actual position (see 5.1). The average time of the execution is $t_{average} = 1.86484$ s.

5.7 Parallel Implementation of scoring functions

The scoring functions (see 2.5.5) can be seen as the fundamental “measurement” of systematic fitting (see 2.5.5). They are calculated at least once per spatial (rotation and translation) sampling node (see 2.5.5). Some of them are computationally expensive and all of them are amenable to parallelization by dint of the underlying data types (see 3.2.2 and 3.2.1). This is the rationale for an implementation of scoring functions on parallel computation devices, which will be described in this section.

5.7.1 The masked correlation-about-mean score

The masked cam-score is one of the most utilized scoring methods (see 2.7) and is integrative to the *Assembline* pipeline. Also, the cam-score is a more sophisticated version of the cross-correlation score (see 2.5) and the overlap score (see 2.4), meaning an implementation of the cam-score can be turned into an implementation of the other scores by simplification of the source code. The implemented version of the cam-score is calculated by

$$\operatorname{cam}(\mathbf{q}, \mathbf{t}) = \frac{\langle \mathbf{q} - q_{ave} | \mathbf{t} - t_{ave} \rangle}{\|\mathbf{q} - q_{ave}\| \cdot \|\mathbf{t} - t_{ave}\|} \tag{5.4}$$

where the average refers to only to areas where the query density $\mathbf{q}$ is defined. Figure (5.19) shows an example situation of a query density situated at some position of a target density, for illustrative purposes these densities are one-dimensional. Equation (5.4) shows that two averages have to be calculated, subtractions, a scalar product and two norms.

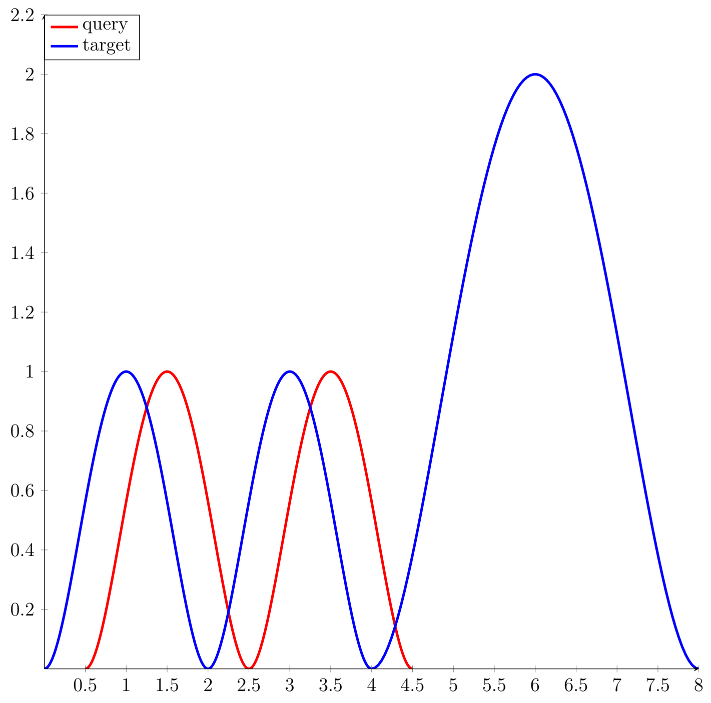


Figure 5.19: Two one dimensional illustrative densities, a target and a query.

Figure (5.21) elucidates some of the intermediate steps of the cam-score calculation as implemented. A few notes are warranted.

- The scalar product $\langle \mathbf{q} - q_{ave} | \mathbf{t} - t_{ave} \rangle$ and norms $\|\mathbf{q} - q_{ave}\|$, $\|\mathbf{t} - t_{ave}\|$ operations are normally modeled as sums. However, on a large enough, continuous volume, they can be seen as integrals and in turn approximated by numerical quadrature, as outlined in figures (5.21a, 5.21c and 5.21f). This has the advantage of being computationally cheaper. The values at the quadrature nodes (see 4.28) are calculated through use of texture memory (see 3.3.3), again for computational efficiency. The number of nodes will determine the "resolution" of the representation of the under-

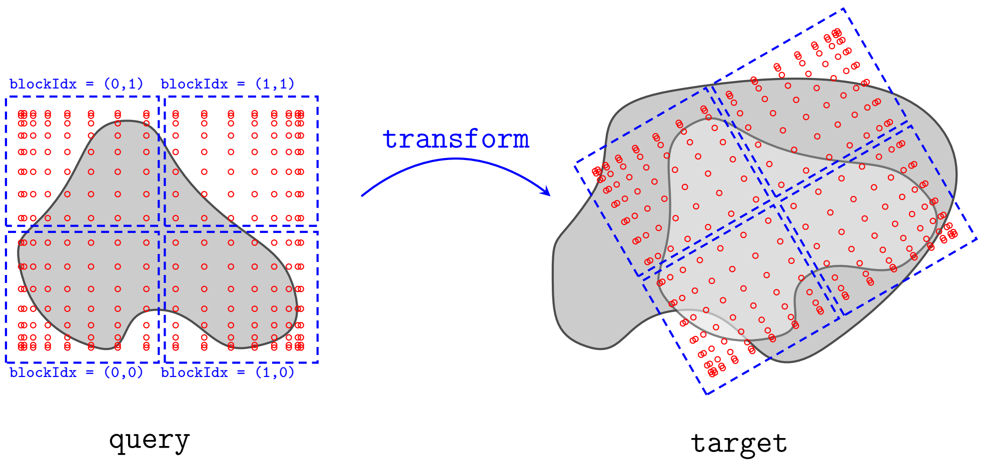


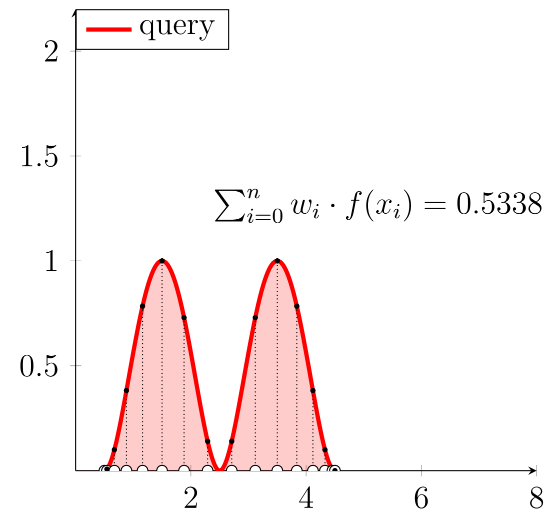
Figure 5.20: Depicted is the parallelization principle employed in the cam-score kernels. The red dots symbolize the Chebyshev scanning nodes, the dashed blue squares the `blockIdx`-sets that handle the single nodes. Every `kernel` in the cam-score implementation uses this scheme.

lying density. This method has the side effect of avoiding unnecessary information of oversampled densities.

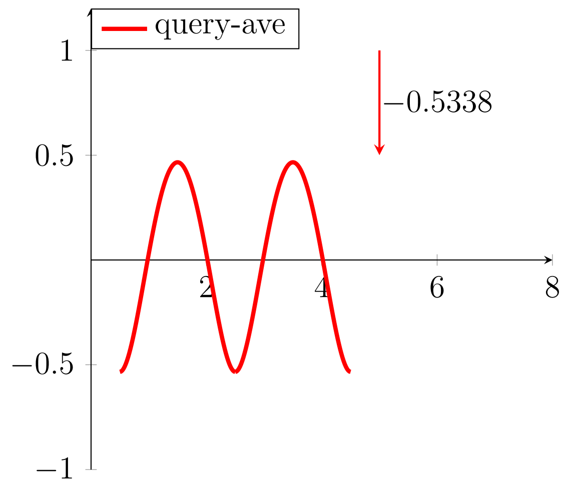
- The calculation of q_{ave} (cf 5.21a) has to be performed only once. After that, it is only needed to determine $\mathbf{q} - q_{ave}$ for the scalar product. It is therefore performed outside of the main loop.

- The average t_{ave} has to be calculated in the domain where the query density $\mathbf{q}$ is defined. Therefore, the "scanning" nodes, which evaluate the query density are the sole input, together with the specific transformation that is being scored and the texture object representing the target density $\mathbf{t}$, which is needed for the calculation of the value t_{ave}. This means that the scanning nodes and their respective transformations have to be accessed regularly. This suggests to save them in `register` memory, the fastest memory type, instead of `global` memory (see 3.3.1). Figure (5.20) shows the basic principle of this parallelization scheme.

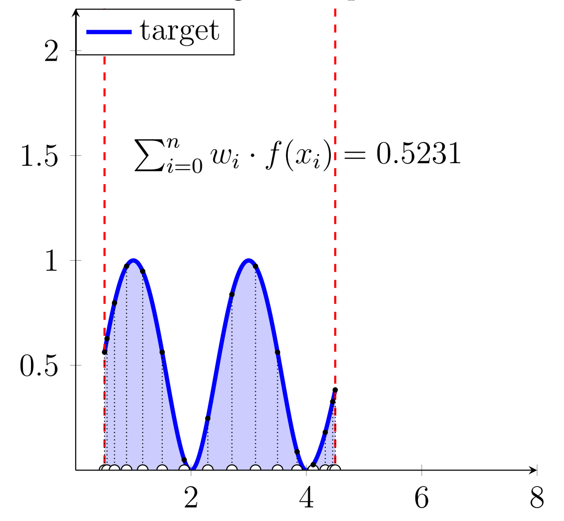
- Partition the Chebyshev scanning nodes into sets, and index these sets using the three-dimensional intrinsic variables `blockIdx.x`,`blockIdx.y`,`blockIdx.z` and `threadIdx.x`,`threadIdx.y`,`threadIdx.z` (see 3.3). Then, loop over all possible transformations within the kernel, for each transform create a copy of the scanning nodes to the appropriate target coordinates and calculate the *contribution* of these nodes to the quantities mentioned (scalar product, the norms) above and save them to the `global` memory using `atomicAdd` operations (see 3.3.4). When the kernel is run successfully, the contribution of each node will be accumulated at an appropriate place in `global` memory, the address of the place signifying the transformation it expresses.



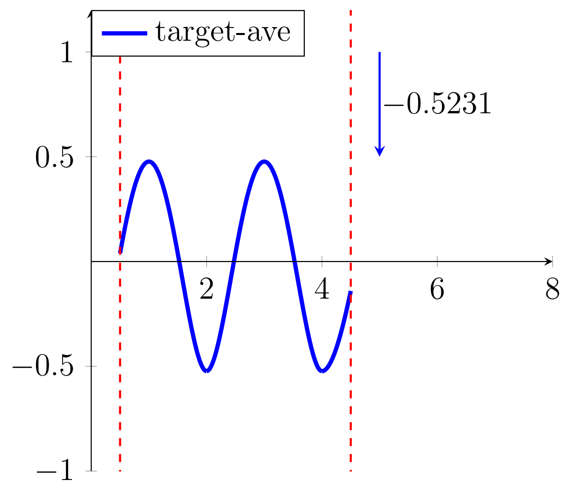
(a) Quadrature of the query to find the average value q_{ave}.



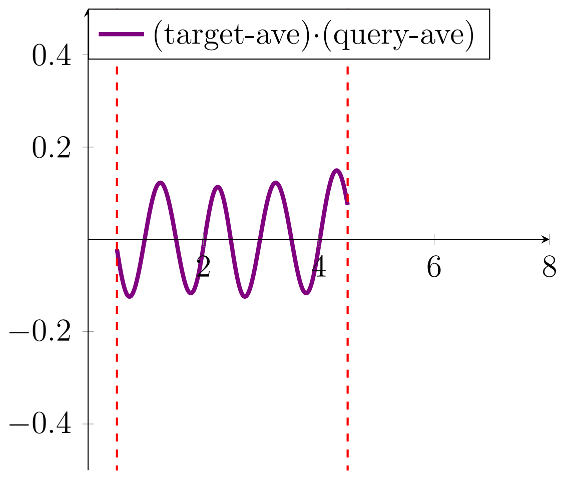
(b) Shifting of the query to obtain $\mathbf{q} - q_{ave}$.



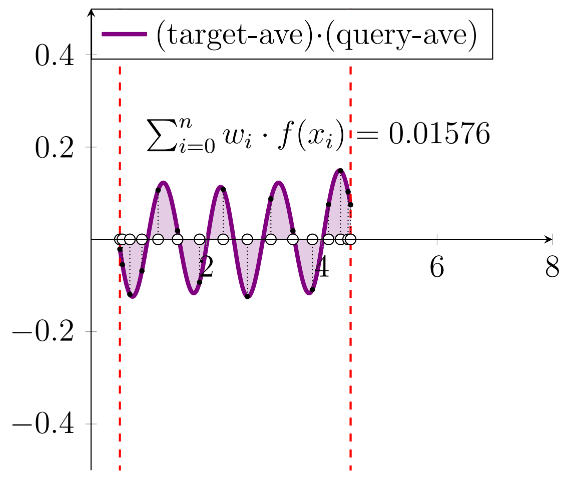
(c) Quadrature of the target in the domain where the query is defined to obtain t_{ave}.



(d) Shift of the target to obtain $\mathbf{t} - t_{ave}$.



(e) Forming $(\mathbf{t} - t_{ave}) \cdot (\mathbf{q} - q_{ave})$.



(f) Calculating $\langle \mathbf{t} - t_{ave} | \mathbf{q} - q_{ave} \rangle$.

Figure 5.21: Illustration of the calculation of the masked cam-score.

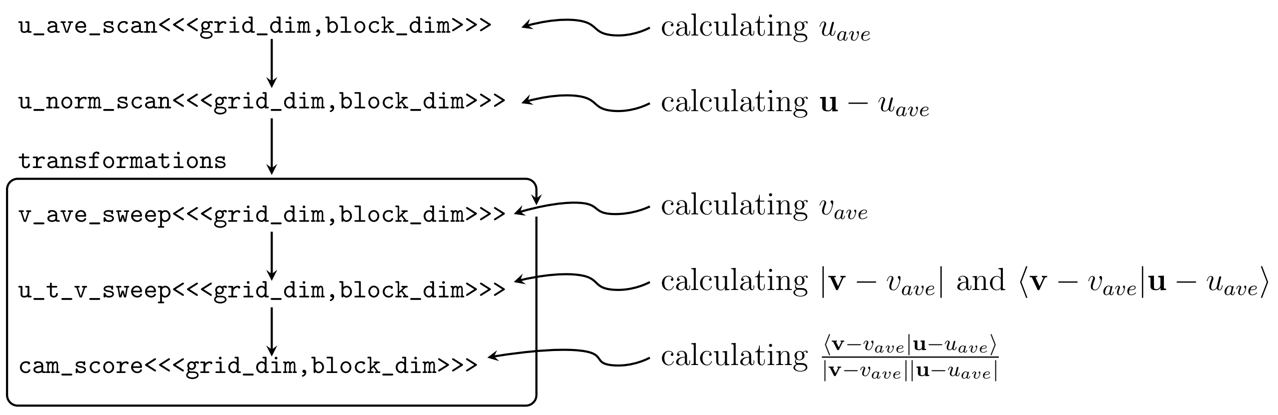

u_ave_scan<<<grid_dim,block_dim>>>
calculating u_{ave}
u_norm_scan<<<grid_dim,block_dim>>>
calculating $\mathbf{u} - u_{ave}$
transformations
v_ave_sweep<<<grid_dim,block_dim>>>
calculating v_{ave}
u_t_v_sweep<<<grid_dim,block_dim>>>
calculating $|\mathbf{v} - v_{ave}|$ and $\langle \mathbf{v} - v_{ave} | \mathbf{u} - u_{ave} \rangle$
cam_score<<<grid_dim,block_dim>>>
calculating $\frac{\langle \mathbf{v} - v_{ave} | \mathbf{u} - u_{ave} \rangle}{|\mathbf{v} - v_{ave}||\mathbf{u} - u_{ave}|}$

Figure 5.22: Call pattern of the implementation of the cam-score.

- As figure (5.19) indicates, the quantities needed to calculate the cam-score partially build on each other, so they have to be calculated in sequence. One either calculates these quantities in sequence *per transformation* or in a parallel manner for all transformations at the same time. As hinted at in the last bullet point, this is the approach taken here.

- The Chebyshev scanning nodes might be scaled to any volume necessary (see 4.29. The choices for texture fetching (cf 3.3.3) are is between $[0, 1] \times [0, 1] \times [0, 1]$, which would amount to the use of normalized coordinates, or pixel dimension coordinates. The former choice does not work well with rotations and is disregarded. Initially, the nodes are shifted to be in the range $[-0.5 \cdot \text{pixeldim}, 0.5 \cdot \text{pixeldim}]$, so that the rotational part of the transformation can be carried out around the origin (see 4.2.3).

Figure (5.22) show the complete call structure of the cam-scoring implementation. The kernels operate in a very similar manner, figure (5.23) shows an representation of such a typical kernel layout. The other kernels differ only in certain details, so their schematics are omitted. The use of `register` memory for the much-used nodes and of `shared` memory for the transformations stand out. The latter are saved to `shared` memory since they need to be accessed by multiple `threads`. The transformation grid is iterated by two indexes, `r` for rotations and `t` for translations. The latter are situated in the innermost loop, since a translation is computationally less intense than a rotation. The indexes `r` and `t` can be uniquely mapped to specific rotations and translations respectively, the double-loop structure effectively creates the Cartesian product of both indexes, resulting in a transformation grid. Each `thread` handles a specific Chebyshev scanning node, there are two more `float3` data structures per `thread` allocated in the `register` memory, to be able to keep the original node coordinates and avoid retrieving it again from `global` memory.

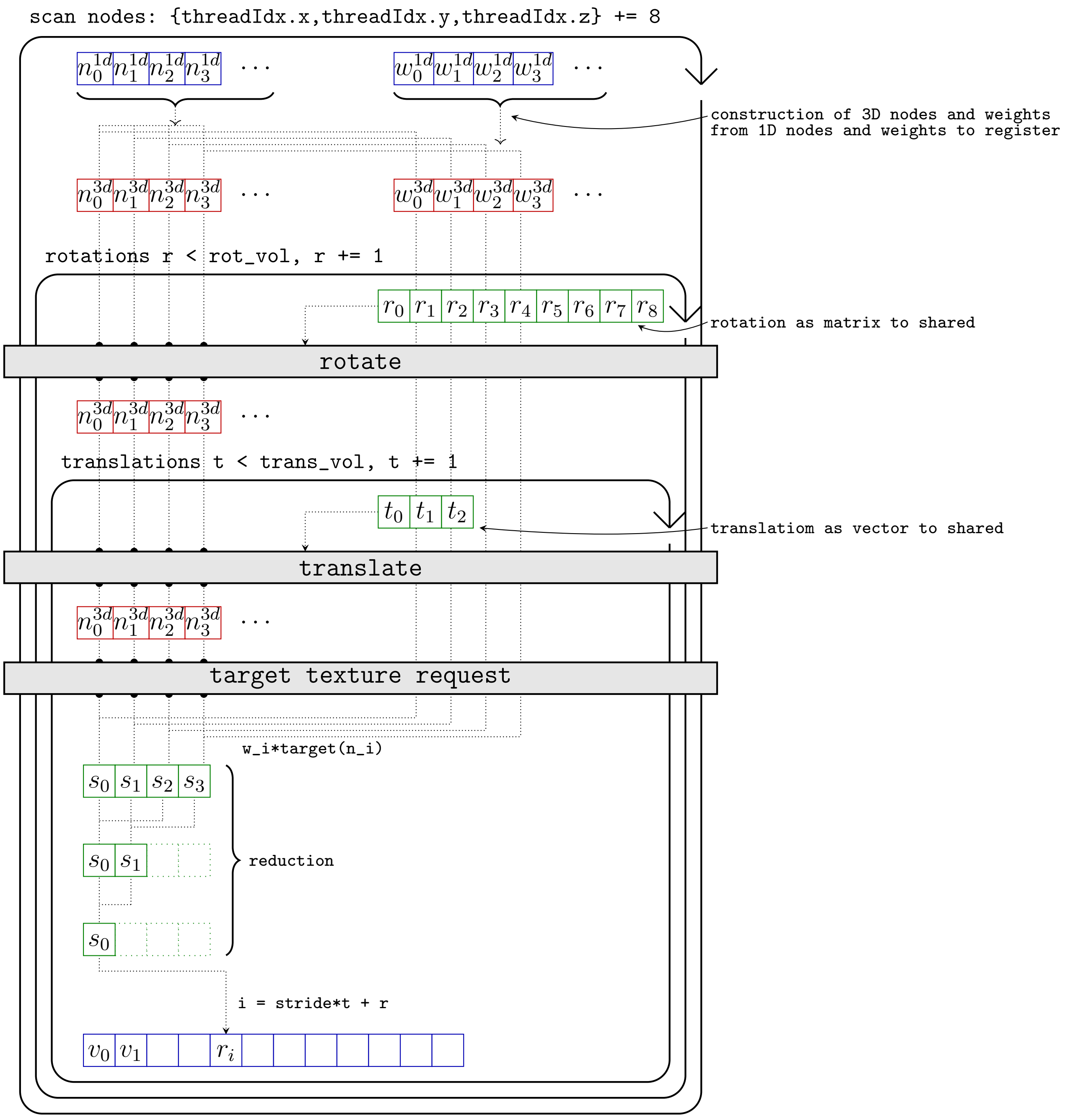

scan nodes: {threadIdx.x,threadIdx.y,threadIdx.z} += 8
n_0^{1d} n_1^{1d} n_2^{1d} n_3^{1d} ...
w_0^{1d} w_1^{1d} w_2^{1d} w_3^{1d} ...
construction of 3D nodes and weights from 1D nodes and weights to register
n_0^{3d} n_1^{3d} n_2^{3d} n_3^{3d} ...
w_0^{3d} w_1^{3d} w_2^{3d} w_3^{3d} ...
rotations r < rot_vol, r += 1
r_0 r_1 r_2 r_3 r_4 r_5 r_6 r_7 r_8
rotation as matrix to shared
rotate
translations t < trans_vol, t += 1
t_0 t_1 t_2
translatiom as vector to shared
translate
target texture request
w_i*target(n_i)
s_0 s_1 s_2 s_3
reduction
i = stride*t + r
v_0 v_1 r_i

Figure 5.23: Schematic of the cam-score v_{ave}-kernel. Blue denotes `global`, green `shared` and red `register` memory.

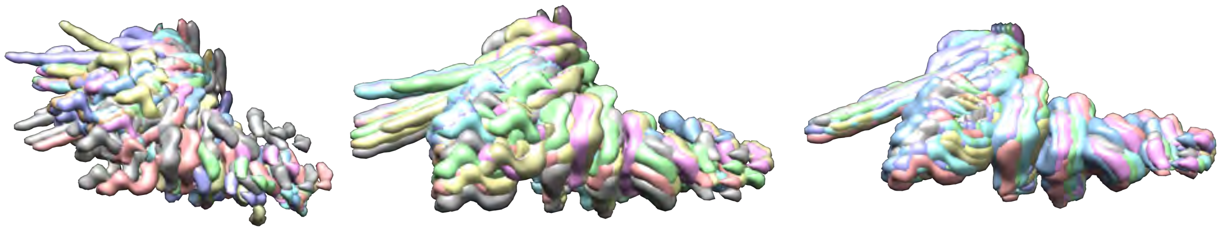
User interface

Two command line interfaces and one visual evaluation tool have been implemented. The command line interfaces mirror the basic usage needs of the modeler engaged in systematic fitting. The respective interfaces are

```
./cam_fit target_file query_file.pdb output_file.csv target_threshold
          resolution solution_num transl_refine rot_refine scan_refine
./cam_fit target_file query_file.mrc output_file.csv target_threshold
          solution_num transl_refine rot_refine scan_refine
```

Both interfaces are visibly similar, the first serves the fitting of `pdb` and the second of `mrc` files. The parameters are to be interpreted in the following manner:

1. `target_file`: This must be any existing `mrc` file. The file specified by the query will be fitted systematically against this file.
2. `query_file.pdb` and `query_file.mrc`: This must be any existing `mrc` or `pdb` file. This file will be fitted systematically against the target file.
3. `output_file.csv` This file will contain the top scoring results. It will also contain the paths to both target and query file. The results are lines of the form `t0,t1,t2,v0,v1,v2,a,s`, all `floats`. The first three numbers specify the translation in accordance with the convention outlined in (5.5.1). The next four numbers specify the rotation in the axis-angle representation (see 4.2.3), the angle being specified in terms of radians. The last number specifies the score.
4. `target_threshold`: The density threshold used for cropping of the target volume. Since systematic fitting is more efficient in a smaller search space it is advisable to crop the target density. This parameter specifies the density value below which other density values can be ignored and the target density will be cropped so that none of these minor values is part of it anymore. Since the data represents an electron density, 0.0 is always a valid choice.
5. `resolution`: This `float` parameter is only applicable in the case of a `pdb` query file. It determines the spread of the Gaussians used in modeling a density from a set of atoms (see 3.2.3).
6. `solution_num`: The maximum number of solutions that will be written to the `output_file.csv`.



(a) Refinement level (4,3). (b) Refinement level (5,4). (c) Refinement level (6,3).

Figure 5.24: The output of increasing refinement levels exemplified.

7. `transl_refine`: The degree of refinement of the translational shifts, a higher number will produce a finer grid in the sense of (5.5.1).

8. `rot_refine`: The degree of refinement of the rotational "shifts", a higher number will produce a finer grid in the sense of (5.5.2).

9. `scan_refine`: The degree of refinement of the Chebyshev nodes used to "scan" the target and query density. A higher number will result in a more faithful representation yet might also lead to oversampling.

For direct visual inspection there is an option which will output the `PDB` files which correspond to the transformations of the top fits in `output_file.csv`. Figure (5.24) depicts some of the results of the consistency test (see 5.26).

Benchmarks

A number of benchmarks have been completed to gain a general idea of the performance and the functionality of the implementation. Figure (5.25) depicts a speed test, the measure of speed being $\frac{\#\text{transformations}}{\text{second}}$. The varied value on the x axis is the level of refinement of the scan nodes, which determines the degree of detail that is being assessed. This benchmark has been run on 8 `GeForce RTX 3090` devices and the framework to submit the workloads to the devices was as described in (5.3). The constance of the first three degrees of refinement stands out, suggesting no loss of performance up to the fifth degree and therefore setting a reasonable default.

Figure (5.26) shows the consistency of the cam-score. A random, real, non-symmetric density map of a structure has been chosen and has been fitted to itself with growing degrees of search refinement. A consistent score should always assign the best score for the identity transformations. Due to the nature of the McEwen rotational nodes (see 5.5.2) and the Chebyshev translational nodes (see 5.5.1) the identity solution ($\mathbf{t} =$

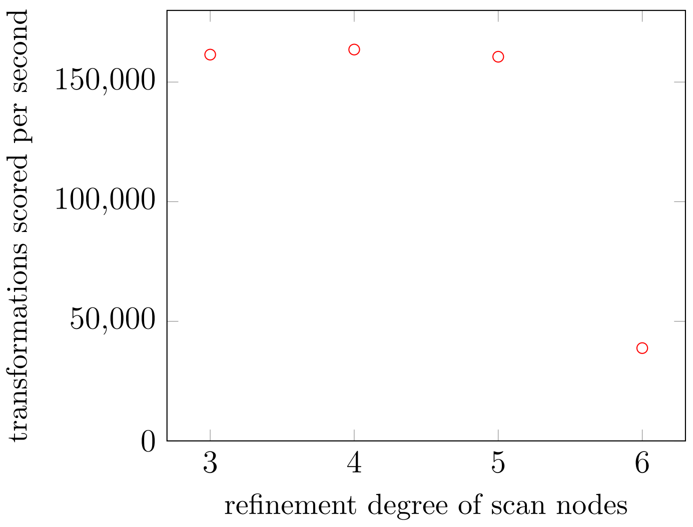


Figure 5.25: Speed benchmark with different degrees of scan node refinement.

