MONASH University

The Bonding Charge Density in Copper by Parallelised Multislice Differential Quantitative Convergent-Beam Electron Diffraction, Using a New and Improved Independent-Atom Model

Tianyu Liu

A thesis submitted for the degree of

Doctor of Philosophy

Department of Materials Science and Engineering

Monash University

Victoria, Australia

December 2018

Copyright notice

Declarations

I, Tianyu Liu, hereby certify that this thesis is my own work and contains no material which has been accepted previously for the award of any other degree or diploma in any university or any institution and, to the best my knowledge, this thesis does not contain any material previously published or written by another person, except where due reference is made in the text of the thesis.

Tianyu Liu

10/03/2019

Abstract

Conventionally, bonds in different solids are classified into three types: ionic, covalent and metallic. However, this fundamental concept has been challenged by suggestions that metallic bonding should be classified as a sub-set of covalent bonding [1]. Yet detailed knowledge of metallic bonding like interatomic bonding positions, shapes and electron density distributions remains limited. Little has been learned since Drude [2] proposed the model of cations in a "sea of delocalised electrons" in 1900. To date, there have been few experimental studies of bonds in metals because of the limited resolution of the available techniques [3].

Recent advances in quantitative convergent beam electron diffraction (QCBED) have made it possible to probe and measure chemical bonds in defect-free crystals with unprecedented resolution [4]. This has allowed more precise determination of bonding in materials [4-8] than previous X-ray, γ-ray and electron diffraction experiments. Inspired by these successful studies, this work focuses on further advancing the development of QCBED techniques in application to a more accurate measurement of the bonding in copper. Beside the measurement of bonding in copper, a highlight of this work is a two order of magnitude acceleration in the QCBED refinement program, which implements parallelisation of both CPU and GPU devices. With this significant acceleration, in-situ QCBED analysis during TEM experiments becomes possible as a future direction for the technique. The uniqueness of the present study also lies in the application of the multislice formalism for electron scattering in all of the present QCBED analyses, coupled with a differential approach for removing the influence of thermal diffuse scattering and a new method for incorporating absorption necessitated by this new methodology. In addition, an improved independent-atom model (IAM) calculated by density functional theory (DFT) is also determined, which has included the electron correlation effects neglected by the conventional relativistic Hartree-Fock (RHF) calculations published 50 years ago by Doyle and Turner. The new IAM calculated by DFT has shown about 1% difference comparing to the RHF IAM, which has a comparable magnitude to the bonding being measured in copper. The outcomes of this work reveal a mixed tetrahedral and octahedral bonding morphology that is the origin of the highly anisotropic elastic response of copper and gives approximately the correct magnitude of this response, unlike most previous studies.

Acknowledgements

I would like to thank my supervisors, A/Prof Philip N.H. Nakashima and A/Prof Laure Bourgeois, for their expertise and especially guidance and enlightenment from the very beginning of my research. I appreciate Dr Andrew E. Smith for the DFT calculations and his comprehensive help to this thesis. I am grateful to my colleagues, Dr Yueming Guo, Dr Zezhong Zhang and Mr Ding Peng, for their inspiring discussions and assistance related to my project.

I acknowledge the financial support of the Faculty of Engineering International Postgraduate Research Scholarship (FEIPRS) Monash University, the living allowance provided by my supervisor A/Prof Philip N. H. Nakashima, and the Monash Centre for Electron Microscopy (MCEM) PhD Scholarship. I would like to thank Mr. Yu-Tsun for the energy-filtered and scanning CBED data collection, Prof. Jian-Min Zuo for support on scanning CBED and sharing of QCBED software codes, and Frederick Seitz Materials Research Laboratory for the energy-filtered QCCBED experiments. I acknowledge the use of facilities within the MCEM for CBED pattern collection, and I am grateful for the expertise and support from MCEM staff including but not limited to Dr Russell King, Mr Renji Pan, Dr Timothy Williams, Dr Emily Chen, Dr Xi-Ya Fang and Katherine O'Rourke etc. Data processing in this research was supported by the Monash eResearch Centre and eSolutions-Research Support Services through the use of the Monash Campus HPC Cluster. All charge density images were generated with VESTA [10].

Finally, I would like to thank my parents, Dr Xifan Wang and Dr Dongnian Liu, who have always supported me in various aspects, including those that I have not yet realised by now.

Contents

Chapter 1. Introduction

In recent decades, great advancements have been made to determine the bonding in metals [3, 4, 6-8, 11-18], with the help of improved experimental techniques and computational capabilities. The knowledge of metallic bonding thus has become more definite, compared to the “sea of electrons” model proposed by Drude in 1900 [2]. However, the metallic bonds in some common metals in everyday lives, such as copper, are still not fully characterised, as reflected by a certain level of diversity is still presented in the latest reported measurements [3, 4, 7, 11, 15]. In this chapter, a brief introduction regarding common knowledge of metallic bond and copper, the definition and characterisation techniques of bonding, and the aims and outline of this thesis will be given.

1.1. Chemical bonding: the metallic bond

There are three primary chemical bond types used to describe electrostatic interactions between atoms: ionic, covalent and metallic. The three bond types can be distinguished by electronegativity, which describes the ability of atoms to attract electrons [19]. For example, a van Arkel-Ketelaar triangle [20, 21], which categorises different bond types based on the average electronegativity and electronegativity difference, is shown in Figure 1.1. Ionic compounds are placed in the top corner of the van Arkel-Ketelaar triangle, consisting of atoms with large electronegativity difference. In ionic bonds, valence electrons transfer from low to high electronegativity atoms, forming positive and negative ions respectively. The oppositely charged ions will be bonded by strong electrostatic forces. Covalent materials are in the bottom-right corner of the van Arkel-Ketelaar triangle. In contrast to ionic bonds, covalent bonds involve atoms with low electronegativity difference and relatively high electronegativities. Because these atoms have strong attractions to electrons, valence electrons in a covalent bond will be shared by different atoms without delocalisation. For metals and alloys, which are placed in the bottom-left corner of the van Arkel-Ketelaar triangle, the atoms have low average electronegativities and low electronegativity difference. The valence electrons tend to delocalise from the metallic atoms due to the low electronegativities, forming a "sea of electrons" which bonds the positive metallic ions together. Although this ternary categorisation of chemical bonds has been a conventional concept [22], the significance of the metallic bond has often been challenged. Some researchers propose that the metallic bond should be classified as a subset of the covalent bond because of their similarities compared to the ionic bond [1, 23], while others insist covalent and metallic bonds exhibit distinguishing properties [24, 25]. Apart from properties such as electronegativity and band gap discussed in the controversy [1, 23-25], however, detailed knowledge of the metallic bond, like bonding electron distribution, has remained limited [20, 22, 24, 26]. As will be described in Section 1.3, this is mainly due to the difficulties of bonding experiments, especially for metallic materials.

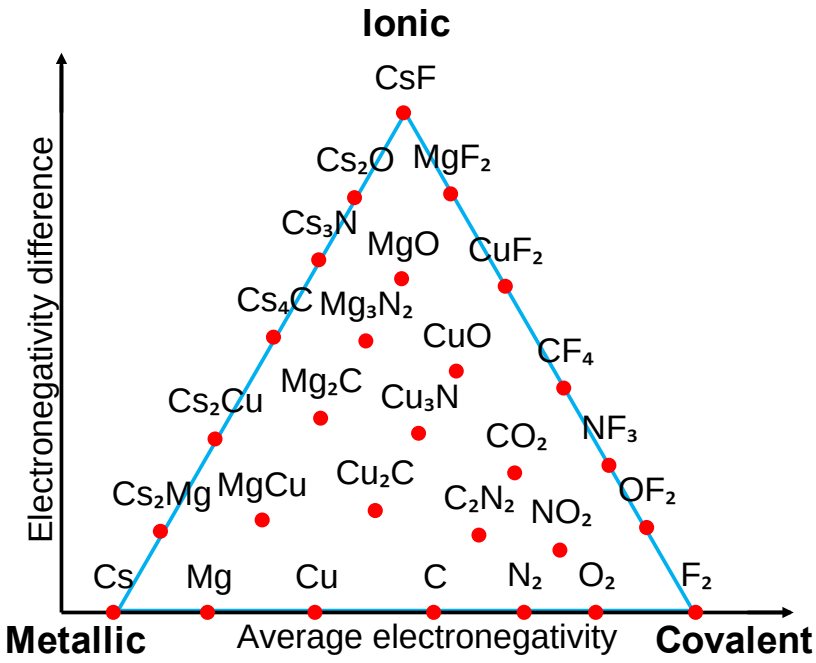


Figure 1.1. A van Arkel-Ketelaar triangle (fashioned after Ref. [19]). Materials with both low average electronegativities and low electronegativity differences are placed in the bottom-left corner of the triangle and are metallic. Materials with high average electronegativities but low electronegativity differences are placed in the bottom-right corner and are covalent. The top corner of the triangle contains materials with high electronegativity differences and are ionic.

The "sea of electrons" model, also known as the classic free-electron model, was proposed 118 years ago by Drude [2] in order to explain the transportation of electrons in materials and has become the most commonly used description for bonding electron distributions in metals [27]. In the Drude model, the delocalised electrons are assumed to form an ideal electron gas, which interacts with the metallic ions by collisions only [28]. Any long-range interaction between the delocalised electrons and/or the metallic ions is neglected [28, 29]. The high thermal and electrical conductivities, high malleability, high ductility and lustre of metals and alloys can be well explained by this simplified model [28, 29]. In contrast, the actual bonding electron distributions in metals have been experimentally observed to be anisotropic [3, 4, 6-8, 12-18, 30-49], which differ from the isotropic ideal electron gas assumed by the Drude model. For instance, the detailed bonding electron distribution in aluminium (a face-centred cubic (FCC) crystal) has been accurately measured by Nakashima *et al.* [6] using quantitative convergent-beam electron diffraction. The experimentally determined bonding distribution of aluminium exhibits anisotropic bonding localisation at the tetrahedral sites (the bonding in FCC crystal, like aluminium and copper, is often described as tetrahedral, bridging, octahedral types. As illustrated in Figure 1.2. or the combinations thereof) which is contrary to many previous studies. This result also shows good correlation to the mechanical properties of aluminium (correlations between

electronic structures and mechanical properties are expected for metals [30]), as well as good agreements to theoretical calculations using first-principal method [6]. However, such successful characterisation of electron distribution in metallic bonding is still missing for many other metals [3, 7, 8, 43], such as copper. This is mainly due to the fact that the localisation of bonding electrons in metals is very weak, which typically peaks at 3% of the overall electron intensity and often requires error finer than 0.1% to draw reliable conclusions. Thus, the bonding in metals is infinitesimal for most experimental techniques [4, 32, 50, 51], compared to the cases in ionic and covalent bonds, where the peak localisation of bonding electron can be 30% of the overall electron intensity.

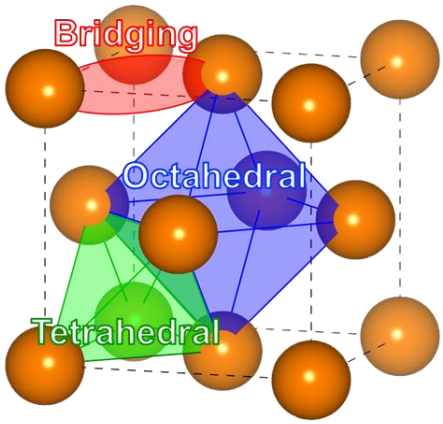


Figure 1.2. Illustration of common bonding sites in an FCC unit cell. The strongest bonding in an FCC metal often locates at any of the three sites 1) the octahedral site which is at the centre, 2) the tetrahedral sites at quarters of the unit cell, and 3) the bridging sites between nearest-neighbour atoms.

1.2. The metallic bond in copper

Copper is a non-ferrous transition metal, with an FCC crystal structure. Unlike other groups of transition metals, copper has a filled 3*d* shell and a partially filled 4*s* shell. Furthermore, the energies of its 3*d* and 4*s* shells are close enough for transitions between $3d^{10}4s^{1}$ and $3d^{9}4s^{2}$ to occur and absorb "blueish" photons in visible lights, giving rise to copper's unique colour [52]. Similar rules also apply to gold. Silver, gold and copper are in the same group in the periodic table and have similar electronic configurations, but because the distance between the 4*d* and 5*s* shells of silver is too far, silver has a similar appearance to other metals [52]. Compared to silver and gold, because copper has a relatively small atomic number, experimental measurements (weaker extinction in X-ray diffraction) and quantum chemical calculations of its electron distributions (weaker relativistic effects) are expected to be less difficult [53]. Compared to metals like tungsten, molybdenum and aluminium, copper is relatively mechanically anisotropic [54]. With knowledge of the correlations between the mechanical

properties and bonding electron distributions in metal [30], the metallic bond in copper may have relatively strong localised electron distributions [6], which departs significantly from the ideal electron gas. Such interesting properties of copper lie in its unique electronic structure, motivating many investigations [3, 4, 11-16, 18, 31, 32, 39, 41-49, 55, 56] since the first attempt using X-ray in 1930. Agreements on bonding distribution of copper are found across some studies, but no common conclusions can be drawn. Some studies agree copper has tetrahedral bonds [4, 14, 18]; some others suggest the bonds in copper are combinations of octahedral and bridging types [7, 12]; while others propose that bonding in copper is mainly localised between the nearest-neighbour atoms [7, 11, 16].

1.3. Definition and characterisation of bonding

For bonding electron distribution studies, the concept "deformation density" is often used. It is defined as the difference in the electron density distribution caused by the presence of bonding, when comparing an actual crystal to a theoretical unbonded system:

Eq. 1.1
$$\Delta\rho = \rho_{actual} - \rho_{IAM},$$

where $\Delta\rho$ is the deformation density of electron distribution, ρ_{actual} is the electron distribution of a crystal with bonding and ρ_{IAM} is the electron distribution of an unbonded system, which is also called the independent-atom model (IAM). The actual electron distribution of a crystal can be determined by theoretical calculations or experimental measurements, while the IAM electron distribution can only be calculated theoretically [6]. The electron distribution of a crystal, $\rho(\mathbf{r})$, can be expressed by its Fourier coefficients, $F_{\mathbf{g}}$, which are also known as the X-ray structure factors:

Eq. 1.2
$$\rho(\mathbf{r}) = \sum_{\mathbf{g}} F_{\mathbf{g}} \cdot \exp(2\pi i \mathbf{g} \cdot \mathbf{r}),$$

where **r** is the real space vector in a unit cell and **g** is a reciprocal lattice vector. It is known that the low-order structure factors are the most significant to the deformation density [6, 7, 57]. This is because the redistribution of electrons caused by bonding occurs mostly among the valence electrons, which are Fourier transformed to the low-order structure factors (i.e. F_{111}, F_{200} and F_{220} in FFC structures). Moreover, due to the weak localisation in metallic bonding, the differences ($\Delta F_{\mathbf{g}}$) between the structure factors of a real-world crystal ($F_{\mathbf{g}}^{Real}$) and of the IAM ($F_{\mathbf{g}}^{IAM}$) are very small in metals. For FCC metals, typically, ΔF_{111} and ΔF_{200} (the first two lowest-order structure factors) are only 3-4% of F_{111}^{Real} and

F_{200}^{Real} respectively, and ΔF_{220} (the third lowest-order structure factor) is smaller than 0.5% of F_{220}^{Real} [4, 32, 50]. For other structure factors after F_{220}, $\Delta F_{\mathbf{g}}$ are very subtle ($\Delta F_{\mathbf{g}}/F_{\mathbf{g}}^{Real} < 0.1\%$), which are even smaller than the experimental errors of any available technique at this higher-order region. Hence, the reliability of experimental determination of bonding for FCC metals generally ends by the third or fourth lowest-order structure factors, where the electron redistribution becomes insignificant compared to the measurement errors. However, a more comprehensive inclusion of structure factors should lead to more accurate bonding conclusions, if the experimental errors allow.

To calculate the X-ray structure factors of the IAM electron distribution, knowledge of the X-ray atomic scattering factors is required:

Eq. 1.3
$$F_{\mathbf{g}} = \sum_j f_{X,j}(s) \cdot \exp\left(-B_j s^2\right) \cdot \exp\left(-2\pi i \mathbf{g} \cdot \mathbf{r}_j\right),$$

where $f_{X,j}(s)$ is the X-ray atomic scattering factor of atom j as a function of scattering angle θ and electron wavelength λ, and $s = \sin\theta/\lambda$. B_j is the Debye-Waller factor of atom j, which describes its thermal motion. Conventionally, the X-ray atomic scattering factors for the IAM are calculated using relativistic Hartree-Fock (RHF) atomic potentials [58]. A collection of IAM atomic scattering factors of several elements calculated using RHF was published by Doyle & Turner in 1968 [9]. These atomic scattering factors are tabulated in the range of $0 \leq s \leq 6$ Å^{-1}, which covers the scattering angles necessary to bonding studies [59]. Atomic scattering factors required by the IAM electron distribution calculations can be retrieved by interpolation of the tabulated values [9, 58-64]. The parameterised interpolation method by Doyle & Turner [9] published with the tabulated atomic structure factors is the most cited in bonding-related studies [3, 4, 7, 12-15, 17, 18, 31-39]. However, the high-order atomic scattering factors calculated by this interpolation method have been suggested to be inaccurate due to the poor fitting at larger s [59, 65]. As mentioned above, due to the bonding deformations in metals being very small and sensitive, the accuracy of metallic bond studies using the Doyle & Turner interpolation for IAM may be affected.

For theoretical calculations of the actual electron distributions in crystals, the band theory and density functional theory (DFT) are popular methods [3, 4, 7, 8, 13, 16, 31, 66]. The band theory solves the electron distribution by self-consistent calculations of the electronic band structure, with local exchange-correlation potentials, which assumes the electrons have no dynamical interactions [67].

Similarly, DFT simulations rely on correct density functionals to estimate the energy exchange among electrons; however the functionals used for transition metals are semi-empirical [68, 69].

There are several experimental methods to measure the actual electron distribution in crystals via structure factor measurements: conventional X-ray diffraction (XRD), γ-ray diffraction, critical voltage, X-ray Pendellösung and quantitative convergent-beam electron diffraction (QCBED).

XRD and γ-ray diffraction can measure accurate high-order structure factors, while the low-order structure factor measurements of periodic crystals tend to suffer from "extinction" of the low-order reflections [32, 50, 51]. This is because both methods are based on the kinematic theory of diffraction, which models the incident X-rays or γ-rays being scattered by the specimen no more than once. However, effects of the multiple scattering become more significant in the low-order reflections of periodic crystals. For heavy elements (i.e. higher atomic numbers), the extinction effects are more intense due to the stronger scattering of X-rays and γ-rays [15, 53].

Convergent-beam electron diffraction (CBED) experiments in high-voltage electron microscopy make use of the fact that intensities in reflections higher than the first-order will drop significantly due to destructive interference under particular acceleration voltages, which are called the "critical voltages" [70, 71]. The critical voltages are very sensitive to the ratio between the low-order structure factors [12]. However, the absolute values of the structure factors cannot be determined by this method directly [4, 32, 51].

The X-ray Pendellösung method, based on the dynamical theory of diffraction which also accounts for multiple scattering of the incident X-rays, can determine low-order structure factors accurately [15, 32, 51]. The "Pendellösung" refers to the thickness fringe oscillations caused by dynamic diffraction [72], which are very sensitive to low-order structure factors [73]. Large edge-shaped single crystals of high qualities are required to produce analysable Pendellösung oscillations [72].

Like the critical voltage method, QCBED is also a high-voltage electron microscopy technique based on CBED, which can directly yield accurate absolute measurements of low-order structure factors [6, 7, 32, 50, 51, 66, 74]. Compared to the X-ray Pendellösung method, QCBED has relatively low specimen requirements (i.e. does not require large perfect single-grain specimens) [32]. Because the low-order structure factors are the most significant for bonding measurements [6, 7, 57], QCBED may

be the most accurate and convenient experimental method. A more detailed description of CBED and QCBED is given in the following section.

1.4. A brief introduction to quantitative convergent-beam electron diffraction

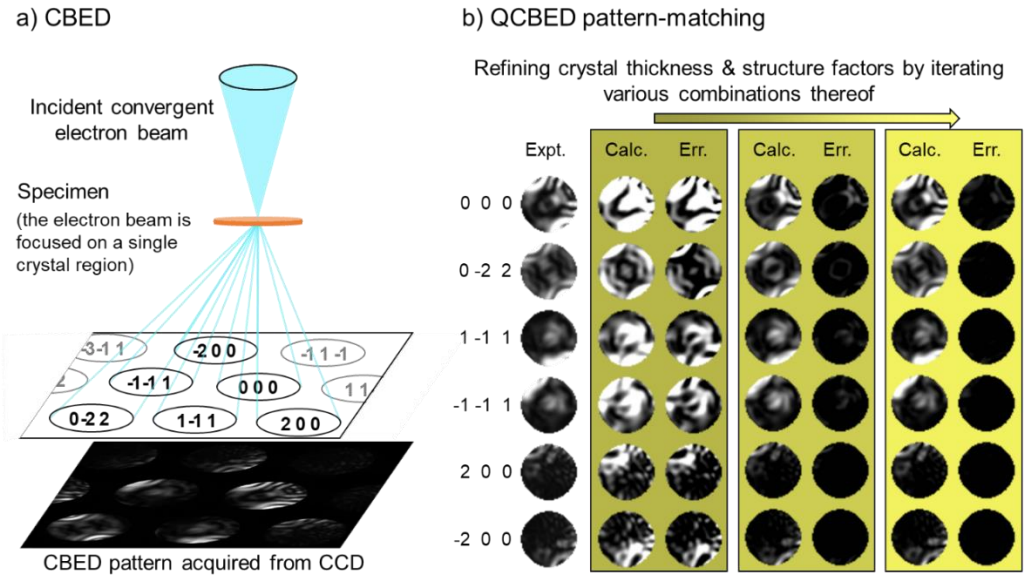


Figure 1.3. A schematic diagram of a) CBED: by focusing the electron beam on a perfect and single crystal region of the specimen, a CBED pattern of reflection discs is obtained (an experimental CBED pattern of copper is used as an example); and b) QCBED pattern-matching of the reflection discs from a): with iterative CBED pattern simulations of various crystal parameters (e.g. thickness and structure factors), the residual (denoted as "Err.") of the experimental (denoted as "Expt.") and simulated (denoted as "Calc.") reflection discs is minimised, producing refined crystal parameters.

An illustration of the CBED technique is shown in Figure 1.3a. With a convergent electron beam focused on the specimen, reflection discs are formed in the CBED pattern. The intensity distributions within CBED reflection discs are highly sensitive to the crystal thickness and electronic structure. Compared to X-rays and γ-rays, the negatively charged incident electrons have strong interactions with the electrons and ions in the specimen [50], resulting in much stronger multiple scattering in low-order reflections. Thus, dynamic diffraction analyses of the intensity distributions within reflection discs in CBED patterns can yield highly accurate low-order structure factors [7, 51]. The probe size of CBED on the specimen is typically 2-20 nm in diameter [32, 50]. With proper alignments, CBED patterns with symmetric reflection discs can be obtained from perfect single crystals when particular Bragg conditions are satisfied [75]. Therefore, with the small probe size and symmetry in CBED patterns, it is possible to select a defect-free single crystal region from a multi-grain specimen for structure factor measurements.

QCBED analysis is accomplished by pattern-matching of CBED patterns, as illustrated in Figure 1.3b. This involves iteratively comparing the intensity distributions within the reflection discs in CBED patterns simulated with different crystal parameters (e.g. thickness and structure factors) to the experimental intensity distributions. Parameters that yield the best matching patterns to the targets are considered as the refined result [6], which have been shown to be theoretically close to the actual values [50]. Because bonding deformation has little effects on the high-order structure factors [4, 6, 32, 57], QCBED only refines the bonding-sensitive low-order structure factors and assumes the actual high-order structure factors to be identical to the IAM structure factors. If inaccurate IAM atomic scattering factors are used for QCBED refinements, the incorrect high-order structure factors will contribute perturbations to the intensity distributions in the calculated CBED reflection discs. Subsequently, the pattern-matching refinements of the low-order structure factors will be altered and inaccurate. Hence, a reliable IAM is essential to accurate QCBED structure factor measurements, as we shall see in Chapter 5.

1.5. Aims and thesis outline

The primary aim of this project is to obtain accurate low-order structure factors of copper by QCBED, and therefore accurately characterise the bonding electron distribution of copper. In particular, whether copper has tetrahedral, octahedral or bridging bonding shall be determined. In order to achieve this subjective, the present work set two additional goals: the parallelisation of computer codes to accelerate QCBED calculations, and a more accurate IAM of copper to satisfy its QCBED refinement and bonding determination.

In contrast to previous QCBED studies of copper [4, 5, 7] using the Bloch-wave approach, this project will use the multislice formulation upon near-axis 2D CBED patterns. More detailed background knowledge, regarding 1) a literature review of bonding studies of copper, 2) available IAMs of copper, 3) the multislice formulation used to calculate the dynamic electron diffraction, and 4) the angular differential QCBED analysis technique applied in this project, will be given in Chapter 2. Experimental methodologies, which describes the specimen preparation, data collection and data analysis conditions of QCBED will be explained in Chapter 3. A sample size of more than a hundred CBED patterns, which is more than ten times of previous QCBED studies, is involved in this work. Therefore, the batch processing treatments for a bulky sample size is demonstrated in the last section of Chapter 3.

As briefly described in Section 1.4, QCBED refinements involve iterative dynamical-theory calculations. Even with the dramatically increased computational capacity in modern computers, because the resolution in pattern-matching are also increased in modern QCBED studies to produce more accurate approximations, high-precision dynamical-theory calculations are still time-consuming. Moreover, the sample size is also increasing, as fast-response CCD detectors and acquisition techniques like scanning-QCBED (more details in Section 3.3) become available. Therefore, reducing the time required by QCBED calculation is considered a tool-sharpening task, which will significantly benefit the current and future QCBED projects. The acceleration of multislice QCBED pattern-matching program, mainly by parallelisation processing, will be demonstrated in Chapter 4.

As discussed in Sections 1.3 and 1.4, an accurate IAM is necessary to obtain reliable bonding determination as well as QCBED refinements. Validation of the long-existed IAM calculation method using the RHF atomic scattering factors by Doyle and Turner [9] for copper, along with a new proposed IAM calculated by DFT, will be given in Chapter 5. Finally, with the new accelerated version of QCBED pattern-matching, results of bonding in copper featuring different IAMs and post-acquisition noise reduction technique will be given and discussed in Chapter 6.

Chapter 1. Introduction

Chapter 2. Background

Additional to Chapter 1, which briefly introduced the bonding definition and characterisation techniques, more detailed background information will be given in this chapter, which covers bonding and charge determination, the IAM of copper, the multislice formulation of the dynamical theory and the post-acquisition noise reduction technique used for QCBED analysis in this work.

2.1. Bonding and charge density in copper

As explained in Section 1.3, the bonding electron density $\rho(\mathbf{r})$ can be expressed by its X-ray structure factors $F_{\mathbf{g}}$ (in Eq. 1.2), while the $F_{\mathbf{g}}$ can be further expressed by the X-ray scattering factors f_X (in Eq. 1.3) and the temperature factor. By combining Eq. 1.2 and Eq. 1.3, for temperature at 0 K, the electron density can be derived from X-ray scattering factors by [76]:

Eq. 2.1
$$\rho(\mathbf{r}) = \sum_{\mathbf{g}} \sum_{j} f_{X,j}(s) \cdot \exp\left(2\pi i \mathbf{g} \cdot (\mathbf{r} - \mathbf{r_j})\right)$$

Similarly to the relationship between X-ray scattering factors and electron density, the electrostatic Coulomb potential distribution $\varphi(\mathbf{r})$ can be derived from electron scattering factors $f_{el}(s)$, which constitutes a description of electron scattering properties of a material during electron diffraction and will be utilised during QCBED calculations [76]:

Eq. 2.2
$$\varphi(\boldsymbol{r}) = K \sum_{\mathbf{g}} \sum_{j} f_{el,j}(s) \cdot \exp\left(2\pi i \mathbf{g} \cdot (\mathbf{r} - \mathbf{r_j})\right)$$

in which $K = 2\pi m_0 e/h^2$. m_0 and e are the mass and charge of an electron, and h the Planck constant. With the relation between electron density and potential field of all the atoms in a system described by the Poisson's equation, conversion between electron scattering factors $f_{el}(s)$ and X-ray scattering factors $f_X(s)$ can be derived by the Mott-Bethe formula [76]:

Eq. 2.3
$$f_{el}(s) = \frac{m_0 e^2}{8\pi\varepsilon_0 h^2} \cdot \frac{Z - f_X(s)}{s^2}$$

More than 30 studies of low-order structure factors of copper have been published since 1930. Several X-ray diffraction experiments using powder or single crystal of copper [42-44, 46-49] were conducted prior to 1970. However, limited agreement was found among these experiments. A detailed review on these researches was published in 1969 by Sirota [39]. In 1990, Tabbernor, *et al.* [3] published a deformation density study of copper, which also included several band theory calculations, low-angle electron diffraction, X-ray Pendellösung method and γ-ray results [39]. Results from different methods were considered to show sufficient consistency. It suggested that band theory results from Bagayoko *et al.* [16] would yield the best description of bonding in copper at then, which indicates bonding charge localised between nearest-neighbour atoms and next-nearest-neighbour atoms.

A more recent deformation density study on copper was published in 2005 by Friis *et al.* [7], which included rescaled γ-ray [15], QCBED [4, 32], and theoretical results by multipole analysis, maximum-entropy method and DFT, and band theory [16]. Conclusions were drawn that the two QCBED results had good consistency but were relatively far from the theoretical results. Furthermore, DFT and the multipole results have shown spherical charges at the atomic sites, which are expected as metallic bonding features. Compared to reviews by Tabbernor *et al.* [3] in 1990, better consistency is achieved by QCBED than by X-ray in low-order structure factors. The most recent study of copper structure factors was published in 2013 by Sang *et al.* [5], which aims to validate different density functionals with their QCBED results, but only the F_{111} and F_{200} are reported. The selected DFT values of Sang *et al.* have the best matching to their QCBED results, which were calculated using a functional tuned towards the QCBED values.

	hkl	Rusterholz (1930)	Brindley & Spiers (1935)	Brindley (1936)	Rovinski (1937)	Ageev *et al.* (1948)	Batterman *et al.* (1959)
X-ray	111	18.07	20.11	21.55	21.21	21.30	22.20
	200	16.52	18.56	19.85	19.74	19.50	20.60
	220	14.05	15.11	16.14	16.23	16.30	17.00

	hkl	Batterman *et al.* (1961)	Jennings *et al.* (1964)	Hosoya & Yamagichi (1966)	Temkin *et al.* (1972)	Freund (1973)
X-ray	111	21.29 ± 0.34	21.52 ± 0.10	22.02 ± 0.09	21.93 ± 0.15	22.63 ± 0.20
	200	19.75 ± 0.34	-	20.62 ± 0.18	20.36 ± 0.15	-
	220	16.37 ± 0.30	-	16.99 ± 0.13	16.70 ± 0.16	-

	hkl	Arlinghaus (1967)	Snow (1968) α=1	Snow (1968) α=5/6	Snow (1968) α=2/3	Wakoh & Yamashita (1971)	Bagayoko *et al.* (1980)	MacDonald *et al.* (1982)	Eckardt *et al.* (1984)
Band Theory	111	21.54	22.33	21.90	21.63	21.72	21.68	21.73	21.95
	200	20.25	21.04	20.66	20.40	20.46	20.35	20.39	20.68
	220	16.39	17.12	16.86	16.64	16.63	16.62	-	16.90

	hkl	Humphreys (1975)	Thomas *et al.* (1974)	Smart & Humphreys (1980)	Fox & Fisher (1988)
Critical voltage	111	21.75 ± 0.03	21.70 ± 0.02	21.76	21.72 ± 0.04
	200	20.43 ± 0.04	20.42 ± 0.02	20.43	20.45 ± 0.04
	220	-	-	16.68	16.68 ± 0.08

	hkl	Smart & Humphreys (1980)	Takama & Sato (1982)
Pendellösung	111	21.79	21.80 ± 0.06
	200	20.45	20.28 ± 0.11
	220	16.70	16.75 ± 0.08

	hkl	Schneider (1981)	Mackenzie & Mathieson (1984)	Petrillo *et al.* (1998)
γ-ray	111	21.51 ± 0.05	-	21.68 ± 0.14
	200	20.22 ± 0.04	-	20.38 ± 0.13
	220	16.45 ± 0.05	16.76	16.60 ± 0.12

	hkl	Saunders *et al.* (1999)	Friis *et al.* (2005)	Sang *et al.* (2013)
DFT	111	21.71	21.70	21.79
	200	20.37	20.38	20.45
	220	16.66	16.67	-

	hkl	Saunders *et al.* (1999)	Friis *et al.* (2005)	Sang *et al.* (2013)
QCBED	111	21.78 ± 0.02	21.69 ± 0.03	21.80 ± 0.03
	200	20.44 ± 0.02	20.44 ± 0.02	20.45 ± 0.04
	220	16.72 ± 0.11	16.68 ± 0.02	-

Table 2.1. Summary of low-order structure factors of copper from previous studies [3, 4, 7, 11-18, 31, 32, 39-49, 77-79]. Only the three lowest-order structure factors are listed, while higher-order structure factors are also available for band theory, γ-ray, some DFT and QCBED studies in their original publications. Because different lattice parameters of copper are reported in the studies, the scattering angles are unique for each study and not listed.

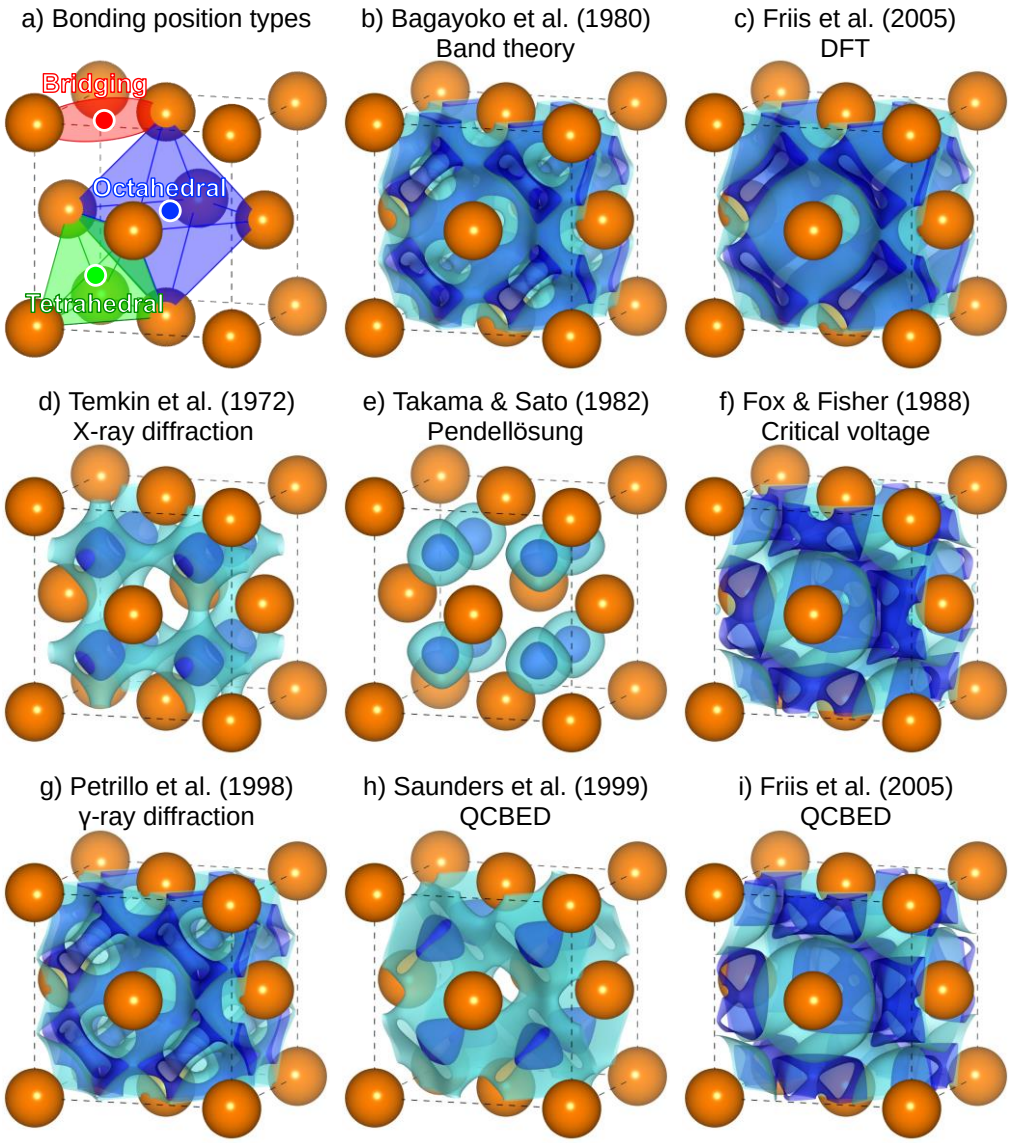


Figure 2.1. The bonding electron distributions in copper: a) (and any FCC crystal) are described by the location of the major electron density position; b) – i) measured/calculated by different studies [4, 7, 11, 12, 14, 16, 18] (these studies are selected because they are either the most cited in bonding studies of copper or the most recent publications of the techniques). The RHF-IAM (derived from Doyle and Turner's atomic scattering factors [9]) is used for bonding determination. The iso-surfaces are plotted at 50% (light blue) and 80% (dark blue) of the maximum bonding electron density of each result. As in b), c) & g), the bonding of copper is suggested to localise at the bridging sties; d), e) & h) indicate copper has tetrahedral bonding; while both f) & i) agree the major bonding in copper is localised around the octahedral sites with lower densities at the octahedron centre, as well as some bonding localised at the bridging sites. Images of bonding electron densities in this graph are generated by VESTA [10].

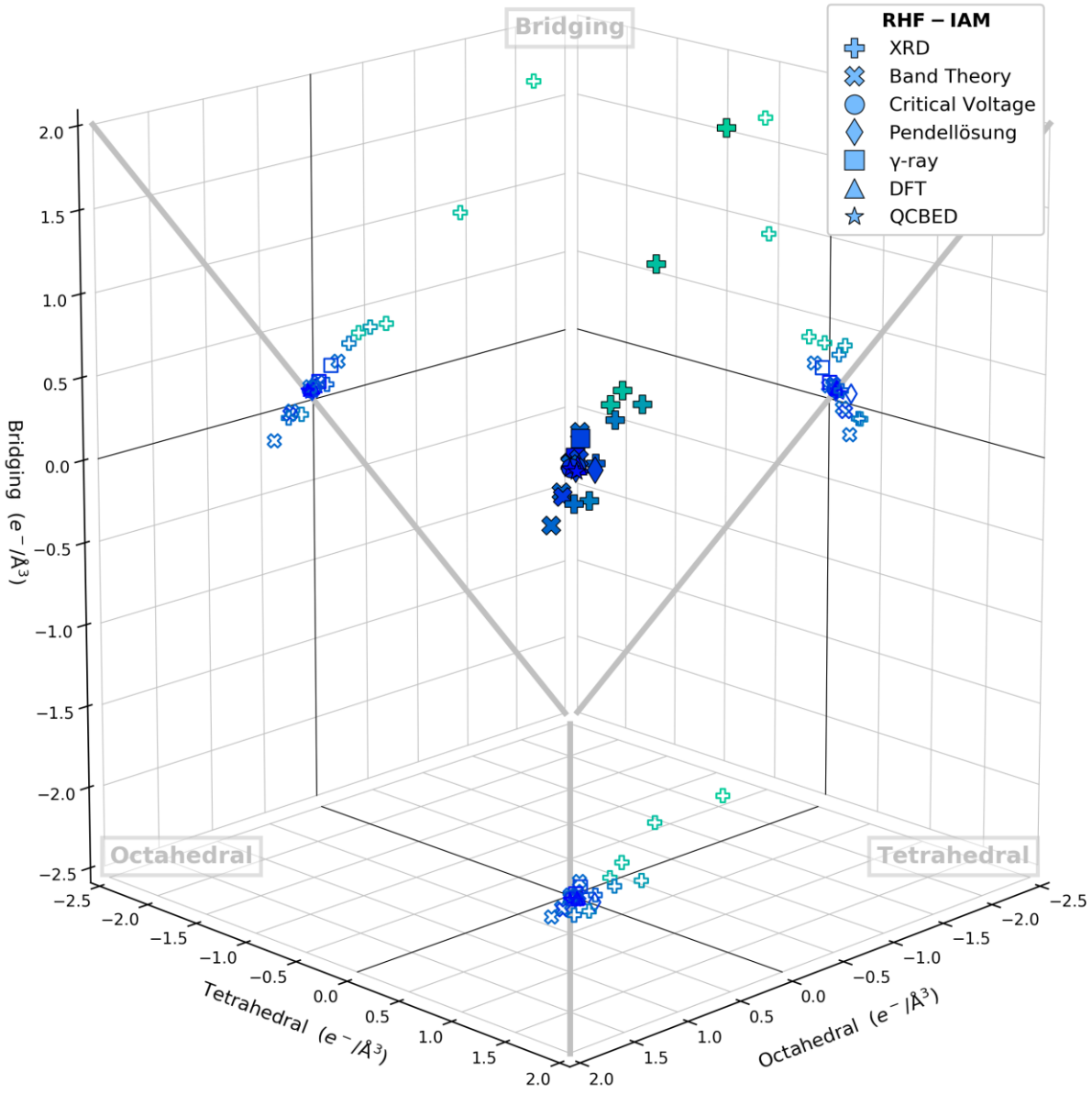


Figure 2.2. Summary of bonding in copper from previous studies, using the RHF-IAM (derived from Doyle and Turner's atomic scattering factors [9]). The intensities of charge density ($e^-/\text{Å}^3$) at the tetrahedral sites, octahedral centre and bridging sites of copper's unit cell are used to estimate the shapes of bonding distributions (please refer to Figure 2.1 for shapes of bonding). For values above 0 $e^-/\text{Å}^3$, anti-bonding is indicated to occur at the corresponding location. Results of different studies are grouped by their techniques. Shades from light to dark blue are used to indicate the publication years from 1930 to 2005. Experimental techniques include XRD (9 sets) [18, 39, 43, 45, 47-49, 80], critical voltages (2 sets) [4, 12], Pendellösung (2 sets) [14, 32], γ-ray (2 sets) [11, 15] and QCBED (2 sets) [12, 78]. Theoretical calculations include band theory (7 sets) [13, 16, 39, 41, 79] and DFT (2 sets) [4, 7]. The bonding charge densities are calculated with three lowest-order structure factors. The diversities among XRD results are the greatest (even excluding the farthest two points, which are from the two earliest studies). Relatively, critical voltages, DFT and QCBED studies have better agreement among different studies. Additionally, results from critical voltages, DFT and QCBED studies suggest that the magnitudes of copper bonding are relatively small, comparing to the other techniques.

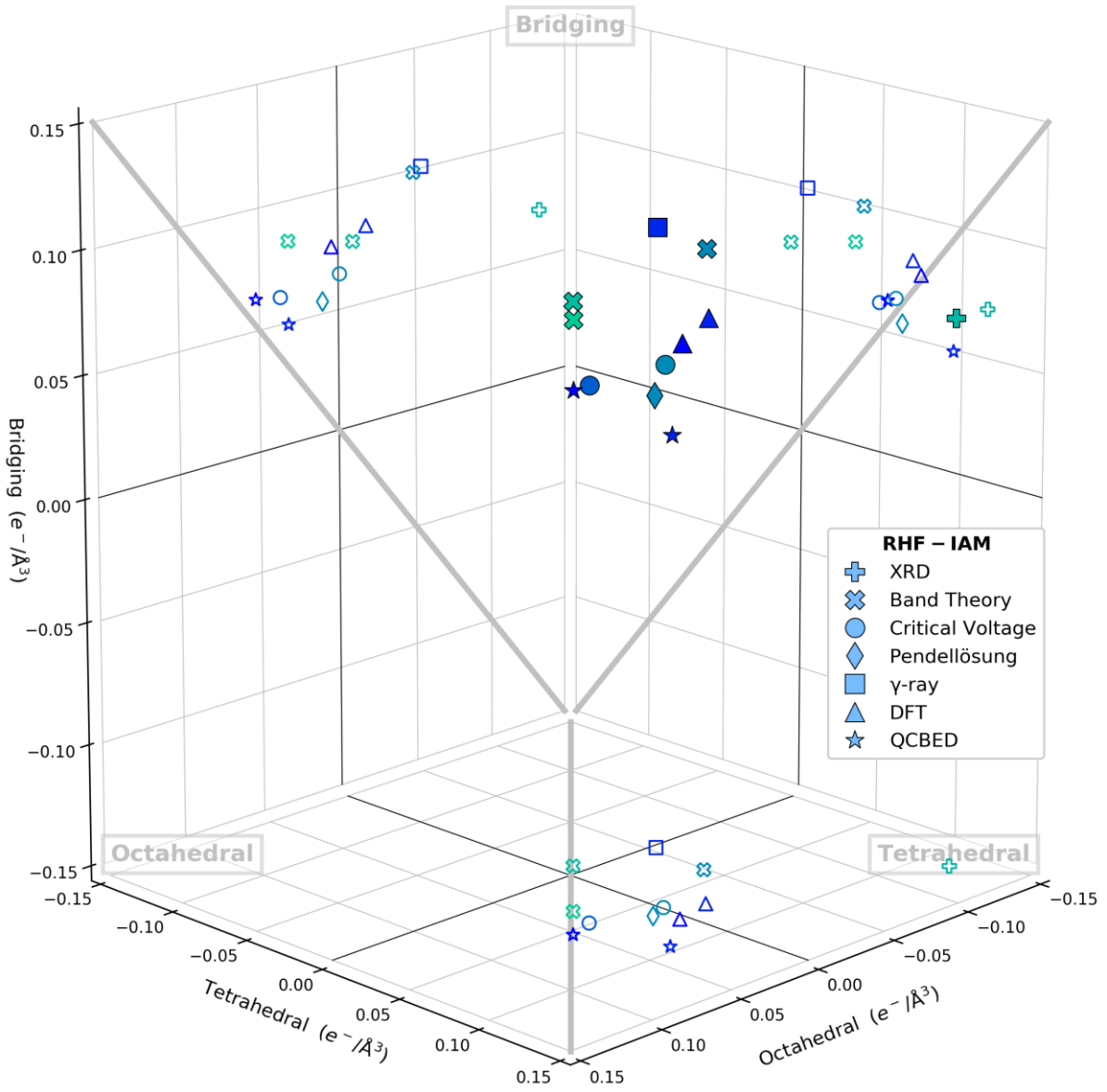


Figure 2.3. A zoom-in graph of Figure 2.2 which includes most of the more recent studies [4, 7, 11, 12, 16, 18, 41, 79]. Summary of bonding in copper from previous studies, using the RHF-IAM (derived from Doyle and Turner's atomic scattering factors [9]). The intensities of charge density ($e^-/Å^3$) at the tetrahedral sites, octahedral centre and bridging sites of copper's unit cell are used to estimate the shapes of bonding distributions (please refer to Figure 2.1 for shapes of bonding). For values above 0 $e^-/Å^3$, anti-bonding is indicated to occur at the corresponding location. Results of different studies are grouped by their techniques. Shades from light to dark blue are used to indicate the publication years from 1968 to 2005. Results include XRD (1 set) [18], band theory (3 sets) [16, 41, 79], critical voltages (2 sets) [4, 12], Pendellösung (1 set) [78], γ-ray (2 sets) [11, 15] and QCBED (2 sets) [4, 7] and DFT (2 sets) [4, 7]. The bonding charge densities are calculated with three lowest-order structure factors. The diversities among XRD results are the greatest (even excluding the farthest two points, which are from the two earliest studies). Relatively, critical voltages, DFT and QCBED studies have better agreement among different studies. Additionally, results from critical voltages, DFT and QCBED studies suggest that the magnitudes of copper bonding are relatively small, comparing to the other techniques.

A summary of 34 sets of low-order structure factors of copper measured/calculated by different techniques is listed in Table 2.1. Generally, recent studies have smaller uncertainties, although their differences are still outstanding (except the DFT and QCBED results of Sang *et al.* [5]). Because most studies have adopted different lattice parameters for copper, the values of scattering angles and IAM structure factors were reported to be different as well. Lattice constant $a = 3.6024$ Å for copper [81] is used for unified scattering angles. Atomic scattering factors by Doyle and Turner using RHF [9] are used to yield the IAM of copper. The bonding electron distributions in copper measured/calculated by different studies [4, 7, 11, 12, 14, 16, 18] (these studies are selected because they are either the most cited in bonding studies of copper or the most recent publications of each technique) are plotted in Figure 2.1. Although the bonding distribution iso-surfaces are plotted as 50%/80% of the maximum bonding intensity in each result (to represent only the morphology of bonding), various bonding types are concluded by different studies.

A visualisation of bonding distributions derived from 26 sets (data sets without any of F_{111}, F_{200} and F_{220} are ignored) is summarised by the bonding electron densities at the octahedral, tetrahedral and bridging sites in Figure 2.2. The ratios of bonding densities of the three bonding sites indicate the localisation of bonding electrons of different data sets. For instants, the two XRD data points that depart from the others, because they are in the bridging region and far away from the octahedral and tetrahedral regions, have indicated that very strong localised bonding in the bridging site. The shades of data points, which is coloured according to the publication times, have shown that results from more recent studies cluster near the origin and relatively less bridging-like. The overall linear trend is likely raising from the systemic errors in the magnitudes of structure factors, as earlier results (light blue shaded data points in Figure 2.2) have smaller F_{111}, F_{200} and F_{220} as well. A zoom-in image is plotted to show the distributions of the clustered recent studies in Figure 2.3. Generally, the clustered recent studies, which have relatively better agreement on bonding magnitudes and morphologies, have shown bridging- and tetrahedral-like bonding, with weak octahedral intensities. Except for the Fox and Fisher critical voltage [12] and Friis *et al.* QCBED [7] results as shown in Figure 2.1, where most bonding densities are located within the octahedral sites but lower at the very centre of the octahedral atom cage.

In 2003, Friis *et al.* [8] conducted an experiment aiming to experimentally verify DFT calculations. QCBED (systemic row) and X-rays were used to measure low- and high-order structure factors respectively of magnesium and beryllium. Agreement between DFT and QCBED results showed that QCBED had the ability to produce sufficiently accurate deformation density maps. Comparison

between deformation density maps of the two HCP metals, magnesium and beryllium also showed that bonding in magnesium is closer to the free-electron model of metals as expected.

As a comparison of another FCC metal, a study on the deformation charge of aluminium, was published by Nakashima *et al.* [6] in 2011, using QCBED and DFT. In contrast to the QCBED and DFT copper studies by Saunders *et al.* in 1999 [4] and Friis *et al.* [8] in 2005, results of Nakashima *et al.* [6] show very close agreement between QCBED and DFT. The interatomic bonding was concluded to be tetrahedral and centred in four-nearest-atom tetrahedra only and without bonding deformation in the octahedra. This bonding geometry is similar to DFT calculations of copper (also an FCC metal) by Friis *et al.* [7], but apparently different from the QCBED results [4, 32] reviewed in Friis *et al.* [7] and band theory calculation by Bagayoko *et al.* [16]. Bagayoko, *et al.*'s work [16] was attributed as the most reliable study in the review by Tabbernor *et al.* [3].

2.2. Independent-atom model of copper

2.2.1. Atomic scattering factors

For calculations of the independent neutral atom electron density or potential field, the RHF atomic wave function method by Coulthard [82] is the most commonly adopted [3, 4, 6-9, 15, 31-33, 62, 76]. However, to obtain an electron density for analytical applications, rather than solving RHF directly, it is usually preferred to use numerically approximated scattering factors, which are carefully curve fitted to the scattering factors that were calculated from RHF electron density. This is due to the fact that this method is more computationally effective in practice. The most widely used atomic scattering factors were published by Doyle and Turner in 1968 [9]. These tabulated atomic scattering factors are regarded as the standard values since the publication. This publication [9] covers 76 common atoms and ions, giving tables with 27 values of X-ray and electron scattering factors respectively for each atom and ion from 0 to 6 Å^{-1} of s ($\frac{\sin\theta}{\lambda}$). X-ray scattering factors from 0 to 2 Å^{-1} of s for other atoms and ions were calculated by Cromer and Waber [83], and later the neutral atoms in these tables were extended to 6 Å^{-1} of s by Fox *et al.* [59]. This collection of X-ray and electron scattering factors was later published in the International Tables for Crystallography [58, 84], forming tables that cover atoms and ions from number 1 to 98. The neutral atoms tables include up to 62 values from 0 to 6 Å^{-1} of s for both X-ray and electron scattering factors, while the ion tables include up to 56 values from 0 to 2 Å^{-1} of s. Note that for atoms and ions that are not published by Doyle and Turner [9] (e.g. hydrogen), their electron scattering factors were converted from their X-ray values using the Mott-Bethe formula [76] as shown in Eq. 2.3.

In practice, these scattering factor tables will be interpolated to return either X-ray or electron scattering factors for the desired *s*. There are two interpolation solutions: the original and more popular method called parameterisation [9, 59, 60, 62, 63, 85], and polynomial interpolation [64]., as described in the following section.

2.2.2. Numerical parameterisation and polynomial interpolation

The electron and X-ray scattering factors determined from RHF calculations for scattering angles s from 0 to 6 Å^{-1} for most neutral atoms are tabulated by Doyle and Turner in 1968 [9], which are regarded as the most accurate scattering factors and are the most cited. In the same paper, a formula is also given to allow the interpolation of scattering factors. It involves parameterised Gaussian sums, curve fitted for the range from $s = 0$ to 2 Å^{-1}:

Eq. 2.4
$$f(s) = \sum_{i=1}^{4} a_i e^{-b_i s^2} + c,$$

where a_i, b_i and c are fitting parameters ($c = 0$ for $f_{el}(s)$).

For high-resolution electron microscopy image simulations and QCBED calculations, electron scattering factors higher than 2 Å^{-1} are required to produce accurate simulations [58]. However, Fox *et al.* [59] showed that the parameterisation by Doyle and Turner [9] has poor accuracy at higher angles above $s = 2$ Å^{-1} (100% error for X-ray scattering factors in some cases, and 10% for electron scattering factors derived by the Mott formula from X-ray scattering factors at high angles). Polynomial curve fittings of X-ray scattering factors were given as an alternative for the range $s = 2$ to 6 Å^{-1}:

Eq. 2.5
$$f_x(s) = \exp(a_0 + a_1 s + a_2 s^2 + a_3 s^3)$$

The correlation coefficients of these fits were above 0.999 and the errors were below 3% for most elements. Electron scattering factors derived from these X-ray scattering factors had errors less than 1% [59].

In 1994, a new table of X-ray scattering factors was published by Rez *et al.* [86], [87]. Although the X-ray scattering factors of copper were of small difference to Doyle and Turner's publication [9],

updated Gaussian sum curve fitting parameters were given for a range of s from 0 to 2 Å^{-1} with better fitting and for an extended range of s from 0 to 6 Å^{-1} with relatively poorer accuracy respectively. Electron scattering factors were not given, but it was suggested to use the Mott formula to derive them from the X-ray scattering factors. However, for low angle scattering factors, the Mott-formula conversion from X-ray scattering factors to electron scattering factors may result in additional errors as X-ray scattering factors are close to their atomic numbers, which results in the terms $Z - f_x(s)$ and s^2 in Eq. 2.3 tending to zero.

In 1996, with a newly developed algorithm based on simulated-annealing and least-squares method for parameter optimisation, Peng *et al.* [62] developed new parameterised formulas for electron scattering factors in the form of five Gaussian sums:

Eq. 2.6

$$f(s) = \sum_{i=1}^{5} a_i e^{-b_i s^2}.$$

Similarly to Rez *et al.* [86], parameters for ranges from 0 to 2 Å^{-1} and from 0 to 6 Å^{-1} of s were given separately. Formulas of Peng *et al.* [62] for both ranges are currently included in the International Tables for Crystallography [58].

There are several parameterised electron scattering factor numerical approximations published in forms other than Gaussian summations [60, 63, 85]. Kirkland [85] has used 3 Lorentzian and 3 Gaussian summations to fit the electron scattering factors. As the most recent publication, Lobato and Van Dyck [60] have provided 5-Lorentzian-summation formulas and resulted the best fitting to the tabulated electron scattering factors. The parameterised formulas by Lobato and Van Dyck [60] were also proposed with an extrapolation range of s up to 12 Å^{-1}.

Apart from parameterisation, a less popular interpolation method is polynomial interpolation. Bird and King [64] published a FORTRAN subroutine, *ATOM*, which primarily aims to return both the real and imaginary components of electron atomic scattering factors. For the real component, *ATOM* uses cubic Lagrange interpolation to interpolate the electron scattering factors. Due to the nature of polynomial interpolation, *ATOM* can return a curve of electron scattering factors with perfect fitting to its embedded tabulated values.

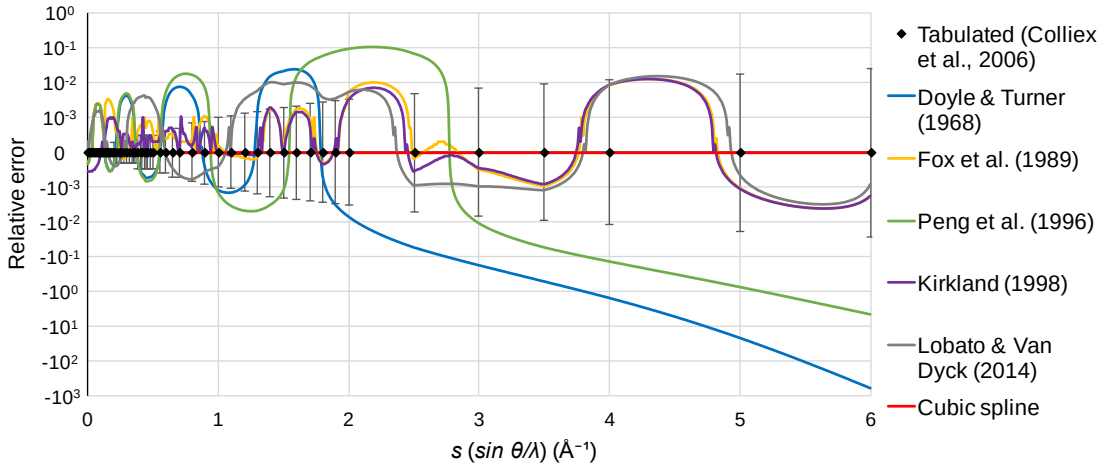


Figure 2.4. The relative errors of electron atomic scattering factors of copper by different interpolation methods (parameterisation for $0 \leq s$ (sin θ/λ) ≤ 6 Å^{-1} is used for Peng *et al.*) [9, 59-62, 76], using the cubic spline interpolation of the tabulated atomic scattering factors[58] as the reference (because polynomial interpolations guarantee perfect fitting at known values). The error bars stand for the rounding errors introduced by the thousandth-precision of the tabulated values. It is shown that the relative errors of Doyle & Turner[9] and Peng *et al.*[62] exceed 10% when the scattering angle $s > 3$ and 4 Å^{-1} respectively.

In addition, all previous copper bonding studies [3, 7, 39] used IAM derived from atomic scattering factor interpolation of Doyle & Turner [9] or Peng *et al.* [62]. However, there are studies [59, 65] suggesting that the numerical values of copper atomic scattering factors derived from these two interpolation methods have poor fitting of high scattering angles (as shown in Figure 2.4). As discussed previously in Section 1.4, inaccurate atomic scattering factors will lead to inaccurate structure factors and hence inaccurate IAM charge distributions. Besides, for QCBED analysis, the high-order structure factors will be affected by inaccurate high-order atomic scattering factors. Thus, reliable atomic scattering factors and interpolations are essential for IAM and QCBED charge distribution calculations and deformation charge studies. This issue will be the focus of Chapter 5.

2.3. Multislice formulation of dynamic electron diffraction

To describe dynamic electron diffraction in a specimen, different formulations aiming to solve the time-independent Schrödinger equation are available. The Bloch-wave and multislice formulations are the most commonly used. In 1928, Hans Bethe, in his PhD thesis, used "Bloch waves" to successfully describe the dynamic electron diffraction behaviours in reflection experiments [51] reported by Davisson and Germer [88] and Thomson and Reid [89]. In 1939, Blackman extended Bethe's Bloch-wave theory to transmission cases [90]. In the late 1950s, Lorenzo Sturkey developed the "scattering matrix" approach [91, 92], which has simplified the Bloch-wave formulation and become the general

form of Bloch-wave formalism at present [93]. Also in the late 1950s, the multislice method, a formulation different to the Bloch-wave, was developed by Cowley and Moodie [94].

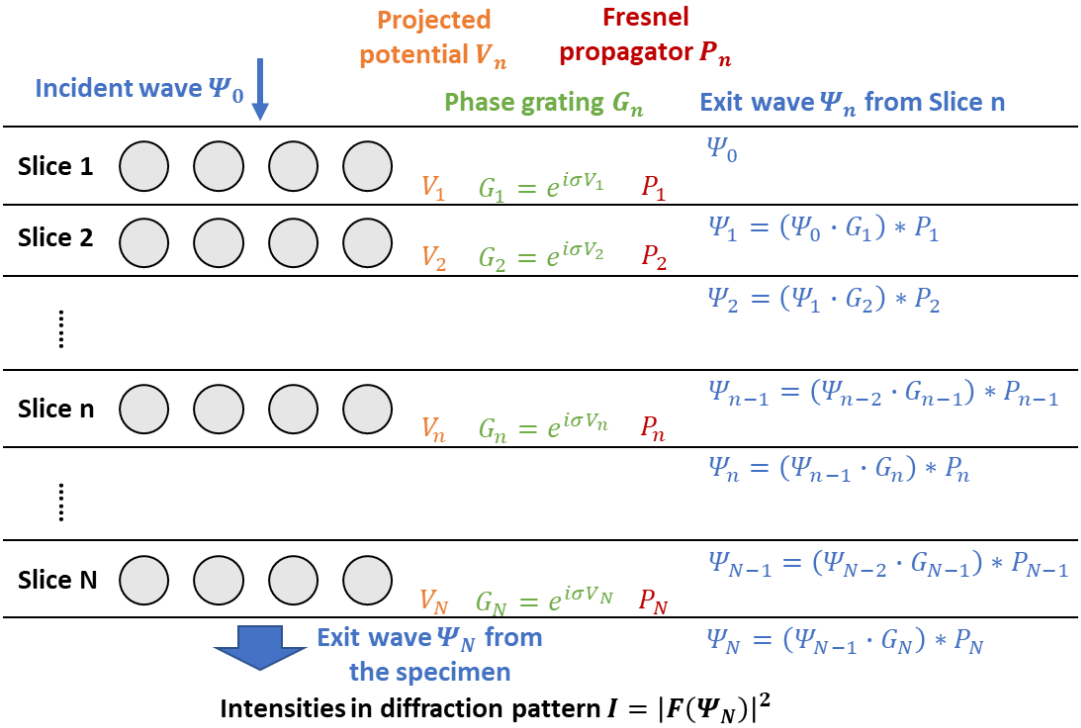


Figure 2.5. A schematic diagram of the multislice approach. A specimen is cut in into *N* slices (by individual layer of atoms). The wave Ψ_n exited from the n^{th} slice will be modified by the slice's potential V_n as well as its propagation distance. The modification caused by V_n is summarised as the phase grating $G_n = \Psi_{n-1} \cdot e^{i\sigma V_n}$, where $\sigma = \frac{2\pi me\lambda}{h^2}$. The propagation over the distance of the slice is described by the Fresnel propagator P_n. $*$ denotes convolution of the Fresnel propator and the wave modified by the phase grating. After iterating over the *N* slices, the intensities in the diffraction pattern of the specimen can be obtained from the absolute square of the Fourier transformation of the exit wave of the specimen, denoted as $|F(\Psi_N)|^2$.

A schematic diagram of the multislice approach is shown in Figure 2.5. To solve the time-independent Schrödinger equation, the multislice approach, utilising the Huygens-Fresnel principle, describes a specimen as multiple slices (usually splitting by individual layer of atoms) [94]. The exit wave Ψ_n from the n^{th} slice is described as:

Eq. 2.7

$$\Psi_n = (\Psi_{n-1} \cdot G_n) * P_n,$$

where Ψ_{n-1} is the exit wave from the previous slice (the incident wave is denoted as Ψ_0). G_n is the phase grating, which describes the modification to the wave caused by the potential, V_n, of the slice.

$G_n = e^{i\sigma V_n}$, where $\sigma = \frac{2\pi me\lambda}{h^2}$ and is called as the interaction constant. P_n is the Fresnel propagator, which describes the propagation of the wave across the distance of the slice. Convolution, denoted as $*$, of the Fresnel propator and the wave modified by the phase grating results in the exit wave of the slice. By iterating over the N slices of the specimen, the final resulted intensities in the diffraction pattern are obtained as the absolute square of the Fourier coefficients of the exit wave Ψ_N: $I = |F(\Psi_N)|^2$, where $F(\Psi_N)$ denotes the Fourier transformation.

The Bloch-wave and multislice approaches have different advantages. Bloch-wave calculations can be very fast for crystals with small unit cells, where a relatively small number of reflection beams are required. However, for crystals with large unit cells, or specimens with multi-layered structures, the amount of computation will increase significantly, as the number of included reflection beams and/or extra scattering matrices required for heterogenous structures increases. On the other hand, the multislice approach can generally achieve faster calculations than the Bloch-wave method when more and more reflection beams are included. Also because of its "multislice" assumption, the multislice approach is more suitable for multi-layered structures (smaller computation-time penalty). The multislice method was used for QCBED calculations of the bonding in copper in this project, both as a distinguish approach to previous studies of copper [4, 7] and as a chance to further develop the approach itself. A more detailed discussion regarding the computation performance of multislice will be given in Section Chapter 4.

2.4. Angular differential QCBED

There are several significant sources of noise/uncertainties that contribute to the total intensities of acquired experimental CBED patterns and differ them from calculated theoretical patterns. To characterise these uncertainties and reduce their significance, the angular differential QCBED (ADQCBED) technique [95] is used in this project. The main features of ADQCBED include: 1) point-spread effect characterisation and elimination, 2) identification and reduction of electric current noise from CCD detector readout, 3) pattern-matching on angularly differentiated 2D near-axis patterns, 4) a gaussian function to model the absorption structure factors, and 5) geometrical distortion correction. The characterisation and elimination of the point-spread effect and current noise and pattern angular differentiation are data pre-processing steps. The scattering factor absorption modelling is executed during QCBED pattern-matching along other refining parameters (e.g. thickness and structure factors). The geometrical distortion correction is executed as an extra step that relies on theoretical QCBED

simulation to identify and correct distortions in the experimental data that are independent to parameters being refined in pattern-matching.

The pattern-matching program used in this work is a modified version of [6, 96], which will be introduced in Chapter 4. The aim of the program is to find the optimised parameters (sample thickness and structure factors etc.) that produce the lowest residuals between the experimental and theoretical pattern. The residual is evaluated as:

Eq. 2.8

$$\chi = \sqrt{\frac{\sum_{i=1}^{n} \frac{w_i \cdot \left(I_i^{expt} - I_i^{theor}\right)^2}{1+\sigma_i^2}}{\sum_{i=1}^{n} \frac{w_i \cdot I_i^{expt\,2}}{1+\sigma_i^2}}}$$

where i stands for each pixel in the pattern, w_i are the weights associated with each pixel (typically 0 or 1, to define the region to match), I_i^{expt} and I_i^{theor} are the experimental and the corresponding theoretical intensities of a pixel respectively, and σ_i^2 is the variance and the characterised electric current noise from CCD detector readout. Perfect matching will result in $\chi = 0$, when all $I_i^{expt} = I_i^{theor}$.

2.4.2. Measuring the point-spread function

The point-spread effect describes the over-spreading intensities of a single electron on the detector instead of a point signal. This phenomenon causes blurring of the detected signal, by overflowing intensities to the neighbouring pixels. This is intrinsic to conventional CCD detectors, and occurs whenever electrons hit the detector. By analysing the diffraction aperture images (without the presence of specimen), it is possible to isolate this issue and characterise the shape of the spreading, which is represented as the point-spread function [97]. Once the point-spread function is characterised from the aperture image, the point-spread effect can be deconvoluted from the captured CBED patterns.

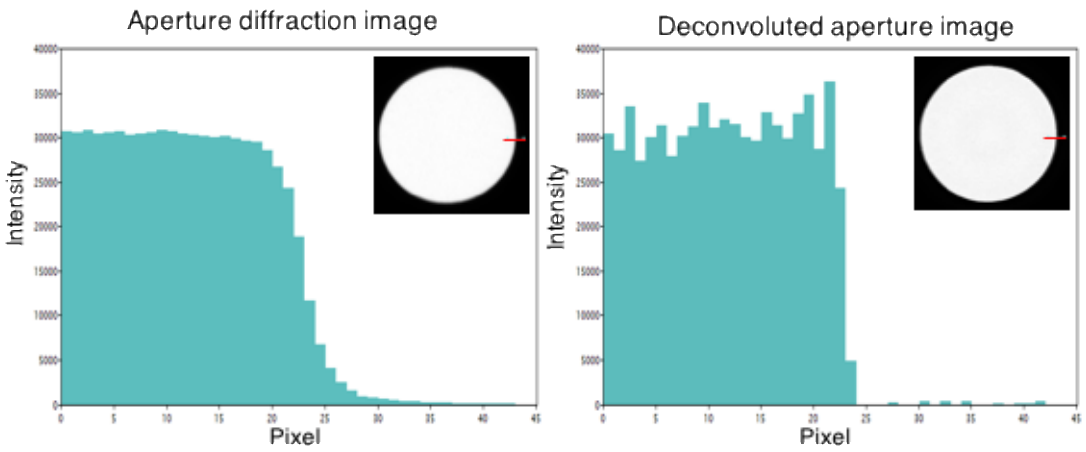


Figure 2.6. Image (left) of a captured diffraction image of the aperture on TEM, with a blurring edge. Image (right) of the same aperture image after deconvolution of its characterised point-spread-function, with a sharpened edge.

2.4.3. Characterisation of signal noise from CCD detector readout

Apart from the point-spread function, electron signals will also cause noise, from detection to readout of a CCD detector [98]. To account for the total uncertainties contributed from this source, an *in-situ* noise characterisation algorithm [99] was used, which assumes the noise σ is a function of signal I per pixel:

Eq. 2.9

$$\sigma = a \cdot I + b \cdot \sqrt{I} + c$$

where a, b and c are fitting efficiencies determined *in-situ*. The determined uncertainties associated to each pixel are used as a weighting factor during pattern-matching.

2.4.4. Tackling inelastic scattering

Compared to other sources of errors, inelastic scattering is a factor that will significantly affect the simulation of CBED patterns, and is even noticeable visually. Inelastic scattering is the phenomenon of electrons losing energy after being scattered by the specimen. In contrast, elastic scattering of electrons remains unchanged energy and are predictable. Major sources of inelastic scattering signals are plasmon and thermal diffuse scattering (TDS). It is known that the plasmon and some TDS signals have a similar intensity distribution to the elastic scattering signals, but in a more slow-varying and blurring fashion; other TDS signals are much less sensitive to the specimen and behave like a constant background noise contribution to the total intensities [96]. Energy-filtering optics, which include post column energy filters and in-column Omega filters, can be used to filter out most of the inelastic signal.

In this project, both energy-filtered and -unfiltered patterns were collected and analysed. Extra post-collection processing techniques were used to reduce the significance of inelastic scattering in the unfiltered patterns.

a) Angular differentiation of QCBED: TDS reduction

Established theories explain that the signals in a CBED pattern consist of 1) a parameter-sensitive (e.g. specimen thickness, structure factors and scattering angle) elastic component, 2) a parameter-sensitive inelastic component and 3) a less-parameter-sensitive inelastic component [96]. The elastic component summarises all the electron elastic behaviours, the parameter-sensitive inelastic component consists of plasmon and partially TDS signals, and the less-parameter-sensitive is mostly TDS. As shown in Eq. 2.10, the total intensity distribution *I* in CBED can be described as [96]:

Eq. 2.10
$$I = I_{el}\left(F_{el,\mathbf{g}}, H, \theta\right) + I_{inel}\left(F_{el,\mathbf{g}}, H, \theta\right) + I_{inel}(\theta)$$

where $F_{el,\mathbf{g}}$, H and θ are the structure factors, specimen thickness, and scattering angle respectively, $I_{el}\left(F_{el,\mathbf{g}}, H, \theta\right)$ the elastic component, $I_{inel}\left(F_{el,\mathbf{g}}, H, \theta\right)$ the sensitive inelastic component and $I_{inel}(\theta)$ the relatively less sensitive inelastic component. In this expression, the intensity distribution of the $I_{el}\left(F_{el,\mathbf{g}}, H, \theta\right)$ component can be well predicted; the $I_{inel}\left(F_{el,\mathbf{g}}, H, \theta\right)$ intensity distribution is similar to $I_{el}\left(F_{el,\mathbf{g}}, H, \theta\right)$ but in a more diffuse manner; and $I_{inel}(\theta)$ is usually modelled as a locally linear function of θ. Therefore, a computable equation of the CBED signals can be expressed as [96]:

Eq. 2.11
$$I \approx \eta \cdot I_{el}\left(F_{el,\mathbf{g}}, H, \theta\right) + \beta \cdot \theta + \gamma \quad (\eta > 1)$$

where $I_{inel}\left(F_{el,\mathbf{g}}, H, \theta\right)$ in Eq. 2.10 is included in $\eta \cdot I_{el}\left(F_{el,\mathbf{g}}, H, \theta\right)$, and $I_{inel}(\theta)$ is modelled by $\beta \cdot \theta + \gamma$, with β and γ representing constant coefficients.

To reduce the influence of inelastic components and achieve better pattern-matching, techniques like energy-filtering and differential QCBED were developed [100]. Energy-filtering, which include post column energy filters and in-column Omega filters, can be used to filter out most of the inelastic signal including plasmon losses (i.e. losses greater than about 2eV). The η component from Eq. 2.11 in energy-filtered patterns can be close to 1 [101].

Although energy filters can eliminate most of the plasmon signals, they cannot remove the TDS contribution as it involves energy loss less than 0.1 eV, which cannot be filtered out by energy filters [95]. Fortunately, the insensitive inelastic (TDS) signals can be reduced by the post-acquisition technique of differential QCBED. As the TDS intensity distribution is considered as insensitive to the three parameters, $F_{el,\mathbf{g}}$, H and θ, subtraction of two CBED patterns with small variation of any parameter will theoretically eliminate the TDS signals to a good approximation [96]. For example, the ADQCBED technique will have an approximate expression as [95]:

Eq. 2.12
$$\frac{dI}{d\theta} \approx \eta \frac{dI_{el}\left(F_{el,\mathbf{g}}, H, \theta\right)}{d\theta} + \beta \quad (\eta > 1)$$

which in brief is equivalent to subtracting every pixel in a CBED pattern by a neighbouring pixel, because each pair of pixels is considered to have linearly related TDS contributions, as it is a slowly varying background, but different $I_{el}\left(F_{el,\mathbf{g}}, H, \theta\right)$ and $I_{inel}\left(F_{el,\mathbf{g}}, H, \theta\right)$ contributions that are sensitive to varying scattering angle. This is carried out in practice by subtracting the shifted patterns in four pairs of opposite directions (north, south, west, east, north-west, south-east, south-west and north-east, in order to eliminate the shifting shadow) from the original CBED patterns.

a) Phenomenological absorption

With angular differentiation, the significance of TDS in unfiltered data can be well reduced. However, while differential QCBED removes the slowly varying diffuse background in CBED patterns, the structured background largely contributed by plasmon losses may persist. Because the absorption contribution mimics the elastic signal, simulated patterns can be better approximated by applying a phenomenological absorption model. In the calculation of a CBED pattern, phenomenological absorption can be dealt with in terms of an imaginary component that contributes to the complex scattering factor associated with electron scattering factor. The overall electron scattering factor $f_{el,\mathbf{g}}^{tot}$ for a type of atom may be expressed roughly as:

Eq. 2.13
$$f_{el,\mathbf{g}}^{tot} = f_{el,\mathbf{g}} + i f'_{el,\mathbf{g}}$$

where $f_{el,\mathbf{g}}$ is the atomic electron scattering factor, and the imaginary component $f'_{el,\mathbf{g}}$ is the absorption term. A well-known model for $f'_{el,\mathbf{g}}$ developed by Bird and King [64] is available for energy-filtered data. Because both filtered and unfiltered data are used in this project, a new function of scattering

angle s was used to model the absorption of each atom type (introduced in an unpublished paper by Nakashima *et al.*, the draft of which is attached in Appendix D as Paper 1):

Eq. 2.14 $$f'_{el,g}(s) = a \cdot \exp(-10^6 \cdot s^2) + b \cdot \sqrt{B} \cdot \exp(-c \cdot s^d)$$

where a, b, c and d are fitting parameters refined for each pattern and B the Debye-Waller factor. With such configurations, the temperature, which affects the absorption significantly, is also taken into account.

2.4.5. The geometric distortion correction

Apart from the above uncertainties, geometric distortion caused by aberration may also occur during signal collection. A non-linear geometric distortion correction program written in C [102] was used in this project to reduce such distortions. This program serves to recover the undistorted experimental pattern by comparing it to a theoretical pattern and applying geometric transformations to the experimental pattern for further pattern-matching. In order to obtain a correct distortion characterisation, the theoretical refinement results have to be very near to the true (distortion-free) experimental pattern. For example, the sample thickness and pattern coordinates, which are sensitive to pattern geometry, must be determined accurately before distortion correction. Thus further pattern-matching can yield more accurate and reliable refinement results for structure factors, which are sensitive to detailed features in CBED patterns.

With these complex noise characterisation and reduction techniques of ADQCBED, uncertainty from data processing of experimental CBED pattern can be minimised. In addition to these post-acquisition techniques, reliable raw CBED data that exhibit bonding information from copper are apparently more vital to accurate bonding determination. In the next chapter, details about TEM specimen preparation and CBED data collection condition will be given, targeting minimum uncertainties prior to and during acquisition.

Chapter 2. Background

Chapter 3. QCBED experiments

The whole process of QCBED experiments and analysis can be divided into four sections: specimen preparation, data collection, data preparation and QCBED refinement, which take place in the stated order. This chapter will discuss the methodologies associated with these four tasks.

3.1. Specimen preparation

In order to obtain high quality data, specimen preparations, such as annealing and electro-polishing, and careful TEM alignments were employed. The pure copper specimens used for this project were made from a 1-mm-thick Puratronic® copper sheet manufactured by *Alfa Aesar*, featuring 99.9999% purity. After coarse sandpapering the copper sheet down to about 500-micron thick with SiC 600 sandpapers, discs of 3 mm in diameter were then punched from the flat sheet. Excess thickness was left on purpose, and would be removed after the heat treatment described below, which may cause oxidation.

Heat treatments were applied to the punched discs with the aim to eliminate defects like dislocations, which might be generated during grinding and disc-punching or existed in the as-purchased copper sheet. The 500-micron thick discs were 1) heated up to and kept at 550 °C (the melting point of copper is 1085 °C [103]) for 4 hours, then 2) slowly cooled down (self-cooling in air) and kept at 250 °C for 2 hours to avoid thermal shock, and finally 3) self-cooling in air (for about an hour) down to room temperature to complete the annealing.

After annealing, the discs were hand-polished again with SiC 1200 sandpapers from 500 microns down to about 100 – 150 microns thick. The excess removal ensured the oxidation layer formed during annealing was completely ground off the specimens, as shiny surfaces with metallic lustre were observed on both sides of the discs. Sonication with distilled water was used to clean the hand-polished thin specimens.

The final thinning procedure employed electro-polishing to create electron transparent regions for TEM experiments. The thin copper discs were polished with a Struers TenuPol-5 twin-jet electro-polishing unit, using a 25% orthophosphoric acid, 25% ethanol and 50% distilled water solution. The electro-polishing conditions used were 10 V and approximately 150 mA, at 20 °C.

3.2. CBED data collection

The pure copper TEM specimens were single- or multi-grained crystalline discs, with diameters of 3 mm and edge thickness of about 100 – 150 microns. The crystal thicknesses, from where CBED patterns were collected, were between 500 to 3000 Å. In total 122 CBED patterns were collected and analysed in this project. 97 of these patterns were collected on a JEOL 2100F TEM (with a field

emission gun and a Gatan UltraScan 1000 CCD) at room temperature (293 K), at 200 kV, unfiltered. The other 25 patterns were collected on a JEOL 2200FS TEM (also with a field emission gun and a Gatan UltraScan 1000 CCD) by Yu-Tsun Shao, also at room temperature and 200 kV, with an omega-energy-filter. The JEOL 2100F TEM locates at the Monash Centre of Electron Microscopy, while the JEOL 2200FS TEM locates at the Frederick Seitz Materials Research Laboratory of University Illinois. Before collecting CBED patterns, the TEMs were carefully aligned. Apart from common alignments (e.g. gun and beam alignments), the diffraction focus was aligned with extra care by obtaining a sharp and symmetrical edge of a diffraction aperture pattern as described in [97]. The gain reference of CDD detectors were aligned at the level of 30,000 counts, which are typically the maximum intensities of the collected CBED patterns.

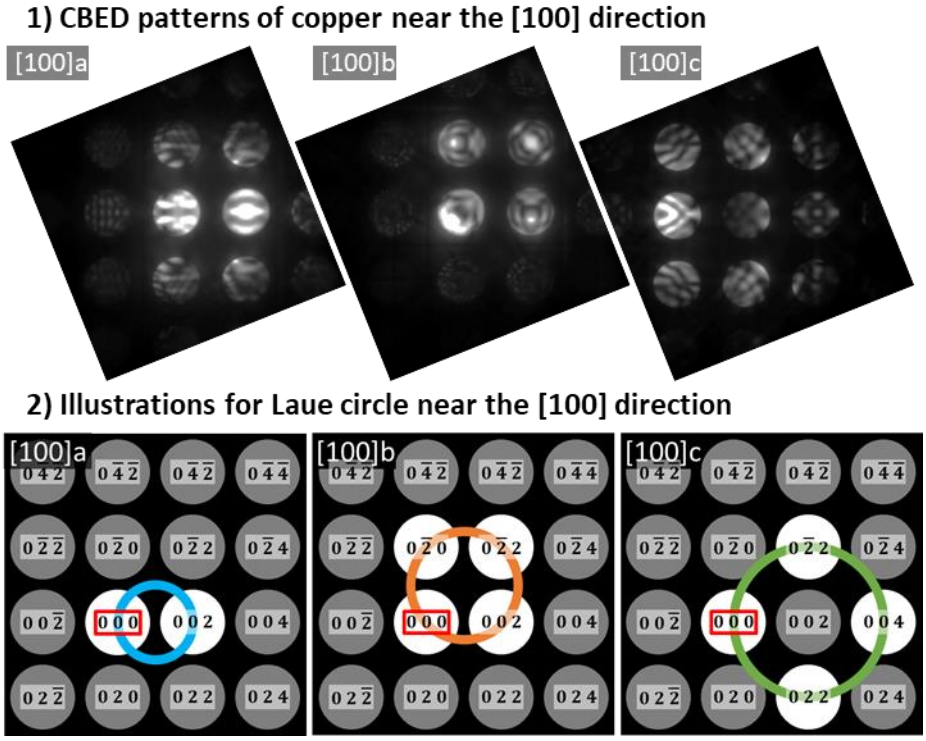


Figure 3.1. 1) Experimental CBED patterns of copper near the [100] direction and 2) the illustrations of their Laue circle geometries. In 2), the central reflections are outlined with red boxes, and reflection discs overlapped with the Laue circles are filled with a white shade, while other discs are filled with grey.

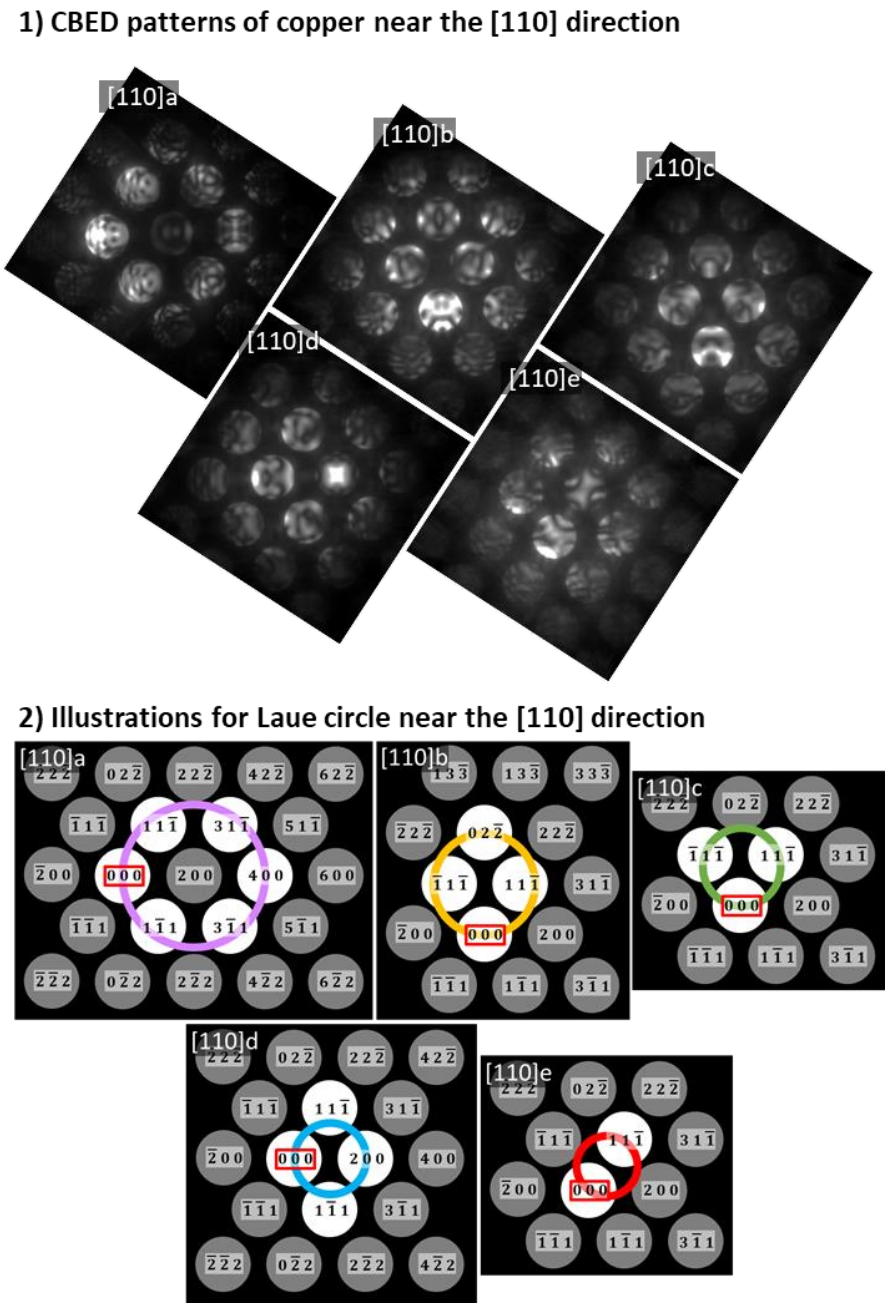


Figure 3.2. 1) Experimental CBED patterns of copper near the [110] direction and 2) the illustrations of their Laue circle geometries. In 2), the central reflections are outlined with red boxes, and reflection discs overlapped with the Laue circles are filled with a white shade, while other discs are filled with grey.

Because bonding occurs mostly in the low-order structure factors [4-7], CBED patterns were collected slightly off three major zone axes, <100>, <110> and <112>, from different crystal thicknesses. This enabled access to all the low-order structure factors with higher sensitivities from various crystal orientations. On the JEOL 2100F TEM, the following CBED patterns were collected: a) 48 patterns from near <100>, with three different Laue circle geometries as shown in Figure 3.1; b) 35 patterns near <110> with five different geometries as shown in Figure 3.2; and c) 14 patterns near <112> with

three different Laue-circle geometries as shown in Figure 3.3. On the JEOL 2200FS TEM, 25 energy-filtered patterns were collected near <110> of four geometries as shown in "[110]a-d" of Figure 3.2, using the scanning CBED technique [104] discussed in Section 3.3.