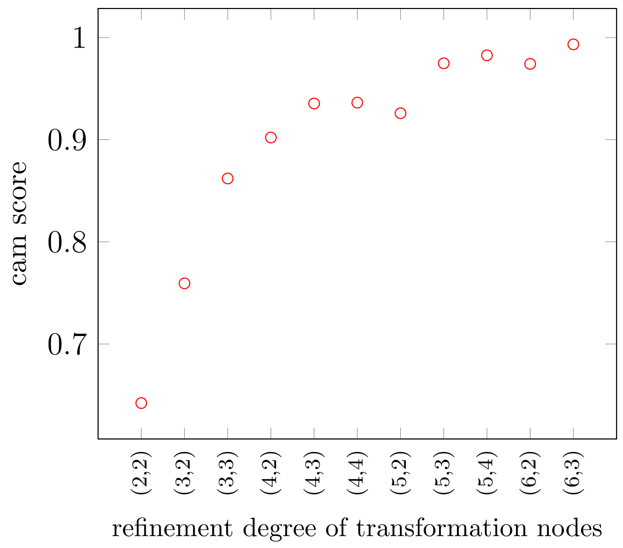
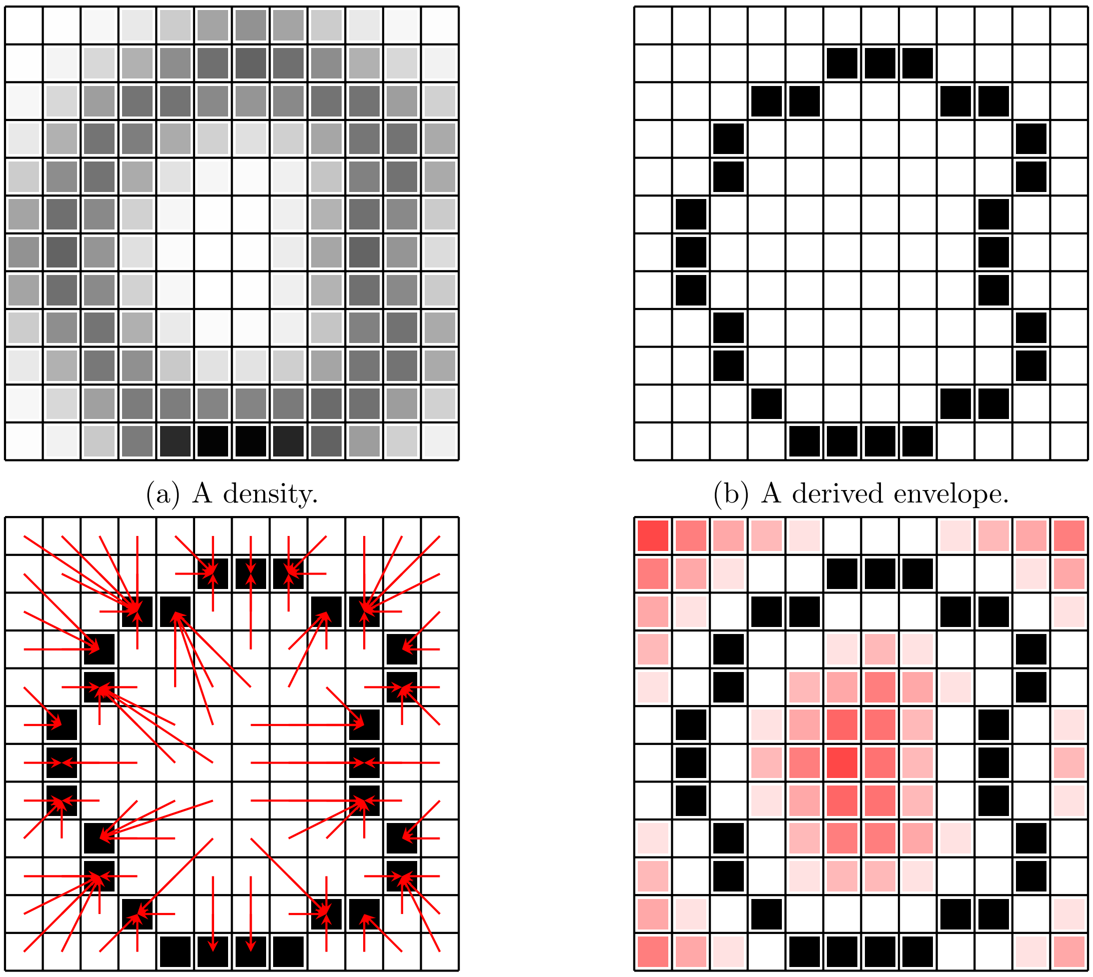


Figure 5.26: Consistency measurement for different degrees of rotational and translational search node refinement.


(a) A density.

(b) A derived envelope.

(c) Least distances.

(d) Least distances as density.

Figure 5.27: Illustration of the difference map procedure.

$(0.5, 0.5, 0.5), \mathbf{r} = (0, 0, 0))$ is not part of any sampling set. With growing search grid refinement degrees the cam-score approaches the best possible score, which is 1.0.

5.7.2 The chamfer distance score

A implementation of the Chamfer distance (see 2.5.5) as a score for systematic fitting has been accomplished. The transformational sampling and the process management are discussed in (5.5.1) and (5.3), respectively. Instead of calculating the minimum distance of each pixel belonging to one surface to any pixel belonging to another surface, as required by the definition of the Chamfer distance, an intermediate step has been taken to reduce computation time. This intermediate step uses the fact that the minimum distance of any pixel on the target map to the target surface does not change. Therefore it is possible to define a *difference map*, which is defined by the requirement that every pixel holds the distance to the nearest pixel on the surface of the original map. Pixels that are already

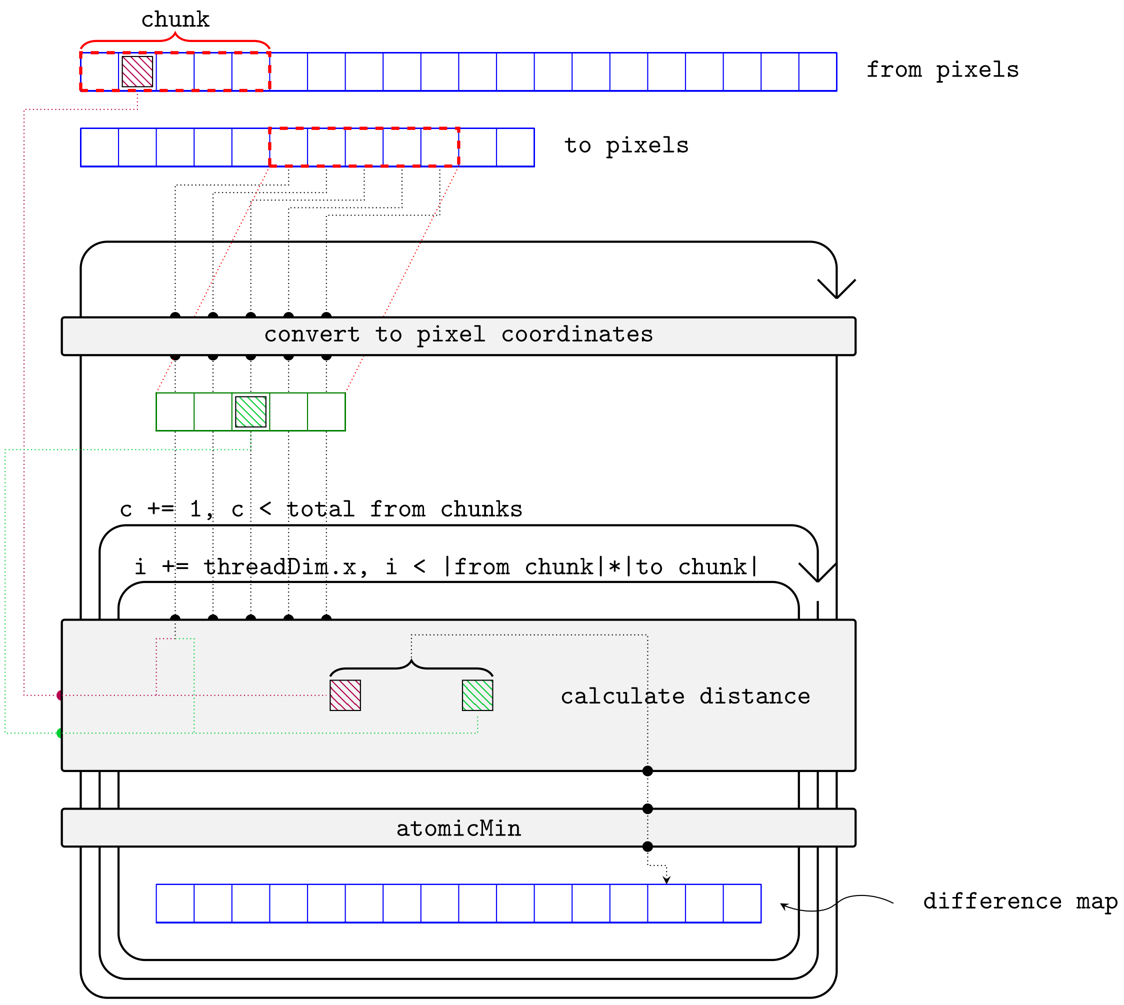
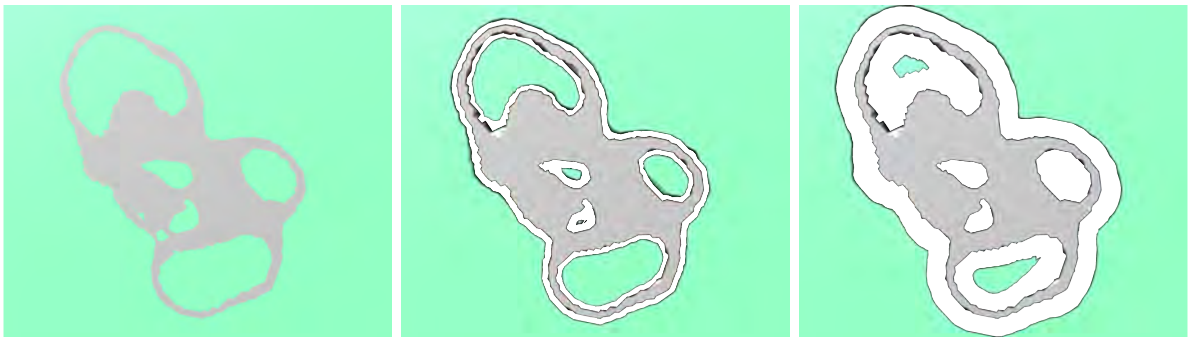

chunk
from pixels
to pixels
convert to pixel coordinates
c += 1, c < total from chunks
i += threadDim.x, i < |from chunk|*|to chunk|
calculate distance
atomicMin
difference map

Figure 5.28: Schematic of the difference map kernel.

on the surface hold the value 0.0. If another surface is now placed, under an arbitrary transformation, on the distance map, every pixel of that surface will map to a pixel on the difference map and the Chamfer distance can be calculated by adding up the values of the mapped-to pixels. In this way, the smallest distances are calculated once and once only independent of the number of query placements in the target map, resulting in an overall gain of computational speed.

Figure (5.27) shows the procedure of creating a difference map. A initial density map (5.27a) is turned into a surface (5.27b) by specifying two density thresholds $t_0 < t_1$ and selecting all pixels whose values fall in this range. Subsequently one calculates for every pixel the minimum distance to the next pixel belonging to that surface (5.27c) and assigns the magnitude of that distance to the original pixel (5.27d). A GPU kernel has been constructed and implemented to deal with this expensive operation. Figure (5.28) shows the schematic of this kernel, which operates the following way:

- After a surface has been defined, the pixels fall into two classes: The pixels belonging to the surface and the remaining pixels. The minimum distance to the next pixel



(a) No distance to surface. (b) Low distance to surface. (c) Higher distance to surface.

Figure 5.29: Slices of the same distance map and surface at different density threshold settings as rendered in Chimera.

on the surface of the former class is trivially 0. Additionally, all pixels of the latter class are not valid "target" pixels and do not need to be regarded as such. The kernel therefore starts with the linear indexes of these two classes of pixels. The linear index, together with the geometry of the density map (see 3.2.2) determine the geometric position of each pixel and therefore the distance between two pixels.

- Without further information about the intrinsic geometry of the surface all pixel not on the surface (`from_pixels`) have to be tested against all pixels on the surface (`to_pixels`). This is done by dividing both of these sets into `chunks`. The geometric position of a `chunk` of the `to_pixels` is written as `float3` data instances to the `shared` memory to reduce computational time. So every `block` of `threads` defined by this kernel holds a subset of the `to_pixels` and will iterate over a `chunk` of the `to_pixels`.
- This is what happens in the inner loop. For every `from_pixel` and every `to_pixel` (still in the `shared` memory) of their respective `chunks` the distance is calculated. An `atomicMin` operation is used to compare the distance values in the `difference map` memory range in `global` memory. This memory range has been initialized with `FLT_MAX` values, the maximum possible `float` number, so that it is guaranteed to be minimized by at least one value.

Figure (5.29) shows a negative stain density maps of the TFIII protein complex and a surface defined by the two thresholds 0.0395 and 0.07. Figure (5.29a), (5.29b) and (5.29c) show how the visible density values retract the higher the volume density cutoff is chosen. This corresponds to the fact that the difference map holds the minimum distances, which also can be seen by the fact that the isosurfaces of the difference map are parallel to the out layer of the surface.
Trivially the computation time varies with the surface size. A command line interface which produces surfaces and the distance map has been created for testing purposes:

```
diffmap input.mrc outfile_name.mrc t0 t1
```

The thresholds `t0` and `t1` correspond to the threshold mentioned above. `input.mrc` is the input density file and `outfile_name.mrc` is the location and name of the resulting difference map. Additionally, the file `out_file_name_surface.mrc` will be generated. It holds the surface defined by `input.mrc`, `t0`, `t1` and the procedure defined above.

User interface

The user interface of the Chamfer score fitting engine is a command line interface:

```
./chamfer_fit target_file query_file.pdb output_file.csv target_threshold_0
target_threshold_1 query_threshold_0 query_threshold_1 resolution
solution_num transl_refine rot_refine
```

The interface is similar to the interface of 5.7.1, with the added parameters of the respective thresholds for query and target.

Benchmarks

Extensive benchmarks have been undertaken. The following results record timing benchmarks which were conducted on 8 `GeForce RTX 3090`. The fitting task involved fitting a density map of `pixel dimension [90,54,90]` to itself. The defined surface consisted of `5.39%` of the total pixels. Figure (5.30) shows the results of those benchmarks. While the alternative - recalculating the distances for every transformation that is scored - is not tested, it can be said that with higher transformational sampling refinement the computational efficiency gain increases since the likelihood that pixels will be covered more than once increases also. On the same setup and the same test runs an accuracy test has been performed. The best possible result for a Chamfer score is 0. Figure (5.31) shows the Chamfer score for varying transformational refinement degrees. With greater degrees of refinement an convergence towards 0 is noticeable.

5.7.3 The envelope score

The envelope score has been implemented for a GPU environment. Due to the simplicity of the score and its implementation, only a short description of the latter will be given.

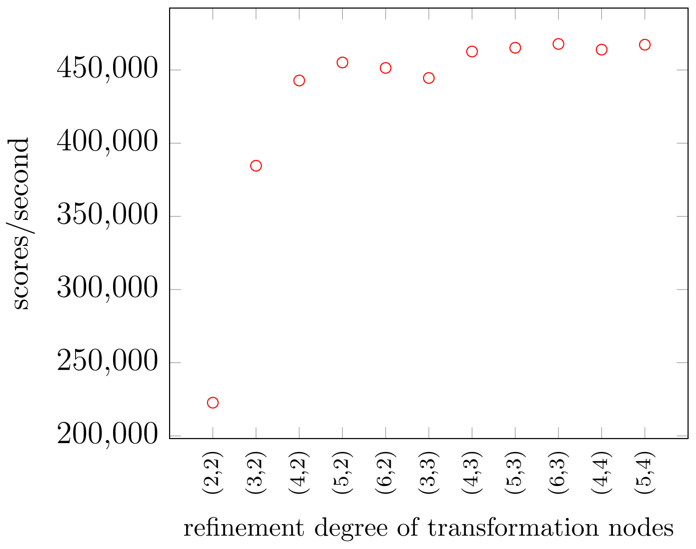


Figure 5.30: Scoring speed measurement for different degrees of rotational and translational search node refinement.

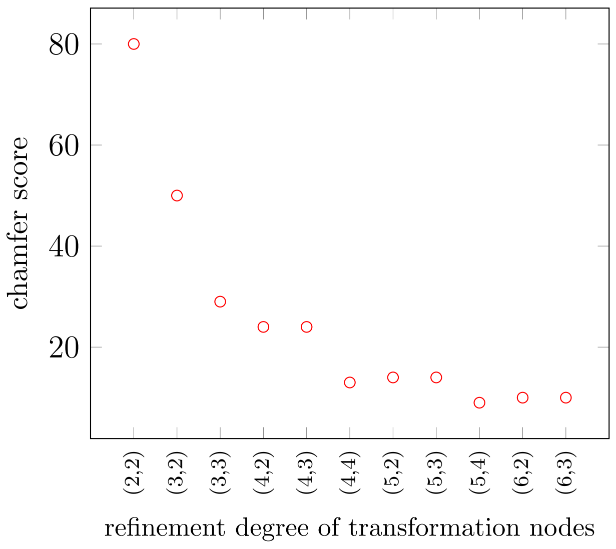
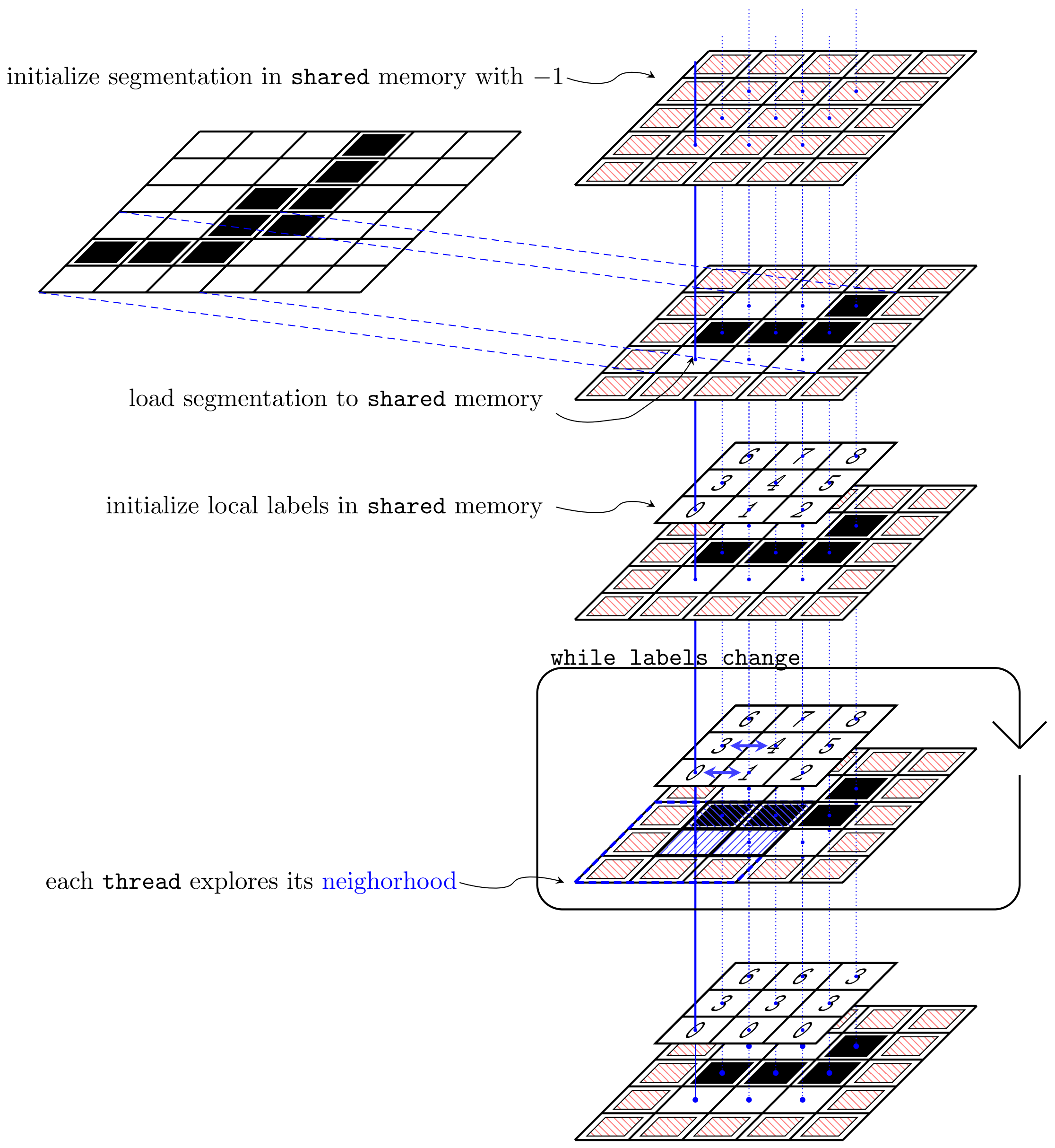


Figure 5.31: Scoring accuracy measurement for different degrees of rotational and translational search node refinement.

- Per definition this scoring method only works on queries that are represented as particles. This can be circumvented by interpretation of a the pixel coordinates of a density map as particles.
- After a threshold is defined for the target density, an “envelope” density can be defined. It has the same dimensions as the target density, its values however are `integer`s defined according to the rules laid down in 2.5.5.
- Due to the data type of the envelope density being `integer` and its values being within a very small range, this data structure can be allocated on the GPU multiple times without any problems. These multiple allocations can be regarded as different “workspaces”, where the envelope scores of different transformations can be calculated in parallel.
- For every transformation `t` the coordinates of the particles of the query will be transformed according to `t` and subsequently be “imprinted” on the envelope density by first mapping their transformed coordinates to the geometric volume of a given pixel within the envelope density and then changing the value associated with the pixel according to the rules laid down in 2.5.5.
- Due to the facts that one the one hand, typically the number of pixels will exceed the number of particles by far and on the other hand, the envelope density will have to be restored to its original state to score the next transformation, the indexes of the “imprinted upon” pixels are saved. This way, only these pixels have to be restored in the end.
- Once all particles have been imprinted this, the values of the envelope density are summed up and the result is written to global memory.
- Using the earlier saved indexes, the envelope density is restored to its original state.

5.8 Connected Components Labeling

A three-dimensional version of the two dimensional CCL-algorithm which was described in [66] in the form of pseudocode was adapted and implemented. The general `kernel` structure as described in 3.2.4 was kept and adapted to the problem of three-dimensional densities (see 3.2.2). No existing version of the two dimensional implementation is available to the knowledge of the author, so this implementation can be seen as novel. First, the description of the four `kernels` will be given and subsequently some notes on the usage of the implementation will follow.



initialize segmentation in shared memory with −1
load segmentation to shared memory
initialize local labels in shared memory
6 7 8
3 4 5
0 1 2
while labels change
each thread explores its neighorhood
6 6 3
3 3 3
0 0 0

Figure 5.32: CCL-`kernel1` mechanisms illustrated.

5.8.1 Structure of the GPU kernels

As stated, the basic functionality of the `kernels` was maintained. Therefore, `ccl_kernel1` solves the task locally in subvolumes. Figure (5.32) illustrates the basic principle. The illustration itself depicts the problem in two dimensions, while the implementation proper is set in three-dimensional space. From top to bottom, the essential steps are as follows:

- Two ranges of `shared` memory are allocated: One for the local copy of a segmentation subvolume of the size $10 \times 10 \times 10$ and one for the local labels corresponding to that subvolume, the size being $8 \times 8 \times 8$. The different sizes are due to the fact that later every thread needs will "run through" the labels and segmentations and check their respective neighbors. By choosing the segmentation array slightly bigger, one can ensure that every pixel has a well defined neighbor.

- In the beginning, the `shared` segmentation array is filled with -1-values, to later distinguish the edge region from the actual segmentation subvolume.

- Then, a specific subvolume is loaded into the `shared` segmentation array. The position of this subvolume in the total segmentation volume depends on the internal variable `blockIdx`. The total number of `blocks` is therefore determined by the size of total segmentation volume. Additionally, the `shared` label array is initialized with local labels ranging from `0 .. 511`. The last initial step is the allocation and initialization of a `shared` memory field that will be used as a flag to indicate a change in the local labels.

- At this point, the main loop starts. Each one of the `512 threads` performs the same actions per iteration of the loop. It checks every of its 26 neighbors, whose position in linear memory is stipulated by the standard convention (see 3.2.2). If a pixel and one of its neighbors belongs to the same segments, the `UnionFind` operation will be run with these two pixels as input (see 3.2.4), therefore changing their label and updating the label array. If this pixel pair does not already share the same label, the above mentioned flag will be set to `1`, indicating that labels have changed. In this case, the loop will be run again. This means effectively more work, since in the worst case (depending on the geometry of the segmentation), one root label will migrate through the complete subvolume, one pixel of the 26-neighborhood at a time. This is the trade-off for parallelization.

- Once the loop has been terminated, the local labels will be translated into a global range, ranging from `0 .. pixel_vol-1`, and written to `global` memory, concluding the process of local subvolume labeling.

The successful call of `kernel1` results in locally labeled subvolumes. The next two kernels, `kernel2` and `kernel3`, carry out this task. `kernel3` only serves to flatten the equivalence tree once per merging iteration, which, according to [66], leads to a performance gain. `kernel2` carries out the actual merging. At its heart there is the geometric merging scheme, which is depicted in figure (5.33). Since all subvolumes have the dimension $8 \times 8 \times 8$ in the beginning, they can be merged by a scheme similar to figure (5.33a [the actual degree of refinement was lowered for illustrative purposes]). At the respective interfaces of 8 $8 \times 8 \times 8$ subvolumes the `UnionFind` operation is conducted, if two pixels at one of the interfaces are of the same segment. This results in one $16 \times 16 \times 16$ subvolume, which subsequently is flattened by `kernel3`. This process will be run in parallel for all $8 \times 8 \times 8$ subvolumes, resulting in $16 \times 16 \times 16$ subvolumes, 8 times fewer in number. The

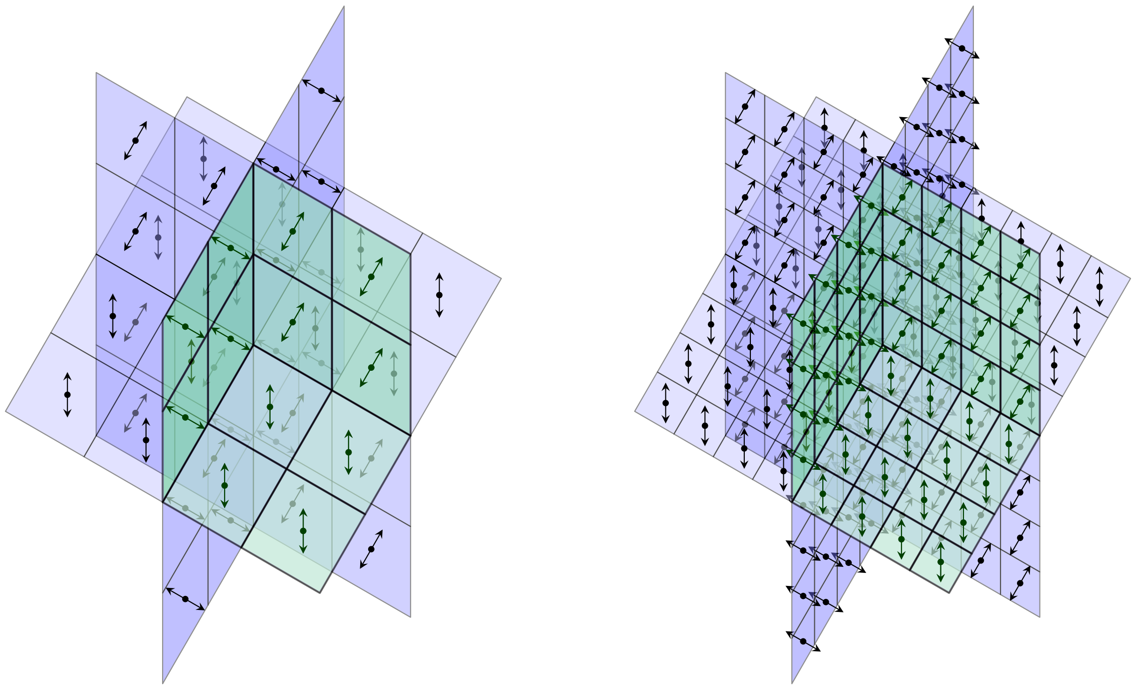

(a) Lower degree of merging refinement. (b) Higher degree of merging refinement.

Figure 5.33: Illustration of the merging scheme of `kernel2`.

merging scheme is written so that it is scalable by factors of 2. Figure (5.33b) indicates this. Thus, in an iterative fashion, the $16 \times 16 \times 16$ subvolumes can be merged again, until the volume is merged in total.
Because of this merging scheme the original segmentation volume is embedded in a larger, cube-shaped volume, whose pixel dimensions are powers of 2 and at least $\{8, 8, 8\}$. The last kernel, `kernel4`, flattens the equivalence trees on a global level and finally provides the complete labeling. All 4 kernels are called from a central kernel, making therefore use of dynamic parallelism (see 3.3.5). No host interaction is required during the labeling procedure.

5.8.2 User interface and convenience functions and tests

The programming API of the `Labeler` class, which contains the kernels, device functions and data type definitions, offers two functions of convenience:

- The function to count and list the occurrence of every distinct label.
- The option to write out the segments corresponding to the labels in `mrc` files and to inspect them visually.

Beyond these convenience function, a static function

```
Labeler::ccl(int * d_seg_data, uint * d_labels,uint log_2_dim,const int & gpu_index)
```

is exposed, where the inputs have the following meaning:

- `int * d_seg_data` is the segmentation. It can be binary or any number of segment types. It is allocated on the `global` memory of the GPU device. It needs to be embedded in a volume as described above. Note that the geometry of the density needs to be maintained during this embedding.
- `uint * d_labels` is also located in `global` memory. It only needs to be allocated and once the CCL operation is completed it will hold the labeling. It needs to be the same size as `int * d_seg_data`, it will have the same geometry (each element with a label will belong to the same pixel as the corresponding segment).
- `uint log_2_dim` is the pixel dimension of the embedding volume.
- `const int & gpu_index` is the index of the device to be used.

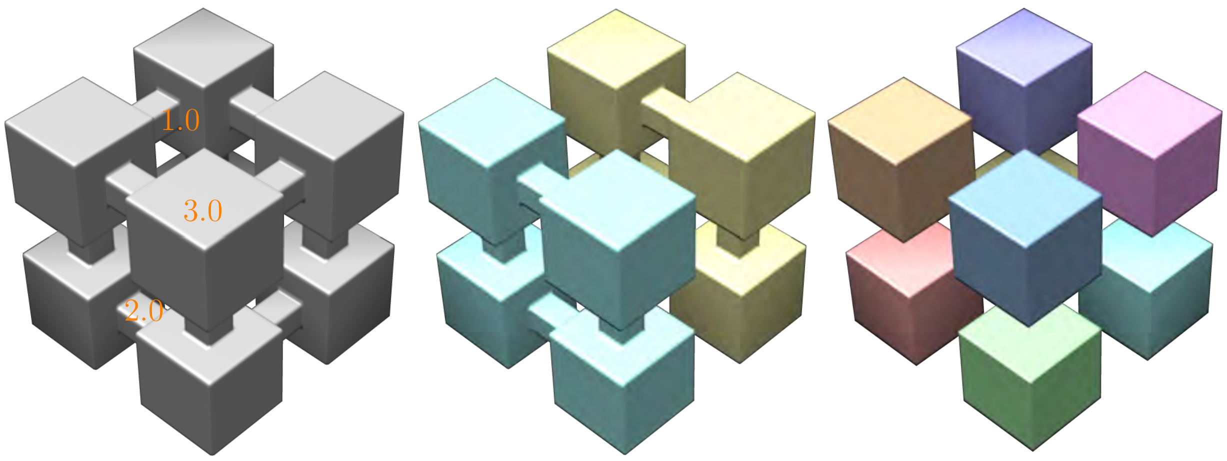
Due to the novelty of the implementation, several tests have been conducted in conjecture with a possible use case. The compilation target `ccl` builds a binary which functions as follows:

```
./ccl mrc_file threshold
```

The parameters are to be understood as follows:

- `mrc_file`: A density file in the `mrc` format.
- `threshold`: A floating point number which serves as a threshold for the density. Using this threshold, a mask is applied to the density which serves as a segmentation map. For every pixel which surpasses the threshold, the mask will hold a `1` and `0` otherwise.

Figure (5.34) shows a possible use case. The initial density (5.34a) is one connected volume made up out of 8 smaller cubes, all of which have the density 3.0. 4 of the connecting cuboids have either the density 1.0 or 2.0, as indicated. Figure (5.34b) highlights which ones have the density 1.0, since it is the output of the `ccl` program with the `threshold` parameter set to 1.5, it results in two connected components. Figure (5.34c) shows the result of the program run with the `threshold` 2.5, so that none of the connecting cuboids contributes to the mask anymore. The results are 8 different connected components.



(a) Initial density. (b) Output for $t = 1.5$. (c) Output for $t = 2.5$.

Figure 5.34: Example use of the `ccl` program.

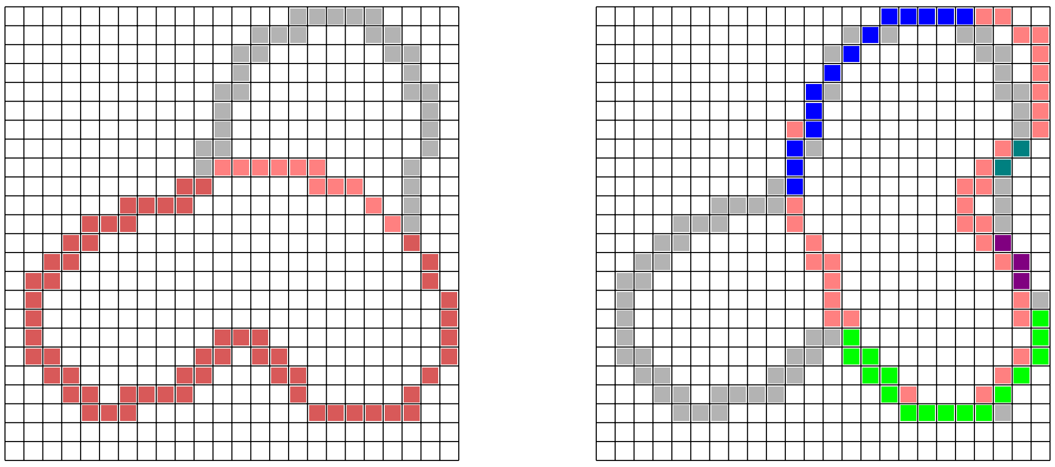


Figure 5.35: An illustration of the basic principle behind the partial surface score. To the left, the is just one,big continuous overlap. It would be scored very high as opposed to the right placement, which has four, disconnected, smaller labels. It would score considerably worse.

5.9 The partial surface score

The partial surface score was conceived as a reaction towards the perceived lack of scoring methods in the context of systematic fitting of *partial surfaces*, which are derived from query structures, in greater surface target, e.g. the result of a negative stain EM acquisition of a protein complex. Scores that are typically suitable to match surfaces, such as the envelope score (see 2.5.5) or the chamfer distance (see 2.5.5) do not perform well when there is only a partial surface fit available. The concept and implementation of the partial surface score will be outlined in this section.

5.9.1 Score definition

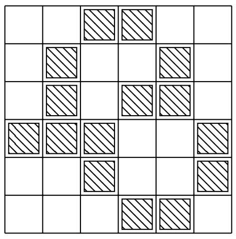
Figure (5.35) illustrates two contrasting situations in a two dimensional setting and can be used to understand the procedure that is laid out in the following:

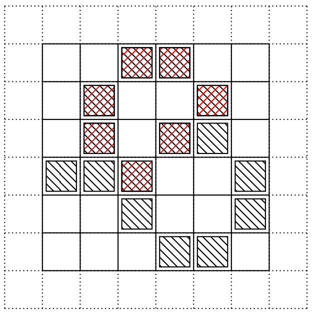
- Initialize a mask density M that has the same pixel dimensions as the target density T. Initialize its values to 0.
- Define for surface for both the target by specifying two thresholds, $\left(t_0^T, t_1^T\right)$, and the query, $\left(t_0^Q, t_1^Q\right)$. These two surface s^T and s^Q are subsets of pixels of both densities and constitute the basic geometric objects that will be scored against each other. Apply padding to both the target density and the mask density M.
- Apply a transformation consisting of a translation and a rotation $(\mathbf{t}, \mathbf{r})$ to the pixels that constitute the query surface s^Q. The transformation should be applied to the coordinates (x, y, z) of each pixel as defined in 3.2.2.
- Map the transformed coordinates (x', y', z') to their respective pixels within the target surface density. There they will either coincide with pixel in s^T or not. If they are, write a 1 to M.
- Run a Connected Components Labeling routine on M (see 5.8). This will result in a label volume L that has the same dimensions as M.
- For every label in M, count its occurrences. For the labels $(l_0, l_1, \ldots, l_{L-1})$ this will result in a list of counts $c_0, c_1, \ldots, c_{L-1}$.
- In this step, disregard the label that marks the background. Calculate

  $$s(\mathbf{t}, \mathbf{r}) = \sum_{i=0}^{L-1} b^{c_i} \tag{5.5}$$

  with b being a suitable basis. This will be the final partial surface score associated with the transformation $(\mathbf{t}, \mathbf{r})$.

The basis b should be chosen in such a way that the largest possible label volume V_L, being the pixel volume of the target surface s^T, does not exceed the maximum floating point value of the system. This scoring method will score the *largest continuous overlap of target and query surfaces* the highest.


(a) The target density.


(b) A placed query density plus padding.

56	57	58	59	60	61	62	63
48	49	50	51	52	53	54	55
40	41	42	43	44	45	46	47
32	33	34	35	36	37	38	39
24	25	26	27	28	29	30	31
16	17	18	19	20	21	22	23
8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7

(c) An initial labeling.

0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0
0	0	1	0	0	1	0	0
0	0	1	0	1	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(d) A segmentation of the overlap in 5.36b.

0	0	0	0	0	0	0	0
0	0	0	27	27	0	0	0
0	0	27	28	28	27	0	0
0	0	27	28	27	0	0	0
0	0	0	27	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(e) After a complete labeling.

0	0	0	0	0	0	0	0
0	0	0	27	27	0	0	0
0	0	27	0	0	27	0	0
0	0	27	0	27	0	0	0
0	0	0	27	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

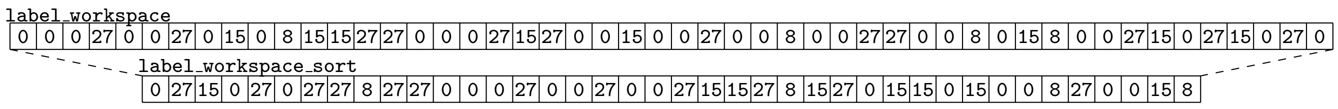
(f) Labeling cleaned of enclosed spaces.

Figure 5.36: Illustrations of the computational steps of the partial surface score.

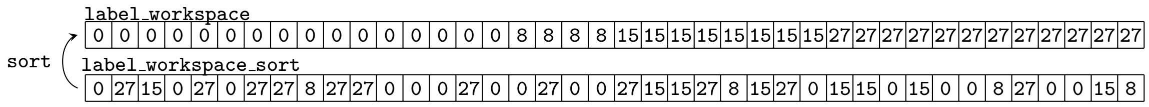
5.9.2 Score implementation

The score was implemented in a single `kernel`, which calls different, complex `kernels` and algorithms, thus using dynamic parallelism (see 3.3.5). Counting the label occurrences requires a sorting step, which is a non-trivial task, especially in a parallel environment. For this step the external `cub`-library, which underlies the more high level `thrust`-library (see 3.3.6), has been used. The figures (5.36) and (5.37) accompany the implementation description:

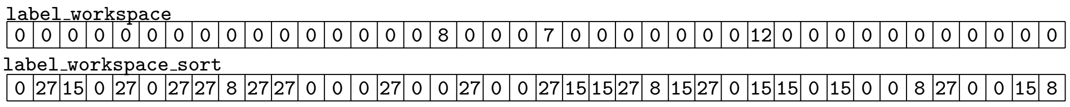
- The target surface is present as a `density` data structure (3.2.2), for ease of algorithmic design. The query surface however is only present as a set of pixel coordinates (see 5.36a).
- The query is placed within the target `density`, which is padded with a number of pixels (see 5.36b). The volume of the padding depends on the geometrical size of



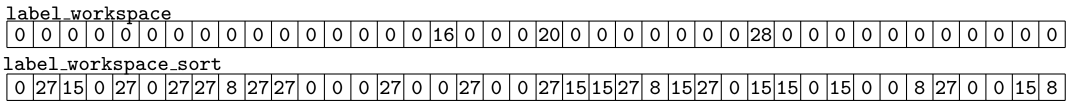
(a) Extracting the labels not corresponding to the padding.



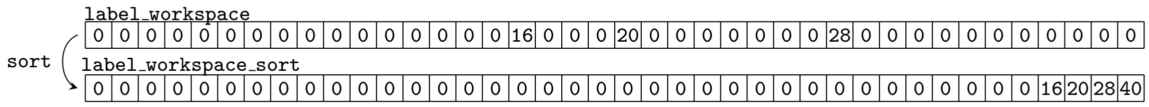
(b) Sorting the labels.



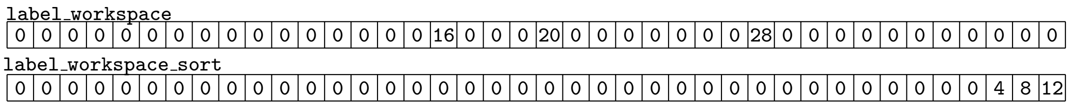
(c) Calculating the adjacent differences.



(d) Replacing non-zero labels with their position.



(e) Sorting again and writing pixel volume to the last index.



(f) Calculating the adjacent differences, modified.

Figure 5.37: Computational steps of the partial surface score illustrated in linear memory.

the query surface and the transformational space that is supposed to be searched. This padding is not strictly necessary, yet it circumvents a lot of `if` statements that check if a transformed pixel still maps into the `density`. This is an example of trading memory for computational time.

- The overlap between target and transformed query surfaces in pixels is assessed and the mask is filled according to the rule specified in 5.9.1 (see 5.36d). This, together with a label array that will be initiated with one individual label per pixel (within the `ccl`-kernel) (see 5.36c), will be handed to a `ccl`-kernel that is called from within the partial surface score kernel. Note that both the segmentation mask array and the label array have the dimensions of the padded volume.

- The `ccl`-kernel performs the labeling (5.36e). There is the possibility of enclosed spaces existing in the overlapping surface. This labels cannot be allowed to contribute to the overall score. Therefore they are set to `0` by simple multiplication with the segmentation mask (see 5.36f).

The next part concerns sorting and the calculation of the label occurrences. Geometrical intuition is less important, so the illustrations will be portraying linear memory (see 5.37). Since the function used for sorting, `cub::DeviceRadixSort::SortKeys`, does not allow for in-place sorting, there are two memory ranges reserved for label processing.

- First, the labels relevant to the problem, meaning those who belong to pixels that are not part of the padding, are copied into a smaller memory range (`label_workspace_sort`) for the next steps (see 5.37a). The original, larger range (`label_workspace`) will still be used, but in a reduced capacity. Since the volume of the padding can be quite substantial, this step improves computational efficiency.
- The labels are sorted and written to the other memory range,`label_workspace_sort`, (see 5.37b).
- The adjacent differences are calculated in-place (see 5.37c). This serves for "edge detection", to find the indexes where the labels change.
- Where the entries of the `label_workspace_sort` memory range are not zero, they will be replaced by their index. This will be used to assess their occurrence (see 5.37d).
- Yet another sorting step takes place to list the index values contiguously (see 5.37e). Yet again, the memory ranges are switched for input and output. Additionally the total pixel volume is written to the last element of the memory range.
- Another, modified version of the adjacent difference operation is applied. Now the final version of the `label_workspace_sort` contains the occurrences of the labels. (see 5.37f).
- Now, in the last step, the summed exponent operation defined in 5.9.1 is applied by first exponentiating the label occurrences and then summing them via a reduction operation.

This concludes the partial surface score calculation for a single transformation. It should be pointed out that this scoring method is extremely expensive compared to the Chamfer distance and the envelope score (see 2.5.5).

5.10 Particle Alignment and RMSD Calculation implementation

This section contains a description of the implementation of the parallel RMSD (root mean square deviation) calculation and parallel particle alignment algorithms. The specific data layouts and the general description of the algorithms that were employed can be found in 3.2.1. A general primer on GPU programming can be found in 3.3. Here I will limit my discussion to the software itself and the applicability of parallel programming in certain aspects of structural integrative modeling. Generally speaking parallel programming shines when there are a lot of similar procedures to be executed with multiple, uniform data fields as input. This paradigm is sometimes called Single Instruction, Multiple Data (SIMD). Structural integrative modeling employs data types that almost exclusively refer to sets of particles or densities, which both are comprised of similar data fields (particles and pixels,respectively), and are therefore suitable for a SIMD approach. A straightforward example is the RMSD calculation implementation.

5.10.1 RMSD calculation

An RMSD calculation is useful to measure the similarity between 2 or more structural confirmations of particle sets. These can be sets pseudo atoms as the result of a Monte Carlo simulation as described above are a set of atomic structures that were deformed in a molecular dynamics simulation. The important feature is, that there is a shared identity between each of the particles in each of their different confirmations. I will call this confirmation *frames* from hereon out (it is useful to think of them as "snapshots"). The situation is therefore a set of P particle coordinates in F different frames is written to memory as described in 3.2.1. If one looks at a number of different frames one is interested on how similar they are to each other, therefore the calculation of the RMSD of every frame i with every other frame j with $i, j \in \{0, 1, \ldots, \mathrm{F}-1\}$ has been implemented. The result I will call the RMSD matrix $\texttt{rmsd}_{ij}$. The general procedure of the core algorithm is the following:

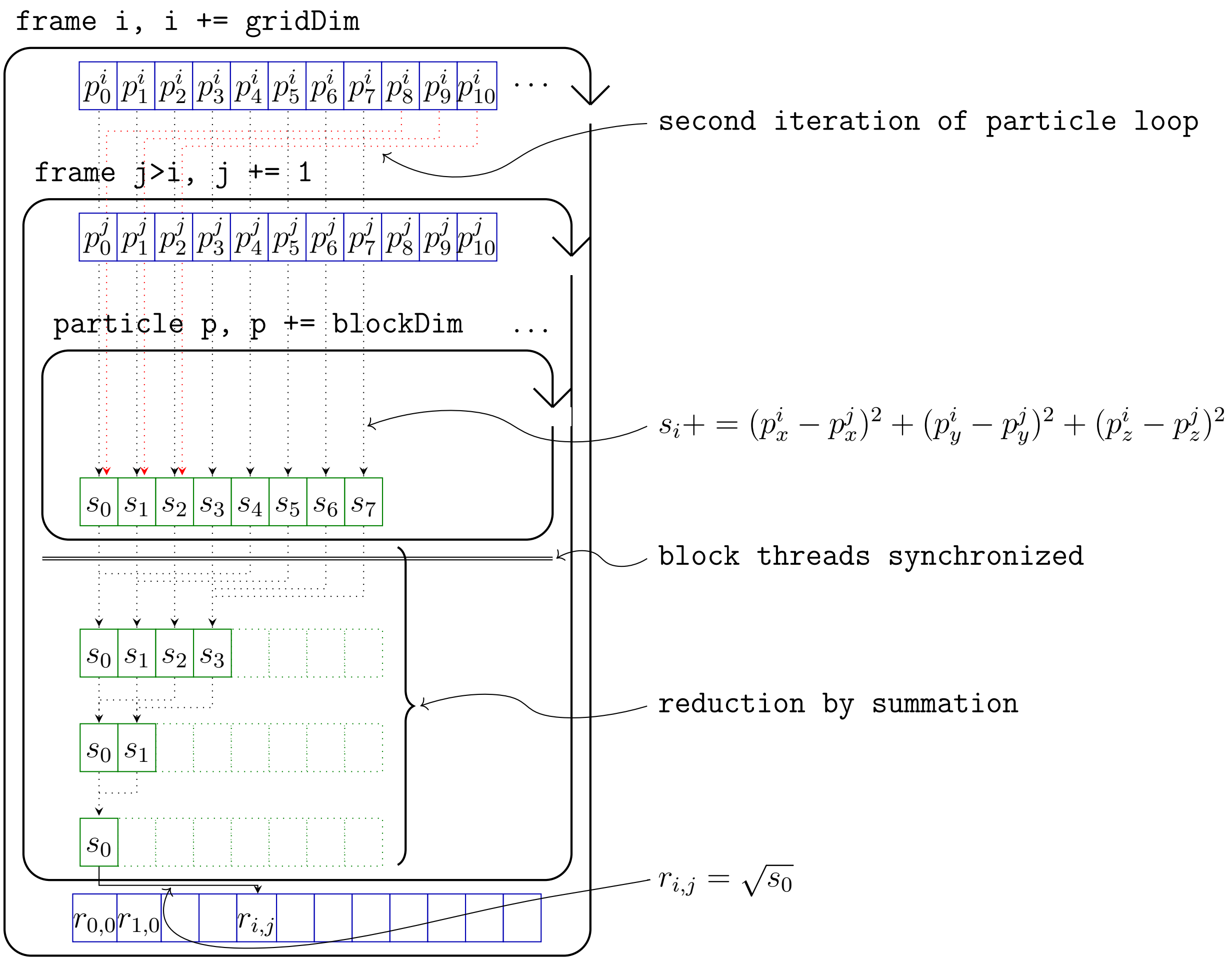
1. Allocate the necessary memory volumes both on CPU and GPU working memory. The central data structures are the particle set `h_particle_set` (`d_particle_set` on device) containing the frames and coordinates and the result of the calculations `h_rmsds` (`d_rmsds` on device).
2. Generate or load the data to `h_particle_set` and copy it to `d_particle_set`.

3. Determine the blockDim, gridDim and shared_mem_volume parameters to launch the RMSD calculation kernel rmsd. Launch the kernel providing it with the necessary information.

4. After the device is synchronized, copy the contents of d_rmsds to h_rmsds and conduct eventual further steps of evaluation.

All of these steps are straight forward except 3., which warrants further explanation. Figure (5.38) illustrates the process within the kernel. It follows a three loop structure. The outer loop refers to frame i, the intermediate loop to the frame j, the innermost loop to particles p. The outer two loops correspond to the requirement to calculate the rmsd between any two frames. While the increment of the outer loop is gridDim, which corresponds to the number of blocks and therefore to active workers of the kernel, the increment of the inner loop is 1. This means that every worker takes care of one frame i and all of its potential pairings, frame j with j = 0...F-1. These two outermost loops just serve to address the different frame pairs. The innermost loop iterates over all pairs of particles. Its increment is blockDim, which corresponds to the number of threads that is assigned to every worker. In figure (5.38) this number is set to 8 for purpose of illustration. The group of threads iterate collectively through all particles in chunks of 8. The red guide lines illustrate the actions of the first few threads in the second iteration of the inner loop. The actual calculation of the term within the sum of the RMSD definition is carried out as indicated in the inner, with pairs of particles of their respective frames as input. The sum is carried out partially within the inner loop and directly written to shared memory over the iterations of the inner loop. Green denotes shared memory and blue global memory. When the inner loop addressed all particle pairs, the threads are synchronized to make sure all have finished their task properly before proceeding. The following step just adds up the values written to shared memory, reducing it to a single value. This value corresponds now to the complete sum. Lastly, the square root is taken and the result is written to global memory.
Some steps, which are not essential to the concept of the kernel, but to the computation in general (such as initialization of the shared memory), have been omitted for brevity. The kernel execution parameters are determined the following way. gridDim corresponds to the number of workers employed and was chosen to not be smaller than the number of streaming multiprocessors times 2 of the respective GPU. The goal is that every processor is used to its maximum capacity. The blockDim, so the number of threads, was chosen to be twice the warp size, for the reason that it hardly makes sense to go lower and to provide some room for the hardware to optimize its calculations. The volume of the shared memory is simply the volume needed for the innermost loop. It is capped by the specific capacities of the GPU in question, generally this number will however be generous enough to not be of concern. The optimal choice of gridDim and blockDim is subject to

experimentation and will vary with the specifications of the GPU. Additionally a larger choice of `blockDim` would mean that every thread has less particle pairs to process. This will affect the trade off between serial and parallel calculations. The usage of the memory hierarchy is designed to minimize writing and reading to and from `global` memory (blue) and maximize the same for the `shared` memory (green). use patterns. output.



frame i, i += gridDim
second iteration of particle loop
frame j>i, j += 1
particle p, p += blockDim
$s_i += (p^i_x - p^j_x)^2 + (p^i_y - p^j_y)^2 + (p^i_z - p^j_z)^2$
block threads synchronized
reduction by summation
$r_{i,j} = \sqrt{s_0}$

Figure 5.38: Schematic of the RMSD matrix calculation kernel

Alignment by RMSD minimization

In a given situation we might not be interested in the RMSD as a measure to discern differences but as a guide to minimize differences. Two confirmations might be the same or quite similar and oriented differently in space or placed in different positions. This will make comparing them difficult or impossible. The following algorithm is described in 3.8. The data structure used to address and represent the single frames and particles will yet again be the particle set. This time however the goal is to place and orient all frames with reference to a single frame, so that a comparison is possible. This single frame is arbitrarily chosen and will be referred to as the `frame0`. The general procedure is set as follows:

1. Allocate the necessary memory volumes both on CPU and GPU working memory. The central data structures are the particle set `h_particle_set` (`d_particle_set` on device) containing the frames and coordinates, the intermediate matrices ($\mathbf{M}_i^\mathbf{j}$) `d_m_alphas`(only on device since they are an intermediate result), the results of the center-of-mass calculations `h_coms` (`d_coms` on device) and the result of eigenvector calculations `h_U` (`d_U` on device).