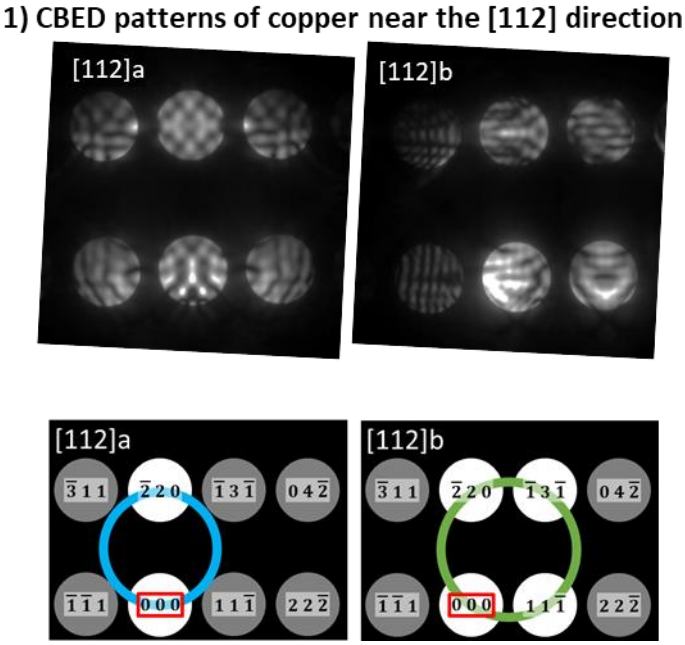


Figure 3.3. 1) Experimental CBED patterns of copper near the [112] direction and 2) the illustrations of their Laue circle geometries. In 2), the central reflections are outlined with red boxes, and reflection discs overlapped with the Laue circles are filled with a white shade, while other discs are filled with grey.

Additionally, it is known that patterns exhibit different levels of sensitivity to various structure factors with different Laue circle geometries [50]: a structure factor would have greater constraint in a pattern if 1) the coupling of its scattering vectors among the reflection discs crossed by the Laue circle (satisfying Bragg condition) occurs more frequently, 2) the coupling of reflections involves the central reflection disc and 3) the scattering vectors are shorter than that of other structure factors. Comparisons of sensitivities to various structure factors for each geometry regarding the above considerations are summarised in Table 3.1. Generally, patterns with smaller Laue circles, such as [100]a and [100]b in Figure 3.2 and [110]c, [110]e in Figure 3.1, are expected to be less sensitive to relatively higher-order structure factors like F_{222}, F_{311}, and F_{400}. In contrast, patterns with relatively large Laue circles, such as [110]a in Figure 3.2, should show relatively lower sensitivities to the low-order structure factors F_{111}, F_{200} and F_{220} compared to those with smaller Laure circles. However, the sensitivities to F_{111}, F_{200} and F_{220} are still higher than F_{222}, F_{311}, and F_{400} in these large-Laue-circle geometries.

Geometry		Sensitivities to F_g
[100]	a	$F_{200} > F_{220}$
	b	$F_{200} > F_{220}$
	c	$F_{220} \approx F_{200} > F_{400}$
[110]	a	$F_{111} > F_{311} > F_{400} \approx F_{200} > F_{220} > F_{222}$
	b	$F_{111} > F_{220} > F_{200}$
	c	$F_{111} > F_{220} \approx F_{200}$
	d	$F_{111} > F_{200} > F_{220}$
	e	$F_{111} > F_{200}$
[112]	a	$F_{111} > F_{220} > F_{311} \gg F_{222}$
	b	$F_{111} > F_{220} > F_{311} \gg F_{222}$

Table 3.1. Geometries of the collected CBED patterns and their sensitivities to the low-order structure factors. The geometries are denoted as shown in Figure 3.1, Figure 3.2 and Figure 3.3. Generally, geometries with smaller Laue circles are only sensitive to the lowest-order structure factors F_{111}, F_{200} and F_{220}, while geometries with larger Laue circles, like [100]c, [110]a, [112]a and [112]b, are also sensitive to the relatively higher-order structure factors.

3.3. Beam damage and scanning CBED

Because CBED experiments in this work utilise a highly focused beam in a field-emission gun TEM, beam damage may occur and destroy the perfect copper crystal during CBED pattern acquisition. For bulk copper, the electron sputtering damage threshold is about 147 keV [105]. To reduce beam damage, lower electron dose during acquisition is desired. Experiments on both the JEOL 2100F and JEOL 2200FS TEMs were conducted at 200kV. Different beam-time minimisation methods were used to reduce the total electron dose.

On the JEOL 2100F TEM, acquisitions were undertaken in the following fashion to minimise the dose and potential beam damage: 1) align desired Laue-circle geometry and focus specimen height (which might be time-consuming and causing beam damage), 2) slightly translate (typically a few nano-meters) the specimen away from the previous region with specimen tilting and Laue-circle geometry unchanged, 3) acquire a CBED pattern from the neighbouring region as soon as possible, and 4) check specimen height focus after acquisition (only patterns acquired with correct height-focus were kept). Although small drifting may persist soon after specimen stage translation, the diffraction pattern of neighbour regions from pure copper should be identical if the thickness and tilting of the crystal remain unchanged.

For CBED patterns collected on the JEOL 2200FS TEM, the scanning CBED technique [104] was used to minimise dose. Scanning CBED is a technique which acquires multiple CBED patterns across

a region on a specimen, with accurately programmed real-space distances and exposure time. Conventionally it is used to determine spatially varying characteristics for more complex materials [104]. In this work, scanning CBED was used because of its advantages in programmed acquisition. Compared to manual operation, scanning CBED can repeat the align-crystal-and-acquire-from-neighbour practice described above with almost no delay to maximise the exposure/beam time ratio, resulting in minimised beam damage.

3.4. Batch processing of QCBED

Because the QCBED experimental data in this project consists of a very large number of data sets (122 CBED patterns), QCBED data collection, preparation and processing were executed in a manner of batch processing, while featuring the angular differential QCBED technique. Computer scripts were developed by the author to achieve higher efficiency and removed human error associated with the repeated management of many data sets.

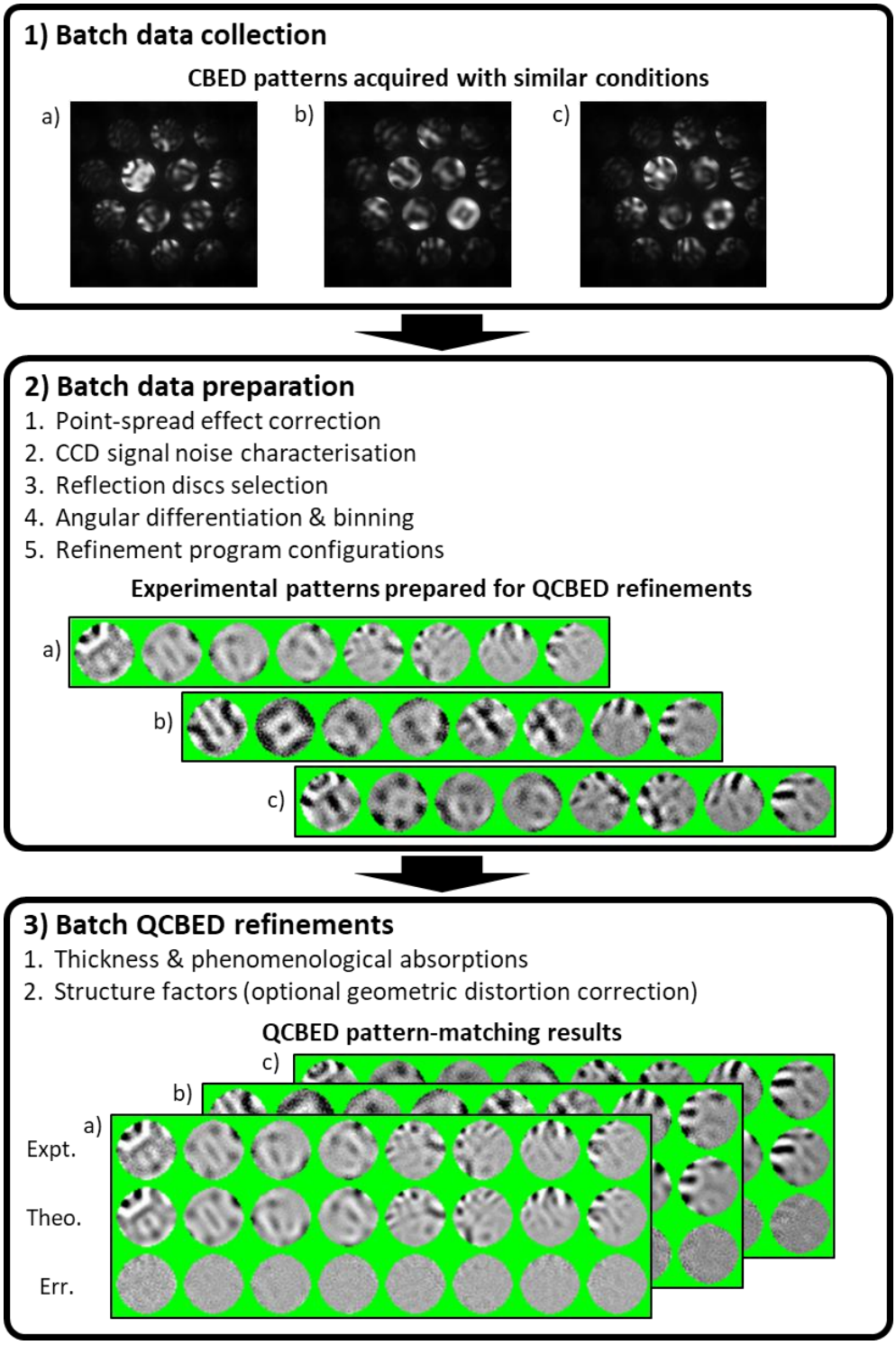


Figure 3.4. Flowchart for batch processing of QCBED copper bonding characterisation experiments: 1) Collection of similar CBED patterns (e.g. same Laue circle geometry) with constant TEM alignments, denoted as a), b) and c) 2) batch data preparation for QCBED refinements, and 3) batch QCBED pattern-matching refinements (the refinement step may be repeated multiple times).

3.4.1. Batch data collection

For convenience of batch processing, data sets exhibiting certain similarities were grouped together. For bonding characterisation of copper, as shown in Figure 3.4, a batch of patterns were collected with constant TEM alignments, different crystal thicknesses and same Laue circle geometry (small differences in actual crystal tilting may occur due to bending of the specimen). These grouping criteria are mainly due to two reasons: 1) significantly different crystal thicknesses for a given Laue circle geometry are required for uncertainty reduction and 2) patterns collected with fewer different conditions (except thickness) require less individual attention during data preparation and refinements, as the refinement outcomes are expected to be similar.

Additionally, during pattern collection of the same Laue circle geometry, the locations of reflection discs in CBED patterns may remain unchanged, if these patterns are collected from the same crystal grain and all TEM alignment conditions are kept constant. This may make "reflection discs selection" in the following data preparation procedures easier across patterns in one batch. However, patterns acquired from different regions on the detector may reduce systematic errors arising from collection, for instance, if some pixels of the detector are defected.

CBED patterns without the presence of any specimen (also called as "aperture patterns") should also be acquired with the same TEM alignments for a batch of specimen CBED patterns for point-spread effect characterisation and correction in the following data preparation stage [97].

3.4.2. Batch data preparation

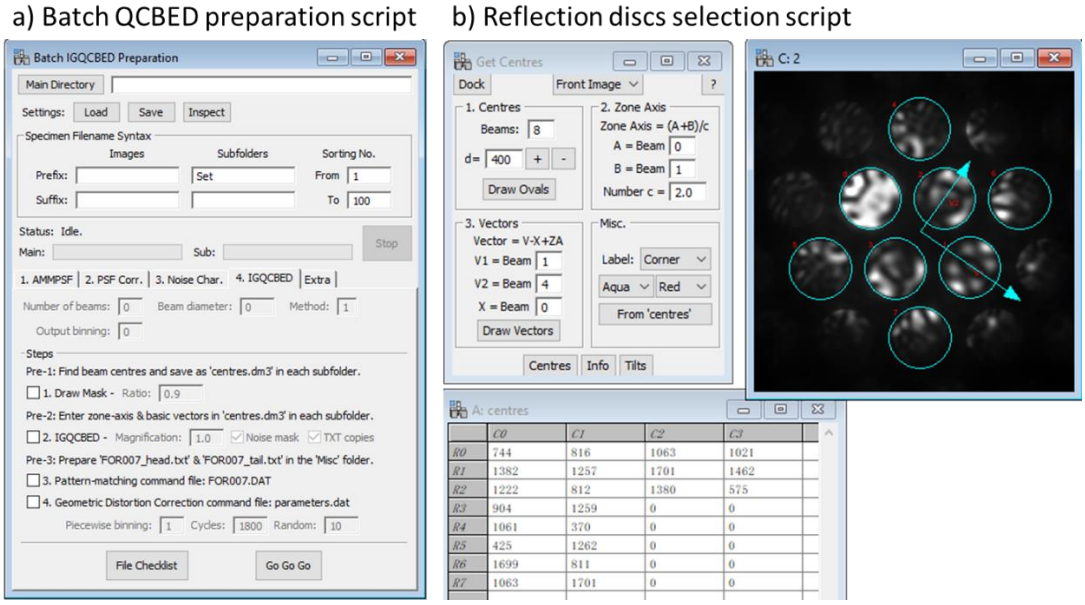


Figure 3.5. a) The graphical user-interface of the batch angular differential QCBED data preparation script *QCBED-Bprep* and b) the reflection discs selection script, both written for Gatan DigitalMicrograph in its scripting language. These scripts are designed for bulk QCBED data sets, in order to provide a guided data preparation process with minimised duplication labour.

Batch ADQCBED data preparation, as shown in Figure 3.4, consists of 1) point-spread effect correction [97], 2) CCD signal noise characterisation [99], 3) reflection discs selection, 4) angular differentiation and pattern binning [96], and 5) refinement program configuration files preparation. A Gatan DigitalMicrograph script, named as *QCBED-Bprep* (batch preparation), with a graphical user-interface aiming to guide users through the data preparation stage was developed for batch data preparation, as shown in Figure 3.5a. This script can process a batch of CBED pattern samples, with the same configurations/conditions for: 1) point-spread effect correction (samples should share the same point-spread function); 2) CCD signal noise characterisation (the noise characterisation is an *in-situ* process for individual pattern); 3) angular differentiation and pattern binning (the sizes of reflection discs and images should be constant across samples); and 4) refinement program parameters files. *QCBED-Bprep* consists of modified scripts from [97], [99] and [96]. A separate script, *Get Centres*, was developed to assist manual selection of reflection discs, as shown in Figure 3.5b. Because patterns in one batch have the same Laue circle geometry and experimental conditions, the coordinates of their reflection discs are similar (as shown in Figure 3.4) and thus batch reflection discs selection can be efficient.

3.4.3. Batch QCBED refinement

Different samples in a batch should have the same crystal structure and refinement configurations. For example, as shown in Figure 3.4, in copper bonding characterisation experiments only the crystal thicknesses and fine-tilting of the crystal might be different. Pattern-matching of a batch of data sets usually requires similar routes to yield relaxed and refined results, due to the similarities of the raw data and processing procedures (multiple batches with distinguishing experimental conditions should be therefore assessed to reduce systematic errors). Multiple cycles of pattern-matching refinements with various configurations will be needed to approach the final conclusion. Between the refinement cycles, editing of the configuration files, such as refinement switches of different parameters (e.g. thickness, phenomenological absorptions, structure factors) and number of refinement loops, is required to control the refinement program for each pattern. For a bulk sample size, such file editing requires a large amount of duplication labour which may cause human errors and lead to incorrect refinements.

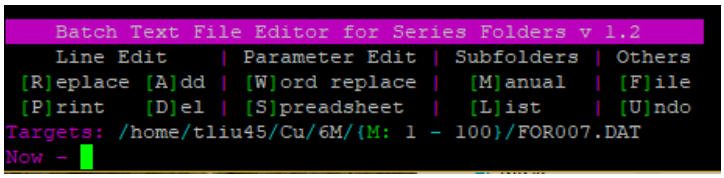


Figure 3.6. An interactive batch files editing tool for batch QCBED management written in Shell script.

An interactive command-line Shell script program, *bedit* (as shown in Figure 3.6), was developed for this batch file-editing purpose, since QCBED refinements of this project were mainly executed on computer clusters installed with Linux systems where Shell script is available. Assuming the files of different samples of a batch are placed under individual sub-folders within one parent folder, *bedit* can edit files common in every sub-folder. Line and individual-word editing with constant content and advanced spread-sheet-like content replacing are available in *bedit* (utilising *GNU sed* [106], *awk* [107] and *grep* [108]), aiming to assist unexperienced users of shell scripts for batch file-editing.

The detailed QCBED experimental methods used in this project, as well as a batched data processing approach, were described in this chapter. Accurate, precise and more efficient QCBED experiments and data preparation were demonstrated with these careful experiments and processing techniques. In the next chapter, a parallel-accelerated QCBED multislice pattern-matching program will be presented. As the most time-consuming stage for QCBED analysis, the efficiency of QCBED refinements has been accelerated by this new version of the QCBED program vastly.

Chapter 3. QCBED experiments

Chapter 4. Acceleration of QCBED multislice pattern-matching

The process of QCBED pattern-matching refinement to measure certain parameters, like structure factors, usually requires tens of thousands of iterations to search for the optimum solution. The time-consumption of such refinements depends on: 1) time consumed for each iteration, 2) the total number of iterations, which is controlled by the optimum-searching strategy/algorithm and the starting point of a refinement. In this chapter, a parallel-accelerated QCBED multislice program with optimised processing speed will be presented. The performance of a single threaded multislice pattern-matching program, *QCBEDMS*, will be first reviewed. Then its parallelised version, *QCBEDMS-PF* (Parallel FFT), will be introduced in detail, regarding its speed performance, further potential speed improvements and numerical precision.

In this chapter, various units of information storage will be used. They are: 1) basic units, bit and byte (B); 2) SI prefixed units, kilobyte (KB), megabyte (MB), gigabyte (GB) and terabyte (TB); and 3) Binary prefixed units, kibibyte (KiB), mebibyte (short for mega-binary MiB), gibibyte (GiB) and tebibyte (TiB). 1 byte equals 8 bits. The conversions for units with SI prefixes are: 1 TB = 10^3 GB = 10^6 MB = 10^9 KB = 10^{12} B. For binary prefixed units: 1 TiB = 1024 GiB = 1024^2 MiB = 1024^3 KiB = 1024^4 B. Binary prefixed storage units are preferred for sizes of hard drives, memories and memory caches, due to the designed sizes of memories and memory caches. However, SI prefixes units are preferred for data transfer rates, because the transfer rates are determined by speed frequencies of memories or hard drives, which are in SI-prefixed units – MHz or GHz.

4.1. Identification of time-consuming regions in the single threaded program

When applying parallel acceleration to a single threaded program, the first step is always to identify the computation-intense regions and whether they can be modified to parallel processes. In this section, the workflow and time-consumption of the single threaded multislice pattern-matching program *QCBEDMS* will be first examined, and then its potentials of parallelisation acceleration and other optimisations will be discussed.

As discussed in Section 2.3.1, the multislice approach uses the Huygens-Fresnel principle to calculate the propagation of the wave function between slices, which involves iterative convolution operations using the fast Fourier transform (FFT):

Eq. 4.1 $$\Psi_{t,n} = IFFT\left(FFT\left(\Psi_{t,n-1} \cdot G_n\right) \cdot FFT\left(P_{t,n}\right)\right),$$

where $\Psi_{t,n}$ and $P_{t,n}$ are the exit wave function and Fresnel propagator of the n^{th} slice at incident angle (tilt) t respectively, G_n is the phase grating of the slice, and FFT and $IFFT$ denote the forward and inverse FFT respectively. The phase gratings are identical for slices with constant atomic structures, while the Fresnel propagators are identical for slices with constant thickness.

A flowchart, illustrating the major procedures in an iteration of pattern-matching of the single-threaded program *QCBEDMS*, is presented in Figure 4.1. The Fresnel propagation (FP) approximation of a CBED pattern consisting of T incident tilts, with an N-slices thickness, is demonstrated in the flowchart. In *QCBEDMS*, the shape of the complex matrices $\Psi_{t,n}$, G_n and $P_{t,n}$, which represents the mesh of reflection beams being sampled during the FP, must be radix of 2 (the numbers of rows and columns may be different). Additionally, the inverse FFT of the last slice is omitted to yield intensities of diffraction patterns in reciprocal space. The process time of different procedures in *QCBEDMS*, as denoted in Figure 4.1, has the following relations: 1) $t_{Iter} \approx T \cdot t_{Tilt}$, 2) $t_{Tilt} \approx t_{PR} + t_{WPS}$, and 3) $t_{WPS} = N \cdot (t_{MG} + t_{MP} + t_{FFT} + t_{IFFT}) - t_{IFFT}$. On modern machines, the process time of operations not mentioned in the equation is subtle, therefore:

Eq. 4.2 $$t_{Iter} \approx T \cdot [t_{FP} + N \cdot (t_{MG} + t_{IFFT} + t_{MP} + t_{IFFT}) - t_{IFFT}],$$

where t_{FP}, t_{MG}, t_{MP}, t_{FFT} and t_{IFFT} are proportional to the number of included reflection beams (the size of the $\Psi_{t,n}$, G_n and $P_{t,n}$ matrices).

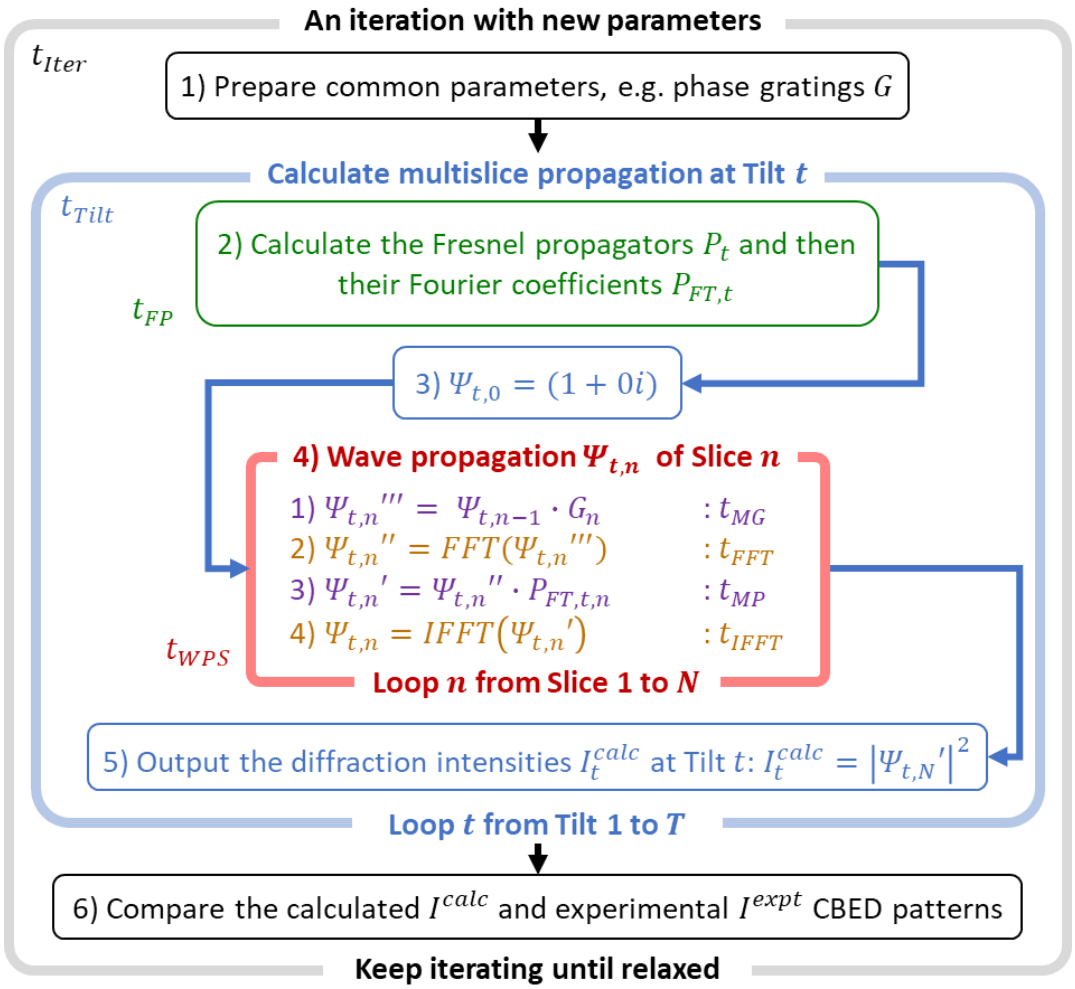


Figure 4.1. The flowchart of an iteration in pattern-matching of the single-threaded *QCBEDMS*. In the case of matching a pattern that consists of T incident angles (tilts) from a specimen with N slices, *QCBEDMS* will: 1) Calculate phase gratings G; 2) Calculate the Fresnel propagators P_t at Tilt t, and then their Fourier coefficients $P_{FT,t}$; 3) Prepare the initial wave $\Psi_{t,0}$; 4) Iterate the wave $\Psi_{t,n}$ propagating through the N-slices specimen; 5) Output the diffraction intensities I_t^{calc} at Tilt t; 6) Repeat Steps from 2) to 5) for all tilts to generate the intensities I^{calc} for selected reflections and compare them to the experimental intensities I^{expt}; Finally, keep iterating with new parameters until a matched pattern is found. FFT and $IFFT$ denote forward and inverse 2-D complex FFT respectively. The process times of major procedures/steps are denoted: one iteration as t_{Iter}, one tilt as t_{Tilt}, the calculations of P_t and $P_{FT,t}$ for one tilt as t_{FP}, the propagations of wave through all slices for one tilt as t_{WPS}, and the element-wise multiplications and FFT operations in the wave propagation for one slice as t_{MG} and t_{MP}, and t_{FFT} and t_{IFFT} respectively. The process times will have the following relation: a) $t_{Iter} \approx T \cdot t_{Tilt}$, b) $t_{Tilt} \approx t_{PR} + t_{WPS}$, and c) $t_{WPS} = N \cdot (t_{MG} + t_{FFT} + t_{MP} + t_{IFFT}) - t_{IFFT}$.

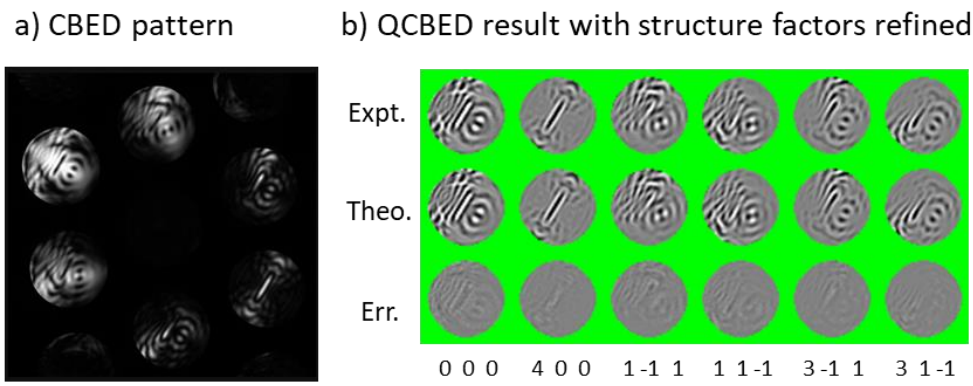


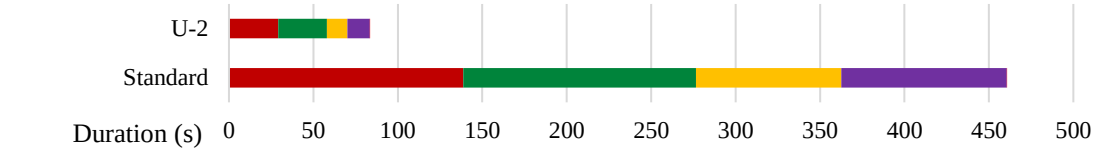
Figure 4.2. An energy-filtered CBED pattern of copper, labelled as "U-2", collected from a 259-nm-thick region, near [110] zone-axis: a) the original pattern, and b) the QCBED refinement patterns showing the experimental discs in "Expt.", calculated discs in "Theo." And their differences in "Err.". Because the pattern is the thickest in this project and hence consumes the longest processing time with the multislice approach, it is used to time and benchmark the performance of *QCBEDMS* and *QCBEDMS-PF*. For both programs, the "U-2" test will be timed and calculate the theoretical pattern only once (only one iteration), with angular differentiation.

Machine	**Desktop A**	**Desktop B**	**Desktop C**	**Laptop A**	**Cluster K80 Node**	**Cluster P100 Node**	**Cluster CPU Node**
CPU	**Intel Core i7-4770**	**Intel Core i7-4690**	**Intel Core i7-4770K**	**Intel Core i7-4960HQ**	**Intel Xeon E5-2680 v3 (×2)**	**Intel Xeon E5-2680 v4 (×2)**	**Intel Xeon Gold 6150 (×2)**
Core frequency (GHz)	3.4-3.9	3.5-3.9	3.5-3.9	2.6-3.8	3.3	2.4	2.7
Cores/Threads	4/8	4/4	4/8	4/8	12/12	14/14	18/18
Last level cache (MiB)	8	6	8	6	30	35	24.75
GPU	**Nvidia GeForce Titan Black**	**Nvidia GeForce GTX 760**	**AMD Radeon R9 270X**	**Nvidia GeForce GT 750M**	**Nvidia Tesla K80 (single chip) (×8)**	**Nvidia Tesla P100-16GB (×2)**	**N/A**
Double (single) precision GFLOPS	1707 (5121)	94 (2258)	168 (2688)	30 (723)	1456 (4368)	4670 (9340)	-
Max memory bandwidth (GB/s)	336	192	179.2	80	240	732	-
Stream multiprocessor (SMX)	15	6	20	2	13	56	-
Last level cache (KiB)	1536	512	512	256	1536	4096	-
Platform	Ubuntu 18.04	Ubuntu 16.04	Windows 10	macOS 10.13.6	CentOS Linux 7	CentOS Linux 7	CentOS Linux 7
Intel compilers' optimisation flags	Linux & macOS: -O3 -fp-model fast=2 -march=core-avx2 -ipo; Windows: /O3 /fp:fast=2 /Qxcore-avx2						

Table 4.1. Specifications of machines used to benchmark *QCBEDMS* and *QCBEDMS-PF*. Three personal desktops, one laptop and three cluster computer nodes were used.

An energy-filtered CBED pattern of copper collected near the [110] zone-axis, labelled as "U-2" and shown in Figure 4.2, was used to test the actual process time of *QCBEDMS*, because its specimen

thickness was measured to be 259 nm – the thickest in this project. The U-2 Test used a 32×16 mesh for reflections sampling in X and Y directions respectively (i.e. totally 512 reflection beams included), 2029 slices for thickness and 2500 tilts for pattern resolution (the reflection discs were binned to 50 × 50 pixels, and *QCBEDMS* calculated all 50 × 50 tilts including those outside the aperture). Only one iteration of pattern-matching was timed in the test. Beside the U-2 Test, a "Standard Test", which calculated a pattern with a 64×64-reflections mesh, 1000 slices and 2500 tilts, is used to test the process time when more reflection beams are included. The specifications of machines used to test the programs are listed in Table 4.1.



	Standard	U-2
Fresnel propagator	0.32	0.03
FFT	138.46	29.17
IFFT	137.67	28.79
MG	86.16	12.18
MP	97.99	13.28
Others	0.26	0.16

Figure 4.3. Process times of *QCBEDMS* running the U-2 and Standard Tests. The single-threaded program was executed and timed on Desktop A. The U-2 Test simulates the thickest (about 259 nm) copper CBED pattern in this project, using a 32×16 reflections-mesh, 2500 tilts and 2029 slices. While the Standard Test performs a 1000-slices calculation with a 64×64 reflections-mesh and 2500 tilts. The process times of different steps are: 1) *Fresnel propagator* for calculations related to Fresnel propagators and their Fourier coefficients; 2) *FFT* and *IFFT* for forward and inverse FFT operations respectively; 3) *MG* and *MP* for element-wise complex array multiplications with the phase gratings and Fresnel propagators' Fourier coefficients respectively during wave propagation; and 4) *Others* for all other operations. The total process times were 83.61/460.85 seconds for the U-2/Standard Tests respectively. The process times of the FFT and IFFT operations tend to be very similar and both occupy about 35%/30% of the total process time in the U-2/Standard Tests. The multiplication operations with the Fresnel propagators' Fourier coefficients take slightly longer than the multiplications with the phase gratings and occupy about 16% and 15%/21% and 19% in the U-2/Standard Tests respectively.

The time-consumption of the major operations in *QCBEDMS* was timed on Desktop A, as shown in Figure 4.3. The total process times of the single-threaded program were 83.61 and 460.85 seconds for the U-2 and Standard Tests respectively. The process times of the FFT and IFFT operations tend to be equal and both occupy about 35%/30% of the total process time in the U-2/Standard Tests. The multiplication operations with the Fresnel propagators' Fourier coefficients take slightly longer than

the multiplications with the phase gratings and occupy about 16% and 15%/21% and 19% in the U-2/Standard Tests. It is obvious that in these tests, the wave propagation (including FFT operations and matrix multiplications, excluding calculations of the Fresnel propagators) occupies almost 100% of the total process time. However, because patterns of pure copper, a homogeneous and simple crystalline material, are calculated in the U-2 and Standard Tests, only two Fresnel propagators are needed for each incident tilt in *QCBEDMS*. In contrast, for specimens requiring more Fresnel propagators, the corresponding process times will be increased linearly.

Given that the FP (which consists of Fresnel propagator calculations and the following wave propagation) of different incident tilts are independent to other tilts, as shown in Figure 4.1, parallelisation can be applied to the looping of multiple incident tilts, which occupies almost 100% of the total process time. As *QCBEDMS-PF* is designed for QCBED analysis, neither overlapping reflection discs are expected as the experimental input, nor spatial coherence is included to allow the calculations of theoretical patterns with overlapping discs. This ensures there is no exception to the independence and parallelisability of the calculations for different incident tilts. Moreover, because the process times are mostly spent on the FFT and matrix multiplications during wave propagation, these steps should be optimised (on top of the per-tilt-parallelisation) to result in significant improvements in processing speed.

4.2. Different architectures and parallelisation for CPU and GPU

As indicated in the previous section, parallelising the calculations of every incident tilt in a CBED pattern is possible, while a nearly 100% parallelisation efficiency can be achieved as other operations outside the per-tilt calculations cost almost no process time. For the detailed parallelisation architecture, there could be two different routes to parallelising the per-tilt calculations. The first route, namely “tilt-oriented”, would be parallelised as:

1. each processing core collects an incident tilt from the pattern,
2. finish this tilt’s Fresnel propagator calculations and the following wave propagation, and then
3. move on to another tilt, until all tilts are finished.

The other route, namely “operation-oriented”, would be carried out as:

1. each processing core collects an incident tilt from the pattern and finishes the calculations of its Fresnel propagators, and then move on to another tilt until Fresnel propagators of all tilts are finished,
2. each processing core collects an incident tilt from the pattern and finishes the calculations of a step in the wave propagation, and then move on to another tilt until this step of all tilts are finished, and
3. follow this path until all steps of all tilts are finished.

To summarise, the tilt-oriented parallelisation would have each processing core finish all the operations of each incident tilt, while the operation-oriented route would have each operation of all tilts finished one by one. The efficiencies of the two routes will be different, and should be chosen depending on the architectures of CPU and GPU.

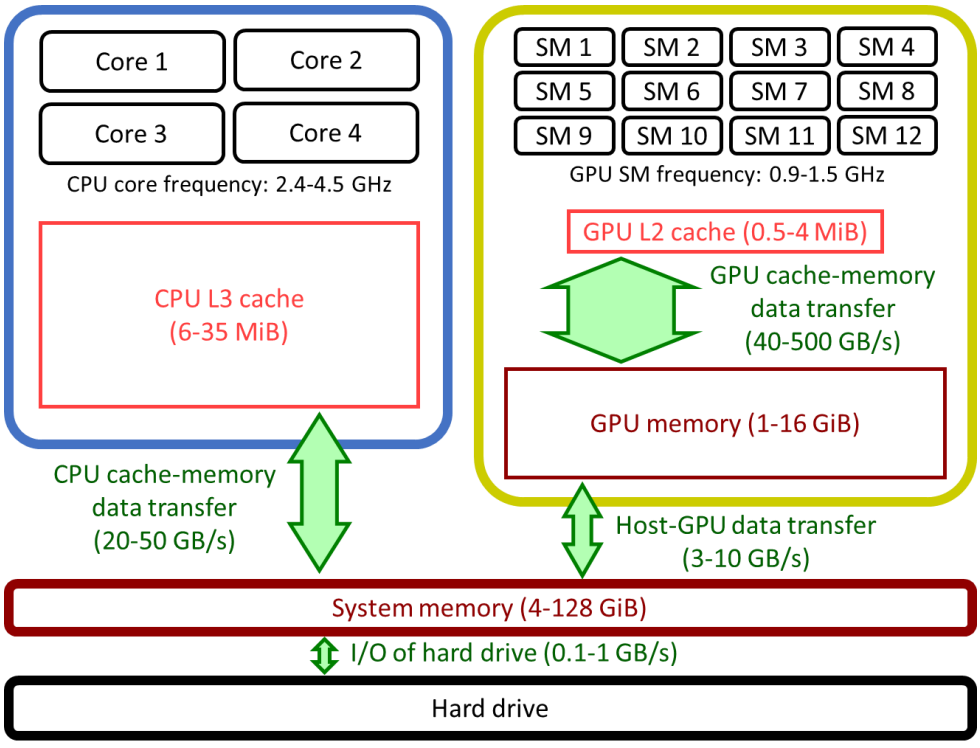


Figure 4.4. Schematic of CPU-GPU computing architecture. There are three major components in modern processing units (CPU/GPU): 1) processing cores, where computations are operated, 2) memories, where values of parameters are stored, and 3) caches, featuring as temporary memories with super-fast speed but small capacities.

As shown in Figure 4.4, there are three major components in both CPU and GPU: 1) the processing cores, which operate the computations, 2) the memories, which store the values of parameters, and 3) the caches, which may be regarded as temporary memories with super-fast speed but small capacities. For CPU, the multiple processing cores have very high processing speed (which is usually proportional

to their core frequencies). For GPU, there are more processing cores (or called streaming multiprocessors, SM, in GPU). The processing speed of a SM in GPU is much slower than that of a CPU core. However, a GPU SM can process hundreds of identical operations (e.g. element-wise addition of two matrices) in parallel, while a CPU core has to proceed in serial. During processing, both CPU and GPU prefer to temporarily store information in lower level caches, which are located separately in every CPU core/GPU SM (not shown in the schematic). Low-level caches, Levels 1 (L1) & 2 (L2) in CPU and L1 in GPU, have almost no transfer latency but very small capacities (about 256 KB for CPU, 64 KB for GPU). When larger capacities are required, the last-level caches will be used, which 1) are shared by all the cores/SMs, 2) are named as Level 3 (L3) and L2 caches for CPU and GPU respectively, and 3) have slightly larger capacities and super-fast transfer rates (but slower than lower-level caches). When even the last-level caches are not large enough, temporary values will be pushed to the memories, which have significantly reduced transfer rate compared to caches. For CPU computation, transfers between cache and memory are essential only when data is not already in cache or writing data to hard drive. For GPU, however, implicit data syncing between cache and memory also occurs before and after every GPU kernel, therefore extra kernel execution will cause extra memory caching operations. Due to different architecture designs, CPU's are more efficient for serial computations, while GPU's are optimised for matrix operation.

4.2.1. CPU parallelisation: tilt-oriented parallelisation

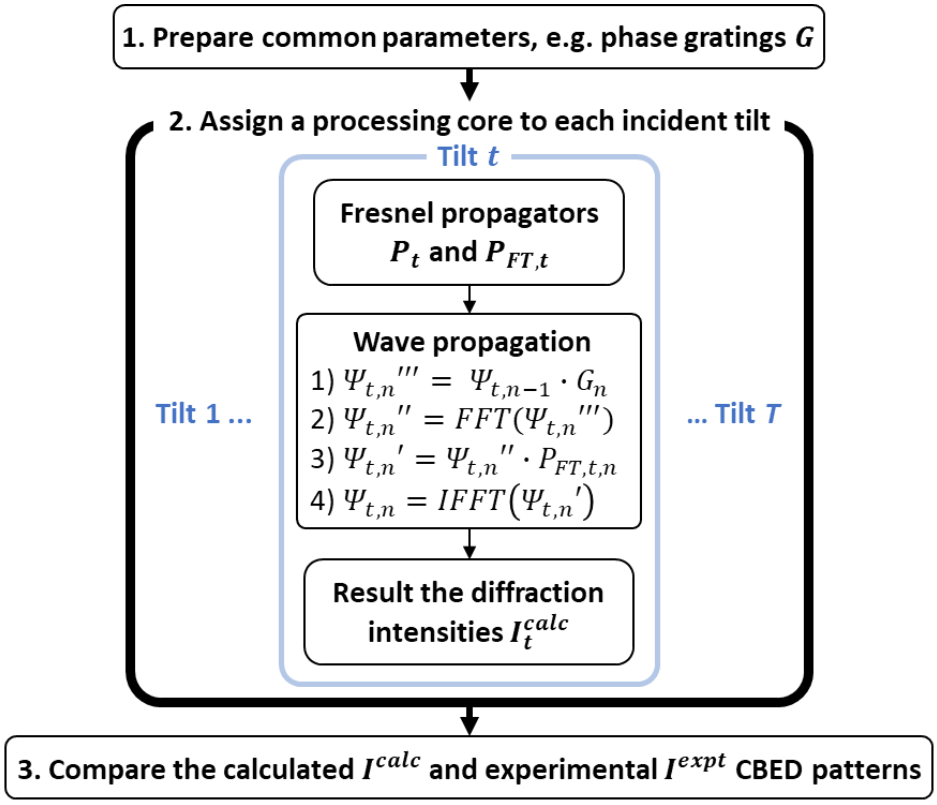


Figure 4.5. Flowchart of tilt-oriented parallelisation for CPU in *QCBEDMS-PF*. According to the micro architecture of CPU, the tilt-oriented parallelisation is employed: 1. Prepare common parameters (including calculations of the phase gratings G) in the host thread; 2. Each processing core collect an incident tilt from the pattern, finish this tilt's Fresnel propagator calculations and the following wave propagation, then move on to another tilt, until all tilts are finished, and; 3. Finally, compare the calculated I^{calc} and experimental I^{expt} CBED patterns on the host thread.

The tilt-oriented parallelisation, which assigns all calculates of each tilt to a processing core (as shown in Figure 4.5), is preferable for CPU. In fact, even for single-threaded multislice program, like *QCBEDMS*, "finishing calculations tilt-by-tilt" will also be much faster than "finishing one operation of all tilts then proceeding to the next operation of all tilts". This is due to the "memory caching" process required for every tilt: all matrices (e.g. the wave Ψ and Fresnel propagators P) unique to a tilt have to be loaded from the memory to the caches for the processing core to perform operations for this tilt, and then the outcomes of this tilt will be put back to the memory from the caches when the calculations are finished. The tilt-oriented approach will make the processing core focus on the same tilt, which minimises the number of caching operations between each step of multislice propagation. This is vital to the speed performance of multislice programs on CPU, where the caching speed is generally slow comparing to the processing speed. This is also true for the multi-threaded *QCBEDMS-PF*, where all processing cores focus on their associated tilts, respectively, before moving to the rest

of tilts. For GPU, however, the caching speed, caching behaviours and matrix operation speed are different, where the operation-oriented parallelisation is preferred.

4.2.2. GPU parallelisation: operation-oriented parallelisation

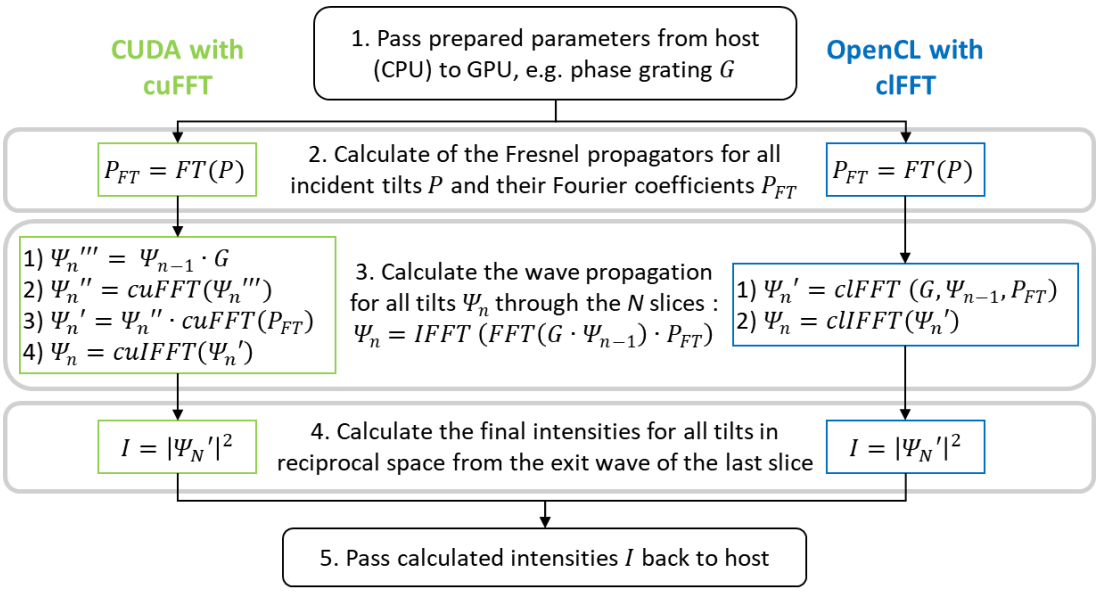
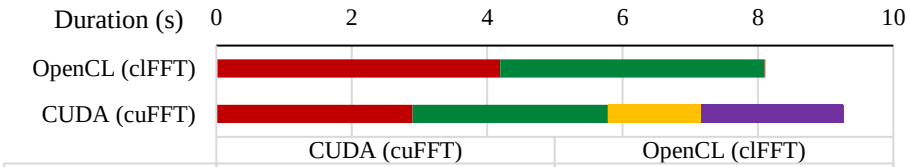


Figure 4.6. Flowchart of OpenCL and CUDA implementations in *QCBEDMS-PF*. By using the clFFT and cuFFT libraries respectively, the accelerations of OpenCL and CUDA have similar structures: 1. Passing calculated parameters, e.g. the phase gratings G, from the main program (on CPU) to the GPU device; 2. Calculate the Fourier coefficients of the Fresnel propagators, $FFT(P)$, required by the following multislice propagation for all the incident tilts; 3. Calculate the wave functions, Ψ_n, propagating through the N slices of the specimens for all the tilts; 4. Calculate the diffraction intensities I from the exit wave of the last slice Ψ_N, and; 5. Finally, pass the calculated intensities back to the main program. Because the clFFT library supports advanced custom pre/post-operations of FFT (called as FFT Callback), the FFT operations and their prior and posterior element-wise array multiplications can be combined into one step (denoted as "$clFFT\,(G, \Psi_{n-1}, P_{FT})$"), resulting in only two steps for calculations of the wave in one slice. In contrast, because cuFFT does not support this kind of FFT Callback, four independent steps are required to calculate the wave for each slice.

In order to gain better compatibility on various devices, both OpenCL and CUDA, the two most common low-level general-purpose GPU (GPGPU) application program interfaces (APIs), are implemented in *QCBEDMS-PF* for GPU acceleration. Both the calculations of the Fresnel propagators and multislice propagation have been parallelised, as shown in Figure 4.6. The clFFT and cuFFT libraries were used for the FFT operations in OpenCL and CUDA respectively. Overall, the workflows, which employ the operation-oriented parallelisation, are similar for both GPU frameworks: 1. Start by passing calculated parameters from the main program on CPU to the GPU device; 2. Calculate the Fourier coefficients of the Fresnel propagators, which is required by the following multislice

propagation for all the incident tilts; 3. Calculate the wave functions, which propagate through all the slices of the specimens for all the tilts; 4. Calculate the diffraction intensities from the exit wave of the last slice, and; 5. Finally, pass the calculated intensities back to the main program on CPU. Because the clFFT library supports advanced custom pre/post-operations of FFT (termed as "FFT Callback"), the FFT operations and their prior and posterior element-wise array multiplications can be combined into one step. In contrast, because cuFFT does not support advanced FFT Callbacks which involve extra matrices, four independent steps are required to for the wave calculations of each slice. Mostly due to the differences of the FFT Callback, the speed performances of the two GPU frame are different.



	CUDA (cuFFT)	OpenCL (clFFT)
Fresnel propagator	0.010	0.018
FFT	2.888	4.177
IFFT	2.886	3.906
MG	1.383	-
MP	2.093	-
Others	0.010	0.010

Figure 4.7. Time-consumption of *QCBEDMS-PF* with different GPU accelerations running the Standard Test on Desktop A.

As shown in Figure 4.7, *QCBEDMS-PF* was timed for the Standard Test using the OpenCL and CUDA accelerations respectively. The testing platform was Desktop A, which is equipped with a Nvidia GTX Titan Black GPU. OpenCL is about 15% faster than CUDA in this test. This may be unexpected, as CUDA and its cuFFT library are optimised for Nvidia devices, while the clFFT library was originally motivated by non-Nvidia devices that only supports OpenCL. However, the mentioned combined steps in the multislice propagation process are the key components that make the OpenCL implement faster than CUDA. On GPU, the major bottleneck of the speed performance of *QCBEDMS-PF* is the "memory caching" process. The term "memory caching" is defined as the process of data moving between a device's global memory and its caches.

GPU framework	OpenCL			CUDA		
Memory object	**Phase grating**	**Fresnel propagators**	**Wave function**	**Phase grating**	**Fresnel propagators**	**Wave function**
Size per slice (MiB)	0.06	156.25	156.25	0.06	156.25	156.25
Caching operations of GPU kernel						
MG	*N/A*			Read	-	Read & write
FFT (2 kernels)	Read by 1st kernel	Read by 2nd kernel	Read & write by both kernels	-	-	Read & write by both kernels
MP	*N/A*			-	Read	Read & write
FFT (2 kernels)	-	-	Read & write by both kernels	-	-	Read & write by both kernels
Least-number of total caching operations						
MG	*N/A*			1,000	-	2,000
FFT	1,000	1,000	4,000	-	-	4,000
MP	*N/A*			-	1,000	2,000
IFFT	-	-	3,996	-	-	3,996
Total least-caching size (GiB)						
MG	*N/A*			0.06	-	305.18
FFT	0.06	152.59	610.35	-	-	610.35
MP	*N/A*			-	152.59	305.18
IFFT	-	-	609.74	-	-	609.74
Sum in GiB (GB)	1372.74 (1473.97)			1983.09 (2129.33)		

Table 4.2. Memory caching operations of GPU kernels in *QCBEDMS-PF*, in the case of a (double-precision) Standard Test. In the Standard Test, 1) the phase gratings, Fresnel propagators and wave functions are 64-by-64 double-precision complex matrices, and a double-precision complex number occupies 16 bytes; 2) the operations will be repeated for 1000 slices; and 3) there are 2500 incident tilts. Because all tilts are processed in parallel, the memory objects of the Fresnel propagators and wave functions both consist of 2500 64-by-64 matrices. MG stands for the multiplications of the phase grating and wave function matrices, while MP for that of the Fresnel propagators and wave functions. FFT/IFFT operations by both OpenCL's clFFT and CUDA's cuFFT libraries consist of two kernels, therefore read/write operations of the wave functions are repeated by the two kernels as well. For OpenCL, because both the MG and MP operations are combined into the FFT operations, no extra kernels are required. While for CUDA, since separated kernels are required for MG and MP, the wave functions will be read and write once, resulting in two more caching operations. As the IFFT operation of the last slice is omitted in *QCBEDMS-PF*, the total number of caching operations of IFFT is always four less than that of FFT. If the last level cache of a GPU is capable of containing all memory objects throughout the four operations (MG, FFT, MP and then IFFT) of each slice, the least-number of total caching can be achieved. Otherwise, the actual caching size will be larger than the least-caching size, causing longer process time. Please note that, the numbers of caching operations are constant for samples with single or multiple phase grating and Fresnel propagator slices.

The memory caching behaviour of GPU is slightly different to that of CPU: in addition to the caching required by the loading of new variables and saving of the outcomes, implicit caching is forced between GPU "kernels (chunks of operations)". In *QCBEDMS-PF*, individual steps demonstrated in Figure 4.6 represent different GPU kernels that cannot be combined in OpenCL and CUDA, receptively. Moreover, 2D FFT operations require two separated kernels for both clFFT and cuFFT. The memory caching operations in *QCBEDMS-PF* on GPU, in the case of the Standard Test, is summarised in Table 4.2. Because the clFFT library supports advanced FFT Callbacks, matrix multiplications can be combined into the FFT operations on the OpenCL framework. Such practices lead to fewer kernels, and thus less memory caching when OpenCL is used by *QCBEDMS-PF*. The overall caching size of OpenCL is about 30% smaller than that of CUDA in *QCBEDMS-PF*.

GPU	Nvidia GeForce Titan Black	Nvidia GeForce GTX 760	AMD Radeon R9 270X	Nvidia GeForce GT 750M	Nvidia Tesla K80 (single chip)	Nvidia Tesla P100-16GB
Processing power (GFLOPS)	1707	94	168	30	1456	4670
Memory bandwidth (GB/s)	336	192	179.2	80	240	732
OpenCL (least-caching size: 1473.97 GB)						
Measured bandwidth (GB/s)	237.49	150.00	144.36	58.49	-	539.68
Least-caching time (s)	6.21	9.83	10.21	25.20	-	2.73
Total process time (s)	8.08	22.98	22.10	110.92	-	3.26
CUDA (least-caching size: 2129.33 GB)						
Measured bandwidth (GB/s)	239.16	149.92	-	52.99	171.50	540.60
Least-caching time (s)	8.90	14.20	-	40.18	12.42	3.94
Total process time (s)	9.18	25.76	-	96.27	12.91	3.99

Table 4.3. Speed performance of *QCBEDMS-PF* running the Standard Test on various GPUs, compared with the least-caching time. Although the GPUs were located on different machines, because the majority of computations, i.e. the FP, were executed by the GPUs, the variety of systems should have very small impacts.

The speed performances of various GPUs are summarised in Table 4.3. Typically, a 15% reduction of the total process time can be achieved with the decreased overall caching size, comparing the speeds of OpenCL and CUDA where both frameworks are available. The Nvidia GeForce GT 750M is an exception. This is probably because the GPU has a too-small L2 cache size (only 256 KiB). The OpenCL framework in *QCBEDMS-PF*, with the FFT Callbacks, needs a larger cache size to contain all required variables in the combined kernels. If the L2 cache is too small, the least-caching size stated in Table 4.2 and the least-caching time in Table 4.3 become impossible, resulting in extra caching inside kernels and thus significantly longer total process time. Comparing the speeds between different

GPUs, it is shown that the caching operations have occupied a significant proportion of the total process time, especially for GPUs with greater double-precision performance (e.g. Nvidia GeForce Titan Black, Nvidia Tesla K80 and Nvidia Tesla P100-16GB). For GPU products targeting the entertainment market with reduced (double-precision) processing power, like the Nvidia GeForce GTX 760, AMD Radeon R9 270X and Nvidia GeForce GT 750M, considerably more time was consumed by other processes than the caching.

4.3. Other optimisations

As indicated in the previous section, most of the computation time in multislice calculation is spent on the multislice propagation process and the calculation of Fresnel propagators. Especially for applications to specimens with simple crystal structure, such as copper, the multislice propagation process occupies nearly 100% of the total calculation time. In this section, optimisation of these computation intense regions will be discussed.

4.3.1. Fast Fourier-transform

As shown in Figure 4.3, the FP, consisting of FFT and element-wise multiplication of the complex matrices, occupies almost 100% of the processing time in *QCBEDMS*, since these operations are repeated as many times as the number of slices of a specimen in every iteration. Therefore, modifications to the FFT and the multiplications may result in maximum reduction to the processing time of *QCBEDMS*, as these operations are repeated as many times as the number of slices of a specimen in every iteration. Inspired by previous studies [109-112] using different FFT algorithms (including FFT libraries utilising GPU acceleration) to accelerate the FP, a modified version of *QCBEDMS* featuring various FFT libraries and GPU acceleration has been developed and named *QCBEDMS-PF* (parallel FFT). In the original version of *QCBEDMS*, the FORTRAN version of the FFTSG library was used for FFT operation. In *QCBEDMS-PF*, FFTW, clFFT and cuFFT libraries, which are written in C/C++ languages, are implemented as alternative options for FFT operations. FFTW, generally recognised as the fastest FFT library on CPU, is implemented for machines without a GPU. clFFT is a GPU FFT library written in OpenCL language, while cuFFT is a GPU FFT library written in CUDA language. The performances of *QCBEDMS-PF* using various FFT libraries are benchmarked with the U-2 Test and summarised in Figure 4.8. Generally, when using FFT libraries that utilise GPU acceleration, *QCBEDMS-PF* can reduce the process time by 50% compared to the faster CPU FFT library, FFTW.

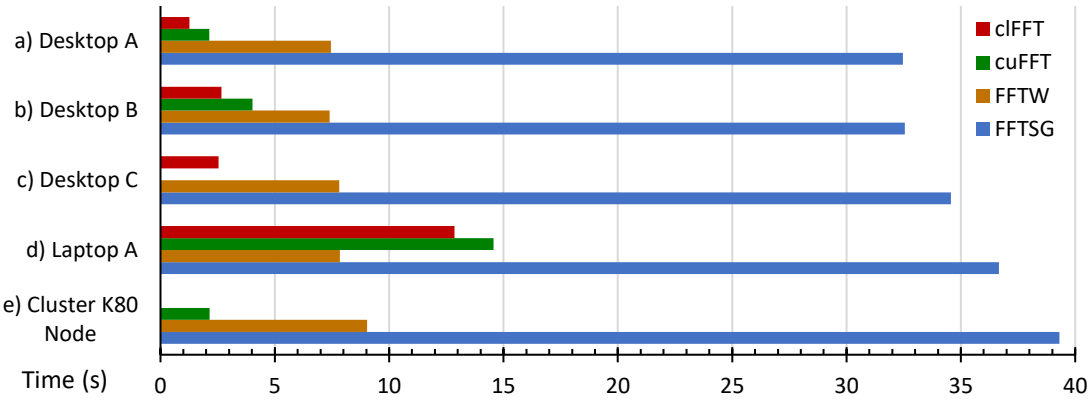


Figure 4.8. Results of the U-2 process time benchmarks by various FFT libraries in *QCBEDMS-PF* (shorter process times represent faster processing speed). For clFFT and cuFFT, 1 GPU and 1 CPU were used, while for FFTW and FFTSG, 1 CPU was used. Double floating-point precision was used for all FFT libraries. cuFFT was not available on Desktop C because an AMD GPU was used. clFFT was not available on the Cluster K80 Node because OpenCL was not installed. Generally, when using FFT libraries that utilise GPU acceleration, *QCBEDMS-PF* can reduce the process time by 50% compared to the faster CPU FFT library, FFTW. However, for GPUs with low double precision floating-point performance (e.g. the case in d) Laptop A), the FFTW can achieve better performance.

4.3.2. Array operations

Although the underneath math operations are constant for the multislice formulation, the realisation of the calculations by programming can vary: different programming languages and different programming syntaxes, which may lead to different processing speed. In this section, the speed performance of 1) various array types, 2) variable-passing methods (VPM) and 3) array operation syntaxes (AOS) of Fortran and C/C++ will be discussed, which mostly benefits the computational efficiency on CPU. Beside the FFT operations, as shown in Figure 4.3, the multiplication of complex arrays inside the FP are the other time-consuming components. Therefore optimisations and tests were focused on these regions. Because the original program, *QCBEDMS*, was written in Fortran, and it is generally accepted that Fortran has faster operations than C/C++ (while both Fortran and C/C++ are faster than other programming languages), discussions of Fortran will be first given.

Different array types and syntaxes in the scripting codes will affect the optimisations of compilers, which will lead to changes in processing speeds of the final executable of the program. Therefore choosing the appropriated combination of array type and syntax will benefit the overall speed performance. As shown in Figure 4.9, various combinations of array types, variable passing and array operation syntaxes were tested by different compilers for Fortran. Only array types that allow varying numbers of elements were included, because the program aims for different QCBED configurations

which will involve varying matrix sizes across different experiments. Overall, the combination of 1) the "fftw_complex" array type (provided by the FFTW library), 2) passing as function arguments (called from C++) and 3) the syntax $A(1{:}\,mt) = A(1{:}\,mt) \cdot B(1{:}\,mt)$ (where mt is the index of the last element) showed the best performance across various compilers.

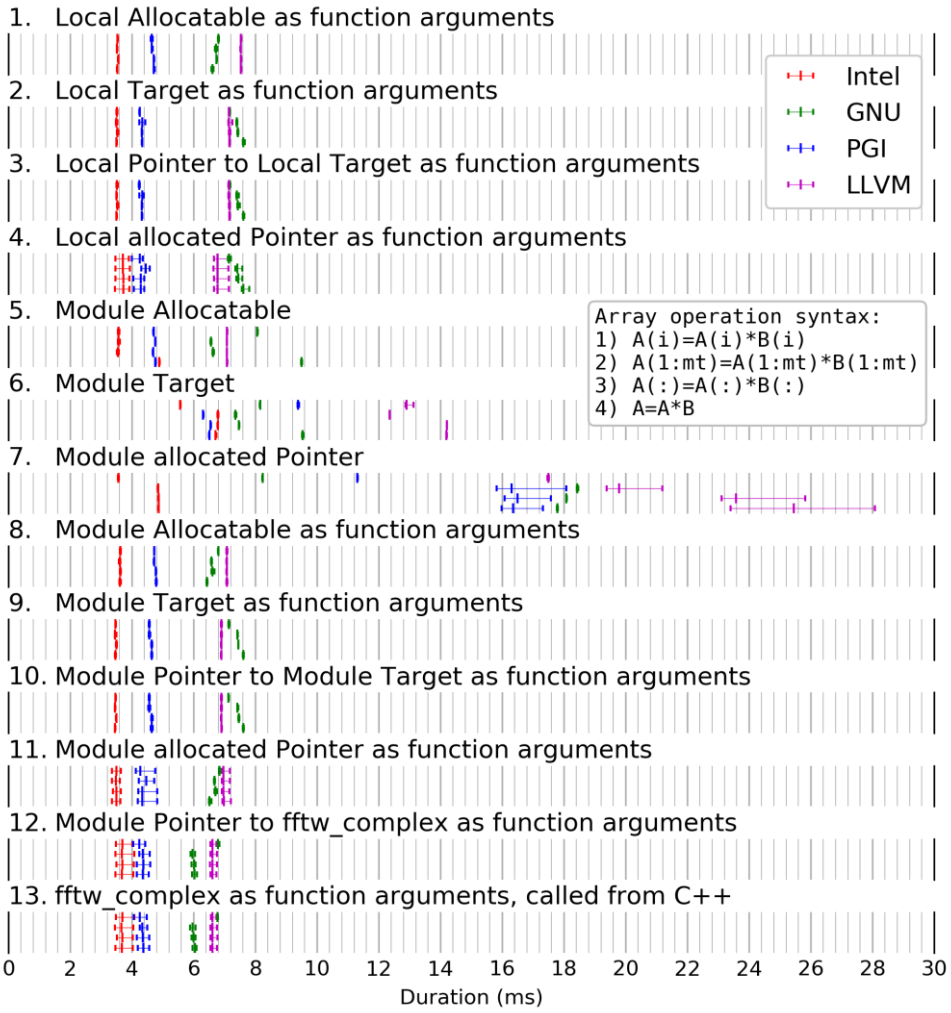


Figure 4.9. Speed of array operations with various array types in Fortran. Double precision complex array element-wise multiplications of $A = A \cdot B$ then $A = A \cdot C$ were repeated 1000 times, single-threaded, and timed to simulate the time-consumption of array multiplications in multislice propagation. A, B and C are complex arrays with size of 64×64. Ten loops of all tests were timed and plotted with the average, minimum and maximum values. A combination of various array types, VPM and AOS were tested as annotated. There are four AOS's for every array type and plotted in the same sequence as annotated, which all indicate the element-wise multiplication of all elements between A, B and C (mt stands for the index of the last element). Four sets of Fortran and C++ compilers were tested: 1) Intel compilers with flags "-fast -march=core-avx2", 2) GNU compilers with "-Ofast -march=native", 3) PGI compilers with "-fast", and 4) LLVM (flang and clang++) compilers with "-Ofast -march=native". The testing platform was a 64-bit Linux system, with a 4-cores CPU operating at 3.5 GHz fixed frequency. Overall, the "fftw_complex" array type, provided by the FFTW library, showed the best performance across various compilers.

The speed of FFT operations by FFTW, using different array allocation methods, is also tested and shown in Figure 4.10, as FFTW will be the preferred FFT library for CPU regarding the overall performance. Similarly to the test in Figure 4.9, the "fftw_complex" array type, which is provided and suggested by the FFTW library, showed the best speed performance. Because the FTTW library is written in C, C/C++ implementations of the library are involved in the *QCBEDMS-PF* program (by C/Fortran mixed languages programming). The speeds of array operations in C/C++ were also tested and shown in Figure 4.11, to determine whether a pure C/C++ script would result comparable speed performance to Fortran. Comparing to Figure 4.9, it is obvious that array operations are faster in Fortran than C/C++. Static arrays and looping syntax with fixed lengths of arrays were also tested for Fortran, as shown in Figure 4.12, which generally allow compilers to employ further optimisations compared to arrays and looping structures with variable lengths. However, scripting codes with array operations of fixed lengths are less flexible and harder to maintain.

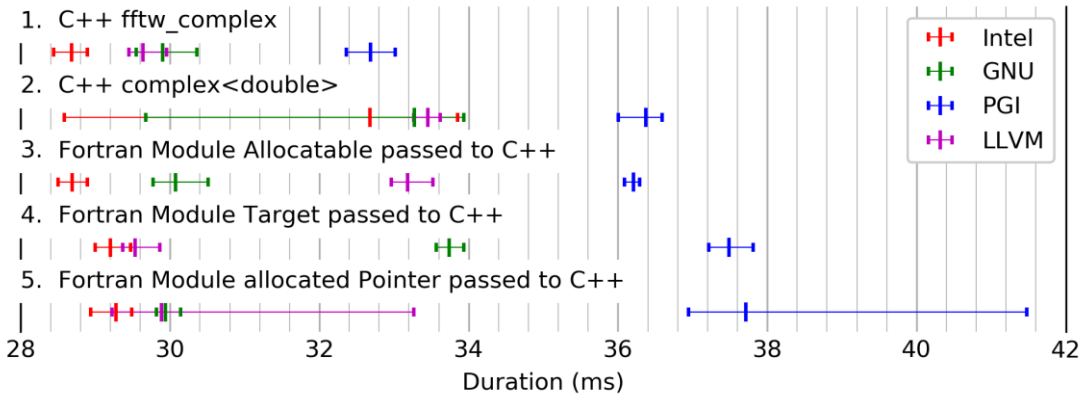


Figure 4.10. Speed performances of the FFTW library using different types of arrays for FFT operations. Double precision complex FFT and IFFT of a 64×64 array were repeated 1000 times, single-threaded, and timed to simulate the time-consumption of FFT operations in multislice propagation. Ten loops of all tests were timed and plotted with the average, minimum and maximum values. A combination of various array types and VPM were tested as annotated. Four sets of Fortran and C++ compilers were tested: 1) Intel compilers with flags "-fast -march=core-avx2", 2) GNU compilers with "-Ofast -march=native", 3) PGI compilers with "-fast", and 4) LLVM (flang and clang++) compilers with "-Ofast -march=native". The testing platform was a 64-bit Linux system, with a 4-cores CPU operating at 3.5 GHz fixed frequency. Overall, the "fftw_complex" array type, provided and suggested by the FFTW library, showed the best performance across various compilers.

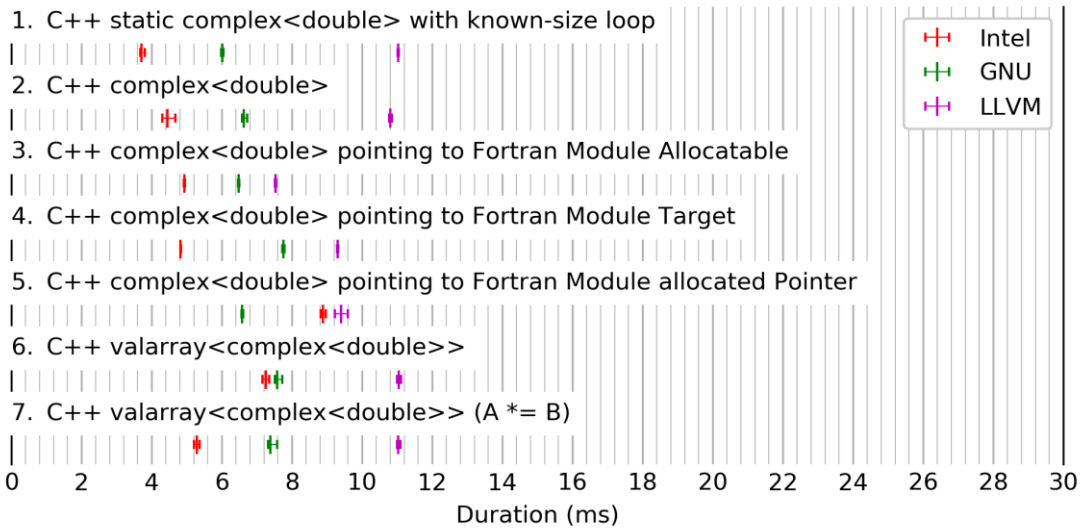


Figure 4.11. Speed of array operations with various array types in C++. Double precision complex array element-wise multiplications of $A = A \cdot B$ then $A = A \cdot C$ were repeated 1000 times, single-threaded, and timed to simulate the time-consumption of array multiplications in multislice propagation. A, B and C are complex arrays with size of 64×64. Ten loops of all tests were timed and plotted with the average, minimum and maximum values. A combination of various array types and VPM were tested as annotated. For "C++ valarray<complex<double>>", a different array operation syntax, A *= B, was also tested and shown in (7). Three sets of Fortran and C++ compilers were tested: 1) Intel compilers with flags "-fast -march=core-avx2", 2) GNU compilers with "-Ofast -march=native", and 3) LLVM (flang and clang++) compilers with "-Ofast -march=native". The testing platform was a 64-bit Linux system, with a 4-cores CPU operating at 3.5 GHz fixed frequency. Overall, the performance of complex array operation in C++ is slower than that of Fortran.

In summary, scripting codes involving C/Fortran mixed-languages programming, which mainly use Fortran for array operations and use the array type "fftw_complex" provided by the FFTW library written in C, were tested to be the most computational efficient. By implementing fixed-length arrays/looping structures, the speed of array operations may be slightly improved. However, complexity of the scripting codes will be increased while the flexibility of the program is reduced, which will also lead to more difficult maintenance.

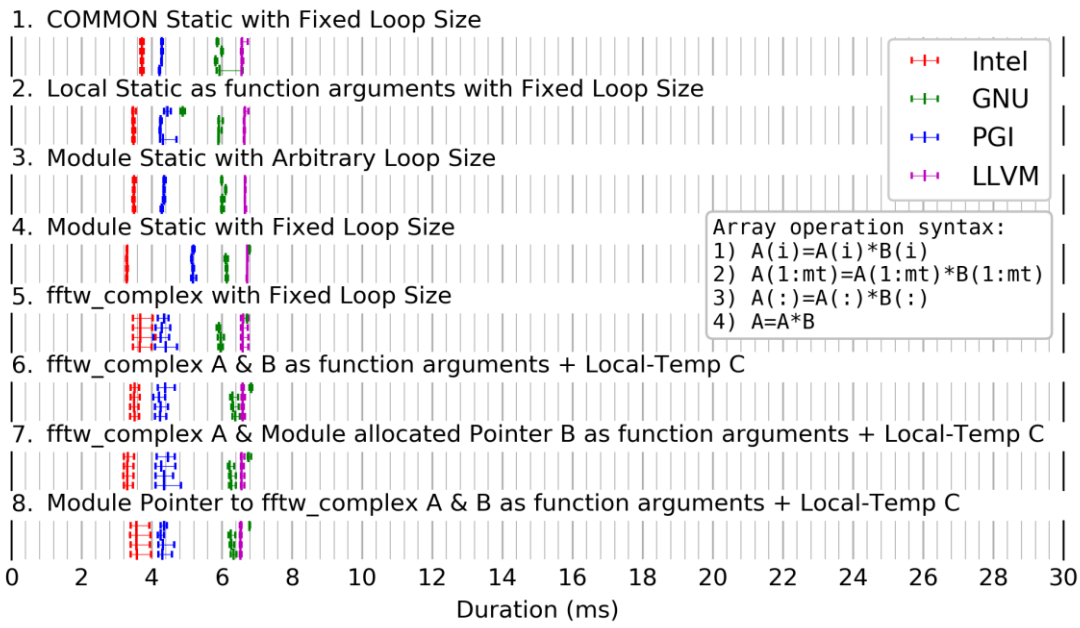


Figure 4.12. Speed of array operations with various static and local-temporary array types in Fortran. Double precision complex array element-wise multiplications of $A = A \cdot B$ then $A = A \cdot C$ were repeated 1000 times, single-threaded, and timed to simulate the time-consumption of array multiplications in multislice propagation. A, B and C are complex arrays with size of 64×64. Ten loops of all tests were timed and plotted with the average, minimum and maximum values. A combination of various array types, VPM and AOS were tested as annotated. There are four AOS's for every array type and plotted in the same sequence as annotated. Four sets of Fortran and C++ compilers were tested: 1) Intel compilers with flags "-fast -march=core-avx2", 2) GNU compilers with "-Ofast -march=native", 3) PGI compilers with "-fast", and 4) LLVM (flang and clang++) compilers with "-Ofast -march=native". The testing platform was a 64-bit Linux system, with a 4-cores CPU operating at 3.5 GHz fixed frequency. Compared to Figure 4.9, the processing speed can be slightly improved when 1) static arrays with known sizes, 2) dynamically-allocated arrays but with fixed loop sizes, or 3) local-temporary arrays are used.

4.3.3. Single instruction, multiple data (SIMD) and vectorisation

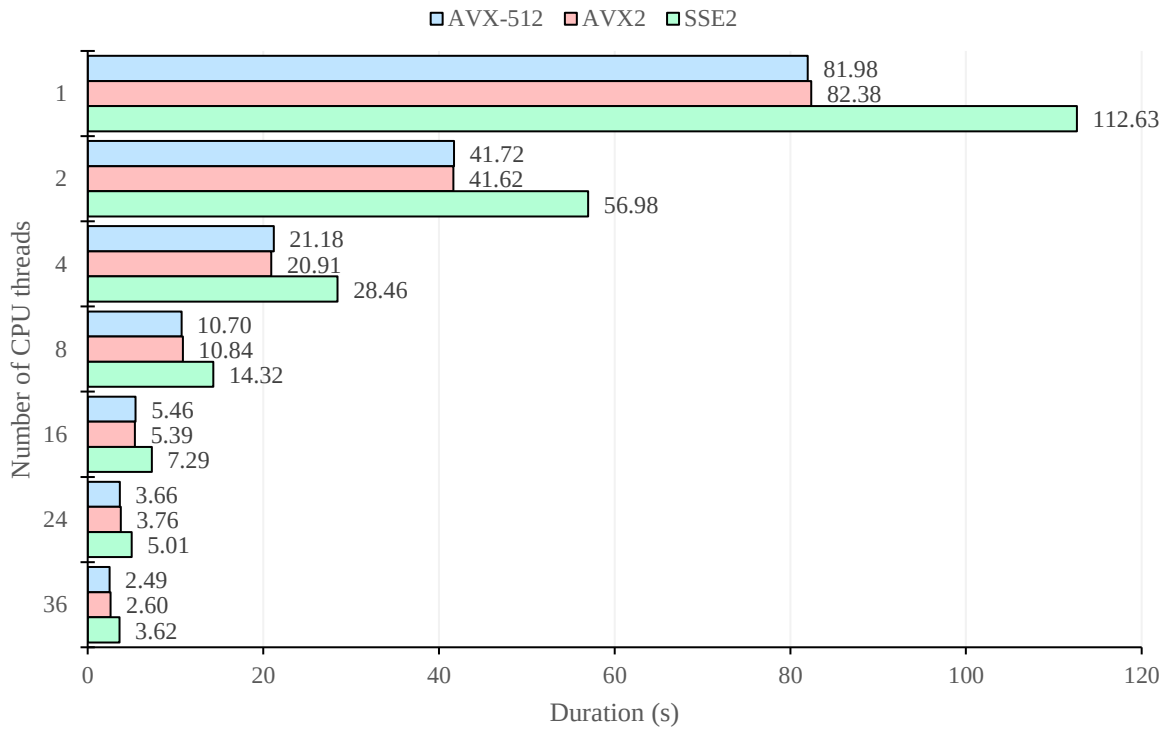


Figure 4.13. Performance of *QCBEDMS-PF* compiled with different SIMD instruction sets, SSE2, AVX2 and AVX-512. The Standard Test was executed five times on the Cluster CPU Node (with 36 CPU cores), to compare the averaged process times using different numbers of CPU threads and various auto-vectorisation configurations. The FFTW library was used for all FFT operations. The Intel compilers were used to compile both the *QCBEDMS-PF* program and the FFTW library: 1) for SSE2 tests, *QCBEDMS-PF* was compiled with Flag "-xSSE2", and FFTW was compiled with Option "—enable-sse2"; 2) for AVX2, *QCBEDMS-PF* with Flag "-xCORE-AVX2" and FFTW with Option "—enable-avx2"; and 3) for AVX-512, *QCBEDMS-PF* with Flag "-xCORE-AVX512" and FFTW with Options "—enable-avx2" and "—enable-avx512" ("—enable-avx2" is also suggested by FFTW when using "—enable-avx512"). Overall, AVX2 and AVX-512 have similar performances on *QCBEDMS-PF*, which are about 25% faster than using the SSE2 instruction set.

In addition to choosing the appropriated combination of array type and scripting syntax, vectorisation is another optimisation that can significantly benefit the speed of array operations on CPU. Vectorisation is also sometimes called as "single instruction, multiple data (SIMD)", which represents the action of applying identical operation upon multiple (and especially serial) data. On modern CPUs, vectorisation often leads to faster array operations, because the number of instructions (commands of operations given to the CPUs) that can be processed within a certain time is limited. For array operations, where identical operations are casted on multiple elements serially, such instruction processing speed becomes the limiting factor. With vectorisation, the total number of instructions can be reduced for array operations, thus resulting in higher processing speed. For *QCBEDMS-PF*, such

optimisations rely on the auto-vectorisation provided by the compilers. This is because manual implementation of SIMD will increase the complexity of the scripting codes, while the actual speed improvement is subtle compared to auto-vectorisation (since the most time-consuming components in the program have relatively simple syntax structures). In Figure 4.13, the speed performances of different vectorisations are summarised. Both the FFTW library and the *QCBEDMS-PF* program were complied with specific vectorisations. It is shown that both AVX2 and AVX-512 (usually available on mid- and high-end CPUs launched after 2011) have about 25% speed improvements compared to SSE2 (usually available on CPUs launched after 2003). There is no significant difference between AVX2 and AVX-512, although AVX-512, which allows doubled-amounts of data processed with one instruction compared to AVX2, should theoretically further accelerate the array operations. Therefore, according to the CPU on a computing machine, *QCBEDMS-PF* should be compiled with AVX2 or AVX-512 as long as they are available.

4.4. Precision considerations

Number of slices	Process time (s)		χ
	Double	Single	
1	0.07	0.05	1.16×10^{-7}
100	0.81	0.43	8.04×10^{-6}
1000	7.31	3.56	9.10×10^{-5}
2000	14.64	7.13	1.88×10^{-4}

Table 4.4. Processing speeds and errors (represented by χ as defined in Section 2.4) of *QCBEDMS-PF* using single-precision, compared to using double-precision. Like the Standard Test, with 64×64 reflection beams and 50×50 incident angles, CBED patterns of copper consisting of various numbers of slices were calculated and timed. Double and single floating-point number precisions were used for Fresnel propagators and wave propagation and tested respectively. The testing platform was Desktop A, using OpenCL (clFFT) on GPU. A fixed scale of intensities is applied to both single- and double-precision calculations. It is shown that with 2000-slices, single-precision calculation results in errors of approximately 0.0188%.

Although floating point operations using single-precision can achieve significant speed improvement, the accuracy may be compromised as: 1) single-precision variables have only 7 significant digits, 2) after multiplication/division etc., their accuracy may be reduced to 6 significant digits, and 3) after thousands of slices of iterative wave propagation, as shown in Table 4.4 and Figure 4.14, the accuracy will be reduced to only four or five significant digits in the final intensities. For thin specimens, e.g. those used in STEM experiments, single-precision may provide sufficient accuracy and half of the process time compared to double-precision. However, for QCBED analyses involve very thick specimens should avoid using single-precision.

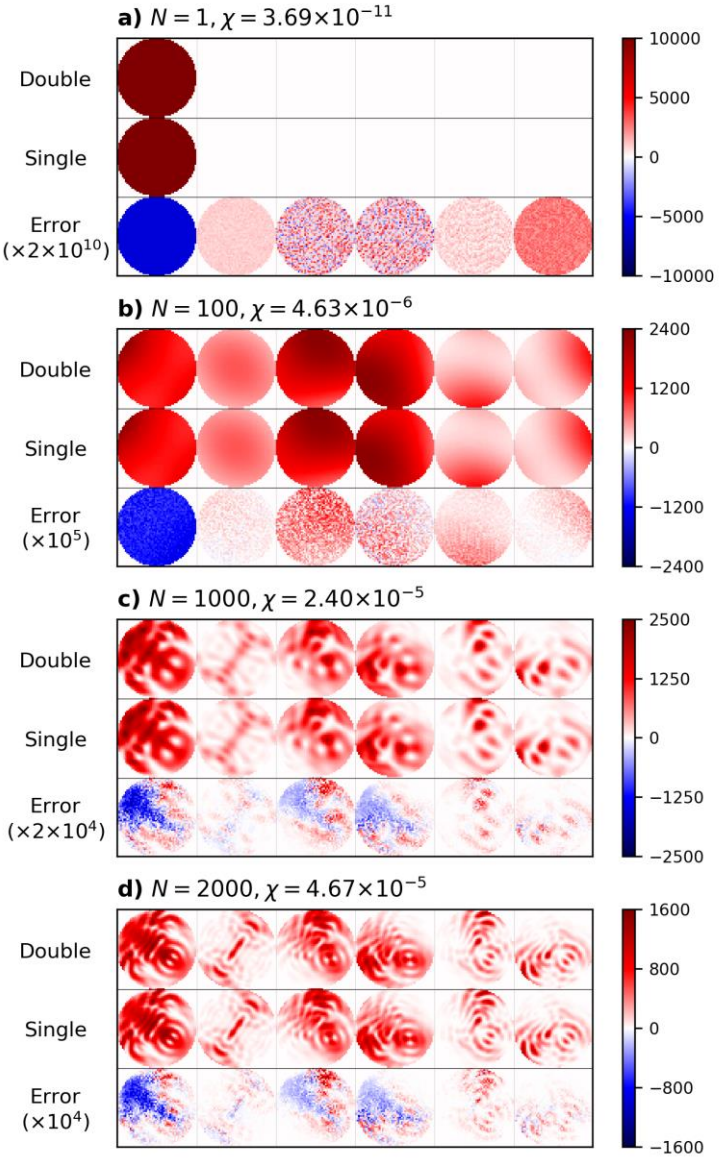


Figure 4.14. Errors caused by single-precision floating number operation. Different numbers of slices (*N*) were used, resulting in different magnitudes of errors. Instead of applying a fixed scale, the intensities of the single-precision patterns were scaled to match that of the double-precision patterns, resulting in lower χ, which mimics the actual QCBED refinement practices. It is shown that with 2000-slices, single-precision calculation results in errors of approximately 0.00467%.

Beside common floating-point number operations, optimisations of precision were also applied to sine and cosine calculations. In Fortran, the double-precision operations of sine and cosines have the following features: 1) sin(0) = sin(Pi) = 0, 2) cos(0) = 1 and cos(Pi) = -1, but 3) cos(Pi/2) ≈ 10^-17, which does not exactly equal 0. Due to this small residual, small differences will occur during the calculations of structure factors for symmetric reflections. Although the differences may be too small to harm the calculations, it is still visible to the double-precision operations. Therefore, for all sine, cosine and Pi related operations, quadruple-precision will be used in *QCBEDMS-PF*, except for

calculations of the Fresnel propagator, because quadruple-precision reduces the computing speed dramatically.

4.5. Final performance

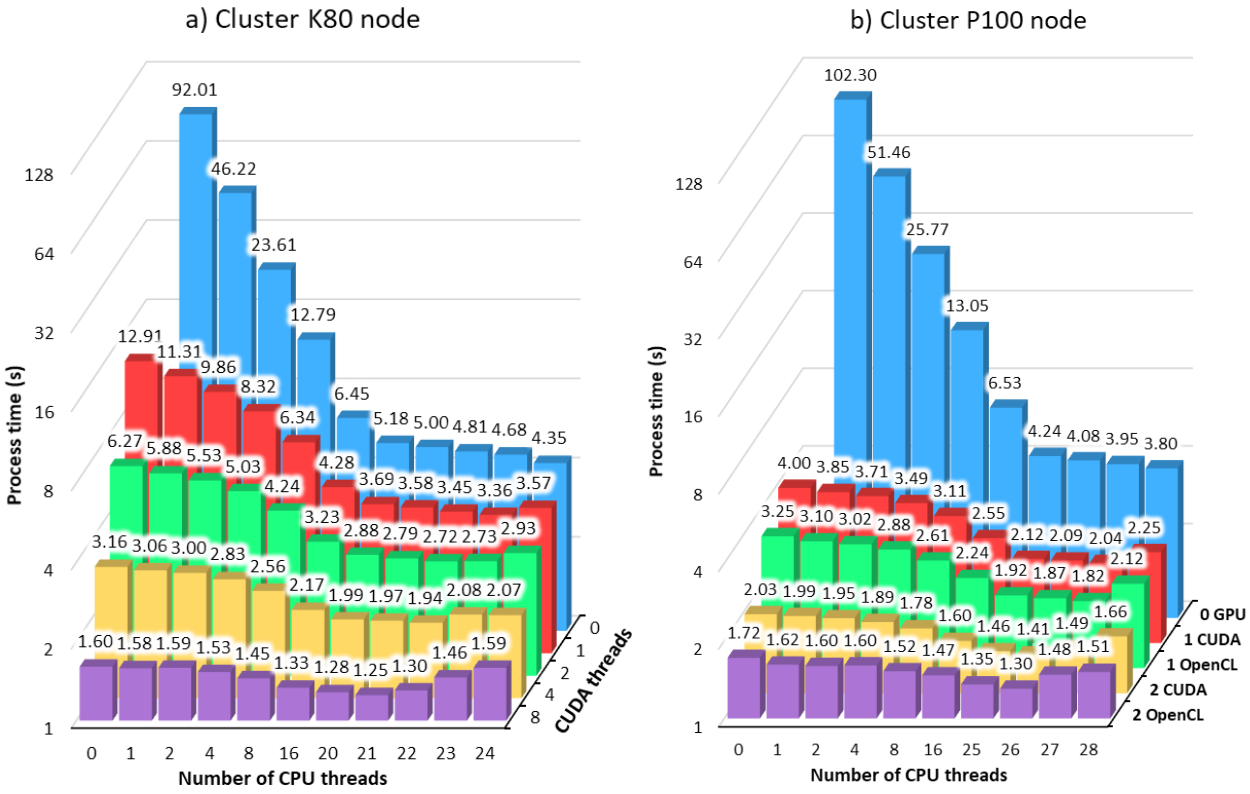


Figure 4.15. Speed performance of *QCBEDMS-PF* running the Standard Test on cluster nodes with GPUs. On the a) Cluster K80 Node, only GPU accelerations with CUDA were tested because OpenCL was not installed. While on the b) Cluster P100 Node, both OpenCL and CUDA were tested with various combinations of GPUs. Linear reduction in total process times was observed for both increasing numbers of CPU cores and GPUs. Except for cases where GPUs are used while almost all CPU cores are occupied, the total process times will increase as the coordination of GPUs is also CPU-intense. The minimum process time for the (medium resolution) Standard Test on such cluster nodes can be as fast as near a second.

The overall calculation times of *QCBEDMS-PF* on modern machines, which contain multiple CPU cores and may also contain one or more GPUs, can be reduced to be within 10 seconds for one iteration with medium resolution (i.e. the Standard Test). This performance is about 50 times faster than what was observed with the original single-thread program *QCBEDMS*. The process times of *QCBEDMS-PF* with various combinations of CPUs and GPUs by using cluster nodes are shown in Figure 4.15. With increased numbers of CPU cores, the process times decrease linearly. This is also true for multiple GPUs parallelisation. Moreover, while a mixture of CPU and GPU resources on a machine is used, the total computing capabilities of different units are still summed linearly. This indicates that

the *QCBEDMS-PF* has a near-100% parallelisation efficiency, which suggests further linearly reduction of total process time can be achieved with multiple cluster nodes (enabled by the implemented MPI functions in the program). Although the communications between CPU and GPU, as well as between different cluster nodes, may limit the minimum process time while massive computing resources are involved.