2. Generate or load the data to `h_particle_set` and copy it to `d_particle_set`.

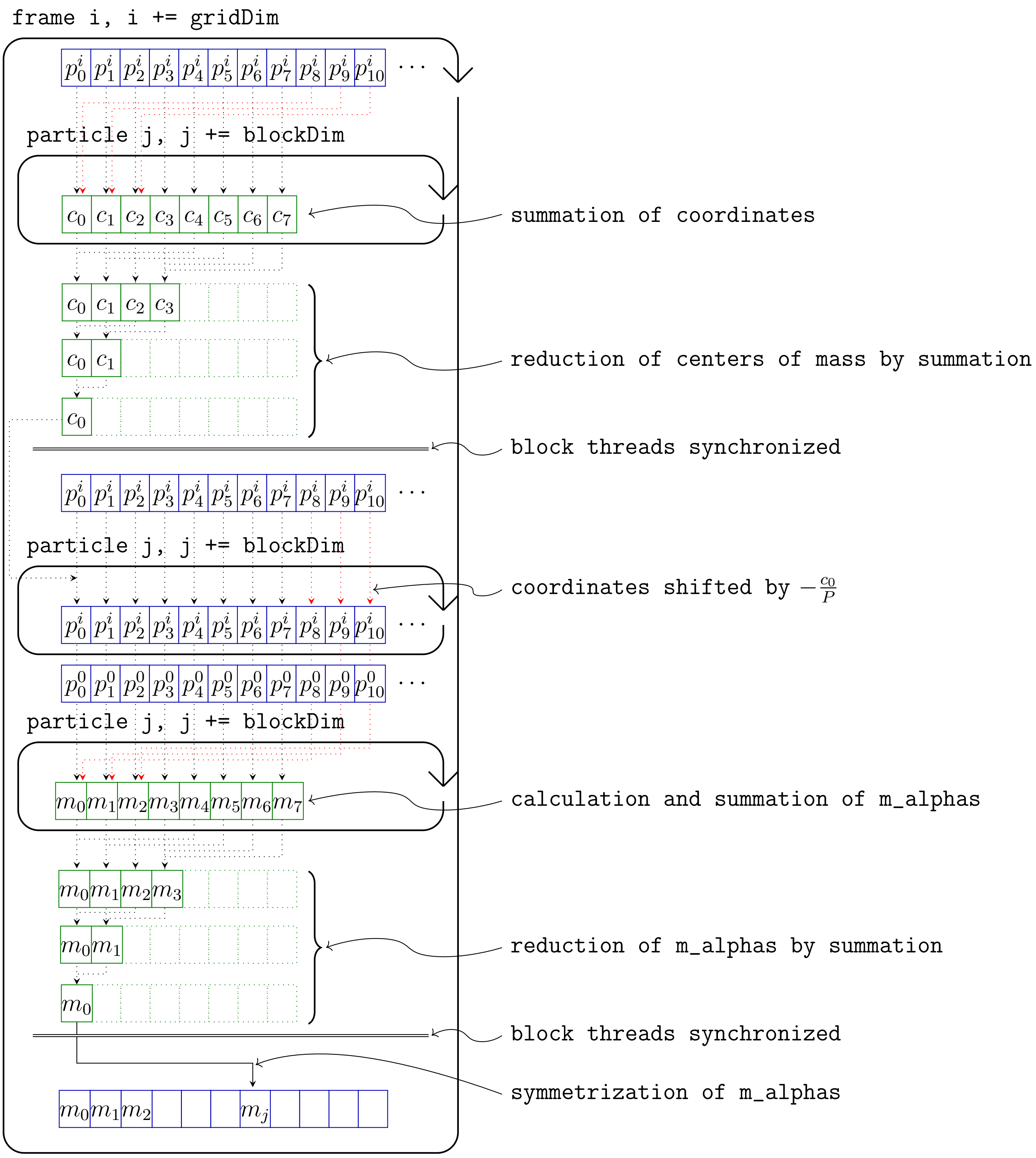
3. Calculate the center of mass for all `frames` and translate them so that the center of mass of all `frames` coincides in the origin. Calculate the intermediate matrices `d_m_alphas` and symmetrize them.

4. Calculate eigenvalues and eigenvectors of all matrices using the `cusolver` library.

5. Select the eigenvector corresponding to the smallest eigenvalue and either apply the rotations in place on device or copy the data to the host for further processing.

As above I will not explain trivial steps such as initialization or copying. This procedure has been implemented as three distinct host function calls roughly corresponding to 3, 4 and 5. I will describe them in order.
Step 3 can be understood as a sequence of reductions and transformations that operate on the `frames` in memory. Figure (5.39) illustrates the kernel that accomplishes this. The kernel is designed after the `worker` idiom. The outer loop iterates over the `frames` and processes each single frame. There are `gridDim` instances of this loop and therefore `gridDim` `workers`. Thus the increment of this loop is `gridDim`. The outer loop contains three inner loops and two reduction operations. The inner loops all iterate over the particle coordinates and are instantiated once per `thread`. Thus their increment is `blockDim`. The first and the last loop perform a reduction operation, the intermediate loop a transformation. As above, red guide lines indicate the actions of the subsequent iteration of the inner loops. The first inner loop simply adds up the particle coordinates per frame using a `shared` memory as a buffer. These sum are then further reduced to the sum total. This final sum is devided by the number of particles to get the center of mass. The second inner loop operates on the global memory of the `frames` and shifts the origin to the calculated center of mass. The last inner loop calculates the alignment matrices as indicated at 3.2.1 and sums them up. The `frame0` is an input in all of these calculations, serving as a constant reference for the alignments. Before they are written to global memory, the symmetric components are initialized to enable eigenvector calculation with the `cusolver` library. The consideration regarding the optimal `gridDim` and `blockDim`

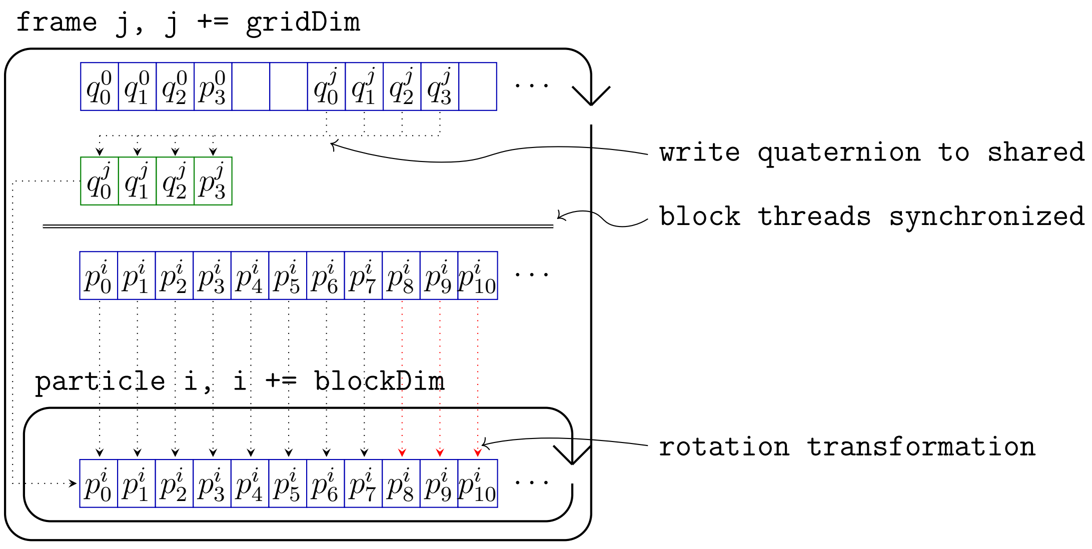
are the same as in the preceding section. The `shared` memory has to be large enough to hold the centers of mass and the alignment matrix for every frame. The entries of the latter are saved as `double` to accommodate the `cusolver` library. The `shared` memory volume is therefore `num_frames*(16*sizeof(double) + 3*sizeof(float))`.



frame i, i += gridDim
particle j, j += blockDim
summation of coordinates
reduction of centers of mass by summation
block threads synchronized
particle j, j += blockDim
coordinates shifted by $-\frac{c_0}{P}$
particle j, j += blockDim
calculation and summation of m_alphas
reduction of m_alphas by summation
block threads synchronized
symmetrization of m_alphas

Figure 5.39: Schematic of the center of mass and alignment matrix calculation kernel.

The next step is initiated by a host call after a setup of the resources necessary for the `cusolver` library. The 4×4 matrices whose eigenvalues and eigenvectors are to be determined are already on device and are supplied via memory address. The results are also left on device, there is an option to copy them to the host if need be. The `cusolver` library performs best when it can solve batches of similar problems, thus the whole stack of `frames` is processed in parallel. The options provided by the library also enable ordered output of the eigenvalues and eigenvectors, which is used to pick the right eigenvector to interpret as a quaternion and subsequently as a rotation.
Once the rotations for each frame are calculated, the frames as a whole are finally rotated. This, too, is done on the device in a double-loop structured procedure. Figure (5.40) illustrates the implementation of this process. The outer loop iterates over the `frames`, its increment is `gridDim`, which is equal to the number of `workers`. So each `worker` processes a `frame` at a time. The first `thread` of each iteration is tasked to retrieve the `quaternion`, which encodes the rotation, and write it to `shared` memory. This has the reason that all `threads` must be able to access this information, since each particle of any `frame` will be rotated in the same manner. Therefore it pays to have it in fast `shared` memory. The inner loop iterates over the particles of each `frame`. Every `thread` rotates a given particle, the increment is thus `blockDim`. The rotation itself is modeled after the mathematical operation specified in 4.2.3. The memory volume of the `shared` memory must be large enough to hold one quaternion, so `4*sizeof(float)`.



frame j, j += gridDim
q_0^0 q_1^0 q_2^0 p_3^0 q_0^j q_1^j q_2^j q_3^j ...
write quaternion to shared
q_0^j q_1^j q_2^j p_3^j
block threads synchronized
p_0^i p_1^i p_2^i p_3^i p_4^i p_5^i p_6^i p_7^i p_8^i p_9^i p_{10}^i ...
particle i, i += blockDim
rotation transformation
p_0^i p_1^i p_2^i p_3^i p_4^i p_5^i p_6^i p_7^i p_8^i p_9^i p_{10}^i ...

Figure 5.40: Schematic of in-place rotation transformation kernel.

Environment, Tests and Benchmarks

In this section I will describe how the implementation described above are embedded in a given computational environment. I will first describe the user perspective and following

that the hardware embedding.
The user has two input options:

- The command line interface. This option provides three functionalities:
  - `align input.rmf output.rmf`
    This takes an input file of the `rmf` format (see [27]) and aligns every frame following the first frame to the first frame. All frames in the file `input.rmf` are expected to represent the same assortment of particles.
  - `align input.rmf`
    This call pattern takes a valid `rmf` file as input and outputs the `rmsd` matrix as a comma separated list to the standard output.
  - `align` This runs a benchmark on the machine and outputs the results to the file `benchmark.csv` in a comma separated table.
- The developers interface. This offers an interface for use in `C++` source code. It offers a similar functionality as the first interface.
  - `align(float4 * h_particles_4,float4 * d_particles_4, int frames,int particle_num, int gridDim,int blockDim)`. This function aligns frames in the same manner as outlined above. Additionally it is possible to align frames with a specific focus, e.g. around one particular subunit. The last two parameters allow for different workflow designs for the GPU call.
  - `rmsd(float4 *& d_particle_set,float *& h_rmsds, int & num_particles, int & num_frames, unsigned int gridDim, unsigned int blockDim)`. Calculates the above mentioned `rmsd` matrix and saves it to `h_rmsds`. The last two parameters are again for GPU configuration.

Additionally a compilation flag enables the code compilation without the `IMP` library, making it independent of this particular library.
To ensure correct results of the implemented algorithms, different tests are implemented:

- `cubeTest`. A set of 8 `particles`, arranged in a cube around the origin and rotated in 24 different orientations, covering all possible ways to rotate a cube to coincide with itself. These are then aligned, the `rmsd` matrix calculated. It should be close to 0 everywhere.
- `pureTest`. A set of `F frames` and `P particles` is generated by first generating one `frame` using randomly positioned `particles` and then rotating it into `F-1` additional `frames` using randomly generated unit quaternions. Alignment and `rmsd` calculation follow. The result should be close to 0 everywhere.

- `rmfTest`. For this test a valid `rmf` file is necessary. A `frame` is loaded from the file, randomly oriented in the above sense, written to another `rmf` file (in random orientation), loaded yet again, aligned, written to a third `rmf` file, loaded again and the `rmsd` is calculated. The result should be close to 0 everywhere.

- `rmfAlignTest`. Requires a valid `rmf` file with `F frames` signifying different confirmations. Aligns the file and generates another `rmf` file as output. Loads this file, and for every single frame: Reorients it globally according to a set of orientations which are supposed to produce a representative sampling. Then calculates the `rmsds` between the supposedly aligned orientation and the first `frame` of the original file and does the same for the rotated `frames`. The original `frame` orientation should yield the minimal `rmsd`. In the same manner every single `frame` is perturbed locally by use of a unit quaternion close to `[1,0,0,0]` which signifies identity (= no rotation). Again, the unperturbed frame should yield the minimal `rmsd`.

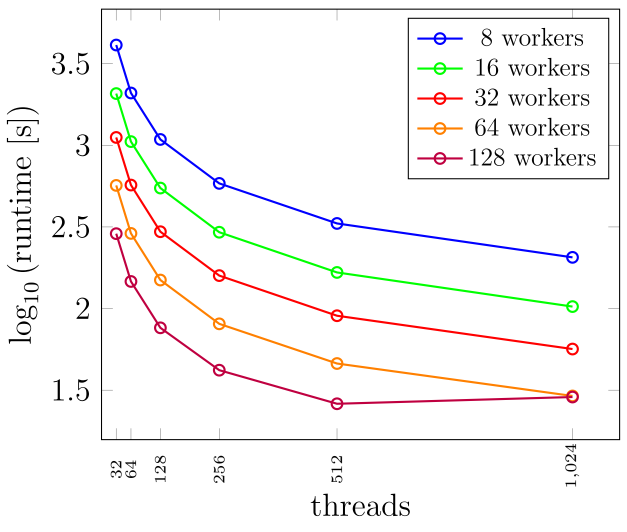
All tests check out correctly in the submitted version.
To enable comparison with similar software an extensive benchmarking has been conducted on a GPU of the model `NVIDIA Tesla V100-GPU`. The following benchmarks were executed:

- Two processes were characterized by benchmarks. First, the `align` procedure which aligns a given number of `frames` with a given number of `particles` to minimize `rmsd`. Second, the `rmsd` procedure, which calculates the $\texttt{rmsd}_{i,j}$ matrix between any two `frames` of a given number of `frames` with a given number of `particles`. Both procedures have been characterized in three different ways, as indicated in the following.

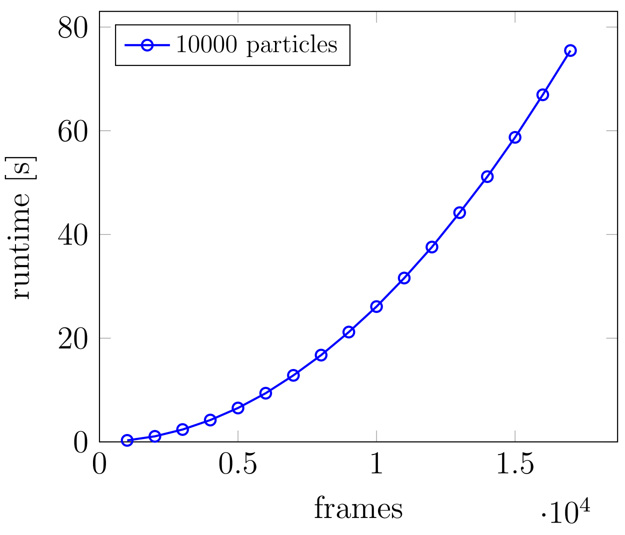
- The first benchmark tested the optimal combination of `threads` and `workers` by measuring the execution time. The results are shown in figure (5.41a) for the `rmsd` procedure and in figure (5.41d) for the `align` procedure. In both cases a constant number of `10000 frames` and `10000 particles` has been selected as a representative test case.

- The second benchmark tests the behavior of both procedures under variation of the number of `frames` by measuring the execution times. he results are shown in figure (5.41b) for the `rmsd` procedure and in figure (5.41e) for the `align` procedure.

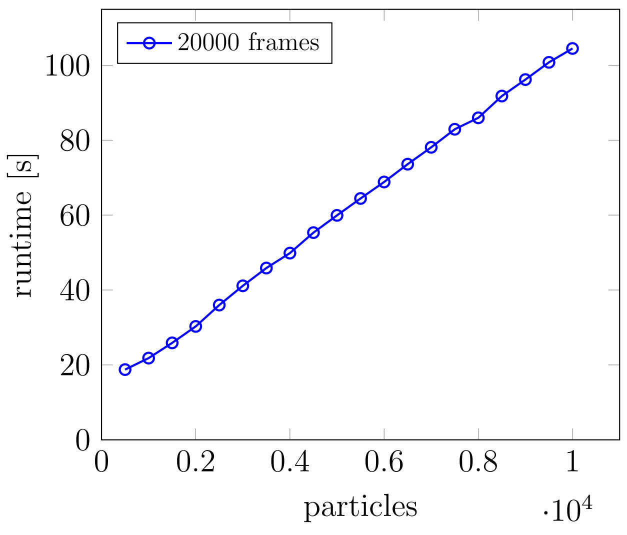
- The second benchmark tests the behavior of both procedures under variation of the number of `particles` by measuring the execution times. he results are shown in figure (5.41c) for the `rmsd` procedure and in figure (5.41f) for the `align` procedure.



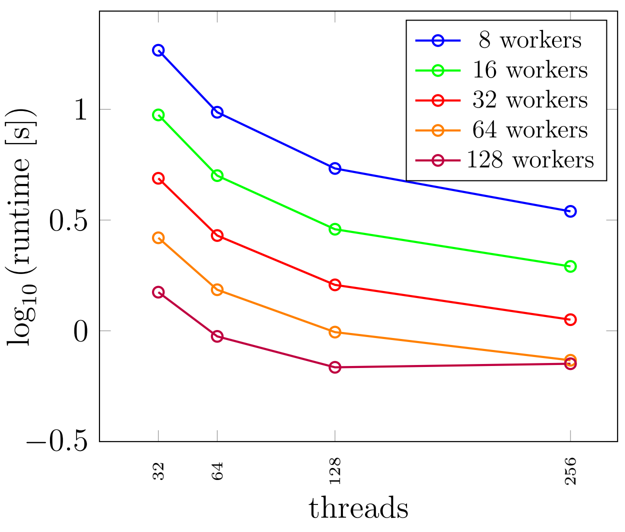
(a) Worker/threads rmsd



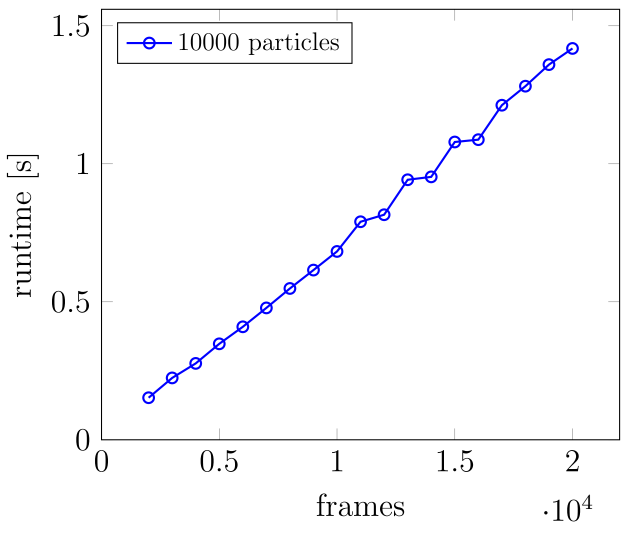
(b) Frames rmsd



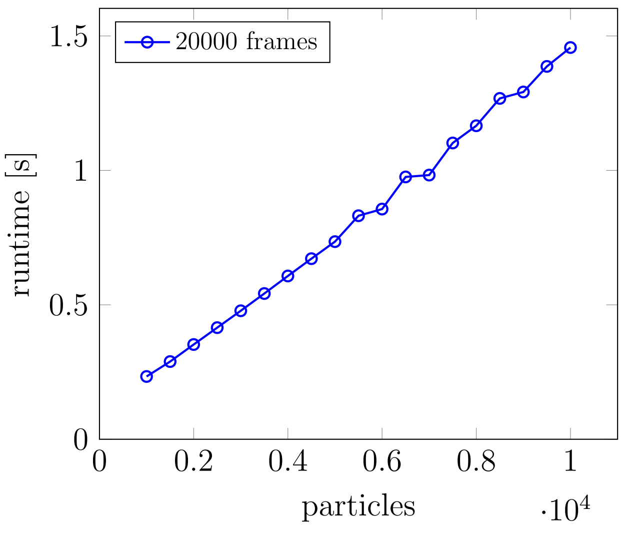
(c) Particles rmsd



(d) Worker/threads align



(e) Frames align



(f) Particles align

Figure 5.41: Benchmark data

5.11 Reinterpretation of systematic fitting results

In this section I will discuss arguments regarding a reinterpretation of the results of a systematic fitting protocol as described in 2.5.5. These arguments result in the implementation of a general framework for systematic fitting. The latter makes only sense in their context. I will therefore lay them out in the results section as simply and clearly as possible, so that they might either be taken up or refuted.

5.11.1 Conceptual exposition

The scope of applicability of structural integrative modeling is limited. If the resolution (see 2.4.4) is high enough (meaning: small details, such as the are recognizable), structural integrative modeling is not needed. Only when it seizes to be possible to assign positions of single atoms in residues using only information that is given by the density map alone, the method becomes useful. As outlined above, systematic fitting plays the role of searching the space of orientations and positions for every subunit or subcomplex for “good” fits. This search must be ambiguous. If it were not, one could determine the position and orientation of the query structure or map in question. The impossibility to do that is the very reason we turn to structural integrative modeling. This ambiguity shows itself in several aspects of structural integrative modeling:

- Ambiguity in choice of the scoring method. As outlined in 2.5.5, there are a number of different methods to calculate the quality of a given fit. While there are heuristic considerations of what scoring method might be best for a given case, there is no provable or demonstrable way of making the right decision. One might even invent a completely new score by scaling and shifting an existing one or combining two existing scoring methods in some manner and would have no way of knowing if this new score is “good”.
- Ambiguity in the input data. The input data might be processed in a number of varying ways, e.g. a cryo-EM target density map might be filtered using a density threshold, effectively omitting parts of the signal, or a query structure might be a result of homology modeling or be broken to account for flexibility to name a few. The consequences of this manipulations for the final models can be significant, yet one always introduces bias at this point.
- Ambiguity in the modeling procedure. Several aspects of protein complexes are codified in the restraints of the Monte Carlo model sampling procedure, such as the two subunits should not occupy the same place or a certain symmetry needs to

be upheld. This codification in itself is ambiguous since there are more than one ways to accomplish it. Even greater is the ambiguity in the scoring function and its weight, were the modeler has to introduce a bias about the weight that is given to each source of information.

These sources of ambiguity are systemic to structural integrative modeling. One may reduce them, but never eliminate them completely. The idea at the heart of structural integrative modeling is that different ambiguous statements about the placement of the parts of a protein complex might complement each other and reduce the overall ambiguity. The reduction of ambiguity as a measure of quality of modeling procedures is one of the central arguments of this thesis.

5.11.2 Ambiguity in systematic fitting and scoring methods

Experience and intuition tell us that the result of a systematic fitting protocol holds the most significant amount of information in a structural integrative modeling project. The ambiguity stems from data preprocessing, the choice of scoring method and the choice of query or target data, if multiple options are possible. To get a measure of this ambiguity I propose to treat the results of a systematic fitting as a 6 dimensional probability distribution. This interpretation warrants a justification.

- Two different scores, meaning the result of two scoring methods of the same query, the same target and the same placement, are incomparable. The reason for this is that two different scores might follow different conventions of what is the “best” score (lowest or highest) and might extend over completely different ranges (e.g. correlation-about-mean-score vs. envelope score). The consequence of this is that the function values of every scoring method can only be evaluated with regard to its own distribution. This argument does not make a statement about the comparability of scoring methods as a whole.
- For any systematic fitting score distribution it is possible to find minimum and a maximum. Either be theoretical consideration or by searching it computationally, which is always possible, since scores must be well ordered. It is therefore always possible to normalize a score distribution.
- If within a score distribution for the same query and target two scores are given, meaning the scoring function is evaluated for two different placements, one resulting in 0.8 and the other in 1.6, we will consider the latter score to be twice as “good” as the former. This assumes that the scoring method favors higher scores over

lower scores and has its minimum at zero, but both can be achieved by shifting and inverting the score. This means that we only care about the ratio between different scoring function values, not the absolute value. This is a consequence of the first point. Moreover in later stages of the structural integrative modeling procedure the score distribution will effectively be used in this way: In the Monte-Carlo procedure a placement whose score is twice as big as some other placements should be sampled twice as much.

We can see that no information is lost if we interpret a scoring distribution as a probability distribution. Thus I propose the following procedure. Let $\mathcal{S}(\mathbf{r}, \mathbf{t})$ be a scoring function and $\mathbf{r}$ and $\mathbf{t}$ be taken from a set of possible query placements $\{(\mathbf{r}, \mathbf{t})_0, (\mathbf{r}, \mathbf{t})_1, \ldots, (\mathbf{r}, \mathbf{t})_{N-1}\}$ with N being the size of the set.

1. Find the minimum $\mathcal{S}_{\text{min}}$ and maximum $\mathcal{S}_{\text{max}}$ of $\mathcal{S}(\mathbf{r}, \mathbf{t})$. Depending on the interpretation of the score, i.e. if a high score is "good" or "bad", transform the entire distribution according to the following linear transformation:

$$\mathcal{S}'(\mathbf{r}, \mathbf{t}) = \begin{cases} \frac{1}{\mathcal{S}_{\text{max}}-\mathcal{S}_{\text{min}}} \cdot s - \frac{\mathcal{S}_{\text{min}}}{\mathcal{S}_{\text{max}}-\mathcal{S}_{\text{min}}} & \text{if the highest score is the best} \\ \frac{1}{\mathcal{S}_{\text{min}}-\mathcal{S}_{\text{max}}} \cdot s - \frac{\mathcal{S}_{\text{max}}}{\mathcal{S}_{\text{min}}-\mathcal{S}_{\text{max}}} & \text{if the lowest score is the best} \end{cases} \tag{5.6}$$

   This achieves a scaling of the values of $\mathcal{S}(\mathbf{r}, \mathbf{t})$ to the interval $[0, 1]$ and if necessary an inversion of the scoring distribution.

2. $\mathcal{S}(\mathbf{r}, \mathbf{t})$ is a sample of a 6-dimensional distribution. The underlying search space of systematic fitting is a Cartesian product of the spaces $\mathcal{SO}(3)$ (representing the rotations) and $\mathbb{R}^3$ (representing the translations), $\mathcal{SO}(3) \times \mathbb{R}^3$. Find an approximation of the integral

$$\mathcal{S}_0 = \int\limits_{\mathcal{SO}(3)\times\mathbb{R}^3} \mathcal{S}(\mathbf{r}, \mathbf{t})\, \mathrm{d}\mathbf{r}\, \mathrm{d}\mathbf{t}$$

   and scale the result above $\mathcal{S}'(\mathbf{r}, \mathbf{t})$

$$\mathcal{S}''(\mathbf{r}, \mathbf{t}) = \frac{1}{\mathcal{S}_0}\mathcal{S}'(\mathbf{r}, \mathbf{t})$$

   to get a probability distribution $\mathcal{S}''(\mathbf{r}, \mathbf{t})$.

Step 2 requires a numerical quadrature to calculate the integral. This is a procedure intimately linked to the set of "measurement" positions $\{(\mathbf{r}, \mathbf{t})_0, (\mathbf{r}, \mathbf{t})_1, \ldots, (\mathbf{r}, \mathbf{t})_{N-1}\}$ and will be elaborated later. The scoring distribution is now accessible for methods of probability theory. This has several advantages. Scoring distributions $\mathcal{S}_1(\mathbf{r}, \mathbf{t})$ and $\mathcal{S}_2(\mathbf{r}, \mathbf{t})$ that

arise from different situations are now comparable. This holds for a range of variations in the basic setup. One could employ different scoring methods or two different query structures, or different values of parameters of data preprocessing.
This provides the basis for the central argument of what a "good" scoring distribution is. Given two different scoring distributions I *define* that scoring distribution to be the better one which exhibits smaller ambiguity. This *definition* is very much in the sense of the modeler. Structural integrative modelings sole purpose could be described as the maximal possible reduction of ambiguity of possible models (the ideal case would be to have just one possible model). Therefore we have an interest in reducing ambiguity in this very first step of the process. The next part of the argument pertains to a concrete definition of the intuitive term "ambiguity".
Let $\mathcal{S}(\mathbf{r}, \mathbf{t})$ be a probability distribution obtained from a scoring distribution in the manner outlined above. Then it is possible to calculate the *entropy* of this distribution:

$$\mathcal{H}(\mathcal{S}) = - \int\limits_{\mathcal{SO}(3)\times\mathbb{R}^3} \mathcal{S}(\mathbf{r}, \mathbf{t}) \log \mathcal{S}(\mathbf{r}, \mathbf{t}) \, \mathrm{d}\mathbf{r} \, \mathrm{d}\mathbf{t}$$

The integral itself has again to be approximated by numerical quadrature. Entropy is a measure of uncertainty. It has two properties that make it suitable as the measure of ambiguity needed. It is bounded above for a uniform distribution and is 0 for a distribution which signifies a certain outcome (see 4.3 for details). Entropy can therefore serve the purpose of a metric of comparison between two different scoring setups. The normalized entropy (see 4.3) can serve as comparison between sampling spaces of different sizes. The smallest entropy will correspond to the least ambiguous scoring.

5.12 Entropy measurements of systematic fitting score populations using model densities

A central part of this thesis is the attempt to measure the ambiguity inherent in systematic fitting. This section describes these attempts. Several tentative entropy measurements have been conducted to evaluate if the hypothesis laid out in 5.11 translates to actual applications. Several visualization methods have been used to provide an accessible intuition of the results. The test cases are constructed, simplified and in a sense artificial to allow for a test of the *basic hypothesis*, independent from the complications that are expected when actual data (e.g. real density maps resulting from single particle cryo-EM or cryo-ET acquisitions and crystal structures) are used.

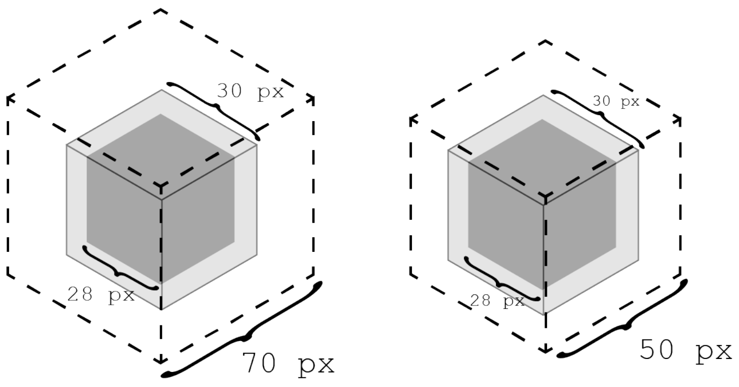
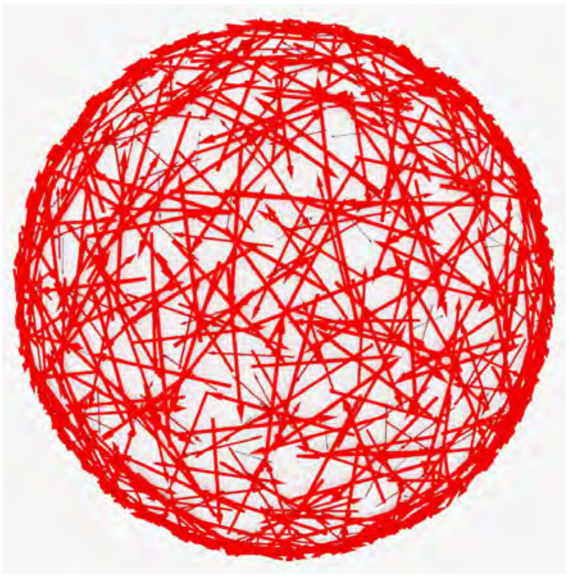


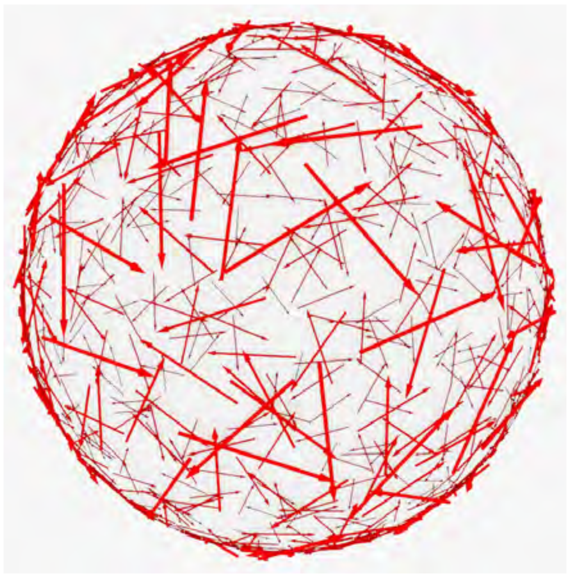
Figure 5.42: A schematic illustration the densities used for the scoring test.

5.12.1 Comparing different scoring methods

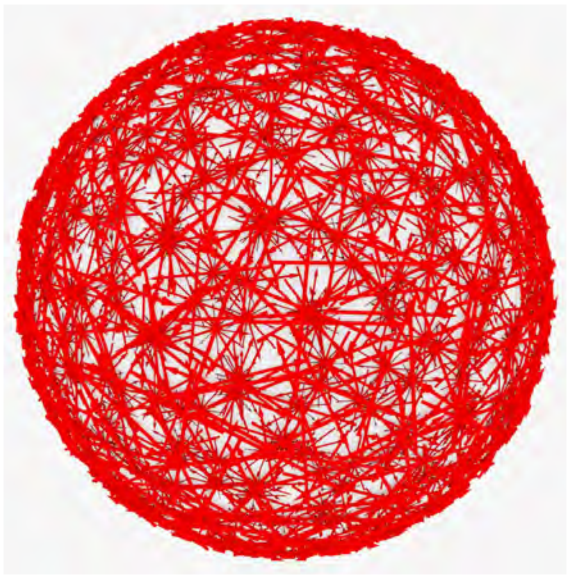
For the three of the scoring methods that are estimated to be suitable for the fitting of surfaces, namely the Chamfer distance, the envelope score (see for both (2.5.5)) and the partial surface score described in this thesis (5.9) a rotational measurement has been conducted. The test case used are two densities that are congruent cubes as illustrated in figure 5.42. The goal of this test was to evaluate of the measurement produces the expected "peaks" and to assess wether an entropy measurement would behave as expected. The expectation was that the partial surface score would at least outperform the other two scoring methods that were used.

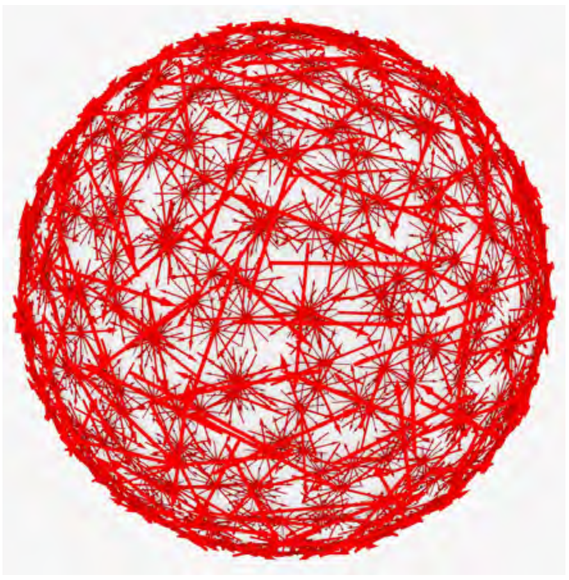
Figure (5.43) shows visualizations of the rotational samplings. The translation of the query relative to the target is always such that both are placed in each others geometric centers. The position and the orientation of the arrows on the sphere denote the orientation of the query relative to the target and the size and color symbolize the magnitude of the score. A large, red arrow symbolizes a high score and a small, black arrow a low score. The two rotational sampling sets, one of the size `1776` and the other of the size `5880`, where taken from [78]. Looking at the population of the arrows as a whole, a uniform population of equally size red arrows can be seen as the worst case with regards to ambiguity and a heterogeneous population with regard to color and size can be seen as an unambiguous sampling.
The specific order of rotational sampling nodes in this specific set of nodes offers another perspective on the same data. Figure (5.44) shows the result displayed as a distribution. The specific order of the samples is a fortunate coincidence. The differences in "peak width" are clearly visible. Also, all 24 peaks coincide, which hints at the consistency of all three scoring methods (see 6.3.1). The number 24 confirms a complete and consistent sampling, since the number of possibilities to rotate a cube so that it is congruent to itself is 24.


(a) The chamfer distance for 1776 nodes.


(b) The envelope score for 1776 nodes.

(c) The partial surface score for 1776 nodes.


(d) The chamfer distance for 1776 nodes.


(e) The envelope score for 1776 nodes.

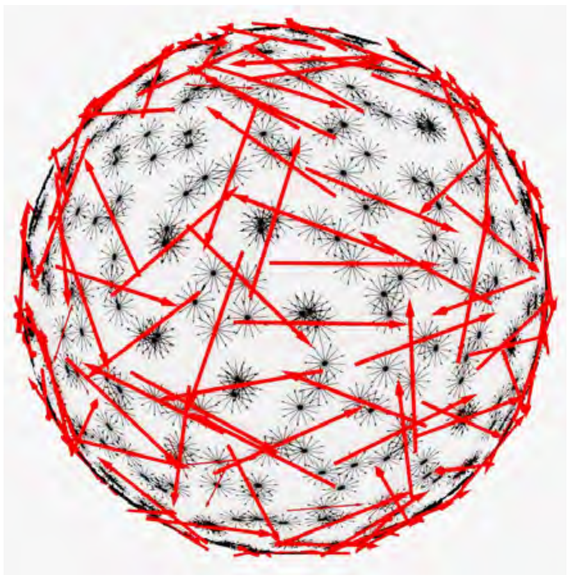
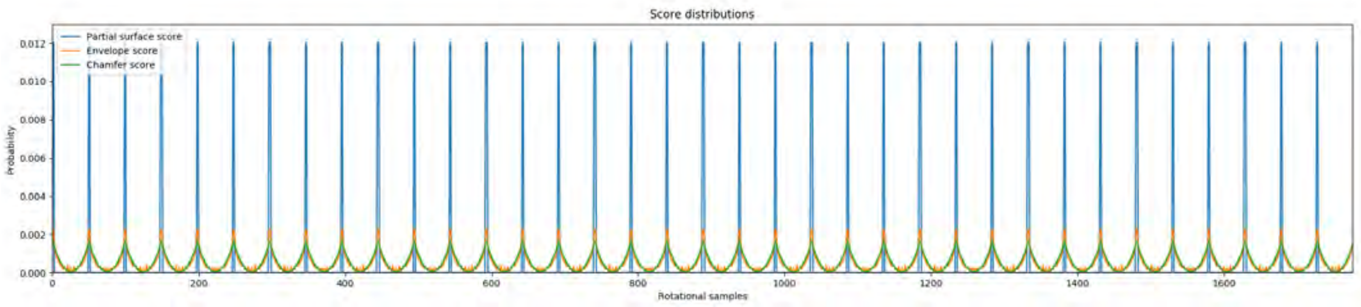
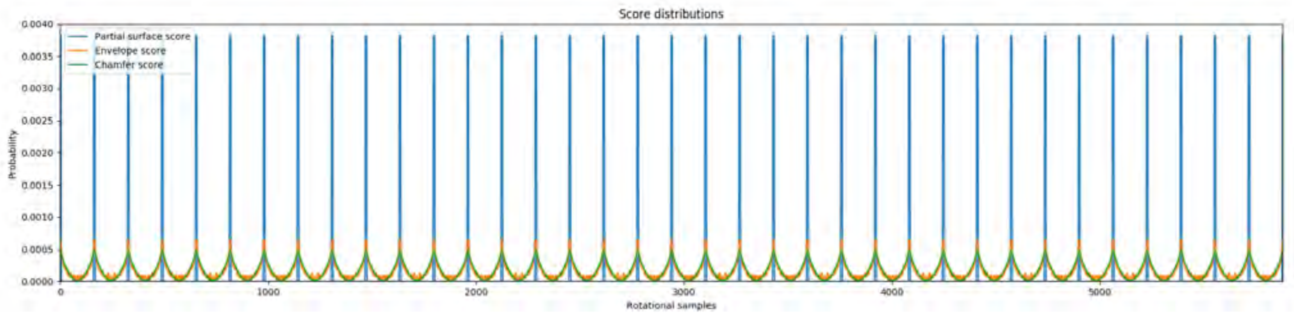

(f) The partial surface score for 1776 nodes.

Figure 5.43: Visualization of different rotational scoring as outlined in 4.2.5



(a) Scoring distribution for different scoring types with 1776 rotational nodes.



(b) Scoring distribution for different scoring types with 5880 rotational nodes.

Figure 5.44: Scoring distributions for two rotational sample set sizes.

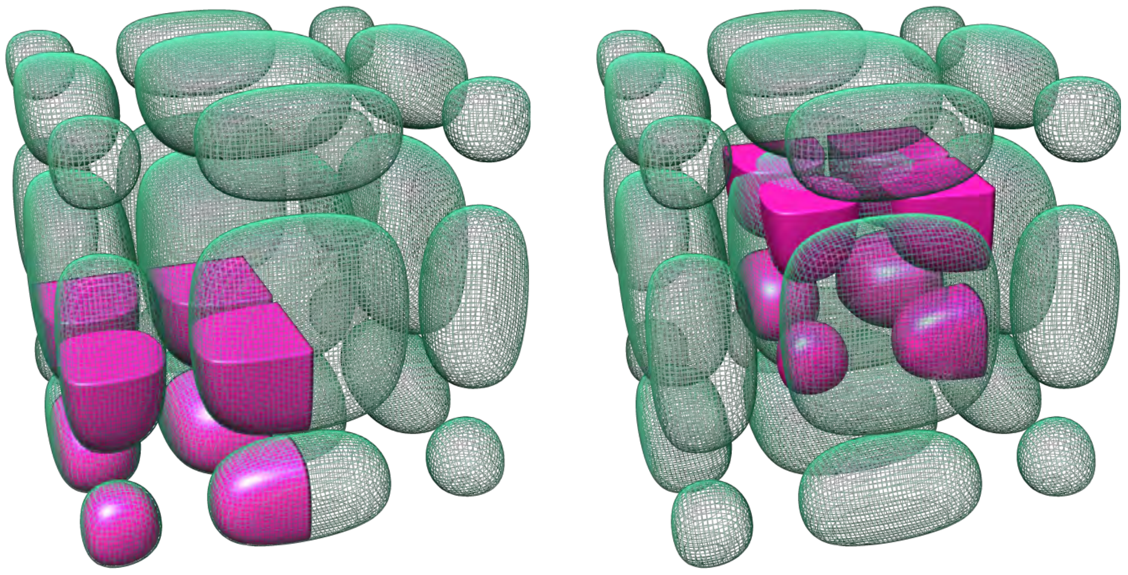
Scoring method	H 1776 nodes	H 1776 nodes	$\|\Delta H\|$
chamfer	0.9813	0.9295	0.0518
envelope	0.9450	0.9591	0.0141
partial	0.5917	0.6295	0.0378

Table 5.1: Normalized entropy measurements of three scoring methods and two rotational sampling sets.

The normalized entropy was measured with the results shown in table (5.1). The normalized entropy has the value range $[0, 1]$. 1 would represent a completely ambiguous distribution with the lowest information content. A distinct difference between the partial surface score and the other methods is visible. Also, the entropy measurement is approximately stable, as can be seen in the third column $|\Delta H|$. A surprising result is the negligible difference between the Chamfer distance and the envelope score. This is potentially useful because the envelope score is computationally much cheaper than the Chamfer distance.

5.12.2 Comparing different translational positions

To assess if entropy could be used to assess the "local complexity" of a given translational placement of the query relative to the target a slightly more complex test case has been


(a) Teal target map with a cut out part of it as a query.

(b) Teal target map with a cut out part of it as a query, shifted to the center.

Figure 5.45: Two different translational samples of a constructed target and query.

constructed (see figure 5.45) and assessed. The chosen scoring method was the masked cam-score (see 5.7.1). To ensure a good signal for comparisons sake the query density has been cut out (see 5.45a) for one translation (well-fitting) and the same cut out query shifted to the middle of the target for the second translation (ill-fitting). For direct visual assessment the resulting sample set of $SO(3)$ has been projected onto a rotating sphere. The angles (α, β) of the Euler angles (see 4.2.3) are mapped onto a static sphere and the remaining angle γ is mapped onto the time dimension, resulting in a movie that gives an immediate apprehension of the rotational sampling. Figure (5.46) shows two frames of the complete $SO(3)$-sampling, corresponding to the different two translational placements (see 5.45) and the same angle γ_0. The entropy measurement of the rotational sampling represented by figure 5.46a resulted in a lower value than the measurement of the rotational sampling represented by figure 5.46b. This is also visible as a more concentrated distribution of the scoring values. It is therefore conceivable to use entropy measurements as a metric for an adaptive search.

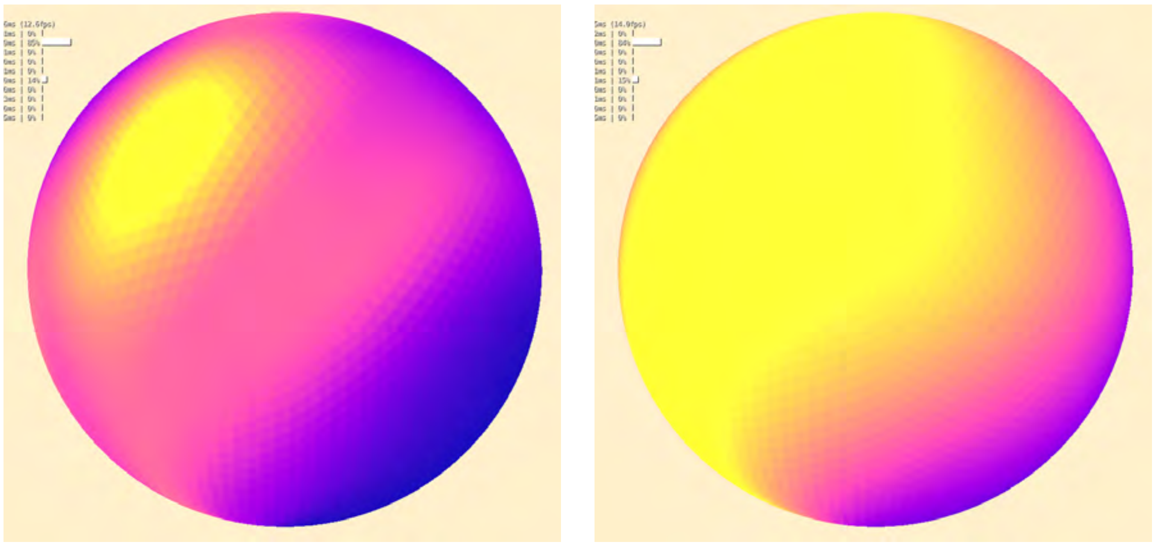

(a) "Well-fitting", low entropy placement. (b) "Ill-fitting", high entropy placement.

Figure 5.46: Graphical representation of the (α, β) set of a specific γ_0 in a cam-score rotational sampling. Yellow corresponds to a high score, blue to a low score.

5.13 Entropy measurements of systematic fitting score populations using experimental data

This section contains the workflow definition and the results of entropy measurements using the data of the TFIIIC - complex (see 5.1). After initial tests of the behavior of entropy measurement (see 5.12) this serves to assess the use of it in practical situations. To do this, two application cases have been tested: first, the comparison of the entropies of different scoring methods and second the comparison of the entropies of different parameter ("resolution" parameter) choices for the query density simulation (see 3.2.3). To accomplish both goals, the following protocol has been employed:

1. For the case of the entropy measurement regarding different parameter choices for density simulation, the different densities have been generated using the method described in 5.4. For the case of the entropy measurement of different scoring methods one density has been generated using the same method and the resolution parameter that data was annotated with.

2. The results of the search algorithm outlined above (see 5.6), that was already tested on the TFIIIC case, were used to ensure that the entropy measurement was conducted in the vicinity of a local optimum (maximum in the case of both scoring methods).

3. An FFT-accelerated multi-GPU program has been implemented for two scoring methods: The overlap score (see 2.4) and the masked version of the correlation-

about-mean score (see 2.5.5 and 2.7). For the latter, a clean definition had to be derived first.

4. Using these programs, a large population of scores have been calculated and their entropies have been measured using McEwen quadrature (see 4.36) for the rotational dimension of the search space.

These points, in so far as they are novel, will be discussed briefly in the following sections.

5.13.1 Definition of the masked correlation-about-mean score

To enable efficient computation and the best possible use of the FFT-acceleration feature, the mathematical form of the score should be simplified as much as possible. The standard definition of the correlation-about-mean ("cam"-score) is

$$\mathrm{cam}\,(\mathrm{t},\mathrm{q}) = \frac{\langle \mathrm{t} - \bar{\mathrm{t}}\mathbb{1} | \mathrm{q} - \bar{\mathrm{q}}\mathbb{1}\rangle}{\sqrt{\langle \mathrm{t} - \bar{\mathrm{t}}\mathbb{1} | \mathrm{t} - \bar{\mathrm{t}}\mathbb{1}\rangle\langle \mathrm{q} - \bar{\mathrm{q}}\mathbb{1} | \mathrm{q} - \bar{\mathrm{q}}\mathbb{1}\rangle}} \tag{5.7}$$

Where t and q denote the target and query densities, respectively. They are understood as vectors. $\bar{\mathrm{t}}$ and $\bar{\mathrm{q}}$ denote their averages over their total volume and $\mathbb{1}$ denotes the density of respective indicator function (every entry is 1). This formulation does not allow for masks yet. To reduce the influence of target regions, where the query is not placed for a given transformation, on the score a query mask $\mathrm{m_q}$ if defined for a certain threshold. It can be applied by element-wise vector multiplication. The masked version of the cam score now reads:

$$\mathrm{cam}\,(\mathrm{t}\cdot\mathrm{m_q},\mathrm{q}) = \frac{\langle \mathrm{t}\cdot\mathrm{m_q} - \overline{\mathrm{t}\cdot\mathrm{m_q}}\mathrm{m_q} | \mathrm{q} - \bar{\mathrm{q}}\mathrm{m_q}\rangle}{\sqrt{\langle \mathrm{t}\cdot\mathrm{m_q} - \overline{\mathrm{t}\cdot\mathrm{m_q}}\mathrm{m_q} | \mathrm{t}\cdot\mathrm{m_q} - \overline{\mathrm{t}\cdot\mathrm{m_q}}\mathrm{m_q}\rangle\langle \mathrm{q} - \bar{\mathrm{q}}\mathrm{m_q} | \mathrm{q} - \bar{\mathrm{q}}\mathrm{m_q}\rangle}} \tag{5.8}$$

Note that $\mathrm{m_q}\cdot\mathrm{q} = \mathrm{q}$, if the query density is masked at the beginning of the processing. The factor $\langle \mathrm{q} - \bar{\mathrm{q}} | \mathrm{q} - \bar{\mathrm{q}}\rangle$ only refers to the query density and does not change if the query density is translated and rotated. Because it will be constant for all transformations, it can be calculated before the scoring itself.
The other two factors can be simplified in a similar manner. The numerator can be expanded like so:

$$\langle \mathrm{t}\cdot\mathrm{m_q} - \overline{\mathrm{t}\cdot\mathrm{m_q}}\mathrm{m_q} | \mathrm{q} - \bar{\mathrm{q}}\mathrm{m_q}\rangle = \langle \mathrm{t}\cdot\mathrm{m_q} | \mathrm{q}\rangle - \langle \mathrm{t}\cdot\mathrm{m_q} | \bar{\mathrm{q}}\mathrm{m_q}\rangle - \langle \overline{\mathrm{t}\cdot\mathrm{m_q}}\mathrm{m_q} | \mathrm{q}\rangle + \langle \overline{\mathrm{t}\cdot\mathrm{m_q}}\mathrm{m_q} | \bar{\mathrm{q}}\mathrm{m_q}\rangle$$

The first two terms can be simplified via $\langle \mathrm{t} \cdot \mathrm{m_q} | \mathrm{q} \rangle = \langle \mathrm{t} | \mathrm{m_q} \cdot \mathrm{q} \rangle = \langle \mathrm{t} | \mathrm{q} \rangle$. Additionally, the scalar factors can be pulled out of the scalar products:

$$\begin{aligned}
&\langle \mathrm{t} \cdot \mathrm{m_q} | \mathrm{q} \rangle - \langle \mathrm{t} \cdot \mathrm{m_q} | \overline{\mathrm{q}}\mathrm{m_q} \rangle - \langle \overline{\mathrm{t} \cdot \mathrm{m_q}}\mathrm{m_q} | \mathrm{q} \rangle + \langle \overline{\mathrm{t} \cdot \mathrm{m_q}}\mathrm{m_q} | \overline{\mathrm{q}}\mathrm{m_q} \rangle \\
&\quad = \langle \mathrm{t} | \mathrm{q} \rangle - \overline{\mathrm{q}}\langle \mathrm{t} | \mathrm{m_q} \rangle - \overline{\mathrm{t} \cdot \mathrm{m_q}}\langle \mathrm{m_q} | \mathrm{q} \rangle + \overline{\mathrm{t} \cdot \mathrm{m_q}}\,\overline{\mathrm{q}}\langle \mathrm{m_q} | \mathrm{m_q} \rangle
\end{aligned} \tag{5.9}$$

Using the definition of the average , the last term can be rewritten:

$$\overline{\mathrm{t} \cdot \mathrm{m_q}}\,\overline{\mathrm{q}}\langle \mathrm{m_q} | \mathrm{m_q} \rangle = \overline{\mathrm{t} \cdot \mathrm{m_q}}\frac{\langle \mathrm{q} | \mathrm{m_q} \rangle}{\langle \mathrm{m_q} | \mathrm{m_q} \rangle}\langle \mathrm{m_q} | \mathrm{m_q} \rangle = \overline{\mathrm{t} \cdot \mathrm{m_q}}\langle \mathrm{q} | \mathrm{m_q} \rangle$$

This term cancels out the second to last in 5.9. The numerator reads then:

$$\langle \mathrm{t} \cdot \mathrm{m_q} - \overline{\mathrm{t} \cdot \mathrm{m_q}}\mathrm{m_q} | \mathrm{q} - \overline{\mathrm{q}}\mathrm{m_q} \rangle = \langle \mathrm{t} | \mathrm{q} \rangle - \overline{\mathrm{q}}\langle \mathrm{t} | \mathrm{m_q} \rangle$$

The average $\overline{\mathrm{q}}$ is also constant and can be calculated before the actual scoring. An similar argument can be made for the second factor of the denominator in 5.7:

$$\langle \mathrm{t} \cdot \mathrm{m_q} - \overline{\mathrm{t} \cdot \mathrm{m_q}}\mathrm{m_q} | \mathrm{t} \cdot \mathrm{m_q} - \overline{\mathrm{t} \cdot \mathrm{m_q}}\mathrm{m_q} \rangle = \langle \mathrm{t}^2 | \mathrm{m_q} \rangle - \frac{1}{\langle \mathrm{m_q} | \mathrm{m_q} \rangle}\langle \mathrm{t} | \mathrm{m_q} \rangle$$

where t^2 is the element-wise squared target density and $\frac{1}{\langle \mathrm{m_q} | \mathrm{m_q} \rangle}$ is again a constant. In conclusion the only quantities that need to be recalculated per transformation are: $\langle \mathrm{t} | \mathrm{q} \rangle, \langle \mathrm{t} | \mathrm{m_q} \rangle, \langle \mathrm{t}^2 | \mathrm{m_q} \rangle$. As outlined in 3.1.1, these terms and similar terms for translations of q with regard to t can be very efficiently calculated by interpreting them as samples of two convoluted signals. This is used in the GPU implementation.