Additionally, the total process time can be further decreased with reduced resolution (real-space resolution as data points in the CBED patterns, or diffraction-space resolutions as the number of included reflection beams or coarser slicing, i.e. including multiple layers of atoms in one slice) or reduced operation precision. For examples, 1) fewer incident angles (lower real-space resolution), 2) smaller reflection beam mesh sizes (lower horizontal resolutions in diffraction-space), 3) coarser slicing (lower vertical resolutions in diffraction-space) or 4) using single floating-point precision for GPU acceleration, will all lead to linear reduction of the total process time with compromised accuracy and precision. However, such rough approximations enable very fast QCBED analysis for certain scenario, e.g. “live QCBED” analysis during TEM experiments. For example, using a 32-by-32 reflection beam mesh size, 100-slices, 30-by-30 real-space resolutions and single-precision for CBED pattern simulations (GPU-only), the total operand can be reduced to 1/220 of the Standard Test, which is about 0.02 second on a Nvidia P100 GPU (neglecting the limitations of network speed and machine respond time etc.). With conservative estimate including other limiting factors, if four Nvidia P100 GPUs are used in such case, one iteration should practically take less than 0.2 second, which enables about 3000 iterations in a minute. In scenarios where the TEM specimens are very thin, e.g. STEM experiments, the concern of decreased accuracy caused by rough QCBED approximation could be minimised, making “live QCBED” analysis possible while relatively reliable.

Chapter 4. Acceleration of QCBED multislice pattern-matching

Chapter 5. The role of the independent-atom model

As introduced in Section 1.2, in order to characterise the bonding of copper, its absolute charge density with bonding and IAM charge density without bonding have to be determined. In this project, QCBED was used to measure the absolute charge density, while the IAM was determined using different methods. Additionally, due to the nature of QCBED, the process of measuring the absolute charge density also involves the usage of IAM. Therefore, the first step of bonding characterisation of copper using QCBED is to determine the IAM for copper.

Conventionally, the IAM is derived from the tabulated atomic scattering factors published by Doyle and Turner in 1968 [9]. The tabulated atomic scattering factors were calculated using the relativistic Hartree-Fock (RHF) atomic wave function method by Coulthard [82], which takes account of the electron exchange in the charge density of the independent atoms; however the electron correlation is neglected. To obtain the atomic scattering factors at scattering angles required by the IAM of copper, interpolation of the tabulated values is needed. In this project, the parameterisation published by Peng *et al.* in 1996 [62], which is included in the latest version of the International Tables for Crystallography [58], was first used for interpolated electron atomic scattering factors of copper. However, as pointed out in Section 2.2.2, Peng *et al.*'s parameterisation of copper has limited fitting to the tabulated atomic scattering factors at higher scattering angles. In order to gain perfect fitting and thus obtain more accurate atomic scattering factors for the IAM of copper, a cubic-spline method was developed and was also used for IAM of copper. Furthermore, the tabulated RHF atomic scattering factors by Doyle and Turner were also reviewed. DFT simulations, which include both the electron exchange and correlation, were performed to calculate the charge density of an isolated copper atom (by Dr. Andrew E. Smith). IAM was then derived from the DFT-calculated charge density of the isolated copper atom and used for QCBED refinements and bonding determination.

5.1. Atomic scattering factors of different IAMs

The parameterisation of the electron atomic scattering factors of copper published by Peng *et al.* in 1996 [62] was used to determine the IAM of copper at the early stage of this project. This parameterisation was chosen because: 1) QCBED refinements require electron atomic scattering factors for unrefined higher-order structure factors, 2) it is an update to the original parameterisation published by Doyle and Turner, featuring improved fitting to the electron scattering factors, and 3) it is considered as the standard approach since it is included in the latest version of the International Tables for Crystallography [58]. As discussed in Section 2.2, the fitting of Peng *et al.*'s parameterisation may be considered as poor at higher scattering angles, yielding inaccurate higher-order atomic scattering factors. Errors in the higher-order scattering factors are likely to lead to inaccurate QCBED pattern-matching refinements, as the calculated intensities will always contains incorrect information from higher scattering angles which are not refined. Additionally, an inaccurate IAM will also cause errors during the bonding determination process, where the charge density of IAM is subtracted from the experimental charge density. Therefore, atomic scattering factors of copper with higher accuracy are required for more accurate bonding characterisation.

There are several parameterisations available for electron [60, 61] and X-ray [86, 87] atomic scattering factors respectively, providing improved accuracy of atomic scattering factors of copper. However, instead of parameterisation, cubic-spline was used for interpolation of the tabulated electron and X-ray atomic scattering factors from the International Tables for Crystallography respectively. This is because, comparing to parameterisation, polynomial interpolations like the cubic-spline can provide perfect fitting to the tabulated values. Moreover, for new data sets of atomic scattering factors other than the RHF values by Doyle and Turner, no parameter fitting is required by polynomial interpolations. For example, for the IAM of copper calculated by DFT which will be discussed later in this section, parameterisation becomes impossible as there are millions of atomic scattering factor values within the scattering angles $0 < s < 5$ Å.

hkl	*s* (Å^{-1})	X-ray atomic scattering factor			Electron atomic scattering factor (Å)		
		X1) Doyle & Turner (1968)	X2) Cubic-splined RHF	X3) DFT-IAM	E1) Peng *et al.* (1996)	E2) Cubic-splined RHF	E3) DFT-IAM
111	0.2404	22.046	22.042	21.875	2.885	2.882	3.022
200	0.2776	20.689	20.676	20.522	2.597	2.586	2.697
220	0.3926	16.739	16.746	16.686	1.900	1.903	1.958
311	0.4603	14.737	14.748	14.740	1.599	1.610	1.649
222	0.4808	14.190	14.198	14.204	1.523	1.532	1.569
400	0.5552	12.421	12.420	12.455	1.286	1.288	1.316
331	0.6050	11.420	11.412	11.454	1.157	1.150	1.175
420	0.6207	11.132	11.124	11.166	1.119	1.110	1.135
422	0.6800	10.159	10.151	10.193	0.991	0.976	0.997
333	0.7212	9.578	9.573	9.616	0.911	0.894	0.913
511	0.7212	9.578	9.573	9.616	0.911	0.894	0.913
440	0.7852	8.816	8.817	8.857	0.799	0.784	0.801
531	0.8211	8.453	8.456	8.494	0.742	0.729	0.745
442	0.8328	8.344	8.348	8.386	0.724	0.713	0.729
600	0.8328	8.344	8.348	8.386	0.724	0.713	0.729
620	0.8778	7.961	7.967	8.001	0.660	0.653	0.668
533	0.9101	7.719	7.726	7.756	0.618	0.614	0.629
622	0.9207	7.645	7.652	7.681	0.605	0.602	0.616
444	0.9616	7.381	7.387	7.411	0.557	0.559	0.572
551	0.9912	7.209	7.215	7.234	0.525	0.531	0.543
711	0.9912	7.209	7.215	7.234	0.525	0.531	0.543
640	1.0009	7.156	7.161	7.180	0.516	0.522	0.534
642	1.0387	6.962	6.964	6.979	0.479	0.489	0.500
553	1.0661	6.831	6.832	6.844	0.455	0.467	0.478
731	1.0661	6.831	6.832	6.844	0.455	0.467	0.478
800	1.1104	6.638	6.636	6.643	0.420	0.435	0.444

Table 5.1. Comparison of different X-ray and electron atomic scattering factors for copper. X1) and E1) are derived from the parameterisations by Doyle & Turner's parameterisation [9] and Peng *et al.*[62] (fitted up to s = 6 Å^{-1}). X2 and E2) are cubic-splined from the tabulated RHF atomic scattering factors in [84] and [58], respectively. X3 and E3) are derived from the DFT-IAM calculation (interpolated with cubic-spline). The X-ray atomic scattering factors are used as the IAM for bonding charge density determination, while the electron atomic scattering factors are used as the IAM for potential density measurements during QCBED refinements.

Comparisons of the low-order structure factors between the parameterised-IAM, cubic-splined RHF-IAM and DFT-IAM are shown in Table 5.1. Because the differences between the parameterisation and cubic-spline of the RHF values are considerably small when compared to the DFT-IAM, as well as the parameterisation by Peng *et al.* [62] is known to be less accurate, discussions will be focused on only the cubic-splined RHF-IAM (hereinafter referred to as "RHF-IAM") and DFT-IAM. Comparing the RHF-IAM and DFT-IAM, the largest differences are observed at $0.1 \leq s \leq 0.4$ Å^{-1}, where the DFT X-ray and electron atomic scattering factors can be about 1% and 5% offsetting from the RHF values respectively. Smaller differences, which are still significant, are observed at higher s of atomic scattering factors.

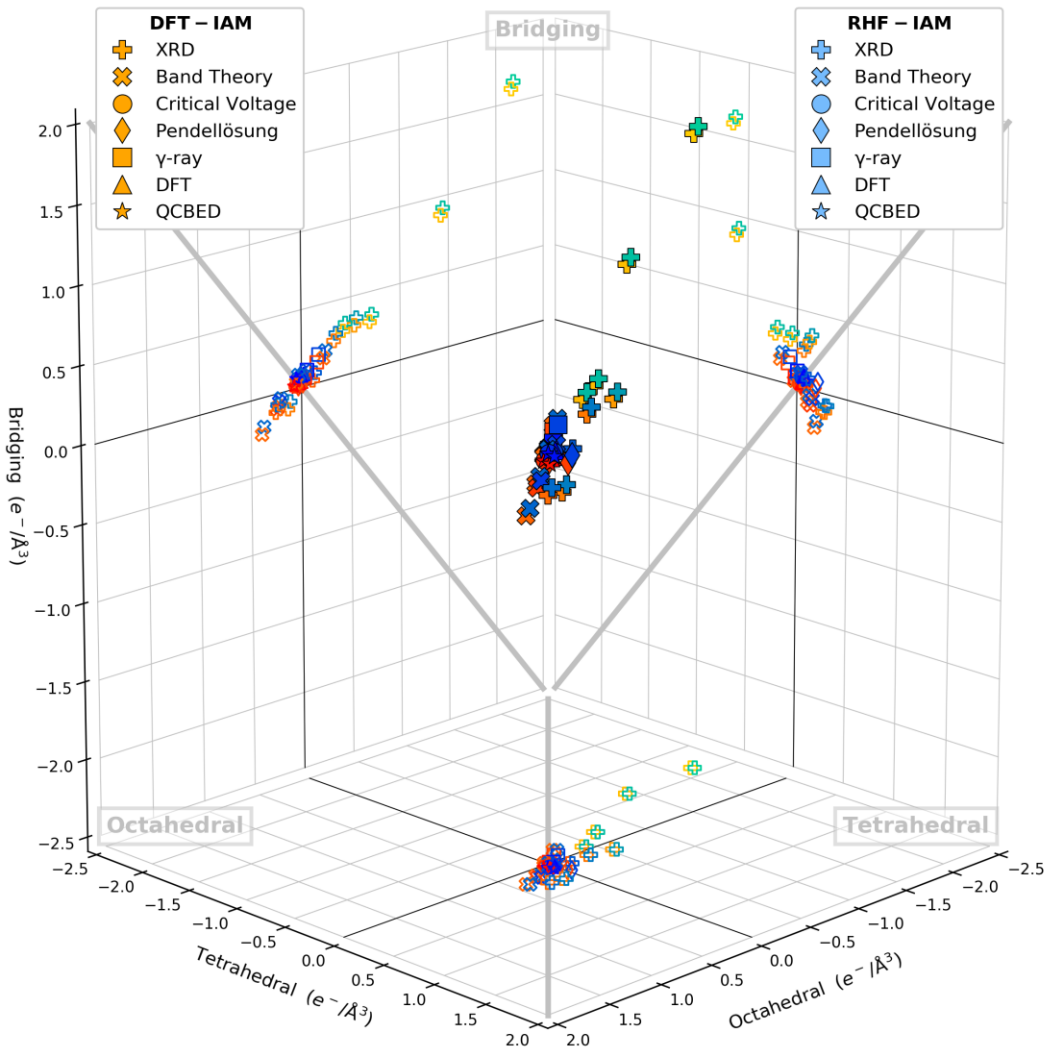


Figure 5.1. Summary of bonding in copper for previous studies, using the RHF-IAM and DFT-IAM. The intensities of charge density ($e^-/\text{Å}^3$) at the tetrahedral sites, octahedral centre and bridging sites of copper's unit cell are used to estimate the shapes of bonding distributions. For values below 0 $e^-/\text{Å}^3$, anti-bonding is indicated to occur at the corresponding location. Results of different studies are grouped by their techniques. Shades from orange to red are used to indicate the publication years from 1930 to 2005 using the DFT-IAM. The light to dark blue shades are used to show the corresponding studies from using the RHF-IAM. Experimental techniques include XRD (9 sets) [18, 39, 43, 45, 47-49, 80], critical voltages (2 sets) [4, 12], Pendellösung (2 sets) [14, 32], γ-ray (2 sets) [11, 15] and QCBED (2 sets) [4, 7]. Theoretical calculations include band theory (7 sets) [13, 16, 39, 41] and DFT (2 sets) [4, 7]. The bonding charge densities are calculated with only the three lowest-order structure factors. The difference between bonding concluded using RHF-IAM and DFT-IAM are constant, while the absolute values of bonding charge density are changed for all results. For example, the intensities at tetrahedral sites are decreased by about 0.02 $e^-/\text{Å}^3$, increased by about 0.02 $e^-/\text{Å}^3$ at octahedral centres, and increased by about 0.05 $e^-/\text{Å}^3$ at the bridging sites.

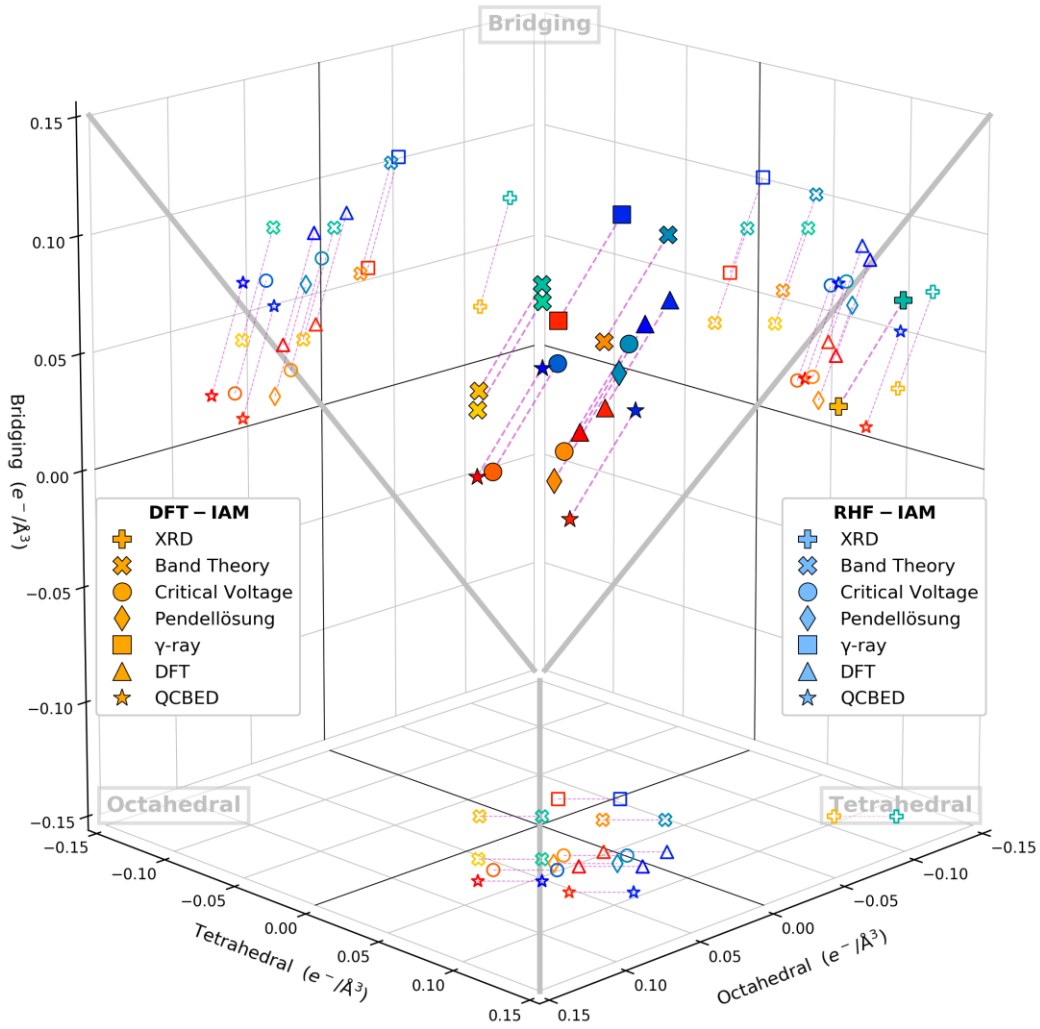


Figure 5.2. A zoom-in graph of Figure 5.1 which includes most of the more recent studies [4, 7, 11, 12, 16, 18, 41, 79]. Shades from light to dark blue are used to indicate the publication years from 1968 to 2005 using the RHF-IAM. Shades from orange to red are used to show the DFT-IAM values and linked to the corresponding RHF-IAM values. Results of different studies are grouped by their techniques. Results include XRD (1 set) [18], band theory (3 sets) [16, 41, 79], critical voltages (2 sets) [4, 12], Pendellösung (1 set) [78], γ-ray (2 sets) [11, 15] and QCBED (2 sets) [4, 7] and DFT (2 sets) [4, 7]. It is shown that the changes caused by a different IAM is very significant compared the differences between various studies.

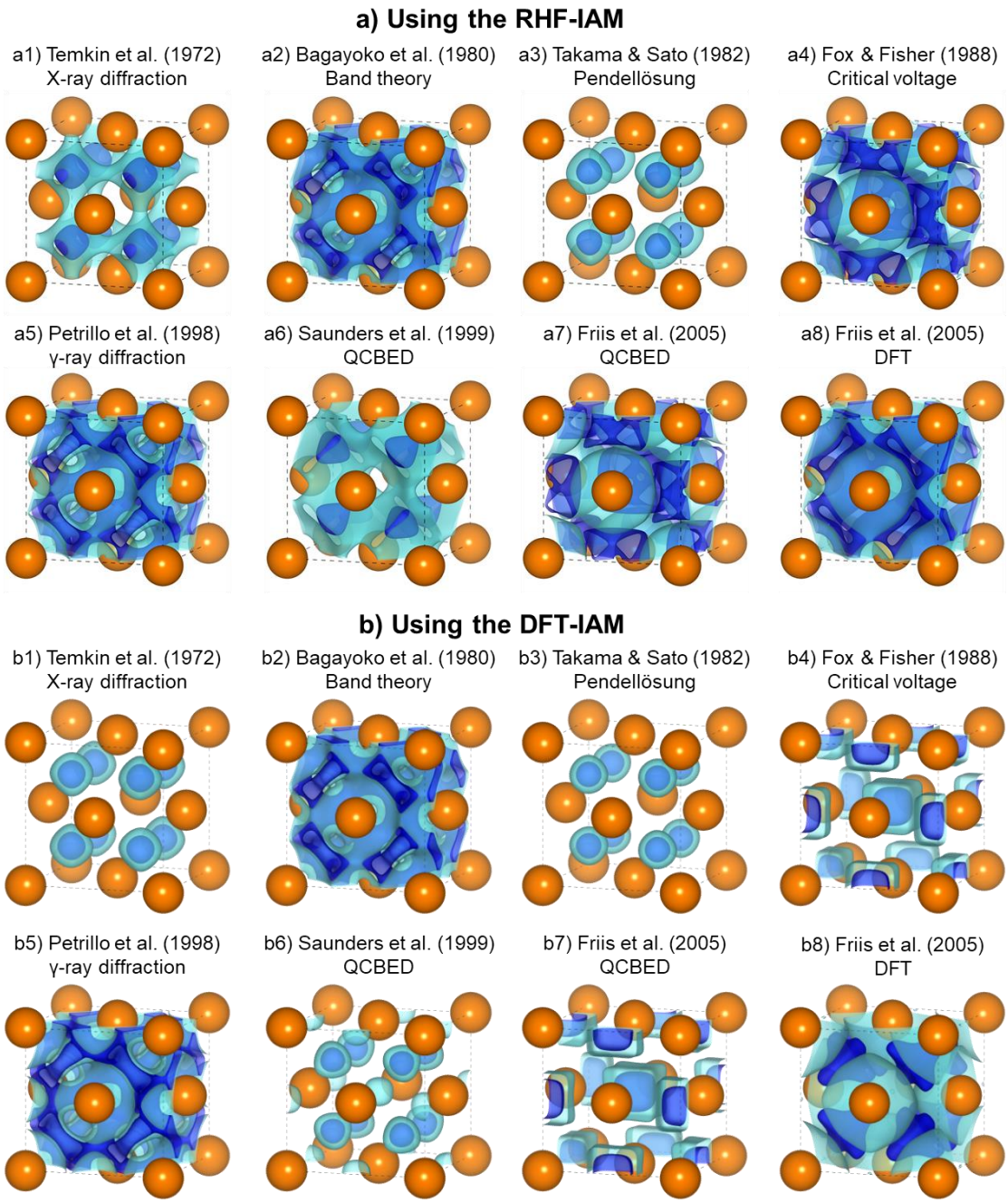


Figure 5.3. The bonding charge density of copper using various IAMs. The iso-surfaces are plotted at 50% (light blue) and 80% (dark blue) of the maximum bonding electron density of each result. Only the three lowest-order structure factors were included. The IAM of a) is cubic-splined from the tabulated RHF atomic scattering factors in [84], while b) is cubic-splined from the DFT-IAM. With a different IAM, the bonding distributions have changed significantly.

For copper, as well as other FCC metals, only the first two-to-three lowest-order structure factors are expected to significantly differ from the IAM structure factors [5, 99], since these structure factors cover most of the charge distribution in the valence electrons. Since the lowest-order structure factors in copper measured by QCBED [4, 7] are typically no more than 2% different from the RHF-IAM structure factors, very different conclusion of the bonding in copper can be yielded with the 1%-altered

DFT-IAM. The three lowest-order structure factors, which contribute the most bonding deformations, are all within the scattering angle range ($0.1 \leq s \leq 0.4$ Å^{-1}) where largest differences are observed between the RHF and DFT calculations. The differences between using the RHF-IAM and DFT-IAM of bonding determination for previous studies is summarised in Figure 5.1. Cold shades and warm shades are used to indicate the corresponding results using the RHF-IAM and DFT-IAM respectively. Replacing the RHF-IAM with the DFT-IAM for bonding determination results in a constant shifting for all data sets. For example, the intensities at tetrahedral sites are decreased by about 0.02 $e^-/\text{Å}^3$, increased by about 0.02 $e^-/\text{Å}^3$ at octahedral centres, and increased by about 0.05 $e^-/\text{Å}^3$ at the bridging sites, as summarised in the figure. Generally, such shifting causes a less-bridging-like and more-octahedral-like transformation in the bonding morphologies. A zoom-in graph which includes most of the more recent studies [4, 7, 11, 12, 16, 18, 41, 79] is also shown in Figure 5.2 for more detailed discussions. In this magnified figure, it can be illustrated that the impact of IAM alternation is very significant when comparing the difference among various studies. In particular, bonding types for some data sets have migrated from bridging-dominant to tetrahedral-dominant. The actual bonding distributions of copper from the most recent studies using the two IAMs can be found in Figure 5.3. Although the general types (octahedral-, tetrahedral- or bridging-like) of bonding have not changed, significant changes of the morphologies of bonding in these studies are caused by using a different IAM. Especially for the results of critical voltage by Fox and Fisher (1988) [12] and QCBED by Friis *et al.* (2005) [7], the locations of the maximum bonding density are also changed. The changes in the distributions of band theory by Bakayoko *et al.* (1980) [16] and γ-ray by Petrillo *et al.* (1998) [11] are relatively small, because the ratios of ΔF_{111}, ΔF_{200} and ΔF_{220} are similar. However, the magnitudes of bonding have changed, which is also true for all literature results.

5.2. IAM of copper calculated by DFT

The RHF approximation by Doyle and Turner [9] has been considered the most accurate atomic scattering factors since it was published 50 years ago. However, it only calculates the electron exchange in the charge density of the independent atom while the electron correlation is neglected. In the current work, DFT calculations (by Dr. Andrew E. Smith), which include both the exchange and correlation effects, are performed to generate a more accurate copper IAM.

s	X-ray	Electron	*s*	X-ray	Electron	*s*	X-ray	Electron	*s*	X-ray	Electron	*s*	X-ray	Electron
0.01	-	-	0.51	13.4791	1.4282	1.01	7.1281	0.5132	1.51	5.2713	0.2491	2.01	3.8269	0.1491
0.02	-	-	0.52	13.2416	1.3948	1.02	7.0742	0.5044	1.52	5.2422	0.2461	2.02	3.8002	0.1478
0.03	-	-	0.53	13.0095	1.3625	1.03	7.0218	0.4958	1.53	5.2105	0.2432	2.03	3.7741	0.1465
0.04	-	-	0.54	12.7835	1.3310	1.04	6.9711	0.4875	1.54	5.1800	0.2404	2.04	3.7478	0.1452
0.05	-	-	0.55	12.5630	1.3005	1.05	6.9214	0.4793	1.55	5.1497	0.2376	2.05	3.7218	0.1440
0.06	-	-	0.56	12.3481	1.2709	1.06	6.8728	0.4713	1.56	5.1194	0.2349	2.06	3.6960	0.1427
0.07	-	-	0.57	12.1389	1.2421	1.07	6.8254	0.4635	1.57	5.0891	0.2322	2.07	3.6708	0.1415
0.08	-	-	0.58	11.9347	1.2141	1.08	6.7791	0.4560	1.58	5.0596	0.2295	2.08	3.6449	0.1403
0.09	-	-	0.59	11.7374	1.1869	1.09	6.7338	0.4485	1.59	5.0288	0.2269	2.09	3.6196	0.1391
0.10	-	-	0.60	11.5449	1.1605	1.10	6.6894	0.4413	1.60	4.9989	0.2244	2.10	3.5945	0.1379
0.11	27.0584	4.6470	0.61	11.3579	1.1347	1.11	6.6460	0.4342	1.61	4.9688	0.2219	2.11	3.5689	0.1367
0.12	26.7200	4.5097	0.62	11.1763	1.1097	1.12	6.6034	0.4273	1.62	4.9386	0.2194	2.12	3.5450	0.1356
0.13	26.3698	4.3716	0.63	10.9994	1.0855	1.13	6.5616	0.4206	1.63	4.9088	0.2170	2.13	3.5197	0.1344
0.14	26.0102	4.2341	0.64	10.8287	1.0618	1.14	6.5205	0.4140	1.64	4.8791	0.2146	2.14	3.4950	0.1333
0.15	25.6438	4.0983	0.65	10.6627	1.0388	1.15	6.4803	0.4075	1.65	4.8490	0.2123	2.15	3.4704	0.1322
0.16	25.2722	3.9653	0.66	10.5017	1.0164	1.16	6.4403	0.4013	1.66	4.8195	0.2100	2.16	3.4464	0.1311
0.17	24.8973	3.8356	0.67	10.3455	0.9946	1.17	6.4016	0.3951	1.67	4.7895	0.2078	2.17	3.4173	0.1300
0.18	24.5202	3.7099	0.68	10.1941	0.9734	1.18	6.3631	0.3891	1.68	4.7597	0.2056	2.18	3.3982	0.1289
0.19	24.1421	3.5885	0.69	10.0469	0.9528	1.19	6.3253	0.3832	1.69	4.7303	0.2034	2.19	3.3745	0.1279
0.20	23.3860	3.3591	0.70	9.9051	0.9327	1.20	6.2878	0.3775	1.70	4.7004	0.2012	2.20	3.3510	0.1268
0.21	23.0094	3.2512	0.71	9.7673	0.9131	1.21	6.2511	0.3719	1.71	4.6709	0.1991	2.21	3.3273	0.1258
0.22	22.6343	3.1478	0.72	9.6331	0.8941	1.22	6.2145	0.3664	1.72	4.6414	0.1971	2.22	3.3041	0.1248
0.23	22.2613	3.0488	0.73	9.5043	0.8756	1.23	6.1787	0.3610	1.73	4.6121	0.1950	2.23	3.2805	0.1238
0.24	21.8906	2.9541	0.74	9.3785	0.8576	1.24	6.1432	0.3558	1.74	4.5826	0.1930	2.24	3.2580	0.1228
0.25	21.5226	2.8634	0.75	9.2576	0.8400	1.25	6.1080	0.3506	1.75	4.5535	0.1911	2.25	3.2353	0.1218
0.26	21.1575	2.7766	0.76	9.1394	0.8230	1.26	6.0730	0.3456	1.76	4.5242	0.1891	2.26	3.2128	0.1208
0.27	20.7955	2.6936	0.77	9.0259	0.8063	1.27	6.0386	0.3407	1.77	4.4953	0.1872	2.27	3.1903	0.1199
0.28	20.4370	2.6141	0.78	8.9150	0.7901	1.28	6.0043	0.3359	1.78	4.4662	0.1853	2.28	3.1681	0.1189
0.29	20.0821	2.5379	0.79	8.8085	0.7743	1.29	5.9703	0.3312	1.79	4.4372	0.1835	2.29	3.1456	0.1180
0.30	19.7309	2.4649	0.80	8.7044	0.7590	1.30	5.9369	0.3266	1.80	4.4083	0.1817	2.30	3.1239	0.1171
0.31	19.3838	2.3949	0.81	8.6042	0.7440	1.31	5.9035	0.3221	1.81	4.3773	0.1799	2.31	3.1021	0.1162
0.32	19.0407	2.3277	0.82	8.5072	0.7294	1.32	5.8704	0.3177	1.82	4.3509	0.1781	2.32	3.0806	0.1153
0.33	18.7020	2.2632	0.83	8.4129	0.7152	1.33	5.8375	0.3134	1.83	4.3223	0.1764	2.33	3.0585	0.1144
0.34	18.3678	2.2013	0.84	8.3216	0.7014	1.34	5.8046	0.3092	1.84	4.2941	0.1747	2.34	3.0374	0.1135
0.35	18.0383	2.1417	0.85	8.2326	0.6879	1.35	5.7722	0.3050	1.85	4.2652	0.1730	2.35	3.0177	0.1126
0.36	17.7121	2.0846	0.86	8.1468	0.6748	1.36	5.7400	0.3010	1.86	4.2371	0.1713	2.36	2.9963	0.1117
0.37	17.3936	2.0291	0.87	8.0637	0.6620	1.37	5.7077	0.2970	1.87	4.2090	0.1697	2.37	2.9755	0.1109
0.38	17.0787	1.9759	0.88	7.9830	0.6496	1.38	5.6758	0.2931	1.88	4.1808	0.1681	2.38	2.9550	0.1100
0.39	16.7683	1.9247	0.89	7.9048	0.6374	1.39	5.6441	0.2893	1.89	4.1529	0.1665	2.39	2.9347	0.1092
0.40	16.4645	1.8751	0.90	7.8294	0.6255	1.40	5.6123	0.2856	1.90	4.1250	0.1649	2.40	2.9145	0.1084
0.41	16.1654	1.8274	0.91	7.7559	0.6140	1.41	5.5809	0.2819	1.91	4.0963	0.1634	2.41	2.8945	0.1076
0.42	15.8716	1.7812	0.92	7.6845	0.6027	1.42	5.5495	0.2783	1.92	4.0698	0.1619	2.42	2.8747	0.1068
0.43	15.5834	1.7367	0.93	7.6149	0.5918	1.43	5.5181	0.2748	1.93	4.0426	0.1604	2.43	2.8551	0.1060
0.44	15.2999	1.6937	0.94	7.5478	0.5811	1.44	5.4869	0.2714	1.94	4.0147	0.1589	2.44	2.8355	0.1052
0.45	15.0236	1.6519	0.95	7.4828	0.5706	1.45	5.4558	0.2680	1.95	3.9874	0.1574	2.45	2.8180	0.1044
0.46	14.7520	1.6116	0.96	7.4194	0.5604	1.46	5.4249	0.2647	1.96	3.9603	0.1560	2.46	2.7972	0.1036
0.47	14.4861	1.5725	0.97	7.3576	0.5505	1.47	5.3941	0.2615	1.97	3.9334	0.1546	2.47	2.7780	0.1029
0.48	14.2259	1.5347	0.98	7.2980	0.5408	1.48	5.3633	0.2583	1.98	3.9064	0.1532	2.48	2.7595	0.1021
0.49	13.9713	1.4981	0.99	7.2395	0.5314	1.49	5.3325	0.2551	1.99	3.8787	0.1518	2.49	2.7408	0.1014
0.50	13.7218	1.4626	1.00	7.1828	0.5222	1.50	5.3017	0.2521	2.00	3.8530	0.1505	2.50	2.7222	0.1006

Table 5.2. Tabulated X-ray and electron atomic scattering factors of copper calculated by DFT (for $0.1 \leq s \leq 2.5$) Å^{-1}. Cubic-spline was used to interpolate the atomic scattering factors at given *s*. The given range of *s* is because the calculation is simulated with an orthorhombic primitive unit cell (lattice constants a=15.88 Å, b=15.93 Å and c=15.98 Å), hence too few numbers of atomic scattering factors were obtained for $s < 0.1$ Å^{-1} for accurate interpolation, while at very large *s* the WIEN2k program has too many entities of atomic scattering factors to output.

S	X-ray	Electron	s	X-ray	Electron	s	X-ray	Electron	s	X-ray	Electron	s	X-ray	Electron
2.51	2.7043	0.0999	3.01	1.9931	0.0713	3.51	1.5873	0.0533	4.01	1.3557	0.0411	4.51	1.2102	0.0327
2.52	2.6859	0.0992	3.02	1.9822	0.0709	3.52	1.5811	0.0530	4.02	1.3522	0.0409	4.52	1.2078	0.0326
2.53	2.6680	0.0985	3.03	1.9718	0.0705	3.53	1.5754	0.0527	4.03	1.3488	0.0407	4.53	1.2054	0.0324
2.54	2.6503	0.0978	3.04	1.9615	0.0700	3.54	1.5695	0.0524	4.04	1.3452	0.0406	4.54	1.2030	0.0323
2.55	2.6328	0.0970	3.05	1.9512	0.0696	3.55	1.5636	0.0521	4.05	1.3418	0.0404	4.55	1.2001	0.0321
2.56	2.6155	0.0964	3.06	1.9411	0.0692	3.56	1.5579	0.0518	4.06	1.3384	0.0402	4.56	1.1983	0.0320
2.57	2.5980	0.0957	3.07	1.9311	0.0687	3.57	1.5524	0.0515	4.07	1.3351	0.0400	4.57	1.1959	0.0319
2.58	2.5811	0.0950	3.08	1.9211	0.0683	3.58	1.5468	0.0513	4.08	1.3314	0.0398	4.58	1.1936	0.0317
2.59	2.5641	0.0943	3.09	1.9113	0.0679	3.59	1.5417	0.0510	4.09	1.3283	0.0396	4.59	1.1913	0.0316
2.60	2.5474	0.0937	3.10	1.9017	0.0675	3.60	1.5358	0.0507	4.10	1.3250	0.0394	4.60	1.1889	0.0315
2.61	2.5309	0.0930	3.11	1.8925	0.0671	3.61	1.5305	0.0504	4.11	1.3219	0.0392	4.61	1.1866	0.0313
2.62	2.5145	0.0923	3.12	1.8826	0.0667	3.62	1.5250	0.0502	4.12	1.3186	0.0390	4.62	1.1843	0.0312
2.63	2.4982	0.0917	3.13	1.8732	0.0663	3.63	1.5200	0.0499	4.13	1.3154	0.0388	4.63	1.1821	0.0311
2.64	2.4823	0.0911	3.14	1.8639	0.0659	3.64	1.5146	0.0496	4.14	1.3122	0.0387	4.64	1.1798	0.0309
2.65	2.4662	0.0904	3.15	1.8548	0.0655	3.65	1.5096	0.0494	4.15	1.3090	0.0385	4.65	1.1776	0.0308
2.66	2.4504	0.0898	3.16	1.8458	0.0651	3.66	1.5044	0.0491	4.16	1.3058	0.0383	4.66	1.1753	0.0307
2.67	2.4349	0.0892	3.17	1.8368	0.0647	3.67	1.4991	0.0489	4.17	1.3028	0.0381	4.67	1.1731	0.0305
2.68	2.4069	0.0886	3.18	1.8286	0.0643	3.68	1.4944	0.0486	4.18	1.2997	0.0379	4.68	1.1709	0.0304
2.69	2.4032	0.0880	3.19	1.8192	0.0639	3.69	1.4894	0.0484	4.19	1.2966	0.0378	4.69	1.1687	0.0303
2.70	2.3878	0.0874	3.20	1.8106	0.0635	3.70	1.4846	0.0481	4.20	1.2936	0.0376	4.70	1.1665	0.0302
2.71	2.3741	0.0868	3.21	1.8021	0.0632	3.71	1.4797	0.0479	4.21	1.2908	0.0374	4.71	1.1643	0.0300
2.72	2.3593	0.0862	3.22	1.7934	0.0628	3.72	1.4749	0.0476	4.22	1.2864	0.0372	4.72	1.1621	0.0299
2.73	2.3447	0.0856	3.23	1.7855	0.0624	3.73	1.4702	0.0474	4.23	1.2847	0.0371	4.73	1.1600	0.0298
2.74	2.3304	0.0850	3.24	1.7772	0.0621	3.74	1.4655	0.0471	4.24	1.2818	0.0369	4.74	1.1578	0.0297
2.75	2.3155	0.0845	3.25	1.7690	0.0617	3.75	1.4609	0.0469	4.25	1.2788	0.0367	4.75	1.1556	0.0295
2.76	2.3016	0.0839	3.26	1.7609	0.0613	3.76	1.4569	0.0466	4.26	1.2759	0.0366	4.76	1.1535	0.0294
2.77	2.2876	0.0833	3.27	1.7529	0.0610	3.77	1.4518	0.0464	4.27	1.2731	0.0364	4.77	1.1514	0.0293
2.78	2.2735	0.0828	3.28	1.7451	0.0606	3.78	1.4473	0.0462	4.28	1.2703	0.0362	4.78	1.1493	0.0292
2.79	2.2599	0.0822	3.29	1.7374	0.0603	3.79	1.4429	0.0459	4.29	1.2675	0.0361	4.79	1.1472	0.0291
2.80	2.2464	0.0817	3.30	1.7296	0.0599	3.80	1.4384	0.0457	4.30	1.2648	0.0359	4.80	1.1451	0.0289
2.81	2.2329	0.0811	3.31	1.7221	0.0596	3.81	1.4340	0.0454	4.31	1.2619	0.0357	4.81	1.1430	0.0288
2.82	2.2197	0.0806	3.32	1.7147	0.0592	3.82	1.4299	0.0452	4.32	1.2591	0.0356	4.82	1.1409	0.0287
2.83	2.2065	0.0801	3.33	1.7071	0.0589	3.83	1.4265	0.0450	4.33	1.2564	0.0354	4.83	1.1388	0.0286
2.84	2.1936	0.0795	3.34	1.6997	0.0586	3.84	1.4216	0.0448	4.34	1.2538	0.0353	4.84	1.1368	0.0285
2.85	2.1806	0.0790	3.35	1.6924	0.0582	3.85	1.4170	0.0445	4.35	1.2510	0.0351	4.85	1.1347	0.0284
2.86	2.1682	0.0785	3.36	1.6854	0.0579	3.86	1.4131	0.0443	4.36	1.2483	0.0349	4.86	1.1327	0.0282
2.87	2.1554	0.0780	3.37	1.6784	0.0576	3.87	1.4090	0.0441	4.37	1.2457	0.0348	4.87	1.1306	0.0281
2.88	2.1428	0.0775	3.38	1.6712	0.0573	3.88	1.4059	0.0439	4.38	1.2430	0.0346	4.88	1.1286	0.0280
2.89	2.1305	0.0770	3.39	1.6493	0.0570	3.89	1.4008	0.0437	4.39	1.2404	0.0345	4.89	1.1266	0.0279
2.90	2.1182	0.0765	3.40	1.6574	0.0566	3.90	1.3970	0.0434	4.40	1.2378	0.0343	4.90	1.1245	0.0278
2.91	2.1067	0.0760	3.41	1.6506	0.0563	3.91	1.3930	0.0432	4.41	1.2352	0.0342	4.91	1.1225	0.0277
2.92	2.0945	0.0755	3.42	1.6442	0.0560	3.92	1.3891	0.0430	4.42	1.2326	0.0340	4.92	1.1205	0.0276
2.93	2.0825	0.0750	3.43	1.6373	0.0557	3.93	1.3853	0.0428	4.43	1.2300	0.0339	4.93	1.1185	0.0275
2.94	2.0700	0.0746	3.44	1.6307	0.0554	3.94	1.3814	0.0426	4.44	1.2275	0.0337	4.94	1.1165	0.0273
2.95	2.0595	0.0741	3.45	1.6243	0.0550	3.95	1.3777	0.0424	4.45	1.2250	0.0336	4.95	1.1145	0.0272
2.96	2.0481	0.0736	3.46	1.6180	0.0547	3.96	1.3742	0.0422	4.46	1.2225	0.0334	4.96	1.1126	0.0271
2.97	2.0369	0.0732	3.47	1.6116	0.0544	3.97	1.3703	0.0420	4.47	1.2200	0.0333	4.97	1.1106	0.0270
2.98	2.0257	0.0727	3.48	1.6057	0.0541	3.98	1.3651	0.0418	4.48	1.2176	0.0331	4.98	1.1086	0.0269
2.99	2.0147	0.0722	3.49	1.5992	0.0538	3.99	1.3629	0.0415	4.49	1.2151	0.0330	4.99	1.1066	0.0268
3.00	2.0039	0.0718	3.50	1.5932	0.0535	4.00	1.3595	0.0413	4.50	1.2127	0.0328	5.00	1.1047	0.0267

Table 5.3. Tabulated X-ray and electron atomic scattering factors of copper calculated by DFT (for $2.5 < s \leq 5.0$ Å^{-1}). Cubic-spline was used to interpolate the atomic scattering factors at given s. The given range of s is because the calculation is simulated with an orthorhombic primitive unit cell (lattice constants a=15.88 Å, b=15.93 Å and c=15.98 Å), hence too few numbers of atomic scattering factors were obtained for $s < 0.1$ Å^{-1} for accurate interpolation, while at very large s the WIEN2k program has too many entities of atomic scattering factors to output.

S	X-ray			Electron			s	X-ray			Electron		
	RHF	DFT	Err.	RHF	DFT	Err.		RHF	DFT	Err.	RHF	DFT	Err.
0.00	29.000	29.0000	0.0%	5.600	-	-	0.38	17.145	17.0787	-0.4%	1.965	1.9759	0.6%
0.01	28.977	-	-	5.587	-	-	0.40	16.514	16.4645	-0.3%	1.868	1.8751	0.4%
0.02	28.908	-	-	5.547	-	-	0.42	15.904	15.8716	-0.2%	1.777	1.7812	0.2%
0.03	28.794	-	-	5.482	-	-	0.44	15.318	15.2999	-0.1%	1.691	1.6937	0.2%
0.04	28.640	-	-	5.395	-	-	0.45	15.034	15.0236	-0.1%	1.651	1.6519	0.1%
0.05	28.448	-	-	5.287	-	-	0.46	14.757	14.7520	0.0%	1.611	1.6116	0.0%
0.06	28.223	-	-	5.165	-	-	0.48	14.219	14.2259	0.0%	1.535	1.5347	0.0%
0.07	27.971	-	-	5.029	-	-	0.50	13.707	13.7218	0.1%	1.464	1.4626	-0.1%
0.08	27.694	-	-	4.886	-	-	0.55	12.533	12.5630	0.2%	1.303	1.3005	-0.2%
0.09	27.397	-	-	4.737	-	-	0.60	11.507	11.5449	0.3%	1.163	1.1605	-0.2%
0.10	27.084	27.0584	-0.1%	4.585	4.6470	1.4%	0.65	10.621	10.6627	0.4%	1.041	1.0388	-0.2%
0.11	26.758	26.7200	-0.1%	4.434	4.5097	1.7%	0.70	9.861	9.9051	0.4%	0.935	0.9327	-0.2%
0.12	26.422	26.3698	-0.2%	4.285	4.3716	2.0%	0.80	8.663	8.7044	0.5%	0.761	0.7590	-0.3%
0.13	26.077	26.0102	-0.3%	4.139	4.2341	2.3%	0.90	7.799	7.8294	0.4%	0.626	0.6255	-0.1%
0.14	25.726	25.6438	-0.3%	3.998	4.0983	2.5%	1.00	7.166	7.1828	0.2%	0.523	0.5222	-0.2%
0.15	25.370	25.2722	-0.4%	3.862	3.9653	2.7%	1.10	6.681	6.6894	0.1%	0.442	0.4413	-0.2%
0.16	25.009	24.8973	-0.4%	3.731	3.8356	2.8%	1.20	6.285	6.2878	0.0%	0.378	0.3775	-0.1%
0.17	24.645	24.5202	-0.5%	3.607	3.7099	2.9%	1.30	5.939	5.9369	0.0%	0.327	0.3266	-0.1%
0.18	24.278	24.1421	-0.6%	3.488	3.5885	2.9%	1.40	5.617	5.6123	-0.1%	0.285	0.2856	0.2%
0.19	23.910	23.7638	-0.6%	3.375	3.4715	2.9%	1.50	5.308	5.3017	-0.1%	0.252	0.2521	0.0%
0.20	23.540	23.3860	-0.7%	3.267	3.3591	2.8%	1.60	5.005	4.9989	-0.1%	0.224	0.2244	0.2%
0.22	22.798	22.6343	-0.7%	3.067	3.1478	2.6%	1.70	4.705	4.7004	-0.1%	0.201	0.2012	0.1%
0.24	22.057	21.8906	-0.8%	2.885	2.9541	2.4%	1.80	4.413	4.4083	-0.1%	0.182	0.1817	-0.2%
0.25	21.687	21.5226	-0.8%	2.800	2.8634	2.3%	1.90	4.128	4.1250	-0.1%	0.165	0.1649	-0.1%
0.26	21.319	21.1575	-0.8%	2.719	2.7766	2.1%	2.00	3.855	3.8530	-0.1%	0.150	0.1505	0.3%
0.28	20.589	20.4370	-0.7%	2.568	2.6141	1.8%	2.50	2.721	2.7222	0.0%	0.101	0.1006	-0.4%
0.30	19.869	19.7309	-0.7%	2.428	2.4649	1.5%	3.00	2.001	2.0039	0.1%	0.072	0.0718	-0.3%
0.32	19.162	19.0407	-0.6%	2.299	2.3277	1.3%	3.50	1.590	1.5932	0.2%	0.054	0.0535	-0.8%
0.34	18.472	18.3678	-0.6%	2.180	2.2013	1.0%	4.00	1.358	1.3595	0.1%	0.041	0.0413	0.8%
0.35	18.133	18.0383	-0.5%	2.123	2.1417	0.9%	5.00	1.105	1.1047	0.0%	0.027	0.0267	-1.1%
0.36	17.799	17.7121	-0.5%	2.069	2.0846	0.8%	6.00	0.929	-	-	0.019	-	-

Table 5.4. Comparison of X-ray and electron atomic scattering factors of copper calculated by RHF [9] and interpolated DFT results. Cubic-spline was used to interpolate the DFT atomic scattering factors at given *s*. Only $0.1 \leq s \leq 5.0$ Å^{-1} is listed for DFT, because the calculation is simulated with an orthorhombic primitive unit cell (lattice constants a=15.88 Å, b=15.93 Å and c=15.98 Å), too few numbers of atomic scattering factors were resulted for $s < 0.2$ Å^{-1} for accurate interpolation, while at very large *s* the WIEN2k program has too many entities of atomic scattering factors to output. The X-ray atomic scattering factor at 0.0 Å^{-1} of DFT is also available and listed, while the responding electron atomic scattering factor is omitted, as it represented the value at infinity-distance and is considered inaccurate.

The charge density of a very large orthorhombic primitive unit cell (lattice constants a=15.88 Å, b=15.93 Å and c=15.98 Å, or 30.0, 30.1 and 30.2 in Bohr radius respectively) of copper was calculated by *WIEN2k*, while the lattice constant of a copper FCC unit cell is about 3.6 Å (the distance between the nearest atoms is about 2.5 Å). The DFT calculations employed the GGA PBE functional, using 1,000 k points, and the energy converged to 10^{-4} eV. Structure factors of the large primitive cell of copper were derived from its charge density, and then converted into atomic scattering factors. The atomic scattering factors resulted from the DFT independent copper atom are summarised in Table 5.2 and Table 5.3. As limited by the size and shape of the unit cell, only a few numbers of scattering angles are available for $s < 0.1$ Å^{-1}. At very large *s*, because too many entities of atomic scattering factors are

available, only those with $s \leq 5.0$ Å^{-1} were calculated and listed. Therefore, only the X-ray and electron atomic scattering factors with $0.1 \leq s \leq 5.0$ Å^{-1} were interpolated with cubic-spline and listed.

S	X-ray			Electron			*s*	X-ray			Electron		
	RHF	DFT	Err.	RHF	DFT	Err.		RHF	DFT	Err.	RHF	DFT	Err.
0.0544	28.353	28.3580	0.0%	5.235	5.1975	-0.7%	0.1538	25.234	25.1310	-0.4%	3.812	3.9155	2.7%
0.0626	28.161	28.1635	0.0%	5.131	5.1131	-0.3%	0.1540	25.225	25.1214	-0.4%	3.808	3.9122	2.7%
0.0628	28.156	28.1584	0.0%	5.128	5.1102	-0.4%	0.1626	24.914	24.7983	-0.5%	3.698	3.8021	2.8%
0.0630	28.150	28.1533	0.0%	5.125	5.1073	-0.4%	0.1631	24.896	24.7803	-0.5%	3.692	3.7960	2.8%
0.0886	27.438	27.4267	0.0%	4.758	4.7924	0.7%	0.1636	24.878	24.7620	-0.5%	3.686	3.7899	2.8%
0.0888	27.434	27.4220	0.0%	4.755	4.7909	0.7%	0.1773	24.378	24.2447	-0.5%	3.520	3.6213	2.9%
0.0889	27.429	27.4172	0.0%	4.753	4.7894	0.8%	0.1776	24.367	24.2335	-0.5%	3.516	3.6177	2.9%
0.1039	26.960	26.9318	-0.1%	4.527	4.5887	1.4%	0.1779	24.356	24.2223	-0.6%	3.513	3.6143	2.9%
0.1041	26.951	26.9237	-0.1%	4.523	4.5845	1.4%	0.1853	24.084	23.9416	-0.6%	3.428	3.5263	2.9%
0.1044	26.943	26.9155	-0.1%	4.519	4.5805	1.4%	0.1854	24.078	23.9362	-0.6%	3.426	3.5247	2.9%
0.1087	26.800	26.7636	-0.1%	4.453	4.5266	1.7%	0.1856	24.073	23.9310	-0.6%	3.424	3.5230	2.9%
0.1251	26.245	26.1865	-0.2%	4.209	4.2994	2.1%	0.1859	24.063	23.9203	-0.6%	3.421	3.5197	2.9%
0.1256	26.231	26.1716	-0.2%	4.203	4.2937	2.2%	0.1860	24.058	23.9149	-0.6%	3.420	3.5180	2.9%
0.1260	26.217	26.1565	-0.2%	4.197	4.2879	2.2%	0.1861	24.052	23.9097	-0.6%	3.418	3.5164	2.9%
0.1366	25.845	25.7680	-0.3%	4.045	4.1431	2.4%	0.1877	23.994	23.8497	-0.6%	3.400	3.4980	2.9%
0.1368	25.838	25.7609	-0.3%	4.042	4.1406	2.4%	0.1881	23.979	23.8342	-0.6%	3.396	3.4930	2.9%
0.1370	25.831	25.7539	-0.3%	4.039	4.1380	2.4%	0.1883	23.971	23.8263	-0.6%	3.393	3.4905	2.9%
0.1400	25.726	25.6433	-0.3%	3.998	4.0982	2.5%	0.1886	23.963	23.8185	-0.6%	3.391	3.4881	2.9%
0.1401	25.722	25.6398	-0.3%	3.997	4.0969	2.5%	0.1890	23.948	23.8027	-0.6%	3.386	3.4833	2.9%
0.1403	25.716	25.6330	-0.3%	3.994	4.0944	2.5%	0.1979	23.616	23.4638	-0.6%	3.289	3.3818	2.8%
0.1405	25.709	25.6261	-0.3%	3.991	4.0919	2.5%	0.1980	23.614	23.4613	-0.6%	3.288	3.3811	2.8%
0.1407	25.702	25.6192	-0.3%	3.989	4.0893	2.5%	0.1985	23.597	23.4432	-0.7%	3.283	3.3764	2.8%
0.1408	25.699	25.6158	-0.3%	3.988	4.0881	2.5%	0.1986	23.592	23.4379	-0.7%	3.282	3.3752	2.8%
0.1535	25.243	25.1406	-0.4%	3.815	3.9188	2.7%	0.1991	23.572	23.4183	-0.7%	3.276	3.3691	2.8%

Table 5.5. Comparison of X-ray and electron atomic scattering factors of copper calculated by DFT and interpolated RHF [9] results for $s < 0.2$ Å^{-1}. Cubic-spline was used to interpolate the RHF atomic scattering factors at given *s*. Because the DFT calculation is simulated with an orthorhombic primitive unit cell (lattice constants a=15.88 Å, b=15.93 Å and c=15.98 Å), only a few numbers of atomic scattering factors are available for $s < 0.1$ Å^{-1}.

The comparisons of copper X-ray and electron atomic scattering factors calculated by RHF and DFT are listed in Table 5.4 and Table 5.5. Overall, better agreement is observed at higher *s* between the RHF and DFT atomic scattering factors, which indicates that the charge distributions near the atomic core are similar and is as expected. In Table 5.4, the DFT results are cubic-splined to compare with the tabulated RHF values. Due to the limited numbers of scattering factors at very low scattering angles by DFT calculations, no DFT value $0 < s < 0.1$ Å^{-1} is listed as interpolation would be unreliable. The DFT electron atomic scattering factor at 0 Å^{-1} is omitted, as it represented the value at infinity-distance and is considered inaccurate by this simulation. In Table 5.5, comparison is shown with DFT atomic

scattering factors and cubic-splined RHF values for $s < 0.2$ Å^{-1}, where the DFT atomic scattering factors with $s < 0.1$ Å^{-1} are shown without interpolation. The largest differences are observed at $0.1 \leq s \leq 0.4$ Å^{-1}, where the DFT X-ray and electron atomic scattering factors can be about 1% and 3% respectively offset from the RHF values. For $s < 0.1$ Å^{-1}, the differences are relatively smaller, while for $s > 1$ Å^{-1}, the differences are very small. Especially for $s > 2$ Å^{-1}, the offsets are subtle when errors arising from rounding are considered. These may suggest the differences between the RHF and DFT charge distribution simulations of copper IAM: 1) are very small for the core electrons (close values for $s > 2$ Å^{-1}), 2) peak at the valence electrons (significant offsets at $0.1 \leq s \leq 0.4$ Å^{-1}), and 3) decrease at distances far from the atom core (relatively smaller differences at $s < 0.1$ Å^{-1}). These trends are expected and predicted by Doyle and Turner for calculations including the correlation energy [9]. Although the peak difference of X-ray structure factors is smaller than 1% between the RHF and DFT copper IAMs, considering the differences between the QCBED and IAM X-ray structure factors are only within 2%, a different IAM can lead to a significant change for bonding determination in copper. Moreover, as electron IAM structure factors, which consist of up to 3% offsets, are used during QCBED refinements, a different IAM would also considerably alter the QCBED results.

Apart from the likely impact on low-order structure factor measurement and bonding determination, an inaccurate IAM will also affect other experiments, when higher-order structure factors are assumed to equal the IAM values without further refinements. For example, if the Debye-Waller factor, atomic positions or lattice parameter of copper are refined with QCBED using the RHF-IAM, which generally also only refine the low-order structure factors, their accuracy will be compromised by the higher-order structure factors fixed as the RHF-IAM values.

More detailed discussion of impacts caused by altering the IAM to bonding determination will be given in the next chapter, while the DFT simulation of a regular copper unit cell and QCBED results are also considered.

Chapter 5. The role of the independent-atom model

Chapter 6. Final results of bonding in copper

In the previous chapter, the role and importance of IAM in QCBED analysis and bonding determination were discussed. This leads to the conclusion that bonding of copper derived using the DFT-IAM is the most reliable. In this chapter, copper structure factors and bonding calculated with DFT-IAM will be focused on and compared to literature values for discussion. Qualitative linking between copper's bonding distribution and its mechanical properties will also be discussed, which can justify the relationship between copper's relatively isotropic elastic constants and its bonding distribution.

6.1. Structure factors of copper by QCBED

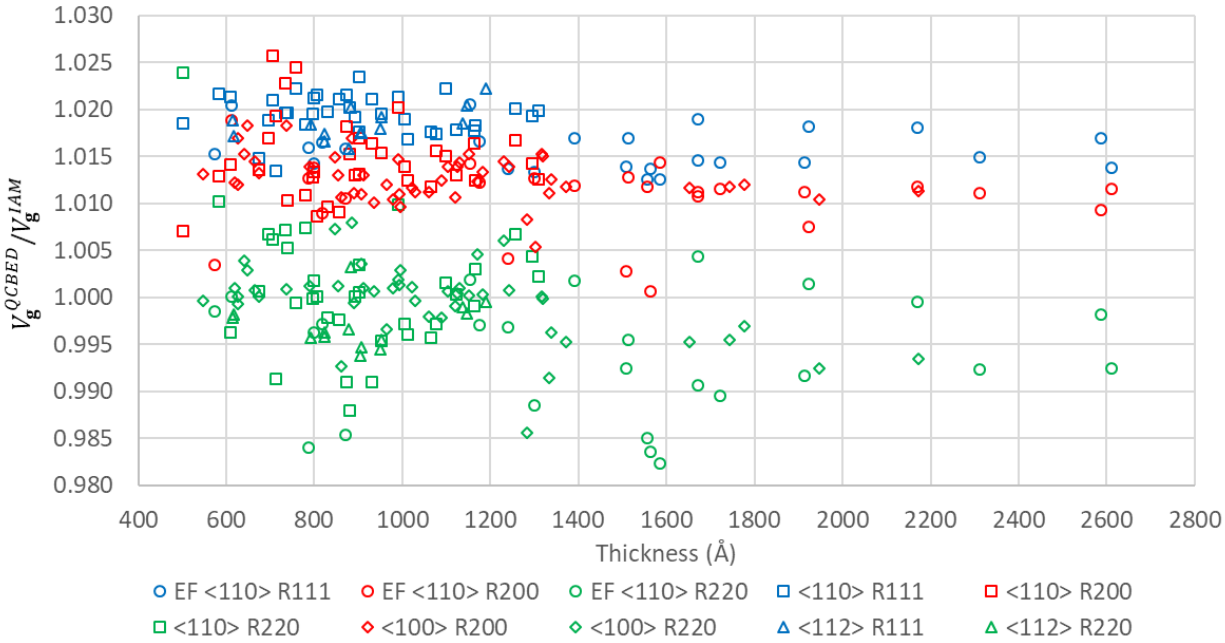


Figure 6.1. QCBED-refined copper structure factors as thickness dependence. The refined electron structure factors V_{111}, V_{200} and V_{220} are plotted as ratios to the IAM by DFT (denoted as "R111", "R200" & "R220" respectively) for 122 sets of copper CBED patterns from <110>, <100> and <112> axes against their thickness. Energy-filtered patterns are available from <110> and denoted as "EF <110>".

hkl	s (Å^{-1})	IAM		XRD	Band Theory	γ-ray	Pendellösung	DFT		QCBED		
		DFT	RHF	Temkin *et al.* (1972)	Bagayoko (1980)	Schneider (1981)	Takama & Sato (1982)	Friis *et al.* (2005)	This work	Saunders *et al.* (1999)	Friis *et al.* (2005)	This work
111	0.2404	21.87	22.04	21.9(2)	21.68	21.51(5)	21.80(6)	21.70	21.77	21.78(2)	21.69(3)	21.74(1)
200	0.2776	20.52	20.68	20.4(2)	20.35	20.22(4)	20.3(1)	20.38	20.45	20.44(2)	20.44(2)	20.41(2)
220	0.3926	16.69	16.75	16.7(2)	16.62	16.45(5)	16.75(8)	16.67	16.76	16.7(1)	16.68(2)	16.70(5)
311	0.4603	14.74	14.75	14.7(2)	14.70	14.70(4)	14.74(4)	14.75	14.85	14.8(1)	14.73(1)	14.79(9)
222	0.4808	14.20	14.20	14.2(2)	14.17	14.22(5)	14.36(6)	14.21	14.32		14.24(7)	14.2(1)
400	0.5552	12.45	12.42	12.3(2)	12.42	12.42(6)	12.46(6)	12.48	12.57		12.45(9)	12.2(3)

Table 6.1. Comparison of low-order structure factors of copper by this work and previous studies (selected as they are either the most cited in bonding studies of copper or the most recent publications of the techniques [4, 7, 11, 12, 14, 16, 18]). Both the DFT and RHF [9] IAMs are interpolated with cubic-spline.

The QCBED-refined electron structure factors are summarised in Figure 6.1 as ratios to the DFT-IAM, which was used as the input atomic scattering factors for QCBED refinements. The thickness of the copper specimens where the CBED patterns were collected spans from about 500 to 2600 Å. No thickness dependence is observed. Although the mean values of the refined structure factors from various axes and pattern geometries are slightly different, they are all within uncertainties. The slightly

different mean values are expected as different Bragg conditions will cause various structure factor sensitivities. When comparing the energy-filtered and -unfiltered results, no significant difference is observed. This further confirms the validity of ADQCBED. Some patterns have yielded structure factors significantly different to most of the other patterns, maybe due to sample defects or acquisition errors. The final X-ray structure factors of QCBED are listed in Table 6.1. When examining the difference between the QCBED and the DFT-IAM structure factors, it can be seen that ΔF_{220} and other $\Delta F_{\mathbf{g}}$ of higher order are very small and well within the errors. This indicates only F_{111} and F_{200} are significant to the bonding in copper, which is as expected for an FCC metal [5, 6].

6.2. QCBED results using alternative refinement recipes

Various combinations of different IAMs and with/without the angular differentiation were used to refine the CBED patterns. Refinement results of 1) using the RHF-IAM [9] with angular differentiation, 2) using the DFT-IAM with angular differentiation and 3) using the DFT-IAM without differentiation are listed in Table 6.2.

hkl	***s* (Å^{-1})**	**IAM**		**1) Using RHF-IAM & differentiation**		**2) Using DFT-IAM & differentiation**		**3) Using DFT-IAM without differentiation**	
		$F_{\mathbf{g}}^{RHF-IAM}$	$F_{\mathbf{g}}^{DFT-IAM}$	$F_{\mathbf{g}}^{QCBED-1}$	$\Delta F_{\mathbf{g}}^{QCBED-1}$	$F_{\mathbf{g}}^{QCBED-2}$	$\Delta F_{\mathbf{g}}^{QCBED-2}$	$F_{\mathbf{g}}^{QCBED-3}$	$\Delta F_{\mathbf{g}}^{QCBED-3}$
111	0.2404	22.04	21.87	21.75(2)	-0.29(2)	21.74(1)	-0.13(1)	21.78(7)	-0.09(7)
200	0.2776	20.68	20.52	20.42(3)	-0.26(3)	20.41(2)	-0.11(2)	20.40(6)	-0.12(6)
220	0.3926	16.75	16.69	16.71(5)	-0.04(5)	16.70(5)	0.01(5)	16.7(1)	0.0(1)
311	0.4603	14.75	14.74			14.79(9)	0.05(9)		
222	0.4808	14.2	14.2			14.2(1)	-0.0(1)		
400	0.5552	12.42	12.45			12.2(3)	-0.3(3)		

Table 6.2. Comparison of low-order structure factors in copper refined using different QCBED refinement recipes. The CBED patterns were refined: 1) using the RHF-IAM [9] with angular differentiation, 2) using the DFT-IAM with angular differentiation and 3) using the DFT-IAM without differentiation, respectively. F_{311}, ΔF_{222} and F_{400} were not refined for Methods 1) and 3) due to large uncertainties in these relatively higher-order structure factors. The refined absolute structure factors suggest that QCBED refinements may be slightly altered by using a different IAM, and that the uncertainties are significantly reduced when the angular differentiation is applied. However, for bonding determination where $F_{\mathbf{g}}^{IAM}$ are subtracted from $F_{\mathbf{g}}^{QCBED}$ to yield $\Delta F_{\mathbf{g}}^{QCBED}$, different IAMs have led to very different $\Delta F_{\mathbf{g}}^{QCBED}$, since the differences between the $F_{\mathbf{g}}^{RHF-IAM}$ and $F_{\mathbf{g}}^{DFT-IAM}$ are as large as 50% of the $\Delta F_{\mathbf{g}}^{QCBED}$.

QCBED structure factors $F_{\mathbf{g}}^{QCBED}$ of copper refined using the RHF-IAM (cubic-splined from tabulated atomic scattering factors by Doyle and Turner [58]) are only slightly different when compared to the

results of using DFT-IAM. As mentioned in Section 1.4, near-zone-axis QCBED refinements are highly sensitive to low-order structure factors, while the higher-order structure factors are assumed to be identical to the input IAM and fixed. Because the corresponding IAM structure factors $F_{\mathbf{g}}^{IAM}$ of copper at higher scattering angles of DFT-IAM and RHF-IAM are relatively similar, the (low-order) $F_{\mathbf{g}}^{QCBED}$ refined by using different IAMs respectively are close to each other as well. However, to calculate the bonding charge density distribution in copper, the IAM charge density is to be subtracted from the QCBED-measured charge density. While comparing to the bonding structure factors $\Delta F_{\mathbf{g}}$, the differences between the $F_{\mathbf{g}}^{RHF-IAM}$ and $F_{\mathbf{g}}^{DFT-IAM}$ are significant, which are as large as 50% of the $\Delta F_{\mathbf{g}}$ measured by QCBED. Such significant changes between different IAMs lead to obvious changes to the $\Delta F_{\mathbf{g}}$ as well as to the bonding density distributions.

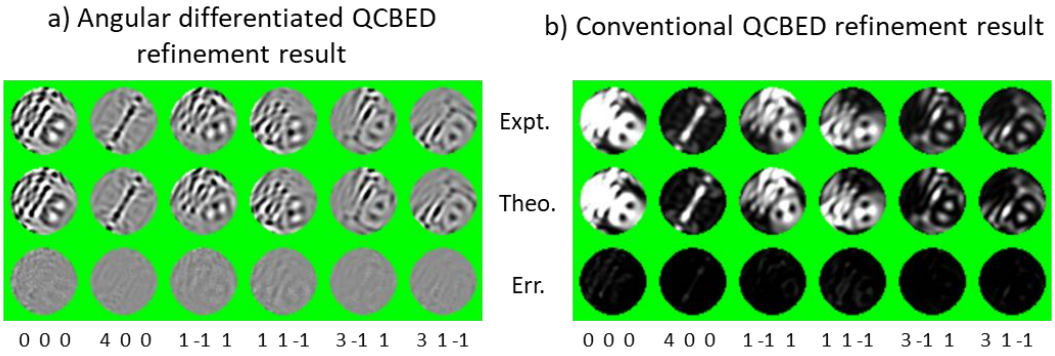


Figure 6.2. The QCBED refinement results a) with and b) without angular differentiation (conventional) of an energy-filtered CBED pattern of copper collected near the [110] zone axis. The collected reflection discs are denoted as "Expt.", calculated discs with refined structure factors as "Theo.", and their residuals as "Err.". Intensity distributions in a) are changing more rapidly than in b), which leads to stronger constraints during refinement using the angular differentiation.

The refined results of an energy-filtered CBED pattern of copper are shown in Figure 6.2, as an example showing the differences between angular differentiated and conventional QCBED refinements. Comparing the conventional refinement results without differentiation to other results in Table 6.2, it is shown that the uncertainties of the refined lowest-order structure factors were decreased from about 0.3% to 0.1% via the use of angular differentiation. This is expected, as discussed in Section 2.4 and reported in Ref. [96], because angular differentiation can notably reduce the diffuse background caused by inelastic scattering for CBED patterns collected without an energy filter, as well as generating more rapidly changing gradients within the reflection discs which leads to better constraints during pattern-matching. The reduced uncertainties have also enabled the assessments of relatively higher-order structure factors, F_{311}, F_{222} and F_{400}, despite the CBED patterns were

collected near low-order zone-axes and most reliable for the lowest-order structure factor measurements.

6.3. Structure factors of copper by DFT

hkl	DFT with relaxed lattice parameter (3.6320 Å)						DFT with fixed lattice parameter (3.6024 Å)					
	s (Å^{-1})	F_g^{Relax}	$F_g^{RHF-IAM}$	$F_g^{DFT-IAM}$	ΔF_g^{R-RHF}	ΔF_g^{R-DFT}	s (Å^{-1})	F_g^{Fix}	$F_g^{RHF-IAM}$	$F_g^{DFT-IAM}$	ΔF_g^{F-RHF}	ΔF_g^{F-DFT}
111	0.2384	21.768	22.118	21.947	-0.351	-0.180	0.2404	21.694	22.046	21.875	-0.352	-0.181
200	0.2753	20.452	20.771	20.604	-0.318	-0.151	0.2776	20.371	20.689	20.523	-0.318	-0.152
220	0.3894	16.763	16.841	16.788	-0.078	-0.025	0.3926	16.665	16.739	16.690	-0.074	-0.025
311	0.4566	14.849	14.841	14.844	0.008	0.005	0.4603	14.749	14.737	14.743	0.011	0.006
222	0.4769	14.315	14.293	14.306	0.023	0.009	0.4808	14.215	14.190	14.205	0.025	0.010
400	0.5507	12.573	12.519	12.548	0.053	0.025	0.5552	12.476	12.421	12.450	0.055	0.025
331	0.6001	11.558	11.513	11.543	0.045	0.015	0.6050	11.465	11.420	11.451	0.046	0.015
420	0.6157	11.273	11.223	11.254	0.051	0.019	0.6207	11.183	11.132	11.163	0.051	0.020
422	0.6744	10.290	10.243	10.277	0.048	0.013	0.6800	10.207	10.159	10.194	0.048	0.013
511	0.7153	9.714	9.657	9.695	0.057	0.019	0.7212	9.637	9.578	9.618	0.058	0.019
333	0.7153	9.700	9.657	9.695	0.044	0.005	0.7212	9.623	9.578	9.618	0.044	0.005
440	0.7788	8.933	8.886	8.929	0.047	0.004	0.7852	8.864	8.816	8.860	0.047	0.004
531	0.8144	8.566	8.517	8.560	0.049	0.006	0.8211	8.501	8.453	8.496	0.049	0.005
600	0.8260	8.466	8.407	8.450	0.059	0.016	0.8328	8.403	8.344	8.387	0.059	0.016
442	0.8260	8.450	8.407	8.450	0.043	0.000	0.8328	8.387	8.344	8.387	0.043	0.000
620	0.8707	8.068	8.018	8.058	0.051	0.010	0.8778	8.010	7.961	8.000	0.050	0.010
533	0.9027	7.807	7.772	7.809	0.035	-0.002	0.9101	7.752	7.719	7.754	0.033	-0.002
622	0.9132	7.738	7.697	7.733	0.041	0.005	0.9207	7.684	7.645	7.679	0.039	0.005
444	0.9538	7.452	7.429	7.458	0.023	-0.006	0.9616	7.403	7.381	7.409	0.022	-0.007
711	0.9831	7.290	7.254	7.279	0.035	0.010	0.9912	7.243	7.209	7.233	0.034	0.010
551	0.9831	7.277	7.254	7.279	0.022	-0.002	0.9912	7.230	7.209	7.233	0.021	-0.003
640	0.9927	7.224	7.201	7.224	0.023	0.000	1.0009	7.178	7.156	7.178	0.022	0.000

Table 6.3. Structure factors and bonding of copper calculated by DFT. Two sets of DFT charge density calculations of copper were performed: 1) the lattice parameter was set to relax according to energy and 3.6320 Å was the relaxed value (structure factors listed as F_g^{Relax}), and 2) the lattice parameter was fixed to be 3.6024 Å (structure factors listed as F_g^{Fix}. This lattice parameter is determined from extrapolation of experimental data [81]). Because the two sets of DFT calculations have different lattice parameters, their scattering angles and IAM structure factors are also different and listed respectively. Both the structure factors of RHF-IAM ($F_g^{RHF-IAM}$) derived from Doyle and Turner's publication and the DFT-IAM ($F_g^{DFT-IAM}$) in this work were listed, as well as their differences (ΔF_g^{RHF} and ΔF_g^{DFT}) compared to the corresponding F_g^{Relax} and F_g^{Fix}. F_g^{Relax} and F_g^{Fix} have different values for different reflections with identical scattering angles, which suggest the DFT calculations have yielded aspherical charge distributions. However, both IAMs have respectively constant values for identical scattering angles, because they are monotonic curves against the scattering angles and assume to have spherical charge distributions. Although the absolute values of F_g^{Relax} and F_g^{Fix} are different, the corresponding ΔF_g^{R-RHF} and ΔF_g^{F-RHF}, and ΔF_g^{F-DFT} and ΔF_g^{F-DFT} respectively are very close. This suggests different lattice parameters have small contributions to the bonding differences.

Chapter 6. Final results of bonding in copper

As comparisons to the QCBED and DFT-IAM results, charge density calculations of copper by DFT were also performed (by Dr. Andrew E. Smith), using the same functional (GGA PBE) used for the DFT-IAM calculations. Two sets of DFT charge density calculations of copper were performed: 1) the lattice parameter was set to vary, resulting in the lowest possible energy state, and the calculations finished with a lattice parameter of 3.6320 Å, and 2) the lattice parameter was fixed as 3.6024 Å (determined from experimental data by extrapolation [81]), resulting in a higher energy state compared to 1). Note that when different lattice parameters are used, the scattering angles of structure factors are also changed.

Comparisons between DFT results and IAM values are summarised in Table 6.3. Because the two sets of DFT calculations have different lattice parameters, the scattering angles and IAM structure factors are also different and listed respectively. Both the IAM derived from the atomic scattering factors of Doyle and Turner ($F_g^{RHF-IAM}$) and the IAM calculated by DFT ($F_g^{DFT-IAM}$) in this work are presented, as well as their differences compared to the F_g^{Relax} and F_g^{Fix} (differences listed as ΔF_g^{R-RHF} and ΔF_g^{F-RHF}, and ΔF_g^{R-DFT} and ΔF_g^{F-DFT} respectively). Because both DFT and DFT-IAM structure factors are calculated with similar functional configurations, ΔF_g^{DFT} are mostly contributed by bonding effects in the valance electrons, resulting in significant ΔF_{111}^{DFT} and ΔF_{200}^{DFT} for both lattice parameters. Noticeable ΔF_g^{DFT} also exist at higher-order structure factors, which are caused by the aspherical/spherical charge distribution difference between the DFT (F_g^{Relax} and F_g^{Fix}) and DFT-IAM ($F_g^{DFT-IAM}$). For example, the F_{511}^{Relax} / F_{333}^{Relax} and F_{511}^{Fix} / F_{333}^{Fix} have different values respectively (representing aspherical charge distribution), while the $F_{511}^{DFT-IAM}$/$F_{333}^{DFT-IAM}$ are identical respectively for different lattice parameters (representing spherical charge distribution). The raw data of DFT-IAM calculations, which have used the same GGA functional, also have aspherical features. However, the asphericity was removed, because conventional IAM applications (e.g. the QCBED pattern-matching program) can only accept spherical IAMs. Therefore, both the RHF-IAM and DFT-IAM have respectively constant values for identical scattering angles. Although the absolute values of F_g^{Relax} and F_g^{Fix} are different, the corresponding ΔF_g^{R-RHF} and ΔF_g^{F-RHF}, and ΔF_g^{F-DFT} and ΔF_g^{F-DFT} respectively are very close. This suggests different lattice parameters have small contributions to the bonding differences, which is also observed and suggested in [5]. 3D visualisation of the bonding by DFT calculations are shown in Figure 6.3. The figure has demonstrated that different lattice parameters have almost no effect on the bonding determination, while alteration of IAM has a very strong impact.

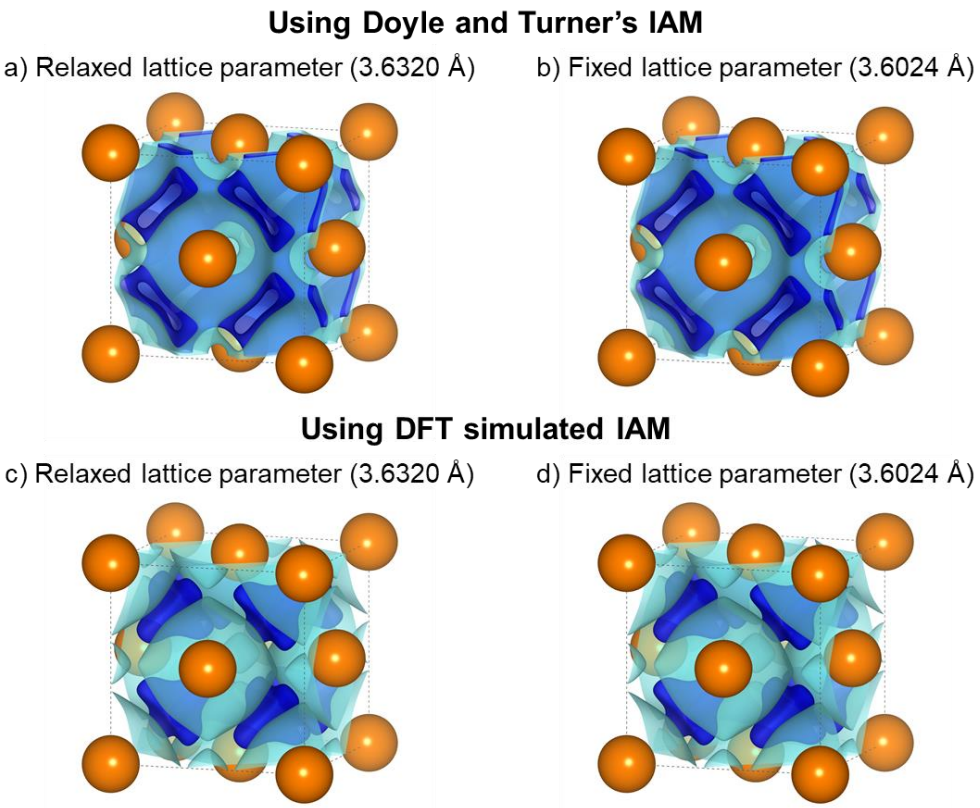


Figure 6.3. Bonding in copper calculated by DFT. Two sets of DFT charge density calculations of copper were performed: 1) the lattice parameter was set to relax according to energy and 3.6320 Å was the relaxed value, and 2) the lattice parameter was fixed to be 3.6024 Å (determined from extrapolation of experimental data [81]). The bonding charge distributions were plotted using only the lowest three structure factors. The conventional RHF-IAM derived from Doyle and Turner's atomic scattering factors and the DFT-IAM were used for bonding determination, respectively. No obvious difference is found between different lattice parameter configurations. Using the RHF-IAM has yielded a tetrahedral-bridging type of bonding, while using the DFT-IAM has yielded a more tetrahedral-like bonding.

hkl	**This work**			**Saunders *et al.* (1999)**		**Friis *et al.* (2005)**		**Sang *et al.* (2013)**	
	DFT (relaxed)	**DFT (fixed)**	**QCBED**	**DFT**	**QCBED**	**DFT**	**QCBED**	**DFT**	**QCBED**
111	21.768	21.694	21.74(1)	21.71	21.78(2)	21.70	21.69(3)	21.794	21.80(3)
200	20.452	20.371	20.41(2)	20.37	20.44(2)	20.38	20.44(2)	20.450	20.45(4)
220	16.763	16.665	16.70(5)	16.66	16.7(1)	16.67	16.68(2)		
311	14.849	14.749	14.79(9)	14.75	14.8(1)	14.75	14.73(1)		
222	14.315	14.215	14.2(1)			14.21	14.24(7)		
400	12.573	12.476	12.2(3)			12.48	12.45(9)		

Table 6.4. Low-order structure factors of copper calculated by DFT. Two sets of DFT charge density calculations (using the GGA functional) of copper were performed in this work: 1) the lattice parameter was fixed as 3.6024 Å (determined from extrapolation of experimental data [81]). Results from some recent studies using DFT and QCBED are also included for comparison, and 2) the lattice parameter was set to relax according to energy and 3.6320 Å was the relaxed value. Note that the LDA + U calculation from Sang *et al.* (2013) was selected as their DFT values, which has the best agreement to their QCBED result.

Chapter 6. Final results of bonding in copper

Comparisons between the DFT and QCBED results, as well as values from previous studies, are summarised in Table 6.4. DFT results by Saunders *et al.* [4] were calculated using the WIEN95 program. Although no further details of DFT calculations were reported, their DFT values are very close to Friis *et al.*'s [7] DFT results and our DFT results using a fixed lattice parameter, which suggests similar configurations were employed. Friis *et al.*'s DFT results were calculated using the GGA functional. The same set of results were also reported in [31], where more precise scattering angles were presented, suggesting 3.6054 Å was used as the lattice parameter. Friis *et al.*'s DFT structure factors are very close to their QCBED values (except F_{200}), although their QCBED F_{111} is different from that of other QCBED studies. In Sang *et al.*'s publication in 2013 [5], various functionals were tested against their QCBED results, using 3.610 Å as the fixed lattice parameter. Among their DFT calculations of copper, the LDA + U (U = 0.5 Ry) functional resulted in best agreements to their QCBED structure factors. For DFT calculations by this work, using relaxed lattice parameter generally result better agreement to QCBED studies on F_{111} and F_{200}, while using a fixed experimental lattice parameter yields structure factors closer to QCBED values at higher scattering angles. Although for some other metals, like aluminium, excellent agreement can be found between experimental and DFT results using the GGA functional [6], structure factors of copper are likely to differ from the experimental values using this method. This is because the GGA functionals are reported to have difficulties evaluating materials with strongly localised *d* or *f* electrons [5]. As reported in Sang *et al.*'s paper [5], after evaluation it is possible to conclude a specific functional to obtain near-perfect fitting to the experimental results. However, as discussed in Chapter 5, an accurate IAM is prerequisite for the experimental structure factors of copper determined by QCBED. For consistency, the IAM simulated by DFT should employ the same functional as well. Hence, extra refinement loops to relax the QCBED, DFT and DFT-IAM calculations may be applied to determine the bonding in copper with improved accuracy and consistency, if a different functional (other than the GGA PBE functional which is the most universal) is used.

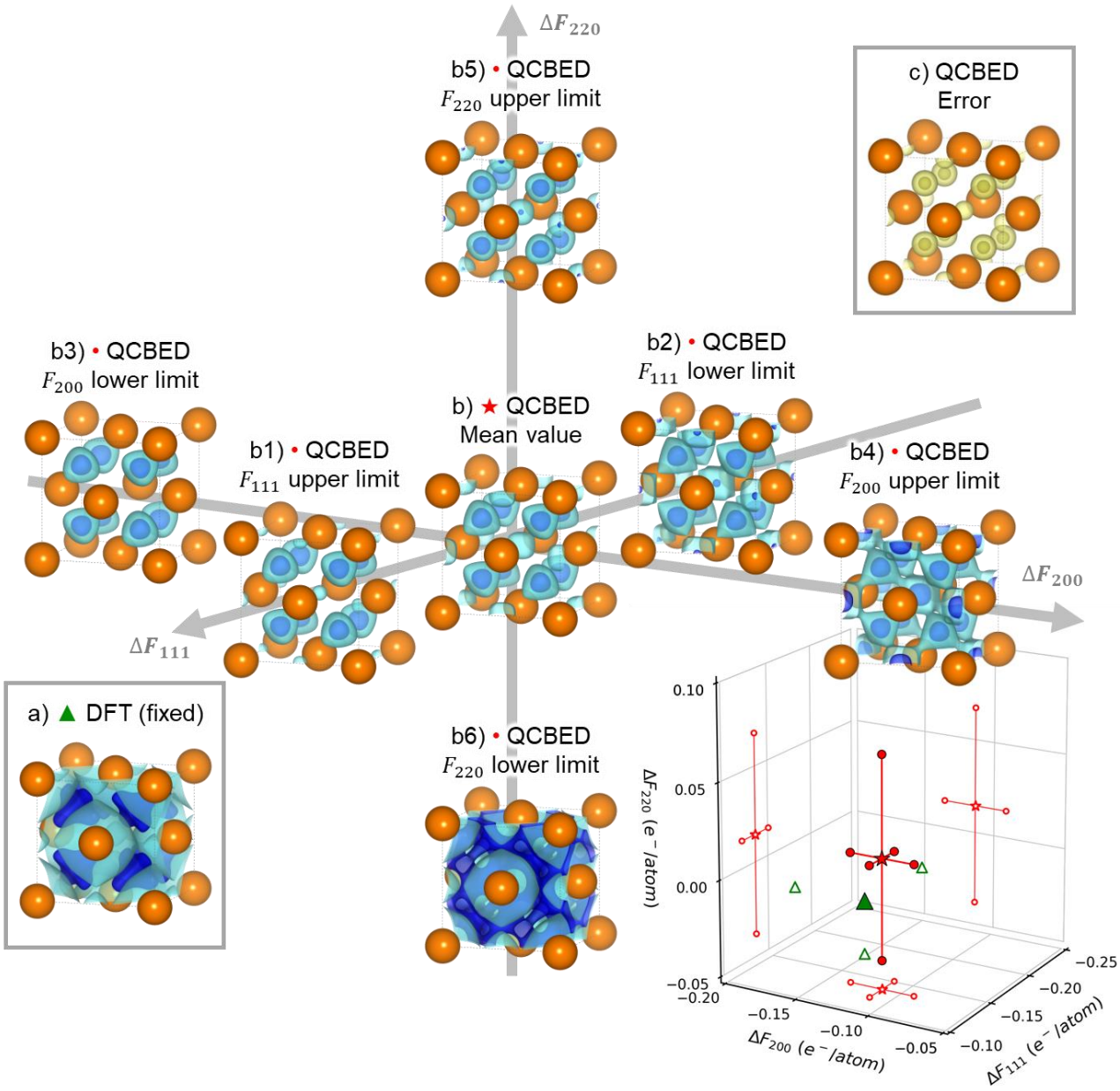


Figure 6.4. Comparison of bonding in copper between the DFT calculation and the QCBED results of this work including error bars. The bonding charge distributions are plotted as: a) DFT with a fixed lattice parameter, b) QCBED mean value, b1) – b6) QCBED at the extremities of each error bar, and c) the error of QCBED results listed in **Table 6.4**. Only the three lowest structure factors are included. The DFT-IAM is used. The iso-surfaces are plotted at 50% (light blue) and 80% (dark blue) of the maximum bonding electron density of each result; except for c) the error, where the iso-surfaces are plotted at the same levels of b) the mean and coloured in yellow. The ΔF_{111}, ΔF_{200} and ΔF_{220} calculated by DFT and QCBED are also plotted (inset) for comparison. Although the absolute $F_{\mathbf{g}}$ are different for the DFT and QCBED values, similar bonding morphologies may be yielded within the errors of QCBED results. The error distribution in c), which has a much smaller spreading than the QCBED charge distribution, indicates the reliability of the measurements.

Comparison of bonding in copper between the DFT and QCBED results is plotted in Figure 6.4. As discussed in previous paragraphs, the DFT simulations are likely to be different from the QCBED measurements for copper, as long as the GGA PBE functional is used (which leads to inaccurate calculations of strongly localised *d* electrons). The low-order structure factors of DFT and QCBED's mean values are noticeably different. However, similar ratios between the F_{111}, F_{200} and F_{220} to the DFT values can be obtained within the uncertainties of QCBED measurements. This suggests similar bonding morphologies in copper may be obtained between the current QCBED and DFT results, although the magnitudes of the bonding are different. The errors of QCBED measurements for this work listed in Table 6.4 are also plotted as charge distribution in Figure 6.4c). The light and dark shades of yellow are errors corresponding to the light and dark blue iso-surfaces plotted for the mean values in Figure 6.4b), respectively. The much smaller spreading of the error distribution indicates that the QCBED measurements have sufficient precisions for a reliable bonding determination.

6.4. Comparison with previous studies

A summary of bonding in copper obtained by previous studies and this work is shown in Figure 6.5. The charge density at the tetrahedral sites, octahedral centre and bridging sites of copper's unit cell are plotted to summarise the morphologies of bonding distributions. The ratios of bonding densities of the three bonding sites for each dataset indicate where the bonding electrons are mostly localised. For example, the two XRD data points that depart from the others have indicated that the bonding is mostly localised in the bridging sites, because they have very high electron densities in the bridging sites compared to the tetrahedral and octahedral sites. A zoom-in graph which contains most results from more recent studies is plotted as Figure 6.6, showing the cluster of more recent results including those from this work. The shifting caused by replacing the RHF-IAM with the DFT-IAM is constant (except for the QCBED results of this work, since the measured charge density is also altered). The electron density differences are about 0.02 $e^-/\text{Å}^3$ at the tetrahedral sites, about 0.02 $e^-/\text{Å}^3$ at the octahedral centres and increased by about 0.05 $e^-/\text{Å}^3$ at the bridging sites. An obvious linear aggregation can be observed, where earlier results are located on the strong-bridging corner departing from more recent results. Moreover, the result from this work is surrounded by values from recent studies closely. Generally, these results from previously recent studies are distributed around the results from the current work. From this magnified graph, it is obvious that the bonding of different studies has become significantly more octahedral-like and less bridging-like.