5.13.2 GPU implementation, user interface and benchmark

The preceding discussion of the mathematical procedure in conjunction with the convolution theorem discussed in 3.1.1 have been used to construct the computational workflow of the GPU-accelerated masked correlation-about-mean score implementation, which will be discussed here in brief. For a lot of aspects of the implementation the multi-GPU modeling classes and the GPU processing models outlined above have been utilized. Therefore, the implementation can use any number of GPUs of any kind to distribute the work optimally. The `CUDA CUFFT` library has been used to calculate the three-dimensional FFT transformation and its inverse. The implementation divides into the following steps:

- In the first step, the relevant data are loaded, initialized and masked given the specific user input. The rotational McEwen nodes are initialized and an offset is calculated to make sure that the user-input maximal rotation is caught in at least one node. The user-input maximal scoring translation is used to cut out a predefined window in the target density around that translation. The user-input optimal translation and rotation need only be approximations. After initialization of the data resources, `CUDA CUFFT` library function and `CUDA` memory management functions are used to assess how many rotations can be calculated simultaneously scored on the respective GPU. This information is referred to the main thread so it can be shared with other GPUs.

- Before the actual score calculation is conducted, the quantities that have been mentioned to be constant in 5.13.1 are calculated and stored in memory. This includes the averages and norms mentioned as well as the Fourier transformation of the target density and its square. To get the correct score later on, the target density must be embedded in an pixel space of pixel dimension $\mathrm{target}_{\mathrm{dim}} + \mathrm{query}_{\mathrm{dim}} - 1$. The rotational offsets and local volumes, which determine which rotation is scored by which GPU, are calculated. Finally the memory space for the simultaneously scored rotations is allocated to enter the main processing loop.
- Figure (5.47) outlines the basic workflow of the score calculation. The figure represents what happens for one specific rotation, index by the rotation index $\mathrm{r_i}$. On a given GPU g, a number $\mathrm{R_g}$ of rotation indexes is calculated simultaneously. This is because the Fourier transforms are more efficiently calculated in big batches. The calculation mirrors the mathematical definition above very closely. Once a batch of rotational indexes $\mathrm{R_g}$ is scored, the result is saved to `mrc` and the averages are saved to `txt`. Every GPU g participating in the calculation will output its own set of `mrcs` files and its own averages `txt` file.
- In the final processing step the files produced in the preceding step are loaded and processed as outlined in 5.11. The entropy score is then written to the standard output.

The user interface to generate the 6-dimensional scoring distribution is self-explanatory:

```
cam_fft target.mrc query.mrc /working_dir/
```

Where `/working_dir/` will hold the final result. To calculate the entropy of a given distribution one would evoke

```
cam_ent target.mrc query.mrc /working_dir/
```

Not yet exposed but easily adaptable are the parameters that define the rotation grid and query and target density thresholds. A program analog to `cam_fft`, the program `ov_fft`

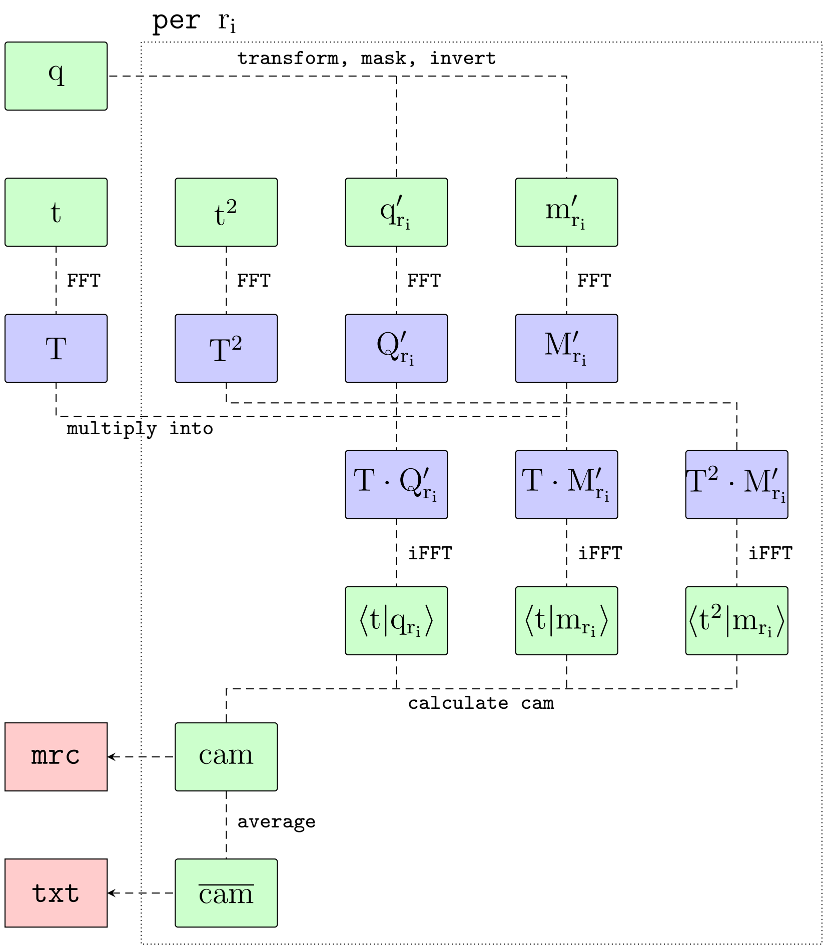


Figure 5.47: Representation of the masked correlation-about-mean calculation workflow. Green rectangles correspond to densities in real space, blue rectangles to densities in complex space.

has been implemented to calculate distributions for the overlap score.
A preliminary benchmark has been conducted, using 4 `NVIDIA GeForce GTX 3090` GPUs. The effective number of transformations per second is $550.000.000 \frac{\text{transformations}}{\text{second}}$.

5.13.3 Comparison of entropies of different scoring types

The implementation of 5.13.2 was used on the TFIIIC case discussed above. A query density was generated from the structure of the TPR-domain of τ_{131} (`PDB: 5AEM`) using the resolution parameter 5.0. The query density was placed cropped/extended to a density of pixel dimension $(100, 100, 100)$ and the target density, the map of τ_A annotated with 10Å, was cropped around the position $t_{\max} = (29.9078, 64.0547, 61.1141)$ (in pixels), which was the local optimal point found by the above mentioned adaptive search algorithm (see 5.6). A rotational grid using McEwen grid nodes of dimension $(32, 16, 32)$ was created resulting in 16384 rotational nodes. Due to the FFT-accelerated features of the fitting engine described above, every possible translation of the query density against the target density was score, by the pixel, resulting in a $(199, 199, 199)$ scoring volume for every rotation. In total 129.115.734.016 transformations have been scored. The rotational grid nodes were oriented so that one of them (the one of index $(0, 0, 0)$) would coincide with $q_{\max} = (0.24625, 0.36173, 0.83875, -0.32405)$, at the rotation containing the predicted maximum.
This setup was used to score the overlap score and the masked version of the correlation-about-mean score defined above. To exclude numerical noise the result of latter scoring was also capped at a value of 0.2, which is slightly below the general noise level. While the actual scores were calculated, the average per rotation was calculated and saved to file. The average is needed to calculate the entropies during the next step. The scores themselves were saved as `mrc` files. Together, these files were used in a separate step to calculate the respective entropies of the scoring distributions. Figure (5.48) shows a rendering of two different scoring densities at the rotation that was predicted to hold the maximal score. Every pixel in these densities corresponds to a unique translation. The difference in volume and spread is observable, which one might expect considering the definitions of the respective scores (see 2.5.5 for an exposition).
The entropy measurement itself also reflects this.

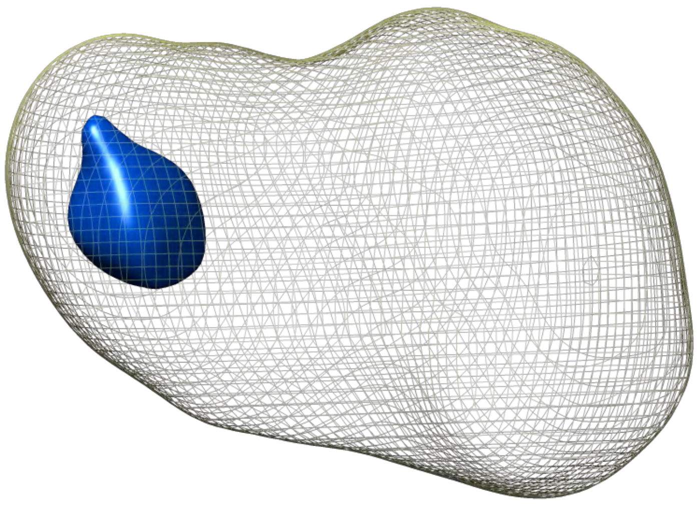


Figure 5.48: A graphical rendering of two different scoring densities at a fixed rotation. The meshed surface corresponds to the overlap score, the blue density to the masked correlation-about-mean score. Both isosurfaces are rendered at approximately 55% of their value ranges.

Distribution	Entropy
Uniform distribution	20.248816
Overlap-score distribution	17.844624
Masked cam-score distribution	15.728468
Rotational peak distribution	8.600217

Table 5.2: Entropies for different distributions in the discussed search space.

Table (5.2) displays different entropy values for different distributions, the top and bottom value are calculated from theoretical distributions. The top value is the entropy of the uniform distribution. One can observe a lowering of entropy of the two empirical distributions in comparison to a uniform distribution, which is the distribution which holds the least amount of information. The overlap score distribution has a distinctly higher entropy than the masked correlation-about-mean score distribution. The bottom entropy is calculated for a distribution which is concentrated exclusively in one rotational grid node. Even in this unrealistic case the value is quite high. This highlights the significance of the difference between the entropy measurements of the two empirical distributions. This result shows that entropy measurements can be used to select on scoring method over others, i.e. to select the optimal scoring method.

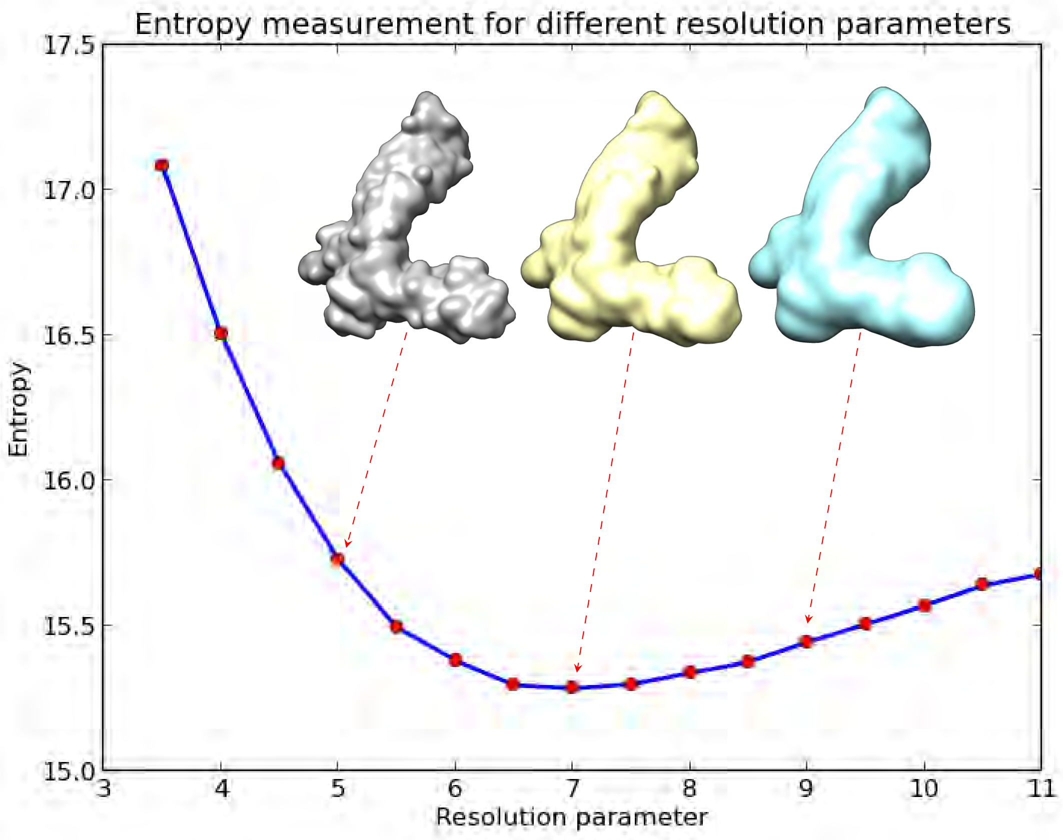


Figure 5.49: Entropy measurements for different resolution parameter values and some representative renderings of the densities that were used to create the score distribution.

5.13.4 Comparison of entropies to optimize parameters

This setup was extended to measure entropies of different fitting parameters. The parameter in question is the resolution parameter of the molecular density simulation. Using again the TFIIIC-case as an example, the parameter for density generation of the query density was set to every value of the range $r = \{3.5, 4, 4.5, \ldots, 11\}$. Using these values, densities were generated and scoring populations were calculated as discussed above. Then, the entropies were calculated. Figure (5.49) shows the results of the entropy measurement. A clear minimum is visible. This confirms the hypothesis that there are choices which are better or worse in the sense defined in 5.11. Additionally, the minimum is around the resolution parameter value 7. This differs from the annotated value of 10. If the data is to be used in the optimal sense (as defined above), the minimum of this curve should be chosen as a parameter value. This, together with the preceding result, shows that the method explored in this thesis can be used to assess the information content of a scoring distribution and therefore its usefulness in an integrative structural modeling project.

Chapter 6

Discussion

6.1 The TFIIIC complex modeling project

The TFIIIC modeling project exemplifies two aspects of integrative structural modeling. One aspect is the strong dependency on suitable data. For a integrative structural modeling project to succeed, all the data which limit the amount of possible models to a small number must be present in the beginning. The procedure itself can only find models the make the data coherent. In the case of the TFIIIC modeling project however, the data quality reached both limits: it was, in the case of the τ_A-complex, "too good" data and other techniques could be employed to solve for the structure. In the case of the TFIIIC negative stain map however, the data was not sufficient for more definitive models.
The second aspect is the use that integrative structural modeling has as an intermediate step guiding the experimenter. If the attention of the modeler is directed at the question: Given the current, perhaps incomplete, information, what models are the most likely? This model ensemble then can serve as a guide for the experimenter to evaluate the predictiveness and coherence of the accumulated data.

6.2 Systematic Fitting and Scoring functions

This discussion expands on issues raised in section 5.11. A systematic fitting protocol can be defined by its constitutive three elements, the *target*, which will be a density map, the *query*, which will either be a density map or a particle set, and a *scoring function*, whose only limitation (for now) is to be able to process its given input. Any of these elements comes with a set of degrees of freedom:

- The *target* density will invariably be modified by a density threshold, which is set to be at least to $t = 0$, because there are some densities with negative values. Additionally there might be image post processing operations, such as denoising via Gaussian filter, edge enhancement via Laplace filter (see 3.1.2), cropping or resampling. There might also be different versions of "the same" map resulting from varying post-processing steps in the single-particle cryo-EM or cryo-ET processing pipeline (see 2.4.2, 2.4.3).
- Similar influences underlie the *query*, if it is rendered as a density map. Additionally the query can undergo variations that address its nature as crystal structure, such as conformational variations via molecular dynamics or different experimental sources, breaking it up or selecting a different structure prediction tool. In the case that a structure must be converted to a density for the scoring function, a density generation method and its parameters must be selected.
- The *scoring function* can be selected from a variety of existing scores (see [46]), some of them require their own parameters to be set (such as the surface based scores, to define the surface, see e.g. 2.5.5, 2.5.5).

Not only are these three aspects connected to a great number of degrees of freedom, but their combination, which is necessary in a systematic fitting protocol, inherits all degrees of freedom. This means that any result of a systematic fitting protocol contributes to a great number of possible results, every one of this has some claim of being "the right one". In practice, the number of degrees of freedom is around 3-4 (e.g. the density thresholds of target and query, choosing a scoring method, the "resolution" parameter for the density map generation).
These observations inspire what can be called the central idea of this thesis: That the best choice of parameters or scoring methods is that choice which provides the most *information.* This criterion ultimately selects a distribution that is created by a triple (*target*, *query*, *scoring method*). For a set target and query it therefore selects a scoring method. It is agnostic towards the specifics of a scoring method, since it is based on the generated distribution itself.

6.3 Demonstrated applications and further possibilities

First applications and tentative proof of the validity of the proposed method can be seen in results 5.13.3 and 6.5.1. In the first result the lower entropy of the masked correlation-about-mean score compared to the entropy of the overlap score reflected the fact that the latter is much less "strict" than the former, which is apparent from the mathematical definition. This sort of comparison can then be conducted in any modeling project where the choice of score might not be clear, given the scores adhere to the definitions laid down in 6.3.1. This principle could be extended to the case where there are multiple target density or query structures or densities given: A ranking after entropy would give an objective measure which pairings "fit" the best.
The entropy measurement of different density generation ("molmap") resolution parameters has shown a way to select the optimal parameter where a parameter choice cannot be avoided. The advantage provided is that guess can be replaced with a conscious choice that minimizes entropy and maximizes information content. It remains to be seen if there is a dependency between annotated target resolution and entropy minimizing resolution and if the practical effect of choosing the information richest scoring distribution is noticeable. An interesting line of investigation might be the method of density generation itself, which, similarly to the choice of the resolution parameter, can take various different forms.

6.3.1 The optimal score

The abundance of possible scoring methods in systematic fitting warrants a short discussion on what a property a scoring method must exhibit at the very least. As outlined above, wether a score is a "good" score for a given systematic fitting protocol can ultimately only decided empirically. There are, however, some necessary preconditions for a function to be suitable for scoring:

- Scoring functions must be comparable. This is why all scores are either $\in \mathbb{N}$ or $\in \mathbb{R}$. While one might theoretically entertain the possibility that a score is $\in \mathbb{R}^2$, it would not be clear how to assess which score instance is "better" in many cases.
- Every target must fit itself. If query and target are the same object and the transformation that is scored is the identity transformation (translation and rotation do "nothing"), the score that is assigned by the scoring method must necessarily be at

least as greater or greater than all other transformations. If this is not so, the score can be called *inconsistent*.

- This leaves open the possibility for a *constant score*, which would simply assign the same value s_C to every possible transformation. This scoring method would have the highest possible entropy and can be considered the "worst" consistent score.
- Scoring functions do not need to be continuous, e.g. the envelope score (see 2.5.5). Continuity is here defined with regard to the transformations, meaning the behavior of the scoring function under small local rotations or translations. It has to be however continuous *almost everywhere* (in the stochastic sense), meaning it cannot "jump around" erratically over small local transformations.

It needs to be emphasized that the computational protocol to actually calculate a score is of no importance whatsoever. Therefore one should always choose an information-rich score with low computational cost in practice. This opens up interesting avenues of investigation, for example, nothing speaks against constructing a new score of already given scores $s_0(\mathbf{t}, \mathbf{r})$ and $s_1(\mathbf{t}, \mathbf{r})$ by some means of combining them:

$$s_n(\mathbf{t}, \mathbf{r}) = f\left(s_0(\mathbf{t}, \mathbf{r}), s_1(\mathbf{t}, \mathbf{r})\right) \tag{6.1}$$

Which will produce a new valid score as long as f preserves the properties outlaid above.

6.4 Regular sampling and numerical quadrature

The transformational grid that is used to scan query against target is, together with the scoring method employed, the most decisive factor in systematic fitting protocols and therefore fundamental to the integrative structural modeling project as a whole. The grid will most generally sample from 6-dimensional space, except in cases where the orientation of position is restricted from the very beginning (as in the case of modeling of proteins/protein complexes in a membrane). One might distinguish between the following categories of grids:

- A Monte-Carlo sampling. Both the translational and rotational dimensions are drawn from a (supposedly) uniform distribution. This approach has the advantage of being on a superficial level conceptually simple, on the downside, it requires a statistical analysis in the end to ensure that a sampling was exhaustive. An example is the FitInMap tools in Chimera (see [28]).

- A coverage-optimized sampling. A sampling of predefined nodes that seeks to cover the search space as efficiently as possible. This approach requires some notion of exhaustiveness and, if this notion is defined, can simply state that for a given level of exhaustiveness the smallest number of nodes is optimal. The challenge of this task mainly lies in solving this question for the rotational aspect of the grid, due to the more unfamiliar structure of the underlying group, $SO(3)$. One example of a notion of exhaustiveness can be found in [75]. Here the "coverage" is defined as

$$c = \frac{N\,(\alpha - \sin\alpha)}{\pi} \tag{6.2}$$

  where α is the maximum angle of the rotation needed for any given arbitrary rotation to be aligned with a rotation on the grid (so to speak the "distance" to the "nearest point" on the grid). N denotes the number of rotational nodes. This approach is suited most to expensive scoring methods for an unbiased search. Another approach was taken by [78] and [86], who seek to optimize quadrature rules rather than grid coverage. This approach is best suited if one seeks to calculate metrics such as the average, entropy or statistical moments over the rotational dimensions. Something similar holds true for the translational dimensions, where one can utilize Gaussian or Chebyshev nodes.

- Refinable grids form the last category. For the rotational dimension [79] and [75] offer solutions. To the authors knowledge most of the refinable grids offer also a quadrature formula. This makes them suited for adaptive searches, which is an interesting option for local minimum/maximum detection. An additional property of McEwen nodes for the rotational dimensions and Chebyshev nodes for the translational dimensions is that of a Fourier analysis and synthesis using orthogonal polynomials and an inner product (see 4.2.5 and 4.2.4).

An additional remark is called for when considering the case of `FFT`-accelerated fitting routines (see 3.1.1). These methods implicitly take the translational grid that is defined by the coordinates of the pixels. In the case of fast rotational fitting (as in [55]) the grid is defined by the maximal degree of the Wigner-D-functions that are employed to facilitate the calculation (see 4.31). These options will now be discussed under the aspect of several practical problems.

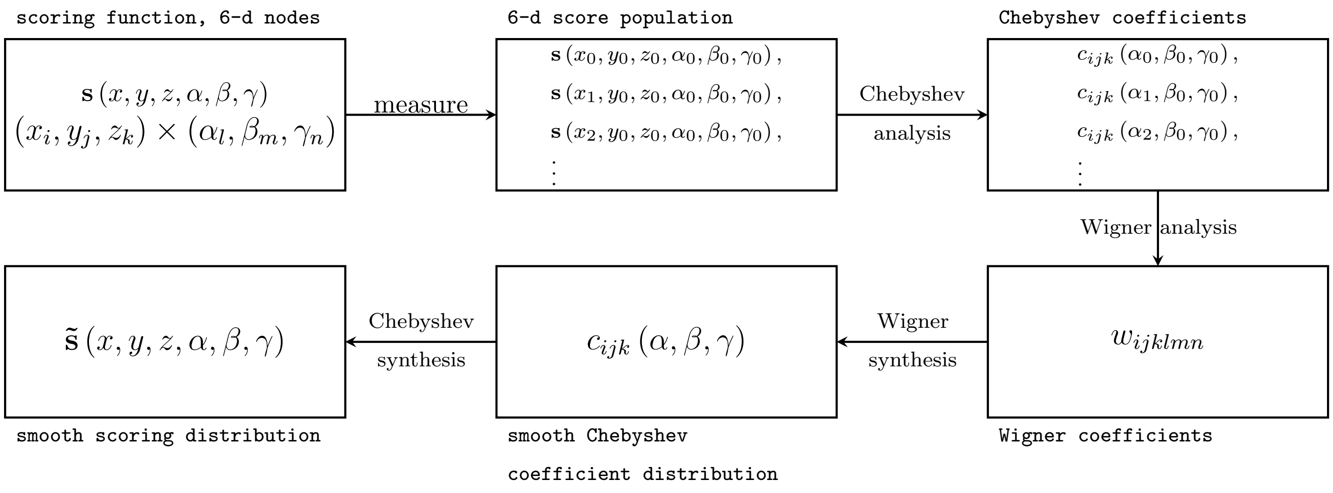
6.4.1 Finding local minima and maxima

Both regular and random search grids will have the issue of likely missing local minima or maxima of a scoring distribution. This poses an issue since, by definition, these locations

are the best possible fits. There are at least two ways to address the issue: A gradient and a refined search grid. The former is not always possible and the latter can very easily lead to prohibitive computational costs given the high dimension of the search space. The result detailed in 5.6 has provided a possible solution, which borrows from both gradient descend and search grid refinement. This method has advantages and disadvantages. It is adaptive, exhaustive and independent from the scoring method. The first two properties mean that any point in 6-dimensional space can be reached. The consecutive halving of every single search dimension leads to a very fast convergence. The convergent point however might not be local minima or maxima, but just borders of the initial cells. An easy workaround for this would be to run a last, independent check on every solution provided. Additionally, due to the construction of the descend algorithm, tricks like FFT-accelerated search will not work. The scoring method will have to be implemented in such a manner that it takes a pair (t, q) of a translational shift and a quaternion and returns a score. However, multiple of these calculations can be conducted in parallel.
A possible application is the exploration of local optima in the case of the systematic fitting of Brf1-TBP (see 5.1. In this case, the local minima of the cross-correlation score had to be used to calculate the Chamfer distance score and the envelope score. It is not guaranteed that the local optimal points of all three scores are the same, resulting in a possible loss of information.

6.4.2 Reconstructing scoring spaces

One major problem in integrative structural modeling in the *Assembline* pipeline is the need to represent the results of the systematic fitting step during the Monte-Carlo procedure. The common method of calculating the scores ad-hoc has been found the be too slow for large systems and a generous number of runs. The current method suffers from the fact that it is essentially a list of discrete positions and orientations and in one Monte-Carlo move one rigid body can only jump between these discrete nodes, which might potentially impair the convergence of the Monte-Carlo run, because rigid bodies are prevented from “sliding” past each other in smooth movements. Most scoring methods are inherently smooth, meaning a small rotation or translation will not cause a big jump in the score. However, to access the *complete* information contained in one systematic fitting protocol a lot of memory space is needed. Consider for example the result of a `FFT`-accelerated cam-score fitting protocol using `5880` rotational nodes, concerning a target of `(100,100,100)` pixels and a query of the same size. According to 3.1.1 the resulting score population per rotational node will also be of dimension `(100,100,100)` pixels, meaning in total $100^3 \cdot 5880$ `float` values. This amounts to roughly 21.9 GB memory space.
One possible way to address this issue is to change the way the scoring spaces are repre-



scoring function, 6-d nodes
$\mathbf{s}(x,y,z,\alpha,\beta,\gamma)$
$(x_i,y_j,z_k)\times(\alpha_l,\beta_m,\gamma_n)$
measure
6-d score population
$\mathbf{s}(x_0,y_0,z_0,\alpha_0,\beta_0,\gamma_0)$,
$\mathbf{s}(x_1,y_0,z_0,\alpha_0,\beta_0,\gamma_0)$,
$\mathbf{s}(x_2,y_0,z_0,\alpha_0,\beta_0,\gamma_0)$,
$\vdots$
Chebyshev analysis
Chebyshev coefficients
$c_{ijk}(\alpha_0,\beta_0,\gamma_0)$,
$c_{ijk}(\alpha_1,\beta_0,\gamma_0)$,
$c_{ijk}(\alpha_2,\beta_0,\gamma_0)$,
$\vdots$
Wigner analysis
w_{ijklmn}
Wigner coefficients
Wigner synthesis
$c_{ijk}(\alpha,\beta,\gamma)$
smooth Chebyshev coefficient distribution
Chebyshev synthesis
$\tilde{\mathbf{s}}(x,y,z,\alpha,\beta,\gamma)$
smooth scoring distribution

Figure 6.1: Conceptual workflow of the creation of a 6-dimensional scoring representation and its recovery.

sented and to use orthogonal polynomials. To represent a function using this technique, one can do something similar to a band-limited Fourier analysis (for further details see 4.2.5 and 4.2.4). One would first choose a band-limit for the polynomials, effectively limiting their degree and the amount of detail this representation is able to portray. Once this is done, an base set of polynomials is chosen and the score needs to be evaluated at specific nodes and, given these values, the Fourier coefficients with respect to this base set of polynomials can be calculated. As pointed out earlier, these coefficients serve as a form of building instruction. Now, using these coefficients, one can synthesize a reconstructed polynomial which can be evaluated at any point (not only the "input evaluation nodes") and a gradient can be taken of these polynomials. This enables gradient descent methods. For illustration, two methods to approach this issue in a practical manner will be outlined. A 6-dimensional space needs to be represented, part translational (isomorphous to $\mathbb{R}^3$) and part rotational (isomorphous to representations of $SO(3)$). The base polynomials chosen are the 3-dimensional Chebyshev polynomials (each one dimensional base polynomials degree-limited by T, meaning the total maximal degree is T^3) and the Wigner D-matrices (in turn limited by $(\mathrm{M},\mathrm{L},\mathrm{N})$). This will result in a set of translation nodes (see 4.28) $\{(x_0,y_0,z_0),(x_1,y_0,z_0),\ldots\}$ and a set of McEwen nodes (see 4.36) $\{(\alpha_0,\beta_0,\gamma_0),(\alpha_1,\beta_0,\gamma_0),\ldots\}$, which can be combined using the Cartesian product (see 5.5). Then, giving the measured scores, the Chebyshev expansion coefficients c_{ijk} can be calculated for each rotation $(\alpha_l,\beta_m,\gamma_n)$. The c_{ijk} can therefore be seen as a function of the Euler angles $c_{ijk}(\alpha,\beta,\gamma)$. In yet another step the Wigner coefficients w_{lmn} for every c_{ijk} can be calculated. This enables one to see the c_{ijk} as a smooth function of the Euler angles and therefore, in turn, to get the Chebyshev coefficients for any orientation and therefore the complete distribution, since from the Chebyshev coefficients one can synthesize a distribution in $\mathbb{R}^3$. Figure (6.1) shows an illustration of that process. The other method would be to inverse the order of Chebyshev and Wigner analysis. This process is to the authors knowledge up to the point of writing this purely speculative, so some cautious remarks and caveats seem in order:

- This tentative method offers two big advantages: Compact representation and mathematical tractability. The first advantage means that, in extreme cases, a 20GB scoring space might be represented in a few kilobytes. This, of course, depends on the complexity of the scoring space. For less complex spaces, the Wigner and Chebyshev coefficients of higher order will be 0. Also, the amount of non-zero higher order coefficients will give an immediate measure of information content of the scoring space. This has interestingly been corroborated by experiments which involved systematic fitting scoring spacing being compressed. More information rich spaces were harder to compress, which aligns well with basic perspectives from information theory.

- The second advantage, mathematical tractability, stems from the fact that the reconstruction is an analytic function. This offers the possibility to derivate or integrate it. This might result in massive advantages during the Monte-Carlo step of structural integrative modeling. The base polynomials themselves are all factorized, so that derivation is at least *analytically* simple.

- However, the compact representation poses *numerical* challenges. The pipeline laid out in 6.1 requires to construct new Chebyshev polynomials every time new Chebyshev coefficients are calculated. This is the trade-off between compactness of representation and computational time required for evaluation. It might be best to broker a compromise between both aspects in practical situations, such as synthesizing Chebyshev polynomials for a given set of rotational nodes at the beginning of the Monte-Carlo step.

- Even if this challenge is mastered, one would still need a measure to evaluate how well the representation encoded in the coefficients w_{ijklmn} is faithful to the actual scoring distribution. A direct way is to assess the size of the higher order coefficients, since they will approach 0 if the representation is faithful. Another way would be to assess the difference between the empirical score $\mathbf{s}$ and its reconstruction $\tilde{\mathbf{s}}$ at random six-dimensional nodes $(x, y, z, \alpha, \beta, \gamma)$.

Another point of view can be taken in light of 6.4.2. Since the local optimal points are the most likely locations (in 6-dimensional space) for the structures to be placed, on could attempt to expand the scoring space at these locations locally. This would have several advantages. There is, in any well-behaved scoring distribution, a significantly smaller number of optimal points than points overall. Local expansions are much simpler in the sense that they typically need a smaller degree of polynomial to approximate. In combination, this could lead to a very compact, faithful, tractable and fast representation of a given scoring distribution.

6.5 Information theoretic perspective on systematic fitting and integrative modeling in general

6.5.1 Scoring method assessment

As shown in , entropy measurements can be used to optimize parameter selection in structural integrative modeling projects. In this light, one might begin a structural integrative modeling project with a massive optimization procedure. It remains to be shown, whether this affects the model sampling step (the Monte Carlo runs) in the *Assembline* pipeline or similar pipelines. In theory, the individual sampling spaces of scoring distributions, which has been obtained with information theoretically optimized parameters, should be "narrower", while still retaining local optima. This could lead to a speed up in finding lower scoring models and "sharper" clusters of model ensembles.
As shown in the test systematic fitting in 5.12.1 and 5.13.3, entropy is a good metric to distinguish different scoring methods with regard to the ambiguity of their score distribution. The differences can be quite significant, as was shown. On a practical level, in an integrative structural modeling project, this difference can result in finding valid models or finding no significant models at all. It seems prudent to point out the varying heuristics that underlie the construction of the three scoring methods that have been compared. The Chamfer distance score is based on closest distances between pixels, the envelope score is based on pixel counts and the partial surface score is based on connected components. While these heuristics are valuable for the construction of new scores they are ultimately moot with regard to the actual performance of the scoring method. The same holds for the overlap score and the correlation-about-mean score. The entropy minimization method in this sense provides a method of assessing scoring methods, which is independent of assumptions about concrete assumptions about what constitutes a "good fit".
Another, yet unexplored alley is the use of further information theoretic measures. One interesting candidate among them is the Kullback-Leibler distance (see 4.3) $(\mathbf{s}_0 \parallel \mathbf{s}_1)$. It can be interpreted as relative entropy, which denotes how much information one would gain (or lose) if one would use one scoring method compared to the other. This could be used to assess if two scoring methods "tell us something different". For example, two

scoring methods with a comparable entropy and a non-zero Kullback-Leibler-Divergence would contain mutually exclusive information and it would pay off to consider *both* distributions as an informations source for integrative structural modeling.

6.5.2 Higher order considerations

In this section, the original considerations fueling the employment of information theoretic methods will be laid out. Structural integrative modeling will per definition be only applied if one source of experimental information is not enough to make a clear prediction for a structural model. This brings with it the possibility that even the combined information might not be enough to narrow down the space of possible configurations enough to speak of "several, well distinguished models". The converse question, if all possible models have been sampled according to the restraints put on them by experimental information, is called the exhaustiveness problem (see [45]). Given that a protein complex comprises N rigid bodies, being either subcomplexes, subunits, domains or single residues, the total "model space" would be a distribution of the form

$$\Psi\left(\mathbf{t}_0, \mathbf{r}_0, \mathbf{t}_1, \mathbf{r}_1, \ldots, \mathbf{t}_{\mathrm{N}-1}, \mathbf{r}_{\mathrm{N}-1}\right), \tag{6.3}$$

which describes all possible models and their respective likelihood. For example, the δ-distribution

$$\delta\left(\mathbf{t}_0 - \mathbf{t}_0^s, \mathbf{r}_0 - \mathbf{r}_0^s, \mathbf{t}_1^s - \mathbf{r}_0, \mathbf{r}_1 - \mathbf{r}_1^s, \ldots, \mathbf{t}_{\mathrm{N}-1} - \mathbf{t}_{\mathrm{N}-1}^s, \mathbf{r}_{\mathrm{N}-1} - \mathbf{r}_{\mathrm{N}-1}^s\right), \tag{6.4}$$

for some set of specific positions $\mathbf{t}_0^s, \mathbf{t}_1^s, \ldots, \mathbf{t}_{\mathrm{N}-1}^s$ and orientations $\mathbf{r}_0^s, \mathbf{r}_1^s, \ldots, \mathbf{r}_{\mathrm{N}-1}^s$ would describe one concrete model with infinite precision. It would therefore contain infinite information. The uniform distribution on the other hand

$$\mathbb{1}\left(\mathbf{t}_0, \mathbf{r}_0, \mathbf{t}_1, \mathbf{r}_1, \ldots, \mathbf{t}_{\mathrm{N}-1}, \mathbf{r}_{\mathrm{N}-1}\right), \tag{6.5}$$

encodes no information at all. It represents a complete lack of knowledge about the model. The question arises, how would one encode experimental knowledge this way? Firstly, one would need to model systematic fitting protocols, crosslinks, biochemical domain proximity and steric clashes as probability distributions, the possibility of which is reasonable to assume. For example, in a modeling project consisting only of two rigid bodies (N = 2), one can write Ψ as

$$\Psi_0\left(\mathbf{t}_0, \mathbf{r}_0, \mathbf{t}_1, \mathbf{r}_1\right) = \mathbf{S}_0\left(\mathbf{t}_0, \mathbf{r}_0\right) \cdot \mathbf{S}_1\left(\mathbf{t}_1, \mathbf{r}_1\right), \tag{6.6}$$

where $\mathbf{S}_0, \mathbf{S}_1$ denote the distributions derived from systematic fitting. Were the scoring methods perfect in the above sense, they would be δ-distributions. In practice however, they are more uncertain. This is why further restraints are needed. The information that two rigid bodies cannot occupy the same space, i.e. the avoidance of steric clashes, can be added to the model as such:

$$\Psi_1\left(\mathbf{t}_0, \mathbf{r}_0, \mathbf{t}_1, \mathbf{r}_1\right) = \mathbf{S}_0\left(\mathbf{t}_0, \mathbf{r}_0\right) \cdot \mathbf{S}_1\left(\mathbf{t}_1, \mathbf{r}_1\right) \cdot \mathbf{R}_0\left(\mathbf{t}_0, \mathbf{r}_0, \mathbf{t}_1, \mathbf{r}_1\right), \tag{6.7}$$

therefore introducing interaction between the two rigid bodies. The result of this multiplication might not be a distribution and likely needs to be normalized again. At this point one could now measure the Kullback-Leibler-Divergence

$$D(\Psi_0 \parallel \Psi_1) \tag{6.8}$$

to assess the information gain which was obtained by adding the restraint. Not every restraint will actually add information, e.g. a faulty crosslink might “scatter” the distribution Ψ. This sketch of a framework would enable several conceptual advances:

- The question of exhaustion could be addressed in a quantifiable sense. By choosing a limit for the entropy of Ψ, one would effectively set model precision. The higher the entropy, the more “wiggle space” would there be for each rigid body and the *ensemble* of models would be more uncertain.
- This formulation makes explicit that the result of every integrative structural modeling project is an ensemble of models.
- It becomes clearer why modeling projects with too small and too many rigid bodies are unfeasible: Ψ would have a lot of factors and each factor would contribute its own uncertainty to the overall entropy, if not balanced out by a myriad of restraints.
- By calculating $D(\Psi_0 \parallel \Psi_1)$, one could add restraints one after the other and assess how much more information they add. If they add a lot of information but are experimentally uncertain, one would have a problem. If they add no information at all, they either contain no information by experimental design or they are at odds with the already used restraints. This of course becomes computationally challenging if there are a lot of restraints.
- The framework laid out above is effectively one attempt to answer the question of exhaustiveness. Given a set of experimental restraints, what is the ensemble of likely and less likely models? This question amounts to searching the space wherein Ψ lives, even if one would model Ψ as a scoring function (the formulations are equivalent). These spaces are of very high dimension and it is highly questionable if

they can be sampled sufficiently to derive reliable statistics about these spaces. Until this question is answered, effectively every structural integrative modeling project presents in the best case samples of models without the assurance that these are the most likely ones.

6.6 Parallel computing and integrative structural modeling

Parallel computing has found its way into structural biology in general and into structural modeling specifically (e.g. PowerFit [53], gEMfitter [54], cryoSPARC [87]). The availability of modern day GPUs has improved, their integration with CPUs has become more prevalent and the programming interface ([67]) available to scientists has become more powerful. This thesis has dealt in large parts with the questions if problems in structural integrative modeling are treatable with these new computational resources (objective (D)).

6.6.1 Inherent suitability of structural data and algorithms in structural biology for parallel computing

The question if parallel computing is an avenue worth exploring in integrative structural modeling can be affirmed most emphatically. The reasons for this are manifold and interconnected.
One of the main factors is the prevalence of the density map data type in the field. GPU computing has evolved out of increased demands in computer graphics. The logic of transformations in 2- and 3-dimensional space, a subset of the theory of linear transformations in real vector spaces, was the very logic that GPU processing units were meant to address. A number of other computational procedures, such as filter operation (see 3.1.2) or the ubiquitous Fourier transform (see 3.1.1), have images and density maps as direct applications and are, by mathematical nature, linear and therefore amenable to parallel processing. The hardware features of modern GPUs have been modeled to accommodate the particular density data types and can be harnessed to great effect. The memory hierarchy (see 3.3.1) lends itself to enhancement of operations localized in a small subvolume. The texture memory feature (3.3.3) provides an often needed operation (texture fetching or density value extrapolation) to the modeler, with the added bonus of hardware acceleration.
From a higher-level perspective that is closer to systematic fitting and integrative struc-

tural modeling more possible gains of computational efficiency become apparent. This is the case for two main reasons. On the one hand, a lot of computational operations consist on their lowest level of instructions that operate independently *per-pixel*, *per-particle*, *per-pixel-per-pixel* or *per-particle-per-pixel*, so that they can be easily cast into a parallel form. On the other hand, systematic fitting needs to cover 6 dimensions when defined on a regular grid and the scoring function has to be executed *per-transformation* or *per-translation-per-rotation*. This, too, is easy to parallelize. This thesis contains a number of examples that deal with parallelization tasks in varied situations in systematic fitting, volume processing or particle alignment.

6.6.2 Implementation and Applications

As outlined in section 5.3, a programmer friendly framework yes been set up. There are two key aspects that define this specific implementation of a structural integrative modeling GPU framework. One is the necessity for a double representation of the data on both host and device memory. A particularly challenging aspect of this is the synchronization. Some tasks are handled better by traditional sequential CPU-implementations, some are suited better for a GPU approach. Many workflows consist of both kinds of operations in various orders of succession. Before and after each step there has to be a careful consideration of what data resource is needed on either CPU or GPU memory and what data resource needs to be synchronized to the other environment. A workable solution is the `Register` approach described in 5.3, where each data type is boiled down to its essentials and mirrored in GPU and CPU by various memory segments. The functions of a given data class (e.g. Density, Particles, Scores) then operate implicitly on the memory segments, hiding the details on the API level.
The second key aspect is the need for any forward-looking parallel processing framework to be able to address multiple GPUs. This challenge has been addressed by extending the `Register` approach to handle multiple representations of the same data structure for multiple GPUs. There is the issue of synchronization of multiple representations of the same object in CPU memory, this has been basically disregarded by having a CPU-version for each GPU-version of a given data object. This solution has been chosen with the idea in mind that objects that are processed on different GPUs might differ between each other after the process was completed. This means, in effect, that for a system with `G` GPUs and an object with the memory volume of `M`, `G*M` total CPU working memory will be needed. Although this restriction is of no further consequence in modern high performance computing environments, it is worth keeping in mind for the purposes of memory management in these environments.

From a process management perspective two possible modi operandi have emerged, as described in 5.3. One of them treats the multiple GPUs that may be present in the system as different workers and distributes "packages" of work, that they can solve independently from each other. The second modus is designed to process *massive* tasks. This modus utilizes all GPUs in parallel for the same task and synchronizes the results afterwards. In this modus synchronization between GPUs is an obvious bottle neck. Both modi have been implemented in a `DispatchQueue`-like engine, which makes both modi suitable to be run as a server which can receive tasks from outside, process them and provide the results. Both modes automatically detect the number of present GPUs and distribute the work accordingly.
On an application level different standard density volume processing routines have been implemented (see 5), as well as scoring methods with varying levels of refinement, with the option of adaptive refinement search procedures. Parallel `RMSD`-measurement routines and `RMSD`-alignment routines have been implemented as well, with the addition of an `RMF`-interface. With the exception of the latter two, every implementation is capable of harnessing multiple GPUs at the same time. Some applications adapt to the hardware specifications of different GPU models. This feature could easily be extended.

6.7 Partial surface score

The partial surface score was conceived as an alternative to the Chamfer distance and the envelope score that would not suffer their defects, specifically their inability to assign a high score to a matching partial surface. In the idealized case that guided the construction it outperforms the other scores, as also shown in a simplified case (see 5.12.1). It remains to be shown if it can be of value in practical applications. A simple way to measure its effectiveness would be to compare the average predicted position and orientation to the "true" position within an existing and experimentally validated model to the same metric using the Chamfer distance and the envelope score.
A potential problem might arise due to the very "sharpness" of the partial surface score. In a realistic setting, the query surface must be either simulated by heuristic means or an experimental density map signifying a subcomplex, both are subject to uncertainty. This might be remedied by choosing a thick surface, which in turn might obscure the very advantage of the partial surface score. Therefore one might want to define a "soft surface mask", which modifies the step-function density of the surface using a Gaussian distribution.

Bibliography

[1] Alexander von Appen, Jan Kosinski, Lenore Sparks, Alessandro Ori, Amanda L. DiGuilio, Benjamin Vollmer, Marie-Therese Mackmull, Niccolo Banterle, Luca Parca, Panagiotis Kastritis, Katarzyna Buczak, Shyamal Mosalaganti, Wim Hagen, Amparo Andres-Pons, Edward A. Lemke, Peer Bork, Wolfram Antonin, Joseph S. Glavy, Khanh Huy Bui, and Martin Beck. "In situ structural analysis of the human nuclear pore complex". In: *Nature* 526 (2015), pp. 140–143. DOI: `10.1038/nature15381`.

[2] J. (Joachim) Frank. *Three-dimensional electron microscopy of macromolecular assemblies : visualization of biological molecules in their native state / Joachim Frank.* [2nd ed.] Oxford ; New York: Oxford University Press, 2006.

[3] H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, and P.E. Bourne. "The Protein Data Bank." In: *Nucleic Acids Research* 28 (2000), pp. 235–242.

[4] M. Perutz, M. Rossmann, and A. et al. Cullis. "Structure of Hæmoglobin: A Three-Dimensional Fourier Synthesis at 5.5-Å. Resolution, Obtained by X-Ray Analysis." In: *Nature* 185 (1960), pp. 416–422.

[5] J. Kendrew, R. Dickerson, and B. et al. Strandberg. "Structure of Myoglobin: A Three-Dimensional Fourier Synthesis at 2 Å. Resolution." In: *Nature* 185 (1960), pp. 422–427.

[6] C. Giacovazzo, H.L. Monaco, International Union of Crystallography, G. Artioli, D. Viterbo, G. Ferraris, G. Gilli, G. Zanotti, and M. Catti. *Fundamentals of Crystallography.* IUCr texts on crystallography. Oxford University Press, 2002. ISBN: 9780198509585.

[7] G. Rhodes. *Crystallography Made Crystal Clear: A Guide for Users of Macromolecular Models.* Complementary Science. Elsevier Science, 2012. ISBN: 9780323137782. URL: `https://books.google.de/books?id=RVcnvWEbaM4C`.

[8] Alexander McPherson and Jose A Gavira. "Introduction to protein crystallization". In: *Acta Crystallographica Section F: Structural Biology Communications* 70.1 (2014), pp. 2–20.

[9] E. S. Ameh. "A review of basic crystallography and x-ray diffraction applications". In: *The International Journal of Advanced Manufacturing Technology* 105.7 (2019).

[10] G. Hunter, M. Vella, R. Bonetta, D. Farrugia, and T. Hunter. "The Structure of Protein Molecules: In Celebration of the International Year of Crystallography". In: *Xjenza Online* 2 (Mar. 2014).

[11] Christopher Russo. *Eukaryotic Ribosomes in Single Particle.* 2021. URL: https://www.academia-net.org/artikel/1256219.

[12] Yifan Cheng, Nikolaus Grigorieff, Pawel A Penczek, and Thomas Walz. "A primer to single-particle cryo-electron microscopy". In: *Cell (Cambridge)* 161.3 (2015), pp. 438–449. ISSN: 0092-8674.

[13] Georgios Skiniotis and Daniel R. Southworth. "Single-particle cryo-electron microscopy of macromolecular complexes". In: *Microscopy* 65.1 (Nov. 2015), pp. 9–22. ISSN: 2050-5698. DOI: 10.1093/jmicro/dfv366.

[14] Richard Henderson. "Avoiding the pitfalls of single particle cryo-electron microscopy: Einstein from noise". In: *Proceedings of the National Academy of Sciences* 110.45 (2013), pp. 18037–18041. ISSN: 0027-8424. DOI: 10.1073/pnas.1314449110.

[15] Alistair Siebert. *Data Collection Strategies: Tomography.* 2021. URL: https://www.ccpem.ac.uk/training/biochemsoc_sep2016/Siebert_cryoET%20data%20collection%20lecture.pdf.

[16] W. Wan and J.A.G. Briggs. "Chapter Thirteen - Cryo-Electron Tomography and Subtomogram Averaging". In: *The Resolution Revolution: Recent Advances In cryoEM.* Ed. by R.A. Crowther. Vol. 579. Methods in Enzymology. Academic Press, 2016, pp. 329–367. DOI: https://doi.org/10.1016/bs.mie.2016.04.014. URL: https://www.sciencedirect.com/science/article/pii/S0076687916300325.

[17] Daniel Castaño-Díez and Giulia Zanetti. "In situ structure determination by subtomogram averaging". In: *Current Opinion in Structural Biology* 58 (2019). Cryo electron microscopy • Biophysical and computational methods • Biophysical and computational methods - Part B, pp. 68–75. ISSN: 0959-440X. DOI: https://doi.org/10.1016/j.sbi.2019.05.011. URL: https://www.sciencedirect.com/science/article/pii/S0959440X19300454.

[18] Kendra E. Leigh, Paula P. Navarro, Stefano Scaramuzza, Wenbo Chen, Yingyi Zhang, Daniel Castaño-Díez, and Misha Kudryashev. "Chapter 11 - Subtomogram averaging from cryo-electron tomograms". In: *Three-Dimensional Electron Microscopy.* Ed. by Thomas Müller-Reichert and Gaia Pigino. Vol. 152. Methods in Cell Biology. Academic Press, 2019, pp. 217–259. DOI: https://doi.org/10.1016/bs.mcb.2019.04.003. URL: https://www.sciencedirect.com/science/article/pii/S0091679X19300548.

[19] Victor R.A. Dubach and Albert Guskov. "The Resolution in X-ray Crystallography and Single-Particle Cryogenic Electron Microscopy". In: *Crystals* 10.7 (2020). DOI: 10.3390/cryst10070580.

[20] Shaoxia Chen, Greg McMullan, Abdul R. Faruqi, Garib N. Murshudov, Judith M. Short, Sjors H. W. Scheres, and Richard Henderson. "High-resolution noise substitution to measure overfitting and validate resolution in 3D structure determination by single particle electron cryomicroscopy". In: *Ultramicroscopy* 135 (Dec. 2013), pp. 24–35. DOI: 10.1016/j.ultramic.2013.06.004.

[21] Pawel A. Penczek. "Resolution measures in molecular electron microscopy". In: *Methods in enzymology* 482 (2010), pp. 73–100. DOI: 10.1016/S0076-6879(10)82003-8.

[22] A. Kucukelbir, F.J. Sigworth, and H.D. Tagare. "Quantifying the Local Resolution of Cryo-EM Density Maps". In: *Nature Methods* 11 (Nov. 2014), pp. 63–65.

[23] UCSF Chimera. *Chimera Molmap.* 2021. URL: https://www.cgl.ucsf.edu/chimera/docs/UsersGuide/midas/molmap.html.

[24] Francis J O'Reilly and Juri Rappsilber. "Cross-linking mass spectrometry: methods and applications in structural, molecular and systems biology". English. In: *Nature Structural & Molecular Biology* (Oct. 2018). DOI: 10.1038/s41594-018-0147-0.

[25] Yoav Benjamini and Yosef Hochberg. "Controlling the false discovery rate: a practical and powerful approach to multiple testing". In: *Journal of the Royal statistical society: series B (Methodological)* 57.1 (1995), pp. 289–300.

[26] Vasileios Rantos, Kai Karius, and Jan Kosinski. "Integrative structural modeling of macromolecular complexes using Assembline". In: *Nature Protocols* 17.1 (2022), pp. 152–176. DOI: 10.1038/s41596-021-00640-z.