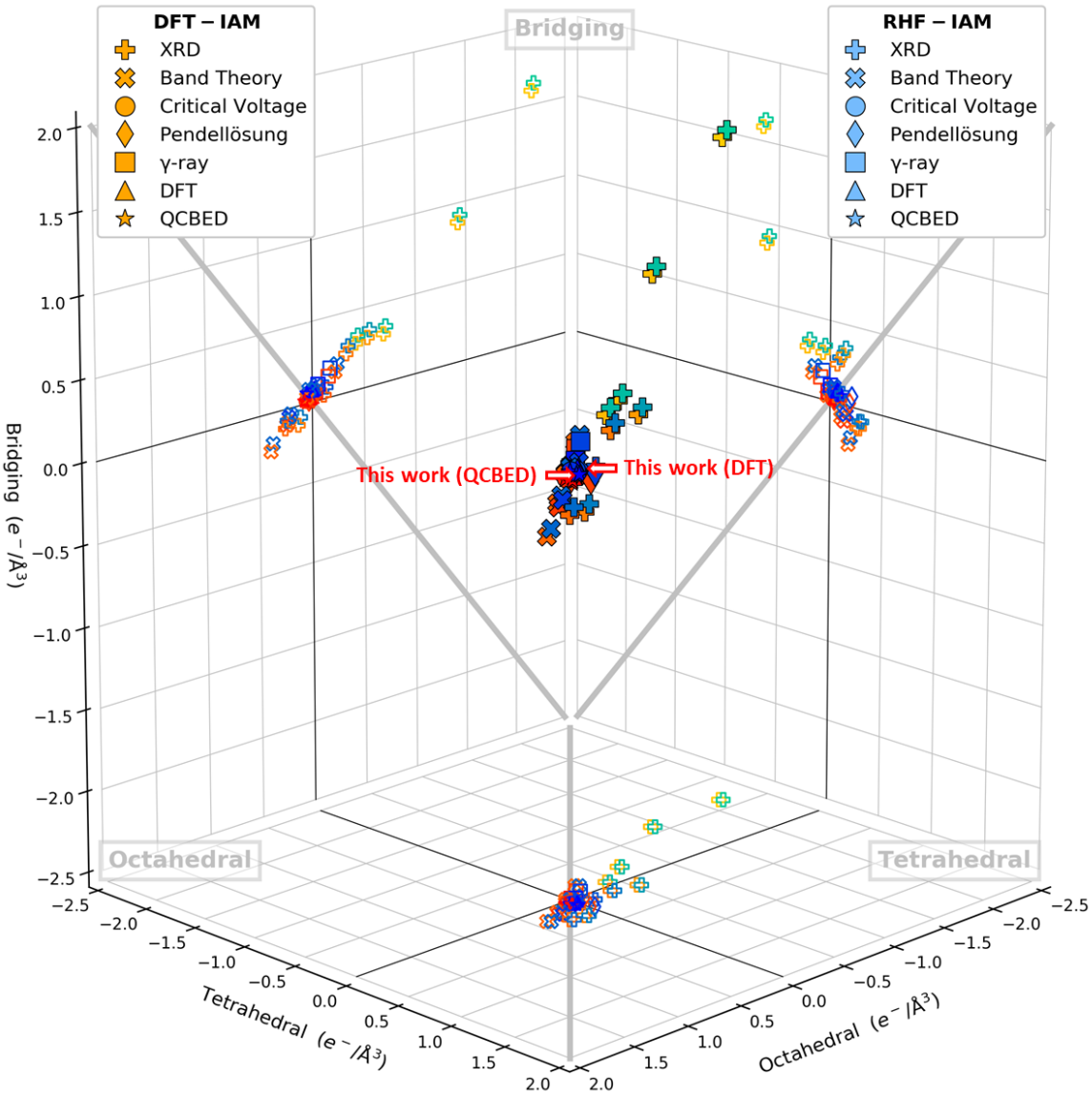


Figure 6.5. Summary of bonding in copper for all studies (including this work). The charge densities at the tetrahedral sites, octahedral centre and bridging sites of copper's unit cell are used to estimate the shapes of bonding distributions. Higher density indicates stronger electron density localisation at a certain site, while density < 0 $e^-/\text{Å}^3$ indicates the presence of anti-bonding. Results of different studies are grouped by their techniques. Shades from orange to red are used to indicate the publication years from 1930 to 2005 using the DFT-IAM. The light to dark blue shades are used to show the corresponding studies from using the RHF-IAM. Experimental techniques include XRD (9 sets) [18, 39, 43, 45, 47-49, 80], critical voltages (2 sets) [4, 12], Pendellösung (2 sets) [14, 32], γ-ray (2 sets) [11, 15] and QCBED (3 sets including the current work) [4, 7]. Theoretical calculations include band theory (7 sets) [13, 16, 39, 41] and DFT (3 sets including the current work) [4, 7]. The bonding charge densities are calculated with only the three lowest-order structure factors. The results of more recent studies are clustered with those of the current work. The electron density difference is about 0.02 $e^-/\text{Å}^3$ at the tetrahedral site, about 0.02 $e^-/\text{Å}^3$ at the octahedral centres and about 0.05 $e^-/\text{Å}^3$ at the bridging sites. This constant shifting leads to less bridging-like bonding conclusions for all previous studies and the current work.

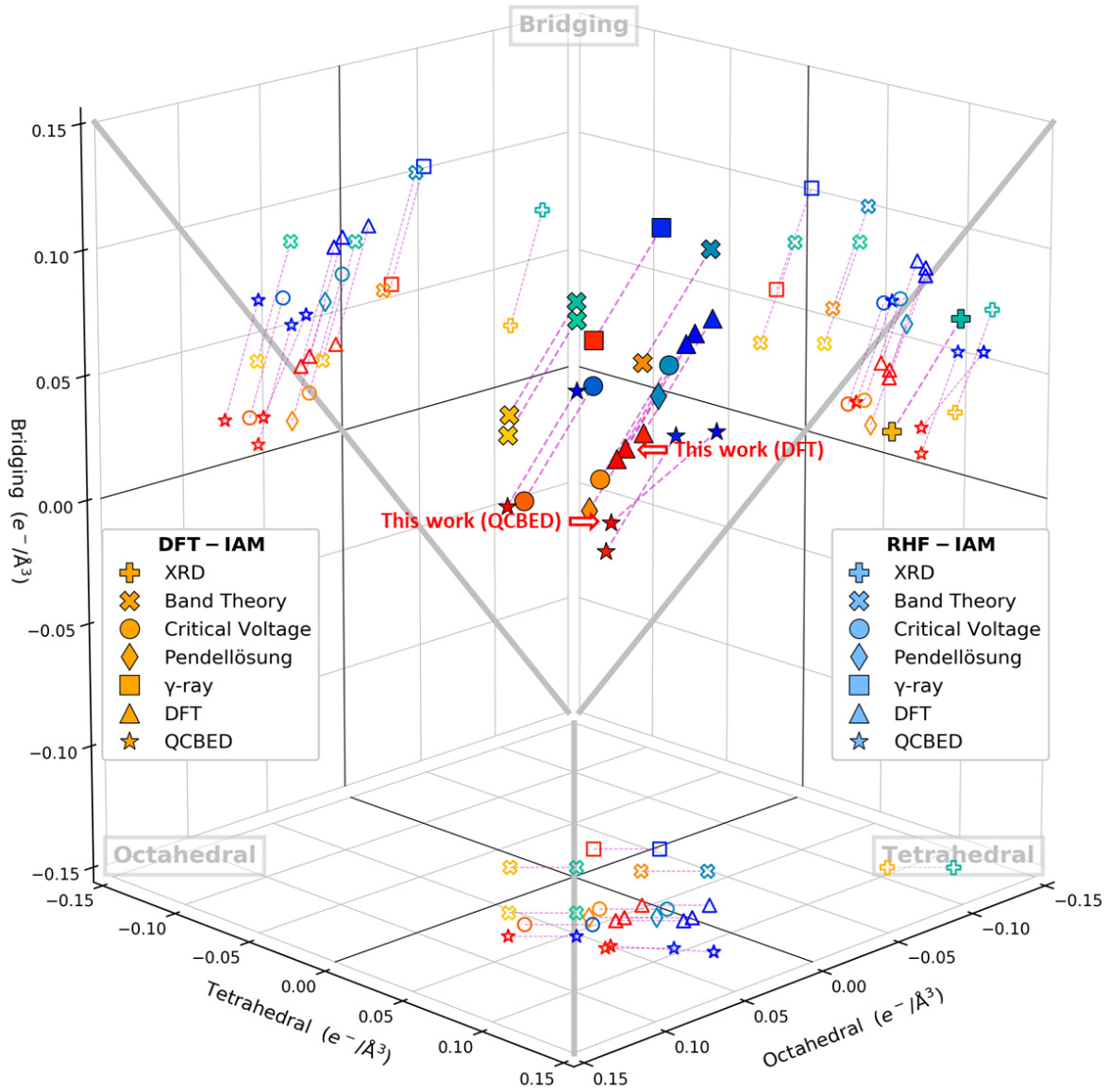


Figure 6.6. A zoom-in plot of Figure 6.5, which shows a small region clustered with the most recent studies (including the results from this work). Shades from orange to red are used to indicate the publication years from 1959 to 2005 using the DFT-IAM. The light to dark blue shades are used to show the corresponding studies from using the RHF-IAM. Experimental techniques include XRD (3 sets) [18] [42, 45], band theory (3 sets) [13, 16, 41, 79], critical voltages (2 sets) [4, 12], Pendellösung (2 sets) [14, 32], γ-ray (1 set) [11]and QCBED (3 sets) [4, 7] and DFT (3 sets, with results from this work highlighted in red box) [4, 7]. The shifting between the usages of the DFT-IAM and RHF IAM is constant (except for the QCBED results of this work, since the measured structure factors were also altered). The changes caused by a different IAM is very significant compared the differences between various studies.

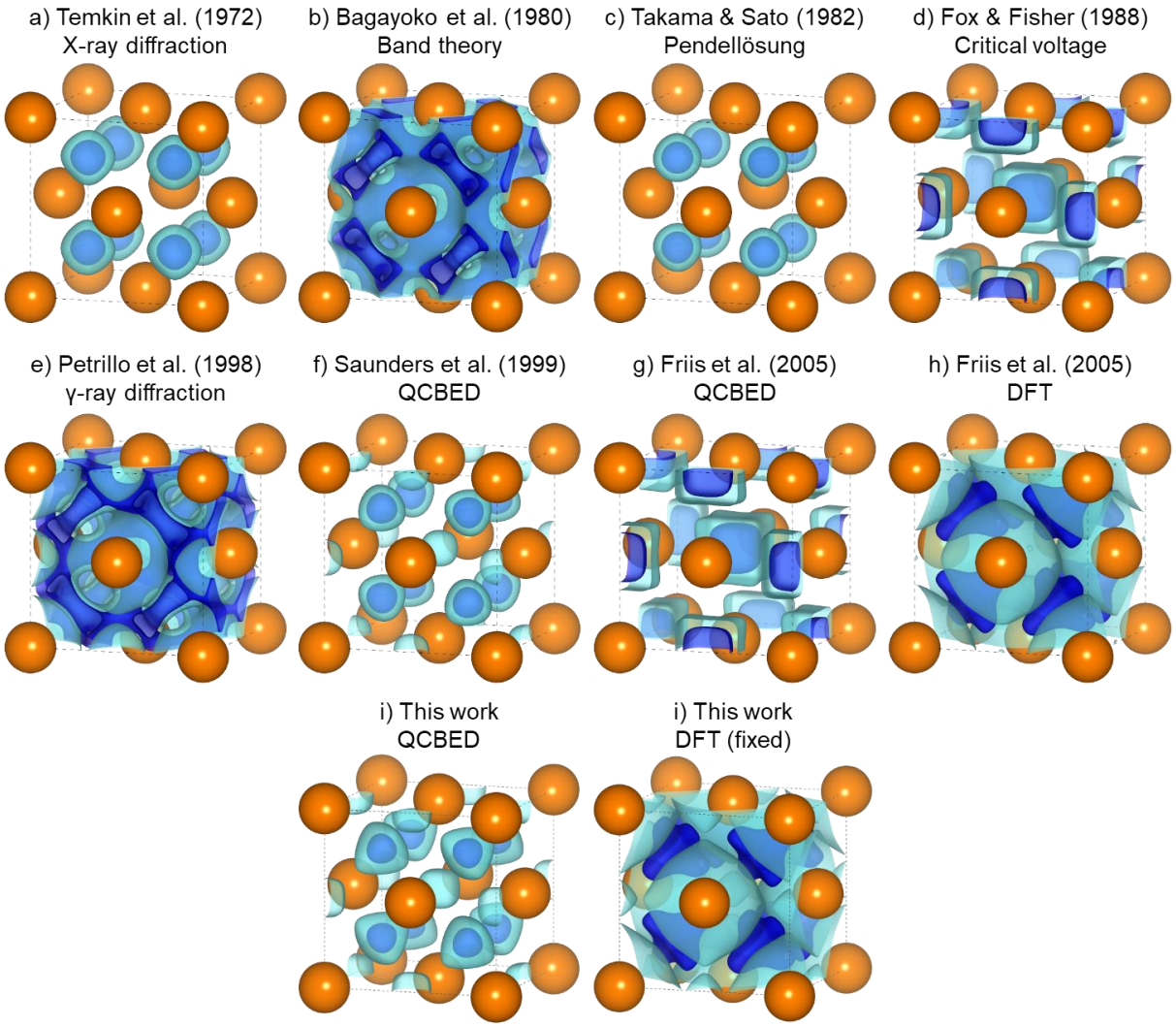


Figure 6.7. The bonding electron distributions for copper in the present work and previous studies (selected as they are either the most cited in bonding studies of copper or the most recent publications of the techniques [4, 7, 11, 12, 14, 16, 18]) using the DFT-IAM. The iso-surfaces are plotted at 50% (light blue) and 80% (dark blue) of the maximum bonding electron density of each result. Both a) & c) and h) and i) indicate tetrahedral localisation with different shapes respectively; b) & e) indicate a bridging-site bonding; both d) & g) agree with an octahedral type of bonding; while in f) & i), the bonding of copper is suggested to localise at the tetrahedral sites, as well as some bonding at the octahedral sites. In detail, the QCBED result from this suggests a pointier bonding distribution comparing to f).

Morphologies of the most recent previous studies and this work are plotted in Figure 6.7. Although the results from recent studies are relatively close compared to those of earlier studies as shown in Figure 6.5 and Figure 6.6, different shapes of bonding are found due to the different ratios of the octahedral, tetrahedral and bridging charge densities in these recent studies. Temkin *et al.*'s XRD (1972) [18], Takama & Sato's Pendellösung (1982) [14], Suanders *et al.*'s QCBED (1999) [4], Friis *et al.*'s DFT (2005) [7] and the current work's QCBED and DFT results have shown tetrahedral-like bonding, which are relatively similar to the aluminium bonding measured by Nakashima *et al.* (2011) [6]. The structure factors of these recent studies are also listed in Table 6.1. The low-order structure factors of copper measured by different QCBED experiments are comparable. However, due to the significance of F_{111} in an FCC structure, the bonding density maps of copper are different between the QCBED studies as shown in Figure 6.7: the QCBED results of this work and Saunders *et al.* [4] suggest the bonding is localised at the tetrahedral and octahedral sites, while the QCBED result of Friis *et al.* [7] suggest copper has an octahedral type bonding. While compared to Saunders *et al.*'s QCBED result, the QCBED result of the current work has shown a pointer bonding distribution.

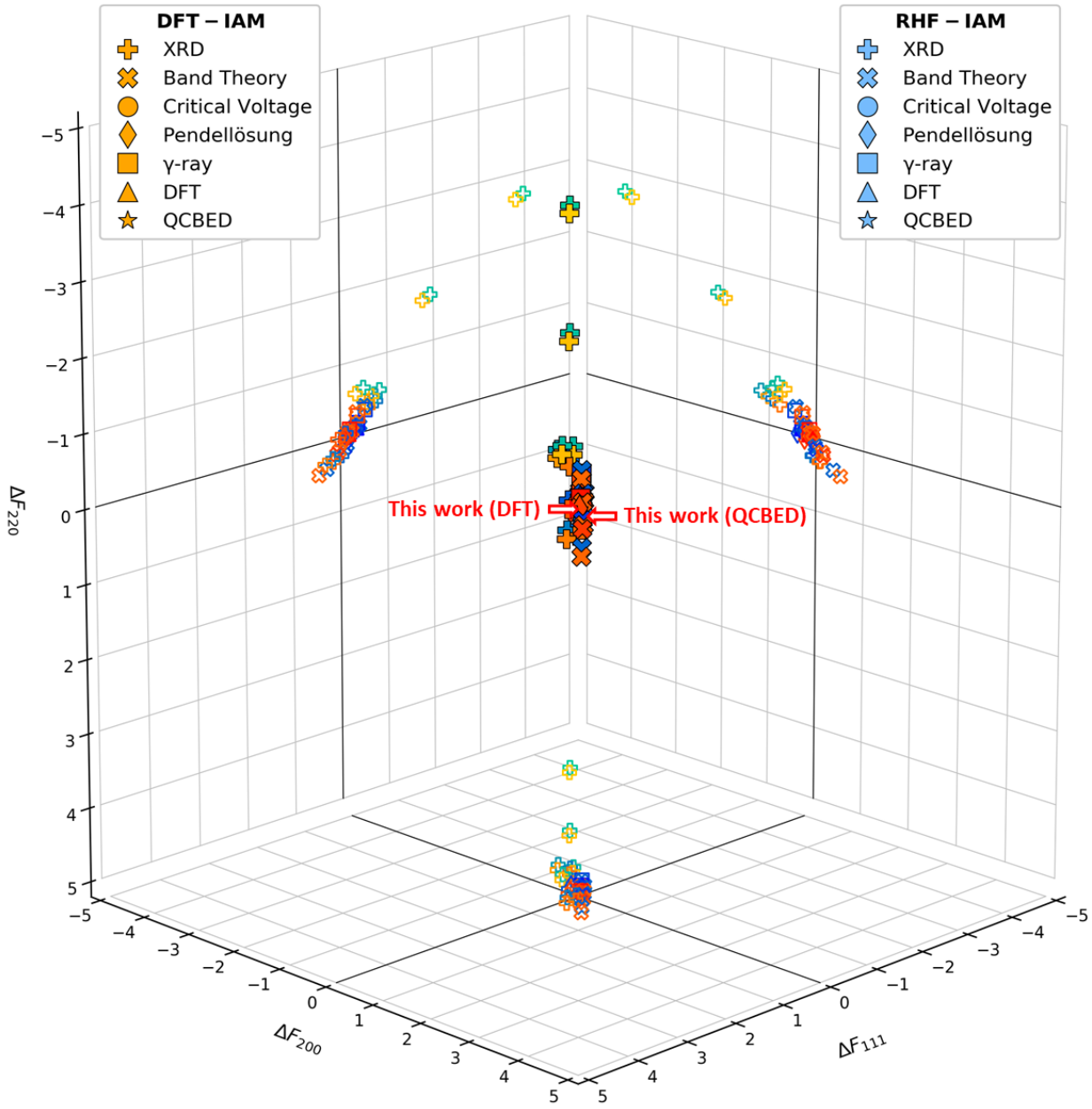


Figure 6.8. Summary of the lowest-orders of bonding structure factors $\Delta F_{\mathbf{g}}$ of bonding in copper for all studies (including this work). The ΔF_{220} axis is inversed for better overall distribution representation. Results of different studies are grouped by their techniques. Shades from orange to red are used to indicate the publication times (from 1930 to 2005) using the DFT-IAM. The shadows in grey are the corresponding studies using the RHF-IAM. Experimental techniques include XRD (9 sets) [18, 39, 43, 45, 47-49, 80], critical voltages (2 sets) [4, 12], Pendellösung (2 sets) [14, 32], γ-ray (2 sets) [11, 15] and QCBED (3 sets, including the current work) [4, 7]. Theoretical calculations include band theory (7 sets) [13, 16, 39, 41] and DFT (3 sets, including the current work) [4, 7]. Only the ΔF_{111}, ΔF_{200} and ΔF_{220}, were included. The $\Delta F_{\mathbf{g}}$ are changed constantly by: $\Delta F_{111}^{DFT} = \Delta F_{111}^{RHF} + 0.17$, $\Delta F_{200}^{DFT} = \Delta F_{200}^{RHF} + 0.15$, and $\Delta F_{220}^{DFT} = \Delta F_{220}^{RHF} + 0.06$.

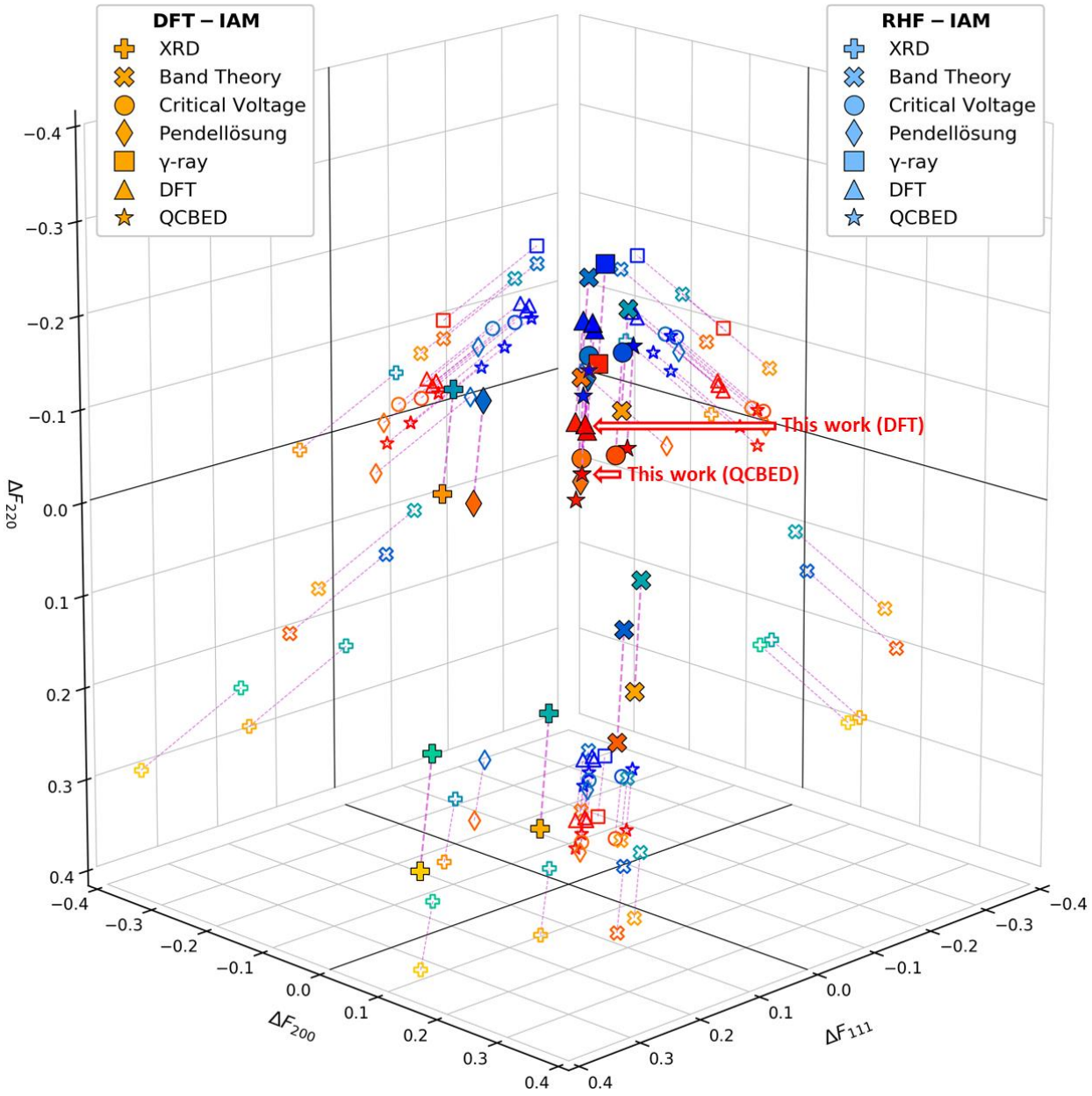


Figure 6.9. A zoom-in plot of Figure 6.8, showing a small region filled with results from most of the recent studies and the current work (highlighted in red boxes). Colourful markers are used to indicate the publications using the DFT-IAM, while the shadows in grey are the corresponding studies using the RHF-IAM. Experimental techniques include XRD (1 set) [18], band theory (3 sets) [16, 41, 79], critical voltages (2 sets) [4, 12], Pendellösung (1 set) [78], γ-ray (2 sets) [11, 15] and QCBED (3 sets) [4, 7] and DFT (3 sets, with results from this work highlighted in red box) [4, 7]. Compared to the $\Delta F_{\mathbf{g}}$ interpreted with the RHF-IAM, the shifting caused by the DFT-IAM are constant (except for results of this work, where the measured charge density is also altered).

The linear trend of weaker bridging bonding shown in Figure 6.5 may suggest systematic errors exist in the older studies, which result in smaller absolute values of structure factors. A summary graph of ΔF_{111}, ΔF_{200} and ΔF_{220} in copper for previous studies and this work and its zoom-in plot are shown as Figure 6.8 and Figure 6.9, respectively. The $\Delta\rho$ ($e^-/\text{Å}^3$) at the three bonding sites are related to the $\Delta F_{\mathbf{g}}$ by: 1) the octahedral $\Delta\rho^{Octa} = -8 \cdot \Delta F_{111} + 6 \cdot \Delta F_{200} + 12 \cdot \Delta F_{220}$, 2) the tetrahedral $\Delta\rho^{Tetra} = -6 \cdot \Delta F_{200} + 12 \cdot \Delta F_{220}$, and 3) the bridging $\Delta\rho^{Bri} = -2 \cdot \Delta F_{200} - 4 \cdot \Delta F_{220}$. These relations indicate that magnitudes of $\Delta\rho^{Bri}$ are directly proportional to the sums of ΔF_{200} and ΔF_{220}, while the magnitudes of $\Delta\rho^{Octa}$ and $\Delta\rho^{Tetra}$ consist of subtracting the magnitudes of a $\Delta F_{\mathbf{g}}$ from the others. Therefore the linear aggregation originating from the $\Delta F_{\mathbf{g}}$ distributions in Figure 6.8 leads to a similar trend in the $\Delta\rho$ summary of different studies in Figure 6.5. A good example of the impacts of systematic errors can be found in the magnified graph, Figure 6.9. The γ-ray measurements by Schneider in 1981, with $\Delta F_{111}^{DFT} = -1.65$, $\Delta F_{200}^{DFT} = 0.99$ and $\Delta F_{220}^{DFT} = -0.24$, have been considered as too low [3, 7, 11]. In contrast, for the result published by Petrillo *et al.* in 1998, which applied a scaling factor to correct the systematic errors (the bonding structure factors are increased to $\Delta F_{111}^{DFT} = -1.49$, $\Delta F_{200}^{DFT} = 1.15$ and $\Delta F_{220}^{DFT} = -0.09$), locates much closer to the results of other recent works. Compared to the bonding structure factors interpreted with the RHF-IAM, constant shifting caused by the usage of the DFT-IAM can also be found in Figure 6.8 and Figure 6.9 (except for the results of this work, where the measured charge density is also altered). The constant changes in the $\Delta F_{\mathbf{g}}$ are: $\Delta F_{111}^{DFT} = \Delta F_{111}^{RHF} + 0.17$, $\Delta F_{200}^{DFT} = \Delta F_{200}^{RHF} + 0.15$, and $\Delta F_{220}^{DFT} = \Delta F_{220}^{RHF} + 0.06$. Generally, $\Delta F_{\mathbf{g}}$ of the current work is surrounded by the most recent literature values.

The QCBED and DFT results of the current work are close to the most recent copper studies, while the values of earlier studies are significantly different. Both the QCBED and DFT results of the current work have roughly agreed that the bonding in copper is tetrahedral-like. However, the morphologies of bonding are still different, even for the most recent studies.

6.5. Mechanical properties of copper determined from the bonding density

As mentioned in Section 1.2, correlation may be drawn from the bonding electron density and deformation electrostatic potential of a solid material to its mechanical properties [6].

In Figure 6.10, plots of the $\Delta V = 0\ V$ iso-surface are shown for different copper studies and IAMs. The surfaces are coloured by the deformation electron density, Δρ(r), in each case. Beside the present QCBED results using the DFT-IAM and RHF-IAM respectively, the work of Saunders *et al.* (1999) [4] was chosen because it shows the closest agreement with the present study, while the results of Schneider *et al.* (1981) [15] are very different to the present results.

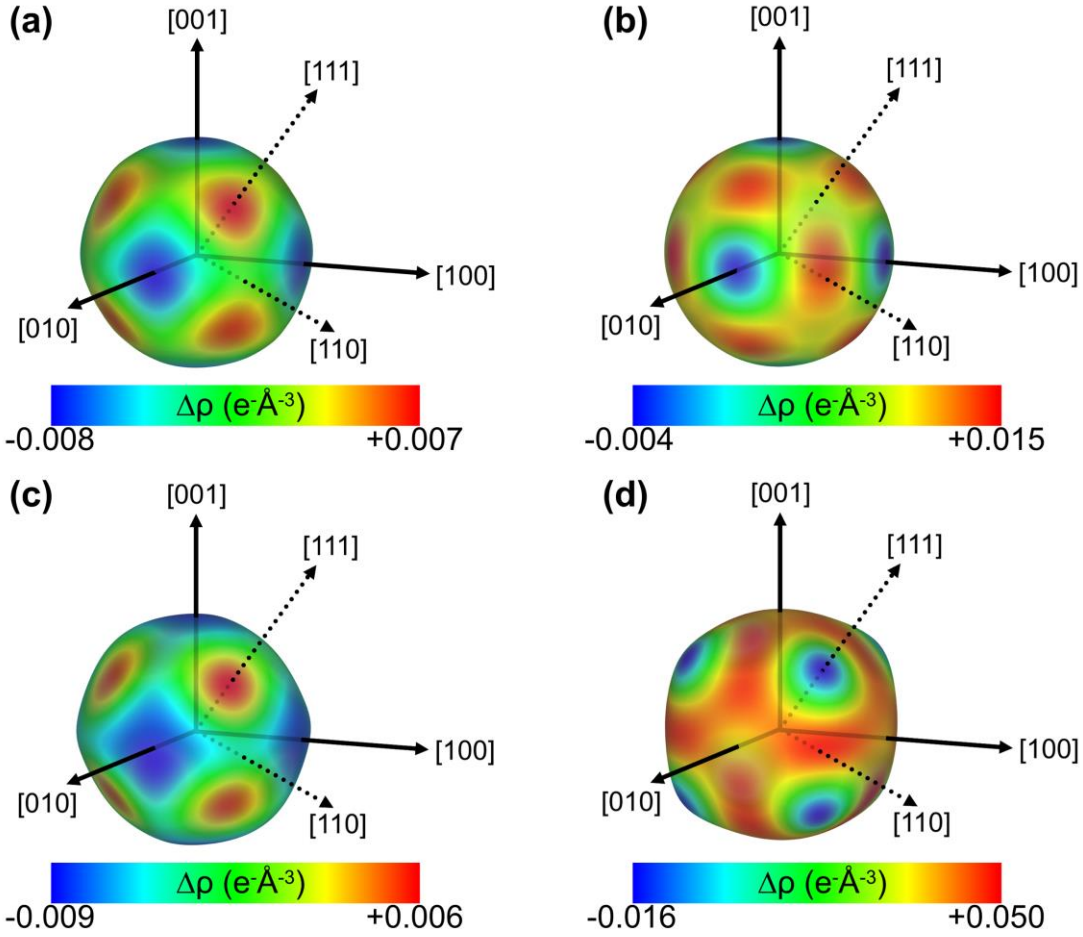


Figure 6.10. Deformation electrostatic potential iso-surface plots for $\Delta V = 0\ V$, coloured by the intersected bonding charge, $\Delta\rho(\mathbf{r})$), for: (a) the present QCBED results with the DFT-IAM, (b) the present QCBED results with the RHF-IAM, (c) the QCBED results of Saunders *et al.* (1999) [4] with the DFT-IAM, and (d) the γ-ray study of Schneider *et al.* (1981) [15]with the DFT-IAM. The principal axes are drawn from an origin located at the nucleus of a copper atom in all cases. This figure was prepared using VESTA [10].

Chapter 6. Final results of bonding in copper

In the study of bonding in aluminium by Nakashima *et al.* [6], the correlation between elastic moduli in different crystallographic directions, $E_{<uvw>}$, was explained as being proportional to the value of $\Delta\rho(\mathbf{r})$ intersected by the ΔV=0 iso-surface. Comparing (a) and (c) in Figure 6.10, it can be seen that both the QCBED results of the present work and Saunders *et al.* (1999) [4] have agreed that $E_{<001>} < E_{<011>} < E_{<111>}$ in copper, while the DFT-IAM is used. A similar trend is observed in aluminium [6] but the differences are smaller in the case of aluminium. The magnitudes of $E_{<uvw>}$ of Figure 6.10(a) and (c) are close to each other, as well. Figure 6.10(b) shows the QCBED results of this work while the RHF-IAM is used, which suggests that $E_{<001>} < E_{<111>} < E_{<011>}$ instead. This contradicts not only (a) and (c) in Figure 6.10, but also the elastic moduli measured mechanically from bulk single crystals of copper where $E_{<001>} = 67.1$ GPa, $E_{<011>} = 131.3$ GPa and $E_{<111>} = 192.8$ GPa [113]. Figure 6.10(d) of the γ-ray results of Schneider *et al.* (1981) [15], comparing to the other three plots, shows very different ratios, $E_{<111>} < E_{<001>} < E_{<011>}$, and magnitudes of $E_{<uvw>}$.

To compare the current work to the previous studies of copper in the context of mechanical properties, the ratios of $\Delta\rho_{<001>,\Delta V=0}$, $\Delta\rho_{<011>,\Delta V=0}$ and $\Delta\rho_{<111>,\Delta V=0}$ (where $\Delta\rho_{<uvw>,\Delta V=0}$ is the deformation electron density intersected by the $\Delta V = 0\ V$ iso-surface in the <*uvw*> direction) for all studies listed in Table 6.1 in the previous section are summarised in Figure 6.11. The maximum elastic modulus, E_{max}, in each set of the results, derived from the bonding, is listed in column on the right. The correlation between the mechanical properties and bonding can be expressed as [6]:

Eq. 6.1
$$\frac{U_{bonding}}{\Omega_{WS}} = \int \Delta\rho(\mathbf{r}) \cdot V_{bonded}(\mathbf{r})\, d\mathbf{r},$$

where $\Delta\rho(\mathbf{r})$ is the bonding charge density and $V_{bonded}(\mathbf{r})$ is the full crystal potential with bonding, integrating over every point within the Wigner-Seitz cell specified by $\mathbf{r}$. $U_{bonding}$ is the bonding potential energy (in Joules) and Ω_{WS} is the volume of the Wigner-Seitz cell surrounding each atom (in m^3). The potential energy density therefore has unit of Pascals, and $U_{bonding}$ per atom is equivalent to the maximum value attainable by the elastic modulus.

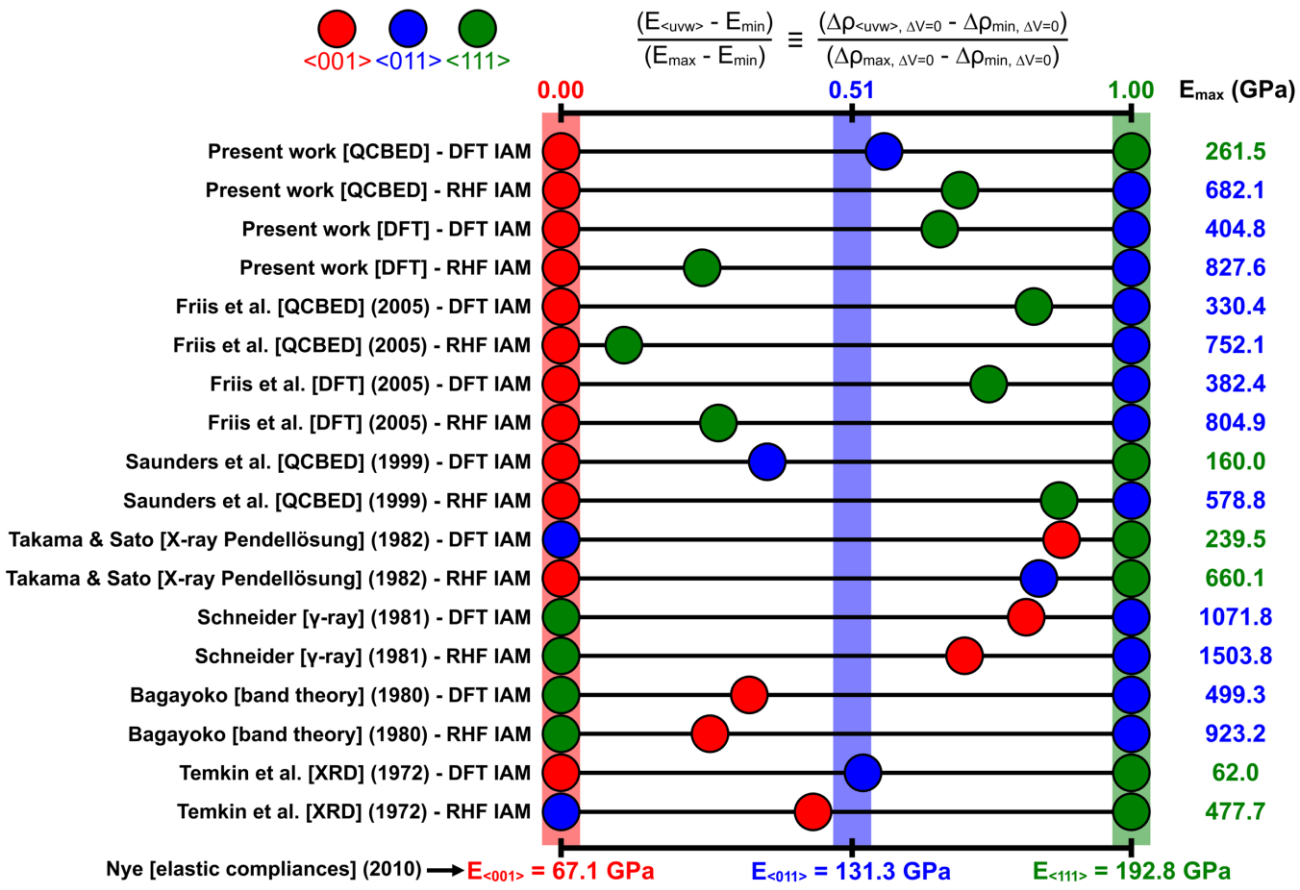


Figure 6.11. A comparison of the ratios of the deformation charge density intersected by the zero-deformation potential iso-surface, $\Delta\rho_{<uvw>,\Delta V=0}$, in each of the major crystallographic directions, <001>, <011> and <111> from the literature and the present work as listed in Table 6.1 in Section 0. Comparisons of using both the DFT-IAM and RHF-IAM are presented. All of these ratios are compared to the known elastic moduli for copper in the three major crystallographic directions, calculated from elastic compliances listed in [113]. Vertical colour coded bands are drawn in to facilitate this comparison to the reference values. The colour coding is red for <001>, blue for <011> and green for <111>. The bonding energy density within the Wigner-Seitz cell is listed for each interpretation at right in GPa, which corresponds to the maximum elastic modulus in any direction in copper.

Several points of discussion arise from Figure 6.11. The first is that, as indicated in Section 5.1, replacing the IAM leads to significantly different conclusions. The magnitudes of E_{max} are generally multiplied when the RHF-IAM is used instead of the DFT-IAM. In some cases, e.g. the QCBED results of the present work, the ratios of $E_{<uvw>}$ are also changed with a different IAM. Generally, the DFT-IAM helps to conclude $E_{<uvw>}$ trends closer to the mechanically measured values by [113], compared to the RHF-IAM.

The second point is that, while considering only the cases where the DFT-IAM is used, only three data sets satisfy the known relationship $E_{<001>} < E_{<011>} < E_{<111>}$ of Nye's [113]. These are the present QCBED results (DFT-IAM), Saunders *et al.*'s QCBED results [4] (DFT-IAM) and Temkin *et al.*'s

single crystal X-ray structure factors [18] (DFT-IAM). Temkin *et al.*'s XRD results (DFT-IAM) have the closest fit of the ratios of $E_{<uvw>}$, although the maximum elastic modulus is only a fracture of Nye's values [113]. The present QCBED results using the DFT-IAM come second closest in terms of order and ratios.

The third point is that the maximum elastic modulus, determined from the potential energy density in each scenario, is extremely sensitive to even the smallest variations in the structure factors as listed in Table 6.1. The present QCBED results with the DFT-IAM are in very good agreement with $E_{max} = 261.5$ GPa compare to the $E_{max} = 192.8$ GPa in [113] (still about 35% too large), with very good fitting of the ratios discussed previously. Whilst the QCBED results of Saunders *et al.* [4] with the DFT-IAM do not yield ratios of $\Delta\rho_{<uvw>,\Delta V=0}$ as accurate as the present QCBED work, the E_{max} of 160 GPa is only about 15% in error.

A fourth and final point for this section, although many other points could be made, is that the two most accurate results revealed from Figure 6.11, namely those of Saunders *et al.* and the present work (with the DFT-IAM), both involved measuring structure factors with 2D on- or near -axis QCBED.

This section has illustrated a method of verifying the accuracy of charge density measurements in a crystalline solid and a superposition of independent, unbonded atoms by deriving elastic moduli that can be compared to the known properties of bulk crystals obtained from mechanical tests. This naturally begs the question as to whether these methods can be developed to the point where one might obtain a bonding charge density in the context of an IAM directly from mechanical properties alone. In other words, can mechanical testing determine bonding morphologies with even greater accuracy than performing extremely accurate sets of diffraction experiments, which are ultimately much more complex? This section concludes on this open question.

Chapter 6. Final results of bonding in copper

Chapter 7. Conclusion

The bonding in copper was characterised using the angular differential QCBED (AGQCBED) technique in this work, with a new proposed IAM of copper and atomic scattering factor interpolation method. An accelerated multislice pattern-matching program and a batch-processing framework for bulk QCBED data were also developed, reducing the time-consumption required by QCBED analyses. In this chapter, the key results will be summarised, following the sequence of previous chapters.

7.1. The parallelised and accelerated program *QCBEDMS-PF*

QCBEDMS-PF enabled a vast improvement in speed performance compared to its original single threaded version *QCBEDMS*. With the use of the high performance FFT library FFTW and array operation optimisations, the single-threaded speed of *QCBEDMS-PF* is already improved by a factor of five, typically. While using multiple CPU cores and especially with multiple GPUs, the processing speed can be further multiplied, with nearly 100% of parallelisation efficiency. Generally, with 4 CPU cores and a powerful GPU on a modern desktop computer, *QCBEDMS-PF* can finish one calculation of a relatively thick pattern including thousands of reflection beams within seconds. This typically leads to about 50 times acceleration to comparing to the calculation speed before *QCBEDMS-PF* was implements. If enough computing resources are presented, for example, using multiple GPUs and many CPU cores, the same calculation can be finished within a second. The process time can be further reduced when lower precision and lower resolution are used, achieving hundreds of QCBED iterations within minutes. With such fast calculations, "live QCBED" – in-situ QCBED analyses of specimen thickness or even bonding information become possible.

7.2. The independent-atom model calculated by density functional theory

WIEN2k, a DFT program, was used to simulate an independent copper atom, by constructing a huge unit cell with only one copper atom. The charge density of the huge unit cell is then calculated, and regarded as the IAM charge density of copper. By including the effects of both electron exchanges and correlations, the DFT calculations have provided a more accurate description of the IAM for copper comparing to the RHF calculations by Doyle and Turner [9]. X-ray atomic scattering factors were found to be decreased in the new IAM at smaller scattering angles *s*, which is expected with the inclusion of electron correlation. Comparing to the conventional IAM derived from Doyle and Turner's RHF calculations, the largest differences are observed at $0.1 \leq s \leq 0.4$ Å^{-1}, where the DFT X-ray and electron atomic scattering factors can be about 1% and 5% offsetting from the RHF values respectively. Smaller differences are observed at higher *s* of atomic scattering factors. Although the peak difference of X-ray structure factors is smaller than 1% between the RHF and DFT copper IAMs, it can lead to a significant change for bonding determination in copper, since the difference between the IAM and absolution charge density is only about 1%. The more significant changes in electron atomic scattering factors, especially at higher scattering angles, can also impact on the measurements of the overall absolute charge density in copper, as QCBED bonding measurements only refine the

low-order structure factors, while the higher-order structure factors are assumed as identical to the IAM values.

7.3. The bonding in copper

From the analysis of a combination of energy filtered and unfiltered CBED data collected near the <100>, <110> and <112> axes, the results of this work have shown that the bonding in copper is mostly localised at its tetrahedral sites, while a smaller proportion is located at the octahedral site. This bonding distribution is expected, as the bonding in aluminium, another FCC metal, also exhibits a tetrahedral-like bonding type [6]. This conclusion is similar to the previous QCBED study by Saunders *et al.* [4] also using 2D near-zone data, while the results of another QCBED study by Friis *et al.* [7] suggested the localisation occurs near the octahedral centre. The measured low-order structure factors of copper in this work are comparable to the previous studies. However, due to the significance of F_{111} in an FCC structure, the absolute magnitudes of bonding in copper are significantly different. The use of a more accurate IAM in the present work has led to more accurate QCBED measurements and bonding determination compared to previous copper studies. The interpretations of the anisotropic Young's modules for different bonding studies of copper also suggest the results of the current work have better agreement to the mechanical properties of copper.

7.4. Future work

For bonding measurements, refinement cycles including IAM calculations by tuned DFT configurations may further improve the accuracy of experimental measurements. Since the current QCBED measurements used only a universal density functional (which appears less accurate for copper), an DFT-IAM using a functional tuned for copper will likely benefit the accuracy of QCBED refinements.

For QCBED program acceleration, machine-learning may be implemented to guide the refinement cycles. Since the current version of the QCBED pattern-matching program only utilise generalised parameter-searching algorithms, a more intelligent optimisation algorithm should significantly reduce the total amount of parameter-searching steps, leading to closer realisation of "live QCBED".

Chapter 7. Conclusion

References

1. Allen, L. C. & Burdett, J. K., *The Metallic Bond—Dead or Alive? A Comment and a Reply.* Angewandte Chemie International Edition in English, 1995. **34**(18): p. 2003-2003.
2. Drude, P., *Zur Elektronentheorie der Metalle.* Annalen der Physik, 1900. **306**(3): p. 566-613.
3. Tabbernor, M. A., Fox, A. G., & Fisher, R. M., *An Accurate Reappraisal of the Elemental Form-Factors and Charge-Density of Copper.* Acta Crystallographica Section A, 1990. **46**: p. 165-170.
4. Saunders, M., Fox, A. G., & Midgley, P. A., *Quantitative Zone-Axis Convergent-Beam Electron Diffraction (CBED) Studies of Metals. I. Structure-Factor Measurements.* Acta Crystallographica Section A, 1999. **55**(3): p. 471-479.
5. Sang, X., Kulovits, A., Wang, G., & Wiezorek, J., *Validation of Density Functionals for Transition Metals and Intermetallics Using Data from Quantitative Electron Diffraction.* The Journal of Chemical Physics, 2013. **138**(8): p. 084504.
6. Nakashima, P. N. H., Smith, A. E., Etheridge, J., & Muddle, B. C., *The Bonding Electron Density in Aluminum.* Science, 2011. **331**(6024): p. 1583-1586.
7. Friis, J., Jiang, B., Marthinsen, K., & Holmestad, R., *A Study of Charge Density in Copper.* Acta Crystallographica Section A, 2005. **61**: p. 223-230.
8. Friis, J., Madsen, G. K. H., Larsen, F. K., Jiang, B., Marthinsen, K., & Holmestad, R., *Magnesium: Comparison of Density Functional Theory Calculations with Electron and X-ray Diffraction Experiments.* The Journal of Chemical Physics, 2003. **119**(21): p. 11359-11366.
9. Doyle, P. A. & Turner, P. S., *Relativistic Hartree-Fock X-ray and Electron Scattering Factors.* Acta Crystallographica Section A, 1968. **24**(3): p. 390-397.
10. Momma, K. & Izumi, F., *VESTA 3 for Three-Dimensional Visualization of Crystal, Volumetric and Morphology Data.* Journal of Applied Crystallography, 2011. **44**(6): p. 1272-1276.
11. Petrillo, C., Sacchetti, F., & Mazzone, G., *Relevance of Charge-Density Measurements for High-Precision Calculations.* Acta Crystallographica Section A, 1998. **54**(4): p. 468-480.
12. Fox, A. & Fisher, R., *A Summary of Low-Angle X-ray Atomic Scattering Factors Measured by the Critical Voltage Effect in High Energy Electron Diffraction.* Australian Journal of Physics, 1988. **41**(3): p. 461-468.
13. Eckardt, H., Fritsche, L., & Noffke, J., *Self-Consistent Relativistic Band Structure of the Noble Metals.* Journal of Physics F: Metal Physics, 1984. **14**(1): p. 97.
14. Takama, T. & Sato, S., *Atomic Scattering Factors of Copper Determined by Pendellösung Intensity Beat Measurements Using White Radiation.* Philosophical Magazine Part B, 1982. **45**(6): p. 615-626.
15. Schneider, J. R., Hansen, N. K., & Kretschmer, H., *A Charge-Density Study of Copper by γ-ray Diffractometry on Imperfect Single-Crystals.* Acta Crystallographica Section A, 1981. **37**(SEP): p. 711-722.
16. Bagayoko, D., Laurent, D. G., Singhal, S. P., & Callaway, J., *Band Structure, Optical Properties, and Compton Profile of Copper.* Physics Letters A, 1980. **76**(2): p. 187-190.
17. Gray, A. M., *Metallic-Cu X-ray Form Factors.* Physical Review B, 1974. **9**(6): p. 2773-2774.
18. Temkin, R. J., Raccah, P. M., & Henrich, V. E., *Experimental Charge-Density of Copper.* Physical Review B, 1972. **6**(10): p. 3572-3581.
19. Jensen, W. B., *A Quantitative van Arkel Diagram.* Journal of Chemical Education, 1995. **72**(5): p. 395.
20. Ketelaar, J. a. A., *Chemical Constitution: An Introduction to the Theory of the Chemical Bond*. 2d rev. ed. 1958, Amsterdam; New York: Elsevier Publishing Company.
21. Van Arkel, A. E., *Molecules and Crystals*. 1949, London: Butterworths.
22. Meek, T. L. & Garner, L. D., *Electronegativity and the Bond Triangle.* Journal of Chemical Education, 2005. **82**(2): p. 325.

References

23. Allen, L. C. & Capitani, J. F., *What is the Metallic Bond?* Journal of the American Chemical Society, 1994. **116**(19): p. 8810-8810.

24. Jensen, W. B., *The Origin of the Metallic Bond.* Journal of Chemical Education, 2009. **86**(3): p. 278.

25. Schön, J. C., *Does the Death Knell Toll for the Metallic Bond?* Angewandte Chemie International Edition in English, 1995. **34**(10): p. 1081-1083.

26. Anderson, W. P., Burdett, J. K., & Czech, P. T., *What Is the Metallic Bond?* Journal of the American Chemical Society, 1994. **116**(19): p. 8808-8809.

27. Coll, R. K. & Treagust, D. F., *Learners' Mental Models of Metallic Bonding: A Cross-Age Study.* Science Education, 2003. **87**(5): p. 685-707.

28. Ashcroft, N. W. & Mermin, N. D., *Solid State Physics*. 1976, New York: Holt, Rinehart and Winston.

29. Levi, A. F. J., *Essential Classical Mechanics for Device Physics*. 2016, Morgan & Claypool Publishers.

30. Eberhart, M. E., *The Metallic Bond: Elastic Properties.* Acta Materialia, 1996. **44**(6): p. 2495-2504.

31. Jiang, B., Friis, J., Holmestad, R., Zuo, J. M., O'keeffe, M., & Spence, J. C. H., *Electron Density and Implication for Bonding in Cu.* Physical Review B, 2004. **69**(24).

32. Friis, J., Jiang, B., Spence, J. C. H., & Holmestad, R., *Quantitative Convergent Beam Electron Diffraction Measurements of Low-Order Structure Factors in Copper.* Microscopy and Microanalysis, 2003. **9**(5): p. 379-389.

33. Saunders, M., Bird, D. M., Zaluzec, N. J., Burgess, W. G., Preston, A. R., & Humphreys, C. J., *Measurement of Low-Order Structure Factors for Silicon from Zone-Axis CBED Patterns.* Ultramicroscopy, 1995. **60**(2): p. 311-323.

34. Saka, T. & Kato, N., *Accurate Measurement of the Si Structure Factor by the Pendellösung Method.* Acta Crystallographica Section A, 1986. **42**(6): p. 469-478.

35. Matsuhata, H., Tomokiyo, Y., Watanabe, H., & Eguchi, T., *Determination of the Structure Factors of Cu and Cu_3Au by the Intersecting Kikuchi-Line Method.* Acta Crystallographica Section B, 1984. **40**(6): p. 544-549.

36. Bauer, G. E. W. & Schneider, J. R., *Tight-Binding Model Wave-Functions for Compton Profiles and Structure Factors of Copper Metal.* Journal of Physics and Chemistry of Solids, 1984. **45**(6): p. 675-683.

37. Prakash, S., *Form Factors of Copper and Nickel.* Physics Letters A, 1971. **37**(3): p. 177-178.

38. Moriarty, J., *Pseudopotential Form Factors for Copper, Silver, and Gold.* Physical Review B, 1970. **1**(4): p. 1363-1370.

39. Sirota, N. N., *Survey of Results in the Determination of X-ray Intensities and Structure Factors for Metals, Alloys and Covalent Compounds.* Acta Crystallographica Section A, 1969. **25**(1): p. 223-243.

40. Saunders, M. & Fox, A. G., *Quantitative Convergent Beam Electron Diffraction (CBED) Measurements of Low-Order Structure Factors in Metals*, in *Electron Microscopy and Analysis 1997*, Rodenburg, J. M., Editor. 1997. p. 129-132.

41. Snow, E. C., *Self-Consistent Energy Bands of Metallic Copper by the Augmented-Plane-Wave Method. II.* Physical Review, 1968. **171**(3): p. 785-789.

42. Hosoya, S. & Yamagishi, T., *Absolute Measurement of X-ray Scattering Factors of Copper.* Journal of the Physical Society of Japan, 1966. **21**(12): p. 2638-2644.

43. Batterman, B., Chipman, D., & Demarco, J., *Absolute Measurement of the Atomic Scattering Factors of Iron, Copper, and Aluminum.* Physical Review, 1961. **122**(1): p. 68-74.

44. Chipman, D. R., Demarco, J. J., & Batterman, B. W., *Measurement of the X-ray Atomic Scattering Factors of Iron, Copper, and Aluminum.* Acta Crystallographica, 1960. **13**(12): p. 994-995.

45. Batterman, B. W., *X-ray Measurement of the Atomic Scattering Factor of Iron.* Physical Review, 1959. **115**(1): p. 81-86.

46. Brindley, G. W., *Atomic Scattering Factors of Aluminium, Potassium Chloride and Copper for X Rays.* Proceedings of the Physical Society, 1938. **50**(1): p. 96-107.

References

47. Brindley, G. W., *The Atomic Scattering Factors of Aluminium Nickel, and Copper for CuKα Radiation and Their Relation to the Theory of X-ray Dispersion.* Philosophical Magazine, 1936. **21**(142): p. 778-790.

48. Brindley, G. W. & Spiers, F. W., *Atomic Scattering Factors of Nickel, Copper and Zinc.* Philosophical Magazine, 1935. **20**(131-37): p. 865-881.

49. Rusterholz, A. A., *Über Die Streuung von Röntgenstrahlen an Kupfer und Silber*. Zeitschrift für Physik, 1930. **65**(3): p. 226-232.

50. Nakashima, P. N. H., *Quantitative Convergent-Beam Electron Diffraction and Quantum Crystallography—the Metallic Bond in Aluminium.* Structural Chemistry, 2017. **28**(5): p. 1319-1332.

51. Zuo, J. M. & Spence, J. C. H., *Advanced Transmission Electron Microscopy: Imaging and Diffraction in Nanoscience*. 2016: Springer New York.

52. Guerrero, A. H., Fasoli, H. J., & Costa, J. L., *Why Gold and Copper Are Colored but Silver Is Not.* Journal of Chemical Education, 1999. **76**(2): p. 200.

53. Eickerling, G., Mastalerz, R., Herz, V., Scherer, W., Himmel, H.-J., & Reiher, M., *Relativistic Effects on the Topology of the Electron Density*. Journal of Chemical Theory and Computation, 2007. **3**(6): p. 2182-2197.

54. Authier, A. & Zarembowitch, A., *Elastic Properties*, in *International Tables for Crystallography*. 2006, John Wiley & Sons, Ltd.

55. Humphreys, C. J., *The Scattering of Fast Electrons by Crystals.* Reports on Progress in Physics, 1979. **42**(11): p. 1825-1887.

56. Macdonald, A. H., Daams, J. M., Vosko, S. H., & Koelling, D. D., *Non-Muffin-Tin and Relativistic Interaction Effects on the Electronic Structure of Noble Metals.* Physical Review B, 1982. **25**(2): p. 713-725.

57. Zuo, J. M., *Quantitative Convergent Beam Electron Diffraction*, in *Electron Crystallography: Novel Approaches for Structure Determination of Nanosized Materials*, Weirich, T. E., Lábár, J. L., & Zou, X., Editors. 2006, Springer Netherlands: Dordrecht. p. 143-168.

58. Colliex, C., Cowley, J. M., Dudarev, S. L., Fink, M., Gjønnes, J., Hilderbrandt, R., Howie, A., Lynch, D. F., Peng, L. M., Ren, G., Ross, A. W., Smith, V. H., Spence, J. C. H., Steeds, J. W., Wang, J., Whelan, M. J., & Zvyagin, B. B., eds. *Electron Diffraction*. 1st online ed. International Tables for Crystallography, ed. Prince, E. Vol. C. 2006, International Union of Crystallography: Chester. 259-429.

59. Fox, A. G., O'keefe, M. A., & Tabbernor, M. A., *Relativistic Hartree-Fock X-ray and Electron Atomic Scattering Factors at High Angles.* Acta Crystallographica Section A, 1989. **45**(11): p. 786-793.

60. Lobato, I. & Van Dyck, D., *An Accurate Parameterization for Scattering Factors, Electron Densities and Electrostatic Potentials for Neutral Atoms That Obey All Physical Constraints.* Acta Crystallographica Section A, 2014. **70**(6): p. 636-649.

61. Kirkland, E. J., *Advanced Computing in Electron Microscopy: Second Edition*. 2010. 1-289.

62. Peng, L.-M., Ren, G., Dudarev, S. L., & Whelan, M. J., *Robust Parameterization of Elastic and Absorptive Electron Atomic Scattering Factors.* Acta Crystallographica Section A, 1996. **52**(2): p. 257-276.

63. Weickenmeier, A. & Kohl, H., *Computation of Absorptive Form Factors for High-Energy Electron Diffraction.* Acta Crystallographica Section A, 1991. **47**(5): p. 590-597.

64. Bird, D. M. & King, Q. A., *Absorptive form factors for high-energy electron diffraction.* Acta Crystallographica Section A, 1990. **46**(3): p. 202-208.

65. Lobato, I. & Van Dyck, D., *A Complete Comparison of Simulated Electron Diffraction Patterns Using Different Parameterizations of the Electron Scattering Factors.* Ultramicroscopy, 2015. **155**(0): p. 11-19.

66. Streltsov, V. A., Nakashima, P. N. H., & Johnson, A. W. S., *A Combination Method of Charge Density Measurement in Hard Materials Using Accurate, Quantitative Electron and X-ray Diffraction: The [alpha]-Al2O3 Case.* Microscopy and Microanalysis, 2003. **9**(05): p. 419-427.

67. Gertrud, Z., *The Utility of Band Theory in Strongly Correlated Electron Systems.* Reports on Progress in Physics, 2016. **79**(12).

References

68. Luo, S., Averkiev, B., Yang, K. R., Xu, X., & Truhlar, D. G., *Density Functional Theory of Open-Shell Systems. The 3d-Series Transition-Metal Atoms and Their Cations.* Journal of Chemical Theory and Computation, 2014. **10**(1): p. 102-121.

69. Cramer, C. J. & Truhlar, D. G., *Density Functional Theory for Transition Metals and Transition Metal Chemistry.* Physical Chemistry Chemical Physics, 2009. **11**(46): p. 10757-10816.

70. Lally, J. S., Humphreys, C. J., Metherell, A. J. F., & Fisher, R. M., *The Critical Voltage Effect in High Voltage Electron Microscopy.* Philosophical Magazine, 1972. **25**(2): p. 321-343.

71. Tomokiyo, Y., *Applications of Convergent Beam Electron Diffraction to Extract Quantitative Information in Materials Science.* Journal of Electron Microscopy, 1992. **41**(6): p. 403-413.

72. Kato, N., *Accurate Charge Density–Pendellösung Methods.* Australian Journal of Physics, 1988. **41**(3): p. 337-350.

73. Kato, N. & Lang, A. R., *A Study of Pendellösung Fringes in X-ray Diffraction.* Acta Crystallographica, 1959. **12**(10): p. 787-794.

74. Saunders, M., *Quantitative Zone-Axis Convergent Beam Electron Diffraction: Current Status and Future Prospects.* Microscopy and Microanalysis, 2003. **9**(05): p. 411-418.

75. Tanaka, M. & Tsuda, K., *Convergent-Beam Electron Diffraction.* Journal of Electron Microscopy, 2011. **60**(suppl 1): p. S245-S267.

76. Peng, L. M., *Electron Atomic Scattering Factors and Scattering Potentials of Crystals.* Micron, 1999. **30**(6): p. 625-648.

77. Berghuis, J., Haanappel, I. M., Potters, M., Loopstra, B. O., Macgillavry, C. H., & Veenendaal, A. L., *New Calculations of Atomic Scattering Factors.* Acta Crystallographica, 1955. **8**(8): p. 478-483.

78. Smart, D. J., And Humphreys, C. J. Unpublished work, 1980. Reported in *Electron microscopy and analysis, 1979: proceedings of the Institute of Physics Electron Microscopy and Analysis Group conference held at the University of Sussex, Brighton, 3-6 September 1979 (EMAG 79) (Ed. T. Mulvey), Institute of Physics, Conference series No. 52, p 211*.

79. Wakoh, S. & Yamashita, J., *THEORETICAL FORM FACTORS OF 3D TRANSITION METALS.* Journal of the Physical Society of Japan, 1971. **30**(2): p. 422-+.

80. Ladd, M. F. C., *Accurate Interpolation of Atomic Scattering Factors.* Zeitschrift Fur Kristallographie Kristallgeometrie Kristallphysik Kristallchemie, 1966. **123**(5): p. 373-&.

81. Giri, A. K. & Mitra, G. B., *Extrapolated Values of Lattice Constants of Some Cubic Metals at Absolute Zero.* Journal of Physics D: Applied Physics, 1985. **18**(7): p. L75.

82. Coulthard, M. A., *A Relativistic Hartree-Fock Atomic Field Calculation.* Proceedings of the Physical Society, 1967. **91**(1): p. 44.

83. Cromer, D. T. & Waber, J. T. Unpublished work, 1968. Reported in *Electron Diffraction. 1st online ed. International Tables for Crystallography. Vol. C. 2006, International Union of Crystallography: Chester. 259-429.*

84. Brown, P. J., Fox, A. G., Maslen, E. N., O'keefe, M. A., & Willis, B. T. M., *Intensity of Diffracted Intensities*, in *International Tables for Crystallography*, Prince, E., Editor. 2006, International Union of Crystallography: Chester. p. 554-595.

85. Kirkland, J., *Advanced Computing in Electron Microscopy*. Journal for the Study of the. 1998: Springer US.

86. Rez, D., Rez, P., & Grant, I., *Dirac–Fock Calculations of X-ray Scattering Factors and Contributions to the Mean Inner Potential for Electron Scattering.* Acta Crystallographica Section A, 1994. **50**(4): p. 481-497.

87. Rez, D., Rez, P., & Grant, I., *Dirac-Fock Calculations of X-ray Scattering Factors and Contributions to the Mean Inner Potential for Electron Scattering. Erratum.* Acta Crystallographica Section A, 1997. **53**: p. 522-522.

88. Davisson, C. J. & Germer, L. H., *Reflection of Electrons by a Crystal of Nickel.* Proceedings of the National Academy of Sciences of the United States of America, 1928. **14**(4): p. 317-322.

References

89. Thomson, G. P. & Reid, A., *Diffraction of Cathode Rays by a Thin Film.* Nature, 1927. **119**: p. 890.

90. Blackman, M., *On the Intensities of Electron Diffraction Rings.* Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences, 1939. **173**(952): p. 68.

91. Sturkey, L., *The Use of Electron-Diffraction Intensities in Structure Determination.* Acta Crystallogr, 1957. **10**.

92. Lorenzo, S., *The Calculation of Electron Diffraction Intensities.* Proceedings of the Physical Society, 1962. **80**(2): p. 321.

93. Ophus, C., *A Fast Image Simulation Algorithm for Scanning Transmission Electron Microscopy.* Advanced Structural and Chemical Imaging, 2017. **3**(1): p. 13.

94. Cowley, J. M. & Moodie, A. F., *The Scattering of Electrons by Atoms and Crystals. I. A New Theoretical Approach.* Acta Crystallographica, 1957. **10**(10): p. 609-619.

95. Nakashima, P. N. H. & Muddle, B. C., *Differential Quantitative Analysis of Background Structure in Energy-Filtered Convergent-Beam Electron Diffraction Patterns.* Journal of Applied Crystallography, 2010. **43**(2): p. 280-284.

96. Nakashima, P. N. H. & Muddle, B. C., *Differential Convergent Beam Electron Diffraction: Experiment and Theory.* Physical Review B, 2010. **81**(11): p. 115135.

97. Nakashima, P. N. H. & Johnson, A. W. S., *Measuring the PSF from Aperture Images of Arbitrary Shape—an Algorithm.* Ultramicroscopy, 2003. **94**(2): p. 135-148.

98. Zuo, J. M., *Electron Detection Characteristics of Slow-Scan CCD Camera.* Ultramicroscopy, 1996. **66**(1): p. 21-33.

99. Nakashima, P. N. H., *In Situ Quantification of Noise as a Function of Signal in Digital Images.* Optics Letters, 2012. **37**(6): p. 1023-1025.

100. Sang, X. H., Kulovits, A., & Wiezorek, J. M. K., *Determination of Debye–Waller Factor and Structure Factors for Si by Quantitative Convergent-Beam Electron Diffraction Using Off-Axis Multi-Beam Orientations.* Acta Crystallographica Section A, 2010. **66**(6): p. 685-693.

101. Nakashima, P. N. H., *Thickness Difference: A New Filtering Tool for Quantitative Electron Diffraction.* Physical Review Letters, 2007. **99**(12): p. 125506.

102. Nakashima, P. N. H., *Improved Quantitative CBED Structure-Factor Measurement by Refinement of Nonlinear Geometric Distortion Corrections.* Journal of Applied Crystallography, 2005. **38**(2): p. 374-376.

103. Holman, S. W., Lawrence, R. R., & Barr, L., *Melting Points of Aluminum, Silver, Gold, Copper, and Platinum.* Proceedings of the American Academy of Arts and Sciences, 1895. **31**: p. 218-233.

104. Shao, Y.-T., Kim, K.-H., & Zuo, J.-M., *Principles and Applications of Energy-Filtered Scanning CBED for Ferroelectric Domain Imaging and Symmetry Determination.* Microscopy and Microanalysis, 2015. **21**(S3): p. 1245-1246.

105. Egerton, R. F., Mcleod, R., Wang, F., & Malac, M., *Basic Questions Related to Electron-Induced Sputtering in the TEM.* Ultramicroscopy, 2010. **110**(8): p. 991-997.

106. *GNU sed*. Available from: https://www.gnu.org/software/sed/ [Accessed 2018-5-2].

107. *Gawk*. Available from: https://www.gnu.org/software/gawk/ [Accessed 2018-5-2].

108. *GNU Grep*. Available from: https://www.gnu.org/software/grep/ [Accessed 2018-5-2].

109. Dwyer, C., *Simulation of Scanning Transmission Electron Microscope Images on Desktop Computers.* Ultramicroscopy, 2010. **110**(3): p. 195-198.

110. Van Den Broek, W., Jiang, X., & Koch, C. T., *FDES, a GPU-Based Multislice Algorithm with Increased Efficiency of the Computation of the Projected Potential.* Ultramicroscopy, 2015. **158**: p. 89-97.

111. Hosokawa, F., Shinkawa, T., Arai, Y., & Sannomiya, T., *Benchmark Test of Accelerated Multi-Slice Simulation by GPGPU.* Ultramicroscopy, 2015. **158**: p. 56-64.

112. Eggeman, A. S., London, A., & Midgley, P. A., *Ultrafast Electron Diffraction Pattern Simulations Using GPU Technology. Applications to Lattice Vibrations.* Ultramicroscopy, 2013. **134**: p. 44-47.

References

113. Nye, J. F. & Nye, P. P. L. J. F., *Physical Properties of Crystals: Their Representation by Tensors and Matrices*. 1985: Clarendon Press.

References

Appendix A. Manuals for *QCBEDMS-PF*

A.1. Parallel processing options of *QCBEDMS-PF*

A.1.1. Parallel Multislice Propagation Methods

The Multislice Propagation (MSP) options expect three arguments: 1) Name of FFT Library, 2) Device Option and 3) Incident Tilts Splitting. Examples:

- To use FFTW with 8 threads:

```
fftw   8  0
```

- To use OpenCL (clFFT) with the second OpenCL device:

```
clfft  1  0
```

- To use CUDA (cuFFT) with the first CUDA device:

```
cufft  0  0
```

- To list available OpenCL GPU devices and their Device Number:

```
clfft -1  0
```

- To list available CUDA devices and they Device Number:

```
cufft -1  0
```

- To use OOURA FFT:

```
fftsg  0  0
```

a) The "Name of FFT Library" argument

This argument should be a string. Available options: FFTSG (single thread only), FFTW, clFFT & cuFFT. It controls which FFT library and the associated MSP framework to use.

b) The Device Option argument

This argument is an integer. It has no effect for FFTSG.

(a) For FFTW

It controls how many threads to use, which can exceed the number of available CPU cores on a machine.

(b) For OpenCL (clFFT) and CUDA (cuFFT)

It specifies which GPU device to use. Numbers of the devices on the system starts from "0". Set this argument to "-1" to query the available devices and their Device Number. Set this argument to "-2" to use all OpenCL GPU/CUDA devices. However, if the GPU core on CPU (supporting OpenCL) is also presented, such option may fail.

c) The "Incident Tilts Splitting" argument

This argument is an integer. It only affects the GPU frameworks, as CPU frameworks FFTW and FFTSG always process 1 tilt (per thread) at once.

By setting this argument to "0", the program will first try to process all tilts at once (on GPU), and try to split the number of tilts into 2 or more splits If the GPU device does not have enough memories to contain all tilts at once.

If the auto option "0" could not solve the memory usage, set this argument to integers larger than 0 to force the program to split the tilts.

A.1.2. Hybrid mode: CPU + GPU

To use hybrid MSP, start the MSP Methods with a "Hybrid Option" line with three arguments: 1) "Hybrid", 2) Number of MSP Methods and 3) Reassign-Tilts Option. Also insert a "Share of tilts" argument before each following MSP Method. Examples:

- Using clFFT and 8 threads of FFTW, and clFFT processes "5" parts, each FFTW thread processes "1" part of tils:

```
hybrid 2  0
5  clfft  0  0
1  fftw  8  0
```

- Using cuFFT and 8 threads of FFTW, and clFFT processes "5" parts, each FFTW thread processes "1" part of tils. And re-distribute the shares of tilts every "10" iterations according to the measured speeds of each MSP method:

```
hybrid 2  10
5  cufft 0  0
1  fftw  8  0
```

a) The "Hybrid Option" argument

Just enter "Hybrid" to enable the Hybrid Option.

b) The "Number of MSP Methods" argument

This argument expects an integer. It controls how many lines of MSP Methods to read in FOR007.DAT.

c) The "Reassign-Tilts" Option

This argument expects an integer. It controls the number of iterations to perform a "share-of-tilts re-distribution".

For example, "1" will let the program re-distribute the number of tilts of each MSP method before every iteration (according to the processing speed of the previous iteration); "10" will let the program average the speed of 10 iterations and re-distribute the tilts every 10 iteration. "0" will disable the re-distributing.

Please note that, because there are small (8 bytes) memory leaks in OpenCL and CUDA drivers, the Reassign-Tilts Option may cause errors if the re-distribution happened too many times in a refinement.

d) Ratio of Share-of-Tilts for each MSP method

Insert a "Ratio of Share-of-Tilts" argument before every MSP method when "Hybrid" method is used. This argument can be a decimal or an integer. The larger the more shares.

For FFTW with multi-threads, the ratio will be copied to each thread:

```
hybrid 2 0
5 cufft 0 0
1 fftw 4 0
```

will have the tilts distributed as 5:1:1:1:1 for cuFFT:FFTW-thread1:FFTW-thread2:FFTW-thread3:FFTW-thread4.

A.1.3. Hybrid mode: multiple GPUs

To use multiple GPUs, just specify separate GPU MSP methods in hybrid mode. For example, to use the #0, #4 and #5 OpenCL GPU device:

```
hybrid 3 0
1 clfft 0 0
1 clfft 4 0
1 clfft 5 0
```

A.1.4. MPI mode

The MPI mode is similar to the Hybrid mode: to use MPI mode, enter "mpi" instead of "hybrid" for the Hybrid Option argument.

a) Default MPI mode - Number of MSP Methods argument > 0

This mode is designed for scenarios when, many CPU cores spreading on multiple cluster nodes, without GPU MSP methods. For example, if a cluster job has 5 CPU cores on Node A and 3 cores on Node B, using

```
mpi 1 0
1 fftw 1 0
```

and execute *QCBEDMS-PF* with mpirun -np 8 (before ./*QCBEDMS-PF*) will have the program to use the 5 and 3 CPU cores.

b) Per-node MPI mode - Number of MSP Methods argument < 0

This mode is designed for scenarios when, equal numbers of CPU + GPU resources are distributed on multiple cluster nodes. For example, if there are 2 GPUs and 4 CPU cores on each of Node A, Node B and Node C, using

```
mpi-30
10 cufft 0 0
10 cufft 1 0
1 fftw 4 0
```

or

```
mpi-20
10 cufft -20
1 fftw 4 0
```

and execute *QCBEDMS-PF* with mpirun -pernode will have the program to use 2 GPUs and 4 CPU cores or all GPUs and 4 CPU cores on every node.

A.1.5. Multislice Propagation Testing mode

There is a testing mode for timing MSP. To enable this mode, start the MSP Methods with a "Testing Option" line with three arguments: 1) "test", 2) Number of Tests and 3) Repeat Option. The results of processing times will be output (append) to File "test_info.txt". Examples:

- To time an 8-threads FFTW:

```
test 1 1
fftw 8 0
```

- To time an 8-threads FFTW 5 times:

```
test 1 5
fftw 8 0
```

- To time a hybrid method 5 times, while the hybrid mode keep re-distributing tilts:

```
test 1 5
hybrid 2 1
5 cufft 0 0
1 fftw 8 0
```

- To time 2 tests, 5 times each:

```
 test 2  5
cufft   0  0
hybrid 2  0
5  cufft 0  0
1  fftw  8  0
```

- To time 2 tests, 5 times each and collect detailed processing times of key components as well ("-5"):

```
 test 4  -5
clfft  0  0
cufft   0  0
fftw    8  0
fftsg   0  0
```

a) The "Test Option" argument

Enter "test" for the first argument.

b) The "Number of Tests" argument

This argument expects an integer. It controls how many lines of (hybrid) MSP Methods to read in FOR007.DAT.

(a) The "Repeat Option" argument

This argument expects an integer. In controls how many times each test will be repeated. If the argument is > 0, then only the overall time of every iteration is collected. If the argument is < 0, then the processing time of "Fresnel propagator calculations", and "FFT operation", "IFFT operation", "Wave-Phase-grating Multiplication" and "Wave-Propagator Multiplication" in Multislice Propagation will be measured as well.

Please note that, because detailed time-measurement will trigger many more time-collect operations, which also include syncing of GPU task-status to CPU, it should not be used when Hybrid Mode is enable. Otherwise the GPU will be blocked for every time-measurement and require much more time to finish.

A.2. Input Pattern and Simulation Options of *QCBEDMS-PF*

A.2.1. Input Pattern Options

The first line of the input pattern file (e.g. "in.dat") controls the width, height and differentiation of the pattern. Number of reflection beams is determined in File "FOR007.DAT", outside this input pattern file.

The arguments are: 1) Width (number of pixels) and 2) Height (number of pixels) of reflection beams, and 3) Differentiation Options. Integers are expected for these three arguments.

The Width and Height could be different.

The Differentiation Option could be any integer larger than or equals to "0". If it is "0", then there is no differentiation. If it is > 0, then it specifies the differentiation length – the number of pixels of the differentiation shifting.

A.2.2. Simulation Options

QCBEDMS-PF can run without input pattern intensities. *QCBEDMS-PF* will enter the simulation mode if the Input Pattern Filename specified in File "FOR007.DAT" starts with "out" and "sim", while a filename never contains whitespace. The simulated pattern will be output to the Output Pattern Filename specified in File "FOR007.DAT". Examples:

- To calculate a pattern with 50×50 reflection-beam resolution and no differentiation:

```
out  50  50  0
```

- To calculate a pattern with 50×50 reflection-beam resolution and differentiation of 1 pixel:

```
out  50  50  1
```

- To generate a synthetic Input Pattern File, which consist of the Input Pattern Options header, pattern intensities, uncertainties and weightings:

```
sim  50  50  1
```

A.3. Compilation of *QCBEDMS-PF*

A.3.1. Possible combinations of systems, compilers and frameworks

OOURA FFT (FFTSG) is always supported, as it comes in Fortran source files.

FFTW frameworks in *QCBEDMS-PF* should be compiled without difficulty as well. To achieve best performance, one may wish to compile FFTW libraries on the local machine.

Supports of OpenCL and CUDA will require correct installation of the OpenCL/CUDA toolkits and GPU drivers. Please check that whether the target system and hardware have supported OpenCL and CUDA drivers.

a) Linux

- Intel compilers + FFTW + OpenCL + CUDA (tested)
- GNU compilers + FFTW + OpenCL + CUDA (tested)
- PGI compilers + FFTW + OpenCL + CUDA (tested)
- LLVM compilers + FFTW + OpenCL + CUDA (tested)

Combination using Fortran and C++ compilers from different vendors may also succeed. Although not tested, other compiler suites may succeed easily as well.

b) MacOS

- Intel compilers + FFTW + OpenCL + CUDA (tested)
- GNU compilers + FFTW + OpenCL + CUDA (tested; if `-march=native` causes errors, use `-march=avx2`)
- PGI compilers + FFTW + OpenCL + CUDA (tested)
- LLVM compilers + FFTW + OpenCL + CUDA (not tested; installation of Flang could be complex)
- Clang++ and Intel, GNU or PGI Fortran compiler + FFTW + OpenCL + CUDA (tested)

Support of the OpenCL framework on MacOS is out-of-box, requiring no extra configuration. CUDA on MacOS may require an older version of Xcode, because the Clang compilers supplied with the latest version of MacOS usually are not yet supported by the CUDA toolkit. Although not tested, other compiler suites may succeed easily as well.

c) Cygwin/MSYS on Windows

- GNU compilers + FFTW + OpenCL (tested)
- LLVM compilers + FFTW + OpenCL (not tested; installation of LLVM compilers could be complex)

Other compilers are not likely to work on Cygwin/MSYS, because their command line interface on Windows may differ from the syntax in "Makefile".

d) Native Windows

Visual Studio is required. Check the supported versions of Visual Studios by (Intel and CUDA) compilers

- Intel compilers + FFTW + OpenCL + CUDA (tested)
- Intel Fortran and Visual Studio C++ compiler + FFTW + OpenCL + CUDA (not tested; require modifications to the Makefile "*QCBEDMS*.mak")
- PGI Fortran and Visual Studio C++ compiler + FFTW + OpenCL + CUDA (not tested; require modifications to the Makefile "*QCBEDMS*.mak")

Other compiler suites (Fortran and C++) from the same vendor may also work with some effort. Other Fortran compilers may also work with the Visual Studio C++ compiler with more effort.

A.3.2. Compilation on Linux and MacOS

Such environments will rely on the (GNU) make command and the File "Makefile". After correctly modifying the Makefile, just call make or make all to compile. Call make clean to clean.

a) Settings that must be checked in Makefile

(a) Fortran and C++ Compilers

FC, CXX and OPT. These variables specify the Fortran, C++ compilers and optimisation flags.

Fortran source files of *QCBEDMS-PF* are in fix-form requiring extended-length of 132 characters. The extended-length flags of the following Fortran compilers will be added automatically: ifort, gfortran, pgfortran and flang. For other compilers, the extended-length flag should be added to the FGEN variable manually.

By default, the Fortran compiler will be used to link Fortran and C++ object files. The linking operation usually requires specifying the basic C++ library explicitly. The C++ library will be added to the linking option automatically for the following C++ compilers: icpc, g++, pgc++ and clang++. For other compilers, the C++ library should be added to the LDFLAGS variable manually.

(b) OOURA FFT, FFTW, OpenCL and CUDA

Variables FFTSG, FFTW, OPENCL and CUDA will control whether to compile the OOURA FFT, FFTW, OpenCL and CUDA frameworks respectively ("true" to compile, any other string to disable). At least one of

FFTSG and FFTW must be "true". OpenCL and CUDA (clFFT) must be installed to compile/execute the frameworks respectively.

NVCC_FLAG sets the (path to) nvcc compiler and flags of CUDA.

Portable libraries of FFTW and clFFT for different platforms are supplied in Folder "PORT_LIBS".

FFTW_DIR sets the directory of the FFTW library.

CLFFT_DIR sets the directory of the clFFT library.

OPENCL_LIB sets the directory of the OpenCL library on the system. The default values are meant for 64-bit systems. MacOS ignores this argument.

CUDA_DIR sets the directory of CUDA on the system. The default values are meant for 64-bit systems.

b) Link with dynamic FFTW and clFFT libraries if the statics failed

FFT_STATIC sets whether to link with the static libraries of FFTW and clFFT. To get an independent executable without the needs of dynamic FFTW & clFFT libraries, set it to "true". To get better compatibility, set it to anything other than "true".

c) sincos() from C++ library

SINCOS sets whether to explicitly call the sincos() function provided in C++ library for Fortran. This option should normally NOT be "true", because Fortran compilers will enable this feature automatically during optimisation.

d) MPI

To use MPI, the program should be compiled with MPI set to "true". Also set the MPI Fortran and C++ compilers in MPIFC and MPICXX. Correctly setting of the FC and CXX variables is also required, because the Fortran generate flag FGEN and Linking libraries flag LDFLAGS are determined by them.

A.3.3. Compilation on Cygwin/MSYS and Native Windows

a) Cygwin/MSYS

Follow the Linux and MacOS guidelines and use GNU compilers. Copy the dynamic-link libraries (*.DLL) of FFTW and clFFT and place the copies next to the executable, as it is where they will be searched during runtime.

To obtain an executable even portable outside Cygwin/MSYS environments (e.g. in Windows CMD or PowerShell), link with GNU compiler static libraries.

b) Native Windows

For compilation in native Windows environment, installation of Visual Studio is required for C++. Call nmake -f *QCBEDMS*.mak to compile, and nmake clean -f *QCBEDMS*.mak to clean.

(a) Fortran and C++ Compilers

Because only Intel Fortran compiler plus Intel C++ compilers have been tested to work in this condition, the Makefile "*QCBEDMS*.mak" is set for Intel compilers. Use Intel compiler's command line environment to compile.

(b) OOURA FFT, FFTW, OpenCL and CUDA

Similar to the "Makefile" for Linu and MacOS, modify the FFTSG, FFTW, OPENCL, CUDA options and library paths accordingly.

A.3.4. Auto-Vectorisation Optimisation Options

Because currently no SIMD instructions are implemented in the source codes of *QCBEDMS-PF*, vectorisations are relied on compilers' optimisation options.

For CPUs released after 2013, AVX2 is recommended (or just AVX if AVX2 is not available). For CPUs not supporting AVX/AVX2, SSE2, SSE3, SSSE3 or SSE4 should be enabled (higher the version, faster the speed). For CPUs that support AVX-512, related auto-vectorisation options may be enabled as well, although no significant performance improvement was observed compared to AVX2-only compilations in current version of *QCBEDMS-PF*

Because the supplied portable FFTW libraries are compiled with only AVX2 (and only SSE/SSE2 for Windows), users may wish to compile the FFTW library themselves as well with extra vectorisation supports, like AVX-512. Please refer to FFTW's website for compilation options. However, for the implement of the FFTW library in *QCBEDMS-PF*, no obvious improvement was observed by enabling AVX-512 (comparing to AVX2-only).

a) Intel compilers

For Intel compilers on Intel CPUs, the -xHost option (vectorisation according to the host machine) only enables SSE2 by default. To enable AVX2 or AVX-512, use -march=core-avx2 or -march=common-avx512. Please refer to the online document Intel® Compiler Options for Intel® SSE and Intel® AVX generation processor-specific optimizations and Intel compilers' manual for further details.

b) GNU compilers

GNU compilers can have -march=native to enable auto-vectorisation according to the host machine. On MacOS, -march=native has been reported to fail to detect the supported instruction sets; use the name of the CPU architecture, like -march=haswell, instead.

c) PGI compilers

With the optimisation flag -fast, PGI compilers will choose the best vectorisation (AVX-512, AVX/AVX2 or SSE2) automatically.

d) LLVM compilers

Both Clang++ and Flang also support -march=native.

Appendix A. Manuals for QCBEDMS-PF

Appendix A. Manuals for QCBEDMS-PF

Appendix B. Program scripts

B.1. *QCBEDMS-PF* – a paralleled multi-slice pattern-matching program

B.1.1. Multi-slice propagation scripts

a) Fortran subroutines and functions

```
! Multi-slice Propagation (MSP) related Fortran scripts

      module msp_c_wrapper
       implicit none
       interface
! Share Global share variables to C C wrappers
       subroutine Fortran_global_vars_to_C(pg,
     *   deltaz,xshift,yshift,applieddilation,cell2rpr,tiltarray,beam,
     *   res2adr,ib,
     *   ay,bee,cee,dee,meshx,meshy,mt,nslice,mbout,nump,numl,res2,tottilts,ncalcs)
     *   bind(c,name="Fortran_global_vars_to_C")
         use iso_c_binding, only: c_double,c_double_complex,c_int
         complex(c_double_complex)::pg(*)
         real(c_double)::deltaz(*),xshift(*),yshift(*),applieddilation(*),
     *     cell2rpr(*),tiltarray(*),beam(*)
         integer(c_int)::res2adr(*),ib(*)
         integer(c_int),value::ay,bee,cee,dee,meshx,meshy,mt,nslice,mbout,
     *     nump,numl,res2,tottilts,ncalcs
       end subroutine Fortran_global_vars_to_C

       subroutine Slice_beam_global_vars_to_C(sl_beam,mseq,sl_hi,sl_lo,sl_opt)
     *   bind(c,name="Slice_beam_global_vars_to_C")
         use iso_c_binding, only: c_double,c_int,c_bool
         real(c_double)::sl_beam(*)
         integer(c_int)::mseq(*)
         integer(c_int),value::sl_hi,sl_lo
         logical(c_bool),value::sl_opt
       end subroutine Slice_beam_global_vars_to_C

       subroutine MSP_global_vars_to_C(ratio,meth,
     *   dev,splt,chid,as_nt,as_toff,ntilt,
     *   MPI_rank,nhyb,hyb_redi,fftw_bat,nfftsg,nfftw,nclfft,ncufft,
     *   sgl) bind(c,name="MSP_global_vars_to_C")
         use iso_c_binding, only: c_double,c_char,c_int,c_bool
         real(c_double)::ratio(*)
         character(c_char)::meth(*)
         integer(c_int)::dev(*),splt(*),chid(*),as_nt(*),as_toff(*),ntilt(*)
         integer(c_int),value::MPI_rank,nhyb,hyb_redi,fftw_bat,nfftsg,nfftw,nclfft,ncufft
         logical(c_bool),value::sgl
       end subroutine MSP_global_vars_to_C

       subroutine Timer_global_vars_to_C(ch,pr,fft,ifft,mpg,mpr,msp)
     *   bind(c,name="Timer_global_vars_to_C")
         use iso_c_binding, only: c_double,c_bool
         real(c_double)::ch(*),pr(*),fft(*),ifft(*),mpg(*),mpr(*)
         logical(c_bool),value::msp
       end subroutine Timer_global_vars_to_C

! OpenCL & CUDA get total device counts OR print all devices C wrappers
       subroutine OpenCLGetGPU(in,n) bind(c,name="OpenCLGetGPU")
         use iso_c_binding, only: c_int
         integer(c_int),value::in
         integer(c_int)::n
       end subroutine OpenCLGetGPU

       subroutine CUDAGetDevices(in,n) bind(c,name="CUDAGetDevices")
         use iso_c_binding, only: c_int
         integer(c_int),value::in
         integer(c_int)::n
       end subroutine CUDAGetDevices

! MSP libraries global initiation/termination C wrappers
       subroutine fftw_init(pmap) bind(c, name="fftw_init")
         use iso_c_binding, only: c_double_complex
         complex(c_double_complex)::pmap(*)
       end subroutine fftw_init

       subroutine fftw_term() bind(c, name="fftw_term")
       end subroutine fftw_term

       subroutine clfft_init() bind(c, name="clfft_init")
       end subroutine clfft_init

       subroutine clfft_term() bind(c, name="clfft_term")
       end subroutine clfft_term

       subroutine cufft_init() bind(c, name="cufft_init")
       end subroutine cufft_init

       subroutine cufft_term() bind(c, name="cufft_term")
       end subroutine cufft_term

! MSP libraries per child initiation/termination C wrappers
       subroutine fftw_child_init(f) bind(c, name="fftw_child_init")
         use iso_c_binding, only: c_int
         integer(c_int),value::f
       end subroutine fftw_child_init

       subroutine fftw_child_term(f) bind(c, name="fftw_child_term")
         use iso_c_binding, only: c_int
         integer(c_int),value::f
       end subroutine fftw_child_term

       subroutine clfft_child_init(f) bind(c, name="clfft_child_init")
         use iso_c_binding, only: c_int
         integer(c_int),value::f
       end subroutine clfft_child_init

       subroutine clfft_child_term(f) bind(c, name="clfft_child_term")
         use iso_c_binding, only: c_int
         integer(c_int),value::f
       end subroutine clfft_child_term

       subroutine cufft_child_init(f) bind(c, name="cufft_child_init")
         use iso_c_binding, only: c_int
         integer(c_int),value::f
       end subroutine cufft_child_init

       subroutine cufft_child_term(f) bind(c, name="cufft_child_term")
         use iso_c_binding, only: c_int
         integer(c_int),value::f
       end subroutine cufft_child_term

! Hybrid MSP C++11 <thread> wrapper
       subroutine msp_cppthread_exec(nf,offset) bind(c, name='msp_cppthread_exec')
         use iso_c_binding, only: c_int
         integer(c_int),value::nf,offset
       end subroutine msp_cppthread_exec

! One-time FFT & per child MSP C wrappers
       subroutine fftw_one(dir) bind(c, name="fftw_one")
       use iso_c_binding, only: c_int
       integer(c_int),value::dir
       end subroutine fftw_one

       subroutine fftw_msp_bridge(f,ct1,ct2) bind(c, name="fftw_msp_bridge")
         use iso_c_binding, only: c_int
         integer(c_int),value::f,ct1,ct2
       end subroutine fftw_msp_bridge

       subroutine clfft_msp(f,ct1,ct2) bind(c, name="clfft_msp_cast")
         use iso_c_binding, only: c_int
         integer(c_int),value::f,ct1,ct2
       end subroutine clfft_msp

       subroutine cufft_msp(f,ct1,ct2) bind(c, name="cufft_msp")
         use iso_c_binding, only: c_int
         integer(c_int),value::f,ct1,ct2
       end subroutine cufft_msp
       end interface
      end module msp_c_wrapper


      recursive subroutine msp_init_bridge ()
       use qcbedms_var
       use utils
       use mpi
       use msp_c_wrapper
       implicit none
       logical::fftsg_check,fftw_check,clfft_check,cufft_check
       real(8)::rsum
       integer(c_int)::f
       character(len=8)::list_meth(4)
       integer::list_n(4),i,j,k

       nhyb=0
       hyb_redi=0
       nfftsg=0
       nfftw=0
       nclfft=0
       ncufft=0
       fftsg_check=.false.
       fftw_check=.false.
       clfft_check=.false.
       cufft_check=.false.
       hybrid_opt=.false.
       msp_sgl=.false.
       MPI_rank=0
       mpi_pernode=.false.

       call read_f7
       read(7,*) msp_opt1,msp_opt2,msp_opt3
       call format_case(msp_opt1,"l")

       select case (msp_opt1)
         case ('fftsg','fftw','clfft','cufft')
           nhyb0=1
           allocate(msp_ratio0(nhyb0),msp_meth0(nhyb0),msp_dev0(nhyb0),msp_splt0(nhyb0))

           msp_ratio0(1)=1.d0
           msp_meth0(1)=msp_opt1
           msp_dev0(1)=msp_opt2
           msp_splt0(1)=msp_opt3

           call case_method(1)

           if(msp_meth0(1) == 'clfft' .and. msp_splt0(1) < 0) msp_sgl=.true.

         case ('hybrid','mpi')
           hybrid_opt=.true.
```

Appendix B. Program scripts

```
          if(msp_opt2 == 0) then
            write(13,*) "Number of MSP methods = 0!"
            stop "Number of hybrid Multi-slice Propagation methods = 0!"
          end if

          if(msp_opt1 == 'mpi' .and. msp_opt2 < 0) mpi_pernode=.true.

          nhyb0=abs(msp_opt2)
          hyb_redi=msp_opt3
          allocate(msp_ratio0(nhyb0),msp_meth0(nhyb0),msp_dev0(nhyb0),msp_splt0(nhyb0))

          do f=1,nhyb0
            call read_f7
            read (7,*) msp_ratio0(f),msp_meth0(f),msp_dev0(f),msp_splt0(f)

            call case_method(f)

            if(msp_splt0(f) < 0) stop
     *        "Single precision (negative split) is not supported in hybrid multi-slice!"
          end do

         case ('test')
          msp_test=logical(.true.,kind=c_bool)
          if(msp_opt3 < 0) then
            msp_timer=logical(.true.,kind=c_bool)
            msp_opt3=-msp_opt3
          end if

          call test_msp_methods(msp_opt2,msp_opt3)
          return

         case default
          stop 'Unknown Multi-slice Propagation method!'
        end select


! Count total threads required by all MS methods as 'nhyb'
        nhyb=nfftsg+nfftw+nclfft+ncufft
        allocate(msp_ratio(nhyb),msp_meth(nhyb),msp_dev(nhyb),msp_splt(nhyb),
     *   msp_chid(nhyb))

        list_meth=(/ 'fftsg  ','fftw   ','clfft  ','cufft  ' /)
        list_n=(/ nfftsg,nfftw,nclfft,ncufft /)

! Assign each thread/device of the listed MS methods to an individual slot
        j=0
        do f=1,nhyb0
          do k=1,4
            if(msp_meth0(f) == list_meth(k)) then
              if(msp_dev0(f) == -2 .or. msp_meth0(f)(1:3) == 'fft') then
                do i=1,list_n(k)
                  j=j+1
                  msp_dev(j)=i-1
                  msp_meth(j)=msp_meth0(f); msp_ratio(j)=msp_ratio0(f)
                  msp_splt(j)=msp_splt0(f); msp_chid(j)=i
                enddo
              else
                j=j+1
                msp_dev(j)=msp_dev0(f)
                msp_meth(j)=msp_meth0(f); msp_ratio(j)=msp_ratio0(f)
                msp_splt(j)=msp_splt0(f); msp_chid(j)=i+1
              endif
            endif
          enddo
        enddo

! Initiate MPI
        if(msp_opt1 == 'mpi') call msp_mpi_init()

! Normalise the ratios of tilt assignment
        rsum=sum(msp_ratio)
        msp_ratio(:)=msp_ratio(:)/rsum

        allocate(as_nt(nhyb),as_toff(nhyb),msp_ntilt(nhyb),msp_spd(nhyb))
        allocate(timer_pr(nhyb),timer_ch(nhyb),timer_fft(nhyb),timer_ifft(nhyb),
     *   timer_mpg(nhyb),timer_mpr(nhyb))

! Share global variables to C++;
!   slice_beam related variables are shared in 'oneit' for every iteration
        call Fortran_global_vars_to_C(pgarray,
     *   deltaz,xshift,yshift,applieddilation,cell2rpr,tiltarray,beam,
     *   res2adr,ib,
     *   ay,bee,cee,dee,meshx,meshy,mt,nslice,mbout,nump,numl,res2,tottilts,ncalcs)

        call MSP_global_vars_to_C(msp_ratio,msp_meth,
     *   msp_dev,msp_splt,msp_chid,as_nt,as_toff,msp_ntilt,
     *   MPI_rank,nhyb,hyb_redi,fftw_bat,nfftsg,nfftw,nclfft,ncufft,
     *   msp_sgl)

        call Timer_global_vars_to_C(timer_ch,timer_pr,timer_fft,timer_ifft,timer_mpg,timer_mpr,
     *   msp_timer)

! Setup each MSP library's global environment
#ifdef COMPILE_FFTW
        call fftw_init(pmap0)
        fft_one_lib=""
        if(msp_opt1 /= 'fftsg') then
          fft_one_lib='fftw'
        end if
#endif
        if(nfftw>0) allocate(fftw_pr(mt,fftw_bat,nslice,nfftw))

#ifdef COMPILE_OPENCL
        call clfft_init
#endif
#ifdef COMPILE_CUDA
        call cufft_init
#endif
#ifdef COMPILE_FFTSG
        if(fft_one_lib /= 'fftw') then
          fft_one_lib='fftsg'
          allocate(fsgi(0:2**(int(log(max(meshx,meshy)+0.5d0)/log(2.d0))/2)+1),
     *     fsga(0:2*meshx-1,0:meshy-1),fsgt(0:8*meshy-1),fsgw(0:max(meshx,meshy)/2-1))
          allocate(prarray(mt,nslice))
          fsgi(0)=0
        end if
#endif

        write(13,'("Using ",a," for FFT outside multi-slice propagation.")')
     *   fft_one_lib
        if(hybrid_opt) then
          write(13,'(/,"Using ",i0,2x,a," Multi-slice Propagation items (threads).")')
     *     nhyb,trim(msp_opt1)
          if(hyb_redi>0) then
            write(13,'("Auto-redistribute tilts every ",i0," iterations.")') hyb_redi
          end if
        else
          write(13,'(/,"Using ",a," for multi-slice propagation.")') msp_opt1
        end if

! Initiate each MSP method
        call msp_child_init_bridge()

!       call c_shared_var_test
!       call f_c_var_test
      contains
        subroutine case_method(ff)
          implicit none
          integer(c_int)::ff

          select case (msp_meth0(ff))
            case ('fftsg')
              msp_dev0(ff)=1
              nfftsg=msp_dev0(ff)
              if(hybrid_opt) stop 'FFTSG is not implemented in Hybrid Multi-slice!'

            case ('fftw')
              if(msp_dev0(ff) == 0) msp_dev0(ff)=1
              nfftw=msp_dev0(ff)
              fftw_bat=msp_splt0(ff)
              if(fftw_bat < 1) fftw_bat=1
              if(fftw_check) stop 'Only ONE entity of FFTW is allowed!'
              fftw_check=.true.

            case ('clfft')
              if(msp_dev0(ff) < 0) then
                call OpenCLGetGPU(msp_dev0(ff),nclfft)
                if(clfft_check) stop
     *            'Only ONE entity of clFFT is allowed while device option = -2 (all)!'
                clfft_check=.true.
              else
                nclfft=nclfft+1
              endif

            case ('cufft')
              if(msp_dev0(ff) < 0) then
                call CUDAGetDevices(msp_dev0(ff),ncufft)
                if(cufft_check) stop
     *            'Only ONE entity of cuFFT is allowed while device option = -2 (all)!'
                cufft_check=.true.
              else
                ncufft=ncufft+1
              endif

            case default
              print'("Multi-slice method: ",a)', msp_meth0(ff)
              stop "Unknown multi-slice method!"
          end select
        end subroutine case_method
      end subroutine msp_init_bridge


      subroutine msp_term_bridge()
        use qcbedms_var
        use msp_c_wrapper
        implicit none

        call msp_child_term_bridge()

#ifdef COMPILE_FFTW
        call fftw_term
#endif
#ifdef COMPILE_OPENCL
        call clfft_term
#endif
#ifdef COMPILE_CUDA
        call cufft_term
#endif

        if(msp_opt1 == 'mpi') call msp_mpi_term()

        if(allocated(fsga)) deallocate(fsga,fsgw,fsgi,fsgt)
        if(allocated(as_nt)) deallocate(as_nt,as_toff,msp_ntilt,msp_spd)

        if(allocated(msp_ratio)) deallocate(msp_ratio,msp_meth,msp_dev,msp_splt,msp_chid)
        if(allocated(msp_ratio0)) deallocate(msp_ratio0,msp_meth0,msp_dev0,msp_splt0)
        if(allocated(fftw_pr)) deallocate(fftw_pr)
        if(allocated(prarray)) deallocate(prarray)
```

Appendix B. Program scripts

```
      deallocate(timer_ch,timer_pr,timer_fft,timer_ifft,timer_mpg,timer_mpr)
    end subroutine msp_term_bridge


    subroutine msp_child_init_bridge()
      use qcbedms_var
      use msp_c_wrapper
      implicit none
      integer(c_int)::f,offset,n
! Heterogeneous job & memory allocation, Pt. 1 - Global
! 1. Distribute all tilts to be calculated to each child according to the allocated ratios:
      as_nt(:)=floor(tottilts*msp_ratio(:))
!    If there is any residual tilts caused by flooring, increase tilts from the 1st child:
      do f=1,tottilts-sum(as_nt)
        as_nt(f)=as_nt(f)+1
      end do
!    Note - tottilts === sum( as_nt )
! 2. Range of calculating tilts distributed to Child 'f' -
!    ( 1 + as_toff(f) : as_nt(f) + as_toff(f) ):
      as_toff=0
      do f=2,nhyb
        if(as_nt(f) > 0) as_toff(f)=as_toff(f-1)+as_nt(f-1)
      end do
! 3. Numbers of parallelised tilts for each child will be determined by each's restrictions
!    during their initialisations, satisfying the following rule:
!      msp_ntilt(:) = ceil( as_nt(:) / msp_splt(:) )
!  The memory usage (GPU memory for GPU methods) of each child is dominated by
!      'msp_ntilt(:)' - the number of paralleling tilts of each child.
!  Note - msp_ntilt(:) * msp_splt(:) >= as_nt(:) because of ceiling

      offset=0; n=nhyb
      if(msp_opt1 == 'mpi') then
        call mpi_calc_tilt_init()
        offset=mhyb_oset(mpir); n=mhyb(mpir)+mhyb_oset(mpir)
      end if

      do f=1+offset,n
        write(13,'(i0,". ",a," - ",f6.2,"%, ",i0," tilts")')
    *     f,msp_meth(f),msp_ratio(f)*100,as_nt(f)

        select case (msp_meth(f))
        case ("fftsg")
#ifndef COMPILE_FFTSG
          stop "QCBEDMS-PF was compiled without FFTSG!"
#endif
          msp_ntilt(f)=1

        case ("fftw")
#ifdef COMPILE_FFTW
          call fftw_child_init(f-1)
#else
          stop "QCBEDMS-PF was compiled without FFTW!"
#endif

        case ("clfft")
#ifdef COMPILE_OPENCL
          call clfft_child_init(f-1)
#else
          stop "QCBEDMS-PF was compiled without OpenCL!"
#endif

        case ("cufft")
#ifdef COMPILE_CUDA
          call cufft_child_init(f-1)
#else
          stop "QCBEDMS-PF was compiled without CUDA!"
#endif

        case default
          write(*,'("Multi-slice Propagation method: ",a)') msp_meth(f)
          stop "Unknown Multi-slice Propagation method!"
        end select
      end do
    contains
      subroutine mpi_calc_tilt_init()
#ifdef MPI
        use mpi
        integer::nct,ct1,ierr

! Summarise tilts of MPI slave threads
        nct=sum(as_nt(1+mhyb_oset(mpir):mhyb(mpir)+mhyb_oset(mpir)))
        ct1=as_toff(1+mhyb_oset(mpir))

        call
MPI_ALLGATHER(nct,1,MPI_INTEGER,mas_nt,1,MPI_INTEGER,MPI_COMM_WORLD,ierr)
        call
MPI_ALLGATHER(ct1,1,MPI_INTEGER,mas_toff,1,MPI_INTEGER,MPI_COMM_WORLD,ierr)
!       print*,'mpir',mpir,'mas_toff',mas_toff,'c',as_toff
#endif
      end subroutine mpi_calc_tilt_init

    end subroutine msp_child_init_bridge


    subroutine msp_child_term_bridge()
      use qcbedms_var
      use msp_c_wrapper
      implicit none
      integer::f,offset,n

      offset=0; n=nhyb
      if(msp_opt1 == 'mpi') then
        offset=mhyb_oset(mpir); n=mhyb(mpir)+mhyb_oset(mpir)
      end if
      do f=1+offset,n
        select case (msp_meth(f))
          case ("fftsg")
            ! Nothing to do
          case ("fftw")
#ifdef COMPILE_FFTW
            call fftw_child_term(f-1)
#endif
          case ("cufft")
#ifdef COMPILE_CUDA
            call cufft_child_term(f-1)
#endif
          case ("clfft")
#ifdef COMPILE_OPENCL
            call clfft_child_term(f-1)
#endif
        end select
      end do
    end subroutine msp_child_term_bridge


    subroutine msp_redi_calc_ntilt()
      use qcbedms_var
      implicit none
      integer::f
      real(8)::rsum=0

      write(13,'(/,"Redistributing tilts of Multi-slice Propagation methods")')
      msp_ratio(1:nhyb)=msp_spd(:)/hyb_redi
      write(13,'("Measured speeds: ",100(i0,". ",a8," - ",g14.5," tilt/s, "))')
    *   (f,msp_meth(f),msp_ratio(f),f=1,nhyb)

      rsum=sum(msp_ratio(1:nhyb))
      msp_ratio(1:nhyb)=msp_ratio(1:nhyb)/rsum
      write(13,'("Ratio updated: ",100(i0,". ",a8," - ",f7.2,"%, "))')
    *   (f,msp_meth(f),msp_ratio(f)*100,f=1,nhyb)

      call msp_child_term_bridge()
      call msp_child_init_bridge()
      write(13, '("Multi-slice Propagation tilts redistributed!",/)')
    end subroutine msp_redi_calc_ntilt


    subroutine msp_exec_bridge()
      use qcbedms_var
      use msp_c_wrapper
      implicit none

      timer_pr=0; timer_fft=0; timer_ifft=0; timer_mpg=0; timer_mpr=0

      select case (msp_opt1)
        case ('mpi')
          call msp_mpi_exec
        case ('hybrid')
          call msp_cppthread_exec(nhyb,0)
        case ('fftsg','fftw','clfft','cufft')
          call msp_child_exec_bridge(1)
      end select
    end subroutine msp_exec_bridge


    subroutine msp_mpi_init()
#ifdef MPI
      use qcbedms_var
      use mpi
      implicit none
      integer::ierr,r
      character(len=8,kind=c_char),target,allocatable::imeth(:)
      integer(c_int),target,allocatable::idev(:),isplt(:),ichid(:)
      real(c_double),target,allocatable::iratio(:)
      logical::initd

      call MPI_INITIALIZED(initd,ierr)
      if(.not. initd) call MPI_INIT(ierr)
      call MPI_COMM_RANK(MPI_COMM_WORLD,MPI_rank,ierr)
      call MPI_COMM_SIZE(MPI_COMM_WORLD,nmpi,ierr)

      mpir=MPI_rank+1

      allocate(mhyb(nmpi),mhyb_oset(nmpi),mas_nt(nmpi),mas_toff(nmpi))
      allocate(iratio,source=msp_ratio)
      allocate(imeth,source=msp_meth)
      allocate(idev,source=msp_dev)
      allocate(isplt,source=msp_splt)
      allocate(ichid,source=msp_chid)

! Gather & sync number of MSP methods
      call
MPI_ALLGATHER(nhyb,1,MPI_INTEGER,mhyb,1,MPI_INTEGER,MPI_COMM_WORLD,ierr)
      nhyb=sum(mhyb)

! Prepare the new MSP method containers
      deallocate(msp_ratio,msp_meth,msp_dev,msp_splt,msp_chid)
      allocate(msp_ratio(nhyb),msp_meth(nhyb),msp_dev(nhyb),msp_splt(nhyb),msp_chid(nhyb))

! Prepare displacements
      mhyb_oset(1)=0
      do r=2,nmpi
        mhyb_oset(r)=mhyb_oset(r-1)+mhyb(r-1)
      end do

! Gather & sync parameters of MSP methods
      call MPI_ALLGATHERV(iratio,mhyb(mpir),MPI_DOUBLE_PRECISION,
```

Appendix B. Program scripts

```
     *   msp_ratio,mhyb,mhyb_oset,MPI_DOUBLE_PRECISION,MPI_COMM_WORLD,ierr)
         call MPI_ALLGATHERV(imeth,8*mhyb(mpir),MPI_CHARACTER,
     *   msp_meth,8*mhyb,8*mhyb_oset,MPI_CHARACTER,MPI_COMM_WORLD,ierr)
         call MPI_ALLGATHERV(idev,mhyb(mpir),MPI_INTEGER,
     *   msp_dev,mhyb,mhyb_oset,MPI_INTEGER,MPI_COMM_WORLD,ierr)
         call MPI_ALLGATHERV(isplt,mhyb(mpir),MPI_INTEGER,
     *   msp_splt,mhyb,mhyb_oset,MPI_INTEGER,MPI_COMM_WORLD,ierr)
         call MPI_ALLGATHERV(ichid,mhyb(mpir),MPI_INTEGER,
     *   msp_chid,mhyb,mhyb_oset,MPI_INTEGER,MPI_COMM_WORLD,ierr)

         deallocate(iratio,imeth,idev,isplt,ichid)

!        print*,'rank=',MPI_rank,msp_dev
!        print*,'rank=',MPI_rank,msp_meth
#else
         stop 'QCBEDMS-PF was compiled without MPI!'
#endif
       end subroutine msp_mpi_init


       subroutine program_mpi_term()
#ifdef MPI
         use mpi
         integer::ierr
         call MPI_FINALIZE(ierr)
#endif
       end subroutine program_mpi_term


       subroutine msp_mpi_term()
#ifdef MPI
         use qcbedms_var
         use mpi
         if(MPI_rank == 0) call msp_mpi_loops(.true.,p)
         deallocate(mhyb,mhyb_oset,mas_nt,mas_toff)
#endif
       end subroutine msp_mpi_term


       subroutine msp_mpi_loops(opt,pt)
#ifdef MPI
         use qcbedms_var
         use mpi
         integer::ierr,l
         logical::opt
         real(8)::pt(npar)

         l=-1
         if(opt) l=-2
! Broadcase the 'loops' parameter to MPI slave threads
         call MPI_BCAST(l,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
! Broadcase parameters for current iteration
         call MPI_BCAST(PT,npar,MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)
!         print*,'MPI_rank',MPI_rank,pt

         if(MPI_rank > 0) loops=l
!         print*,MPI_rank,i,loops,ncalcs,initd,finad
#endif
       end subroutine msp_mpi_loops


       subroutine msp_mpi_exec()
#ifdef MPI
         use qcbedms_var
         use mpi
         use msp_c_wrapper
         implicit none
         integer::ierr=0,b

! Execute MSP
         if(mpi_pernode) then
           call msp_cppthread_exec(mhyb(mpir),mhyb_oset(mpir))
         else
           call msp_child_exec_bridge(mpir)
         end if

! Gather calculated tilts from MPI slave threads
         do b=1,mbout
           call MPI_GATHERV(beam(mas_toff(mpir)+1,b),mas_nt(mpir),MPI_DOUBLE_PRECISION,
     *     beam(:,b),mas_nt,mas_toff,MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)
         end do
!         print'(2i4,1000g15.7)',MPI_rank,ncalcs,beam(:,:)

! Gather Timers
           call
MPI_GATHERV(timer_ch(1+mhyb_oset(mpir)),mhyb(mpir),MPI_DOUBLE_PRECISION,
     *     timer_ch,mhyb,mhyb_oset,MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)
           call
MPI_GATHERV(timer_pr(1+mhyb_oset(mpir)),mhyb(mpir),MPI_DOUBLE_PRECISION,
     *     timer_pr,mhyb,mhyb_oset,MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)
           call
MPI_GATHERV(timer_fft(1+mhyb_oset(mpir)),mhyb(mpir),MPI_DOUBLE_PRECISION,
     *     timer_fft,mhyb,mhyb_oset,MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)
           call
MPI_GATHERV(timer_ifft(1+mhyb_oset(mpir)),mhyb(mpir),MPI_DOUBLE_PRECISION,
     *     timer_ifft,mhyb,mhyb_oset,MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)
           call
MPI_GATHERV(timer_mpg(1+mhyb_oset(mpir)),mhyb(mpir),MPI_DOUBLE_PRECISION,
     *     timer_mpg,mhyb,mhyb_oset,MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)
           call
MPI_GATHERV(timer_mpr(1+mhyb_oset(mpir)),mhyb(mpir),MPI_DOUBLE_PRECISION,
     *     timer_mpr,mhyb,mhyb_oset,MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)
#endif
       end subroutine msp_mpi_exec
```

```
       recursive subroutine msp_child_exec_bridge(f)
         use iso_c_binding,only:c_int
         interface
           recursive subroutine msp_child_exec_bridge_f(f)
     *     bind(c,name='msp_child_exec_bridge_f')
             use iso_c_binding,only:c_int
             integer(c_int),value::f
           end subroutine msp_child_exec_bridge_f
         end interface

         integer(c_int)::f
         call msp_child_exec_bridge_f(f)
       end subroutine msp_child_exec_bridge


       recursive subroutine msp_child_exec_bridge_f(f) bind(c,name='msp_child_exec_bridge_f')
         use utils
         use qcbedms_var
         use msp_c_wrapper
         implicit none
         integer(c_int),value::f
         integer(c_int)::ct1,ct2
         integer(8)::t_ch

         call system_clock(t_ch)

         ct1=as_toff(f)
         ct2=as_toff(f)+as_nt(f)

         select case (msp_meth(f))
           case ("fftsg")
#ifdef COMPILE_FFTSG
             call fftsg_msp(f)
#endif

           case ("fftw")
#ifdef COMPILE_FFTW
             call fftw_msp_bridge(f-1,ct1,ct2)
#endif

           case ("clfft")
#ifdef COMPILE_OPENCL
             call clfft_msp(f-1,ct1,ct2)
#endif

           case ("cufft")
#ifdef COMPILE_CUDA
             call cufft_msp(f-1,ct1,ct2)
#endif
         end select
!        write(*,'(1000g25.16)') beam(:,:)
         timer_ch(f)=timer(t_ch)
       end subroutine msp_child_exec_bridge_f
```

b) C++ functions

```
/*
 * Common C++ functions of QCBEDMS-PF
 */

#ifdef SINCOS
# ifndef _GNU_SOURCE
#     define _GNU_SOURCE
# endif
# ifdef INTEL_MATH_LIB
#     include <mathimf.h>
# endif
extern "C" void c_sincos(double *x, double *sn, double *cs)
{
   sincos(*x, sn, cs);
}
#endif

#include "cpp_comm.hpp"

Fortran_global_vars F;
MSP_global_vars MSP;
Timer_global_vars Tr;

extern "C" void Fortran_global_vars_to_C(complex<double> *pg,
       double *deltaz, double *xshift, double *yshift, double *applieddilation,
       double *cell2rpr, double *tiltarray, double *beam,
       int *res2adr, int *ib,
       int ay, int bee, int cee, int dee, int meshx, int meshy, int mt, int nslice, int mbout,
       int nump, int numl, int res2, int tottilts, int ncalcs)
{
   F.pg = pg;
   F.deltaz = deltaz, F.xshift = xshift, F.yshift = yshift, F.applieddilation = applieddilation,
        F.cell2rpr = cell2rpr, F.tiltarray = tiltarray, F.beam = beam;
   F.res2adr = res2adr, F.ib = ib;
   F.ay = ay, F.bee = bee, F.cee = cee, F.dee = dee, F.meshx = meshx, F.meshy = meshy,
        F.mt = mt, F.nslice = nslice, F.mbout = mbout, F.nump = nump, F.numl = numl,
        F.res2 = res2, F.tottilts = tottilts,
        F.ncalcs = ncalcs;
}

extern "C" void Slice_beam_global_vars_to_C(double *sl_beam, int *mseq, int sl_hi, int sl_lo,
       bool sl_opt)
{
```

Appendix B. Program scripts

```
    F.sl_beam = sl_beam, F.mseq = mseq, F.sl_hi = sl_hi, F.sl_lo = sl_lo, F.sl_opt = sl_opt;
}

extern "C" void MSP_global_vars_to_C(double *ratio, char *meth,
        int *dev, int *splt, int *chid, int *as_nt, int *as_toff, int *ntilt,
        int MPI_rank, int nhyb, int hyb_redi, int fftw_bat,
        int nfftsg, int nfftw, int nclfft, int ncufft,
        bool sgl)
{
    MSP.ratio = ratio;
    MSP.meth = meth;
    MSP.dev = dev, MSP.splt = splt, MSP.chid = chid, MSP.as_nt = as_nt,
          MSP.as_toff = as_toff, MSP.ntilt = ntilt;
    MSP.MPI_rank = MPI_rank, MSP.nhyb = nhyb, MSP.hyb_redi = hyb_redi, MSP.fftw_bat =
fftw_bat,
          MSP.nfftsg = nfftsg, MSP.nfftw = nfftw, MSP.nclfft = nclfft, MSP.ncufft = ncufft;
    MSP.sgl = sgl;
}

extern "C" void Timer_global_vars_to_C(double *ch, double *pr, double *fft, double *ifft,
        double *mpg, double *mpr, bool msp)
{
    Tr.ch = ch, Tr.pr = pr, Tr.fft = fft, Tr.ifft = ifft, Tr.mpg = mpg, Tr.mpr = mpr;
    Tr.msp = msp;
}

extern "C" void F_C_var_test()
{
    printf("Variables read by C:\n");
    printf("meshx meshy mt = %d %d %d, nslice = %d,\n"
          "mbout = %d, nump numl res2 = %d %d %d, tottilts = %d, \n"
          "nhyb hyb_redi = %d %d, sl_hi sl_lo = %d %d,\n"
          "msp_sgl sl_opt msp_timer = %s %s %s\n",
          F.meshx, F.meshy, F.mt, F.nslice,
          F.mbout, F.nump, F.numl, F.res2, F.tottilts,
          MSP.nhyb, MSP.hyb_redi, F.sl_hi, F.sl_lo,
          MSP.sgl ? "T" : "F", F.sl_opt ? "T" : "F", Tr.msp ? "T" : "F");
    printf("msp_meth = ");
    for (int f = 0 ; f < MSP.nhyb ; f++) {
      char msp_meth[9] = "";
      strncpy(msp_meth, MSP.meth + f * 8, 8);
      printf("%d. '%s', ", f, msp_meth);
    }
    cout << endl;
}

size_t size_R, size_C;

size_t msp_mem(const int f, const int a)
{
    size_R = MSP.sgl ? sizeof(float) : sizeof(double);
    size_C = MSP.sgl ? sizeof(complex<float> ) : sizeof(complex<double> );

    size_t pg = F.mt * F.nslice * size_C;
    size_t pr = F.mt * MSP.ntilt[f] * F.nslice * size_C;
    size_t pmap = F.mt * MSP.ntilt[f] * size_C;
    size_t beam = F.res2 * F.mbout * size_R;

    if (strncmp(MSP.meth + f * 8, "fftw", 4) == 0) {
      pmap = F.mt * a * size_C;
      return pg + pr + pmap * 2;
    }

    return pg + pr + pmap * 2 + beam + MSP.as_nt[f] * sizeof(int);
}

extern "C" void save_as_bmp(int w, int h, double* data, char* name)
{
    unsigned long long filesize = 1078 + w * h;

    unsigned char fileheader[14] = { 'B', 'M', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
    unsigned char infoheader[40] = { 40, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 8, 0 };
    unsigned char rgbquad[1024] = { 0, 255, 0, 0 }; // map '0' indexed color to Magenta
    unsigned char bmppad[3] = { 0, 0, 0 };

    unsigned char* img = new unsigned char[w * h];
    for (int i = 0 ; i < w * h ; i++)
      img[i] = (unsigned char) data[i];

    // '0' was mapped to Magenta, '1' to '255' are still grayscale
    for (int i = 1 ; i < 256 ; i++) {
      rgbquad[4 * i] = i;
      rgbquad[4 * i + 1] = i;
      rgbquad[4 * i + 2] = i;
      rgbquad[4 * i + 3] = 0;
    }

    fileheader[2] = (unsigned char) (filesize);
    fileheader[3] = (unsigned char) (filesize >> 8);
    fileheader[4] = (unsigned char) (filesize >> 16);
    fileheader[5] = (unsigned char) (filesize >> 24);
    fileheader[10] = (unsigned char) (1078);
    fileheader[11] = (unsigned char) (1078 >> 8);
    fileheader[12] = (unsigned char) (1078 >> 16);
    fileheader[13] = (unsigned char) (1078 >> 24);

    infoheader[4] = (unsigned char) (w);
    infoheader[5] = (unsigned char) (w >> 8);
    infoheader[6] = (unsigned char) (w >> 16);
    infoheader[7] = (unsigned char) (w >> 24);
    infoheader[8] = (unsigned char) (h);
    infoheader[9] = (unsigned char) (h >> 8);
    infoheader[10] = (unsigned char) (h >> 16);
    infoheader[11] = (unsigned char) (h >> 24);
    infoheader[32] = (unsigned char) (256);
    infoheader[33] = (unsigned char) (256 >> 8);
    infoheader[34] = (unsigned char) (256 >> 16);
    infoheader[35] = (unsigned char) (256 >> 24);

    string filename = "";
    filename += name;
    filename += ".bmp";
    FILE* f = fopen(filename.c_str(), "wb");
    fwrite(fileheader, 1, 14, f);
    fwrite(infoheader, 1, 40, f);
    fwrite(rgbquad, 1, 1024, f);
    for (int j = 0 ; j < h ; j++) {
      fwrite(&img[(h - j - 1) * w], 1, w, f);
      fwrite(bmppad, 1, (4 - w % 4) % 4, f);
    }
    fclose(f);

    delete[] img;
}

typedef chrono::time_point<chrono::high_resolution_clock> time_point;

time_point time_now()
{
    return chrono::high_resolution_clock::now();
}

double timer(time_point* t0)
{
    time_point t1 = time_now(), t2 = *t0;
    *t0 = t1;
    return chrono::duration<double>(t1 - t2).count();
}

extern "C" void c2fwrite(int, char*);

void fortran_write(const int u, const char *__restrict format, ...)
{
    if (MSP.MPI_rank > 0) return;
    va_list args;
    va_start(args, format);
    char fmsg[1024];
    vsprintf(fmsg, format, args);
    c2fwrite(13, fmsg);
}

extern "C" void msp_child_exec_bridge_f(const int f);

extern "C" void msp_cppthread_exec(const int nf, const int offset)
{
    thread *hybrid = new thread[nf];
    for (int f = 0 ; f < nf ; f++) {
      hybrid[f] = thread(msp_child_exec_bridge_f, f + 1 + offset);
//    hybrid[f].join();
    }

    for (int f = 0 ; f < nf ; f++) {
      hybrid[f].join();
    }
    delete[] hybrid;
}
```

B.1.2. Fresnel propagation using FFTW

a) Fortran subroutine "fftw_msp" – main body of multi-slice propagation using FFTW

```
     recursive subroutine fftw_msp(f,ct1,ct2,pmap) bind(c,name='fftw_msp')
#ifdef COMPILE_FFTW
       use utils
       use qcbedms_var
       implicit none
       interface
         subroutine fftw_fft(f,dir) bind(c,name='fftw_fft')
           use iso_c_binding, only: c_int
           integer(c_int),value::f,dir
         end subroutine fftw_fft

         subroutine fftw_time_now(f) bind(c,name='fftw_time_now')
           use iso_c_binding, only: c_int
           integer(c_int),value::f
         end subroutine fftw_time_now

         function fftw_timer(f) bind(c,name='fftw_timer')
           use iso_c_binding, only: c_int,c_double
           integer(c_int),value::f
           real(c_double)::fftw_timer
         end function fftw_timer
       end interface

       integer(c_int),value::f,ct1,ct2
       complex(c_double_complex)::pmap(mt,fftw_bat)
       integer::k,t,t1,kpg,ct3,j,ifftw
```

Appendix B. Program scripts

```
      real(8)::dzdil(nslice)

      ifftw=msp_chid(f)
      dzdil=deltaz*APPLIEDDILATION

      do t=ct1,ct2,fftw_bat
       ct3=min(t+fftw_bat-1,as_nt(f)+as_toff(f))

! Prepare the Fresnel propagators 'pr'
       if(msp_timer) call fftw_time_now(f-1)
       do t1=t,ct3
        DO k=1,NSLICE
         CALL PROPG(k,TILTARRAY(:,t1,k),
     *      dzdil(k),XSHIFT(k),YSHIFT(k),fftw_pr(:,t1-t+1,k,ifftw))
        END DO
       end do
       if(msp_timer) timer_pr(f)=timer_pr(f)+fftw_timer(f-1)

       pmap(1:mt,1:fftw_bat)=cmplx(1.d0,0.d0,kind=8)

       if(msp_timer) call fftw_time_now(f-1)
       DO k=1,sl_hi
        kpg=MSEQ(k)
        do t1=1,fftw_bat
         pmap(1:mt,t1)=pmap(1:mt,t1)*pgarray(1:mt,kpg)
        end do
        if(msp_timer) timer_mpg(f)=timer_mpg(f)+fftw_timer(f-1)

        call fftw_fft(f-1,-1)
        if(msp_timer) timer_fft(f)=timer_fft(f)+fftw_timer(f-1)

        pmap(1:mt,:)=pmap(1:mt,:)*fftw_pr(1:mt,:,mseq(k),ifftw)
        if(msp_timer) timer_mpr(f)=timer_mpr(f)+fftw_timer(f-1)

        if((sl_opt .and. k >= sl_lo) .or. k == sl_hi) then
         do j=1,mbout
          if(sl_opt) then
           sl_beam(t:ct3,j,k-sl_lo+1)
     *      =real(pmap(ib(j),1:ct3-t+1))**2
     *      +aimag(pmap(ib(j),1:ct3-t+1))**2
          else
           beam(t:ct3,j)
     *      =real(pmap(ib(j),1:ct3-t+1))**2
     *      +aimag(pmap(ib(j),1:ct3-t+1))**2
          end if
         end do
         if(k == sl_hi) exit
        end if

        call fftw_fft(f-1,1)
        if(msp_timer) timer_ifft(f)=timer_ifft(f)+fftw_timer(f-1)
       end do
      end do
#endif
    end subroutine fftw_msp
```

b) File "fftw_func.cpp" – FFTW C++ functions

```
/*
 * FFTW functions of QCBEDMS-PF
 */
#include "cpp_comm.hpp"
#include <fftw3.h>

using namespace std;
typedef fftw_complex CD;

struct fftw_vars {
   CD *pmap;
   fftw_plan msf, msb;
   time_point t_msp;
} *fwv;

fftw_plan p1f, p1b;

extern "C"
{
void fftw_init(complex<double>* pmap0)
{
   fwv = new fftw_vars[MSP.nhyb];
   p1f = fftw_plan_dft_2d(F.meshy, F.meshx,
         reinterpret_cast<CD*>(pmap0), reinterpret_cast<CD*>(pmap0),
         FFTW_FORWARD, FFTW_PATIENT);
   p1b = fftw_plan_dft_2d(F.meshy, F.meshx,
         reinterpret_cast<CD*>(pmap0), reinterpret_cast<CD*>(pmap0),
         FFTW_BACKWARD, FFTW_PATIENT);
}

void fftw_term()
{
   delete[] fwv;
   fftw_destroy_plan(p1f);
   fftw_destroy_plan(p1b);
}

void fftw_one(const int dir)
{
   if (dir == -1)
      fftw_execute(p1f);
   else if (dir == 1)
      fftw_execute(p1b);
}

void fftw_child_init(const int f)
{
   MSP.ntilt[f] = MSP.fftw_bat;

   fortran_write(13, "  FFTW will calculate %d tilts; "
         "calculating %d tilt(s) at once, using at least %g MB\n",
         MSP.as_nt[f], MSP.fftw_bat, msp_mem(f, MSP.fftw_bat) / 1024.0 / 1024);

   int n[2] = { F.meshy, F.meshx };
   fwv[f].pmap = fftw_alloc_complex(F.mt * MSP.fftw_bat);
   fwv[f].msf = fftw_plan_many_dft(2, n, MSP.fftw_bat,
         fwv[f].pmap, n, 1, F.mt, fwv[f].pmap, n, 1, F.mt,
         FFTW_FORWARD, FFTW_PATIENT);
   fwv[f].msb = fftw_plan_many_dft(2, n, MSP.fftw_bat,
         fwv[f].pmap, n, 1, F.mt, fwv[f].pmap, n, 1, F.mt,
         FFTW_BACKWARD, FFTW_PATIENT);
}

void fftw_child_term(const int f)
{
   fftw_free(fwv[f].pmap);
   fftw_destroy_plan (fwv[f].msf);
   fftw_destroy_plan (fwv[f].msb);
}

void fftw_time_now(const int f)
{
   fwv[f].t_msp = time_now();
}

double fftw_timer(const int f)
{
   return timer(&fwv[f].t_msp);
}

void fftw_fft(const int f, const int dir)
{
   if(dir == -1)
      fftw_execute(fwv[f].msf);
   else
      fftw_execute(fwv[f].msb);
}

void fftw_msp(const int f, const int ct1, const int ct2, CD *pmap);

void fftw_msp_bridge(const int f, const int ct1, const int ct2)
{
   fftw_msp(f+1,ct1+1,ct2+1,fwv[f].pmap);
}
}
```

B.1.3. Fresnel propagation using clFFT

```
/*
 * OpenCL functions of QCBEDMS-PF
 */

#define CL_USE_DEPRECATED_OPENCL_1_2_APIS
#include <clFFT.h>
#include "cpp_comm.hpp"

using namespace std;

struct clfft_vars {
   cl_context ctx;
   cl_command_queue q;
   cl_mem pmap, ptmp, *pg, *pr, beam, ib, tiltarray;
   cl_program program;
   cl_kernel propg, pad0oshift, fill_pmap, fill_beam, beam_out;
   clfftPlanHandle *plans_f, plan_b;
   size_t G_pmap, L_pmap, G_beam_out[2] = { 1, 1 }, L_beam_out[2] = { 1, 1 },
         G_propg[3] = { 1, 1, 1 }, L_propg[3] = { 1, 1, 1 }, G_pad, L_pad;
   complex<float> *host_pgf;
}*clv;

extern "C"
{
void clfft_init(int *count)
{
   clv = new clfft_vars[MSP.nhyb];
}

void clfft_term()
{
   delete[] clv, clv = NULL;
}

void cl_dbg(string msg, cl_int err)
{
   if(err != 0)
      printf("%s error code = %d\n", msg.c_str(), err);
}

cl_device_id OpenCLGetGPU(int idev, int *gpu_count)
{
   cl_device_id dev_id;
```

Appendix B. Program scripts

```
   cl_uint num_plat;
   clGetPlatformIDs(0, NULL, &num_plat);
   cl_platform_id *plats = new cl_platform_id[num_plat];
   clGetPlatformIDs(num_plat, plats, NULL);

   cl_uint num_dev = 0;
   cl_uint idev_gb = 0;
   char plat_name[128];
   char device_name[128];
   char vend_name[128];
   int count = 0;

   if (idev == -1)
      printf("OpenCL GPU devices:\n");

   for (cl_uint i = 0 ; i < num_plat ; i++, num_dev = 0) {
      clGetPlatformInfo(plats[i], CL_PLATFORM_NAME,
            sizeof(plat_name), plat_name, NULL);
      clGetDeviceIDs(plats[i], CL_DEVICE_TYPE_GPU, 0, NULL, &num_dev);
      count += num_dev;

      cl_device_id* devices = new cl_device_id[num_dev];
      clGetDeviceIDs(plats[i], CL_DEVICE_TYPE_GPU, num_dev, devices, NULL);

      for (cl_uint j = 0 ; j < num_dev ; j++, idev_gb++) {
         clGetDeviceInfo(devices[j], CL_DEVICE_NAME,
               sizeof(device_name), device_name, NULL);
         clGetDeviceInfo(devices[j], CL_DEVICE_VENDOR,
               sizeof(vend_name), vend_name, NULL);
         if (idev == -1)
            printf("  %i. %s, Vendor: %s, Platform: %s.\n",
                  idev_gb, device_name, vend_name, plat_name);
         if (idev == (int) idev_gb) dev_id = devices[j];
      }

      delete[] devices, devices = NULL;
   }

   delete[] plats, plats = NULL;

   if (idev == -1)
      exit(0);
   else if (idev == -2) {
      *gpu_count = count;
      return 0;
   }

   cl_platform_id plat;
   clGetDeviceInfo(dev_id, CL_DEVICE_NAME, sizeof(device_name), device_name, NULL);
   clGetDeviceInfo(dev_id, CL_DEVICE_VENDOR, sizeof(vend_name), vend_name, NULL);
   clGetDeviceInfo(dev_id, CL_DEVICE_PLATFORM, sizeof(plat), &plat, NULL);
   clGetPlatformInfo(plat, CL_PLATFORM_NAME, sizeof(plat_name), plat_name, NULL);
   fortran_write(13, "  Device: %d. %s, Vendor: %s, Platform/Driver: %s. \n",
         idev, device_name, vend_name, plat_name);

   return dev_id;
}

void clfft_child_init(const int f)
{
   cl_int err;
   cl_device_id dev_id = OpenCLGetGPU(MSP.dev[f], NULL);
   cl_platform_id plat;
   err = clGetDeviceInfo(dev_id, CL_DEVICE_PLATFORM, sizeof(plat), &plat, NULL);

/* Check double precision support on GPU */
   fortran_write(13, "  clFFT precision: %s.\n", MSP.sgl ? "single" : "double");
   cl_device_fp_config fp;
   err = clGetDeviceInfo(dev_id, CL_DEVICE_DOUBLE_FP_CONFIG, sizeof(fp), &fp, NULL);
   if (fp == 0 && !MSP.sgl) {
      fortran_write(13, "\nDouble floating point precision not supported by the device!");
      cout << "Double floating-point precision not supported by the device!" << endl;
      exit(-1);
   }

/* Calculate GPU memory usage */
   float MB = 1.0 / 1024 / 1024;
   size_t total_gm, max_alloc, work_gm;
   cl_int mem_err1 = clGetDeviceInfo(dev_id, CL_DEVICE_GLOBAL_MEM_SIZE,
         sizeof(total_gm), &total_gm, NULL);
   cl_int mem_err2 = clGetDeviceInfo(dev_id, CL_DEVICE_MAX_MEM_ALLOC_SIZE,
         sizeof(max_alloc), &max_alloc, NULL);
   if (MSP.splt[f] == 0) {
      MSP.splt[f] = 1, MSP.ntilt[f] = MSP.as_nt[f];
      /* Apply total memory and max allocation criterion */
      while (msp_mem(f, 0) > total_gm && F.mt * MSP.ntilt[f] * size_C > max_alloc
            && MSP.splt[f] < MSP.as_nt[f])
         ++MSP.splt[f], MSP.ntilt[f] = (int) ceil((float) MSP.as_nt[f] / MSP.splt[f]);
      fortran_write(13, "  (AUTO)");
   }
   else
      MSP.ntilt[f] = (int) ceil((float) MSP.as_nt[f] / MSP.splt[f]);

   fortran_write(13, "  Paralleling %d/%d tilts; will loop %d time(s).\n",
         MSP.ntilt[f], MSP.as_nt[f], MSP.splt[f]);
   work_gm = msp_mem(f, 0);
   fortran_write(13, "  Total GPU memory: %g MB; clFFT will use about %g MB.\n",
         total_gm * MB, work_gm * MB);
   fortran_write(13, "  The max GPU buffer is %g MB (Device's limit is %g MB).\n",
         F.mt * MSP.ntilt[f] * size_C * MB, max_alloc * MB);

   if (work_gm > total_gm || F.mt * MSP.ntilt[f] * size_C > max_alloc) {
      if (F.ncalcs < 1) {
         fortran_write(13, "\nNOT enough GPU memory OR the max array excesses GPU's limit! "
               "\nPlease reduce paralleled tilts manually.\n");
         cout << "NOT enough GPU memory OR the max GPU array excesses device's limit! "
               "Please reduce paralleled tilts manually." << endl;
         exit(-1);
      } else if (mem_err1 != CL_SUCCESS || mem_err2 != CL_SUCCESS) {
         fortran_write(13, "\nWARNING: \"OpenCL get memory info\" failed "
               "during refinement! Ignored and continue.\n");
      }
   }
   if (work_gm > total_gm * 0.5)
      fortran_write(13, "  GPU memory may be intensive; reduce paralleled "
            "tilts manually if any error occurs.\n");

/* Initiate memory objects */
   err = clGetDeviceInfo(dev_id, CL_DEVICE_PLATFORM, sizeof(plat), &plat, NULL);
   cl_context_properties props[3] = { CL_CONTEXT_PLATFORM, 0, 0 };
   props[1] = (cl_context_properties) plat;
   clv[f].ctx = clCreateContext(props, 1, &dev_id, NULL, NULL, &err);
   cl_dbg("clCreateContext", err);
   clv[f].q = clCreateCommandQueue(clv[f].ctx, dev_id, 0, &err);
   cl_dbg("clCreateCommandQueue", err);

   if(MSP.sgl)
      clv[f].host_pgf = new complex<float>[F.mt * F.nslice];
   clv[f].pg = new cl_mem[F.nslice];
   clv[f].pr = new cl_mem[F.nslice];
   clv[f].tiltarray = clCreateBuffer(clv[f].ctx,
         CL_MEM_READ_ONLY | CL_MEM_HOST_WRITE_ONLY,
         2 * MSP.ntilt[f] * size_R, NULL, &err);
   for (int i = 0 ; i < F.nslice ; i++) {
      clv[f].pg[i] = clCreateBuffer(clv[f].ctx, CL_MEM_READ_ONLY |
CL_MEM_HOST_WRITE_ONLY,
            F.mt * size_C, NULL, &err);
      clv[f].pr[i] = clCreateBuffer(clv[f].ctx, CL_MEM_READ_ONLY |
CL_MEM_HOST_WRITE_ONLY,
            F.mt * MSP.ntilt[f] * size_C, NULL, &err);
   }
   clv[f].pmap = clCreateBuffer(clv[f].ctx, CL_MEM_READ_WRITE |
CL_MEM_HOST_WRITE_ONLY,
         F.mt * MSP.ntilt[f] * size_C, NULL, &err);
   clv[f].ptmp = clCreateBuffer(clv[f].ctx, CL_MEM_READ_WRITE |
CL_MEM_HOST_NO_ACCESS,
         F.mt * MSP.ntilt[f] * size_C, NULL, &err);
   clv[f].beam = clCreateBuffer(clv[f].ctx, CL_MEM_READ_WRITE |
CL_MEM_HOST_READ_ONLY,
         F.mt * MSP.ntilt[f] * size_C, NULL, &err);
   clv[f].ib = clCreateBuffer(clv[f].ctx,
         CL_MEM_READ_ONLY | CL_MEM_HOST_WRITE_ONLY,//|
CL_MEM_COPY_HOST_PTR | CL_MEM_HOST_NO_ACCESS,
         F.mbout * sizeof(int), NULL, &err);
   // CL_MEM_COPY_HOST_PTR appears to cause memory leak on Nvidia
   err = clEnqueueWriteBuffer(clv[f].q, clv[f].ib, CL_FALSE,
         0, F.mbout * sizeof(int), F.ib, 0, NULL, NULL);

/* Initialise clFFT */
   clfftSetupData fftSetup;
   err = clfftInitSetupData(&fftSetup);
   err = clfftSetup(&fftSetup);

/* Plan clFFT backward transform */
   size_t clfftLengths[2] = { (size_t) F.meshx, (size_t) F.meshy };
   err = clfftCreateDefaultPlan(&clv[f].plan_b, clv[f].ctx, CLFFT_2D, clfftLengths);
   err = clfftSetPlanBatchSize(clv[f].plan_b, MSP.ntilt[f]);
   err = clfftSetPlanDistance(clv[f].plan_b, F.mt, F.mt);
   err = clfftSetPlanPrecision(clv[f].plan_b, MSP.sgl ? CLFFT_SINGLE : CLFFT_DOUBLE);
   err = clfftSetLayout(clv[f].plan_b, CLFFT_COMPLEX_INTERLEAVED,
CLFFT_COMPLEX_INTERLEAVED);
   err = clfftSetResultLocation(clv[f].plan_b, CLFFT_INPLACE);
   err = clfftBakePlan(clv[f].plan_b, 1, &clv[f].q, NULL, NULL);

/* Set precision & global constants for kernels */
   string p1 = MSP.sgl ? "float" : "double";
   string p2 = MSP.sgl ? "float2" : "double2";
   string p4 = MSP.sgl ? "float4" : "double4";
   string mx = to_string(F.meshx / 2), my = to_string(F.meshy / 2), mthf = to_string(F.mt / 2),
         meshx = to_string(F.meshx), meshy = to_string(F.meshy), mt = to_string(F.mt),
         ntilt = to_string(MSP.ntilt[f]);

/* Setup clFFT callbacks for forward transform */
   string preXpg_str = ""
""+p2+" preXpg(__global "+p2+"* pmap, uint offset,                    \n"
"      const __global "+p2+"* pg)                                    \n"
"{                                                                   \n"
"  "+p2+" pgi = *(pg + offset%"+mt+"), pmapi = *(pmap+offset);       \n"
//"   if(offset<5) printf(\"pg %i %g %g\\n\",offset,pgi.x,pgi.y);     \n"
//"   if(offset<5) printf(\"pmap %i %g %g\\n\",offset,pmapi.x,pmapi.y); \n"
"  return ("+p2+") (pmapi.x * pgi.x - pmapi.y * pgi.y,              \n"
"                   pmapi.x * pgi.y + pmapi.y * pgi.x);             \n"
"}\n";
   string postXpr_str = ""
"void postXpr(__global "+p2+"* pmap, uint offset,                    \n"
"      const __global "+p2+"* pr, "+p2+" fftO)                       \n"
"{                                                                   \n"
"  "+p2+" pri = *(pr + offset);                                      \n"
//"   if(offset<5) printf(\"pr %i %g %g\\n\",offset,pri.x,pri.y);     \n"
"  *(pmap + offset) = ("+p2+") (fftO.x * pri.x - fftO.y * pri.y,     \n"
"                       fftO.x * pri.y + fftO.y * pri.x);           \n"
"}";

/* Plan clFFT forward transform */
   clv[f].plans_f = new clfftPlanHandle[F.nslice];
   for (int i = 0 ; i < F.nslice ; i++) {
      err = clfftCopyPlan(&clv[f].plans_f[i], clv[f].ctx, clv[f].plan_b);
      err = clfftSetPlanCallback(clv[f].plans_f[i], "preXpg", preXpg_str.c_str(),
```

Appendix B. Program scripts

```
            0, PRECALLBACK, &clv[f].pg[i], 1);
        err = clfftSetPlanCallback(clv[f].plans_f[i], "postXpr", postXpr_str.c_str(),
            0, POSTCALLBACK, &clv[f].pr[i], 1);
        err = clfftBakePlan(clv[f].plans_f[i], 1, &clv[f].q, NULL, NULL);
    }

/* Setup other kernels */
    string program_str = ""
"__kernel void propg(const "+p4+" xyz, const "+p4+" c2r,                    \n"
"      const __global "+p2+" *tiltarray, __global "+p2+" *pr)               \n"
"{                                                                          \n"
"  const int x = get_global_id(1), y = get_global_id(2), t = get_group_id(0);       \n"
"  const int h = x - "+mx+", k = y -  "+my+";                               \n"
"  const "+p2+" tilt = tiltarray[t];                                        \n"
"  "+p1+" a1, b1, sg, sn, cs;                                               \n"
"  a1 = tilt.x + h, b1 = tilt.y + k;                                        \n"
"  sg = a1 * h * c2r.x + b1 * k * c2r.y + (a1 * k + b1 * h) * c2r.z;        \n"
"  sn = sincos(6.28318530717958647 * (sg * xyz.z + h * xyz.x + k * xyz.y), &cs);  \n"
"  pr[x + y *  "+meshx+" + t *  "+mt+"] = ("+p2+") {cs, sn};                \n"
//"    if (x==0&&y==0)                                                       \n"
//"        printf(\"%d %g %g\\n\",t,tilt.x,tilt.y);                          \n"
//"    if (t == 0 && x==0&&y==0)                                             \n"
//"        printf(\"%g %g %g %g %g %g sizeof(U)=%i\\n\",                     \n"
//"              xyz.x,xyz.y,xyz.z,c2r.x,c2r.y,c2r.z,sizeof(xyz));          \n"
//"    if (t == 1 && x!=0&&y!=0)                                             \n"
//"        printf(\"%i.%i %i %i %i %g %g\\n\", x,y, h,k,x+y*"+meshx+",cs,sn);       \n"
"}\n"

"__kernel void pad0oshift(__global "+p2+" *pr)                              \n"
"{                                                                          \n"
"  int tmt = get_global_id(0);                                              \n"
"  if (tmt < "+ntilt+") {                                                   \n"
"      tmt *=  "+mt+";                                                      \n"
// PAD0
"      "+p2+" m = {0, 0};                                                   \n"
"      for (int i = tmt ; i <  "+meshx+" + tmt ; ++i)                       \n"
"          pr[i] = m;                                                       \n"
"      for (int j = "+meshx+" + tmt ; j < "+mt+" + tmt ; j += "+meshx+")    \n"
"          pr[j] = m;                                                       \n"
// OSHIFT
"      for (int j = tmt ; j < "+mthf+" - "+meshx+" + tmt + 1 ; j += "+meshx+")      \n"
"          for (int i = j, i1 = i +  "+mx+" + "+mthf+" ; i < "+mx+" + j ; ++i, ++i1)   \n"
"              m = pr[i], pr[i] = pr[i1], pr[i1] = m;                       \n"
"      for (int j = "+mthf+" + tmt ; j < "+mt+" - "+meshx+" + tmt + 1 ; j += "+meshx+")\n"
"          for (int i = j, i1 = i + "+mx+" - "+mthf+" ; i < "+mx+" + j ; ++i, ++i1) \n"
"              m = pr[i], pr[i] = pr[i1], pr[i1] = m;                       \n"
//"      if(tmt/"+mt+"==0)                                                  \n"
//"          for (int i = 0 ; i < "+mt+" ; ++i)                             \n"
//"              printf(\"%i-%i. %g %g\\n\",tmt/"+mt+",i,pr[tmt+i].x,pr[tmt+i].y);       \n"
"  }                                                                        \n"
"}\n"

"__kernel void fill_pmap(__global "+p2+"* pmap)             \n"
"{                                                          \n"
"  int i = get_global_id(0);                                \n"
"  pmap[i].x = "+to_string(1.0/F.mt)+", pmap[i].y = 0.;  \n"
"}\n"

"__kernel void beam_out(__global "+p1+"* beam, const __global "+p2+"* pmap,    \n"
"      const __global int* ib)                              \n"
"{                                                          \n"
"  int i = get_global_id(0), j = get_group_id(1);           \n"
"  if (i < "+ntilt+") {                                     \n"
"      "+p2+" m = pmap[i * "+mt+" + ib[j] - 1];             \n"
"      beam[i + j * "+ntilt+"] = ("+p1+") m.x * m.x + m.y * m.y;            \n"
//"      if(j==0) printf(\"beam: %d %d %g\\n\",i,last,beam[i + j * "+ntilt+"]);  \n"
"  }\n"
"}\n";
    const char* program_cstr = program_str.c_str();

    clv[f].program = clCreateProgramWithSource(clv[f].ctx, 1, &program_cstr, NULL, &err);
    err = clBuildProgram(clv[f].program, 1, &dev_id, NULL, NULL, NULL);
    cl_build_status build_status;
    clGetProgramBuildInfo(clv[f].program, dev_id, CL_PROGRAM_BUILD_STATUS,
sizeof(cl_build_status), &build_status, NULL);
    if (build_status != CL_BUILD_SUCCESS) {
        size_t build_log_size;
        clGetProgramBuildInfo(clv[f].program, dev_id, CL_PROGRAM_BUILD_LOG, 0, NULL,
&build_log_size);
        char* build_log = (char*) malloc(build_log_size);
        clGetProgramBuildInfo(clv[f].program, dev_id, CL_PROGRAM_BUILD_LOG,
build_log_size, build_log, NULL);
        printf("%s\n", build_log);
        free(build_log);
    }

    clv[f].propg = clCreateKernel(clv[f].program, "propg", &err);

    clv[f].pad0oshift = clCreateKernel(clv[f].program, "pad0oshift", &err);

    clv[f].fill_pmap = clCreateKernel(clv[f].program, "fill_pmap", &err);
    err = clSetKernelArg(clv[f].fill_pmap, 0, sizeof(cl_mem), &clv[f].pmap);

    clv[f].beam_out = clCreateKernel(clv[f].program, "beam_out", &err);
    err = clSetKernelArg(clv[f].beam_out, 0, sizeof(cl_mem), &clv[f].beam);
    err = clSetKernelArg(clv[f].beam_out, 1, sizeof(cl_mem), &clv[f].pmap);
    err = clSetKernelArg(clv[f].beam_out, 2, sizeof(cl_mem), &clv[f].ib);

/* Setup GPU multi-thread job distribution */
    size_t maxWorkSize[3], maxWorkGroup;
    clGetDeviceInfo(dev_id, CL_DEVICE_MAX_WORK_ITEM_SIZES,
        sizeof(maxWorkSize), maxWorkSize, NULL);
    clGetDeviceInfo(dev_id, CL_DEVICE_MAX_WORK_GROUP_SIZE,
        sizeof(maxWorkGroup), &maxWorkGroup, NULL);

    clv[f].L_propg[1] = F.meshx, clv[f].L_propg[2] = F.meshy;
    while (clv[f].L_propg[2] > min(maxWorkSize[2], maxWorkGroup))
      clv[f].L_propg[2] /= 2;
    while (clv[f].L_propg[1] * clv[f].L_propg[2] > maxWorkGroup)
      clv[f].L_propg[1] /= 2;
    clv[f].G_propg[0] = MSP.ntilt[f], clv[f].G_propg[1] = F.meshx, clv[f].G_propg[2] = F.meshy;

    clv[f].L_pad = MSP.ntilt[f];
    if (clv[f].L_pad > min(maxWorkGroup, maxWorkSize[0]))
      clv[f].L_pad = min(maxWorkGroup, maxWorkSize[0]);
    clv[f].G_pad = ceil(1.0 * MSP.ntilt[f] / clv[f].L_pad) * clv[f].L_pad;

    clv[f].L_pmap = F.mt, clv[f].G_pmap = F.mt * MSP.ntilt[f];
    while (clv[f].L_pmap > min(maxWorkSize[2], maxWorkGroup))
      clv[f].L_pmap /= 2;

    clv[f].L_beam_out[0] = MSP.ntilt[f];
    if (clv[f].L_beam_out[0] > maxWorkGroup)
      clv[f].L_beam_out[0] = maxWorkGroup;
    clv[f].G_beam_out[0] = ceil(1.0 * MSP.ntilt[f] / clv[f].L_beam_out[0]) * clv[f].L_beam_out[0];
    clv[f].G_beam_out[1] = F.mbout;

    fortran_write(13, "  Device's max total size of a work group = %d & per dimension =
(%d, %d, %d)\n",
        maxWorkGroup, maxWorkSize[0], maxWorkSize[1], maxWorkSize[2]);
    fortran_write(13, "  Kernels' work group sizes will be:\n");
    fortran_write(13, "    propg      - local = (%d, %d, %d),  global = (%d, %d, %d);\n",
        clv[f].L_propg[0], clv[f].L_propg[1], clv[f].L_propg[2],
        clv[f].G_propg[0], clv[f].G_propg[1], clv[f].G_propg[2]);
    fortran_write(13, "    pad0oshift - local = (%d),  global = (%d);\n", clv[f].L_pad, clv[f].G_pad);
    fortran_write(13, "    pmap       - local = (%d),  global = (%d);\n", clv[f].L_pmap, clv[f].G_pmap);
    fortran_write(13, "    beam_out   - local = (%d, %d),  global = (%d, %d).\n",
        clv[f].L_beam_out[0], clv[f].L_beam_out[1], clv[f].G_beam_out[0], clv[f].G_beam_out[1]);
}

void clfft_child_term(const int f)
{
    if(MSP.sgl) delete[] clv[f].host_pgf;
    for (int i = 0 ; i < F.nslice ; i++) {
      clReleaseMemObject(clv[f].pg[i]);
      clReleaseMemObject(clv[f].pr[i]);
      clfftDestroyPlan(&clv[f].plans_f[i]);
    }
    delete[] clv[f].pg;
    delete[] clv[f].pr;
    delete[] clv[f].plans_f;
    clfftDestroyPlan(&clv[f].plan_b);
    clfftTeardown();
    clReleaseMemObject(clv[f].ptmp);
    clReleaseMemObject(clv[f].pmap);
    clReleaseMemObject(clv[f].tiltarray);
    clReleaseMemObject(clv[f].beam);
    clReleaseMemObject(clv[f].ib);
    clReleaseKernel(clv[f].propg);
    clReleaseKernel(clv[f].pad0oshift);
    clReleaseKernel(clv[f].fill_pmap);
    clReleaseKernel(clv[f].beam_out);
    clReleaseProgram(clv[f].program);
    clReleaseCommandQueue(clv[f].q);
    clReleaseContext(clv[f].ctx);
}
}

template<typename T, typename U>
void clfft_msp(const int f, const int ct1, const int ct2, complex<T>* host_pg)
{
    time_point t_msp;
    for (int i = 0 ; i < F.nslice ; i++)
      cl_dbg("Write pg", clEnqueueWriteBuffer(clv[f].q, clv[f].pg[i], CL_FALSE,
          0, F.mt * size_C, host_pg + i * F.mt, 0, NULL, NULL));

    for (int isplt = 0 ; isplt < MSP.splt[f] ; ++isplt) {
      int tct1 = isplt * MSP.ntilt[f] + ct1;
      size_t copy_size = min(MSP.ntilt[f], ct2 - tct1 + 1) * size_R;
//    printf("%d %d %d %lu\n",tct1,ct2,ct2 - tct1 + 1,copy_size/size_R);

      if (Tr.msp) {
        clFinish(clv[f].q);
        t_msp = time_now();
      }
      for (int i = 0 ; i < F.nslice ; i++) {
        cl_dbg("Write tiltarray", clEnqueueWriteBuffer(clv[f].q, clv[f].tiltarray, CL_FALSE,
            0, 2 * copy_size, ((T*) F.tiltarray) + 2 * (tct1 + i * F.tottilts), 0, NULL, NULL));

        U xyz = { (T) F.xshift[i], (T) F.yshift[i], (T) (F.deltaz[i] * F.appliedditalion[i]) };
        U c2r = { (T) F.cell2rpr[3 * i], (T) F.cell2rpr[3 * i + 1], (T) F.cell2rpr[3 * i + 2] };
//      printf("%g %g %g %g %g %g\n", xyz.s[0],xyz.s[1],xyz.s[2],c2r.s[0],c2r.s[1],c2r.s[2]);
        cl_dbg("propg set xyz", clSetKernelArg(clv[f].propg, 0, sizeof(U), &xyz));
        cl_dbg("propg set c2r", clSetKernelArg(clv[f].propg, 1, sizeof(U), &c2r));
        cl_dbg("propg set tilt", clSetKernelArg(clv[f].propg, 2, sizeof(cl_mem), &clv[f].tiltarray));
        cl_dbg("propg set pr", clSetKernelArg(clv[f].propg, 3, sizeof(cl_mem), &clv[f].pr[i]));
        cl_dbg("propg enqueue", clEnqueueNDRangeKernel(clv[f].q, clv[f].propg, 3, 0,
            clv[f].G_propg, clv[f].L_propg, 0, NULL, NULL));

        cl_dbg("pad0oshift set pr", clSetKernelArg(clv[f].pad0oshift, 0, sizeof(cl_mem),
&clv[f].pr[i]));
        cl_dbg("pad0oshift enqueue", clEnqueueNDRangeKernel(clv[f].q, clv[f].pad0oshift, 1, 0,
            &clv[f].G_pad, &clv[f].L_pad, 0, NULL, NULL));
      }
      if (Tr.msp) {
        clFinish(clv[f].q);
        Tr.pr[f] += timer(&t_msp);
      }
```

Appendix B. Program scripts

```
      // clEnqueueFillBuffer is buggy on Mac; a kernel is required to fill clv[f].pmap
      // complex<T> cmplx1 = { (T) 1.0 / F.mt, 0 };
      // clEnqueueFillBuffer(clv[f].q, clv[f].pmap, &cmplx1,
      //       sizeof(cmplx1), 0, Size.pmap, 0, NULL, NULL);
      clEnqueueNDRangeKernel(clv[f].q, clv[f].fill_pmap, 1, 0,
            &clv[f].G_pmap, &clv[f].L_pmap, 0, NULL, NULL);

      if (Tr.msp) {
         clFinish(clv[f].q);
         t_msp = time_now();
      }

      for (int i = 0 ; i < F.sl_hi ; i++) {
         cl_dbg("clfftEnqueueTransform FFT", clfftEnqueueTransform(clv[f].plans_f[F.mseq[i] - 1],
               CLFFT_FORWARD, 1, &clv[f].q, 0, NULL, NULL, &clv[f].pmap, NULL,
clv[f].ptmp));
         if (Tr.msp) {
            clFinish(clv[f].q);
            Tr.fft[f] += timer(&t_msp);
         }

         if ((F.sl_opt && i >= F.sl_lo - 1) || i == F.sl_hi - 1) {
            cl_dbg("clEnqueueNDRangeKernel beam_out", clEnqueueNDRangeKernel(clv[f].q,
                  clv[f].beam_out, 2, 0, clv[f].G_beam_out, clv[f].L_beam_out, 0, NULL, NULL));

            double* host_beam = F.sl_opt ? F.sl_beam + (i + 1 - F.sl_lo) * MSP.ntilt[f] * F.mbout :
F.beam;

            for (int j = 0 ; j < F.mbout ; j++)
               cl_dbg("clEnqueueReadBuffer beam", clEnqueueReadBuffer(clv[f].q, clv[f].beam,
                     CL_FALSE, j * MSP.ntilt[f] * size_R, copy_size,
                     host_beam + tct1 + j * F.tottilts, 0, NULL, NULL));

            if (i == F.sl_hi - 1) break;
         }

         if (Tr.msp) {
            clFinish(clv[f].q);
            t_msp = time_now();
         }
         cl_dbg("clfftEnqueueTransform IFFT", clfftEnqueueTransform(clv[f].plan_b,
               CLFFT_BACKWARD, 1, &clv[f].q, 0, NULL, NULL, &clv[f].pmap, NULL,
clv[f].ptmp));
         if (Tr.msp) {
            clFinish(clv[f].q);
            Tr.ifft[f] += timer(&t_msp);
         }
      }
   }

   clFinish(clv[f].q);
}

extern "C"
{
void clfft_msp_cast(const int f, const int ct1, const int ct2)
{
   if (MSP.sgl){
      for (auto i = 0 ; i < F.mt * F.nslice ; ++i)
         clv[f].host_pgf[i] = (complex<float> ) F.pg[i];

      clfft_msp<float,cl_float4>(f, ct1, ct2, clv[f].host_pgf);
   }
   else
      clfft_msp<double,cl_double4>(f, ct1, ct2, F.pg);
}
}
```

B.1.4. Fresnel propagation using cuFFT

```
/*
 * CUDA functions of QCBEDMS-PF
 */
#include <cufft.h>
//#include <cuda_profiler_api.h>
#include "cpp_comm.hpp"

using namespace std;
typedef cuDoubleComplex CD;
typedef cuFloatComplex CF;

__constant__ int mx, my, mthf, meshx, meshy, mt, nslice, mbout, ntilt;

template<typename T>
__host__ __device__ static __inline__ T CmplxMul(T a, T b)
{
   T r = { a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x };
   return r;
}

bool debug = false;
void cu_dbg(string msg, cudaError_t err)
{
   if(debug) printf("%s: %s\n", msg.c_str(), cudaGetErrorString(err));
}

struct cufft_vars {
   CD *pmap, *pg, *pr;
   double2 *tiltarray;
   double *beam;
   int *ib;
   cufftHandle plan;
   dim3 g_propg, b_propg, g_pmap, b_pmap, g_beam, b_beam;
   size_t g_pad, b_pad;
} *cuv;

extern "C"
{
void cufft_init()
{
   cuv = new cufft_vars[MSP.nhyb];
}

void cufft_term()
{
   delete[] cuv;
}

void CUDAGetDevices(int in, int *count)
{
   cudaGetDeviceCount(count);
   if (in == -2)
      return;

   printf("Found %i CUDA device(s):\n", *count);
   cudaDeviceProp prop;
   for (int i = 0 ; i < *count ; i++) {
      cudaGetDeviceProperties(&prop, i);
      printf("\t%i. %s, compute capability %d.%d\n",
            i, prop.name, prop.major, prop.minor);
   }
   exit(0);
}

void cufft_child_init(const int f)
{
/* Choose CUDA device */
   cudaDeviceProp prop;
   cudaSetDevice(MSP.dev[f]);
   cudaGetDeviceProperties(&prop, MSP.dev[f]);
   cu_dbg("set dev", cudaSetDeviceFlags(cudaDeviceScheduleYield));
   fortran_write(13, "  Device: %d. %s.\n", MSP.dev[f], prop.name);

/* Measure cuFFT working memory and optional auto-determine beam pixels split */
   fortran_write(13, "  cuFFT precision: %s.\n", MSP.sgl ? "single" : "double");
   float MB = 1.0 / 1024 / 1024;
   size_t free_gm, total_gm;
   cudaError_t mem_err = cudaMemGetInfo(&free_gm, &total_gm);
   if (MSP.splt[f] == 0) {
      MSP.splt[f] = 1, MSP.ntilt[f] = MSP.as_nt[f];
      while (msp_mem(f, 0) > free_gm && MSP.splt[f] < MSP.as_nt[f])
         MSP.ntilt[f] = (int) ceil((float) MSP.as_nt[f] / ++MSP.splt[f]);
      fortran_write(13, "(AUTO) ");
   }
   else
      MSP.ntilt[f] = (int) ceil((float) MSP.as_nt[f] / MSP.splt[f]);

   fortran_write(13, "  Paralleling %d/%d tilts; will loop %d time(s).\n",
         MSP.ntilt[f], MSP.as_nt[f], MSP.splt[f]);
   fortran_write(13, "  Available GPU memory: %g/%g MB, cuFFT will use at least %g MB.\n",
         free_gm * MB, total_gm * MB, msp_mem(f, 0) * MB);

   if (msp_mem(f, 0) > free_gm) {
      if (F.ncalcs < 1) {
         fortran_write(13, "\nNOT enough GPU memory! "
               "Please reduce paralleled tilts manually.\n");
         cout << "NOT enough GPU memory! Please reduce paralleled tilts manually." << endl;
         exit(-1);
      } else if (mem_err != cudaSuccess) {
         fortran_write(13, "\nWARNING: \"CUDA get memory usage\" failed "
               "during refinement! Ignored and continue.\n");
         fortran_write(13, "%s\n", cudaGetErrorString(mem_err));
      }
   }
   if (msp_mem(f, 0) > free_gm * 0.5)
      fortran_write(13, "  GPU memory may be intensive; reduce paralleled "
            "tilts manually if any error occurs.\n");

/* Set global constants (even before choosing devices) */
   int mxt = F.meshx / 2, myt = F.meshy / 2, mthft = F.mt / 2;
   cudaMemcpyToSymbolAsync(mx, &mxt, sizeof(int));
   cudaMemcpyToSymbolAsync(my, &myt, sizeof(int));
   cudaMemcpyToSymbolAsync(mthf, &mthft, sizeof(int));
   cudaMemcpyToSymbolAsync(meshx, &F.meshx, sizeof(int));
   cudaMemcpyToSymbolAsync(meshy, &F.meshy, sizeof(int));
   cudaMemcpyToSymbolAsync(mt, &F.mt, sizeof(int));
   cudaMemcpyToSymbolAsync(nslice, &F.nslice, sizeof(int));
   cudaMemcpyToSymbolAsync(mbout, &F.mbout, sizeof(int));
   cudaMemcpyToSymbolAsync(ntilt, &MSP.ntilt[f], sizeof(int));

/* Allocate GPU arrays */
   cudaMalloc((void **) &cuv[f].pg, F.mt * F.nslice * size_C);
   cudaMalloc((void **) &cuv[f].pr, F.mt * MSP.ntilt[f] * F.nslice * size_C);
   cudaMalloc((void **) &cuv[f].pmap, F.mt * MSP.ntilt[f] * size_C);
   cudaMalloc((void **) &cuv[f].tiltarray, 2 * MSP.ntilt[f] * size_R);
   cudaMalloc((void **) &cuv[f].beam, MSP.ntilt[f] * F.mbout * size_R);
   cudaMalloc((void **) &cuv[f].ib, F.mbout * sizeof(int));
   cudaMemcpyAsync(cuv[f].ib, F.ib, F.mbout * sizeof(int), cudaMemcpyHostToDevice, 0);
   cudaMemsetAsync(cuv[f].beam, 0, MSP.ntilt[f] * F.mbout * size_R);

/* Setup batched 2D cuFFT plan */
   int n[2] = { (int) F.meshy, (int) F.meshx };
   cufftPlanMany(&cuv[f].plan, 2, n, NULL, 1, F.mt, NULL, 1, F.mt,
```

Appendix B. Program scripts

```
        MSP.sgl ? CUFFT_C2C : CUFFT_Z2Z, MSP.ntilt[f]);

    int maxThreads, warp_size, SMs, attr[100];
    for (size_t i = 1 ; i < 92 ; i++)
        cudaDeviceGetAttribute(&attr[i], (cudaDeviceAttr) i, 0);
    maxThreads = attr[1];
    warp_size = attr[10];
    SMs = attr[16];

/* Kernel cu_propg GPU threads mapping
 1. 0 <= threadIdx.x + blockIdx.y * blockDim.x < meshx,
    0 <= threadIdx.y + blockIdx.z * blockDim.y < meshy;
 2. 0 <= blockIdx.x < MSP.ntilt[f], as gridDim.x can hold the most blocks;
 3. Repeat kernel for F.nslice.
 */
    cuv[f].b_propg = cuv[f].g_propg = 1;
    cuv[f].b_propg.x = F.meshx, cuv[f].b_propg.y = F.meshy;
    int i = 1;
    while ((cuv[f].b_propg.x * cuv[f].b_propg.y % warp_size != 0 &&
            cuv[f].b_propg.x * cuv[f].b_propg.y > warp_size) ||
            cuv[f].b_propg.x * cuv[f].b_propg.y > maxThreads) {
        if (i == 0)
            i++, cuv[f].b_propg.x /= 2, cuv[f].g_propg.y *= 2;
        else
            i--, cuv[f].b_propg.y /= 2, cuv[f].g_propg.z *= 2;
    }
    cuv[f].g_propg.x = MSP.ntilt[f];

/* Kernel pad0oshift GPU threads mapping
 1. 0 <= threadIdx.x + blockIdx.x * blockDim.x < MSP.ntilt[f];
 2. Repect kernel for F.nslice.
*/
    cuv[f].b_pad = ceil(1. * MSP.ntilt[f] / SMs);
    if (cuv[f].b_pad > maxThreads)
        cuv[f].b_pad = maxThreads;
    cuv[f].g_pad = ceil(1. * MSP.ntilt[f] / cuv[f].b_pad);

/* Kernels pmapXpg & pmapXpg GPU threads mapping
 1. 'cuv[f].b_pmap.x' holds 'F.mt' as threads per block,
 'cuv[f].g_pmap.x' holds 'MSP.ntilt[f]' as number of blocks;
 2. If 'cuv[f].b_pmap.x' > max allowed threads number 'maxThreads',
 keep halving it and the excess goes to grid 'cuv[f].g_pmap.y';
 3. Finally, try to make total-thread-number divisible by warp 'warp_size'.

 So threads iterate upon 'cuv[f].b_pmap.x' only, while blocks iterate through
 'cuv[f].g_pmap.x' * 'cuv[f].g_pmap.y'.

 Criteria:
 1. 0 <= threadIdx.x * gridDim.y + blockIdx.y < cuv[f].b_pmap.x * cuv[f].g_pmap.y = F.mt,
 they control mesh iteration of every pixel;
 2. 0 <= blockIdx.x < cuv[f].g_pmap.x = MSP.ntilt[f], for pixel iterations.
 */

    cuv[f].b_pmap = cuv[f].g_pmap = 1;
    cuv[f].b_pmap.x = F.mt, cuv[f].g_pmap.x = MSP.ntilt[f];
    while ((cuv[f].b_pmap.x % warp_size != 0 && cuv[f].b_pmap.x > warp_size) ||
            cuv[f].b_pmap.x > maxThreads)
        cuv[f].b_pmap.x /= 2, cuv[f].g_pmap.y *= 2;

/* Kernel calc_beam GPU threads mapping
 Total paralleled jobs = (num of beams 'F.mbout') * (paralleled pixels 'MSP.ntilt[f]').

 1. Primarily, divide 'MSP.ntilt[f]' iterations into blocks 'cuv[f].g_beam.x' same to number
 of multiprocessors, and put the threads in 'cuv[f].b_beam.x', but
 'cuv[f].b_beam.x' * 'cuv[f].g_beam.x' >= 'MSP.ntilt[f]';
 2. Map beam iterations to 'cuv[f].g_beam.y';
 3. If 'cuv[f].b_beam.x' > 'maxThreads', then maximise 'cuv[f].b_beam.x' to 'maxThreads',
 and map the excess jobs back to grid 'cuv[f].g_beam.x'

 So threads iterate upon 'cuv[f].b_beam.x' only, while blocks iterate through
 'cuv[f].g_beam.x' * 'cuv[f].g_beam.y'.

 Criteria:
 1. cuv[f].b_beam.x * cuv[f].g_beam.x = MSP.ntilt[f], they control 'MSP.ntilt[f]' iteration of each
beam
 (threadIdx.x * gridDim.y + blockIdx.y);
 2. cuv[f].g_beam.y = blockIdx.x = F.mbout, for beam iterations.
 */

    cuv[f].b_beam = cuv[f].g_beam = 1;
    cuv[f].b_beam.x = ceil(1. * MSP.ntilt[f] / SMs);
    if (cuv[f].b_beam.x > maxThreads)
        cuv[f].b_beam.x = maxThreads;
    cuv[f].g_beam.x = ceil(1. * MSP.ntilt[f] / cuv[f].b_beam.x);
    cuv[f].g_beam.y = F.mbout;

//  printf("pmap block = %i %i %i, grid = %i %i %i\n",
//         cuv[f].b_pmap.x, cuv[f].b_pmap.y, cuv[f].b_pmap.z,
//         cuv[f].g_pmap.x, cuv[f].g_pmap.y, cuv[f].g_pmap.z);
//  printf("beam block = %i %i %i, grid = %i %i %i\n",
//         cuv[f].b_beam.x, cuv[f].b_beam.y, cuv[f].b_beam.z,
//         cuv[f].g_beam.x, cuv[f].g_beam.y, cuv[f].g_beam.z);

//  cudaDeviceSynchronize();
//  printf("%s\n", cudaGetErrorString(cudaGetLastError()));
}

void cufft_child_term(const int f)
{
    cudaSetDevice(MSP.dev[f]);
    cudaFree(cuv[f].pg);
    cudaFree(cuv[f].pr);
    cudaFree(cuv[f].pmap);
    cudaFree(cuv[f].beam);
    cudaFree(cuv[f].ib);
    cudaFree(cuv[f].tiltarray);
    cufftDestroy(cuv[f].plan);
    cudaDeviceReset();
}
}

// CUDA kernels:
__global__ void propg(const double3 xyz, const double3 c2r, const double2 *tiltarray, CD *pr)
{
    int x = threadIdx.x + blockIdx.y * blockDim.x;
    int y = threadIdx.y + blockIdx.z * blockDim.y;
    int h = x - mx, k = y - my, t = blockIdx.x;
    double2 tilt = tiltarray[t];
    double a1, b1, sg, sn, cs;
    a1 = tilt.x + h, b1 = tilt.y + k;
    sg = a1 * h * c2r.x + b1 * k * c2r.y + (a1 * k + b1 * h) * c2r.z;
    sincospi(2 * (sg * xyz.z + h * xyz.x + k * xyz.y), &sn, &cs);
    pr[x + y * meshx + t * mt] = {cs / mt, sn / mt};
// if (t == 3 && x!=0&&y!=0)
//     printf("pr: %d %g %g\n", t, pr[x + y * meshx + t * mt].x,
//            pr[x + y * meshx + t * mt].y);
// if (t == 3 && x!=0&&y!=0)
//     printf("%i.%i %i %i %i %g %g\n", x,y, h,k,x+y*meshx,pr[x + y * meshx + t * mt].x,
//            pr[x + y * meshx + t * mt].y);
// if (x == 0 && y == 0)
//     printf("%i %g %g\n", t,tilt.x,tilt.y);
}

template<typename T>
__global__ void pad0oshift(T *pr)
{
    int tmt = threadIdx.x + blockIdx.x * blockDim.x;
    if (tmt < ntilt) {
        tmt *= mt;
// PAD0
        for (int i = tmt ; i < meshx + tmt ; ++i)
            pr[i] = {0., 0.};
        for (int j = meshx + tmt ; j < mt + tmt ; j += meshx)
            pr[j] = {0., 0.};
// OSHIFT
        T tmp;
        for (int j = tmt ; j < mthf - meshx + tmt + 1 ; j += meshx)
            for (int i = j, i1 = i + mx + mthf ; i < mx + j ; ++i, ++i1)
                tmp = pr[i], pr[i] = pr[i1], pr[i1] = tmp;
        for (int j = mthf + tmt ; j < mt - meshx + tmt + 1 ; j += meshx)
            for (int i = j, i1 = i + mx - mthf ; i < mx + j ; ++i, ++i1)
                tmp = pr[i], pr[i] = pr[i1], pr[i1] = tmp;
//      if(tmt/mt==0)
//      for (int i = 0 ; i < mt ; ++i)
//          printf("%i-%i. %g %g\n",tmt/mt,i,pr[tmt+i].x*mt,pr[tmt+i].y*mt);
    }
}

template<typename T>
__global__ void pmapCmplx1(T* __restrict__ pmap)
{
    int i = threadIdx.x + (blockIdx.y + blockIdx.x * gridDim.y) * blockDim.x;
    pmap[i].x = 1., pmap[i].y = 0.;
}

template<typename T>
__global__ void pmapXpg(T* __restrict__ pmap, const T* __restrict__ pg)
{
    int i = threadIdx.x + blockIdx.y * blockDim.x;
    int j = blockIdx.x * mt + i;
// if(j<5) printf("pg %i %g %g\n",j,pg[i].x,pg[i].y);
    T a = pmap[j], b = pg[i];
    pmap[j] = CmplxMul(a, b);
// if(j<5) printf("pmap %i %g %g\n",j,pmap[j].x,pmap[j].y);
}

template<typename T>
__global__ void pmapXpr(T* __restrict__ pmap, const T* __restrict__ pr)
{
    int i = threadIdx.x + (blockIdx.y + blockIdx.x * gridDim.y) * blockDim.x;
// printf("pr %i %g %g\n",i,pr[i].x,pr[i].y);
    T a = pmap[i], b = pr[i];
    pmap[i] = CmplxMul(a, b);
}

template<typename T, typename U>
__global__ void calc_beam(U* __restrict__ beam, T* __restrict__ pmap, int* __restrict__ ib)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x, j = blockIdx.y;
    if (i < ntilt) {
        const T m = pmap[i * mt + ib[j] - 1];
        beam[i + j * ntilt] = (U) m.x * m.x + m.y * m.y;
//      if(j==0) printf("beam: %d %d %d %g\n",i,tct1,tct2,beam[i + j * ntilt]);
    }
}

extern "C" void cufft_msp(const int f, const int ct1, const int ct2)
{
    time_point t_msp;
    cudaSetDevice(MSP.dev[f]);

    for (int isplt = 0 ; isplt < MSP.splt[f] ; ++isplt) {
        int tct1 = isplt * MSP.ntilt[f] + ct1;
        int copy_size = min(MSP.ntilt[f], ct2 - tct1 + 1) * size_R;
//      printf("%d %d %d %d %d %lu\n",ct1,ct2);

        cudaMemcpyAsync(cuv[f].pg, F.pg, F.mt * F.nslice * size_C, cudaMemcpyHostToDevice, 0);
```