[27] Dina Schneidman-Duhovny, Andrea Rossi, Agustin Avila-Sakar, Seung Joong Kim, Javier VelÃ¡zquez-Muriel, Pavel Strop, Hong Liang, Kristin A. Krukenberg, Maofu Liao, Ho Min Kim, Solmaz Sobhanifar, Volker Dötsch, Arvind Rajpal, Jaume Pons, David A. Agard, Yifan Cheng, and Andrej Sali. "A method for integrative structure determination of protein-protein complexes". In: *Bioinformatics* 28.24 (Oct. 2012), pp. 3282–3289. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bts628.

[28] E.F. Pettersen, T.D. Goddard, C.C. Huang, G.S. Couch, D.M. Greenblatt, E.C. Meng, and T.E. Ferrin. "UCSF Chimera–a visualization system for exploratory research and analysis." In: *J Comput Chem.* (2004).

[29] Daniel Saltzberg, Charles H. Greenberg, Shruthi Viswanath, Ilan Chemmama, Ben Webb, Riccardo Pellarin, Ignacia Echeverria, and Andrej Sali. *Modeling Biological Complexes Using Integrative Modeling Platform.* New York, NY: Springer New York, 2019, pp. 353–377.

[30] Cyril Dominguez, Rolf Boelens, and Alexandre M. J. J. Bonvin. "HADDOCK: A Protein - Protein Docking Approach Based on Biochemical or Biophysical Information". In: *Journal of the American Chemical Society* 125.7 (2003), pp. 1731–1737.

[31] *A server for modeling of large macromolecular complexes.* URL: http://genesilico.pl/pyry3d.

[32] Frank DiMaio, Michael D. Tyka, Matthew L. Baker, Wah Chiu, and David Baker. "Refinement of Protein Structures into Low-Resolution Density Maps Using Rosetta". In: *Journal of Molecular Biology* 392.1 (2009), pp. 181–190. ISSN: 0022-2836. DOI: https://doi.org/10.1016/j.jmb.2009.07.008.

[33] Jan Kosinski, Shyamal Mosalaganti, Alexander von Appen, Roman Teimer, Amanda L. DiGuilio, William Wan, Khanh Huy Bui, Wim J.H. Hagen, John A. G. Briggs, Joseph S. Glavy, Ed Hurt, and Martin Beck. "Molecular architecture of the inner ring scaffold of the human nuclear pore complex". In: *Science* 352.6283 (2016), pp. 363–365.

[34] Maria I. Dauden, Marcin Jaciuk, Felix Weis, Ting-Yu Lin, Carolin Kleindienst, Nour El Hana Abbassi, Heena Khatter, Rościsław Krutyhołowa, Karin D. Breunig, Jan Kosinski, Christoph W. Müller, and Sebastian Glatt. "Molecular basis of tRNA recognition by the Elongator complex". In: *Science Advances* 5.7 (2019), eaaw2326. DOI: 10.1126/sciadv.aaw2326.

[35] Christian P. Robert and George Casella. *Monte Carlo Statistical Methods.* USA: Springer, 2004. ISBN: 0387212396.

[36] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. "Equation of State Calculations by Fast Computing Machines". In: *The Journal of Chemical Physics* 21.6 (1953), pp. 1087–1092. DOI: 10.1063/1.1699114. URL: http://link.aip.org/link/?JCP/21/1087/1.

[37] Yehouda Harpaz, Mark Gerstein, and Cyrus Chothia. "Volume changes on protein folding". In: *Structure* 2.7 (1994), pp. 641–649. ISSN: 0969-2126. DOI: https://doi.org/10.1016/S0969-2126(00)00065-4. URL: https://www.sciencedirect.com/science/article/pii/S0969212600000654.

[38] Rappsilber J. Combe CW Fischer L. "xiNET: cross-link network maps with residue resolution." In: *Mol Cell Proteomics.* (2015), pp. 1137–47.

[39] Andrew Waterhouse, Martino Bertoni, Stefan Bienert, Gabriel Studer, Gerardo Tauriello, Rafal Gumienny, Florian T Heer, Tjaart A P de Beer, Christine Rempfer, Lorenza Bordoli, Rosalba Lepore, and Torsten Schwede. "SWISS-MODEL: homology modelling of protein structures and complexes". In: *Nucleic Acids Research* 46.W1 (May 2018), W296–W303. ISSN: 0305-1048. DOI: 10.1093/nar/gky427.

[40] Jones DT Buchan DWA. "The PSIPRED Protein Analysis Workbench: 20 years on." In: *Nucleic Acids Research* (2019).

[41] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596.7873 (Aug. 2021), pp. 583–589. ISSN: 1476-4687. DOI: 10.1038/s41586-021-03819-2.

[42] Jan Kosinski, Alexander von Appen, Alessandro Ori, Kai Karius, Christoph W. Müller, and Martin Beck. "Xlink Analyzer: Software for analysis and visualization of cross-linking data in the context of three-dimensional structures". In: *Journal of Structural Biology* 189.3 (2015), pp. 177–183. ISSN: 1047-8477. DOI: https://doi.org/10.1016/j.jsb.2015.01.014.

[43] Y. Perez-Riverol, A. Csordas, J. Bai, M. Bernal-Llinares, S. Hewapathirana, D.J. Kundu, A. Inuganti, J. Griss, G. Mayer, M. Eisenacher, E. Pérez, J. Uszkoreit, J. Pfeuffer, T. Sachsenberg, S. Yilmaz, S. Tiwary, J. Cox, E. Audain, M. Walzer, A.F. Jarnuczak, T. Ternent, A. Brazma, and J.A. Vizcaíno. "The PRIDE database and related tools and resources in 2019: improving support for quantification data." In: *Nucleic Acids Research* 47 (2019), pp. 442–450.

[44] John. Geweke. "Evaluating the Accuracy of Sampling-Based Approaches to the Calculation of Posterior Moments". In: *Proceedings of the Fourth Valencia International Meeting on Bayesian Statistics* 4 (Nov. 1992), pp. 169–193.

[45] S. Viswanath, I.E. Chemmama, P. Cimermancic, and Sali A. "Assessing Exhaustiveness of Stochastic Sampling for Integrative Modeling of Macromolecular Structures." In: *Biophys J.* 113 (2017), pp. 2344–2353.

[46] Daven Vasishtan and Maya Topf. “Scoring functions for cryoEM density fitting”. In: *Journal of Structural Biology* 174.2 (May 2011), pp. 333–343.

[47] Yehouda Harpaz, Mark Gerstein, and Cyrus Chothia. “Volume changes on protein folding”. In: *Structure* 2.7 (1994), pp. 641–649. ISSN: 0969-2126. DOI: https://doi.org/10.1016/S0969-2126(00)00065-4. URL: http://www.sciencedirect.com/science/article/pii/S0969212600000654.

[48] Werner Kühlbrandt. “The Resolution Revolution”. In: *Science* 343.6178 (2014), pp. 1443–1444. DOI: 10.1126/science.1251652.

[49] Willy Wriggers. “Conventions and workflows for using Situs.” In: *Acta Cryst.* 68 (2012).

[50] Pablo Chacón and Willy Wriggers. “Multi-resolution contour-based fitting of macromolecular structures”. In: *Journal of molecular biology* 317.3 (2002), pp. 375–384.

[51] Stefan Birmanns and Willy Wriggers. “Multi-resolution anchor-point registration of biomolecular assemblies and their components”. In: *Journal of structural biology* 157.1 (2007), pp. 271–280.

[52] Bonvin van Zundert and AMJJ Bonvin. “Fast and sensitive rigid-body fitting into cryo-EM density maps with PowerFit”. In: *AIMS Biophysics* 2 (2015).

[53] Gydo van Zundert and Alexandre Bonvin. *PowerFit software.* Sept. 2016. DOI: 10.5281/zenodo.1037228. URL: https://doi.org/10.5281/zenodo.1037228.

[54] Thai V. Hoang, Xavier Cavin, and David W. Ritchie. “gEMfitter: A highly parallel FFT-based 3D density fitting tool with GPU texture memory acceleration”. In: *Journal of Structural Biology* 184.2 (2013), pp. 348–354. ISSN: 1047-8477. DOI: https://doi.org/10.1016/j.jsb.2013.09.010.

[55] Julio A. Kovacs and Willy Wriggers. “Fast rotational matching”. In: *Acta Crystallographica Section D* 58.8 (Aug. 2002), pp. 1282–1286. DOI: 10.1107/S0907444902009794.

[56] Pascal Audet. “Directional wavelet analysis on the sphere: Application to gravity and topography of the terrestrial planets”. In: *Journal of Geophysical Research: Planets* 116.E1 (2011). DOI: https://doi.org/10.1029/2010JE003710.

[57] Pascal Audet. “Toward mapping the effective elastic thickness of planetary lithospheres from a spherical wavelet analysis of gravity and topography”. In: *Physics of the Earth and Planetary Interiors* 226 (Jan. 2014), pp. 48–82. DOI: 10.1016/j.pepi.2013.09.011.

[58] Sean Swenson and John Wahr. “Methods for inferring regional surface-mass anomalies from Gravity Recovery and Climate Experiment (GRACE) measurements of time-variable gravity”. In: *Journal of Geophysical Research: Solid Earth* 107.B9 (2002), ETG 3-1-ETG 3–13. DOI: https://doi.org/10.1029/2001JB000576.

[59] David S. Tuch. “Q-ball imaging”. In: *Magnetic Resonance in Medicine* 52.6 (2004), pp. 1358–1372. DOI: https://doi.org/10.1002/mrm.20279.

[60] James Cooley and John Tukey. “An Algorithm for the Machine Calculation of Complex Fourier Series”. In: *Mathematics of Computation* 19.90 (1965), pp. 297–301.

[61] Wen Mei W. Hwu. *GPU Computing Gems Emerald Edition.* Elsevier Inc., 2011. DOI: 10.1016/C2010-0-65709-9.

[62] Institute of Electrical and Electronics Engineers. *IEEE Standard for Binary Floating-Point Arithmetic.* IEEE Std 754-1985. 1985.

[63] Gerald R. Kneller. "Superposition of Molecular Structures using Quaternions". In: *Molecular Simulation* 7.1-2 (1991), pp. 113–119.

[64] Protein Data Base Foundations. *PDB file format specifications.* 2021. URL: `https://www.wwpdb.org/documentation/file-format`.

[65] MRC. *MRC file format specifications.* 2021. URL: `https://bio3d.colorado.edu/imod/doc/mrc_format.txt`.

[66] Azriel Rosenfeld and John L. Pfaltz. "Sequential Operations in Digital Picture Processing". In: *J. ACM* 13.4 (Oct. 1966), pp. 471–494. DOI: `10.1145/321356.321357`.

[67] NVIDIA. *Cuda Toolkit documentation.* 2021. URL: `https://docs.nvidia.com/cuda/index.html`.

[68] NVIDIA. *Cuda Developer Blog.* 2021. URL: `https://developer.nvidia.com/blog/cuda-dynamic-parallelism-api-principles/`.

[69] NVIDIA. *Thrust Developer Documentation.* 2021. URL: `https://docs.nvidia.com/cuda/thrust/index.html`.

[70] NVIDIA. *Thrust API description.* 2021. URL: `https://thrust.github.io/doc/index.html`.

[71] Nasa. *Monte Carlo Burn In.* 2021. URL: `https://heasarc.gsfc.nasa.gov/xanadu/xspec/manual/node43.html`.

[72] pymc. *PyMc package.* 2021. URL: `https://pymc-devs.github.io/pymc/modelchecking.html`.

[73] Gilbert Strang. *Introduction to Linear Algebra.* Fourth. Wellesley-Cambridge Press, 2009.

[74] Ian R. Cole. "Modelling CPV". PhD thesis. Loughborough University, June 2015.

[75] Charles F.F. Karney. "Quaternions in molecular modeling". In: *Journal of Molecular Graphics and Modelling* 25.5 (2007), pp. 595–604.

[76] J.C. Mason and D.C. Handscomb. *Chebyshev Polynomials.* CRC Press, 2002.

[77] Ian H. Sloan and William E. Smith. "Product integration with the Clenshaw-Curtis points: Implementation and error estimates". In: *Numerische Mathematik* 34.4 (Dec. 1980), pp. 387–401. DOI: `10.1007/BF01403676`.

[78] Manuel Gräf and Daniel Potts. "Sampling Sets and Quadrature Formulae on the Rotation Group". In: *Numerical Functional Analysis and Optimization* 30.7-8 (2009), pp. 665–688. DOI: `10.1080/01630560903163508`.

[79] Jason McEwen, Martin Büttner, Boris Leistedt, Hiranya Peiris, and Yves Wiaux. "A Novel Sampling Theorem on the Rotation Group". In: *IEEE Signal Processing Letters* 22 (Aug. 2015). DOI: `10.1109/LSP.2015.2490676`.

[80] Jason D. McEwen and Yves Wiaux. "A Novel Sampling Theorem on the Sphere". In: *IEEE Transactions on Signal Processing* 59.12 (2011), pp. 5876–5887. DOI: `10.1109/TSP.2011.2166394`.

[81] Claude E. Shannon. "A mathematical theory of communication." In: *Bell Syst. Tech. J.* 27.3 (1948), pp. 379–423.

[82] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing).* USA: Wiley-Interscience, 2006. ISBN: 0471241954.

[83] Matthias K. Vorländer, Anna Jungblut, Kai Karius, Florence Baudin, Helga Grötsch, Jan Kosinski, and Christoph W. Müller. "Structure of the TFIIIC subcomplex τA provides insights into RNA polymerase III pre-initiation complex formation". In: *Nature Communications* 11.1 (2020), p. 4905. DOI: `10.1038/s41467-020-18707-y`.

[84] Irene Farabella, Daven Vasishtan, Agnel Praveen Joseph, Arun Prasad Pandurangan, Harpal Sahota, and Maya Topf. "*TEMPy*: a Python library for assessment of three-dimensional electron microscopy density fits". In: *Journal of Applied Crystallography* 48.4 (Aug. 2015), pp. 1314–1323. DOI: `10.1107/S1600576715010092`.

[85] Matteo Allegretti, Christian E. Zimmerli, Vasileios Rantos, Florian Wilfling, Paolo Ronchi, Herman K. H. Fung, Chia-Wei Lee, Wim Hagen, Beata Turoňová, Kai Karius, Mandy Börmel, Xiaojie Zhang, Christoph W. Müller, Yannick Schwab, Julia Mahamid, Boris Pfander, Jan Kosinski, and Martin Beck. "In-cell architecture of the nuclear pore and snapshots of its turnover". In: *Nature* 586.7831 (2020), pp. 796–800. DOI: `10.1038/s41586-020-2670-5`.

[86] Manuel Gräf. "Efficient Algorithms for the Computation of Optimal Quadrature Points on Riemannian Manifolds". PhD thesis. Universitätsverlag der Technischen Universität Chemnitz, Chemnitz: Technischen Universität Chemnitz, Feb. 2013.

[87] Ali Punjani, John L. Rubinstein, David J. Fleet, and Marcus A. Brubaker. "cryoSPARC: algorithms for rapid unsupervised cryo-EM structure determination". In: *Nature Methods* 14.3 (Mar. 2017), pp. 290–296. DOI: `10.1038/nmeth.4169`.

[88] D. M. Henderson. "Shuttle Program. Euler angles, quaternions, and transformation matrices working relationships". In: *NASA Technical Reports Server* (July 1977).

[89] Gerald R. Kneller. "Superposition of Molecular Structures using Quaternions". In: *Molecular Simulation* 7.1-2 (1991), pp. 113–119.

[90] Ondřej Šťáva and Bedřich Beneš. "Chapter 35 - Connected Component Labeling in CUDA". In: *GPU Computing Gems Emerald Edition.* Ed. by Wen-mei W. Hwu. Applications of GPU Computing Series. Boston: Morgan Kaufmann, 2011, pp. 569–581. ISBN: 978-0-12-384988-5. DOI: `https://doi.org/10.1016/B978-0-12-384988-5.00035-8`.

[91] C.L. Lawson, A. Patwardhan, H.M. Berman, G.J. Kleywegt, and W. Chiu. "EM-DataBank unified data resource for 3DEM." In: *Nucleic Acids Research* 44 (2016).

[92] Wlodawer, Alexander, Minor, Wladek, Dauter, Zbigniew, Jaskolski, and Mariusz. "Protein crystallography for non-crystallographers, or how to get the best (but not more) from published macromolecular structures". In: *FEBS Journal* 275.1 (Jan. 2008), pp. 1–21. DOI: `10.1111/j.1742-4658.2007.06178.x`. URL: `http://dx.doi.org/10.1111/j.1742-4658.2007.06178.x`.

[93] Daniel Russel, Keren Lasker, Ben Webb, Javier Velázquez-Muriel, Elina Tjioe, Dina Schneidman, Bret Peterson, and Andrej Sali. "Putting the Pieces Together: Integrative Modeling Platform Software for Structure Determination of Macromolecular Assemblies". In: *PLoS biology* 10 (Jan. 2012), e1001244. DOI: `10.1371/journal.pbio.1001244`.

[94] Maria I. Dauden, Marcin Jaciuk, Felix Weis, Ting-Yu Lin, Carolin Kleindienst, Nour El Hana Abbassi, Heena Khatter, Rościsław Krutyhołowa, Karin D. Breunig, Jan Kosinski, Christoph W. Müller, and Sebastian Glatt. "Molecular basis of tRNA recognition by the Elongator complex". In: *Science Advances* 5.7 (2019).

Chapter 7

Appendix

7.1 Zusammenfassung

In dieser Dissertation wird die Rolle des systematischen Fittings im Kontext des integrativen strukturellen Modelings betrachtet. Diese Art des Modelings dient dazu, Modelle von makromolekularen Komplexen aus verschiedenen Datenquellen zu generieren und ist zum Beispiel in Form der Softwarepipeline *Assembline* implementiert. Die Datentypen, welche integriert werden, umfassen neben Eektronendichtekarten, welche aus der Kryoelektronenmikroskopie und der Kryoelektronentomographie gewonnen wurden, auch Kristallstrukturen von Komplexen, Proteinen, RNA-Strukturen und Protein-Untereinheiten sowie Crosslinks und biochemische Informationen. Das systematische Fitting ist der erste Schritt in der *Assembline* Softwarepipeline und stellt ein Ensemble möglicher Positionen und Orientierungen der Strukturen, aus welche sich der makromolekulare Komplex zusammensetz, zur Verfügung. Aus diesen möglichen Positionen und Orientierungen wird in der weiteren Pipeline eine Anzahl von wahrscheinlichen Modellen generiert. Das systematische Fitting kann, im Rahmen der *Assembline* pipeline, als einer der informationsreichsten Quellen angesehen werden. Es gibt eine Anzahl von Möglichkeiten, wie man das systematische Fitting durchführen kann. Von Fall zu Fall können die Ergebnisse sehr unterschiedlich ausfallen. Diese Arbeit untersucht die Frage, wie ein Maximum an Information aus den gegebenen Daten gewonnen werden kann und welche computerwissenschaftlichen, numerischen und mathematischen Konzepte genutzt werden können, um das Modellieren von biomolekularen Komplexen zu optimieren.
Verschiedene Modellingprojekte wurden in verschienden Kapazitäten begleitet. Die Strukturbestimmung des TFIIIC-Komplexes (Transkriptionsfaktoruntereinheit C des Präinitiationskomplexes der Polymerase III in Eukaryoten) wurde sowohl durch systematisches Fitting als auch strukturelles integratives Modelling unterstützt. Obwohl die Gesamt-

struktur des TFIIIC nicht bestimmt werden konnte, konnte die Struktur des Subkomplexes τ_A ermittelt werden. Für die zum Subcomplex τ_B gehörigen Domäne Brf1-TBP und τ_A konnten schliesslich wahrscheinliche Positionen und Orientierungen in einer negative stain Elektronendichtekarte von TFIIIC bestimmt werden ([83]).
In einem weiterem Modellingprojekt ist es gelungen einen Subkomplex, den sogenannte P-Komplex, in einem in-situ Tomogram einer Speiche der Kernpore in *s. cerevisia* wahrscheinliche Positionen und Orientierungen durch systematisches Fitting zuzuordnen. Hier wurde programmatische Werkzeuge zur einfacheren Auswertung der statistischen Analyse des systematischen Fittings implementiert und eine Analyse und Rechtfertigung der verschiedenen Scoring-Optionen im systematischen Fitting zur Verfügung gestellt ([85]).
Die *Assembline* Software pipeline wurde erstmals der Öffentlichkeit zugänglich gemacht, zusammen mit ausführlichen Beispielen und Anleitungen, um die Benutzung zu vereinfachen. Hier wurden ein neues Konvergenzkriterium implementiert, welches unnötige lange Monte-Carlo-Läufe verhindert([26]).
Desweiteren wurden die grundlegenden Datentypen und einige der Algorithmen, welche im systematischen Fitting essentiell sind, auf modernen GPU(Graphical Processing Unit)-Systemen implementiert und getestet. Hierbei wurde darauf geachtet, die spezifischen Eigenheiten der GPU-Architektur zu gut wie möglich auszunutzen und die Nutzung mehrer GPUs in großen Clustern zu ermöglichen.
Methoden der numerischen Mathematik wurden genutzt, um dem systematischen Fitting verfeinerbare, 6-dimensionale Suchgitter zur Verfügung zu stellen. Diese ermöglichen zudem das berechnen 6-dimensionaler Statistiken, welche das gewählte Fittingprotokoll als Gesamtes charakterisiert. Desweiteren wurde die partial-surface-score als Scoring Methode eingeführt, welche insbesondere unvollständige Oberflächensegmente fitten kann.
Ein erschöpfender adaptiver paralleler Suchalgorithmus für 6-dimensionale Räume wurde beschrieben, implementiert und getestet im Fall von TFIIIC. Das erwartete lokale Minimum wurde erfolgreich gefunden in einer Zeit von ca 1.8 s. Methoden der Informationstheorie wurden benutzt, um das Ergebnis eines systematischen Fittingprotokolls (im obigen Sinne) auf seinen Informationsgehalt zu überprüfen. Somit wurde ein Kriterium erschaffen, mit dem sich informationsreichere von informationsärmeren Fittingprotokollen unterscheiden lassen. Dies wurde an zwei konkreten Probleme getestet: Der Unterscheidung des Informationsgehaltes zweier verschiedener Scoringmethoden und der Optimierung des Informationsgehaltes einer Scoringmethode mit Hinsicht auf einen Parameter.

7.2 Synopsis

In this dissertation the role of systematic fitting is analyzed in the context of integrative structural modeling. This kind of modeling aims to generate structural models of macromolecular assemblies with the help of varying data sources. The *Assembline* software pipeline is one possible implementation of this procedure. The data types that are integrated are electron density maps obtained through electron microscopy, cryogenic electron microscopy and cryogenic electron tomography and atomic structures that are obtained via crystallography and homology modeling. Additionally chemical crosslinks and biochemical information can be used. Systematical fitting is one of the first steps in the *Assembline* software pipeline and provides an ensemble of possible positions and orientations of the structures that make up the macromolecular complex that is being modeled. From these possible positions and orientations an ensemble of possible models is generated further down the pipeline. Systematic fitting can be seen, in the context of the *Assembline* pipeline, as one of the most information-rich data sources. There is a number of possibilities to conduct the systematic fitting step, often associated with multiple parameters. From case to case the results can vary significantly. This thesis addresses the question, how a maximum of information can be obtained from the data and which computational, numerical and theoretical resources can be used to optimize the modeling of bio-molecular complexes.
Different modeling projects have been supported in different roles. The determination of the structure of the TFIIIC-complex (transcription factor subcomplex C of the preinitiation complex associated with the polymerase III complex in eukaryotes) was accompanied both through systematic fitting and structural integrative modeling. Although the holocomplex of TFIIIC could not be determined, the structure of the subcomplex τ_A was determined using a 3.47 Å cryo-EM density map. The domain Brf1-TBP, which belongs to the τ_B-subcomplex of TFIIIC, and the τ_A subcomplex could be tentatively positioned and oriented within a negative stain map of the holocomplex TFIIIC ([83]).
A further modeling project was concerned with a novel in situ cryo-ET map of the nuclear pore complex of *s. cerevisiae*. It was possible to assign a position and orientation to a negative stain map of the so called P-complex, which has been linked to the terminal steps of mRNA export. Here I provided computational tools to simplify the statistical evaluation of the systematic steps and the preferential use of certain scoring options over others was justified based on the definition of the scores and the nature of the data (see [85]).
The *Assembline* software pipeline was published together with exhaustive tutorials and case studies to grant easier access to new users. Here I implemented a new convergence criterion to prevent ineffectivley long Monte Carlo simulation runs (see [26]).
Furthermore I implemented and tested the fundamental data types and some of the algo-

rithms that are essential to systematic fitting. Care was taken to use the specific hardware features typical to GPUs as efficiently as possible and to enable the use of multiple GPUs for the same task.
I used methods from numerical mathematics to create refinable 6-dimensional search grids for systematic fitting. These grids enable the calculation of statistics of functions in a 6-dimensional space, such as the result of systematic fitting protocols. Furthermore I introduced a new type of score, the partial surface score. This score is constructed to enable the fitting of partial surface segment, e.g. for fitting in negative stain maps.
I introduced an exhaustive adaptive parallel search algorithm for 6-dimensional spaces, and implemented it and tested it on the concrete case of TFIIIC. The expected local maximum was successfully found in a time of ca. 1.8 s.
I used methods from information theory have to assess the information content of a systematic fitting protocol in the above sense. Thus I created a criterion that enables the distinction between information rich and information poor 6-dimensional systematic fitting score distributions. I tested this on two concrete problems: The comparison of the information content of two different scoring methods and the optimization of parameter range for a specific systematic fitting protocol with respect to the information content of the fitting distribution.

7.3 List of publications involving the author

Vasileios Rantos, Kai Karius, and Jan Kosinski. "Integrative structural modeling of macromolecular complexes using Assembline". In: *Nature Protocols* 17.1 (2022), pp. 152–176. DOI: 10.1038/s41596-021-00640-z

Matteo Allegretti, Christian E. Zimmerli, Vasileios Rantos, Florian Wilfling, Paolo Ronchi, Herman K. H. Fung, Chia-Wei Lee, Wim Hagen, Beata Turoňová, Kai Karius, Mandy Börmel, Xiaojie Zhang, Christoph W. Müller, Yannick Schwab, Julia Mahamid, Boris Pfander, Jan Kosinski, and Martin Beck. "In-cell architecture of the nuclear pore and snapshots of its turnover". In: *Nature* 586.7831 (2020), pp. 796–800. DOI: 10.1038/s41586-020-2670-5

Matthias K. Vorländer, Anna Jungblut, Kai Karius, Florence Baudin, Helga Grötsch, Jan Kosinski, and Christoph W. Müller. "Structure of the TFIIIC subcomplex τA provides insights into RNA polymerase III pre-initiation complex formation". In: *Nature Communications* 11.1 (2020), p. 4905. DOI: 10.1038/s41467-020-18707-y

Jan Kosinski, Alexander von Appen, Alessandro Ori, Kai Karius, Christoph W. Müller, and Martin Beck. "Xlink Analyzer: Software for analysis and visualization of cross-linking data in the context of three-dimensional structures". In: *Journal of Structural Biology* 189.3 (2015), pp. 177–183. ISSN: 1047-8477. DOI: `https://doi.org/10.1016/j.jsb.2015.01.014`

Erklärung der Urheberschaft

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Göttingen, 24.03.22	Kai Karius
Ort, Datum	Unterschrift