Appendix B. Program scripts

```
// cudaProfilerStart();
      if (Tr.msp) {
         cudaDeviceSynchronize();
         t_msp = time_now();
      }
      for (int i = 0 ; i < F.nslice ; ++i) {
         cudaMemcpyAsync(cuv[f].tiltarray,
               F.tiltarray + 2 * (tct1 + i * F.tottilts),
               2 * copy_size, cudaMemcpyHostToDevice, 0);
         propg<<<cuv[f].g_propg, cuv[f].b_propg>>>(
               {F.xshift[i], F.yshift[i], F.deltaz[i] * F.applieddilation[i]},
               ((double3*) F.cell2rpr)[i], cuv[f].tiltarray,
               ((CD*) cuv[f].pr) + i * F.mt * MSP.ntilt[f] );

         pad0oshift<<<cuv[f].g_pad, cuv[f].b_pad>>>(cuv[f].pr + i * F.mt * MSP.ntilt[f]);
      }
      if (Tr.msp) {
         cudaDeviceSynchronize();
         Tr.pr[f] += timer(&t_msp);
      }

      pmapCmplx1<<<cuv[f].g_pmap,cuv[f].b_pmap>>>(cuv[f].pmap);

      if (Tr.msp) {
         cudaDeviceSynchronize();
         t_msp = time_now();
      }

      for (int i = 0 ; i < F.sl_hi ; i++) {
         int ipg = (F.mseq[i] - 1) * F.mt;
         pmapXpg<<<cuv[f].g_pmap, cuv[f].b_pmap>>>(cuv[f].pmap, cuv[f].pg + ipg);
         if (Tr.msp) {
            cudaDeviceSynchronize();
            Tr.mpg[f] += timer(&t_msp);
         }

//       MSP.sgl ? cufftExecC2C(cuv[f].plan, (CF*) cuv[f].pmap, (CF*) cuv[f].pmap, -1) :
         cufftExecZ2Z(cuv[f].plan, cuv[f].pmap, cuv[f].pmap, -1);
         if (Tr.msp) {
            cudaDeviceSynchronize();
            Tr.fft[f] += timer(&t_msp);
         }

         pmapXpr<<<cuv[f].g_pmap, cuv[f].b_pmap>>>(cuv[f].pmap, cuv[f].pr + ipg * MSP.ntilt[f]);
         if (Tr.msp) {
            cudaDeviceSynchronize();
            Tr.mpr[f] += timer(&t_msp);
         }

         if ((F.sl_opt && i >= F.sl_lo - 1) || i == F.sl_hi - 1) {
            calc_beam<<<cuv[f].g_beam, cuv[f].b_beam>>>(cuv[f].beam, cuv[f].pmap, cuv[f].ib);

            double *host_beam = F.sl_opt ?
                  F.sl_beam + (i + 1 - F.sl_lo) * F.tottilts * F.mbout : F.beam;

            for (int j = 0; j < F.mbout; j++) {
               cudaMemcpyAsync(host_beam + tct1 + j * F.tottilts,
                     cuv[f].beam + j * MSP.ntilt[f], copy_size, cudaMemcpyDeviceToHost);
            }

            if (i == F.sl_hi - 1) break;
         }

         if (Tr.msp) {
            cudaDeviceSynchronize();
            t_msp = time_now();
         }
//       MSP.sgl ? cufftExecC2C(cuv[f].plan, (CF*) dev_pmap, (CF*) dev_pmap, 1) :
         cufftExecZ2Z(cuv[f].plan, cuv[f].pmap, cuv[f].pmap, 1);
         if (Tr.msp) {
            cudaDeviceSynchronize();
            Tr.ifft[f] += timer(&t_msp);
         }
      }
   }

   cudaDeviceSynchronize();
// cudaProfilerStop();
// printf("%s\n", cudaGetErrorString(cudaGetLastError()));
}
```

B.2. *ShiftBeam* – the reflection disc coordinates refining program

B.2.1. File “shiftbeam.cpp” – main body of *ShiftBeam*:

```
/*
QCBEDMS Whole-Pixel Beam Position Matching.

Updates
   29/11/2017 - Tianyu Liu:
      1. Sigma/Variance corrected: assuming input as 'uncertainty'; outputing 'variance'.
   28/11/2017 - Tianyu Liu:
      1. 'in.dat' output with 8 significant figures.
   27/06/2017 - Tianyu Liu:
      1. 2D Array row and column are swapped to 'array[y][x]' for C/C++ performance,
         but function arguments still have 'x' before 'y'.
      2. Added Template for 'float' or 'double', but shifted 'in.dat' output remains 7 significant figures
      3. C++11 required, for timing


Philip Nakashima, ARC Centre of Excellence for Design in Light Metals, Monash University, Vic 3800, Australia.

Conceived Feb 2003, Latest Gatan DM version modified 11/05/2009, C/C++ version 1.0 last modified 08/06/2009.

This program executes the QCBED geometric distortion correction refinement algorithm of [1].
It can be compiled for any platform.
Please refer to this reference if you publish work that made use of this program.

[1] P.N.H. Nakashima, J. Appl. Cryst. 38 (2005), 374.

*/

#include <cstdio>
#include <cstdlib>
#include <cmath>
#include <iostream>
#include <chrono>
#include "commonfunctions_shift.hpp"

using namespace std;

typedef chrono::time_point<chrono::high_resolution_clock> time_point;

time_point now()
{
   return chrono::high_resolution_clock::now();
}

double timer(time_point t)
{
   return chrono::duration<double, std::milli>(now() - t).count();
}

double timer_bin = 0, timer_grad = 0, timer_fit = 0;

struct image_sizes
{
   int xfull, yfull, xlrg, ylrg, xsml, ysml, avgbin, pcbin, xsel, ysel,
         xsmlsel, ysmlsel, xexsel, yexsel, beams_height;
   int afull, alrg, asml, asel, asmlsel, aexsel, abeams, totbin, totbin2;
};

template<typename T>
class image_arrays
{
public:
   cont_2d<T> pattern, sigma, expt, theo, wts, patfin, sigfin,
         patexsel, sigexsel, patsel, sigsel, patsml, sigsml, theo1b, wts1b,
         patsmlsel, sigsmlsel, patsmlgrad, sigsmlgrad, smlseltmp,
         patgrad, siggrad, patlrg, siglrg;

   image_arrays(image_sizes S)
   {
      pattern.alloc(S.xfull, S.yfull);
      sigma.alloc(S.xfull, S.yfull);
      expt.alloc(S.xsml, S.beams_height);
      theo.alloc(S.xsml, S.beams_height);
      wts.alloc(S.xsml, S.beams_height);
      patsel.alloc(S.xsel, S.ysel);
      sigsel.alloc(S.xsel, S.ysel);
      patsml.alloc(S.xsml, S.ysml);
      sigsml.alloc(S.xsml, S.ysml);
      theo1b.alloc(S.xsml, S.ysml);
      wts1b.alloc(S.xsml, S.ysml);
      patfin.alloc(S.xsml, S.beams_height);
      sigfin.alloc(S.xsml, S.beams_height);
      patsmlsel.alloc(S.xsmlsel, S.ysmlsel);
      sigsmlsel.alloc(S.xsmlsel, S.ysmlsel);
      patsmlgrad.alloc(S.xsmlsel, S.ysmlsel);
      sigsmlgrad.alloc(S.xsmlsel, S.ysmlsel);
      smlseltmp.alloc(S.xsmlsel, S.ysmlsel);
      patexsel.alloc(S.xexsel, S.yexsel);
      sigexsel.alloc(S.xexsel, S.yexsel);

      patgrad.alloc(S.xfull + 2 * S.totbin2,S.yfull + 2 * S.totbin2);
      siggrad.alloc(S.xfull + 2 * S.totbin2,S.yfull + 2 * S.totbin2);
   }

// ~image_arrays()
// {
//    dealloc();
// }

   void dealloc()
   {
      pattern.dealloc(), sigma.dealloc(), expt.dealloc(), theo.dealloc();
      wts.dealloc(), patfin.dealloc(), sigfin.dealloc(), patexsel.dealloc(), sigexsel.dealloc();
      patsel.dealloc(), sigsel.dealloc(), patsml.dealloc(), sigsml.dealloc();
      theo1b.dealloc(), wts1b.dealloc(), patsmlsel.dealloc(), sigsmlsel.dealloc();
      patsmlgrad.dealloc(), sigsmlgrad.dealloc(), smlseltmp.dealloc();
   }
};
```

Appendix B. Program scripts

```
bool G2B = true;

template<typename T>
void ImageShift(cont_2d<T>& in, int sx, int sy, cont_2d<T>& out)
{
   int x, y, top, right, bottom, left;

   if (sx > 0) {
      left = 0, right = in.d2x() - sx;
      for (y = 0; y < in.d2y() ; y++)
         for (x = right; x < in.d2x() ; x++)
            out.d2[y][x] = 0;
   } else {
      left = -sx, right = in.d2x();
      for (y = 0; y < in.d2y() ; y++)
         for (x = 0; x < left ; x++)
            out.d2[y][x] = 0;
   }

   if (sy > 0) {
      top = 0, bottom = in.d2y() - sy;
      for (y = bottom; y < in.d2y() ; y++)
         for (x = 0; x < in.d2x() ; x++)
            out.d2[y][x] = 0;
   } else {
      top = -sy, bottom = in.d2y();
      for (y = 0; y < top ; y++)
         for (x = 0; x < in.d2x() ; x++)
            out.d2[y][x] = 0;
   }

   for (y = top; y < bottom ; y++)
      for (x = left; x < right ; x++)
         out.d2[y][x] = in.d2[y + sy][x + sx];
}

template<typename T>
void ImageRegionFromLarger(cont_2d<T>& in, int offx, int offy, cont_2d<T>& out)
{
   for (int x, y = 0 ; y < out.d2y() ; y++)
      for (x = 0; x < out.d2x() ; x++)
         out.d2[y][x] = in.d2[offy + y][offx + x];
}

template<typename T>
void ImageRegionFromSmaller(cont_2d<T>& in, int offx, int offy, cont_2d<T>& out)
{
   for (int x, y = 0 ; y < in.d2y() ; y++)
      for (x = 0; x < in.d2x() ; x++)
         out.d2[offy + y][offx + x] = in.d2[y][x];
}

template<typename T>
void Bin(image_sizes S, cont_2d<T>& in, cont_2d<T>& out)
{
   int x, y, subx, suby, argx, argy, bintemp = floor(S.pcbin / 2.0) * S.avgbin;
   double sum, r_avebin2 = 1.0 / (S.avgbin * S.avgbin);

   for (y = 0; y < out.d2y() ; y++)
      for (x = 0; x < out.d2x() ; x++) {
         sum = 0;
         for (suby = 0; suby < S.avgbin ; suby++)
            for (subx = 0; subx < S.avgbin ; subx++) {
               argx = x * S.totbin + bintemp + subx;
               argy = y * S.totbin + bintemp + suby;
               sum += in.d2[argy][argx];
            }

         out.d2[y][x] = sum * r_avebin2;
      }
}

template<typename T>
void BinSig(image_sizes S, cont_2d<T>& in, cont_2d<T>& out)
{
   int x, y, subx, suby, argx, argy, bintemp = floor(S.pcbin / 2.0) * S.avgbin;
   double sum, r_avebin2 = 1.0 / (S.avgbin * S.avgbin);

   for (y = 0; y < out.d2y() ; y++)
      for (x = 0; x < out.d2x() ; x++) {
         sum = 0;
         for (suby = 0; suby < S.avgbin ; suby++)
            for (subx = 0; subx < S.avgbin ; subx++) {
               argx = x * S.totbin + bintemp + subx;
               argy = y * S.totbin + bintemp + suby;
               sum += in.d2[argy][argx] * in.d2[argy][argx];
            }

         out.d2[y][x] = sqrt(sum) * r_avebin2;
      }
}

template<typename T>
void Gradients(int gs, cont_2d<T>& in, cont_2d<T>& out, cont_2d<T>& tmp)
{
   out = 0.0;

   ImageShift(in,  gs,   0, tmp), out += tmp;
   ImageShift(in, -gs,   0, tmp), out += tmp;
   ImageShift(in,   0,  gs, tmp), out += tmp;
   ImageShift(in,   0, -gs, tmp), out += tmp;
   ImageShift(in,  gs,  gs, tmp), out += tmp;
   ImageShift(in, -gs,  gs, tmp), out += tmp;
   ImageShift(in,  gs, -gs, tmp), out += tmp;
   ImageShift(in, -gs, -gs, tmp), out += tmp;

   out /= 8.0;
   out -= in;
}

template<typename T>
void GradientsSig(int gs, cont_2d<T>& in, cont_2d<T>& out, cont_2d<T>& tmp)
{
   out = 0.0;
   ImageShift(in,  gs,   0, tmp), tmp *= tmp, out += tmp;
   ImageShift(in, -gs,   0, tmp), tmp *= tmp, out += tmp;
   ImageShift(in,   0,  gs, tmp), tmp *= tmp, out += tmp;
   ImageShift(in,   0, -gs, tmp), tmp *= tmp, out += tmp;
   ImageShift(in,  gs,  gs, tmp), tmp *= tmp, out += tmp;
   ImageShift(in, -gs,  gs, tmp), tmp *= tmp, out += tmp;
   ImageShift(in,  gs, -gs, tmp), tmp *= tmp, out += tmp;
   ImageShift(in, -gs, -gs, tmp), tmp *= tmp, out += tmp;
   out /= 64.0;

   tmp = in, tmp *= tmp;
   out += tmp;
   out.sqrt();
}

template<typename T>
double ChiSQ(cont_2d<T>& expt, cont_2d<T>& sig, cont_2d<T>& wts, cont_2d<T>& theo,
double &norm)
{
   int x, y;
   double totala, totalb, qualfit, tmp;
   totala = totalb = qualfit = 0;

   for (x = 0; x < expt.d1x() ; x++) {
      totala += wts.d1[x] * theo.d1[x] / (sig.d1[x] * sig.d1[x]);
      totalb += wts.d1[x] * expt.d1[x] / (sig.d1[x] * sig.d1[x]);
   }

   norm = totala / totalb;

   for (x = 0; x < expt.d1x() ; x++) {
      tmp = theo.d1[x] - norm * expt.d1[x];
      qualfit += wts.d1[x] * tmp * tmp / sig.d1[x];
   }

   qualfit /= wts.sum();

// if (qualfit == 0)
//    return 1000000000000000;
   return qualfit;
}

template<typename T>
double Fit(image_sizes S, int gradopt, int sx, int sy, image_arrays<T> &I, double &norm)
{
   int x, y;
   time_point t_bin, t_grad, t_fit = now();

   ImageRegionFromLarger(I.patexsel, S.totbin2 + sx, S.totbin2 + sy, I.patsel);
   ImageRegionFromLarger(I.sigexsel, S.totbin2 + sx, S.totbin2 + sy, I.sigsel);

   t_bin = now();
   Bin(S, I.patsel, gradopt ? I.patsmlsel : I.patsmlgrad);
   BinSig(S, I.sigsel, gradopt ? I.sigsmlsel : I.sigsmlgrad);
   timer_bin += timer(t_bin);

   if (gradopt == 1) {
      t_grad = now();
      Gradients(1, I.patsmlsel, I.patsmlgrad, I.smlseltmp);
      GradientsSig(1, I.sigsmlsel, I.sigsmlgrad, I.smlseltmp);
      timer_grad += timer(t_grad);
   }

   ImageRegionFromLarger(I.patsmlgrad, 1, 1, I.patsml);
   ImageRegionFromLarger(I.sigsmlgrad, 1, 1, I.sigsml);

   timer_fit += timer(t_fit);
   return ChiSQ(I.patsml, I.sigsml, I.wts1b, I.theo1b, norm);
}

int main()
{
   time_point start = now();
   image_sizes S;
   int numbeams, refcycles, randcycles, gradientopt;
   int x, y, x1, y1;
   double gof, storedgof, norm;
   int sx, sy, sx0, sy0, ss, newsx, newsy;

   FILE* expt = fopen("pattern.dat", "r");
   FILE* sigma = fopen("sigma.dat", "r");
   FILE* exptoutput = fopen("in.dat", "r");
   FILE* theory = fopen("out.dat", "r");
   FILE* params = fopen("parameters.dat", "r");
   FILE* refoutput = fopen("shiftbeam_log.txt", "w");

   fprintf(refoutput, "Beginning the shifting correction of the experimental QCBED data.\n");
   fflush(refoutput);
   fscanf(params, "%i %i %i %i %i %i %i %i %i %i",
          &gradientopt, &S.xfull, &S.yfull, &S.xlrg, &S.ylrg, &S.xsml, &S.ysml,
          &S.avgbin, &S.pcbin, &refcycles, &randcycles);
   fscanf(params, "%i", &numbeams);
```

Appendix B. Program scripts

```
  S.totbin = S.avgbin * S.pcbin, S.totbin2 = 2 * S.totbin;
  S.afull = S.xfull * S.yfull, S.alrg = S.xlrg * S.ylrg, S.asml = S.xsml * S.ysml;
  S.xsel = S.xlrg + S.totbin2, S.ysel = S.ylrg + S.totbin2, S.asel = S.xsel * S.ysel;
  S.xsmlsel = S.xsel / S.totbin, S.ysmlsel = S.ysel / S.totbin, S.asmlsel = S.xsmlsel * S.ysmlsel;
  S.xexsel = S.xsel + 2 * S.totbin2, S.yexsel = S.ysel + 2 * S.totbin2, S.aexsel = S.xexsel * S.yexsel;
  S.beams_height = S.ysml * numbeams, S.abeams = S.xsml * S.beams_height;

  int *newcentx = new int[numbeams], *newcenty = new int[numbeams],
      *centx = new int[numbeams], *centy = new int[numbeams],
      *steps = new int[numbeams], *fitcounts = new int[numbeams];
  double* bg = new double[numbeams];

  for (int i = 0 ; i < numbeams ; i++) {
    fscanf(params, "%i %i %lg", &centx[i], &centy[i], &bg[i]);
    newcentx[i] = centx[i], newcenty[i] = centy[i];
    steps[i] = fitcounts[i] = 0;
  }
  fclose(params);

  image_arrays<double> I(S);

  cont_2d<double> cmap(2 * S.xexsel + 1, 2 * S.yexsel + 1);

  //
  //  Reading data from intensity files into their respective arrays.
  //

  time_point t_read = now();
  for (x = 0; x < S.afull ; x++)
    fscanf(expt, "%lg", &I.pattern.d1[x]);

  for (x = 0; x < S.afull ; x++)
    fscanf(sigma, "%lg", &I.sigma.d1[x]);

  fscanf(exptoutput, "%i %i %i", &S.xsml, &S.ysml, &gradientopt);

  for (x = 0; x < S.abeams ; x++)
    fscanf(theory, "%lg", &I.theo.d1[x]);

  for (x = 0; x < S.abeams ; x++)
    fscanf(exptoutput, "%lg", &I.expt.d1[x]);
  for (x = 0; x < S.abeams ; x++)
    fscanf(exptoutput, "%lg", &I.wts.d1[x]);
  for (x = 0; x < S.abeams ; x++)
    fscanf(exptoutput, "%lg", &I.wts.d1[x]);
  double timer_read = timer(t_read);

// if(G2B) {
//    time_point t_grad = now();
//    Gradients(1, I.patsmlsel, I.patsmlgrad, I.smlseltmp);
//    timer_read += timer(t_grad);
// }

  fclose(expt);
  fclose(sigma);
  fclose(exptoutput);
  fclose(theory);

  for (int ib = 0 ; ib < numbeams ; ib++) {
    fprintf(refoutput, "\n#\tHere begins the refinement of beam %i.\n", ib);
    fflush(refoutput);

    for (y = 0; y < S.yexsel ; y++)
      for (x = 0; x < S.xexsel ; x++) {
        x1 = newcentx[ib] - (S.xexsel / 2) + x;
        y1 = newcenty[ib] - (S.yexsel / 2) + y;
        if ( (x1 < S.xfull) && (y1 < S.yfull) && (x1 >= 0) && (y1 >= 0)) {
          I.patexsel.d2[y][x] = I.pattern.d2[y1][x1];
          I.sigexsel.d2[y][x] = I.sigma.d2[y1][x1];
        } else {
          I.patexsel.d2[y][x] = 0.0;
          I.sigexsel.d2[y][x] = 1.0;
        }
      }

    I.patexsel -= bg[ib];
    CopyImage(S.asml, I.theo.d1 + ib * S.asml, I.theo1b.d1);
    CopyImage(S.asml, I.wts.d1 + ib * S.asml, I.wts1b.d1);

    //
    //  Raw data binning 8-direactions ChiSQ minimisation.
    //

    cmap = -1.0;
    ss = S.totbin;
    ss = 0;
    sx = sy = newsx = newsy = 0, sx0 = newcentx[ib] - centx[ib], sy0 = newcenty[ib] - centy[ib];
    storedgof = gof = Fit(S, gradientopt, 0, 0, I, norm);
    fitcounts[ib]++;
    fprintf(refoutput, "\nStarting ChiSQ = %lg\n", storedgof);
    fflush(refoutput);

    while (ss > 0 && abs(newsx) < S.totbin2 && abs(newsy) < S.totbin2) {
      int xys[8][2] = { { newsy - ss, newsx }, { newsy + ss, newsx },
          { newsy, newsx - ss }, { newsy, newsx + ss },
          { newsy - ss, newsx - ss }, { newsy - ss, newsx + ss },
          { newsy + ss, newsx - ss }, { newsy + ss, newsx + ss } };

      fprintf(refoutput, "\nBeam %i, step %i, current (relative) centre (%i, %i). "
          "ChiSQ = %lg. Step size = %i\n", ib, ++steps[ib], newsx, newsy, storedgof, ss);
      fflush(refoutput);

      for (int i = 0 ; i < 8 ; i++) {
        int sxi = xys[i][1], syi = xys[i][0];

        if (cmap.d2[syi + S.ysel][sxi + S.xsel] == -1) {
          cmap.d2[syi + S.ysel][sxi + S.xsel] = gof =
              Fit(S, gradientopt, sxi, syi, I, norm);
          fitcounts[ib]++;
        } else
          gof = cmap.d2[syi + S.ysel][sxi + S.xsel];

        if (gof < storedgof)
          storedgof = gof, newsx = sxi, newsy = syi;

        fprintf(refoutput, "\t%i. (%i, %i) - %lg\n", (1 + i), sx0 + sxi, sy0 + syi, gof);
        fflush(refoutput);
      }

      if (sx == newsx && sy == newsy)
        ss = floor(ss / 2.0);
      else
        sx = newsx, sy = newsy;

      fprintf(refoutput, "\tBest: (%i, %i) - %lg\n", sx0 + newsx, sy0 + newsy, storedgof);
      fflush(refoutput);
    }

    newcentx[ib] += newsx;
    newcenty[ib] += newsy;

    if (abs(newsx) >= S.totbin2 || abs(newsy) >= S.totbin2) {
      fprintf(refoutput, "\n!!! Shifting reached edge of selected region with centre (%i,%i) !!!\n"
          "!!! Reload Beam %i from new centre (%i,%i) and restart !!!\n",
          centx[ib], centy[ib], ib, newcentx[ib], newcenty[ib]);
      fflush(refoutput);
      --ib;
      continue;
    }

    //
    //  Summarise final output
    //
    Fit(S, gradientopt, newsx, newsy, I, norm);
    CopyImage(S.asml, I.patsml.d1, I.patfin.d1 + ib * S.asml);
    CopyImage(S.asml, I.sigsml.d1, I.sigfin.d1 + ib * S.asml);
  }

  //
  //  Print output to the pattern-matching input file.
  //

  exptoutput = fopen("in.dat", "w");
  fprintf(exptoutput, "%i %i %i\n", S.xsml, S.ysml, gradientopt);
  fflush(exptoutput);

  for (y = 0; y < S.beams_height ; y++) {
    for (x = 0; x < S.xsml ; x++)
      fprintf(exptoutput, "%.8g\t", I.patfin.d2[y][x]);
    fprintf(exptoutput, "\n");
  }
  for (y = 0; y < S.beams_height ; y++) {
    for (x = 0; x < S.xsml ; x++)
      fprintf(exptoutput, "%.8g\t", I.sigfin.d2[y][x]);
    fprintf(exptoutput, "\n");
  }
  for (y = 0; y < S.beams_height ; y++) {
    for (x = 0; x < S.xsml ; x++)
      fprintf(exptoutput, "%.8g\t", I.wts.d2[y][x]);
    fprintf(exptoutput, "\n");
  }
  fflush(exptoutput);

  gof = ChiSQ(I.patfin, I.sigfin, I.wts, I.theo, norm);

  fprintf(refoutput, "\nThe final normalisation constant is: %f \n", norm);
  fprintf(refoutput, "The final chi squared is: %f \n", gof);
  for (int i = 0 ; i < numbeams ; i++)
    fprintf(refoutput, "Beam %i: final shift (x,y)=(%i,%i), "
        "took %i binning steps and %i new calculations\n",
        i, newcentx[i] - centx[i], newcenty[i] - centy[i], steps[i], fitcounts[i]);
  fflush(refoutput);

  fclose(exptoutput);

  //
  // Create new 'parameters.dat' and backup the old one
  //

  FILE* params_new = fopen("parameters.tmp", "w");
  fprintf(params_new, "%i %i %i %i %i %i %i %i %i %i\n",
      gradientopt, S.xfull, S.yfull, S.xlrg, S.ylrg, S.xsml,
      S.ysml, S.avgbin, S.pcbin, refcycles, randcycles);
  fprintf(params_new, "%i\n", numbeams);
  for (int i = 0 ; i < numbeams ; i++)
    fprintf(params_new, "%i %i %lg\n", newcentx[i], newcenty[i], bg[i]);
  fflush(params_new);
  fclose(params_new);

  remove("parameters.bak");
  rename("parameters.dat","parameters.bak");
  rename("parameters.tmp","parameters.dat");

  fprintf(refoutput, "\nTotal duration: %g ms, image read-in: %g ms\n", timer(start), timer_read);
  fprintf(refoutput, "Fitting took %g ms (binning: %g ms, gradients: %g ms)\n",
      timer_fit, timer_bin, timer_grad);
  fflush(refoutput);
  fclose(refoutput);
}
```

Appendix B. Program scripts

B.2.2. File "commonfunctions_shift.hpp" – functions used by *ShiftBeam*:

```
// This routine takes an input image of arbitrary size and applies a geometric
// transformation to it, returning the transformed image.
//
// Updates
// 27/06/2017 - Tianyu Liu:
//     1. Array row and column are swapped to 'array[y][x]',
//        but arguments of functions still have 'x' before 'y'.
//     2. Added Template for 'float' or 'double'
//

#include <cstdio>
#include <cstdlib>
#include <cmath>
#include <algorithm>

using namespace std;

double randomnumber(int nut)
{
   double rnumb;

   srand(nut);
   rnumb = rand() % 32768;
   rnumb = rnumb / 32768;
   return (rnumb);
}

template<typename T>
T** MakeArray(int dimx, int dimy)
{
   T** newarray = new T*[dimy];

   for (int j = 0 ; j < dimy ; j++)
      newarray[j] = new T[dimx];

   return newarray;
}

template<typename T>
class cont_2d
{
private:
   int d1x_, d2x_, d2y_;

public:
   T* d1;
   T** d2;

   cont_2d()
   {
      d1 = NULL, d2 = NULL, d1x_ = d2x_ = d2y_ = 0;
   }

   cont_2d(int dimx, int dimy)
   {
      alloc(dimx, dimy);
   }

// ~cont_2d()
// {
//    dealloc();
// }

   void alloc(int dimx, int dimy)
   {
      d1x_ = dimx * dimy, d2x_ = dimx, d2y_ = dimy;
      d1 = new T[d1x_], d2 = new T*[d2y_];

      for (int i = 0 ; i < d2y_ ; i++)
         d2[i] = d1 + i * d2x_;
   }

   void dealloc()
   {
      delete[] d1;
      delete[] d2;
      d1x_ = d2x_ = d2y_ = 0;
   }

   int d1x()
   {
      return d1x_;
   }

   int d2x()
   {
      return d2x_;
   }

   int d2y()
   {
      return d2y_;
   }

   inline cont_2d& operator =(T c)
   {
      fill(d1, d1 + d1x_, c);
      return *this;
   }

   inline cont_2d& operator =(const cont_2d& other)
   {
      copy(other.d1, other.d1 + d1x_, d1);
      return *this;
   }

   inline cont_2d& operator +=(T c)
   {
      for (auto x = d1 ; x != d1 + d1x_ ; x++)
         *x += c;
      return *this;
   }

   inline cont_2d& operator +=(const cont_2d& other)
   {
      for (auto x = d1, x2 = other.d1 ; x != d1 + d1x_ ; x++, x2++)
         *x += *x2;
      return *this;
   }

   inline cont_2d& operator -=(T c)
   {
      for (auto x = d1 ; x != d1 + d1x_ ; x++)
         *x -= c;
      return *this;
   }

   inline cont_2d& operator -=(const cont_2d& other)
   {
      for (auto x = d1, x2 = other.d1 ; x != d1 + d1x_ ; x++, x2++)
         *x -= *x2;
      return *this;
   }

   inline cont_2d& operator *=(T c)
   {
      for (auto x = d1 ; x != d1 + d1x_ ; x++)
         *x *= c;
      return *this;
   }

   inline cont_2d& operator *=(const cont_2d& other)
   {
      for (auto x = d1, x2 = other.d1 ; x != d1 + d1x_ ; x++, x2++)
         *x *= *x2;
      return *this;
   }

   inline cont_2d& operator /=(T c)
   {
      T rc = 1.0 / c;
      for (auto x = d1 ; x != d1 + d1x_ ; x++)
         *x *= rc;
      return *this;
   }

   inline cont_2d& operator /=(const cont_2d& other)
   {
      for (auto x = d1, x2 = other.d1 ; x != d1 + d1x_ ; x++, x2++)
         *x /= *x2;
      return *this;
   }

   void sqrt()
   {
      for (auto x = d1 ; x != d1 + d1x_ ; x++)
         *x = std::sqrt(*x);
   }

   T sum()
   {
      T s = 0;
      for (auto x = d1 ; x != d1 + d1x_ ; x++)
         s += *x;
      return s;
   }
}
;

template<typename T>
void CopyImage(int dimx, int dimy, T** inimage, T** outimage)
{
   for (int x, y = 0 ; y < dimy ; y++)
      for (x = 0; x < dimx ; x++)
         outimage[y][x] = inimage[y][x];
}

template<typename T>
void CopyImage(int dimx, T* inimage, T* outimage)
{
   for (auto x = 0 ; x < dimx ; x++)
      outimage[x] = inimage[x];
}

template<typename T>
T SumImage(int dimx, int dimy, T** inimage)
{
   T sum = 0.0;
   for (int x, y = 0 ; y < dimy ; y++)
      for (x = 0; x < dimx ; x++)
```

Appendix B. Program scripts

```
        sum += inimage[y][x];

    return sum;
}

template<typename T>
void ImageSetConst(int dimx, int dimy, T** inimage, T c)
{
    for (int x, y = 0 ; y < dimy ; y++)
        for (x = 0; x < dimx ; x++)
            inimage[y][x] = c;
}

template<typename T>
void ImageDistort(int dimx, int dimy, T** inimage, T pa[12], T** outimage)
{
    int x, y, oldx, oldy;
    T exactoldx, exactoldy, pix1, pix2, pix3, pix4, fracx, fracy, delta, newpix,
        truncoldx, truncoldy;

    delta = 1e-20;

    for (y = 0; y < dimy ; y++)
        for (x = 0; x < dimx ; x++) {
            exactoldx = pa[0] + pa[1] * x + pa[2] * y + pa[3] * x * x + pa[4] * y * y + pa[5] * x * y;
            exactoldy = pa[6] + pa[7] * x + pa[8] * y + pa[9] * x * x + pa[10] * y * y + pa[11] * x * y;

            oldx = exactoldx;
            oldy = exactoldy;
            truncoldx = floor(exactoldx);
            truncoldy = floor(exactoldy);
            fracx = exactoldx - truncoldx;
            fracy = exactoldy - truncoldy;

            pix1 = pix2 = pix3 = pix4 = 0.0;

            if (oldx >= 0 && oldy >= 0 && oldx < dimx && oldy < dimy)
                pix1 = inimage[oldy][oldx];
            if (oldx + 1 >= 0 && oldy >= 0 && oldx + 1 < dimx && oldy < dimy)
                pix2 = inimage[oldy][oldx + 1];
            if (oldx >= 0 && oldy + 1 >= 0 && oldx < dimx && oldy + 1 < dimy)
                pix3 = inimage[oldy + 1][oldx];
            if (oldx + 1 >= 0 && oldy + 1 >= 0 && oldx + 1 < dimx && oldy + 1 < dimy)
                pix4 = inimage[oldy + 1][oldx + 1];

            newpix = (1 - fracx) * (1 - fracy) * pix1 + fracx * (1 - fracy) * pix2 + fracy * (1 - fracx) * pix3
+ fracx * fracy * pix4;
            outimage[y][x] = newpix;
        }
}

template<typename T>
void ImageAddConst(int dimx, int dimy, T** inimage, T constant, T** outimage)
{
    for (int x, y = 0 ; y < dimy ; y++)
        for (x = 0; x < dimx ; x++)
            outimage[y][x] = inimage[y][x] + constant;
}

template<typename T>
void ImageSubConst(int dimx, int dimy, T** inimage, T constant, T** outimage)
{
    for (int x, y = 0 ; y < dimy ; y++)
        for (x = 0; x < dimx ; x++)
            outimage[y][x] = inimage[y][x] - constant;
}

template<typename T>
void ImageMulConst(int dimx, int dimy, T** inimage, T constant, T** outimage)
{
    for (int x, y = 0 ; y < dimy ; y++)
        for (x = 0; x < dimx ; x++)
            outimage[y][x] = constant * inimage[y][x];
}

template<typename T>
void ImageDivConst(int dimx, int dimy, T** inimage, T constant, T** outimage)
{
    T rconst = 1.0 / constant;
    for (int x, y = 0 ; y < dimy ; y++)
        for (x = 0; x < dimx ; x++)
            outimage[y][x] = inimage[y][x] * rconst;
}

template<typename T>
void ImageAddImage(int dimx, int dimy, T** inimage, T** inimage2, T** outimage)
{
    for (int x, y = 0 ; y < dimy ; y++)
        for (x = 0; x < dimx ; x++)
            outimage[y][x] = inimage[y][x] + inimage2[y][x];
}

template<typename T>
void ImageSubImage(int dimx, int dimy, T** inimage, T** inimage2, T** outimage)
{
    for (int x, y = 0 ; y < dimy ; y++)
        for (x = 0; x < dimx ; x++)
            outimage[y][x] = inimage[y][x] - inimage2[y][x];
}

template<typename T>
void ImageMulImage(int dimx, int dimy, T** inimage, T** inimage2, T** outimage)
{
    for (int x, y = 0 ; y < dimy ; y++)
        for (x = 0; x < dimx ; x++)
            outimage[y][x] = inimage[y][x] * inimage2[y][x];
}

template<typename T>
void ImageDivImage(int dimx, int dimy, T** inimage, T** inimage2, T** outimage)
{
    for (int x, y = 0 ; y < dimy ; y++)
        for (x = 0; x < dimx ; x++)
            outimage[y][x] = inimage[y][x] / inimage2[y][x];
}

template<typename T>
void SQRTImage(int dimx, int dimy, T** inimage, T** outimage)
{
    for (int x, y = 0 ; y < dimy ; y++)
        for (x = 0; x < dimx ; x++)
            outimage[y][x] = sqrt(inimage[y][x]);
}
```

B.3. *QCBED-BPrep* – a Gatan DigitalMicrograph script for batch QCBED data preparation

B.3.1. File "Batch IGQCBED Preparation_v1.0.2.s" – main body of *QCBED-BPrep*

```
// Batch IGQCBED Preparation (Batch Intensity-Gradient-QCBED Preparation)
// A intergated hub to prepare QCBED data in bulk
// for DM versions 2.32 and maybe later
// Tianyu Liu, Aug 2016

// Changelog
// v 1.0.2: Fixed 'ImageRegion' out of boundary error
// v 1.0.1: 1) Fixed sigma == 0 ? 1 : sigma; 2) jpg overview; 3) Removed the 4th pair of pattern
coordinates; 4) Draw Mask out of boundary fixed
// V 1.0

// 1.1	AMMPSF child function: Aperture Method for Measuring Point Spread Functions
(AMMPSF)
// by Philip Nakashima  21/08/2001
// P. N. H. Nakashima, A. W. S. Johnson, Measuring the PSF from aperture images of arbitrary
shape—an algorithm. Ultramicroscopy 94, 135-148 (2003).
// Adapted for batch processing by Tianyu Liu,
// for the original version with explanatory comments please refer to
"http://www.philipnakashima.com/"

// 1.2	Noise Characterisation child function: Noise as a function of signal measurement script
// by Philip N.H. Nakashima, July 1 2008 (Bye bye Golf Sniff!), last modified 12th September 2011
- correct physical model for noise ie: sigma = a.I + b.Sqrt(I) + c
// P. N. H. Nakashima, In situ quantification of noise as a function of signal in digital images. Optics
Letters 37, 1023-1025 (2012).
//  Adapted for batch pocessing by Tianyu Liu; correct "noise mask" edge-stroking.

// 1.3	IGQCBED child functions: Intensity gradient QCBED script for preparation of data for
pattern-matching.
// by Philip Nakashima, Conceived 28th Oct 2008, Last Modified 9:30am, 21/06/2011
// P. N. H. Nakashima, B. C. Muddle, Differential convergent beam electron diffraction: Experiment
and theory. Physical Review B 81, 115135 (2010).
// P. N. H. Nakashima, B. C. Muddle, Differential quantitative analysis of background structure in
energy-filtered convergent-beam electron diffraction patterns. Journal of Applied Crystallography
43, 280-284 (2010).
// Adapted for batch pocessing by Tianyu Liu

// Table of Contents (copy to find)
// 0.0	Global Variables Declaration
// 0.1	Variable passing functions
// 0.1.1	Variables to 'settings' TagGroup
// 0.1.2	'settings' TagGroup to Variables
// 0.1.3	Panel Data to Variables
// 0.2	Recording functions
// 0.3	Common functions
// 0.3.1	Image shift (region copy)
// 0.3.2	Image region (region copy)

// 1.1	AMMPSF child function
// 1.2	Noise Characterisation child function
// 1.3	IGQCBED child functions
// 1.3.1	Draw Mask step child function
// 1.3.2	Pattern coordinates function
// 1.3.3	IGQCBED step child function
// 1.3.4	FOR007.DAT step child function
// 1.3.5	parameters.dat step child function

// 2	Batch Processing looping functions
// 2.0.1	Update current paths function
// 2.1	AMMPSF looping function
// 2.2	PSF averaging function
// 2.3	PSF correction looping function
```

Appendix B. Program scripts

```
// 2.4	Noise characterisation looping function
// 2.5	IGQCBED looping function
// 2.6	Extra step looping function

// 3	GUI functions
// 3.1	AMMPSF tag
// 3.2	PSF correction tag
// 3.3	Noise Characterisation tag
// 3.4	IGQCBED tag
// 3.4.1	Check processing files for IGQCBED
// 3.5	Extra tag
// 3.6	Main Panel grouping
// 3.7	The Test function
// 3.8	Background thread function
// 3.9	GUI class: panel respone functions (buttons & options etc.)

// 4	Main function


//
// 0.0	Global Variables Declaration
//

String m_dir		//Main directory
String sub_dir		//Temporary path for subfolders during looping
String img_path		//Temperary path for images furing looping
String img_name		//Temperary filename for images furing looping
String sub_name		//Temporary folder name for subfolders during looping
Number noi		//The loop variable for image looping & subfolder looping

String img_pre,img_suf	//Image name prefix & suffix
String sub_pre,sub_suf	//Subfolder name prefix & suffix
Number num_start,num_end	//Looping tries from 'num_start' to 'num_end'

// AMMPSF variables
Number psf_nummeas = 25	//Number of measurements for each PSF image

// PSF Correction variables
String psf_path		//The Averaged PSF used for PSF Correction (deconvolution)

// Noise Characterisation variables
Number nc_mask = 0		//Option to whether apply pre-determined Mask
String nc_mask_path	//Mask image path
Number nc_normal = 1e+9	//Gain Normalisation target
Number nc_numslice = 1000	//Number of slices for sampling
Number nc_bin = 1		//Piecewise binning factor
Number nc_darknoise	= -1 //Dark Noise
Number nc_txt = 1		//Whether to save the outputs as text images as well
Number nc_raw = 0		//Whether to proceed on raw data instead of PSF corrected ones

// IGQCBED variables
Number ig_numbeams	//Number of beams
Number ig_dia		//Diameter of beams (Width & Height of selection)
Number ig_bin		//Binning factor
Number ig_switch = 1	//Switch for pattern-matching & distortion-correction programs to turn on
"gradient"; does not affect preparation

Number ig_step_mask	//Switch for step '1. Draw Mask'
Number ig_step_igqcbed	//Switch for step '2. IGQCBED'
Number ig_step_for7	//Switch for step '3. FOR007.DAT'
Number ig_step_filepm	//Switch for step '4. parameters.dat'

Number ig_ratio = 0.9	//Mask ratio for step '1. Draw Mask'
Number ig_magn = 1		//Magnification for step '2. IGQCBED'
Number ig_finaltxt = 1	//Text image output option for step '2. IGQCBED'
Number ig_noisemask = 1	//Apply noise mask option for step '2. IGQCBED'
// Variables for GeoMetric DIstortion Correction's command file 'parameters.dat'; do not affect
preparation
Number ig_piecebin = 1	//Piecewise binning for 'parameters.dat'
Number gdc_cycles = 1800 //Total refinement cycles for 'parameters.dat'
Number gdc_rand = 10	//Random step numbers for 'parameters.dat'

// Extra Step variables
Number ex_loop		//Extra external script's looping mode
String ex_script_path	//Path of extra external script

Object mainpanel_win	//Window object of the GUI
Number stage = 0		//Which tag to display (0 = initially the first tag)
String settings_path	//Path of the settings' file 'settings.gtg'
String exit_text		//Exit message for extra external script, passing through
GetUserPersistentTagGroup()
Number exit_code		//Exit code for extra external script, passing through Exit(Number exit_code)
Number running = 0		//Busy indicator for GUI
Number abort = 0		//Abort-halfway indicator for GUI
String log_txt = "", log2_txt = ""//Strings to store batch process logs and child process logs
Number test_mode = 0	//Switch for the 'Test' button in Extra tag

DocumentWindow open_win	//Temporary window object for "Open..." window

TagGroup settings = NewTagGroup()	//The TagGroup to store settings

// GUI items for above variables
// '*_f' for field objects, '*_tabs' for tab lists, '*_r' for radio lists, '*_c' for checkboxes
TagGroup m_dir_f
TagGroup img_pre_f,img_suf_f,sub_pre_f,sub_suf_f,num_start_f,num_end_f
TagGroup stage_tabs

TagGroup psf_nummeas_f
TagGroup psf_path_f

TagGroup nc_mask_r
TagGroup nc_mask_path_f,nc_normal_f
TagGroup nc_numslice_f,nc_bin_f,nc_darknoise_f
TagGroup nc_raw_c
TagGroup nc_txt_c
TagGroup nc_fewer_c

TagGroup ig_numbeams_f, ig_dia_f, ig_bin_f, ig_switch_f
TagGroup ig_step_mask_c, ig_ratio_f
TagGroup ig_step_igqcbed_c, ig_magn_f, ig_noisemask_c, ig_finaltxt_c
TagGroup ig_step_for7_c
TagGroup ig_step_filepm_c, ig_piecebin_f, gdc_cycles_f, gdc_rand_f

TagGroup ex_loop_r
TagGroup ex_script_path_f

//Pre-declaration for class method
Interface PreUIFrame
{
   Void OptionChanged(Object self, TagGroup self1);
}

////////////////////////////////////////////////////////
//
// 0.1	Variable passing functions

// 0.1.1	Variables to 'settings' TagGroup
TagGroup VarToTag()
{
   settings = NewTagGroup()
   TagGroupSetTagAsNumber(settings,"stage",stage)

   TagGroupSetTagAsNumber(settings,"num_start",num_start)
   TagGroupSetTagAsNumber(settings,"num_end",num_end)
   TagGroupSetTagAsString(settings,"img_pre",img_pre)
   TagGroupSetTagAsString(settings,"img_suf",img_suf)
   TagGroupSetTagAsString(settings,"sub_pre",sub_pre)
   TagGroupSetTagAsString(settings,"sub_suf",sub_suf)

   TagGroupSetTagAsNumber(settings,"psf_nummeas",psf_nummeas)
   TagGroupSetTagAsString(settings,"psf_path",psf_path)

   TagGroupSetTagAsNumber(settings,"nc_numslice",nc_numslice)
   TagGroupSetTagAsNumber(settings,"nc_bin",nc_bin)
   TagGroupSetTagAsNumber(settings,"nc_darknoise",nc_darknoise)
   TagGroupSetTagAsNumber(settings,"nc_mask",nc_mask)
   TagGroupSetTagAsString(settings,"nc_mask_path",nc_mask_path)
   TagGroupSetTagAsNumber(settings,"nc_normal",nc_normal)
   TagGroupSetTagAsNumber(settings,"nc_txt",nc_txt)
   TagGroupSetTagAsNumber(settings,"nc_raw",nc_raw)

   TagGroupSetTagAsNumber(settings,"ig_numbeams",ig_numbeams)
   TagGroupSetTagAsNumber(settings,"ig_dia",ig_dia)
   TagGroupSetTagAsNumber(settings,"ig_bin",ig_bin)
   TagGroupSetTagAsNumber(settings,"ig_switch",ig_switch)
   TagGroupSetTagAsNumber(settings,"ig_ratio",ig_ratio)
   TagGroupSetTagAsNumber(settings,"ig_magn",ig_magn)
   TagGroupSetTagAsNumber(settings,"ig_finaltxt",ig_finaltxt)
   TagGroupSetTagAsNumber(settings,"ig_noisemask",ig_noisemask)
   TagGroupSetTagAsNumber(settings,"ig_piecebin",ig_piecebin)
   TagGroupSetTagAsNumber(settings,"gdc_cycles",gdc_cycles)
   TagGroupSetTagAsNumber(settings,"gdc_rand",gdc_rand)

   TagGroupSetTagAsString(settings,"ex_script_path",ex_script_path)
   TagGroupSetTagAsNumber(settings,"ex_loop",ex_loop)
}
// 0.1.2	'settings' TagGroup to Variables
Void TagToVar()
{
   //TagGroupGetTagAsString(settings,"m_dir",m_dir)
   //TagGroupGetTagAsString(settings,"sub_dir",sub_dir)
   //TagGroupGetTagAsString(settings,"img_path",img_path)
   //TagGroupGetTagAsString(settings,"img_name",img_name)
   //TagGroupGetTagAsString(settings,"sub_name",sub_name)
   //TagGroupGetTagAsNumber(settings,"noi",noi)
   //TagGroupGetTagAsString(settings,"exit_text",exit_text)
   //TagGroupGetTagAsNumber(settings,"abort",abort)
   TagGroupGetTagAsNumber(settings,"stage",stage)

   TagGroupGetTagAsNumber(settings,"num_start",num_start)
   TagGroupGetTagAsNumber(settings,"num_end",num_end)
   TagGroupGetTagAsString(settings,"img_pre",img_pre)
   TagGroupGetTagAsString(settings,"img_suf",img_suf)
   TagGroupGetTagAsString(settings,"sub_pre",sub_pre)
   TagGroupGetTagAsString(settings,"sub_suf",sub_suf)

   TagGroupGetTagAsNumber(settings,"psf_nummeas",psf_nummeas)
   TagGroupGetTagAsString(settings,"psf_path",psf_path)

   TagGroupGetTagAsNumber(settings,"nc_numslice",nc_numslice)
   TagGroupGetTagAsNumber(settings,"nc_bin",nc_bin)
   TagGroupGetTagAsNumber(settings,"nc_darknoise",nc_darknoise)
   TagGroupGetTagAsNumber(settings,"nc_mask",nc_mask)
   TagGroupGetTagAsString(settings,"nc_mask_path",nc_mask_path)
   TagGroupGetTagAsNumber(settings,"nc_normal",nc_normal)
   TagGroupGetTagAsNumber(settings,"nc_txt",nc_txt)
   TagGroupGetTagAsNumber(settings,"nc_raw",nc_raw)

   TagGroupGetTagAsNumber(settings,"ig_numbeams",ig_numbeams)
   TagGroupGetTagAsNumber(settings,"ig_dia",ig_dia)
   TagGroupGetTagAsNumber(settings,"ig_bin",ig_bin)
   TagGroupGetTagAsNumber(settings,"ig_switch",ig_switch)
   TagGroupGetTagAsNumber(settings,"ig_ratio",ig_ratio)
   TagGroupGetTagAsNumber(settings,"ig_magn",ig_magn)
   TagGroupGetTagAsNumber(settings,"ig_noisemask",ig_noisemask)
   TagGroupGetTagAsNumber(settings,"ig_finaltxt",ig_finaltxt)
   TagGroupGetTagAsNumber(settings,"ig_piecebin",ig_piecebin)
   TagGroupGetTagAsNumber(settings,"gdc_cycles",gdc_cycles)
```

Appendix B. Program scripts

```
   TagGroupGetTagAsNumber(settings,"gdc_rand",gdc_rand)

   TagGroupGetTagAsString(settings,"ex_script_path",ex_script_path)
   TagGroupGetTagAsNumber(settings,"ex_loop",ex_loop)
}

// 0.1.3   Panel Data to Variables
Void PanelToVar()
{
   DLGGetValue(m_dir_f,m_dir)
   DLGGetValue(num_start_f,num_start)
   DLGGetValue(num_end_f,num_end)
   DLGGetValue(img_pre_f,img_pre)
   DLGGetValue(img_suf_f,img_suf)
   DLGGetValue(sub_pre_f,sub_pre)
   DLGGetValue(sub_suf_f,sub_suf)
   DLGGetValue(stage_tabs,stage)

   DLGGetValue(psf_nummeas_f,psf_nummeas)
   DLGGetValue(psf_path_f,psf_path)

   DLGGetValue(nc_numslice_f,nc_numslice)
   DLGGetValue(nc_bin_f,nc_bin)
   DLGGetValue(nc_mask_r,nc_mask)
   DLGGetValue(nc_darknoise_f,nc_darknoise)
   DLGGetValue(nc_mask_path_f,nc_mask_path)
   DLGGetValue(nc_normal_f,nc_normal)
   DLGGetValue(nc_txt_c,nc_txt)
   DLGGetValue(nc_raw_c,nc_raw)

   DLGGetValue(ig_numbeams_f,ig_numbeams)
   DLGGetValue(ig_bin_f,ig_bin)
   DLGGetValue(ig_dia_f,ig_dia)
   DLGGetValue(ig_switch_f,ig_switch)

   DLGGetValue(ig_step_mask_c,ig_step_mask)
   DLGGetValue(ig_step_igqcbed_c,ig_step_igqcbed)
   DLGGetValue(ig_step_for7_c,ig_step_for7)
   DLGGetValue(ig_step_filepm_c,ig_step_filepm)
   DLGGetValue(ig_ratio_f,ig_ratio)
   DLGGetValue(ig_magn_f,ig_magn)
   DLGGetValue(ig_noisemask_c,ig_noisemask)
   DLGGetValue(ig_finaltxt_c,ig_finaltxt)
   DLGGetValue(ig_piecebin_f,ig_piecebin)
   DLGGetValue(gdc_cycles_f,gdc_cycles)
   DLGGetValue(gdc_rand_f,gdc_rand)

   DLGGetValue(ex_script_path_f,ex_script_path)
   DLGGetValue(ex_loop_r,ex_loop)
}
// 0.1.4   Variables to Panel
Void VarToPanel()
{
   DLGValue(m_dir_f,m_dir)
   DLGValue(num_start_f,num_start)
   DLGValue(num_end_f,num_end)
   DLGValue(img_pre_f,img_pre)
   DLGValue(img_suf_f,img_suf)
   DLGValue(sub_pre_f,sub_pre)
   DLGValue(sub_suf_f,sub_suf)
   DLGValue(stage_tabs,stage)

   DLGValue(psf_nummeas_f,psf_nummeas)
   DLGValue(psf_path_f,psf_path)

   DLGValue(nc_numslice_f,nc_numslice)
   DLGValue(nc_bin_f,nc_bin)
   DLGValue(nc_mask_r,nc_mask)
   DLGValue(nc_darknoise_f,nc_darknoise)
   DLGValue(nc_mask_path_f,nc_mask_path)
   DLGValue(nc_normal_f,nc_normal)
   DLGValue(nc_txt_c,nc_txt)
   DLGValue(nc_raw_c,nc_raw)

   DLGValue(ig_numbeams_f,ig_numbeams)
   DLGValue(ig_bin_f,ig_bin)
   DLGValue(ig_dia_f,ig_dia)
   DLGValue(ig_switch_f,ig_switch)

   DLGValue(ig_step_mask_c,ig_step_mask)
   DLGValue(ig_step_igqcbed_c,ig_step_igqcbed)
   DLGValue(ig_step_for7_c,ig_step_for7)
   DLGValue(ig_step_filepm_c,ig_step_filepm)
   DLGValue(ig_ratio_f,ig_ratio)
   DLGValue(ig_magn_f,ig_magn)
   DLGValue(ig_noisemask_c,ig_noisemask)
   DLGValue(ig_finaltxt_c,ig_finaltxt)
   DLGValue(ig_piecebin_f,ig_piecebin)
   DLGValue(gdc_cycles_f,gdc_cycles)
   DLGValue(gdc_rand_f,gdc_rand)

   DLGValue(ex_script_path_f,ex_script_path)
   DLGValue(ex_loop_r,ex_loop)
}

// 0.1     Variable passing functions end
//
////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////
//
// 0.2     Recording functions

// Use 'Log("Text")' to log batch procedures to 'Notes'
// and save to the 'Misc' folder with 'FinishLog("batch process name")'
Void Log(String text)
{
   log_txt += text
   Notes(text)
}

Void StartLog(String name)
{
   log_txt = ""
   ClearNotes()
   OpenOutputWindow()
   Notes("Previous Notes cleared!\n")

   Log("\nBatch IGQCBED Preparation: " + name)
   Log("\n\tBegan at " + DateStamp())
   Log("\nSpecimens Main directory:")
   Log("\n\t" + m_dir)
}

Void FinishLog(String name)
{
   Log("\n\n\tFinished at " + DateStamp())

   CreateDirectory(m_dir + "Misc\\")
   Number saved = 1,k = -1
   While(saved == 1)
      saved = DoesFileExist(m_dir + "Misc\\Logs_" + name + "_" + ++k + ".txt")

   Number log_id = CreateFileForWriting(m_dir + "Misc\\Logs_" + name + "_" + k + ".txt")
   Object log_stream = NewStreamFromFileReference(log_id,1)
         log_stream.StreamWriteAsText(0,log_txt)
   CloseFile(log_id)
   log_txt = ""

   Notes("\n\nNotes automatically saved to 'Misc\\Logs_" + name + "_" + k + ".txt'")
}

// Use 'Log2("Text")' to log detailed processing information to 'Results'
// and save to a specific path with 'FinishLog2("path")'
Void Log2(String text)
{
   log2_txt += text
   Result(text)
}

Void StartLog2(String text) {
   log2_txt = ""
   ClearResults()
   Result("Previous Results cleared!\n\n")
   Log2(text)
   Log2("\n\tBegan at " + DateStamp() + "\n\n")
}

Void FinishLog2(String path)
{
   Log2("\n\nFinished at " + DateStamp())
   Number log2_id = CreateFileForWriting(path)
   Object log2_stream = NewStreamFromFileReference(log2_id,1)
         log2_stream.StreamWriteAsText(0,log2_txt)
   CloseFile(log2_id)
   log2_txt = ""
   Result("\n\nResults automatically saved to '" + path + "'")
}

// 0.2     Recording functions end
//
////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////
//
// 0.3     Common functions

// 0.3.1   Image shift (region copy)
RealImage ImageShift(RealImage in,Number x,Number y){
   Number w,h
   GetSize(in,w,h)
   RealImage out := RealImage("",4,w,h)

   out[y>0 ? y:0, x>0 ? x:0, y>0 ? h:h+y, x>0 ? w:w+x] \
         = in[y>0 ? 0:-y, x>0 ? 0:-x, y>0 ? h-y:h, x>0 ? w-x:w]

   Return out
}
// 0.3.2   Image region (region copy)
RealImage ImageRegion(RealImage in,Number l,Number t,Number w,Number h) {
   Number w0,h0
   GetSize(in,w0,h0)
   RealImage out := RealImage("",4,w,h)

   out[ t>0 ? 0:-t , l>0 ? 0:-l , t+h<=h0 ? h:h0-t , l+w<=w0 ? w:w0-l]\
         = in[ t>0 ? t:0 , l>0 ? l:0 , t+h<=h0 ? t+h:h0 , l+w<=w0 ? l+w:w0 ]

   Return out
}

// 0.3     Common functions end
//
////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////
//
// 1.1     AMMPSF child function
```

Appendix B. Program scripts

```
Number AMMPSF(String aperture_path,Number nummeas)
{
// Batch processing version adapted by Tianyu Liu
// For the original version with explanatory comments please refer to
"http://www.philipnakashima.com/"
//
//  Aperture Method for Measuring Point Spread Functions (AMMPSF)
//
// by Philip Nakashima  21/08/2001
//
// This script calculates both the modulation transfer function and point spread function for a CCD
detector
// from the image of a focussed aperture of ANY SHAPE (ie. need not be circular).  It is is fully
equivalent to
// the C++ version included in this distribution.
//
// Last Modified 21th October 2012.

   number
xrange,yrange,limit,shifts,squares,pointx,pointy,dim,delta,pixelone,pixeltwo,pixelthree,pixelfour,\
   pixelzero,dispx,dispy,dist,storedgof,storedit,gof,otfmin,sums,currentdim,numstd,storednumstd,rep
y,repx,ang
   Number max

   StartLog2("Welcome to AMMPSF, a script that automates the accurate measurement of
instrumental point spread functions from an aperture image of arbitrary shape.")

   Image aperture = OpenImage(aperture_path)
   GetSize(aperture,xrange,yrange)

   Image shapedhat := RealImage("ShapedHat",4,xrange,yrange)
   Image smoothap := RealImage("smoothap",4,xrange,yrange)
   Image avepsf := RealImage("AvePSF",4,xrange,yrange)
   Image storedshapedhat := RealImage("storedshapedhat",4,xrange,yrange)
   Image normaliser := RealImage("normaliser",4,xrange,yrange)
   Image psf := RealImage("psf",4,xrange,yrange)
   Image realotf := RealImage("realotf",4,xrange,yrange)
   Image new := RealImage("new",4,xrange,yrange)
   Image storedpsf := RealImage("storedpsf",4,xrange,yrange)
   Image hat := RealImage("hat",4,xrange,yrange)
   Image weights := RealImage("weights",4,xrange,yrange)
   Image rotavepsf := RealImage("rotavepsf",4,xrange,yrange)
   Image revpsf := RealImage("revrotavepsf",4,xrange,yrange)
   Image secondrotavepsf := RealImage("secondrotavepsf",4,xrange,yrange)
   Image rot := RealImage("rot",4,xrange,yrange)
   Image psfcore := RealImage("psfcore",4,xrange,yrange)
   Image core := RealImage("core",4,xrange,yrange)
   Image bigcore := RealImage("bigcore",4,xrange,yrange)
   Image deconvedap := RealImage("deconvedap",4,xrange,yrange)
   Image data := RealImage("data",4,xrange,yrange)
   Image mtf := RealImage("mtf",4,xrange,yrange)
   ComplexImage otf := ComplexImage("otf",8,xrange,yrange)
   ComplexImage finalotf := ComplexImage("finalotf",8,xrange,yrange)
   Image decvlapt := RealImage("Deconvoluted Aperture for Dark Noise
Measurement",4,xrange,yrange)

   delta = 1e-30

   core = Sqrt((icol - xrange/2)**2 + (irow - yrange/2)**2)
   core = 1 - (((core - 4.99) + Abs(core - 4.99))/(2*Abs(core - 4.99) + 1e-30))
   bigcore = Sqrt((icol - xrange/2)**2 + (irow - yrange/2)**2)
   bigcore = 1 - (((bigcore - 24.99) + Abs(bigcore - 24.99))/(2*Abs(bigcore - 24.99) + 1e-30))
   normaliser = 1
   for (ang = 1 ; ang <= 16 ; ang = ang + 1){
      If(abort) Return -1//Stop processing if "Stop" pressed
      rot = warp(normaliser,(icol-xrange/2)*cos((360/(2**ang))*Pi()/180)+(irow-
yrange/2)*sin((360/(2**ang))*Pi()/180)+xrange/2,\
         -(icol-xrange/2)*sin((360/(2**ang))*Pi()/180)+(irow-
yrange/2)*cos((360/(2**ang))*Pi()/180)+yrange/2)
      normaliser = normaliser + rot
      revpsf = warp(normaliser,xrange-icol,irow)
      normaliser = normaliser + revpsf
      revpsf = warp(normaliser,icol,yrange - irow)
      normaliser = normaliser + revpsf
   }
   normaliser = normaliser/Max(normaliser)
   while(sums < nummeas){
      If(abort) Return -1//Stop processing if "Stop" pressed
      dim = 1
      storedgof = 1e25
      storednumstd = 100
      gof = 1e30
      {
      number iteration = 0
      while (dim <= 13){
         If(abort) Return -1//Stop processing if "Stop" pressed
         iteration = iteration + 1
         limit = Max(shapedhat)/2
         if (dim == 1){
            data = aperture
            shapedhat = aperture
            limit = Max(shapedhat)*(1.5 + 0.5*(2*UniformRandom() - 1))/3
         }
         Log2("thresholding intensity for iteration "+iteration+" = "+limit+"\n")
         ClearImage(shapedhat)
         ClearImage(psf)
         shifts = (dim)**2
         squares = dim - 1
         {
         for (number step = 1 ; step <= shifts ; step = step + 1){
            If(abort) Return -1//Stop processing if "Stop" pressed
            pointx = (step-1)-((dim)*Trunc((step-1)/(dim)))
            pointy = Trunc((step-1)/(dim))
            dispx = ((squares/(2*(squares + delta)))-pointx/((squares + delta)))
            dispy = ((squares/(2*(squares + delta)))-pointy/((squares + delta)))
            hat = warp(data,icol + dispx,irow + dispy)
            hat = hat-limit
            hat = (hat + Abs(hat))/(2*Abs(hat)+delta)
            shapedhat = shapedhat + hat
         }
         }
         If(abort) Return -1//Stop processing if "Stop" pressed
         shapedhat = shapedhat*Sum(aperture)/Sum(shapedhat)
         psf = RealIFFT(RealFFT(aperture)/RealFFT(shapedhat))
         psf = psf/Sum(psf)
         ShiftCenter(psf)
         rotavepsf = psf
         for (ang = 1 ; ang <= 2 ; ang = ang + 1){
            If(abort) Return -1//Stop processing if "Stop" pressed
            rot = warp(rotavepsf,(icol-xrange/2)*cos((360/(2**ang))*Pi()/180)+(irow-
yrange/2)*sin((360/(2**ang))*Pi()/180)+xrange/2,\
            -(icol-xrange/2)*sin((360/(2**ang))*Pi()/180)+(irow-
yrange/2)*cos((360/(2**ang))*Pi()/180)+yrange/2)
            rotavepsf = rotavepsf + rot
            revpsf = warp(rotavepsf,xrange-icol,irow)
            rotavepsf = rotavepsf + revpsf
            revpsf = warp(rotavepsf,icol,yrange - irow)
            rotavepsf = rotavepsf + revpsf
         }
         If(abort) Return -1//Stop processing if "Stop" pressed
         secondrotavepsf = rotavepsf
         for (ang = 3 ; ang <= 16 ; ang = ang + 1){
            If(abort) Return -1//Stop processing if "Stop" pressed
            rot = warp(secondrotavepsf,(icol-xrange/2)*cos((360/(2**ang))*Pi()/180)+(irow-
yrange/2)*sin((360/(2**ang))*Pi()/180)+xrange/2,\
            -(icol-xrange/2)*sin((360/(2**ang))*Pi()/180)+(irow-
yrange/2)*cos((360/(2**ang))*Pi()/180)+yrange/2)
            secondrotavepsf = secondrotavepsf + rot
            revpsf = warp(secondrotavepsf,xrange-icol,irow)
            secondrotavepsf = secondrotavepsf + revpsf
            revpsf = warp(secondrotavepsf,icol,yrange - irow)
            secondrotavepsf = secondrotavepsf + revpsf
         }
         If(abort) Return -1//Stop processing if "Stop" pressed
         secondrotavepsf = secondrotavepsf/normaliser
         secondrotavepsf = secondrotavepsf/Max(secondrotavepsf)
         rotavepsf = rotavepsf/Max(rotavepsf)
         psfcore = rotavepsf*core + (1 - core)*secondrotavepsf
         ShiftCenter(psf)
         realotf = Modulus(RealFFT(psf))
         for (ang = 1 ; ang <= 16 ; ang = ang + 1){
            If(abort) Return -1//Stop processing if "Stop" pressed
            rot = warp(realotf,(icol-xrange/2)*cos((360/(2**ang))*Pi()/180)+(irow-
yrange/2)*sin((360/(2**ang))*Pi()/180)+xrange/2,\
            -(icol-xrange/2)*sin((360/(2**ang))*Pi()/180)+(irow-
yrange/2)*cos((360/(2**ang))*Pi()/180)+yrange/2)
            realotf = realotf + rot
            revpsf = warp(realotf,xrange-icol,irow)
            realotf = realotf + revpsf
            revpsf = warp(realotf,icol,yrange - irow)
            realotf = realotf + revpsf
         }
         If(abort) Return -1//Stop processing if "Stop" pressed
         realotf = realotf/normaliser
         realotf = realotf/Max(realotf)
         psf = RealIFFT(Complex(realotf,0))
         psf = psf/Sum(psf)
         ShiftCenter(psf)
         rotavepsf = psf
         for (ang = 1 ; ang <= 16 ; ang = ang + 1){
            If(abort) Return -1//Stop processing if "Stop" pressed
            rot = warp(rotavepsf,(icol-xrange/2)*cos((360/(2**ang))*Pi()/180)+(irow-
yrange/2)*sin((360/(2**ang))*Pi()/180)+xrange/2,\
            -(icol-xrange/2)*sin((360/(2**ang))*Pi()/180)+(irow-
yrange/2)*cos((360/(2**ang))*Pi()/180)+yrange/2)
            rotavepsf = rotavepsf + rot
            revpsf = warp(rotavepsf,xrange-icol,irow)
            rotavepsf = rotavepsf + revpsf
            revpsf = warp(rotavepsf,icol,yrange - irow)
            rotavepsf = rotavepsf + revpsf
         }
         If(abort) Return -1//Stop processing if "Stop" pressed
         rotavepsf = rotavepsf/normaliser
         rotavepsf = rotavepsf/Max(rotavepsf)
         psfcore = psfcore/Max(psfcore)
         psf = bigcore*psfcore + (1 - bigcore)*rotavepsf
         if (dim == 1)
            SetPixel(psf,xrange/2,yrange/2,2)
         psf = psf/Sum(psf)
         ShiftCenter(psf)
         otf = RealFFT(psf)
         otfmin = Min(Real(otf))
         Log2("otfmin = "+otfmin+"\n")
         currentdim = dim
         dim = -1
         if ((otfmin >= -1e30) && (otfmin < 0)){
            otf = otf + 2*Abs(otfmin)
            dim = currentdim - 2
            otf = otf/Max(Real(otf))
            psf = RealIFFT(otf)
         }
         if (otfmin > 0)
            dim = currentdim
         deconvedap = RealIFFT(RealFFT(aperture)/otf)
         data = deconvedap
         Log2("The first five pixels at integer distance from the centre are:\n")
         max = Max(psf)
         For(Number i = 1; i <= 5; i++)
```

Appendix B. Program scripts

```
            Log2(" " + GetPixel(psf,i,0)/max)
        smoothap = RealIFFT(RealFFT(shapedhat)*otf)
        weights = Abs(1 - Abs(smoothap - (Max(smoothap)/2))/(Max(smoothap)/2))
        gof = Sum(weights*((smoothap - aperture)**2))/Sum(weights)
        numstd = Sqrt(gof/(Sum(weights*aperture)/Sum(weights)))
        Log2("\ngof = "+gof+" Number of Poisson sigmas mismatch = "+numstd+"\n")
        if ((gof < storedgof) && (numstd <= 50)){
           storedit = iteration
           storedgof = gof
           storednumstd = numstd
           storedpsf = psf
           storedshapedhat = shapedhat
        }
        dim = dim + 2
        Log2("dim = "+dim+"\n")
        mainpanel_win.DLGSetProgress("SubProgress",(dim + 13*sums)/(13*nummeas))    //Set
progress bar
     }
     }
     If(abort) Return -1//Stop processing if "Stop" pressed
     shapedhat = storedshapedhat
     psf = storedpsf
     otf = RealFFT(psf)
     deconvedap = RealIFFT(RealFFT(aperture)/otf)
     psf = psf/Max(psf)
     ShiftCenter(psf)
     mtf = Modulus(otf)
     Log2("The running average intensities so far:\n")
     max = Max(avepsf)
     For(Number i = 1; i <= 5; i++)
        Log2(" " + GetPixel(avepsf,xrange/2 + i,yrange/2)/max)
     Log2("\nThe goodness of fit factor of "+storedgof+" was reached at iteration "+storedit+".\n")
     if (storednumstd <= 25){
        avepsf = avepsf + psf/Max(psf)
        sums = sums + 1
     }
     Log2("The number of summed psfs in the average = "+sums+", " + (nummeas-sums-1) +" more
to go\n")
   }
   If(abort) Return -1//Stop processing if "Stop" pressed
   mainpanel_win.DLGSetProgress("SubProgress",0.99)    //Set progress bar
   //Final finishing
   max = Max(avepsf)
   For(Number i = 1; i <= 5; i++)
      Log("" + GetPixel(avepsf,i,0)/max + " ")
   Image psf0 = avepsf
   avepsf = avepsf/Sum(avepsf)
   ShiftCenter(avepsf)
   finalotf = RealFFT(avepsf)
   decvlapt = RealIFFT(RealFFT(aperture)/finalotf) //Deconvoluted Aperture
   String folder_dir = m_dir + "PSF\\" + PathExtractBaseName(aperture_path,0) + "\\"
   CreateDirectory(folder_dir)
   SaveAsGatan3(psf0,folder_dir + "psf")
   SaveAsGatan3(avepsf,folder_dir + "NormalisedPSF")
   SaveAsGatan3(shapedhat,folder_dir + "ShapedHat")
   SaveAsGatan3(decvlapt,folder_dir + "DeconvolutedAperture")
   FinishLog2(folder_dir + "AMMPSF_log.txt")
   mainpanel_win.DLGSetProgress("SubProgress",1) //Set progress bar

   Return 0
}

// 1.1     AMMPSF child function ends
//
////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////
//
// 1.2     Noise Characterisation child function

Number NoiseChar(String specimen_path,String dir,Number darknoise0,Number maskopt0,\
     Number gainnormalmax0,Number numslice0,Number binning0,Number txt_opt)
{
   // Noise as a function of signal measurement script.
   //
   // Note: If you are operating with a difference image (A - B), then specimeni = A + B.
   //
   // Philip N.H. Nakashima, July 1 2008 (Bye bye Golf Sniff!)
   // Last Modified 12th September 2011 - correct physical model for noise ie: sigma = a.I + b.Sqrt(I)
+ c
   //
   // Adapted for batch pocessing by Tianyu Liu; correct "noise mask" edge-stroking.

   Number xrange,yrange,cyclex,cycley,numslice,intensity,loop,count,aveint,avestdev,sumpoints,\
   ay,bee,cee,storeday,storedbee,storedcee,gof,storedgof,step,sincechange,shot,cuts,higheststdev,\
   scale1,scale2,smoothloop,stdev,expfacx,expfacy,interval,intrange,maxx,maxy,\
   totalpts,a,b,change,times,topdynrange,lowdynrange,inputave,inputstdev,cycle,turn,numpix,\
   storednumpix,binfactor,topdyn,sigref,lowdyn,lasttopdyn,lastlowdyn,highestsig,hfstdev,percentdev,
\
   lastpercentdev,hfdev,hfmean,refdev,lastay,lastcee,swintb,sw,swint2b,swsigintb,\
   swsig,pert,dynrangepct,its,high,xhigh,yhigh,rank,pos,level,lowint,highint,tolerance,error,\
   firstay,firstbee,firstcee,mark,counter,normalisation,infofract,lim,darknoise,laststdev,\
   avelnint,avelnsig,sxx,syy,sxy,radius,numstds,storeddev,devchange,bins,gainnormalmax,\
   hfchange,lastnorm,hf,reduction,storedave,ave,storedchange,storedred,ordercycle,coeff,\
   order,refhf,lastave,stopave,testave,maskopt,binning,xbinned,ybinned,xmax,ymax,highest,intval,\
   noiseval,weightval,lastthresh,highthresh,lowthresh,intbin,sigbin,wtsbin,xcoord,remainder,maxwei
ght

   darknoise = darknoise0
   maskopt = maskopt0
   gainnormalmax = gainnormalmax0
   numslice = numslice0
   binning = binning0
   If(abort) Return -1//Stop processing if "Stop" pressed
   StartLog2("Noise Results")

   Image specimen := OpenImage(specimen_path)
   GetSize(specimen,xrange,yrange)
   Image mask := RealImage("mask",4,xrange,yrange)
   Image zeroth := RealImage("zeroth",4,xrange,yrange)
   Image highfrequency := RealImage("highfrequency",4,xrange,yrange)
   Image sigma := RealImage("sigma",4,xrange,yrange)

   Image smeared := RealImage("smeared",4,xrange,yrange)
   Image starting := RealImage("starting",4,xrange,yrange)
   Image interppsf := RealImage("psf",4,xrange,yrange)
   ComplexImage interpotf := ComplexImage("interpotf",8,xrange,yrange)
   ComplexImage fftspecimeni := ComplexImage("fftspecimeni",8,xrange,yrange)

   If (maskopt == 0)
      mask = 1
   Else
      mask := OpenImage(nc_mask_path)
   zeroth[4,4,yrange - 4,xrange - 4] = 1
   mask *= zeroth
   mask = mask * (specimen > 0 ? 1 : 0)
   If(maskopt == 0)
      mask = mask*(specimen > gainnormalmax ? 0 : 1)

   ClearImage(zeroth)
   a = 0.5
   b = a**2
   SetPixel(interppsf,xrange/2,yrange/2,1)
   SetPixel(interppsf,xrange/2 - 1,yrange/2,a)
   SetPixel(interppsf,xrange/2 + 1,yrange/2,a)
   SetPixel(interppsf,xrange/2,yrange/2 - 1,a)
   SetPixel(interppsf,xrange/2,yrange/2 + 1,a)
   SetPixel(interppsf,xrange/2 + 1,yrange/2 - 1,b)
   SetPixel(interppsf,xrange/2 + 1,yrange/2 + 1,b)
   SetPixel(interppsf,xrange/2-1,yrange/2 - 1,b)
   SetPixel(interppsf,xrange/2-1,yrange/2 + 1,b)
   interppsf = interppsf/Sum(interppsf)
   ShiftCenter(interppsf)
   interpotf = (RealFFT(interppsf))
   fftspecimeni = RealFFT(specimen)
   ShiftCenter(fftspecimeni)
   change = 100
   reduction = 1
   storedave = 1e+30
   storedchange = 1e+30
   reduction = 2
   ComplexImage cutfft := ComplexImage("cutfft",8,xrange/reduction,yrange/reduction)
   Image noise := RealImage("noise",4,xrange/reduction,yrange/reduction)
   Image minimask := RealImage("minimask",4,xrange/reduction,yrange/reduction)
   Image modcutfft := RealImage("modcutfft",4,xrange/reduction,yrange/reduction)
   minimask = Warp(mask,icol*reduction,irow*reduction)
   cutfft = Warp(fftspecimeni,icol + (xrange/2) - (xrange/(2*reduction)),irow + (yrange/2) -
(yrange/(2*reduction)))
   ShiftCenter(cutfft)
   noise = RealIFFT(cutfft)/reduction
   refhf = Sqrt(Sum(minimask*(noise)**2)/Sum(minimask))
   starting = specimen
   order = 4
   coeff = order
   for (ordercycle = 1 ; ordercycle <= order ; ordercycle++) {
      If(abort) Return -1//Stop processing if "Stop" pressed
      smeared = RealIFFT(RealFFT(starting)*interpotf)
      zeroth = zeroth + coeff*smeared
      starting = smeared

      coeff = coeff*(ordercycle - order)/(ordercycle + 1)
   }
   highfrequency = specimen - zeroth
   ave = Sum(mask*highfrequency)/Sum(mask)
   hf = Sqrt(Sum(mask*(highfrequency - ave)**2)/Sum(mask))
   normalisation = refhf/hf
   Log2(""+hf+" "+normalisation+"\n")
   Log2("Output stdev = "+(hf*normalisation)+"\n")
   highfrequency = normalisation*highfrequency
   mask = mask*(zeroth > 0 ? 1:0)
   Image longint := RealImage("longint",4,numslice,1)
   Image longweights := RealImage("longweights",4,numslice,1)
   Image longsig := RealImage("longsig",4,numslice,1)
   Image binnedint := RealImage("binnedint",4,xrange/binning,yrange/binning)
   Image binnedsig := RealImage("binnedsig",4,xrange/binning,yrange/binning)
   Image binnedwts := RealImage("binnedwts",4,xrange/binning,yrange/binning)
   Image slice := RealImage("slice",4,xrange/binning,yrange/binning)
   binnedint = Warp(zeroth,icol*binning,irow*binning)
   binnedsig = Warp(highfrequency,icol*binning,irow*binning)
   binnedwts = Warp(mask,icol*binning,irow*binning)
   interval = (Max(binnedint*binnedwts) - Min(binnedint*binnedwts))/numslice
   maxweight = 0
   for (loop = 0 ; loop < numslice ; loop++) {
      If(abort) Return -1//Stop processing if "Stop" pressed
      highthresh = Max(binnedint) - loop*interval
      lowthresh = highthresh - interval
      slice = binnedint > lowthresh ? 1:0
      slice = slice - (binnedint > highthresh ? 1:0)
      intval = Sum(slice*binnedwts*binnedint)/(Sum(slice*binnedwts) + 1e-30)
      noiseval = Sqrt(Sum(slice*binnedwts*(binnedsig**2))/(Sum(slice*binnedwts) + 1e-30))
      weightval = Sum(slice*binnedwts)
      SetPixel(longint,loop,0,intval)
      SetPixel(longsig,loop,0,noiseval)
      SetPixel(longweights,loop,0,weightval)
      if (Mod(loop,100) == 1)
         Log2(""+intval+"  "+noiseval+"  "+weightval+"\n")
```

Appendix B. Program scripts

```
    if (weightval >= maxweight)
      maxweight = weightval
    mainpanel_win.DLGSetProgress("SubProgress",(loop + 1)/numslice/5)  //Set progress bar
  }
  Log2("\nStart ordering in terms of sigma\n\n")
  Image shortweights := RealImage("shortweights",4,numslice,1)
  Image shortsig := RealImage("shortsig",4,numslice,1)
  Image shortint := RealImage("shortint",4,numslice,1)
  loop = 0
  xcoord = 0
  remainder = Sum(longweights)
  while (remainder > maxweight) {
    sigbin = 0
    wtsbin = 0
    intbin = 0
    while (wtsbin < maxweight) {
      If(abort) Return -1//Stop processing if "Stop" pressed
      wtsbin = wtsbin + GetPixel(longweights,loop,0)
      sigbin = sigbin + GetPixel(longweights,loop,0)*(GetPixel(longsig,loop,0))**2
      intbin = intbin + GetPixel(longweights,loop,0)*GetPixel(longint,loop,0)
      if (wtsbin >= maxweight) {
        intbin = intbin/wtsbin
        sigbin = Sqrt(sigbin/wtsbin)
        SetPixel(shortint,xcoord,0,intbin)
        SetPixel(shortsig,xcoord,0,sigbin)
        SetPixel(shortweights,xcoord,0,wtsbin)
        Log2(""+intbin+" "+sigbin+" "+wtsbin+"\n")
        remainder = remainder - wtsbin
      }
      loop = loop + 1
    }
    xcoord = xcoord + 1
    mainpanel_win.DLGSetProgress("SubProgress",maxweight/remainder/5 + 0.2) //Set progress
bar
  }
  sigbin = 0
  wtsbin = 0
  intbin = 0
  while (loop < numslice) {
    If(abort) Return -1//Stop processing if "Stop" pressed
    wtsbin = wtsbin + GetPixel(longweights,loop,0)
    sigbin = sigbin + GetPixel(longweights,loop,0)*(GetPixel(longsig,loop,0))**2
    intbin = intbin + GetPixel(longweights,loop,0)*GetPixel(longint,loop,0)

    loop = loop + 1
    mainpanel_win.DLGSetProgress("SubProgress",(loop + 1)/numslice/5 + 0.4) //Set progress bar
  }
  intbin = intbin/wtsbin
  sigbin = Sqrt(sigbin/wtsbin)
  SetPixel(shortint,xcoord,0,intbin)
  SetPixel(shortsig,xcoord,0,sigbin)
  SetPixel(shortweights,xcoord,0,wtsbin)
  Log2(""+intbin+" "+sigbin+" "+wtsbin+"\n")
  Image weights := RealImage("weights",4,xcoord,1)
  Image sig := RealImage("sig",4,xcoord,1)
  Image int := RealImage("int",4,xcoord,1)
  Image intfunc := RealImage("intfunc",4,xcoord,1)
  ClearImage(int)
  ClearImage(weights)
  ClearImage(sig)
  ClearImage(intfunc)
  for (loop = 0 ; loop < xcoord ; loop++) {
    If(abort) Return -1//Stop processing if "Stop" pressed
    SetPixel(int,loop,0,GetPixel(shortint,loop,0))
    SetPixel(sig,loop,0,GetPixel(shortsig,loop,0))
    SetPixel(weights,loop,0,GetPixel(shortweights,loop,0))
    mainpanel_win.DLGSetProgress("SubProgress",(loop + 1)/xcoord/5 + 0.6)//Set progress bar
  }
  ay = 1
  bee = 1
  cee = 1
  if (darknoise > 0)
    cee = darknoise
  storeday = ay
  storedcee = cee
  storedbee = bee
  storedgof = 1e+20
  step = 1
  sincechange = 1
  //weights = 1
  swsig = Sum(weights*sig)
  sw = Sum(weights)
  while (sincechange < 1000) {
    swintb = Sum(weights*(int + bee*Sqrt(int)))
    swsigintb = Sum(weights*sig*(int + bee*Sqrt(int)))
    swint2b = Sum(weights*(int + bee*Sqrt(int))**2)
    pert = 1e+20
    its = 0
    While ((pert >= 0.00001) && (its < 1000)) {
      If(abort) Return -1//Stop processing if "Stop" pressed
      lastay = ay
      lastcee = cee
      ay = Abs((swsigintb - (cee*swintb))/swint2b)
      if (darknoise < 0)
        cee = Abs((swsig - (ay*swintb))/sw)
      pert = (Abs((lastay - ay)/lastay) + Abs((lastcee - cee)/lastcee))/2
      its = its + 1
    }
    intfunc = ay*(int + bee*Sqrt(int)) + cee
    gof = Sum(weights*(sig - intfunc)**2)/(Sum(weights) - (3*Sum(weights)/xcoord))
    if (gof < storedgof) {
      storedgof = gof
      sincechange = 0
      storeday = ay
      storedbee = bee
      storedcee = cee
    }
    bee = Abs(storedbee + ((2*uniformRandom() - 1)**1)/((Mod(step,10) + 1)**2))
    sincechange = sincechange + 1
    step = step + 1
    ay = storeday
    cee = storedcee
    if (Mod(step,100) == 1)
      Log2("It. "+step+", gof = "+storedgof+" a*I + b*Sqrt(I) + c: a = "+storeday+" b =
"+(storedbee*storeday)+" c = "+storedcee+"\n")
    mainpanel_win.DLGSetProgress("SubProgress",sincechange/1000/5 + 0.8)  //Set progress bar
  }
  If(abort) Return -1//Stop processing if "Stop" pressed
  mainpanel_win.DLGSetProgress("SubProgress",0.99)  //Set progress bar
  Log2("Final results: "+storeday+"*I + "+(storedbee*storeday)+"*Sqrt(I) + "+storedcee+"\n")
  intfunc = storeday*int + (storedbee*storeday)*Sqrt(int) + storedcee
  Log2("Final output: Iexpt, sigma, calc sigma\n")
  for (cycle = 0 ; cycle < xcoord ; cycle++)
    Log2(""+GetPixel(int,cycle,0)+""+GetPixel(sig,cycle,0)+"   "+GetPixel(intfunc,cycle,0)+"\n")
  sigma = storeday*(Abs(specimen) + storedbee*Sqrt(Abs(specimen))) + storedcee
  // Save output
  SaveAsGatan3(zeroth,dir + "zeroth")
  SaveAsGatan3(sigma,dir + "sigma")
  SaveAsGatan3(mask,dir + "NoiseMask")
  SaveAsGatan3(highfrequency,dir + "highfrequency")
  If(txt_opt == 1) {
    SaveAsText(sigma,dir + "sigma")
    SaveAsText(zeroth,dir + "pattern")
  }
  FinishLog2(dir + "NoiseResults.txt")
  mainpanel_win.DLGSetProgress("SubProgress",1) //Set progress bar

  Return 0
}

// 1.2    Noise Characterisation child function ends
//
////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////
//
// 1.3    IGQCBED child functions

// 1.3.1  Draw Mask step child function
String DrawMask(String dir,String specimen_name,Number numbeams,Number dia,Number ratio)
{
  Image ref
  If(DoesFileExist(dir + "zeroth.dm3"))
    ref := OpenImage(dir + "zeroth.dm3")
  Else
    If(DoesFileExist(dir + "sigma.dm3"))
      ref := OpenImage(dir + "sigma.dm3")
    Else
      If(DoesFileExist(dir + "_PSFCorrected.dm3"))
        ref := OpenImage(dir + "_PSFCorrected.dm3")
      Else
        If(DoesFileExist(dir + specimen_name + ".dm3"))
          ref := OpenImage(dir + specimen_name + ".dm3")
        Else
          Return "No full-size image found (for size reference)!"

  Number xrange,yrange,i,x,y,ra = dia/2
  GetSize(ref,xrange,yrange)

  If(!DoesFileExist(dir + "centres.dm3"))
    Return "Input file 'centres.dm3' unfound"
  Image centres := OpenImage(dir + "centres.dm3")

  Image mask := RealImage("mask",4,xrange,yrange)
  Image beam_mask := RealImage("beam mask",4,dia,dia)
  beam_mask = ra*ratio > iradius ? 1 : 0

  For(i = 0; i < numbeams; i++)
  {
    x = GetPixel(centres,0,i)
    y = GetPixel(centres,1,i)
    Number t,l,b,r
    t = y - ra < 0 ? 0 : y - ra
    l = x - ra < 0 ? 0 : x - ra
    b = y + ra > yrange ? yrange : y + ra
    r = x + ra > xrange ? xrange : x + ra
    mask[t,l,b,r] = beam_mask[t - (y - ra),l - (x - ra),dia - ((y + ra) - b),dia - ((x + ra) - r)]
  }

  SaveAsGatan3(mask,dir + "mask")
  Return "mask.dm3"
}
// 1.3.2  Pattern coordinates function
String PatternCoords(Image centres,Number dia,Number binning)
{
  Number point1x,point1y,point2x,point2y,point3x,point3y,point4x,point4y,\
  AAA,BBB,vec1x,vec1y,vec2x,vec2y,zonex,zoney
  String coords

  point4x = GetPixel(centres,0,0)
  point4y = GetPixel(centres,1,0)
  zonex = GetPixel(centres,2,0)
  zoney = GetPixel(centres,3,0)

  point4x = point4x - zonex
  point4y = point4y - zoney
  vec1x = GetPixel(centres,2,1) - zonex
```

Appendix B. Program scripts

```
    vec1y = GetPixel(centres,3,1) - zoney
    vec2x = GetPixel(centres,2,2) - zonex
    vec2y = GetPixel(centres,3,2) - zoney

    point1x = point4x - ((dia - binning)/2)
    point1y = point4y - ((dia - binning)/2)
    point2x = point4x + ((dia - binning)/2)
    point2y = point4y - ((dia - binning)/2)
    point3x = point4x - ((dia - binning)/2)
    point3y = point4y + ((dia - binning)/2)
    If(abort) Return "Process canceled"    //Stop processing if "Stop" pressed

    AAA = ((point1x*vec2y) - (point1y*vec2x))/((vec1x*vec2y) - (vec1y*vec2x))
    BBB = (point1y/vec2y) - ((vec1y/vec2y)*AAA)
    coords = ""+AAA+"  "+BBB+" \n"

    AAA = ((point2x*vec2y) - (point2y*vec2x))/((vec1x*vec2y) - (vec1y*vec2x))
    BBB = (point2y/vec2y) - ((vec1y/vec2y)*AAA)
    coords += ""+AAA+"  "+BBB+" \n"

    AAA = ((point3x*vec2y) - (point3y*vec2x))/((vec1x*vec2y) - (vec1y*vec2x))
    BBB = (point3y/vec2y) - ((vec1y/vec2y)*AAA)
    coords += ""+AAA+"  "+BBB+" \n"

    /* The 4th pair of pattern coordinates is no longer used by QCBEDMS after 2017 */
    //AAA = ((point4x*vec2y) - (point4y*vec2x))/((vec1x*vec2y) - (vec1y*vec2x))
    //BBB = (point4y/vec2y) - ((vec1y/vec2y)*AAA)
    //coords += ""+AAA+"  "+BBB+" \n"

    If(abort) Return "Process canceled"    //Stop processing if "Stop" pressed
    Return coords
}
// 1.3.3   IGQCBED step child function
String IGQCBED(String dir,Number numbeams,Number dia,Number binning,\
        Number igqcbed_opt,Number expansionfactor,Number noisemask_opt,Number txt_opt)
{
    If(!DoesFileExist(dir + "mask.dm3"))
        Return "Input file 'mask.dm3' unfound!"
    If(noisemask_opt == 1 && !DoesFileExist(dir + "NoiseMask.dm3"))
        Return "Input file 'NoiseMask.dm3' unfound!"
    If(!DoesFileExist(dir + "zeroth.dm3"))
        Return "Input file 'zeroth.dm3' unfound!"
    If(!DoesFileExist(dir + "sigma.dm3"))
        Return "Input file 'sigma.dm3' unfound!"
    If(!DoesFileExist(dir + "centres.dm3"))
        Return "Input file 'centres.dm3' unfound!"

    Image mask          := OpenImage(dir + "mask.dm3")
    If(noisemask_opt == 1)
        mask *= OpenImage(dir + "NoiseMask.dm3")
    Image zeroth     := OpenImage(dir + "zeroth.dm3")
    Image sigmaimage    := OpenImage(dir + "sigma.dm3")
    Image centres       := OpenImage(dir + "centres.dm3")
    Image sigmaimage_2    = sigmaimage**2

    Number xrange,yrange,centx,centy,cycle,loopx,loopy
    GetSize(zeroth,xrange,yrange)
    Number diabe = dia*expansionfactor/binning
    Number diab = dia/binning

    Image gradientsum := RealImage("grandientsum",4,xrange,yrange)
    Image sigmasum := RealImage("sigmasum",4,xrange,yrange)
    Image masksum := RealImage("masksum",4,xrange,yrange)
    Image fullweights := RealImage("fullweights",4,xrange,yrange)
    Image selection := RealImage("selection",4,dia,dia)
    Image hugeselection := RealImage("hugeselection",4,dia*expansionfactor,dia*expansionfactor)
    Image largeselection := RealImage("largeselection",4,diabe,diabe)
    Image binnedselection := RealImage("binnedselection",4,dia,dia)
    Image outputdata := RealImage("outputdata",4,diabe,3*numbeams*diabe)
    Image finaloutput := RealImage("finaloutput",4,diab,3*numbeams*diab)
    Image temp //:= RealImage("temp",4,diab,numbeams*diab)
    Image jpg := RealImage("Finaloutput JPG",4,3*diab,numbeams*diab)

    If(abort) Return "Process canceled"    //Stop processing if "Stop" pressed
    gradientsum = ImageShift(zeroth,binning,0) + ImageShift(zeroth,-binning,0)
    sigmasum = ImageShift(sigmaimage_2,binning,0) + ImageShift(sigmaimage_2,-binning,0)
    masksum = ImageShift(mask,binning,0) + ImageShift(mask,-binning,0)

    gradientsum += ImageShift(zeroth,0,binning) + ImageShift(zeroth,0,-binning)
    sigmasum += ImageShift(sigmaimage_2,0,binning) + ImageShift(sigmaimage_2,0,-binning)
    masksum += ImageShift(mask,0,binning) + ImageShift(mask,0,-binning)

    gradientsum += ImageShift(zeroth,binning,binning) + ImageShift(zeroth,-binning,-binning)
    sigmasum += ImageShift(sigmaimage_2,binning,binning) + ImageShift(sigmaimage_2,-binning,-
binning)
    masksum += ImageShift(mask,binning,binning) + ImageShift(mask,-binning,-binning)

    gradientsum += ImageShift(zeroth,binning,-binning) + ImageShift(zeroth,-binning,binning)
    sigmasum += ImageShift(sigmaimage_2,binning,-binning) + ImageShift(sigmaimage_2,-
binning,binning)
    masksum += ImageShift(mask,binning,-binning) + ImageShift(mask,-binning,binning)

    If(abort) Return "Process canceled"    //Stop processing if "Stop" pressed
    gradientsum = (gradientsum/8) - zeroth
    masksum = masksum + mask
    sigmasum = Sqrt(sigmasum)/8
    sigmasum = Sqrt(sigmasum**2 + sigmaimage_2)

    fullweights = masksum > max(masksum) - 1 ? 1:0

    for (cycle = 1 ; cycle <= numbeams ; cycle++) {
        centx = GetPixel(centres,0,cycle - 1)
        centy = GetPixel(centres,1,cycle - 1)
        selection = ImageRegion(gradientsum,centx - dia/2,centy - dia/2,dia,dia)
        hugeselection = Warp(selection,Trunc(icol/expansionfactor),Trunc(irow/expansionfactor))

        ClearImage(largeselection)

        for (loopy = 1 ; loopy <= binning ; loopy++)
            for (loopx = 1 ; loopx <= binning ; loopx++) {
                If(abort) Return "Process canceled"    //Stop processing if "Stop" pressed
                largeselection = largeselection + Warp(hugeselection,icol*binning + (loopx -
1),irow*binning + (loopy - 1))
            }

        largeselection = (largeselection/(expansionfactor**2))/((binning**2)/(expansionfactor**2))
        outputdata[(cycle - 1)*diabe,0,cycle*diabe,diabe] = largeselection

        selection = ImageRegion(sigmasum,centx - dia/2,centy - dia/2,dia,dia)
        hugeselection = Warp(selection,Trunc(icol/expansionfactor),Trunc(irow/expansionfactor))

        ClearImage(largeselection)

        for (loopy = 1 ; loopy <= binning ; loopy++)
            for (loopx = 1 ; loopx <= binning ; loopx++) {
                If(abort) Return "Process canceled"    //Stop processing if "Stop" pressed
                largeselection = largeselection + (Warp(hugeselection,icol*binning + (loopx -
1),irow*binning + (loopy - 1)))**2
            }

        largeselection = Sqrt(largeselection/(expansionfactor**2))/((binning**2)/(expansionfactor**2))
        outputdata[(numbeams + cycle - 1)*diabe,0,(numbeams + cycle)*diabe,diabe] =
largeselection**2

        selection = ImageRegion(fullweights,centx - dia/2,centy - dia/2,dia,dia)
        hugeselection = Warp(selection,Trunc(icol/expansionfactor),Trunc(irow/expansionfactor))

        ClearImage(largeselection)

        for (loopy = 1 ; loopy <= binning ; loopy++)
            for (loopx = 1 ; loopx <= binning ; loopx++) {
                If(abort) Return "Process canceled"    //Stop processing if "Stop" pressed
                largeselection = largeselection + Warp(hugeselection,icol*binning + (loopx -
1),irow*binning + (loopy - 1))
            }

        largeselection = largeselection < Max(largeselection) ? 0 : 1
        outputdata[(2*numbeams + cycle - 1)*diabe,0,(2*numbeams + cycle)*diabe,diabe] =
largeselection
    }

    If(abort) Return "Process canceled"    //Stop processing if "Stop" pressed
    finaloutput = Warp(outputdata,icol*expansionfactor + Round((expansionfactor - 1)/2),\
            irow*expansionfactor + Round((expansionfactor - 1)/2))

    // Set 'zero' pixels in 'sigma' section of 'finaloutput' to 1, because 'sigma' will be divisor
    Image finalsigma := finaloutput[numbeams*diab,0,2*numbeams*diab,diab]
    finalsigma = finalsigma == 0 ? 1 : finalsigma

    If(abort) Return "Process canceled"    //Stop processing if "Stop" pressed
    // Generate high-contrast JPG overview
    jpg[0,0,numbeams*diab,diab] = finaloutput[0,0,numbeams*diab,diab]
    jpg[0,diab,numbeams*diab,2*diab] = finaloutput[numbeams*diab,0,2*numbeams*diab,diab]
    jpg[0,2*diab,numbeams*diab,3*diab] = finaloutput[2*numbeams*diab,0,3*numbeams*diab,diab]
    temp := jpg[0,0,numbeams*diab,diab]
    temp = temp > 0 ? 1:0
    temp := jpg[0,diab,numbeams*diab,2*diab]
    temp -= Min(temp)
    temp /= Max(temp)
    jpg = jpg > 1 ? 1:jpg

    // Crop 'weights' section and save separately
    temp := finaloutput[2*numbeams*diab,0,3*numbeams*diab,diab]

    If(abort) Return "Process canceled"    //Stop processing if "Stop" pressed
    SaveAsGatan3(finaloutput,dir + "intensities")
    SaveAsGatan3(temp,dir + "weights")
    ImageDocument jpg_doc = CreateImageDocument("JPG")
    ImageDocumentAddImage(jpg_doc,jpg)
    ImageDocumentSaveToFile(jpg_doc,"JPEG/JFIF Format",m_dir + "JPG\\Final_" + sub_name)

    // Generate 'Pattern Coordinates.txt'
    Number patterncoords_id = CreateFileForWriting(dir + "Pattern Coordinates.txt")
    Object patterncoords_stream = NewStreamFromFileReference(patterncoords_id,1)
    StreamWriteAsText(patterncoords_stream,0,PatternCoords(centres,dia,binning))
    CloseFile(patterncoords_id)

    // Save as Text Images
    If(txt_opt == 1) {
        If(abort) Return "Process canceled"    //Stop processing if "Stop" pressed
        SaveAsText(finaloutput,dir + "intensitydata")
        If(abort) Return "Process canceled"    //Stop processing if "Stop" pressed
        SaveAsText(temp,dir + "weights")

        // Generate ready-to-go 'in.dat' from 'intensitydata.txt'
        Number read_id = OpenFileForReading(dir + "intensitydata.txt")
        Object read_stream = NewStreamFromFileReference(read_id,1)
        String content = StreamReadAsText(read_stream,0,StreamGetSize(read_stream))
        CloseFile(read_id)

        Number write_id = CreateFileForWriting(dir + "in.dat")
        Object write_stream = NewStreamFromFileReference(write_id,1)
        StreamWriteAsText(write_stream,0,"" + diab + " " + diab + " " + igqcbed_opt + "\n")
        StreamWriteAsText(write_stream,0,content)
        CloseFile(write_id)
```

Appendix B. Program scripts

```
      Return "in.dat, weights.dat, intensitydata.txt, intensities.dm3, JPG, weights.dm3 & Pattern
Coordinates.txt"
   }

   If(abort) Return "Process canceled"    //Stop processing if "Stop" pressed
   Return "intensities.dm3, JPG, weights.dm3, & Pattern Coordinates.txt"
}
// 1.3.4   FOR007.DAT step child function
String FOR007_DAT(String dir,Number dia,Number binning)
{
   If(!DoesFileExist(dir + "centres.dm3"))
      Return "Input file 'centres.dm3' unfound!"
   If(!DoesFileExist(m_dir + "Misc\\FOR007_head.txt"))
      Return "Input file 'FOR007_head.txt' unfound in folder 'Misc'!"
   If(!DoesFileExist(m_dir + "Misc\\FOR007_tail.txt"))
      Return "Input file 'FOR007_tail.txt' unfound in folder 'Misc'!"

   Image centres := OpenImage(dir + "centres.dm3")
   Number head_id = OpenFileForReading(m_dir + "Misc\\FOR007_head.txt")
   Number tail_id = OpenFileForReading(m_dir + "Misc\\FOR007_tail.txt")
   Object head_stream = NewStreamFromFileReference(head_id,1)
   Object tail_stream = NewStreamFromFileReference(tail_id,1)
   String head_content = StreamReadAsText(head_stream,0,StreamGetSize(head_stream)) + " \n"
   String tail_content = StreamReadAsText(tail_stream,0,StreamGetSize(tail_stream)) + " \n"
   CloseFile(head_id)
   CloseFile(tail_id)

   Number for7_id = CreateFileForWriting(dir + "FOR007.DAT")
   Object for7_stream = NewStreamFromFileReference(for7_id,1)
   StreamWriteAsText(for7_stream,0,head_content + PatternCoords(centres,dia,binning) +
tail_content)
   CloseFile(for7_id)

   If(abort) Return "Process canceled"    //Stop processing if "Stop" pressed
   Return "FOR007.DAT"
}
// 1.3.5   parameters.dat step child function
String Parameters_dat(String dir,String specimen_name,Number numbeams,Number dia,\
      Number binning,Number igqcbed_opt,Number piecebin,Number refcycle,Number randstep)
{
   Image ref
   If(DoesFileExist(dir + "zeroth.dm3"))
      ref := OpenImage(dir + "zeroth.dm3")
   Else
      If(DoesFileExist(dir + "sigma.dm3"))
         ref := OpenImage(dir + "sigma.dm3")
      Else
         If(!DoesFileExist(dir + "mask.dm3"))
            ref := OpenImage(dir + "mask.dm3")
         Else
            If(DoesFileExist(dir + specimen_name + ".dm3"))
               ref := OpenImage(dir + specimen_name + ".dm3")
            Else
               If(DoesFileExist(dir + "_PSFCorrected.dm3"))
                  ref := OpenImage(dir + "_PSFCorrected.dm3")
               Else
                  Return "No full-size image found (for size reference)!"

   Number xrange,yrange,i,x,y
   GetSize(ref,xrange,yrange)

   If(!DoesFileExist(dir + "centres.dm3"))
      Return "Input file 'centres.dm3' unfound!"

   Image centres := OpenImage(dir + "centres.dm3")

   Number filepm_id = CreateFileForWriting(dir + "parameters.dat")
   Object filepm_stream = NewStreamFromFileReference(filepm_id,1)

   StreamWriteAsText(filepm_stream,0,"" + igqcbed_opt + " " + xrange + " " + yrange + " " + dia +
" " + dia + " ")
   StreamWriteAsText(filepm_stream,0,"" + dia/binning + " " + dia/binning + " " + binning + " " +
piecebin + " ")
   StreamWriteAsText(filepm_stream,0,"" + refcycle + " " + randstep + " \n")
   StreamWriteAsText(filepm_stream,0,"" + numbeams + " \n")

   For(i = 0; i < numbeams; ++i) {
      If(abort) Return "Process canceled"    //Stop processing if "Stop" pressed
      x = GetPixel(centres,0,i)
      y = GetPixel(centres,1,i)

      StreamWriteAsText(filepm_stream,0,"" + x + " " + y + " " + "0 \n")
   }

   CloseFile(filepm_id)

   If(abort) Return "Process canceled"    //Stop processing if "Stop" pressed
   Return "parameters.dat"
}

// 1.3      IGQCBED child functions end
//
////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////
//
// 2     Batch Processing looping functions

// 2.0.1   Update current paths function
Void UpdateCurrentPaths(Number tag_opt) {
   img_name = img_pre + noi + img_suf
   img_path = m_dir + img_name + ".dm3"
   sub_name = sub_pre + noi + sub_suf
   sub_dir = m_dir + sub_name + "\\"

   If(tag_opt == 1) {
      TagGroupSetTagAsString(settings,"m_dir",m_dir)
      TagGroupSetTagAsString(settings,"sub_dir",sub_dir)
      TagGroupSetTagAsString(settings,"img_path",img_path)
      TagGroupSetTagAsString(settings,"img_name",img_name)
      TagGroupSetTagAsString(settings,"sub_name",sub_name)
      TagGroupSetTagAsNumber(settings,"noi",noi)
      TagGroupSetTagAsString(settings,"exit_text",exit_text)
      TagGroupSetTagAsnumber(settings,"abort",abort)
   }
}
// 2.1      AMMPSF looping function
Number AMMPSF_Batch()
{
   Number num_apertures = 0, num_tags
   TagGroup one_file,  all_files = GetFilesInDirectory(m_dir + "PSF\\",1)
   String filename
   CreateDirectory(m_dir + "PSF\\Unwanted PSFs\\")

   StartLog("AMMPSF Measurement")
   Log("\nPSF measurement cycles = " + psf_nummeas)
   Log("\n\nBatch processing starts:")
   Log("\nAperture image: first 5 integer pixels of PSF curve (in ratio to max pixel)")

   num_tags = TagGroupCountTags(all_files)
   For(Number i = 0; i < num_tags; i++) {
      If(abort) Return -1//Stop processing if "Stop" pressed
      TagGroupGetIndexedTagAsTagGroup(all_files,i,one_file)
      TagGroupGetTagAsString(one_file,"Name",filename)
      If(PathExtractExtension(filename,0) == "dm3") {
         ++num_apertures
         Log("\n\t" + filename + ": ")
         AMMPSF(m_dir + "PSF\\" + filename,psf_nummeas)
         If(abort) Return -1//Stop processing if "Stop" pressed
      }

      mainpanel_win.DLGSetProgress("MainProgress",(i + 1)/num_tags)
   }
   mainpanel_win.DLGSetProgress("MainProgress",1)
   mainpanel_win.DLGSetProgress("SubProgress",1)
   Log("\n\n\n" + num_apertures + " aperture images processed.")

   FinishLog("AMMPSF")

   ShowAlert("" + num_apertures + " aperture images processed!",2)

   Return 0
}
// 2.2      PSF averaging function
Number PSFAvg_Batch()
{
   Number num_psfs = 0,num_apertures = 0, max, num_tags
   TagGroup one_tag, all_list = GetFilesInDirectory(m_dir + "PSF\\",2)
   TagGroup aperture_list = NewTagList()
   String folder_name,dir,file_name,aperture_path
   Image avgpsf,aperture

   CreateDirectory(m_dir + "PSF\\Averaged PSF\\")

   StartLog("PSFs Average")
   Log("\nAveraging starts:")
   Log("\nPSF file  |  first 5 integer pixels of PSF curve (in ratio to max pixel)")

   num_tags = TagGroupCountTags(all_list)
   For(Number i = 0; i < num_tags; i++) {
      If(abort) Return -1//Stop processing if "Stop" pressed
      TagGroupGetIndexedTagAsTagGroup(all_list,i,one_tag)
      TagGroupGetTagAsString(one_tag,"Name",folder_name)

      If(folder_name != "Averaged PSF" && folder_name != "Unwanted PSFs") {
         dir = m_dir + "PSF\\" + folder_name + "\\"
         file_name = ""
         If(DoesFileExist(dir + "NormalisedPSF.dm3"))
            file_name = "NormalisedPSF.dm3"
         Else If(DoesFileExist(dir + "psf.dm3"))
            file_name = "psf.dm3"

         If(file_name != "") {
            Image psf := OpenImage(dir + file_name)
            psf /= Max(psf)

            Log("\n" + ++num_psfs + ". " + folder_name + "\\" + file_name)
            For(Number p = 0; p < 5; p++)
               Log(", " + GetPixel(psf,p,0))

            If(abort) Return -1//Stop processing if "Stop" pressed
            If(ImageIsValid(avgpsf))
               avgpsf += psf
            Else
               avgpsf = psf

            TagGroupInsertTagAsString(aperture_list,Infinity(),folder_name) // Storage aperture
images
         }
      }
      mainpanel_win.DLGSetProgress("MainProgress",(i + 1)/num_tags/2)
   }
   If(abort) Return -1//Stop processing if "Stop" pressed
   avgpsf /= Sum(avgpsf)
   Log("\n\n" + num_psfs + " PSFs averaged over.")

   max = Max(avgpsf)
```

Appendix B. Program scripts

```
   Log("\n\nFirst 5 integer pixels of PSF curve (in ratio to max pixel):\n")
   For(Number p = 0; p < 5; p++)
      Log(GetPixel(avgpsf,p,0)/max + " ")

   SaveAsGatan3(avgpsf,m_dir + "PSF\\Averaged PSF\\AverageNormalisedPSF.dm3")
   Log("\nAverage PSF normalised and saved as '" + m_dir + "PSF\\Averaged
PSF\\AverageNormalisedPSF.dm3")

   Log("\n\n\nAperture image deconvolution: ")
   num_tags = TagGroupCountTags(aperture_list)
   For(Number i = 0; i < num_tags; i++) {
      If(abort) Return -1//Stop processing if "Stop" pressed
      TagGroupGetIndexedTagAsString(aperture_list,i,file_name)
      aperture_path = m_dir + "PSF\\" + file_name + ".dm3"
      If(DoesFileExist(aperture_path)) {
         Log("\n\t" + ++num_apertures + ". " + file_name + ".dm3 -> Averaged
PSF\\DeconvolutedAperture_" + file_name + ".dm3")
         aperture := OpenImage(aperture_path)
         aperture = RealIFFT(RealFFT(aperture)/RealFFT(avgpsf))
         SaveAsGatan3(aperture,m_dir + "PSF\\Averaged PSF\\DeconvolutedAperture_" + file_name)
      }
      mainpanel_win.DLGSetProgress("MainProgress",(i + 1)/num_tags/2 + 0.5)
   }
   If(abort) Return -1//Stop processing if "Stop" pressed
   mainpanel_win.DLGSetProgress("MainProgress",1)
   mainpanel_win.DLGSetProgress("SubProgress",1)
   Log("\n\n" + num_apertures + " deconvoluted aperture images saved to '" + m_dir +
"PSF\\Averaged PSF\\'")

   FinishLog("PSFAvg")

   ShowAlert("" + num_psfs + " PSFs averaged over!\n\n" + num_apertures + \
         " deconvoluted aperture images saved to 'PSF\\Averaged PSF\\'",2)

   Return 0
}
// 2.3      PSF correction looping function
Number PSFCorr_Batch()
{
   Number num_specimens = 0
   Image psf := OpenImage(psf_path)
   psf = psf/Sum(psf)
   Image otf = RealFFT(psf)

   StartLog("PSF Correction")
   Log("\nTry images from " + img_pre + num_start + img_suf + " to " + img_pre + num_end +
img_suf)
   Log("\nPSF file: " + psf_path)
   Log("\n\nBatch processing starts:")

   For(noi = num_start; noi <= num_end; noi++) {
      If(abort) Return -1//Stop processing if "Stop" pressed
      UpdateCurrentPaths(0)

      If(DoesFileExist(img_path)) {
         Log("\n\t" + img_name + ".dm3 - ")
         CreateDirectory(sub_dir)
         If(abort) Return -1//Stop processing if "Stop" pressed
         SaveAsGatan3(RealIFFT(RealFFT(OpenImage(img_path))/otf),sub_dir + img_name +
"_PSFCorrected")
         Log("Done; saved to '" + sub_dir + img_name + "_PSFCorrected.dm3'")
         num_specimens++
      }
      mainpanel_win.DLGSetProgress("MainProgress",(noi - num_start + 1)/(num_end - num_start +
1))
   }
   If(abort) Return -1//Stop processing if "Stop" pressed
   mainpanel_win.DLGSetProgress("MainProgress",1)
   mainpanel_win.DLGSetProgress("SubProgress",1)

   Log("\n\n\n" + num_specimens + " specimen images processed.")

   FinishLog("PSFCorr")

   ShowAlert("" + num_specimens + " specimen images processed!",2)

   Return 0
}
// 2.4      Noise characterisation looping function
Number NoiseChar_Batch()
{
   Number num_specimens = 0

   StartLog("Noise Characterisation")
   If(nc_raw == 1)
      Log("\nTry images from " + img_pre + num_start + img_suf + " to " + img_pre + num_end +
img_suf)
   Else
      Log("\nTry subfolders from " + sub_pre + num_start + sub_suf + " to " + sub_pre + num_end +
sub_suf)
   Log("\nTotal number of slices for sampling = " + nc_numslice)
   Log("\nPiecewise binning factor for images = " + nc_bin)
   Log("\nDark noise = " + nc_darknoise)
   Log("\nPre-determined mask = " + nc_mask + " (1 - yes, 0 - no)")
   Log("\nMask image: " + nc_mask_path)
   Log("\nGain normalisation target = " + nc_normal)
   Log("\nAlso output *.txt copies = " + nc_txt + " (1 - yes, 0 - no)")
   Log("\nApply on raw data instead of PSF corrected ones = " + nc_raw + " (1 - yes, 0 - no)")
   Log("\n\nBatch processing starts:")

   For(noi = num_start; noi <= num_end; noi++) {
      If(abort) Return -1//Stop processing if "Stop" pressed
      UpdateCurrentPaths(0)

      If(nc_raw == 0)
         img_path = m_dir + img_name + ".dm3"
      Else
         img_path = sub_dir + img_name + "_PSFCorrected.dm3"

      If(DoesFileExist(img_path)) {
         CreateDirectory(sub_dir)
         Log("\n\t" + PathExtractFileName(img_path,0) + " - ")
         NoiseChar(img_path,sub_dir,nc_darknoise,nc_mask,nc_normal,nc_numslice,nc_bin,nc_txt)
         If(abort) Return -1//Stop processing if "Stop" pressed
         Log("Done")
         num_specimens++
      }
      mainpanel_win.DLGSetProgress("MainProgress",(noi - num_start + 1)/(num_end - num_start +
1))
   }
   Log("\n\n\n" + num_specimens + " images processed.")
   mainpanel_win.DLGSetProgress("MainProgress",1)
   mainpanel_win.DLGSetProgress("SubProgress",1)

   FinishLog("NoiseChar")

   ShowAlert("" + num_specimens + " images processed!",2)

   Return 0
}
// 2.5      IGQCBED looping function
Number IGQCBED_Batch()
{
   Number num_subfolders = 0
   CreateDirectory(m_dir + "JPG\\")

   StartLog("IGQCBED Preparation")
   Log("\nTry subfolders from " + sub_pre + num_start + sub_suf + " to " + sub_pre + num_end +
sub_suf)
   Log("\n\n\nProcessing parameters:")
   Log("\n\tNumber of beams = " + ig_numbeams)
   Log("\n\tBeam diameter = " + ig_dia)
   Log("\n\tOutput binning = " + ig_bin)
   Log("\n\tIGQCBED switch = " + ig_switch + " (1 - gradient, 0 - conventional)")
   Log("\n\t1. Draw Mask option: Mask ratio = " + ig_ratio)
   Log("\n\t2. IGQCBED options: Magnification = " + ig_magn + ", Noise mask = " + ig_noisemask
+ \
         ", TXT copies = " + ig_finaltxt + " (1 - yes, 0 - no)")
   Log("\n\t4. parameters.dat options: Piecewise binning = " + ig_piecebin + ", Cycles = " +
gdc_cycles + ", Random steps = " + gdc_rand)
   Log("\n\n\nBatch processing starts:")

   For(noi = num_start; noi <= num_end; noi++) {
      If(abort) Return -1//Stop processing if "Stop" pressed
      UpdateCurrentPaths(0)

      IF(DoesDirectoryExist(sub_dir)){
         Log("\n\t" + sub_name)
         num_subfolders++

         if(ig_step_mask == 1)
            Log("\n\t\t1. Draw Mask: " + DrawMask(sub_dir,img_name,ig_numbeams,ig_dia,ig_ratio))
         If(abort) Return -1//Stop processing if "Stop" pressed
         if(ig_step_igqcbed == 1)
            Log("\n\t\t2. IGQCBED: " +
IGQCBED(sub_dir,ig_numbeams,ig_dia,ig_bin,ig_switch,ig_magn,ig_noisemask,ig_finaltxt))
         If(abort) Return -1//Stop processing if "Stop" pressed
         if(ig_step_for7 == 1)
            Log("\n\t\t3. FOR007.DAT: " + FOR007_DAT(sub_dir,ig_dia,ig_bin))
         If(abort) Return -1//Stop processing if "Stop" pressed
         if(ig_step_filepm == 1)
            Log("\n\t\t4. parameters.dat: " +
Parameters_dat(sub_dir,img_name,ig_numbeams,ig_dia,ig_bin,\
                  ig_switch,ig_piecebin,gdc_cycles,gdc_rand))
         If(abort) Return -1//Stop processing if "Stop" pressed
      }
      mainpanel_win.DLGSetProgress("MainProgress",(noi - num_start + 1)/(num_end - num_start +
1))
   }
   Log("\n\n" + num_subfolders + " sub-folders processed.")
   mainpanel_win.DLGSetProgress("MainProgress",1)
   mainpanel_win.DLGSetProgress("SubProgress",1)

   FinishLog("IGQCBED")

   ShowAlert("" + num_subfolders + " sub-folders processed!",2)

   Return 0
}
// 2.6      Extra step looping function
Number Extra_Batch()
{
   TagGroupSetTagAsTagGroup(GetUserPersistentTagGroup(),"BatchQCBEDPreparation",settings)

   StartLog("Extra Step")
   Log("\nExternal script: " + ex_script_path)
   Log("\nLoop option = " + ex_loop)
   If(ex_loop == 0) {
      Log("\nNo batch looping OR customised looping")
      Log("\n\nExternal script starts:")
      UpdateCurrentPaths(1)
      If(abort) Return -1//Stop processing if "Stop" pressed
      exit_code = ExecuteScriptFile(ex_script_path)
      If(abort) Return -1//Stop processing if "Stop" pressed
      TagGroupGetTagAsString(settings,"exit_text",exit_text)

      If(exit_code == 0 && exit_text == "")
         exit_text = "Seems like finished without error"
```

Appendix B. Program scripts

```
      mainpanel_win.DLGSetProgress("MainProgress",1)
      mainpanel_win.DLGSetProgress("SubProgress",1)

      Log("\n" + exit_text)
      FinishLog("Extra")
      ShowAlert("Extra Step finished!",2)
   } Else {
      Number count = 0
      If(ex_loop == 1)
        Log("\nTry images from " + img_pre + num_start + img_suf + " to " + img_pre + num_end +
img_suf + "\n")
      Else
        Log("\nTry subfolders from " + sub_pre + num_start + sub_suf + " to " + sub_pre + num_end
+ sub_suf + "\n")

      Log("\n\nBatch processing starts:")

      For(noi = num_start; noi <= num_end; noi++) {
        If(abort) Return -1//Stop processing if "Stop" pressed
        UpdateCurrentPaths(1)

        If(ex_loop == 1 && DoesFileExist(img_path)) {
          Log("\n\t" + img_name + ".dm3 - ")
          CreateDirectory(sub_dir)
        } Else If(ex_loop == 2 && DoesDirectoryExist(sub_dir))
          Log("\n\t" + sub_name + " - ")
        Else
          Continue

        exit_code = ExecuteScriptFile(ex_script_path)
        If(abort) Return -1//Stop processing if "Stop" pressed

        TagGroupGetTagAsString(settings,"exit_text",exit_text)
        If(exit_code == 0 && exit_text == "")
          exit_text = "Seems like finished without error"

        Log(exit_text)

        count++

        mainpanel_win.DLGSetProgress("MainProgress",(noi - num_start + 1)/(num_end - num_start
+ 1))
      }
      If(abort) Return -1//Stop processing if "Stop" pressed
      mainpanel_win.DLGSetProgress("MainProgress",1)
      mainpanel_win.DLGSetProgress("SubProgress",1)

      Log("\n\n\n" + count + " specimen images/subfolders processed.")
      FinishLog("Extra")
      ShowAlert("" + count + " specimen images/subfolders processed!",2)
   }

   Return 0
}


// 2     Batch Processing function end
//
//////////////////////////////////////////////////////////////////


//////////////////////////////////////////////////////////////////
//
// 3     GUI functions

TagGroup empty_l          = DLGCreateLabel("")

// 3.1      AMMPSF tag
TagGroup AMMPSF_tag()
{
   TagGroup measure_l1     = DLGCreateLabel("1. Create a folder named 'PSF' under the Main
Directory.").DLGAnchor("West")
   TagGroup measure_l2     = DLGCreateLabel("2. Put all aperture images into the 'PSF'
folder.").DLGAnchor("West")
   TagGroup measure_l3     = DLGCreateLabel("3. All *.dm3 files in this folder will be
processed.").DLGAnchor("West")
   TagGroup psf_nummeas_g  = DLGCreateIntegerField("Measurement
cycles:",psf_nummeas_f,psf_nummeas,5)
   TagGroup go_btn     = DLGCreatePushButton(" Measure AMMPSFs
","AMMPSF_Go").DLGIdentifier("AMMPSFGo").DLGInternalPadding(5,5).DLGExternalPadding(
20,5)
   TagGroup measure_box   = DLGCreateBox("1. AMMPSF Measurement").DLGFill("XY")
        measure_box.DLGAddElement(DLGGroupItems(measure_l1,measure_l2,measure_l3))

   measure_box.DLGAddElement(DLGGroupItems(psf_nummeas_g,go_btn).DLGTableLayout(2,1,
0))

   TagGroup average_l1      = DLGCreateLabel("1. Check and move unwanted PSF-result folders to
'Unwanted PSFs'.").DLGAnchor("West")
   TagGroup average_l2      = DLGCreateLabel("2. All PSF-result folders will be averaged over,
except the 'Averaged PSF'\n    and those in the 'Unwanted PSFs' folders.").DLGAnchor("West")
   TagGroup average_l3      = DLGCreateLabel("3. The averaged PSF-result will be in the 'Averaged
PSF' folder.").DLGAnchor("West")
   TagGroup average_btn     = DLGCreatePushButton(" Check folders && Start Averaging
","PSFAvg_Go").DLGIdentifier("PSFAvgGo").DLGInternalPadding(5,5)
   TagGroup average_box   = DLGCreateBox("2. PSFs Average").DLGFill("XY")
        average_box.DLGAddElement(DLGGroupItems(average_l1,average_l2,average_l3))
        average_box.DLGAddElement(average_btn)

   TagGroup ammpsf_dlg    = DLGCreateDialog("AMMPSF").DLGFill("X").DLGExpand("Y")
        ammpsf_dlg.DLGAddElement(measure_box)
        ammpsf_dlg.DLGAddElement(average_box)

   Return ammpsf_dlg
}
// 3.2      PSF correction tag
TagGroup PSFCorr_tag()
{
   String label1   = "1. Choose a normalised PSF image (whose sum = 1) for deconvolution. \n"
        label1 += "    E.g. 'AverageNormalisedPSF.dm3' or 'NormalisedPSF.dm3'. "
   TagGroup psfc_l1     = DLGCreateLabel(label1)

   String label2  = "2. Measure the background noise of aperture images deconvoluted by \n"
        label2 += "    this PSF for 'Dark noise' in Noise Characterisation."
   TagGroup psfc_l2     = DLGCreateLabel(label2)

   TagGroup psf_path_btn   = DLGCreatePushButton(" PSF ","GetPSFPath")
        psf_path_f      = DLGCreateStringField(psf_path,60)

   TagGroup go_btn     = DLGCreatePushButton(" Start PSF Correction
","PSFCorr_Go").DLGIdentifier("PSFCorrGo").DLGInternalPadding(5,5).DLGExternalPadding(10,
5)

   TagGroup psf_corr_dlg    = DLGCreateDialog("").DLGExpand("XY")
        psf_corr_dlg.DLGAddELement(psfc_l1,"","West")

   psf_corr_dlg.DLGAddELement(DLGGroupItems(psf_path_btn,psf_path_f).DLGTableLayout(2,1,
0))
        psf_corr_dlg.DLGAddELement(go_btn)
        psf_corr_dlg.DLGAddELement(psfc_l2,"","West")

   Return psf_corr_dlg
}
// 3.3      Noise Characterisation tag
TagGroup NoiseChar_tag()
{
        nc_mask_r = DLGCreateRadioList(nc_mask,"OptionChanged")
        nc_mask_r.DLGAddRadioItem("No -",0).DLGExternalPadding(0,2)
        nc_mask_r.DLGAddRadioItem("Yes -",1).DLGExternalPadding(0,2)
   TagGroup nc_normal_g   = DLGCreateRealField("Gain Normalisation
target:",nc_normal_f,nc_normal,22,1)
   TagGroup nc_mask_btn   = DLGCreatePushButton("Mask","GetMaskPath")
   TagGroup nc_mask_path_g = DLGCreateStringField("",nc_mask_path_f,nc_mask_path,40)
   TagGroup mask_box       = DLGCreateBox("Apply pre-determined mask?")
        mask_box.DLGAddElement(nc_mask_r,"Left","")
        mask_box.DLGAddElement(nc_normal_g.DLGIdentifier("GainNormGroup"))

   mask_box.DLGAddElement(DLGGroupItems(nc_mask_btn,nc_mask_path_g).DLGTableLayout(
2,1,0).DLGIdentifier("MaskPathGroup"))

   TagGroup numslice_g  = DLGCreateIntegerField("Total number of slices for
sampling:",nc_numslice_f,nc_numslice,10).DLGAnchor("East")
   TagGroup bin_g          = DLGCreateIntegerField("Piecewise binning factor for
images:",nc_bin_f,nc_bin,5).DLGAnchor("East")
   TagGroup darknoisehelp  = DLGCreatePushButton("?","DarkNoiseHelp").DLGAnchor("East")
   TagGroup darknoise_g = DLGCreateRealField("Dark noise:
",nc_darknoise_f,nc_darknoise,8,1).DLGAnchor("East")
   TagGroup field_gp       =
DLGGroupItems(numslice_g,bin_g,darknoisehelp.DLGSide("Right"),darknoise_g)

        nc_txt_c     = DLGCreateCheckBox("Also output Text Images",nc_txt).DLGAnchor("West")
        nc_raw_c     = DLGCreateCheckBox("Apply on raw data instead of PSF corrected
ones",nc_raw).DLGAnchor("West")
   TagGroup checkbox_gp   = DLGGroupItems(nc_txt_c,nc_raw_c)

   TagGroup go_btn         = DLGCreatePushButton(" Start Noise Characterisation
","NoiseChar_Go").DLGIdentifier("NoiseCharGo").DLGInternalPadding(5,5)

   TagGroup noise_char_dlg = DLGCreateDialog("Noise Characterising
Options").DLGExpand("XY")
        noise_char_dlg.DLGAddElement(field_gp,"","")
        noise_char_dlg.DLGAddElement(mask_box,"","")
        noise_char_dlg.DLGAddElement(checkbox_gp,"","")
        noise_char_dlg.DLGAddElement(empty_l,"","")
        noise_char_dlg.DLGAddElement(go_btn,"","")

   Return noise_char_dlg
}
// 3.4      IGQCBED tag
TagGroup IGQCBED_tag()
{
   TagGroup common_dlg          = DLGCreateDialog("Common
parameters").DLGExternalPadding(3,0).DLGTableLayout(3,2,0)
        common_dlg.DLGAddElement(DLGCreateIntegerField("Number of
beams:",ig_numbeams_f,ig_numbeams,4).DLGIdentifier("NumbeamsGroup"),"","East")
        common_dlg.DLGAddElement(DLGCreateIntegerField("   Beam
diameter:",ig_dia_f,ig_dia,6).DLGIdentifier("DiaGroup"),"","East")
        common_dlg.DLGAddElement(DLGCreateIntegerField("
Method:",ig_switch_f,ig_switch,3).DLGIdentifier("MethodGroup"),"","East")
        common_dlg.DLGAddElement(DLGCreateIntegerField("Output
binning:",ig_bin_f,ig_bin,4).DLGIdentifier("BinningGroup"),"","East")

   TagGroup manual_step_1_l= DLGCreateLabel("Pre-1: Find beam centres and save as
'centres.dm3' in each subfolder.")

        ig_step_mask_c       = DLGCreateCheckBox("1. Draw Mask -
",ig_step_mask,"OptionChanged")
   TagGroup mask_g          =
DLGCreateRealField("Ratio:",ig_ratio_f,ig_ratio,8,3).DLGIdentifier("MaskOptions")

   TagGroup manual_step_2_l= DLGCreateLabel("Pre-2: Enter zone-axis & basic vectors in
'centres.dm3' in each subfolder.")

        ig_step_igqcbed_c = DLGCreateCheckBox("2. IGQCBED -
",ig_step_igqcbed,"OptionChanged")
   TagGroup ig_magn_g         = DLGCreateRealField("Magnification:",ig_magn_f,ig_magn,6,0)
         ig_noisemask_c   = DLGCreateCheckBox("Noise mask",ig_noisemask)
         ig_finaltxt_c = DLGCreateCheckBox("TXT copies",ig_finaltxt)
```

Appendix B. Program scripts

```
   TagGroup igqcbed_opt_gp =
DLGGroupItems(ig_magn_g,ig_noisemask_c,ig_finaltxt_c).DLGTableLayout(3,1,0).DLGIdentifier(
"IGQCBEDOptions")

   TagGroup manual_step_3_l= DLGCreateLabel("Pre-3: Prepare 'FOR007_head.txt' &
'FOR007_tail.txt' in the 'Misc' folder.")

        ig_step_for7_c  = DLGCreateCheckBox("3. Pattern-matching command file:
FOR007.DAT",ig_step_for7,"OptionChanged")

        ig_step_filepm_c  = DLGCreateCheckBox("4. Geometric Distortion Correction command
file: parameters.dat",ig_step_filepm,"OptionChanged")
   TagGroup ig_piecebin_g     = DLGCreateIntegerField("Piecewise
binning:",ig_piecebin_f,ig_piecebin,4)
   TagGroup gdc_cycles_g      = DLGCreateIntegerField("Cycles:",gdc_cycles_f,gdc_cycles,6)
   TagGroup gdc_rand_g     = DLGCreateIntegerField("Random:",gdc_rand_f,gdc_rand,6)
   TagGroup filepm_opt_gp    =
DLGGroupItems(ig_piecebin_g,gdc_cycles_g,gdc_rand_g).DLGTableLayout(3,1,0).DLGIdentifier(
"FilepmOptions")

   TagGroup step_box         = DLGCreateBox("Steps").DLGExternalPadding(-3,0)
        step_box.DLGAddElement(manual_step_1_l,"","West")

   step_box.DLGAddElement(DLGGroupItems(ig_step_mask_c,mask_g).DLGTableLayout(2,1,0),"
","West")
        step_box.DLGAddElement(manual_step_2_l,"","West")

   step_box.DLGAddElement(DLGGroupItems(ig_step_igqcbed_c,igqcbed_opt_gp).DLGTableLayo
ut(2,1,0),"","West")
        step_box.DLGAddElement(manual_step_3_l,"","West")
        step_box.DLGAddElement(ig_step_for7_c,"","West")
        step_box.DLGAddElement(ig_step_filepm_c,"","West")
        step_box.DLGAddElement(filepm_opt_gp,"","")

   TagGroup checklist_btn      = DLGCreatePushButton("File
Checklist","CheckFiles").DLGInternalPadding(5,5).DLGExternalPadding(30,0)
   TagGroup go_btn             = DLGCreatePushButton("Go Go
Go","IGQCBED_Go").DLGIdentifier("IGQCBEDGo").DLGInternalPadding(5,5).DLGExternalPad
ding(30,0)

   TagGroup igqcbed_dlg        = DLGCreateDialog("Process").DLGFill("X")
        igqcbed_dlg.DLGAddElement(common_dlg.DLGFill("X"))
        igqcbed_dlg.DLGAddElement(step_box.DLGFill("X"))

   igqcbed_dlg.DLGAddElement(DLGGroupItems(checklist_btn,go_btn).DLGTableLayout(2,1,0))

   Return igqcbed_dlg
}
// 3.4.1   Check processing files for IGQCBED
Void CheckProcFiles()
{
   String head,tail,label1,label2
   If(DoesFileExist(m_dir + "Misc\\FOR007_head.txt"))
      head = "is in place!"
   Else
      head = "UNFOUND in the 'Misc' folder!"
   If(DoesFileExist(m_dir + "Misc\\FOR007_tail.txt"))
      tail = "is in place!"
   Else
      tail = "UNFOUND in the 'Misc' folder!"

   label1  = "FOR007.DAT templates:\n"
   label1 += "    FOR007_head.txt " + head + "\n"
   label1 += "    FOR007_tail.txt " + tail

   label2  = "Checklist of required input files in each subfolder:\n"
   label2 += "1. Draw Mask: \n        centres.dm3\n"
   label2 += "2. IGQCBED Preparation: \n        sigma.dm3, zeroth.dm3, mask.dm3 \&
centres.dm3\n"
   label2 += "3. File FOR007.DAT: \n        centres.dm3\n"
   label2 += "4. File parameters.dat: \n        centres.dm3"

   TagGroup file_check_dlg = DLGCreateDialog("File Checklist")

   file_check_dlg.DLGAddElement(DLGGroupItems(DLGCreateLabel(label1).DLGAnchor("West")
,DLGCreateLabel(label2).DLGAnchor("West"))).DLGInternalPadding(10,10)

   Object proc_file_check = alloc(UIframe).Init(file_check_dlg)
   proc_file_check.Display("File Checklist")
}
// 3.5      Extra tag
TagGroup Extra_tag()
{
   TagGroup unlock_c         = DLGCreateCheckBox("Unlock all input
fields",0,"OptionChanged").DLGIdentifier("Unlock")
   TagGroup ex_loop_l        = DLGCreateLabel("Batch looping under Main Directory:")
        ex_loop_r          = DLGCreateRadioList(ex_loop)
        ex_loop_r.DLGAddRadioItem("None/Costumised by external script",0)
        ex_loop_r.DLGAddRadioItem("Loop No. of image names",1)
        ex_loop_r.DLGAddRadioItem("Loop No. of subfolder names",2)

   TagGroup extra_btn        = DLGCreatePushButton(" Script ","GetScriptPath")
         ex_script_path_f  = DLGCreateStringField(ex_script_path,50)
   TagGroup go_btn             = DLGCreatePushButton(" Run
","Extra_Go").DLGIdentifier("ExtraGo").DLGInternalPadding(5,5)
   TagGroup external_opt_gp =
DLGGroupItems(extra_btn,ex_script_path_f,go_btn).DLGTableLayout(3,1,0)
   TagGroup external_box       = DLGCreateBox("External script")
        external_box.DLGAddElement(external_opt_gp)
        external_box.DLGAddElement(unlock_c)
        external_box.DLGAddElement(ex_loop_l)
        external_box.DLGAddElement(ex_loop_r)

   TagGroup ext_l            = DLGCreateLabel("Change 'sigma.txt' 'pattern.txt' 'in.txt' 'weights.txt' to
*.dat:")
   TagGroup ext_btn          = DLGCreatePushButton("txt ->
dat","ChangeExt").DLGInternalPadding(5,5).DLGIdentifier("ChangeExt")
   TagGroup test_btn         = DLGCreatePushButton(" Test ","Test").DLGIdentifier("Test")

   TagGroup extra_dlg        = DLGCreateDialog("Extra").DLGFill("X")//.DLGExpand("Y")

   extra_dlg.DLGAddElement(DLGGroupItems(ext_l,ext_btn).DLGTableLayout(2,1,0).DLGAnchor
("West"))
        extra_dlg.DLGAddElement(external_box.DLGFill("X"))
   If(test_mode)
        extra_dlg.DLGAddElement(test_btn)

   Return extra_dlg
}
// 3.6      Main Panel grouping
TagGroup status_l, settings_l
TagGroup Panel()
{
   TagGroup m_btn            = DLGCreatePushButton(" Main Directory ","GetMDir")
   m_dir_f = DLGCreateStringField(m_dir,60,"CheckDirEnding")
   TagGroup m_dir_gp         = DLGGroupItems(m_btn,m_dir_f).DLGTableLayout(2,1,0)

   TagGroup syntax_dlg       = DLGCreateDialog("").DLGTableLayout(4,3,0)
        syntax_dlg.DLGAddElement(DLGCreateLabel(""),"","")
        syntax_dlg.DLGAddElement(DLGCreateLabel("Images"),"","")
        syntax_dlg.DLGAddElement(DLGCreateLabel("Subfolders"),"","")
        syntax_dlg.DLGAddElement(DLGCreateLabel("Sorting No."),"","")
        syntax_dlg.DLGAddElement(DLGCreateLabel("Prefix:"),"","East")
        syntax_dlg.DLGAddElement(DLGCreateStringField("",img_pre_f,"",20),"","East")

   syntax_dlg.DLGAddElement(DLGCreateStringField("",sub_pre_f,"Set",20),"","East").DLGExter
nalPadding(5,0)
        syntax_dlg.DLGAddElement(DLGCreateIntegerField("   From",num_start_f,1,8),"","East")
        syntax_dlg.DLGAddElement(DLGCreateLabel("Suffix:"),"","East")
        syntax_dlg.DLGAddElement(DLGCreateStringField("",img_suf_f,"",20),"","East")

   syntax_dlg.DLGAddElement(DLGCreateStringField("",sub_suf_f,"",20),"","East").DLGExternalP
adding(5,0)
        syntax_dlg.DLGAddElement(DLGCreateIntegerField("To",num_end_f,100,8),"","East")
   TagGroup syntax_box = DLGCreateBox("Specimen Filename
Syntax").DLGIdentifier("FileSyntaxGroup")
        syntax_box.DLGAddElement(syntax_dlg)

        stage_tabs      = DLGCreateTabList(stage,"OptionChanged").DLGIdentifier("StageTabList")
        stage_tabs.DLGAddTab("1. AMMPSF").DLGAddElement(AMMPSF_tag())
        stage_tabs.DLGAddTab("2. PSF Corr.").DLGAddElement(PSFCorr_tag())
        stage_tabs.DLGAddTab("3. Noise Char.").DLGAddElement(NoiseChar_tag())
        stage_tabs.DLGAddTab("4. IGQCBED").DLGAddElement(IGQCBED_tag())
        stage_tabs.DLGAddTab("Extra").DLGAddElement(Extra_tag())

   TagGroup load_btn         = DLGCreatePushButton("  Load  ","LoadSettings")
   TagGroup save_btn         = DLGCreatePushButton("  Save  ","SaveSettings")
   TagGroup inspect_btn  = DLGCreatePushButton("Inspect","InspectSettings")
        settings_l        = DLGCreateLabel("").DLGWidth(35)
   TagGroup settings_box= DLGCreateDialog("Settings")
        settings_box.DLGAddElement(DLGCreateLabel("  Settings:"),"Left","")
        settings_box.DLGAddElement(load_btn,"Left","")
        settings_box.DLGAddElement(save_btn,"Left","")
        settings_box.DLGAddElement(inspect_btn,"Left","")
        settings_box.DLGAddElement(settings_l,"Left","")

   TagGroup stop_btn         = DLGCreatePushButton(" Stop
","Stop").DLGInternalPadding(5,8).DLGIdentifier("Stop")
   TagGroup main_bar         = DLGCreateProgressBar("MainProgress").DLGInternalPadding(65,0)
   TagGroup sub_bar    = DLGCreateProgressBar("SubProgress").DLGInternalPadding(65,0)
   TagGroup progress_gp = DLGGroupItems(DLGCreateLabel("
Main:"),main_bar,DLGCreateLabel("  Sub:"),sub_bar).DLGTableLayout(4,1,0)
        status_l       = DLGCreateLabel("Idle.").DLGWidth(58)
   TagGroup status_gp     = DLGGroupItems(DLGCreateLabel("
Status:"),status_l).DLGTableLayout(2,1,0)
   TagGroup status_dlg    = DLGCreateDialog("Status")
        status_dlg.DLGAddElement(stop_btn,"Right","")
        status_dlg.DLGAddElement(status_gp,"","West")
        status_dlg.DLGAddElement(progress_gp,"","West")

   TagGroup mainpanel_dlg= DLGCreateDialog("Main Panel")
        mainpanel_dlg.DLGAddElement(m_dir_gp.DLGFill("X").DLGExternalPadding(-5,-5,0,5))
        mainpanel_dlg.DLGAddElement(settings_box.DLGFill("X").DLGExternalPadding(5,0))
        mainpanel_dlg.DLGAddElement(syntax_box.DLGFill("X").DLGExternalPadding(5,0))
        mainpanel_dlg.DLGAddElement(status_dlg.DLGFill("X").DLGExternalPadding(5,0))
        mainpanel_dlg.DLGAddElement(stage_tabs.DLGFill("X"))

   Return mainpanel_dlg
}
// 3.7      The Test function
Number Test()
{
   Result("\nTesting. Waiting for interaction for 3 seconds:")
   OpenResultsWindow()
   For(number i = 1; i<=3; i++) {
     Sleep(1)
     mainpanel_win.DLGSetProgress("MainProgress",i/3)
     If(abort) Return -1//Stop processing if "Stop" pressed
   }

   Return 0
}
// 3.8      Background thread function
Class BatchThread: Thread
{
   String method
```

Appendix B. Program scripts

```
    Void Stop(Object self) { abort = 1; status_l.DLGTitle("Stopping..."); }

    Void Start(Object self,String input) {
        method = input
        self.StartThread()  //StartThread() runs thread in background
    }

    Void AutoSaveSettings(Object self) {
        VarToTag()
        CreateDirectory(m_dir + "Misc\\")
        TagGroupSaveToFile(settings,m_dir + "Misc\\settings")
        settings_l.DLGTitle("Auto-saved to 'Misc\\settings.gtg'")
    }

    Void RunThread(Object self) {
        self.AutoSaveSettings()
        running = 1
        mainpanel_win.OptionChanged(Null)
        abort = 0
        status_l.DLGTitle("Running " + method + "; DO NOT change any options!")
        mainpanel_win.DLGSetProgress("MainProgress",0)
        mainpanel_win.DLGSetProgress("SubProgress",0)

        If(method == "Test")
            Test()
        Else If(method == "AMMPSF Meas.")
            AMMPSF_Batch()
        Else If(method == "PSFs Avg")
            PSFAvg_Batch()
        Else If(method == "PSF Corr.")
            PSFCorr_Batch()
        Else If(method == "Noise Char.")
            NoiseChar_Batch()
        Else If(method == "IGQCBED Prep.")
            IGQCBED_Batch()
        Else If(method == "Extra Step")
            Extra_Batch()

        running = 0
        mainpanel_win.OptionChanged(Null)
        If(abort) {
            status_l.DLGTitle(method + " aborted.")
            mainpanel_win.DLGSetProgress("MainProgress",0)
            mainpanel_win.DLGSetProgress("SubProgress",0)
        } Else
            status_l.DLGTitle(method + " finished.")
    }
}

//
// 3.9      GUI class: panel respone functions (buttons & options etc.)
//

Class Panel_UI : UIframe
{
    Object batch_thread

    String CheckDirEnding(Object self,TagGroup dir0)
    {
        String dir
        DLGGetValue(dir0,dir)

        If(dir != "")
            If(Right(dir,1) != "\\") {
                dir = dir + "\\"
                dir0.DLGValue(dir)
            }

        Return dir
    }

    Void GetMDir(Object self) {
        If(GetDirectoryDialog(open_win,"Choose the Specimens Main directory",m_dir,m_dir)) {
            If(DoesFileExist(m_dir + "Misc\\settings.gtg"))
                If(TagGroupLoadFromFile(settings,m_dir + "Misc\\settings.gtg")) {
                    TagToVar()
                    settings_l.DLGTitle("Previous settings loaded.")
                }

            VarToPanel()
        }
    }

    Void CheckMainDirectory(Object self) {
        If(m_dir != "")
            m_dir = self.CheckDirEnding(m_dir_f)
        Else
            Throw("Specimens Main directory unspecified!")

        If(!DoesDirectoryExist(m_dir))
            Throw("Specimens Main directory " + m_dir + " unfound!")
    }

    Void LoadSettings(Object self) {
        If(OpenDialog(open_win,"Select the Settings TagGroup file",m_dir + "Misc\\",settings_path)) {
            PanelToVar()
            VarToTag()

            If(!TagGroupLoadFromFile(settings,settings_path))
                Throw(settings_path + " loading fail!")

            TagToVar()
            VarToPanel()
            settings_l.DLGTitle("Settings loaded.")
        }
    }

    Void SaveSettings(Object self) {
        PanelToVar()
        VarToTag()

        self.CheckMainDirectory()
        CreateDirectory(m_dir + "Misc\\")
        TagGroupSaveToFile(settings,m_dir + "Misc\\settings")
        settings_l.DLGTitle("Saved to 'Misc\\settings.gtg'.")
    }

    Void InspectSettings(Object self) {
        PanelToVar()
        VarToTag()
        TagGroupOpenBrowserWindow(settings,0)
    }

    Void DarkNoiseHelp(Object self) {
        OkDialog("Dark Noise can be determined from background noise of aperture images
deconvoluted by the same PSF. Input '-1' if unknown.")
    }

    Void GetPSFPath(OBject self) {
        If(OpenDialog(open_win,"Choose a normalised PSF image for PSF Corr.",m_dir +
"PSF\\",psf_path))
            psf_path_f.DLGValue(psf_path)
    }

    Void GetMaskPath(OBject self) {
        If(OpenDialog(open_win,"Choose a mask image",m_dir,nc_mask_path))
            nc_mask_path_f.DLGValue(nc_mask_path)
    }

    Void GetScriptPath(Object self) {
        If(OpenDialog(open_win,"Choose a script for the Extra Step","",ex_script_path))
            ex_script_path_f.DLGValue(ex_script_path)
    }

    Void OptionChanged(Object self,TagGroup tmp) {
        PanelToVar()
        Number unlock
        self.DLGGetValue("Unlock",unlock)

        If(nc_mask || unlock) {
            self.SetElementIsEnabled("MaskPathGroup",1)
            self.SetElementIsEnabled("GainNormGroup",0)
        } Else {
            self.SetElementIsEnabled("MaskPathGroup",0)
            self.SetElementIsEnabled("GainNormGroup",1)
        }

        //If(stage > 0 || unlock)
        // self.SetElementIsEnabled("FileSyntaxGroup",1)
        //Else
        // self.SetElementIsEnabled("FileSyntaxGroup",0)

        If(ig_step_mask || ig_step_igqcbed || ig_step_filepm || unlock)
            self.SetElementIsEnabled("NumbeamsGroup",1)
        Else
            self.SetElementIsEnabled("NumbeamsGroup",0)

        If(ig_step_mask || ig_step_igqcbed || ig_step_filepm || ig_step_for7 || unlock)
            self.SetElementIsEnabled("DiaGroup",1)
        Else
            self.SetElementIsEnabled("DiaGroup",0)

        If(ig_step_igqcbed || ig_step_filepm || ig_step_for7 || unlock)
            self.SetElementIsEnabled("BinningGroup",1)
        Else
            self.SetElementIsEnabled("BinningGroup",0)

        If(ig_step_igqcbed || ig_step_filepm || unlock)
            self.SetElementIsEnabled("MethodGroup",1)
        Else
            self.SetElementIsEnabled("MethodGroup",0)

        If(ig_step_mask || unlock)
            self.SetElementIsEnabled("MaskOptions",1)
        Else
            self.SetElementIsEnabled("MaskOptions",0)

        If(ig_step_igqcbed || unlock)
            self.SetElementIsEnabled("IGQCBEDOptions",1)
        Else
            self.SetElementIsEnabled("IGQCBEDOptions",0)

        If(ig_step_filepm || unlock)
            self.SetElementIsEnabled("FilepmOptions",1)
        Else
            self.SetElementIsEnabled("FilepmOptions",0)

        If(running) {
            self.SetElementIsEnabled("Stop",1)
            self.SetElementIsEnabled("AMMPSFGo",0)
            self.SetElementIsEnabled("PSFAvgGo",0)
            self.SetElementIsEnabled("PSFCorrGo",0)
            self.SetElementIsEnabled("NoiseCharGo",0)
            self.SetElementIsEnabled("IGQCBEDGo",0)
            self.SetElementIsEnabled("ExtraGo",0)
        } Else {
            self.SetElementIsEnabled("Stop",0)
            self.SetElementIsEnabled("AMMPSFGo",1)
```

Appendix B. Program scripts

```
        self.SetElementIsEnabled("PSFAvgGo",1)
        self.SetElementIsEnabled("PSFCorrGo",1)
        self.SetElementIsEnabled("NoiseCharGo",1)
        self.SetElementIsEnabled("IGQCBEDGo",1)
        self.SetElementIsEnabled("ExtraGo",1)
    }
}

Void CheckFiles(Object self) {
    PanelToVar()
    CheckProcFiles()
}

Void AMMPSF_Go(Object self) {
    PanelToVar()
    CheckMainDirectory(self)

    If(!DoesDirectoryExist(m_dir + "PSF\\"))
        Throw("PSF folder unfound!")

    batch_thread.Start("AMMPSF Meas.")
}

Void PSFAvg_Go(Object self) {
    PanelToVar()
    CheckMainDirectory(self)

    If(!DoesDirectoryExist(m_dir + "PSF\\"))
        Throw("PSF folder unfound!")

    TagGroup one_folder,folders = GetFilesInDirectory(m_dir + "PSF\\",2)
    String name,yes_list = "",no_list = "",dir
    Number yes_total = 0,no_total = 0
    For(number i = 0; i < TagGroupCountTags(folders); i++){
        TagGroupGetIndexedTagAsTagGroup(folders,i,one_folder)
        TagGroupGetTagAsString(one_folder,"Name",name)
        dir = m_dir + "PSF\\" + name + "\\"
        If(name != "Averaged PSF" && name != "Unwanted PSFs")
            If(DoesFileExist(dir + "\\NormalisedPSF.dm3") || DoesFileExist(dir + "\\psf.dm3"))
                yes_list += "\n\t" + ++yes_total + ". " + name + "\\"
            Else
                no_list += "\n\t" + ++no_total + ". " + name + "\\"
    }

    If(yes_total == 0)
        yes_list = "\n\t(None)"
    If(no_total == 0)
        no_list = "\n\t(None)"

    If(!OkCancelDialog("The following PSF subfolders will be processed:" + yes_list + \
        "\n\nThe following subfolders without 'NormalisedPSF.dm3' nor 'psf.dm3' will be ignored:"
+ no_list))
        Exit(0)

    batch_thread.Start("PSFs Avg")
}

Void PSFCorr_Go(Object self) {
    PanelToVar()
    self.CheckMainDirectory()

    If(!DoesFileExist(psf_path))
        Throw("PSF image '" + psf_path + "' unfound!")

    If(!DoesDirectoryExist(m_dir + "PSF\\"))
        Throw("PSF folder unfound!")

    batch_thread.Start("PSF Corr.")
}

Void NoiseChar_Go(Object self) {
    PanelToVar()
    CheckMainDirectory(self)

    If(nc_mask == 1) {
        If(nc_mask_path == "")
            Throw("Mask image file unspecified!")
        Else If(!DoesFileExist(nc_mask_path))
            Throw("Mask image '" + nc_mask_path + "' unfound!")
    }

    If(nc_bin < 1)
        Throw("Piecewise binning factor must be greater than 0!")

    batch_thread.Start("Noise Char.")
}

Void IGQCBED_Go(Object self) {
    PanelToVar()
    CheckMainDirectory(self)

    If(ig_step_mask || ig_step_igqcbed || ig_step_filepm)
        If(ig_numbeams < 1)
            Throw("Number of beams must be greater than 0!")

    If(ig_step_mask || ig_step_igqcbed || ig_step_filepm || ig_step_for7)
        If(ig_dia < 1 || (ig_bin ? Mod(ig_dia,ig_bin) : 0) || Mod(ig_dia,2))
            Throw("Beam diameter must be: \n1) Divisible by Binning, \n2) Even, and \n2) Greater
than 0!")

    If(ig_step_igqcbed || ig_step_filepm || ig_step_for7)
        If(ig_bin < 1 || Mod(ig_dia,ig_bin))
            Throw("Binning factor must be divisible to beam diameter and greater than 0!")

    If(ig_step_igqcbed == 1)
        If(ig_magn <= 0)
            Throw("IGQCBED magnification must be greater than 0!")

    If(ig_step_for7 == 1)
        If(!DoesFileExist(m_dir + "Misc\\FOR007_head.txt") || !DoesFileExist(m_dir +
"Misc\\FOR007_tail.txt"))
            Throw("'FOR007_head.txt' and/or 'FOR007_tail.txt' unfound in the 'Misc' folder!")

    If(ig_step_filepm == 1)
        If(ig_piecebin < 1)
            Throw("Piecewise binning factor must be greater than 0!")

    batch_thread.Start("IGQCBED Prep.")
}

Void Extra_Go(Object self) {
    PanelToVar()
    VarToTag()
    CheckMainDirectory(self)

    If(ex_script_path == "")
        Throw("Script file unspecified!")
    Else If(!DoesFileExist(ex_script_path))
        Throw("Script file '" + ex_script_path + "' unfound!")


TagGroupSetTagAsTagGroup(GetUserPersistentTagGroup(),"BatchQCBEDPreparation",settings)

    batch_thread.Start("Extra Step")
}

Void ChangeExt(Object self) {
    PanelToVar()
    CheckMainDirectory(self)

    String  bat  = "cd %~dp0 \n"
          bat += "for /d %%a in (..\\*) do "
          bat += "for /r %%f in (sigma.txt,pattern.txt.txt,in.txt,weights.txt) do "
          bat += "ren %%~fa\\%%~nf.txt %%~nf.dat"

    String bat_path = m_dir + "Misc\\txt2dat.bat"
    Number bat_id = CreateFileForWriting(bat_path)
    Object bat_stream = NewStreamFromFileReference(bat_id,1)
          bat_stream.StreamWriteAsText(0,bat)
    CloseFile(bat_id)

    LaunchExternalProcessAsync(bat_path)
}

Void Test(Object self) {
    batch_thread.Start("Test");
}

Void Stop(Object self) { batch_thread.Stop(); }

Object Init(Object self, Number batch_thread_id) {
    batch_thread = GetScriptObjectFromID(batch_thread_id)

    Return self.super.Init(Panel())
}
}

// 3    GUI functions end
//
////////////////////////////////////////////////////////////////

// 4    Main function
Object process_thread = alloc(BatchThread)
mainpanel_win = alloc(Panel_UI).Init(process_thread.ScriptObjectGetID())
mainpanel_win.Display("Batch IGQCBED Preparation")
mainpanel_win.OptionChanged(Null)
```

B.3.2. File “GetCentres_v1.1.s” – script to assist the determination of the centres of reflection discs in CBED patterns

```
// $BACKGROUND$

// Get Centres
// A tool to help finding centre locations etc. for 'Batch QCBED Preparation'
// for DM versions later than 2.* (especially for 2.32.*)
// Tianyu Liu, Aug 2016

// Changelog
// v 1.1, Vector option X added
// V 1.0


////////////////////////////////////////////////////////////////
//
//      Function for Panel

//
// Panel items
//
```

Appendix B. Program scripts

```
TagGroup Centres_tag() {
   TagGroup draw_oval_btn   = DLGCreatePushButton("Draw Ovals","DrawOvals")

   TagGroup nb_l        = DLGCreateLabel("Beams:")
   TagGroup nb_f        =
DLGCreateIntegerField(3,4,"UpdateAnnotations").DLGIdentifier("NumBeams")
   TagGroup d_l       = DLGCreateLabel("d=")
   TagGroup d_f         =
DLGCreateIntegerField(400,6,"UpdateAnnotations").DLGExternalPadding(-
3,0).DLGIdentifier("Diameter")
   TagGroup d_plus_btn      =
DLGCreatePushButton("+","DiameterPlus").DLGWidth(10).DLGExternalPadding(-3,0)
   TagGroup d_minus_btn   = DLGCreatePushButton("-
","DiameterMinus").DLGWidth(10).DLGExternalPadding(-3,0)

   TagGroup centres_dlg  = DLGCreateDialog("")//.DLGExternalPadding(3,0)
        centres_dlg.DLGAddElement(DLGGroupItems(nb_l,nb_f).DLGTableLayout(2,1,0))

   centres_dlg.DLGAddElement(DLGGroupItems(d_l,d_f,d_plus_btn,d_minus_btn).DLGTableLayo
ut(4,1,0))
        centres_dlg.DLGAddElement(draw_oval_btn)

   Return centres_dlg.DLGExpand("Y")
}

TagGroup ZoneAxis_tag() {
   TagGroup za_a_f  = DLGCreateIntegerField(0,4).DLGIdentifier("ZoneAxisA")
   TagGroup za_b_f  = DLGCreateIntegerField(0,4).DLGIdentifier("ZoneAxisB")
   TagGroup za_c_f  = DLGCreateRealField(2,6,10).DLGIdentifier("ZoneAxisC")

   TagGroup fields_dlg    = DLGCreateDialog("").DLGTableLayout(2,3,0).DLGExternalPadding(0,-
2)
        fields_dlg.DLGAddElement(DLGCreateLabel("A =
Beam"),"","East").DLGExternalPadding(0,0,-2,-2)
        fields_dlg.DLGAddElement(za_a_f,"","West").DLGExternalPadding(0,2,-2,0)
        fields_dlg.DLGAddElement(DLGCreateLabel("B =
Beam"),"","East").DLGExternalPadding(-2,-2)
        fields_dlg.DLGAddElement(za_b_f,"","West").DLGExternalPadding(-2,-2)
        fields_dlg.DLGAddElement(DLGCreateLabel("Number c
="),"","East").DLGExternalPadding(2,0,0,-2)
        fields_dlg.DLGAddElement(za_c_f,"","West").DLGExternalPadding(2,2,0,0)

   TagGroup za_dlg      = DLGCreateDialog("")
        za_dlg.DLGAddElement(DLGCreateLabel("Zone Axis = (A+B)/c").DLGExternalPadding(0,-
2))
        za_dlg.DLGAddElement(fields_dlg)

   Return za_dlg.DLGExpand("Y")
}

TagGroup Vectors_tag() {
   TagGroup v1_f        = DLGCreateIntegerField(1,4).DLGIdentifier("Vector1")
   TagGroup v2_f        = DLGCreateIntegerField(2,4).DLGIdentifier("Vector2")
   TagGroup x_f         = DLGCreateIntegerField(0,4).DLGIdentifier("Vector0")

   TagGroup draw_v_btn     = DLGCreatePushButton("Draw
Vectors","DrawVectors").DLGExternalPadding(-2,-3)

   TagGroup fields_dlg    = DLGCreateDialog("").DLGTableLayout(2,3,0).DLGExternalPadding(0,-
2)
        fields_dlg.DLGAddElement(DLGCreateLabel("V1 = Beam")).DLGExternalPadding(0,0,-2,-
2)
        fields_dlg.DLGAddElement(v1_f).DLGExternalPadding(0,2,-2,0)
        fields_dlg.DLGAddElement(DLGCreateLabel("V2 = Beam")).DLGExternalPadding(2,0,-2,-
2)
        fields_dlg.DLGAddElement(v2_f).DLGExternalPadding(2,2,-2,0)
        fields_dlg.DLGAddElement(DLGCreateLabel(" X = Beam")).DLGExternalPadding(2,0,0,-2)
        fields_dlg.DLGAddElement(x_f).DLGExternalPadding(2,2,0,0)

   TagGroup vector_dlg       = DLGCreateDialog("")
        vector_dlg.DLGAddElement(DLGCreateLabel("Vector = V-
X+ZA").DLGExternalPadding(0,-2))
        vector_dlg.DLGAddElement(fields_dlg)
        vector_dlg.DLGAddElement(draw_v_btn)

   Return vector_dlg.DLGExpand("Y")
}

Void AddColourItems(TagGroup &items) {
   items.DLGAddPopupItemEntry("Black")    //2. rgb(0,0,0)
   items.DLGAddPopupItemEntry("Red")    //3. rgb(1,0,0)
   items.DLGAddPopupItemEntry("Yellow")   //4. rgb(1,1,0)
   items.DLGAddPopupItemEntry("Lime")     //5. rgb(0,1,0)
   items.DLGAddPopupItemEntry("Aqua")     //6. rgb(0,1,1)
   items.DLGAddPopupItemEntry("Blus")     //7. rgb(0,0,1)
   items.DLGAddPopupItemEntry("Fuchsia")  //8. rgb(1,0,1)
   items.DLGAddPopupItemEntry("White")    //9. rgb(1,1,1)
}

TagGroup Misc_tag() {
   TagGroup label_l     = DLGCreateLabel("Label:").DLGExternalPadding(0,-2)
   TagGroup label_opt     =
DLGCreatePopup(2,"UpdateAnnotations").DLGIdentifier("LabelOption").DLGInternalPadding(-
5,0).DLGExternalPadding(-2,-2)
        label_opt.DLGAddPopupItemEntry("No label")
        label_opt.DLGAddPopupItemEntry("Corner")
        label_opt.DLGAddPopupItemEntry("Centre")
   TagGroup shape_c_opt=
DLGCreatePopup(6,"UpdateAnnotations").DLGIdentifier("ShapeColour").DLGInternalPadding(-
11,0).DLGExternalPadding(-3,-2)
        shape_c_opt.DLGAddPopupItemEntry("Shape")
        AddColourItems(shape_c_opt)
   TagGroup label_c_opt =
DLGCreatePopup(3,"UpdateAnnotations").DLGIdentifier("LabelColour").DLGInternalPadding(-
11,0).DLGExternalPadding(-2,-2)
        label_c_opt.DLGAddPopupItemEntry("Label")
        AddColourItems(label_c_opt)

   TagGroup from_c_btn     = DLGCreatePushButton("From
'centres'","FromCentres").DLGExternalPadding(-4,-2)

   TagGroup misc_dlg = DLGCreateDialog("")
        misc_dlg.DLGAddElement(DLGGroupItems(label_l,label_opt).DLGTableLayout(2,1,0))

   misc_dlg.DLGAddElement(DLGGroupItems(shape_c_opt,label_c_opt).DLGTableLayout(2,1,0))
        misc_dlg.DLGAddElement(from_c_btn.DLGFill("X"))

   Return misc_dlg.DLGExpand("Y")
}

TagGroup Panel(Number modeless)
{
   TagGroup lock_front_btn=
DLGCreatePopup(1,"ChooseFrontImage").DLGIdentifier("LockFrontImage").DLGInternalPadding(
-12,0).DLGExternalPadding(3,-2)
        lock_front_btn.DLGAddPopupItemEntry("Front Image")
        lock_front_btn.DLGAddPopupItemEntry("Choose & Lock")
   TagGroup float_btn     =
DLGCreatePushButton("Dock","FloatingWindow").DLGExternalPadding(0,-2)
   TagGroup help_btn      = DLGCreatePushButton(" ? ","Help").DLGExternalPadding(-3,-2)

   TagGroup stage_tabs,header_gp = DLGCreateDialog("").DLGFill("X")
   If(modeless) {
      header_gp.DLGAddElement(help_btn,"Right","")
      header_gp.DLGAddElement(float_btn,"Left","")
      header_gp.DLGAddElement(lock_front_btn,"Center","")

      stage_tabs         = DLGCreateDialog("").DLGExternalPadding(3,0).DLGTableLayout(2,2,0)
      stage_tabs.DLGAddElement(DLGCreateBox("1.
Centres").DLGFill("XY").DLGExternalPadding(0,-2)).DLGAddElement(Centres_tag())
      stage_tabs.DLGAddElement(DLGCreateBox("2. Zone
Axis").DLGFill("XY").DLGExternalPadding(0,-2)).DLGAddElement(ZoneAxis_tag())
      stage_tabs.DLGAddElement(DLGCreateBox("3.
Vectors").DLGFill("XY").DLGExternalPadding(0,0)).DLGAddElement(Vectors_tag())

    stage_tabs.DLGAddElement(DLGCreateBox("Misc.").DLGFill("XY").DLGExternalPadding(0,0)
).DLGAddElement(Misc_tag())
   } Else {
      header_gp         = DLGGroupItems(lock_front_btn,help_btn).DLGTableLayout(2,1,0)

      stage_tabs         = DLGCreateTabList(0)
      stage_tabs.DLGAddTab(" C ").DLGAddElement(Centres_tag())
      stage_tabs.DLGAddTab(" Z ").DLGAddElement(ZoneAxis_tag())
      stage_tabs.DLGAddTab(" V ").DLGAddElement(Vectors_tag())
      stage_tabs.DLGAddTab(" * ").DLGAddElement(Misc_tag())
   }

   TagGroup get_l_btn     = DLGCreatePushButton("Centres","ToCentres").DLGExternalPadding(-
2,-2)//.DLGInternalPadding(5,0)
   TagGroup info_btn      = DLGCreatePushButton("Info","Info").DLGExternalPadding(-3,-2)
   TagGroup tilt_btn     = DLGCreatePushButton("Tilts","Tilts").DLGExternalPadding(-2,-2)

   TagGroup container  = DLGCreateDialog("")
        container.DLGAddElement(header_gp)//.DLGExternalPadding(-3,0)
        container.DLGAddElement(stage_tabs)

   container.DLGAddElement(DLGGroupItems(get_l_btn,info_btn,tilt_btn).DLGTableLayout(3,1,0)
)


   //TagGroupOpenBrowserWindow(container,0)

   Return container
}

//
// Panel buttons respone setting
//

Class Panel_UI : UIframe
{
   Number nb,d,fs,label,v1,v2,v0,za,zb,zc,lc,sc,lock,xrange,yrange
   Image front
   ImageDisplay disp
   TagGroup gp_ids

   Void PanelToVar(Object self) {
      self.DLGGetValue("NumBeams",nb)
      self.DLGGetValue("Diameter",d)
      self.DLGGetValue("Vector1",v1)
      self.DLGGetValue("Vector2",v2)
      self.DLGGetValue("Vector0",v0)
      self.DLGGetValue("ZoneAxisA",za)
      self.DLGGetValue("ZoneAxisB",zb)
      self.DLGGetValue("ZoneAxisC",zc)
      self.DLGGetValue("ShapeColour",sc)
      self.DLGGetValue("LabelColour",lc)
      self.DLGGetValue("LabelOption",label)
      self.DLGGetValue("LockFrontImage",lock)
   }

   Void ChooseFrontImage(Object self, TagGroup tg) {
      self.DLGGetValue("LockFrontImage",lock)
      If(lock==2)
         If(!GetOneImage("Choose & lock",front))
            self.DLGValue("LockFrontImage",1)
```

Appendix B. Program scripts

```
}

TagGroup GetAnnotations(Object self,ImageDisplay disp_in) {
  TagGroup an_ids = NewTagGroup()

  For(Number i=0;i<ComponentCountChildrenOfType(disp_in,17);i++) {
    Component gp = ComponentGetNthChildOfType(disp_in,17,i)
    If(ComponentIsOfType(ComponentGetChild(gp,0),6)) {
      String l = TextAnnotationGetText(ComponentGetChild(gp,1))
      Number id = ComponentGetID(gp)
      TagGroupSetTagAsNumber(an_ids,"C"+l,id)
    } Else If(ComponentIsOfType(ComponentGetChild(gp,0),3)) {
      Number id = ComponentGetID(gp)
      TagGroupSetTagAsNumber(an_ids,"V",id)
    }
  }

  Return an_ids
}

Void FrontImage(Object self) {
  self.DLGGetValue("LockFrontImage",lock)
  If(lock==1) {
    If(!GetFrontImage(front))
      Exit(0)
    If(GetName(front) == "centres")
      Exit(0)
  }

  If(!ImageIsValid(front)) {
    self.DLGValue("LockFrontImage",1)
    Exit(0)
  }

  SelectImage(front)
  disp = front.ImageGetImageDisplay(0)
  GetSize(front,xrange,yrange)
  gp_ids = GetAnnotations(self,disp)
}

Component OneNewOvalGroup(Object self,Number i) {
  Component oval = NewOvalAnnotation(0,0,d,d)
  Component no = NewTextAnnotation(0,0,""+i,d/8)
      no.ComponentSetFontFaceName("Microsoft Sans Serif")
  Component gp = NewGroupAnnotation()
      gp.ComponentAddChildAtEnd(oval)
      gp.ComponentAddChildAtEnd(no)

  Return gp
}

Void GetCentreOfOvalGroup(Object self,Component gp,Number &x,Number &y) {
  number t,l,b,r
  Component oval = ComponentGetChild(gp,0)
  Component no = ComponentGetChild(gp,1)
  GroupAnnotationUngroup(gp)
  ComponentGetControlPoint(oval,3,x,y)
  ComponentGetRect(oval,t,l,b,r)
  gp.ComponentAddChildAtEnd(oval)
  gp.ComponentAddChildAtEnd(no)
}

Void SetColour(Object self,Component &obj,Number c) {
  If(c>1)
    obj.ComponentSetForegroundColor((c==3||c==4||c==8||c==9)?1:0,\
                    (c==4||c==5||c==6||c==9)?1:0,\
                    (c==6||c==7||c==8||c==9)?1:0)
}

Number UpdateAnnotations(Object self,TagGroup tg) {
  Number id,x,y,t,l,b,r,fs,updated = 0

  PanelToVar(self)
  FrontImage(self)

  If(TagGroupIsValid(tg) && !TagGroupCountTags(gp_ids))
    Exit(0)

  For(Number i=0;i<nb;i++) {
    Component gp,oval,no

    If(TagGroupGetTagAsNumber(gp_ids,"C"+i,id)) {
      gp = ComponentGetChildByID(disp,id)
      oval = ComponentGetChild(gp,0)
      no = ComponentGetChild(gp,1)
      ComponentGetRect(no,t,l,b,r)
      GroupAnnotationUngroup(gp)
      ComponentGetControlPoint(oval,3,x,y)
      ComponentSetRect(oval,y-d/2,x-d/2,y+d/2,x+d/2)
      ComponentSetControlPoint(no,3,x,y,1)
      gp.ComponentAddChildAtEnd(oval)
      gp.ComponentAddChildAtEnd(no)
      ComponentSetRect(no,0,0,b-t,r-l)
    } Else {
      gp = OneNewOvalGroup(self,i)
      oval = ComponentGetChild(gp,0)
      no = ComponentGetChild(gp,1)
      ComponentAddChildAtEnd(disp,gp)
      ComponentSetControlPoint(gp,3,d/2,(yrange-d)/(nb-1)*i+d/2,1)
      updated = 1
    }


    If(label == 3)
      ComponentSetControlPoint(no,3,d/2,d/2,0)
    ComponentSetVisible(no,label-1)
    SetColour(self,no,lc)
    SetColour(self,oval,sc)
  }

  If(TagGroupGetTagAsNumber(gp_ids,"V",id)) {
    Component gp = ComponentGetChildByID(disp,id)
    Component arrow1 = ComponentGetChild(gp,0)
    Component arrow2 = ComponentGetChild(gp,1)
    Component no1 = ComponentGetChild(gp,2)
    Component no2 = ComponentGetChild(gp,3)

    ComponentSetVisible(no1,label-1)
    ComponentSetVisible(no2,label-1)
    SetColour(self,arrow1,sc)
    SetColour(self,arrow2,sc)
    SetColour(self,no1,lc)
    SetColour(self,no2,lc)
  }

  Return updated
}

Void DrawOvals(Object self) {
  PanelToVar(self)
  FrontImage(self)

  For(Number i=0;i<nb;i++) {
    If(!TagGroupDoesTagExist(gp_ids,"C"+i)) {
      Component gp = OneNewOvalGroup(self,i)
      ComponentAddChildAtEnd(disp,gp)
      ComponentSetControlPoint(gp,3,d/2,(yrange-d)/(nb-1)*i+d/2,1)
    }
  }

  UpdateAnnotations(self,NULL)
}

Void DrawVectors(Object self) {
  Number id,id1,id2,ida,idb,v0x,v0y,v1x,v1y,v2x,v2y,cax,cay,cbx,cby,x,y
  PanelToVar(self)
  FrontImage(self)

  If(TagGroupGetTagAsNumber(gp_ids,"V",id))
    ComponentRemoveFromParent(ComponentGetChildByID(disp,id))

  If(TagGroupGetTagAsNumber(gp_ids,"C"+v0,id) &&
TagGroupGetTagAsNumber(gp_ids,"C"+v1,id1) \
      && TagGroupGetTagAsNumber(gp_ids,"C"+v2,id2) &&
TagGroupGetTagAsNumber(gp_ids,"C"+za,ida) \
      && TagGroupGetTagAsNumber(gp_ids,"C"+zb,idb)) {
    GetCentreOfOvalGroup(self,ComponentGetChildByID(disp,id),v0x,v0y)
    GetCentreOfOvalGroup(self,ComponentGetChildByID(disp,id1),v1x,v1y)
    GetCentreOfOvalGroup(self,ComponentGetChildByID(disp,id2),v2x,v2y)
    GetCentreOfOvalGroup(self,ComponentGetChildByID(disp,ida),cax,cay)
    GetCentreOfOvalGroup(self,ComponentGetChildByID(disp,idb),cbx,cby)
    Number zx = (cax+cbx)/zc, zy = (cay+cby)/zc
    v1x += zx - v0x; v1y += zy - v0y; v2x += zx - v0x; v2y += zy - v0y

    Component tmp = NewBoxAnnotation(0,0,0,0)
    Component arrow1 = NewArrowAnnotation(zy,zx,v1y,v1x)
    Component no1 = NewTextAnnotation((v1x+zx)/2,(v1y+zy)/2,"V1",d/8)
        no1.ComponentSetFontFaceName("Microsoft Sans Serif")
    Component arrow2 = NewArrowAnnotation(zy,zx,v2y,v2x)
    Component no2 = NewTextAnnotation((v2x+zx)/2,(v2y+zy)/2,"V2",d/8)
        no2.ComponentSetFontFaceName("Microsoft Sans Serif")

    Component gp = NewGroupAnnotation()
        gp.ComponentAddChildAtEnd(tmp)
        gp.ComponentAddChildAtEnd(arrow1)
        gp.ComponentAddChildAtEnd(arrow2)
        gp.ComponentAddChildAtEnd(no1)
        gp.ComponentAddChildAtEnd(no2)
    ComponentRemoveFromParent(tmp)

    ComponentAddChildAtBeginning(disp,gp)

    UpdateAnnotations(self,NULL)
  }
}

Image GetLocations(Object self) {
  If(UpdateAnnotations(self,NULL))
    Exit(0)

  Image centres := IntegerImage("centres",4,1,4,nb)

  Number id,x1,x2,y1,y2,v1x,v1y,v2x,v2y,zx,zy
  For(Number i=0;i<nb;i++) {
    TagGroupGetTagAsNumber(gp_ids,"C"+i,id)
    GetCentreOfOvalGroup(self,ComponentGetChildByID(disp,id),x1,y1)
    SetPixel(centres,0,i,Nearest(x1))
    SetPixel(centres,1,i,Nearest(y1))
  }

  If(TagGroupGetTagAsNumber(gp_ids,"V",id)) {
    Component gp = ComponentGetChildByID(disp,id)
    Component arrow1 = ComponentGetChild(gp,0)
    Component arrow2 = ComponentGetChild(gp,1)
    Component no1 = ComponentGetChild(gp,2)
    Component no2 = ComponentGetChild(gp,3)
    GroupAnnotationUngroup(gp)
    ComponentGetRect(arrow1,zy,zx,v1y,v1x)
```

Appendix B. Program scripts

```
      ComponentGetRect(arrow2,zy,zx,v2y,v2x)
      gp.ComponentAddChildAtEnd(arrow1)
      gp.ComponentAddChildAtEnd(arrow2)
      gp.ComponentAddChildAtEnd(no1)
      gp.ComponentAddChildAtEnd(no2)

      SetPixel(centres,2,0,Nearest(zx))
      SetPixel(centres,3,0,Nearest(zy))
      SetPixel(centres,2,1,Nearest(v1x))
      SetPixel(centres,3,1,Nearest(v1y))
      SetPixel(centres,2,2,Nearest(v2x))
      SetPixel(centres,3,2,Nearest(v2y))
   }

   Return centres
}

Void ToCentres(Object self) {
   Image centres := GetLocations(self)
   centres.SetDisplayType(5)
   centres.ShowImage()
}

Void FromCentres(Object self) {
   Number xrange2,yrange2
   PanelToVar(self)
   FrontImage(self)

   Image centres
   If(!GetOneImage("Choose 'centres'",centres))
      Exit(0)
   GetSize(centres,xrange2,yrange2)

   For(Number i=0;i<yrange2;i++) {
      Number x,y
      x = GetPixel(centres,0,i)
      y = GetPixel(centres,1,i)
      Component gp = OneNewOvalGroup(self,i)
      ComponentAddChildAtEnd(disp,gp)
      ComponentSetControlPoint(gp,3,x,y,1)
   }

   If(xrange2 == 4 && yrange2 > 2) {
      Number zx = GetPixel(centres,2,0),zy = GetPixel(centres,3,0),\
           v1x = GetPixel(centres,2,1),v1y = GetPixel(centres,3,1),\
           v2x = GetPixel(centres,2,2),v2y = GetPixel(centres,3,2)

      Component tmp = NewBoxAnnotation(0,0,0,0)
      Component arrow1 = NewArrowAnnotation(zy,zx,v1y,v1x)
      Component no1 = NewTextAnnotation((v1x+zx)/2,(v1y+zy)/2,"V1",d/8)
           no1.ComponentSetFontFaceName("Microsoft Sans Serif")
      Component arrow2 = NewArrowAnnotation(zy,zx,v2y,v2x)
      Component no2 = NewTextAnnotation((v2x+zx)/2,(v2y+zy)/2,"V2",d/8)
           no2.ComponentSetFontFaceName("Microsoft Sans Serif")

      Component gp = NewGroupAnnotation()
           gp.ComponentAddChildAtEnd(tmp)
           gp.ComponentAddChildAtEnd(arrow1)
           gp.ComponentAddChildAtEnd(arrow2)
           gp.ComponentAddChildAtEnd(no1)
           gp.ComponentAddChildAtEnd(no2)
      ComponentRemoveFromParent(tmp)

      ComponentAddChildAtBeginning(disp,gp)
   }

   UpdateAnnotations(self,NULL)
}

Void FloatingWindow(Object self) {
   OpenGadgetPanel("Get Centres")
   self.Close()
}

Void DiameterPlus(Object self) {
   PanelToVar(self)
   self.DLGValue("Diameter",d+5)
}

Void DiameterMinus(Object self) {
   PanelToVar(self)
   self.DLGValue("Diameter",d-5)
}

Void Help(Object self) {
   String text  = ""
   text += "Copy annotations from previous pattern & Press \"Centres\"; Otherwise,\n\n"
   text += "0. \"Front Image\": \n"
   text += "  1) \"Front Image\" - When an action takes place it will\n"
   text += "     apply on the front-most Image (except \"centres\") at then;\n"
   text += "  2) \"Choose & Lock\" - Choose an Image and all actions will\n"
   text += "     always apply on the selected Image.\n\n"
   text += "1. \"Centres\" or Tag \"C\": \n"
   text += "  Press \"Draw Ovals\" and then move the ovals to select the beams;\n"
   text += "  Use \"Oval 0\" for the central beam!\n\n"
   text += "2. \"Zone Axis\" or Tag \"Z\": \n"
   text += "  Example 1. If the pattern is exactly on-zone,\n"
   text += "     then enter A = Beam 0, B = Beam 0, Number c = 2;\n\n"
   text += "  Example 2. If Zone Axis is at the middle of Beam 0 & Beam 1,\n"
   text += "     then enter A = Beam 0, B = Beam 1, Number c = 2;\n\n"
   text += "  Example 3. If Zone Axis is between Beam 0 & Beam 1 and 1/3 away from Beam 0,\n"
   text += "     then enter A = Beam 0, B = Beam 1, Number c = 3.\n\n"
   text += "3. \"Vectors\" or Tag \"V\": \n"
   text += "  If Vector 1 is from Beam 0 to Beam 1, then enter V1 = Beam 1 and X = Beam 0;\n"
   text += "  If Vector 2 is from Beam 0 to Beam 2, then enter V2 = Beam 2.\n\n"
   text += "  Press \"Draw Vectors\" to place the Vectors upon Zone Axis settings;\n"
   text += "     manually move the Vectors' Annotation afterwards\n"
   text += "     to adjust the locations of Zone Axis and Vectors.\n\n"
   text += "4. Press \"Centres\": to create the \"centres\" image and displayed as spreadsheet;\n"
   text += "  1) Locations of Centres will be stored at 1st (x) & 2nd (y) Columns;\n"
   text += "  2) Zone Axis, Vector 1 and then Vector 2 in 3rd (x) & 4th (y) Columns;\n"
   text += "  3) Numbers can be copied from the spreadsheet as tabulated text;\n"
   text += "  4) Double-click a cell in spreadsheet can change its value.\n\n"
   text += "Button \"Info\" displays Vectors' info.\n\n"
   text += "Button \"Tilts\" displays tilting coordinates (the binning factor will be asked).\n\n"
   text += "\"Misc.\" or Tag \"*\" for Misc. settings:\n"
   text += "  1) Label options - No label, or labels at ovals' corners/centres.\n"
   text += "  2) Choose colours for Shapes and Label. If \"Shape\" or \"Label\" is chosen,\n"
   text += "     then the colours will stop updating.\n"
   text += "  3) \"From 'centres'\" - Load Locations from a 'centres' Image."

   DocumentWindow help_win = NewScriptWindow("Help",100,100,800,800)
   EditorWindowAddText(help_win,text)
   WindowShow(help_win)
}

Void Info(Object self) {
   Image centres := GetLocations(self)
   Number c0x = GetPixel(centres,0,0),c0y = GetPixel(centres,1,0),\
        zx = GetPixel(centres,2,0),zy = GetPixel(centres,3,0),\
        v1x = GetPixel(centres,2,1)-zx,v1y = GetPixel(centres,3,1)-zy,\
        v2x = GetPixel(centres,2,2)-zx,v2y = GetPixel(centres,3,2)-zy
   Number v1m = Sqrt(v1x**2+v1y**2),v2m = Sqrt(v2x**2+v2y**2)
   String text = ""
   text += "\n\nInfo:\n"
   text += "Abs(Vector 1) = " + Nearest(v1m)
   text += ", Abs(Vector 2) = " + Nearest(v2m) + " (rounded)\n"
   text += "Ratio Vector 1/Vector 2 = " + v1m/v2m + "\n"
   text += "Angle = " + acos((v1x*v2x+v1y*v2y)/(v1m*v2m))/Pi()*180 + " degree\n"

   Notes(text)
   OpenResultsWindow()
}

Void Tilts(Object self) {
   Image centres := GetLocations(self)
   Number c0x = GetPixel(centres,0,0),c0y = GetPixel(centres,1,0),\
        zx = GetPixel(centres,2,0),zy = GetPixel(centres,3,0),\
        v1x = GetPixel(centres,2,1)-zx,v1y = GetPixel(centres,3,1)-zy,\
        v2x = GetPixel(centres,2,2)-zx,v2y = GetPixel(centres,3,2)-zy
   Number point1x,point1y,point2x,point2y,point3x,point3y,point4x,point4y,\
        AAA,BBB,binning
   TagGroup user_tg = GetUserPersistentTagGroup()
   TagGroupGetTagAsNumber(user_tg,"GetCentres_binning",binning)

   If(!GetNumber("What is the binning factor?",binning,binning))
      Exit(0)
   If(binning <= 0)
      Throw("Binning must be larger than 0!")

   TagGroupSetTagAsNumber(user_tg,"GetCentres_binning",binning)

   String text  = ""
   text += "\n\nTilting Coordinates (selx = sely = diameter = " + d + \
        ", binning = " + binning + "): \n"

   point4x = c0x - zx
   point4y = c0y - zy

   point1x = point4x - ((d - binning)/2)
   point1y = point4y - ((d - binning)/2)
   point2x = point4x + ((d - binning)/2)
   point2y = point4y - ((d - binning)/2)
   point3x = point4x - ((d - binning)/2)
   point3y = point4y + ((d - binning)/2)

   AAA = ((point1x*v2y) - (point1y*v2x))/((v1x*v2y) - (v1y*v2x))
   BBB = (point1y/v2y) - ((v1y/v2y)*AAA)

   text += ""+AAA+"  "+BBB+" \n"

   AAA = ((point2x*v2y) - (point2y*v2x))/((v1x*v2y) - (v1y*v2x))
   BBB = (point2y/v2y) - ((v1y/v2y)*AAA)

   text += ""+AAA+"  "+BBB+" \n"

   AAA = ((point3x*v2y) - (point3y*v2x))/((v1x*v2y) - (v1y*v2x))
   BBB = (point3y/v2y) - ((v1y/v2y)*AAA)

   text += ""+AAA+"  "+BBB+" \n"

   AAA = ((point4x*v2y) - (point4y*v2x))/((v1x*v2y) - (v1y*v2x))
   BBB = (point4y/v2y) - ((v1y/v2y)*AAA)

   text += ""+AAA+"  "+BBB+" \n"

   Notes(text)
   OpenResultsWindow()
 }
}

//       Panel function ends
//
////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////
//
```

Appendix B. Program scripts

```
//      Main function starts

Object float_win = alloc(Panel_UI).Init(Panel(0))
UnregisterScriptPalette("Get Centres")
RegisterScriptPalette(float_win,"","Get Centres")

Object modeless_win = alloc(Panel_UI).Init(Panel(1))
modeless_win.Display("Get Centres")

//      Main function ends
//
////////////////////////////////////////////////////////////////
```

B.4. *bedit* – a Shell-script-based bulk files editing script

```
#!/bin/bash
editortitle="Batch Text File Editor for Series Folders"
editorversion="v 1.2"
#  2017/11/29: v 1.2.1.  awk variables whitespace syntax
#  2017/09/21: v 1.2. "Word replace" added
#  2015/11/20: v 1.1. Bug fixed
#  2014/09/08: v 1.0. Basic functions



#
#  Function of Load settings from ./confs.txt
#

Updateconfs(){

#  Update variables in confs.txt if specified as input

   for sp in foldermod foldername mfno mlno list file; do
      eval spo="\${$sp}"
      if [ -n "$spo" ]; then
         if grep -q "$sp" confs.txt; then
            sed -i "/$sp/c\\$sp=$spo" confs.txt
         else
            echo -e "\n$sp=$spo" >> confs.txt
         fi
      fi
   done

#  Load all variables from confs.txt

   unset foldermod list foldername ffolder lfolder mfno mlno lfno llno
   if [ -f confs.txt ]; then
      . confs.txt
      if [ "$foldermod" = L ]; then
         if [ -f "$list" ]; then
            sed -i '$a\' $list
            sed -i '/^[[:space:]]*$/d' $list
            lfno=1; llno=`sed -n '$=' $list`
            ffolder=$(awk -v line="$lfno" 'NR==line{print $1}' $list)
            lfolder=$(awk -v line="$llno" 'NR==line{print $1}' $list)
         else
            warnmsg=$warnmsg"\033[41m List file '$list' NOT exists! \033[0m \n"
         fi
      elif [ "$foldermod" = "M" ]; then
         ffolder="$foldername$mfno"
         lfolder="$foldername$mlno"
      fi
   fi
}


#########################################################
#
#     File Editing Functions

#  Add a line
AddLine (){
   data0=`awk -v line="${pe[0]}" 'NR==line{print $0}' $file`
   echo -e "Add line '${pe[1]}' \033[33mbefore\033[0m '$data0'"
   sed -i$backup "${pe[0]} i\\${pe[1]}" $file
}

#  Replace a line
ReplaceLine (){
   data0=`awk -v line="${pe[0]}" 'NR==line{print $0}' $file`
   echo -e "'$data0' \033[33m->\033[0m '${pe[1]}'"
   sed -i$backup "${pe[0]} c\\${pe[1]}" $file
}

#  Delete lines
DeleteLine (){
   data0=`awk -v "line=${pe[0]}" 'NR==line{print $0}' $file`
   data1=`awk -v "line=${pe[1]}" 'NR==line{print $0}' $file`
   echo -e "Delete lines from '$data0' \033[33mto\033[0m '$data1'"
   sed -i$backup "${pe[0]},${pe[1]} d" $file
}

#  Print line
PrintLine () {
   echo -e "$(awk -v "line=${pe[0]}" 'NR==line{print $0}' $file)"
}

#  Word replace
WordReplace (){
   cp $file $file$backup
   data0=`awk -v "line=${pe[0]}" -v word="${pe[1]}" 'NR==line{print $word}' $file$backup`
   echo -e "'$data0' \033[33m->\033[0m '${pe[2]}'"
   awk -v line="${pe[0]}" -v word="${pe[1]}" -v val="${pe[2]}" 'NR==line{$word = val} 1'
$file$backup > $file
}

#  Spreadsheet Input
SpreadsheetInput() {
   linei=${pe[0]}
   cp $file $file$backup
   echo -e ""
   for (( wordi=${pe[2]}; wordi <= ${pe[3]}; ++wordi )); do
      data=""

      for (( coli=1; coli <= ${pe[1]}; ++coli )); do
         data0=`awk -v line="$linei" 'NR==line{print $0}' $file`
         data1=`awk -v line="$noi" -v field="$wordi" 'NR==line{print $field}' ../$list`

         if [ $coli = 1 ]; then data=$data1; else data="$data  $data1"; fi
         let wordi=$wordi+1
         if [ $wordi -gt ${pe[3]} ]; then break; fi
      done
      let wordi=$wordi-1

      sed -i "$linei c\\$data" $file
      echo -e "#L $linei, '$data0' \033[31m->\033[0m '$data'"

      let linei=$linei+1
   done
}

#  Restore Backups
RestoreBak (){
   echo -e "Restore from '$file$backup'"
   cp $file$backup $file
}


#     File Editing Functions end
#
#########################################################


#
#  Editor Menu
#

EditorMenu() {
   echo -e "\n"
   #echo -e "Display help: "
   echo -e  "\033[45m    $editortitle $editorversion     \033[0m\033[35m\033[0m"
   emenu[1]="    Line Edit    | Parameter Edit | Subfolders | Others    "
   emenu[2]=" [R]eplace [A]dd | [W]ord replace |  [M]anual  | [F]ile    "
   emenu[3]=" [P]rint   [D]el | [S]preadsheet  |  [L]ist    | [U]ndo    "

   for (( mi=1 ; mi<4 ; mi++ )); do
      emenu[$mi]=$(sed
's/\[/\\033[32m\[\\033[0m/g;s/\]/\\033[32m\]\\033[0m/g;s/|/\\033[35m|\\033[0m/g' <<<
"${emenu[$mi]}")
      echo -e "${emenu[$mi]}"
   done

#  Current directory
   dir=`echo "$(pwd)/" | sed 's\/\/\\\033[36m\/\\\033[0m/g`
#  Briefly Show Targeted Folders and File
   echo -ne "\033[35mTargets:\033[0m $dir"
   if [ -n "$foldermod" ]; then
      echo -ne "\033[36m{\033[32m$foldermod:\033[0m "
      if [ "$foldermod" = L ]; then
         echo -ne "\033[36m(\033[0m$list\033[36m:\033[0m $llno \033[36mfolders)\033[0m "
      fi
      echo -ne "\033[36m\033[0m$ffolder \033[36m-\033[0m $lfolder\033[36m}/\033[0m"
   fi

   if [ -n $file ]; then
      echo -e "$file"
   else
      echo -e "\033[0;41mFile unknown!\033[0m"
   fi

#  Show warning if any
   if [ -n "$warnmsg" ]; then echo -e "$warnmsg"; fi
   unset warnmsg

#  Show messages
   echo -ne "\033[35mNow - \033[0m"
   if [ -n "$editname" ]; then echo -e "\033[44m $editname \033[0m" ; fi
   for (( mi=1 ; mi<11 ; mi++ )); do
      if [ -n "${msg[$mi]}" ]; then
         msg[$mi]=$(sed
's/\[/\\033[32m\[\\033[0m/g;s/\]/\\033[32m\]\\033[0m/g;s|\;|\\033[36m\;\\033[0m|g' <<<
${msg[$mi]})
         if [ "$mi" = 10 ]; then
            echo -ne "${msg[10]}"
         else
            echo -e "${msg[$mi]}"
         fi
      fi
   done
```

Appendix B. Program scripts

```
}


#
#  Editor Main Body
#

EditorBody() {
   while :; do
      while :; do
         unset msg

         case "$ecode" in
            R"|"A"|"D")
               case "$ecode" in
                  R)
                     editcmd="ReplaceLine"; editname="Line Replace"
                     msg[1]="Replace Line #L with CONTENT:"
                     msg[2]="#L;CONTENT"  ;;
                  A)
                     editcmd="AddLine"; editname="Line Addition"
                     msg[1]="Add a new line with CONTENT before Line #L:"
                     msg[2]="#L;CONTENT"  ;;
                  D)
                     editcmd="DeleteLine"; editname="Line Deletion"
                     msg[1]="Delete Lines from #L1 to #L2:"
                     msg[2]="#L1;#L2"  ;;
               esac
               if [ "${pe[1]}" != "" ]; then EditingLoop; fi  ;;
            "P")
               editcmd="PrintLine"; editname="Print Line"
               msg[10]="Enter Line #: "
               if [ "${pe[0]}" != "" ]; then EditingLoop; fi  ;;
            "W")
               editcmd="WordReplace"; editname="Word Replace"
               msg[1]="On Line #L, replace the #W Word with VALUE:"
               msg[2]="#L;#W;VALUE"
               if [ "${pe[2]}" != "" ]; then EditingLoop; fi  ;;
            "S")
               if [ "$foldermod" != L ]; then
                  warnmsg=$warnmsg"Spreadsheet Input only available with List Subfolder Mode! \n"
                  ecode="L"; break
               fi
               editcmd="SpreadsheetInput"; editname="Parameters from Spreadsheet"
               msg[1]="Parameters from Spreadsheet"
               msg[2]="Target - '$file' starting at Line #L, #N words per line"
               msg[3]="Source - '$list' Columns #C1 to #C2 "
               msg[4]="#L;#N;#C1;#C2"
               if [ "${pe[3]}" != "" ]; then EditingLoop; fi  ;;
            "U")
               editname="Restore"; editcmd="RestoreBak"; EditingLoop; ecode=$lecode; break  ;;
         esac
         unset editcmd

         case "$ecode" in
            "F")
               editname="Set Editing File"
               msg[10]="Enter filename ([FF] for 'FOR007.DAT'): "
               if [ "${#pe[@]}" = 1 ]; then file=${pe[0]}; ecode=$lecode; break; fi  ;;
            "FF")
               file="FOR007.DAT"; ecode=$lecode; break ;;
            "L")
               editname="List Folder Mode"; foldermod=L
               msg[10]="Enter List filename ([LL] for list.txt): "
               if [ "${#pe[@]}" = 1 ]; then list=${pe[0]}; ecode=$lecode; break; fi  ;;
            "LL")
               list="list.txt"; foldermod=L; ecode=$lecode; break  ;;
            "M")
               editname="Manual Folder Mode"; foldermod=M
               msg[1]="Specify folders' Base-name, Starting and Last # ([MM] for Set1 - Set200):"
               msg[2]="Base-name;1st #;Last #"
               if [ "${#pe[@]}" = 3 ]; then foldername=${pe[0]}; mfno=${pe[1]}; mlno=${pe[2]};
break; fi  ;;
            "MM")
               foldername=Set ; mfno=1 ; mlno=200 ; foldermod=M; break  ;;
         esac
         unset pe
         Updateconfs
         Debug 1

         EditorMenu

         IFS=';' read -a r
         r0=`echo ${r[0]} | tr [:lower:] [:upper:]`

         case "$r0" in
            R|A|D|P|W|S|M|L|F|U|FF|LL|MM)
               ecode=$r0; pe=( "${r[@]:1}" )
               case "$ecode" in
                  R|A|D|P|W|S) lecode=$ecode  ;;
               esac
               ;;
            "") ecode=$lecode; editname=""; break  ;;
            *) pe=( "${r[@]}" )  ;;
         esac
         Debug 2
      done

      Debug 3
      unset pe editcmd ecode editname
   done
}

Debug (){
   if [ "$debug_check" = 1 ]; then
      echo -ne "\033[36mDebug $1:\033[0m ecode=$ecode; lecode=$lecode; editcmd=$editcmd;
r0=$r0; "
      for pi in "${!pe[@]}"; do echo -ne "pe[$pi]='${pe[$pi]}'  "; done
      echo -e ""
   fi
}


#
#  Batch File Editing Loop
#

EditingLoop() {
   if [ -z $foldermod ]; then ecode=""; warnmsg=$warnmsg"Folders unknown! \n"; break
   elif [ -z $file ]; then code=""; warnmsg=$warnmsg"File unknown! \n"; break
   elif [ -z $editcmd ]; then ecode=""; warnmsg=$warnmsg"Unknown command! Empty editcmd!
\n"; break
   fi

#  Looping starts
   echo -e "\n\033[32m$editname\033[0m start:"

   unset fno lno noi unfolder unfo numfolder

   if [ "$foldermod" = M ]; then
      fno=$mfno; lno=$mlno
   else
      fno=$lfno; lno=$llno
   fi

   for (( noi=$fno; noi <= $lno; ++noi )); do
      if [ "$foldermod" = M ]; then subfolder="$foldername$noi"
      else subfolder=$(awk -v line="$noi" 'NR==line{print $1}' $list)
      fi

      if [ -f $subfolder/$file ]; then
         cd $subfolder
         echo -ne "\033[35m$noi/$lno\033[0m \033[36m$subfolder\033[0m: "
         $editcmd
         cd ../
         let numfolder=$numfolder+1
      elif [[ "$foldermod" = "L" || -d "$subfolder" ]]; then
         echo -e "\033[35m$noi/$lno\033[0m \033[36m$subfolder\033[0m: \033[41m$file'
UNFOUND! \033[0m"
         unfolder=$unfolder"$subfolder, "
         let unfo=$unfo+1
      fi
   done

#  Looping finishes

   echo -ne "\n\033[32m$editname\033[0m "
   if [ "$ecode" != U ]; then echo -ne "on Line ${pe[0]} "; fi
   echo -e "completed, $numfolder folders processed."
   if [ -n "$unfo" ]; then
      echo -e "$unfo file(s) unfound: $unfolder"
   fi
}


#########################################################
#
#     Editor starts

lecode=""; ecode=""; backup=".bak"; warnmsg=""; list="list.txt"
debug_check=1
if [ -n "$1" ]; then
   file=$1; Updateconfs
fi

EditorBody

file=""

#     Exits Editor
#
#########################################################
```

Appendix C. Revised tables of RHF electron atomic scattering factors

Some values in the tables of electron atomic scattering factors from the latest version of the International Tables for Crystallography [56] appears to be incorrect. They either differ from the original publication [57], or far depart from converted values from the corresponding X-ray atomic scattering factors as well as misfits to the trends of neighbouring scattering angles and elements. Such values are assumed to be erroneous caused by inaccurate optical character recognition (OCR). Values from the original publication are copied if they are available. If no original publication value is aviable, assumptions will be made replacing 1-2 digits that might be mistaken by OCR (like 0, 3, 6 & 8 etc.); the corrected values satisfy smaller differences than 0.5% to the converted values from the corresponding X-ray atomic scattering factors. The list and notations of the corrected values are:

4. Corrected to values from [57] and denoted as *:
    a. 27. Co – 0.30 = 2.471;
5. Corrected to assumed values and denoted as †:
    a. 52. Te – 0.24 = 5.291,
    b. 57. Ba – 1.50 = 0.456,
    c. 57. La – 1.30 = 0.591,
    d. 62. Sm – 0.40 = 3.506,
    e. 64. Gd – 1.40 = 0.583,
    f. 67. Ho – 0.70 = 1.805,
    g. 85. At – 0.22 = 7.474, and
    h. 98. Cf – 1.70 = 0.615.

The values are in unit of Å.

Appendix C. Revised tables of RHF electron atomic scattering factors

C.1. Revised tables of neutral atoms 1 – 20

s (Å⁻¹)	1. H	2. He	3. Li	4. Be	5. B	6. C	7. N	8. O	9. F	10. Ne	11. Na	12. Mg	13. Al	14. Si	15. P	16. S	17. Cl	18. Ar	19. K	20. Ca
0.00	0.529	0.418	3.286	3.052	2.794	2.509	2.211	1.983	1.801	1.652	4.778	5.207	5.889	5.828	5.488	5.161	4.857	4.580	8.984	9.913
0.01	-	0.418	3.265	3.042	2.788	2.505	2.209	1.982	1.800	1.651	4.749	5.187	5.867	5.810	5.476	5.152	4.851	4.576	8.921	9.860
0.02	-	0.417	3.200	3.011	2.768	2.492	2.201	1.976	1.796	1.648	4.663	5.124	5.800	5.759	5.439	5.124	4.830	4.559	8.731	9.699
0.03	-	0.415	3.097	2.961	2.736	2.471	2.187	1.966	1.789	1.642	4.527	5.022	5.692	5.675	5.378	5.079	4.795	4.531	8.434	9.442
0.04	0.51	0.413	2.961	2.892	2.693	2.442	2.168	1.953	1.779	1.635	4.348	4.884	5.547	5.561	5.296	5.016	4.746	4.493	8.054	9.104
0.05	0.51	0.410	2.800	2.807	2.638	2.406	2.144	1.937	1.767	1.626	4.138	4.717	5.371	5.421	5.192	4.938	4.685	4.444	7.619	8.703
0.06	0.50	0.407	2.622	2.710	2.574	2.363	2.116	1.917	1.752	1.615	3.908	4.527	5.170	5.258	5.071	4.845	4.613	4.386	7.157	8.258
0.07	0.49	0.404	2.435	2.601	2.502	2.313	2.083	1.893	1.735	1.602	3.667	4.320	4.949	5.077	4.935	4.740	4.529	4.320	6.691	7.789
0.08	0.48	0.399	2.245	2.484	2.423	2.259	2.047	1.867	1.716	1.587	3.425	4.102	4.717	4.882	4.785	4.623	4.436	4.245	6.239	7.312
0.09	0.47	0.395	2.058	2.362	2.339	2.200	2.007	1.839	1.694	1.570	3.190	3.879	4.478	4.677	4.625	4.496	4.335	4.163	5.815	6.841
0.10	0.45	0.390	1.879	2.237	2.250	2.138	1.963	1.808	1.671	1.552	2.967	3.656	4.237	4.467	4.457	4.362	4.227	4.074	5.426	6.388
0.11	0.44	0.384	1.710	2.111	2.159	2.072	1.918	1.774	1.646	1.533	2.759	3.437	3.999	4.255	4.285	4.222	4.113	3.980	5.073	5.959
0.12	0.425	0.378	1.554	1.987	2.067	2.005	1.870	1.739	1.619	1.512	2.569	3.226	3.767	4.043	4.109	4.078	3.994	3.881	4.756	5.560
0.13	0.411	0.372	1.411	1.865	1.974	1.936	1.821	1.702	1.591	1.490	2.395	3.025	3.544	3.835	3.933	3.931	3.871	3.779	4.474	5.192
0.14	0.396	0.366	1.282	1.748	1.882	1.866	1.770	1.664	1.562	1.467	2.239	2.835	3.330	3.632	3.758	3.783	3.746	3.674	4.222	4.855
0.15	0.382	0.359	1.166	1.635	1.791	1.796	1.718	1.625	1.532	1.443	2.099	2.657	3.128	3.437	3.586	3.635	3.620	3.566	3.997	4.550
0.16	0.366	0.352	1.063	1.528	1.702	1.727	1.666	1.585	1.501	1.418	1.974	2.492	2.938	3.249	3.417	3.487	3.493	3.458	3.795	4.273
0.17	0.353	0.345	0.971	1.427	1.616	1.658	1.614	1.545	1.469	1.393	1.863	2.340	2.760	3.070	3.253	3.342	3.367	3.348	3.612	4.023
0.18	0.338	0.338	0.889	1.332	1.533	1.591	1.561	1.504	1.436	1.367	1.763	2.199	2.595	2.900	3.094	3.200	3.242	3.239	3.446	3.797
0.19	0.324	0.330	0.817	1.243	1.453	1.524	1.510	1.463	1.404	1.340	1.674	2.071	2.441	2.740	2.942	3.061	3.118	3.130	3.295	3.593
0.20	0.311	0.323	0.753	1.161	1.377	1.460	1.458	1.422	1.371	1.313	1.594	1.953	2.299	2.589	2.796	2.927	2.997	3.022	3.154	3.408
0.22	0.285	0.308	0.646	1.013	1.235	1.337	1.358	1.341	1.304	1.259	1.458	1.748	2.046	2.315	2.525	2.671	2.763	2.811	2.902	3.086
0.24	0.261	0.293	0.562	0.887	1.107	1.222	1.262	1.261	1.238	1.204	1.344	1.577	1.832	2.076	2.281	2.436	2.543	2.609	2.680	2.815
0.25	0.249	0.286	0.526	0.832	1.048	1.168	1.216	1.222	1.206	1.176	1.295	1.502	1.737	1.969	2.169	2.326	2.438	2.512	2.578	2.695
0.26	0.238	0.278	0.494	0.781	0.993	1.117	1.171	1.184	1.173	1.149	1.249	1.434	1.650	1.869	2.064	2.221	2.337	2.417	2.481	2.584
0.28	0.218	0.264	0.440	0.690	0.892	1.020	1.085	1.110	1.110	1.095	1.167	1.313	1.495	1.689	1.872	2.026	2.148	2.238	2.299	2.383
0.30	0.199	0.250	0.396	0.614	0.803	0.932	1.006	1.040	1.049	1.043	1.095	1.211	1.363	1.534	1.702	1.851	1.974	2.070	2.134	2.206
0.32	0.182	0.236	0.359	0.549	0.725	0.853	0.932	0.974	0.991	0.991	1.031	1.123	1.251	1.400	1.553	1.694	1.816	1.915	1.982	2.048
0.34	0.167	0.224	0.328	0.494	0.657	0.781	0.863	0.911	0.935	0.942	0.973	1.047	1.154	1.284	1.422	1.554	1.672	1.772	1.842	1.905
0.35	0.160	0.217	0.314	0.469	0.625	0.748	0.831	0.881	0.908	0.918	0.946	1.013	1.111	1.231	1.362	1.490	1.606	1.705	1.776	1.838
0.36	0.153	0.211	0.301	0.446	0.596	0.717	0.800	0.853	0.882	0.894	0.921	0.980	1.070	1.182	1.306	1.429	1.542	1.641	1.714	1.775
0.38	0.141	0.200	0.279	0.406	0.543	0.658	0.742	0.798	0.831	0.849	0.872	0.921	0.997	1.094	1.205	1.318	1.425	1.522	1.595	1.657
0.40	0.130	0.189	0.259	0.371	0.497	0.606	0.689	0.747	0.784	0.805	0.827	0.868	0.932	1.017	1.115	1.218	1.319	1.412	1.487	1.548
0.42	0.120	0.178	0.241	0.341	0.455	0.559	0.641	0.700	0.739	0.764	0.785	0.821	0.875	0.949	1.036	1.130	1.224	1.313	1.387	1.449
0.44	0.111	0.169	0.226	0.314	0.419	0.517	0.596	0.656	0.697	0.725	0.746	0.777	0.825	0.888	0.965	1.051	1.138	1.223	1.295	1.357
0.45	0.107	0.164	0.219	0.302	0.402	0.497	0.575	0.635	0.677	0.706	0.727	0.757	0.801	0.861	0.933	1.014	1.098	1.181	1.252	1.314
0.46	0.103	0.159	0.212	0.291	0.387	0.479	0.555	0.615	0.658	0.687	0.709	0.738	0.779	0.834	0.903	0.980	1.061	1.141	1.211	1.272
0.48	0.096	0.151	0.200	0.271	0.358	0.444	0.518	0.577	0.621	0.652	0.675	0.701	0.737	0.786	0.847	0.917	0.991	1.066	1.134	1.194
0.50	0.089	0.143	0.188	0.253	0.333	0.413	0.484	0.542	0.586	0.619	0.642	0.667	0.700	0.743	0.797	0.860	0.928	0.998	1.064	1.123
0.55	0.075	0.125	0.164	0.215	0.280	0.348	0.411	0.466	0.510	0.544	0.569	0.592	0.618	0.651	0.692	0.741	0.796	0.854	0.912	0.966
0.60	0.064	0.110	0.145	0.186	0.239	0.297	0.353	0.403	0.445	0.479	0.505	0.528	0.551	0.578	0.610	0.648	0.692	0.740	0.790	0.838
0.65	0.055	0.097	0.128	0.164	0.207	0.256	0.305	0.350	0.390	0.424	0.450	0.473	0.494	0.517	0.543	0.573	0.609	0.648	0.690	0.733
0.70	0.048	0.086	0.115	0.145	0.182	0.223	0.266	0.307	0.344	0.376	0.403	0.425	0.445	0.465	0.487	0.513	0.541	0.574	0.609	0.647
0.80	0.037	0.068	0.093	0.117	0.144	0.175	0.208	0.241	0.272	0.300	0.325	0.347	0.366	0.383	0.401	0.419	0.440	0.462	0.488	0.515
0.90	0.029	0.055	0.077	0.096	0.118	0.141	0.167	0.193	0.219	0.244	0.266	0.286	0.304	0.320	0.335	0.350	0.366	0.383	0.402	0.422
1.00	0.024	0.046	0.064	0.081	0.098	0.117	0.137	0.159	0.180	0.201	0.221	0.239	0.255	0.270	0.284	0.298	0.311	0.324	0.339	0.354
1.10	0.020	0.038	0.054	0.069	0.083	0.099	0.115	0.133	0.150	0.168	0.185	0.202	0.217	0.231	0.243	0.255	0.267	0.278	0.290	0.303
1.20	0.017	0.032	0.046	0.059	0.072	0.085	0.098	0.113	0.128	0.143	0.158	0.172	0.185	0.198	0.210	0.221	0.232	0.242	0.252	0.262
1.30	0.014	0.028	0.040	0.051	0.062	0.073	0.085	0.097	0.110	0.123	0.135	0.148	0.160	0.172	0.183	0.193	0.202	0.212	0.220	0.230
1.40	0.012	0.024	0.035	0.045	0.055	0.064	0.074	0.085	0.095	0.106	0.117	0.129	0.139	0.150	0.160	0.169	0.178	0.187	0.194	0.202
1.50	0.011	0.021	0.031	0.040	0.048	0.057	0.065	0.074	0.084	0.093	0.103	0.113	0.123	0.132	0.141	0.150	0.158	0.166	0.174	0.181
1.60	-	0.019	0.028	0.035	0.043	0.051	0.058	0.066	0.074	0.083	0.092	0.100	0.109	0.117	0.125	0.133	0.141	0.148	0.156	0.162
1.70	-	0.016	0.024	0.031	0.038	0.045	0.052	0.059	0.066	0.074	0.081	0.089	0.096	0.104	0.111	0.119	0.126	0.132	0.138	0.144
1.80	-	0.015	0.022	0.028	0.035	0.041	0.047	0.053	0.060	0.066	0.073	0.080	0.087	0.093	0.100	0.107	0.113	0.119	0.127	0.132
1.90	-	0.013	0.019	0.026	0.031	0.037	0.043	0.048	0.054	0.060	0.065	0.072	0.078	0.084	0.090	0.096	0.102	0.108	0.112	0.118
2.00	-	0.012	0.017	0.023	0.028	0.034	0.039	0.044	0.049	0.054	0.059	0.065	0.070	0.076	0.082	0.087	0.093	0.098	0.101	0.107
2.50	-	0.008	0.011	0.015	0.019	0.022	0.026	0.029	0.032	0.036	0.039	0.043	0.046	0.050	0.053	0.057	0.060	0.064	0.065	0.071
3.00	-	0.005	0.008	0.011	0.013	0.016	0.018	0.020	0.023	0.025	0.028	0.030	0.032	0.035	0.037	0.040	0.042	0.045	0.047	0.050
3.50	-	0.004	0.006	0.008	0.010	0.012	0.013	0.015	0.017	0.019	0.021	0.022	0.024	0.026	0.028	0.030	0.031	0.033	0.035	0.037
4.00	-	0.003	0.004	0.006	0.007	0.009	0.010	0.012	0.013	0.015	0.016	0.017	0.019	0.020	0.021	0.023	0.024	0.026	0.027	0.028
5.00	-	0.002	0.003	0.004	0.005	0.006	0.007	0.008	0.009	0.009	0.010	0.011	0.012	0.013	0.014	0.015	0.016	0.017	0.018	0.018
6.00	-	0.001	0.002	0.003	0.003	0.004	0.005	0.005	0.006	0.007	0.007	0.008	0.009	0.009	0.010	0.010	0.011	0.012	0.012	0.013

Appendix C. Revised tables of RHF electron atomic scattering factors

C.2. Revised tables of neutral atoms 21 – 40

s (Å⁻¹)	21. Sc	22. Ti	23. V	24. Cr	25. Mn	26. Fe	27. Co	28. Ni	29. Cu	30. Zn	31. Ga	32. Ge	33. As	34. Se	35. Br	36. Kr	37. Rb	38. Sr	39. Y	40. Zr
0.00	9.307	8.776	8.305	6.969	7.506	7.165	6.854	6.569	5.600	6.065	7.108	7.378	7.320	7.205	7.060	6.897	11.778	13.109	12.674	12.166
0.01	9.264	8.740	8.274	6.945	7.484	7.145	6.836	6.552	5.587	6.051	7.088	7.359	7.306	7.192	7.049	6.889	11.699	13.035	-	-
0.02	9.134	8.631	8.180	6.875	7.412	7.081	6.779	6.501	5.547	6.009	7.027	7.303	7.260	7.154	7.016	6.861	11.460	12.816	-	-
0.03	8.926	8.455	8.029	6.762	7.296	6.978	6.687	6.418	5.482	5.941	6.927	7.211	7.184	7.090	6.962	6.814	11.088	12.468	-	-
0.04	8.649	8.220	7.826	6.610	7.140	6.839	6.562	6.306	5.395	5.849	6.792	7.088	7.081	7.004	6.888	6.750	10.613	12.013	11.79	11.41
0.05	8.318	7.937	7.581	6.427	6.949	6.669	6.410	6.169	5.287	5.735	6.629	6.935	6.953	6.895	6.795	6.670	10.073	11.476	11.34	11.04
0.06	7.946	7.618	7.303	6.221	6.732	6.474	6.234	6.010	5.165	5.603	6.441	6.759	6.803	6.767	6.684	6.574	9.504	10.888	10.84	10.62
0.07	7.548	7.274	7.002	5.997	6.493	6.260	6.040	5.834	5.029	5.457	6.236	6.562	6.634	6.621	6.558	6.464	8.934	10.273	10.31	10.15
0.08	7.139	6.917	6.686	5.764	6.241	6.032	5.834	5.646	4.886	5.299	6.017	6.351	6.449	6.460	6.418	6.341	8.385	9.655	9.77	9.68
0.09	6.729	6.556	6.365	5.527	5.981	5.796	5.619	5.449	4.737	5.133	5.792	6.129	6.253	6.288	6.266	6.207	7.872	9.052	9.23	9.20
0.10	6.328	6.199	6.045	5.291	5.719	5.558	5.401	5.249	4.585	4.962	5.564	5.902	6.048	6.105	6.104	6.064	7.402	8.478	8.70	8.72
0.11	5.944	5.853	5.732	5.061	5.459	5.320	5.182	5.048	4.434	4.790	5.337	5.672	5.838	5.916	5.935	5.913	6.976	7.940	8.20	8.26
0.12	5.580	5.522	5.430	4.838	5.206	5.087	4.967	4.848	4.285	4.618	5.113	5.442	5.625	5.722	5.760	5.755	6.593	7.443	7.722	7.818
0.13	5.239	5.209	5.142	4.625	4.962	4.861	4.758	4.654	4.139	4.449	4.896	5.217	5.411	5.525	5.580	5.593	6.248	6.988	7.278	7.400
0.14	4.924	4.916	4.871	4.423	4.728	4.644	4.555	4.465	3.998	4.283	4.686	4.996	5.200	5.328	5.399	5.428	5.938	6.575	6.865	7.007
0.15	4.633	4.643	4.616	4.231	4.506	4.436	4.361	4.283	3.862	4.123	4.486	4.783	4.992	5.132	5.217	5.260	5.658	6.200	6.485	6.640
0.16	4.366	4.390	4.378	4.051	4.297	4.240	4.177	4.110	3.731	3.969	4.295	4.578	4.789	4.938	5.036	5.092	5.403	5.862	6.136	6.299
0.17	4.122	4.157	4.158	3.882	4.100	4.054	4.002	3.944	3.607	3.822	4.114	4.382	4.593	4.749	4.857	4.925	5.170	5.555	5.816	5.983
0.18	3.899	3.943	3.953	3.723	3.916	3.880	3.836	3.788	3.488	3.681	3.942	4.195	4.404	4.564	4.680	4.759	4.954	5.278	5.523	5.689
0.19	3.695	3.745	3.763	3.574	3.743	3.716	3.681	3.640	3.375	3.547	3.781	4.017	4.222	4.384	4.507	4.595	4.754	5.025	5.254	5.419
0.20	3.509	3.564	3.588	3.434	3.583	3.562	3.534	3.500	3.267	3.421	3.629	3.849	4.048	4.211	4.339	4.434	4.566	4.794	5.008	5.168
0.22	3.183	3.242	3.276	3.179	3.292	3.284	3.267	3.245	3.067	3.186	3.352	3.541	3.724	3.884	4.017	4.123	4.224	4.387	4.570	4.721
0.24	2.906	2.967	3.006	2.953	3.039	3.039	3.032	3.018	2.885	2.977	3.108	3.268	3.433	3.585	3.718	3.829	3.916	4.039	4.195	4.333
0.25	2.783	2.844	2.885	2.849	2.924	2.928	2.924	2.914	2.800	2.880	2.997	3.143	3.299	3.446	3.578	3.690	3.773	3.882	4.027	4.158
0.26	2.669	2.730	2.772	2.750	2.817	2.824	2.823	2.816	2.719	2.789	2.892	3.026	3.172	3.314	3.443	3.556	3.636	3.735	3.869	3.995
0.28	2.462	2.523	2.568	2.568	2.620	2.632	2.637	2.636	2.568	2.620	2.701	2.813	2.940	3.069	3.192	3.303	3.382	3.465	3.583	3.697
0.30	2.281	2.341	2.386	2.403	2.445	2.461	2.471*	2.474	2.428	2.468	2.531	2.623	2.733	2.849	2.963	3.071	3.149	3.224	3.329	3.433
0.32	2.119	2.178	2.225	2.252	2.288	2.308	2.321	2.328	2.299	2.329	2.379	2.455	2.548	2.651	2.757	2.858	2.936	3.007	3.101	3.196
0.34	1.974	2.032	2.079	2.114	2.146	2.168	2.184	2.195	2.180	2.203	2.242	2.304	2.384	2.475	2.570	2.665	2.742	2.810	2.895	2.982
0.35	1.906	1.964	2.011	2.049	2.080	2.104	2.121	2.133	2.123	2.144	2.179	2.235	2.308	2.393	2.484	2.575	2.651	2.718	2.799	2.883
0.36	1.842	1.899	1.947	1.987	2.017	2.042	2.060	2.073	2.069	2.087	2.119	2.169	2.237	2.316	2.402	2.490	2.564	2.630	2.708	2.789
0.38	1.722	1.778	1.826	1.870	1.899	1.925	1.946	1.962	1.965	1.980	2.006	2.048	2.105	2.173	2.250	2.330	2.402	2.466	2.538	2.613
0.40	1.612	1.668	1.716	1.761	1.790	1.818	1.841	1.858	1.868	1.882	1.903	1.938	1.986	2.045	2.113	2.186	2.254	2.315	2.383	2.452
0.42	1.511	1.566	1.614	1.660	1.690	1.719	1.743	1.763	1.777	1.790	1.808	1.837	1.878	1.929	1.989	2.055	2.119	2.178	2.241	2.305
0.44	1.418	1.472	1.520	1.567	1.597	1.628	1.653	1.674	1.691	1.704	1.720	1.745	1.780	1.824	1.877	1.936	1.995	2.052	2.111	2.171
0.45	1.374	1.428	1.476	1.523	1.553	1.584	1.610	1.631	1.651	1.663	1.679	1.702	1.734	1.776	1.825	1.881	1.938	1.993	2.049	2.108
0.46	1.332	1.385	1.433	1.480	1.511	1.542	1.569	1.591	1.611	1.624	1.639	1.661	1.691	1.729	1.775	1.828	1.883	1.936	1.991	2.047
0.48	1.252	1.305	1.352	1.399	1.431	1.462	1.490	1.513	1.535	1.549	1.563	1.583	1.608	1.642	1.683	1.730	1.780	1.830	1.881	1.934
0.50	1.179	1.230	1.277	1.323	1.356	1.388	1.416	1.440	1.464	1.478	1.492	1.510	1.533	1.562	1.598	1.640	1.686	1.733	1.780	1.829
0.55	1.018	1.067	1.111	1.155	1.189	1.222	1.251	1.277	1.303	1.319	1.334	1.349	1.367	1.389	1.416	1.447	1.483	1.522	1.562	1.603
0.60	0.885	0.930	0.973	1.014	1.047	1.080	1.110	1.136	1.163	1.181	1.197	1.212	1.228	1.245	1.266	1.290	1.319	1.350	1.383	1.417
0.65	0.775	0.816	0.856	0.894	0.927	0.959	0.988	1.015	1.041	1.061	1.078	1.093	1.108	1.124	1.141	1.160	1.182	1.208	1.235	1.263
0.70	0.684	0.721	0.757	0.792	0.824	0.854	0.883	0.909	0.935	0.955	0.973	0.989	1.004	1.019	1.034	1.050	1.068	1.089	1.111	1.135
0.80	0.544	0.573	0.602	0.631	0.659	0.686	0.712	0.737	0.761	0.781	0.800	0.817	0.832	0.847	0.860	0.873	0.887	0.902	0.918	0.935
0.90	0.444	0.467	0.490	0.514	0.538	0.561	0.583	0.605	0.626	0.646	0.665	0.681	0.697	0.711	0.725	0.737	0.749	0.762	0.774	0.787
1.00	0.371	0.389	0.408	0.427	0.446	0.466	0.485	0.504	0.523	0.541	0.558	0.574	0.589	0.603	0.616	0.628	0.640	0.651	0.662	0.673
1.10	0.316	0.330	0.345	0.361	0.377	0.393	0.409	0.425	0.442	0.457	0.473	0.488	0.502	0.515	0.528	0.540	0.551	0.562	0.572	0.582
1.20	0.273	0.285	0.297	0.310	0.323	0.336	0.350	0.364	0.378	0.391	0.405	0.418	0.431	0.444	0.456	0.467	0.478	0.488	0.498	0.507
1.30	0.239	0.249	0.259	0.269	0.280	0.291	0.303	0.315	0.327	0.339	0.350	0.362	0.374	0.385	0.396	0.407	0.417	0.427	0.436	0.445
1.40	0.211	0.219	0.228	0.237	0.246	0.255	0.265	0.275	0.285	0.296	0.306	0.317	0.327	0.337	0.347	0.357	0.365	0.375	0.384	0.393
1.50	0.188	0.195	0.203	0.210	0.218	0.226	0.235	0.243	0.252	0.261	0.270	0.279	0.288	0.297	0.306	0.315	0.325	0.333	0.341	0.349
1.60	0.169	0.175	0.182	0.188	0.195	0.202	0.209	0.217	0.224	0.232	0.240	0.248	0.256	0.264	0.272	0.280	0.290	0.297	0.303	0.311
1.70	0.151	0.157	0.163	0.169	0.175	0.181	0.188	0.194	0.201	0.208	0.215	0.222	0.229	0.236	0.243	0.250	0.257	0.264	0.272	0.278
1.80	0.137	0.143	0.148	0.154	0.159	0.165	0.170	0.176	0.182	0.188	0.194	0.200	0.206	0.212	0.219	0.225	0.233	0.239	0.244	0.251
1.90	0.124	0.129	0.134	0.139	0.144	0.149	0.154	0.160	0.165	0.170	0.175	0.181	0.187	0.192	0.198	0.204	0.208	0.214	0.221	0.227
2.00	0.112	0.117	0.122	0.127	0.132	0.136	0.141	0.146	0.150	0.155	0.160	0.165	0.170	0.175	0.180	0.185	0.188	0.194	0.201	0.206
2.50	0.075	0.078	0.081	0.085	0.088	0.091	0.094	0.097	0.101	0.104	0.107	0.110	0.113	0.116	0.119	0.123	0.126	0.129	-	-
3.00	0.052	0.055	0.057	0.060	0.062	0.065	0.067	0.069	0.072	0.074	0.076	0.079	0.081	0.083	0.085	0.088	0.090	0.092	-	-
3.50	0.039	0.041	0.043	0.044	0.046	0.048	0.050	0.052	0.054	0.055	0.057	0.059	0.061	0.062	0.064	0.066	0.067	0.069	-	-
4.00	0.030	0.031	0.033	0.034	0.036	0.037	0.039	0.040	0.041	0.043	0.044	0.046	0.047	0.048	0.050	0.051	0.052	0.054	-	-
5.00	0.019	0.020	0.021	0.022	0.023	0.024	0.025	0.026	0.027	0.028	0.029	0.029	0.030	0.031	0.032	0.033	0.034	0.035	-	-
6.00	0.014	0.014	0.015	0.015	0.016	0.017	0.017	0.018	0.019	0.019	0.020	0.021	0.021	0.022	0.023	0.023	0.024	0.024	-	-

Appendix C. Revised tables of RHF electron atomic scattering factors

C.3. Revised tables of neutral atoms 41 – 60

s (Å⁻¹)	41. Nb	42. Mo	43. Tc	44. Ru	45. Rh	46. Pd	47. Ag	48. Cd	49. In	50. Sn	51. Sb	52. Te	53. I	54. Xe	55. Cs	56. Ba	57. La	58. Ce	59. Pr	60. Nd
0.00	10.679	10.260	10.856	9.558	9.242	7.583	8.671	9.232	10.434	10.859	10.974	11.003	10.905	10.794	16.508	18.267	17.805	17.378	16.987	16.606
0.01	-	10.230	-	-	-	-	8.654	9.213	10.406	10.833	10.950	-	10.887	10.777	16.391	18.157	-	-	-	-
0.02	-	10.138	-	-	-	-	8.599	9.153	10.320	10.750	10.876	-	10.828	10.725	16.050	17.828	-	-	-	-
0.03	-	9.989	-	-	-	-	8.510	9.057	10.181	10.615	10.755	-	10.731	10.638	15.521	17.309	-	-	-	-
0.04	10.13	9.790	10.35	9.18	8.90	7.43	8.391	8.926	9.995	10.433	10.591	10.65	10.599	10.520	14.855	16.636	16.45	16.10	15.62	15.30
0.05	9.86	9.548	10.10	8.99	8.73	7.35	8.244	8.764	9.768	10.209	10.387	10.47	10.434	10.371	14.106	15.854	15.79	15.46	14.94	14.67
0.06	9.54	9.272	9.80	8.77	8.53	7.26	8.075	8.577	9.509	9.950	10.150	10.25	10.238	10.194	13.326	15.008	15.05	14.77	14.22	13.97
0.07	9.20	8.972	9.48	8.53	8.31	7.16	7.888	8.369	9.224	9.664	9.884	10.01	10.017	9.993	12.556	14.138	14.28	14.03	13.47	13.25
0.08	8.85	8.655	9.14	8.27	8.01	7.03	7.689	8.144	8.923	9.357	9.596	9.74	9.773	9.771	11.823	13.278	13.51	13.29	12.72	12.52
0.09	8.49	8.330	8.78	8.00	7.83	6.91	7.480	7.909	8.612	9.037	9.291	9.46	9.511	9.530	11.145	12.431	12.74	12.56	11.99	11.82
0.10	8.12	8.004	8.42	7.73	7.58	6.77	7.267	7.666	8.297	8.709	8.976	9.16	9.235	9.274	10.525	11.675	12.01	11.85	11.29	11.15
0.11	7.77	7.680	8.07	7.46	7.33	6.62	7.052	7.421	7.983	8.380	8.654	8.85	8.948	9.007	9.965	10.958	11.32	11.19	10.65	10.52
0.12	7.421	7.364	7.720	7.190	7.079	6.474	6.837	7.176	7.674	8.053	8.331	8.538	8.654	8.732	9.458	10.302	10.671	10.561	10.052	9.944
0.13	7.090	7.058	7.383	6.928	6.836	6.319	6.625	6.933	7.374	7.732	8.010	8.224	8.357	8.451	9.000	9.707	10.072	9.981	9.506	9.412
0.14	6.772	6.763	7.057	6.672	6.598	6.162	6.418	6.695	7.084	7.419	7.694	7.914	8.059	8.167	8.583	9.168	9.522	9.448	9.008	8.928
0.15	6.472	6.481	6.746	6.426	6.366	6.003	6.215	6.464	6.805	7.118	7.386	7.608	7.764	7.884	8.201	8.682	9.017	8.958	8.556	8.486
0.16	6.187	6.213	6.451	6.188	6.143	5.843	6.018	6.240	6.539	6.829	7.088	7.309	7.472	7.603	7.848	8.241	8.555	8.507	8.144	8.084
0.17	5.918	5.957	6.171	5.960	5.929	5.684	5.827	6.024	6.286	6.552	6.800	7.018	7.186	7.325	7.519	7.840	8.131	8.094	7.768	7.717
0.18	5.665	5.715	5.907	5.741	5.722	5.526	5.643	5.817	6.045	6.289	6.524	6.738	6.908	7.053	7.212	7.474	7.742	7.714	7.424	7.380
0.19	5.427	5.486	5.658	5.533	5.524	5.369	5.464	5.618	5.817	6.039	6.261	6.467	6.639	6.787	6.922	7.139	7.384	7.365	7.107	7.071
0.20	5.203	5.269	5.423	5.332	5.334	5.214	5.293	5.427	5.601	5.803	6.010	6.209	6.379	6.529	6.649	6.829	7.053	7.041	6.815	6.785
0.22	4.792	4.868	4.994	4.959	4.976	4.913	4.967	5.070	5.203	5.368	5.547	5.727	5.889	6.039	6.143	6.275	6.462	6.462	6.291	6.272
0.24	4.426	4.507	4.614	4.618	4.648	4.626	4.665	4.745	4.846	4.979	5.131	5.291†	5.442	5.586	5.684	5.791	5.948	5.957	5.831	5.822
0.25	4.258	4.341	4.439	4.459	4.493	4.487	4.522	4.592	4.682	4.801	4.940	5.090	5.234	5.374	5.471	5.570	5.714	5.728	5.620	5.615
0.26	4.099	4.182	4.273	4.306	4.345	4.352	4.384	4.447	4.525	4.633	4.760	4.899	5.036	5.172	5.268	5.361	5.495	5.312	5.421	5.421
0.28	3.804	3.888	3.969	4.021	4.066	4.093	4.122	4.173	4.236	4.323	4.428	4.548	4.670	4.795	4.890	4.975	5.092	5.115	5.053	5.059
0.30	3.539	3.622	3.695	3.759	3.809	3.850	3.878	3.922	3.973	4.044	4.131	4.234	4.341	4.454	4.547	4.628	4.730	4.759	4.719	4.731
0.32	3.298	3.379	3.448	3.518	3.572	3.622	3.651	3.690	3.734	3.792	3.865	3.952	4.046	4.147	4.235	4.313	4.405	4.438	4.414	4.432
0.34	3.080	3.158	3.223	3.296	3.353	3.408	3.440	3.476	3.515	3.564	3.625	3.700	3.780	3.870	3.953	4.028	4.111	4.146	4.136	4.157
0.35	2.978	3.054	3.118	3.192	3.249	3.306	3.339	3.375	3.412	3.458	3.514	3.583	3.658	3.742	3.822	3.893	3.974	4.010	4.006	4.029
0.36	2.880	2.955	3.018	3.092	3.150	3.208	3.242	3.278	3.313	3.356	3.408	3.472	3.541	3.620	3.697	3.769	3.844	3.881	3.882	3.906
0.38	2.698	2.770	2.830	2.904	2.962	3.022	3.058	3.093	3.127	3.165	3.210	3.265	3.325	3.394	3.465	3.533	3.602	3.640	3.648	3.675
0.40	2.531	2.600	2.658	2.730	2.788	2.848	2.886	2.922	2.955	2.990	3.030	3.078	3.130	3.191	3.255	3.318	3.381	3.420	3.434	3.462
0.42	2.379	2.444	2.500	2.570	2.626	2.686	2.726	2.762	2.795	2.828	2.864	2.907	2.953	3.006	3.064	3.123	3.180	3.219	3.238	3.267
0.44	2.239	2.300	2.355	2.421	2.477	2.535	2.576	2.613	2.646	2.678	2.712	2.750	2.791	2.838	2.890	2.944	2.997	3.035	3.057	3.087
0.45	2.173	2.233	2.287	2.351	2.406	2.464	2.505	2.542	2.576	2.608	2.640	2.677	2.715	2.759	2.809	2.861	2.911	2.949	2.973	3.003
0.46	2.110	2.168	2.221	2.284	2.338	2.395	2.436	2.474	2.507	2.539	2.571	2.606	2.642	2.684	2.731	2.781	2.829	2.866	2.891	2.922
0.48	1.991	2.046	2.098	2.157	2.210	2.264	2.306	2.344	2.378	2.409	2.440	2.473	2.506	2.543	2.586	2.631	2.676	2.712	2.739	2.769
0.50	1.883	1.934	1.984	2.040	2.090	2.143	2.185	2.223	2.257	2.288	2.318	2.350	2.380	2.414	2.453	2.494	2.535	2.570	2.598	2.628
0.55	1.646	1.690	1.734	1.782	1.828	1.875	1.915	1.953	1.987	2.019	2.048	2.077	2.104	2.132	2.163	2.197	2.230	2.262	2.291	2.320
0.60	1.452	1.490	1.528	1.569	1.609	1.650	1.688	1.724	1.758	1.790	1.819	1.847	1.871	1.897	1.923	1.951	1.979	2.008	2.037	2.064
0.65	1.292	1.324	1.357	1.391	1.426	1.462	1.497	1.531	1.563	1.594	1.622	1.649	1.673	1.697	1.721	1.745	1.770	1.796	1.824	1.849
0.70	1.159	1.185	1.214	1.243	1.273	1.304	1.335	1.366	1.397	1.426	1.453	1.479	1.503	1.526	1.548	1.570	1.592	1.617	1.643	1.666
0.80	0.952	0.971	0.992	1.013	1.035	1.058	1.082	1.107	1.132	1.157	1.181	1.205	1.227	1.248	1.269	1.288	1.308	1.329	1.351	1.372
0.90	0.800	0.814	0.830	0.845	0.861	0.879	0.897	0.916	0.936	0.956	0.976	0.997	1.016	1.036	1.055	1.073	1.090	1.109	1.128	1.146
1.00	0.684	0.695	0.707	0.719	0.731	0.745	0.758	0.773	0.789	0.805	0.821	0.838	0.855	0.871	0.888	0.904	0.920	0.936	0.953	0.969
1.10	0.591	0.601	0.611	0.621	0.631	0.641	0.652	0.664	0.676	0.688	0.701	0.715	0.729	0.743	0.758	0.772	0.785	0.799	0.814	0.828
1.20	0.516	0.525	0.534	0.542	0.551	0.559	0.568	0.578	0.587	0.597	0.608	0.619	0.630	0.642	0.654	0.666	0.678	0.690	0.702	0.715
1.30	0.454	0.462	0.470	0.478	0.485	0.493	0.500	0.508	0.516	0.525	0.533	0.542	0.551	0.561	0.570	0.580	0.591†	0.602	0.612	0.623
1.40	0.401	0.408	0.416	0.423	0.431	0.437	0.444	0.451	0.458	0.465	0.472	0.480	0.487	0.495	0.502	0.511	0.521	0.530	0.539	0.548
1.50	0.356	0.364	0.371	0.378	0.384	0.391	0.397	0.403	0.409	0.416	0.422	0.428	0.435	0.442	0.450	0.456†	0.463	0.470	0.478	0.486
1.60	0.318	0.325	0.332	0.338	0.345	0.351	0.357	0.362	0.368	0.374	0.379	0.385	0.391	0.397	0.405	0.411	0.415	0.421	0.428	0.435
1.70	0.285	0.291	0.298	0.304	0.310	0.316	0.321	0.327	0.332	0.337	0.343	0.348	0.353	0.358	0.363	0.367	0.374	0.380	0.386	0.392
1.80	0.257	0.263	0.269	0.275	0.281	0.286	0.291	0.297	0.302	0.307	0.311	0.316	0.321	0.325	0.332	0.337	0.340	0.345	0.350	0.355
1.90	0.233	0.238	0.244	0.249	0.255	0.260	0.265	0.270	0.274	0.279	0.284	0.288	0.293	0.297	0.299	0.304	0.310	0.314	0.319	0.324
2.00	0.211	0.216	0.222	0.227	0.232	0.237	0.241	0.246	0.250	0.255	0.259	0.264	0.268	0.272	0.272	0.277	0.284	0.288	0.292	0.296
2.50	-	0.142	-	-	-	-	0.159	0.163	0.166	0.170	0.173	-	0.179	0.183	0.186	0.189	-	-	-	-
3.00	-	0.101	-	-	-	-	0.113	0.115	0.118	0.120	0.122	-	0.127	0.130	0.132	0.134	-	-	-	-
3.50	-	0.076	-	-	-	-	0.084	0.086	0.088	0.090	0.091	-	0.095	0.097	0.098	0.100	-	-	-	-
4.00	-	0.059	-	-	-	-	0.066	0.067	0.068	0.069	0.071	-	0.073	0.075	0.076	0.078	-	-	-	-
5.00	-	0.039	-	-	-	-	0.043	0.044	0.045	0.045	0.046	-	0.048	0.049	0.050	0.051	-	-	-	-
6.00	-	0.027	-	-	-	-	0.030	0.031	0.031	0.032	0.033	-	0.034	0.034	0.035	0.036	-	-	-	-

Appendix C. Revised tables of RHF electron atomic scattering factors

C.4. Revised tables of neutral atoms 61 – 80

s (Å⁻¹)	61. Pm	62. Sm	63. Eu	64. Gd	65. Tb	66. Dy	67. Ho	68. Er	69. Tm	70. Yb	71. Lu	72. Hf	73. Ta	74. W	75. Re	76. Os	77. Ir	78. Pt	79. Au	80. Hg
0.00	16.243	15.897	15.563	15.266	14.974	14.641	14.355	14.080	13.814	13.557	13.486	13.177	12.856	12.543	12.263	11.987	11.718	10.813	10.573	10.964
0.01	-	-	15.486	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	10.559	10.948
0.02	-	-	15.260	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	10.511	10.897
0.03	-	-	14.898	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	10.434	10.813
0.04	14.99	14.70	14.425	14.30	13.90	13.64	13.57	13.16	12.92	12.70	12.74	12.55	12.31	12.06	11.83	11.59	11.37	10.55	10.328	10.698
0.05	14.39	14.12	13.867	13.81	13.37	13.14	13.14	12.70	12.48	12.28	12.38	12.23	12.01	11.80	11.60	11.39	11.18	10.40	10.195	10.555
0.06	13.72	13.48	13.253	13.27	12.81	12.60	12.66	12.19	12.00	11.81	11.95	11.85	11.69	11.51	11.34	11.15	10.96	10.23	10.040	10.387
0.07	13.03	12.81	12.611	12.70	12.22	12.03	12.15	11.66	12.16	11.67	12.17	11.68	12.18	11.69	12.19	11.70	12.20	11.71	12.21	11.72
0.08	12.33	12.14	11.963	12.11	11.62	11.44	11.61	11.11	10.96	10.80	11.03	11.02	10.95	10.83	10.73	10.59	10.45	9.82	9.673	9.989
0.09	11.65	11.49	11.329	11.52	11.02	10.87	11.08	10.58	10.44	10.29	10.55	10.59	10.55	10.47	10.40	10.29	10.17	9.60	9.467	9.766
0.10	11.00	10.86	10.722	10.95	10.45	10.32	10.55	10.06	9.93	9.80	10.08	10.16	10.15	10.10	10.05	9.98	9.88	9.37	9.251	9.533
0.11	10.40	10.27	10.150	10.39	9.91	9.79	10.05	9.56	9.45	9.33	9.62	9.73	9.75	9.74	9.71	9.65	9.58	9.13	9.028	9.291
0.12	9.833	9.722	9.618	9.871	9.407	9.303	9.562	9.095	8.994	8.892	9.180	9.308	9.363	9.369	9.366	9.334	9.281	8.882	8.799	9.045
0.13	9.316	9.218	9.128	9.382	8.942	8.848	9.108	8.662	8.571	8.480	8.762	8.907	8.982	9.011	9.028	9.016	8.982	8.636	8.568	8.796
0.14	8.843	8.758	8.678	8.926	8.512	8.429	8.681	8.262	8.180	8.098	8.370	8.525	8.616	8.663	8.697	8.702	8.686	8.389	8.337	8.547
0.15	8.413	8.336	8.267	8.505	8.121	8.045	8.284	7.895	7.821	7.746	8.001	8.163	8.266	8.327	8.376	8.396	8.395	8.145	8.106	8.299
0.16	8.020	7.953	7.891	8.114	7.761	7.693	7.917	7.557	7.490	7.421	7.660	7.822	7.933	8.006	8.067	8.099	8.111	7.904	7.877	8.055
0.17	7.661	7.602	7.548	7.754	7.430	7.370	7.577	7.247	7.185	7.123	7.343	7.502	7.617	7.699	7.769	7.813	7.836	7.667	7.652	7.815
0.18	7.332	7.280	7.232	7.422	7.128	7.073	7.262	6.962	6.905	6.849	7.047	7.202	7.321	7.408	7.485	7.537	7.570	7.436	7.431	7.579
0.19	7.029	6.983	6.942	7.114	6.849	6.800	6.971	6.698	6.646	6.595	6.774	6.922	7.040	7.132	7.213	7.272	7.313	7.210	7.214	7.350
0.20	6.749	6.710	6.673	6.828	6.591	6.547	6.700	6.454	6.407	6.360	6.520	6.660	6.776	6.870	6.954	7.019	7.067	6.991	7.003	7.128
0.22	6.247	6.218	6.191	6.316	6.127	6.092	6.213	6.017	5.978	5.938	6.063	6.185	6.295	6.388	6.475	6.547	6.604	6.572	6.598	6.702
0.24	5.806	5.787	5.768	5.868	5.720	5.693	5.788	5.632	5.601	5.568	5.664	5.768	5.867	5.957	6.043	6.117	6.180	6.181	6.216	6.305
0.25	5.605	5.589	5.574	5.664	5.534	5.510	5.595	5.457	5.428	5.398	5.483	5.578	5.672	5.759	5.843	5.917	5.982	5.995	6.035	6.116
0.26	5.413	5.402	5.390	5.472	5.358	5.337	5.412	5.290	5.265	5.238	5.312	5.399	5.487	5.571	5.653	5.727	5.792	5.817	5.859	5.934
0.28	5.059	5.055	5.030	5.117	5.030	5.016	5.075	4.981	4.961	4.940	4.996	5.069	5.147	5.224	5.301	5.372	5.437	5.478	5.525	5.591
0.30	4.737	4.739	4.740	4.796	4.731	4.723	4.771	4.699	4.685	4.669	4.712	4.772	4.840	4.910	4.981	5.049	5.113	5.164	5.214	5.272
0.32	4.443	4.450	4.456	4.504	4.457	4.454	4.494	4.440	4.430	4.419	4.453	4.503	4.563	4.626	4.691	4.755	4.816	4.873	4.924	4.976
0.34	4.173	4.185	4.195	4.238	4.205	4.206	4.240	4.200	4.195	4.188	4.215	4.258	4.310	4.366	4.425	4.485	4.543	4.603	4.654	4.702
0.35	4.047	4.060	4.072	4.113	4.086	4.089	4.121	4.087	4.084	4.078	4.103	4.143	4.191	4.245	4.301	4.359	4.415	4.475	4.526	4.572
0.36	3.925	3.940	3.954	3.993	3.971	3.976	4.007	3.978	3.976	3.973	3.996	4.033	4.078	4.129	4.182	4.237	4.293	4.352	4.403	4.447
0.38	3.697	3.715	3.731	3.767	3.755	3.763	3.790	3.771	3.773	3.773	3.793	3.825	3.865	3.910	3.959	4.010	4.061	4.120	4.169	4.211
0.40	3.486	3.506†	3.525	3.559	3.554	3.565	3.591	3.579	3.583	3.586	3.604	3.632	3.668	3.709	3.753	3.800	3.848	3.905	3.952	3.991
0.42	3.292	3.314	3.335	3.367	3.368	3.380	3.405	3.399	3.406	3.411	3.429	3.454	3.486	3.523	3.563	3.606	3.651	3.704	3.750	3.787
0.44	3.114	3.137	3.159	3.189	3.194	3.209	3.233	3.232	3.241	3.248	3.265	3.288	3.317	3.350	3.387	3.427	3.468	3.518	3.562	3.597
0.45	3.029	3.053	3.075	3.105	3.113	3.128	3.151	3.153	3.162	3.170	3.187	3.209	3.237	3.269	3.304	3.342	3.382	3.430	3.472	3.507
0.46	2.948	2.973	2.995	3.025	3.034	3.050	3.073	3.076	3.086	3.095	3.111	3.133	3.159	3.190	3.224	3.260	3.299	3.345	3.386	3.420
0.48	2.796	2.821	2.844	2.872	2.884	2.901	2.924	2.930	2.942	2.952	2.968	2.988	3.013	3.041	3.072	3.105	3.141	3.184	3.223	3.256
0.50	2.655	2.680	2.703	2.730	2.745	2.763	2.785	2.793	2.806	2.818	2.834	2.853	2.876	2.902	2.930	2.961	2.994	3.034	3.070	3.102
0.55	2.346	2.371	2.394	2.419	2.457	2.456	2.477	2.490	2.505	2.518	2.534	2.551	2.571	2.592	2.616	2.641	2.669	2.701	2.732	2.760
0.60	2.089	2.113	2.156	2.138	2.178	2.197	2.216	2.232	2.248	2.263	2.278	2.294	2.311	2.330	2.349	2.371	2.394	2.420	2.446	2.471
0.65	1.872	1.895	1.917	1.937	1.958	1.977	1.995	2.012	2.028	2.043	2.058	2.073	2.089	2.105	2.122	2.140	2.160	2.181	2.203	2.225
0.70	1.688	1.709	1.730	1.749	1.770	1.788	1.805†	1.823	1.839	1.854	1.868	1.882	1.896	1.911	1.926	1.942	1.959	1.976	1.995	2.015
0.80	1.391	1.411	1.429	1.446	1.465	1.482	1.497	1.515	1.530	1.545	1.558	1.571	1.583	1.596	1.608	1.621	1.634	1.647	1.661	1.676
0.90	1.164	1.181	1.198	1.213	1.231	1.246	1.260	1.276	1.291	1.305	1.317	1.329	1.341	1.352	1.363	1.374	1.385	1.396	1.407	1.419
1.00	0.985	1.000	1.016	1.030	1.045	1.060	1.073	1.088	1.101	1.114	1.126	1.138	1.148	1.159	1.169	1.179	1.189	1.198	1.208	1.218
1.10	0.842	0.856	0.870	0.883	0.897	0.910	0.922	0.935	0.948	0.960	0.971	0.982	0.993	1.003	1.012	1.022	1.031	1.040	1.048	1.057
1.20	0.727	0.739	0.752	0.763	0.776	0.787	0.799	0.811	0.822	0.833	0.844	0.854	0.864	0.874	0.883	0.892	0.901	0.909	0.918	0.926
1.30	0.634	0.644	0.655	0.666	0.676	0.687	0.698	0.708	0.719	0.729	0.739	0.748	0.758	0.767	0.776	0.784	0.793	0.801	0.809	0.816
1.40	0.557	0.566	0.575	0.583†	0.595	0.604	0.614	0.623	0.632	0.642	0.651	0.660	0.668	0.677	0.685	0.694	0.702	0.709	0.717	0.724
1.50	0.494	0.502	0.511	0.519	0.527	0.535	0.544	0.552	0.560	0.569	0.577	0.585	0.593	0.601	0.609	0.617	0.624	0.632	0.639	0.646
1.60	0.442	0.449	0.457	0.463	0.470	0.478	0.485	0.492	0.500	0.507	0.515	0.522	0.530	0.537	0.544	0.551	0.558	0.565	0.572	0.579
1.70	0.398	0.404	0.409	0.416	0.423	0.429	0.436	0.442	0.449	0.455	0.462	0.469	0.475	0.482	0.489	0.495	0.502	0.508	0.514	0.521
1.80	0.360	0.366	0.372	0.377	0.382	0.388	0.394	0.399	0.405	0.411	0.417	0.423	0.429	0.435	0.441	0.447	0.453	0.459	0.465	0.471
1.90	0.328	0.333	0.337	0.343	0.348	0.353	0.358	0.363	0.368	0.373	0.379	0.384	0.389	0.395	0.400	0.406	0.411	0.416	0.422	0.427
2.00	0.301	0.305	0.307	0.313	0.318	0.322	0.327	0.331	0.336	0.341	0.345	0.350	0.355	0.360	0.365	0.370	0.374	0.379	0.384	0.389
2.50	-	-	0.209	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.256	0.259
3.00	-	-	0.150	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.184	0.186
3.50	-	-	0.113	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.139	0.141
4.00	-	-	0.087	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.108	0.110
5.00	-	-	0.057	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.071	0.072
6.00	-	-	0.040	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.050	0.050

Appendix C. Revised tables of RHF electron atomic scattering factors

C.5. Revised tables of neutral atoms 81 – 98

s (Å⁻¹)	81. Tl	82. Pb	83. Bi	84. Po	85. At	86. Rn	87. Fr	88. Ra	89. Ac	90. Th	91. Pa	92. U	93. Np	94. Pu	95. Am	96. Cm	97. Bk	98. Cf
0.00	12.109	12.597	13.096	13.368	13.473	13.492	18.715	20.561	20.484	20.115	19.568	19.119	18.759	18.191	17.840	17.710	17.406	16.841
0.01	-	12.573	13.070	-	-	13.470	-	-	-	-	-	19.047	-	-	-	-	-	-
0.02	-	12.494	12.989	-	-	13.403	-	-	-	-	-	18.825	-	-	-	-	-	-
0.03	-	12.366	12.857	-	-	13.292	-	-	-	-	-	18.470	-	-	-	-	-	-
0.04	11.71	12.193	12.678	12.95	13.09	13.139	17.14	18.94	19.10	18.92	18.37	17.999	17.70	17.10	16.80	16.80	16.53	16.28
0.05	11.51	11.979	12.456	12.74	12.89	12.949	16.41	18.15	18.41	18.33	17.77	17.436	17.16	16.55	16.28	16.33	16.08	15.85
0.06	11.27	11.730	12.197	12.49	12.65	12.724	15.64	17.31	17.64	17.66	17.11	16.805	16.55	15.95	15.70	15.80	15.58	15.37
0.07	12.22	11.73	12.23	11.74	12.24	11.75	12.25	11.76	12.26	11.77	12.27	11.78	12.28	11.79	12.29	11.80	12.30	11.81
0.08	10.72	11.155	11.595	11.90	12.08	12.187	14.13	15.54	16.01	16.19	15.66	15.436	15.25	14.65	14.47	14.66	14.48	14.30
0.09	10.42	10.840	11.264	11.57	11.76	11.884	13.42	14.69	15.19	15.43	14.92	14.738	14.58	14.00	13.84	14.06	13.91	13.75
0.10	10.12	10.516	10.921	11.22	11.43	11.565	12.77	13.88	14.40	14.68	14.20	14.052	13.92	13.37	13.24	13.47	13.33	13.20
0.11	9.81	10.186	10.571	10.87	11.08	11.232	12.16	13.12	13.64	13.95	13.51	13.389	13.28	12.76	12.65	12.90	12.78	12.66
0.12	9.500	9.855	10.219	10.509	10.729	10.892	11.605	12.419	12.923	13.255	12.850	12.756	12.665	12.191	12.095	12.344	12.241	12.135
0.13	9.195	9.527	9.869	10.153	10.375	10.546	11.093	11.776	12.253	12.594	12.228	12.157	12.085	11.653	11.572	11.817	11.729	11.637
0.14	8.896	9.203	9.523	9.798	10.021	10.199	10.620	11.187	11.632	11.972	11.646	11.595	11.540	11.149	11.083	11.319	11.243	11.164
0.15	8.603	8.888	9.186	9.449	9.671	9.854	10.180	10.648	11.058	11.388	11.102	11.069	11.029	10.679	10.626	10.848	10.784	10.716
0.16	8.320	8.581	8.857	9.109	9.328	9.512	9.770	10.155	10.528	10.845	10.597	10.579	10.551	10.243	10.200	10.407	10.353	10.294
0.17	8.046	8.285	8.539	8.779	8.991	9.177	9.386	9.702	10.038	10.339	10.128	10.122	10.104	9.836	9.803	9.993	9.948	9.898
0.18	7.781	7.999	8.233	8.459	8.666	8.849	9.023	9.285	9.586	9.868	9.691	9.696	9.688	9.457	9.433	9.605	9.568	9.527
0.19	7.526	7.724	7.939	8.151	8.350	8.531	8.681	8.899	9.168	9.430	9.285	9.299	9.300	9.102	9.086	9.241	9.212	9.178
0.20	7.282	7.461	7.658	7.856	8.046	8.223	8.356	8.540	8.780	9.022	8.906	8.928	8.936	8.770	8.760	8.900	8.878	8.850
0.22	6.822	6.969	7.132	7.303	7.474†	7.639	7.754	7.891	8.083	8.287	8.221	8.254	8.275	8.163	8.164	8.277	8.266	8.249
0.24	6.399	6.520	6.654	6.800	6.952	7.102	7.208	7.318	7.474	7.645	7.617	7.659	7.689	7.619	7.631	7.721	7.720	7.713
0.25	6.201	6.310	6.432	6.567	6.709	6.852	6.954	7.055	7.196	7.353	7.341	7.387	7.420	7.368	7.384	7.465	7.468	7.466
0.26	6.011	6.110	6.221	6.345	6.477	6.612	6.712	6.807	6.935	7.079	7.081	7.129	7.165	7.129	7.148	7.222	7.229	7.231
0.28	5.654	5.736	5.828	5.933	6.047	6.166	6.261	6.347	6.455	6.578	6.600	6.652	6.694	6.683	6.708	6.770	6.784	6.793
0.30	5.327	5.395	5.472	5.560	5.658	5.762	5.852	5.931	6.025	6.129	6.167	6.221	6.266	6.274	6.304	6.358	6.378	6.393
0.32	5.025	5.083	5.148	5.222	5.305	5.397	5.480	5.555	5.637	5.727	5.775	5.830	5.878	5.899	5.933	5.981	6.006	6.026
0.34	4.746	4.797	4.852	4.915	4.987	5.065	5.141	5.212	5.285	5.364	5.418	5.473	5.523	5.553	5.591	5.635	5.664	5.687
0.35	4.614	4.662	4.714	4.772	4.838	4.912	4.984	5.053	5.122	5.196	5.252	5.307	5.357	5.391	5.429	5.472	5.502	5.528
0.36	4.488	4.533	4.581	4.636	4.697	4.765	4.834	4.900	4.966	5.036	5.093	5.148	5.197	5.235	5.274	5.316	5.347	5.374
0.38	4.249	4.290	4.333	4.380	4.433	4.492	4.555	4.616	4.675	4.738	4.796	4.850	4.899	4.940	4.981	5.021	5.055	5.084
0.40	4.028	4.066	4.104	4.146	4.192	4.244	4.300	4.356	4.410	4.466	4.524	4.576	4.625	4.669	4.710	4.749	4.784	4.815
0.42	3.823	3.858	3.893	3.931	3.972	4.017	4.067	4.118	4.168	4.218	4.275	4.325	4.372	4.417	4.459	4.497	4.532	4.565
0.44	3.632	3.665	3.698	3.732	3.769	3.808	3.854	3.901	3.946	3.992	4.046	4.094	4.140	4.185	4.226	4.263	4.299	4.333
0.45	3.541	3.573	3.606	3.639	3.673	3.711	3.754	3.798	3.842	3.885	3.938	3.985	4.030	4.076	4.116	4.152	4.189	4.222
0.46	3.454	3.485	3.517	3.548	3.582	3.617	3.658	3.700	3.742	3.784	3.835	3.881	3.925	3.970	4.010	4.046	4.082	4.116
0.48	3.288	3.318	3.348	3.378	3.408	3.441	3.477	3.516	3.554	3.592	3.641	3.685	3.727	3.771	3.810	3.844	3.880	3.914
0.50	3.133	3.162	3.191	3.219	3.248	3.277	3.311	3.346	3.381	3.416	3.462	3.503	3.543	3.586	3.624	3.657	3.693	3.726
0.55	2.789	2.816	2.842	2.868	2.893	2.918	2.945	2.974	3.003	3.032	3.071	3.106	3.141	3.179	3.213	3.244	3.277	3.309
0.60	2.497	2.522	2.546	2.570	2.593	2.616	2.639	2.663	2.687	2.712	2.744	2.775	2.805	2.839	2.869	2.897	2.927	2.957
0.65	2.248	2.271	2.293	2.315	2.337	2.358	2.378	2.399	2.421	2.442	2.470	2.495	2.522	2.551	2.578	2.603	2.630	2.657
0.70	2.035	2.055	2.076	2.096	2.116	2.135	2.154	2.173	2.193	2.212	2.235	2.257	2.280	2.306	2.330	2.352	2.376	2.400
0.80	1.692	1.708	1.725	1.742	1.758	1.775	1.791	1.808	1.824	1.840	1.857	1.875	1.893	1.912	1.930	1.949	1.968	1.987
0.90	1.431	1.444	1.457	1.471	1.485	1.499	1.513	1.527	1.541	1.554	1.568	1.582	1.597	1.611	1.626	1.641	1.657	1.673
1.00	1.228	1.239	1.249	1.260	1.272	1.283	1.295	1.307	1.318	1.330	1.342	1.353	1.365	1.377	1.389	1.402	1.415	1.427
1.10	1.066	1.075	1.084	1.093	1.102	1.112	1.122	1.132	1.142	1.152	1.161	1.171	1.181	1.191	1.201	1.212	1.222	1.233
1.20	0.934	0.942	0.949	0.957	0.965	0.974	0.982	0.990	0.999	1.007	1.016	1.024	1.033	1.041	1.049	1.058	1.067	1.076
1.30	0.824	0.831	0.838	0.846	0.853	0.860	0.867	0.874	0.882	0.889	0.896	0.904	0.911	0.918	0.926	0.933	0.941	0.948
1.40	0.731	0.738	0.745	0.752	0.758	0.765	0.771	0.778	0.784	0.791	0.797	0.803	0.810	0.816	0.823	0.830	0.836	0.843
1.50	0.653	0.659	0.666	0.672	0.678	0.684	0.690	0.696	0.702	0.708	0.714	0.720	0.725	0.731	0.737	0.743	0.748	0.754
1.60	0.585	0.591	0.598	0.603	0.609	0.615	0.621	0.626	0.632	0.637	0.643	0.649	0.653	0.659	0.664	0.669	0.674	0.679
1.70	0.527	0.533	0.538	0.544	0.550	0.555	0.561	0.566	0.571	0.576	0.581	0.585	0.591	0.596	0.601	0.606	0.611	0.615†
1.80	0.476	0.482	0.488	0.493	0.498	0.503	0.508	0.513	0.518	0.523	0.528	0.534	0.537	0.542	0.547	0.551	0.555	0.560
1.90	0.432	0.438	0.443	0.448	0.453	0.458	0.463	0.468	0.472	0.477	0.481	0.485	0.490	0.495	0.499	0.503	0.507	0.511
2.00	0.394	0.399	0.404	0.409	0.413	0.418	0.423	0.427	0.432	0.436	0.440	0.443	0.449	0.453	0.457	0.461	0.465	0.469
2.50	-	0.265	0.268	-	-	0.278	-	-	-	-	-	0.297	-	-	-	-	-	-
3.00	-	0.190	0.192	-	-	0.199	-	-	-	-	-	0.212	-	-	-	-	-	-
3.50	-	0.144	0.145	-	-	0.150	-	-	-	-	-	0.159	-	-	-	-	-	-
4.00	-	0.112	0.113	-	-	0.117	-	-	-	-	-	0.124	-	-	-	-	-	-
5.00	-	0.073	0.074	-	-	0.077	-	-	-	-	-	0.082	-	-	-	-	-	-
6.00	-	0.052	0.052	-	-	0.054	-	-	-	-	-	0.058	-	-	-	-	-	-

Appendix C. Revised tables of RHF electron atomic scattering factors

Appendix D. Papers, published or in draft

Appendix D. Papers, published or in draft

D.1. Paper 1

A two- or three-parameter equation for describing phenomenological absorption in quantitative electron microscopy – I. Determination of the equation.

Authors

Philip N.H. Nakashima[a]*, Tianyu Liu[a], Andrew E. Smith[b] and Laure Bourgeois[ca]

[a]Department of Materials Science and Engineering, Monash University, Victoria, 3800, Australia

[b]School of Physics and Astronomy, Monash University, Victoria, 3800, Australia

[c]Monash Centre for Electron Microscopy, Monash University, Victoria, 3800, Australia

Correspondence email: philip.nakashima@monash.edu

Synopsis A two- or three-variable-parameter function is empirically derived for incorporating atom-localised and non-local phenomenological absorption into quantitative electron diffraction and imaging.

Abstract An accurate formalism for describing phenomenological absorption of electrons by matter is indispensable when it comes to quantitative simulations of electron microscope images and diffraction patterns. The model of Bird & King (1990), embodied in the *ATOM* subroutine written in Fortran77, has become ubiquitous where phenomenological absorption is included in electron scattering simulations. The present work develops an equation that is a robust and close fit to the Bird & King model across all elements (Z = 1 to 98), Debye-Waller factors (B = 0.05 Å^2 to 2.0 Å^2), scattering angles (s = 0 Å^{-1} to 6.0 Å^{-1}) and electron energies (E_0 = 1 keV to 1 MeV) covered by the *ATOM* subroutine. The motivation for doing this is not only to provide a functional description of absorption but also to provide a way of accounting for absorption fully in elastically filtered as well as unfiltered electron microscope images and diffraction patterns.

Keywords: Electron scattering calculations; phenomenological absorption; quantitative electron diffraction and imaging.

1. Introduction

The highly dynamic process of electron scattering from even the thinnest material samples complicates the accurate simulation of electron microscope images and diffraction patterns. The problem is considerably simplified if an electron-optical energy filter is used to exclude almost all of the inelastically scattered electrons. With the advent of electron energy-loss spectrometers and energy-filtered imaging with transmission electron microscopes (TEM) in the late 1980s and early 1990s, Bird

and King met the need for a rigorous description of phenomenological absorption in elastic electron scattering simulations through their *ATOM* subroutine (Bird & King, 1990).

Phenomenological absorption collectively accounts for all electrons that are lost from the signal recorded in a TEM that is to be matched by simulations, irrespective of whether the intensities correspond to an image of the specimen or a diffraction pattern from it. In the context of quantitative TEM, the matching of an experimental image or diffraction pattern with a simulated one allows fundamental materials properties to be accurately measured via refinement as variable parameters in pattern-matching processes. Electrons lost from the signal to be matched can include those that have lost energy due to:

(i) core-shell excitations,

(ii) bremsstrahlung,

(iii) plasmon excitations, and

(iv) thermal diffuse scattering (TDS) arising from phonon excitation.

Electron energy filters allow the removal of (i), (ii) and (iii) above but not (iv) due to the sub 0.1eV losses associated with TDS.

In their treatment of phenomenological absorption, Bird and King went beyond early approximations which began simply by taking the absorption factors to be 10% of the elastic structure factor magnitudes (references needed). Some works went beyond this approximation by trying to examine the effects of absorption on individual reflections in experimental convergent-beam electron diffraction (CBED) patterns (references needed). With the advent of quantitative convergent-beam electron diffraction (QCBED), Bird and King identified and satisfied the need for a more complete analytical treatment of phenomenological absorption (Bird & King, 1990). Their resulting *ATOM* subroutine became an intrinsic component of any QCBED software as well as a variety of general TEM image and diffraction pattern simulation programs (references needed).

In the last 10 years, a new approach to quantitative TEM has been developed, out of the field of QCBED pattern matching, that does not require the use of energy-filtering electron optics (references needed). Two different types of differential techniques have been developed that involve: (a) differentiation with respect to specimen thickness (Nakashima, 2007; Nakashima & Muddle, 2010b), and (b) differentiation with respect to scattering angle (Nakashima & Muddle, 2010a; Nakashima & Muddle, 2010b). In the development of this field, it was shown that both types of differentiation result in almost complete annulment of the diffuse, slowly varying inelastic signal in CBED patterns that contributes a significant background that is deleterious to QCBED pattern-matching measurements of bonding electron density-sensitive structure factors.

It was shown by Nakashima & Muddle (2010a) that any structure that remains in the background of CBED patterns that have been electron-optically filtered, simply mimics the structure of the elastic rocking curves within the reflection discs of the CBED patterns. It is also well-known that unfiltered CBED patterns retain an angular intensity distribution contributed by electrons inelastically scattered by plasmon excitations, and that this also mimics the elastic intensity distribution, albeit with a slight amount of blurring (references needed). As concluded in (Nakashima & Muddle, 2010b), differentiating an unfiltered CBED pattern with respect to thickness or scattering angle will therefore simply result in a differential intensity distribution that can be matched by an elastic scattering calculation that has been differentiated in the same way. Because a calculated CBED pattern is always normalised to have the same signal magnitude as the experimental pattern before the " goodness of fit" is evaluated, the extra signal contributed by TDS and plasmon excitation in differential CBED patterns that have not been hardware energy filtered, is simply absorbed into this normalisation.

This, however, leaves the approximation that the absorption factors can simply be carried over into QCBED pattern matching of unfiltered CBED patterns using the ubiquitous Bird and King *ATOM* subroutine. In the pursuit of accuracy and precision in QCBED, such an approximation is difficult to justify because the absorption factors from *ATOM* that describe elastic scattering are incorporated into the scattering matrix or phase grating determination (depending on whether the Bloch wave or multislice formalisms are being used respectively) to calculate CBED patterns being matched to unfiltered (albeit differentiated) CBED data.

The purpose of the present paper is to condense the full range of output from the Bird and King model for phenomenological absorption into a single equation that serves as a good approximation for describing absorption in energy-filtered CBED patterns. This is to be followed by combining this expression with a separate one that accounts for the additional signal contributed by inelastically scattered electrons. The form of the desired equation should show the separation of the contributions to phenomenological absorption so that future QCBED could in principle determine the relative magnitudes of the components.

Bird and King's *ATOM* subroutine deals entirely with absorption due to phonon excitation, i.e. TDS, which, via the use of an Einstein model in their work, is approximated as being localised to each atom (Bird & King, 1990). They also describe the other contributions to absorption as being non-local and therefore irrelevant due to the process of normalisation. Here we remove this approximation (also made by Nakashima and Muddle (2010b)) by considering all non-local contributions as a Dirac delta function centred on the origin of reciprocal space. We therefore consider the treatment of absorption in the following form:

$$f_j'\left(Z_j, B_j, E_0, s\right) = A f_j'^{non-local}\left(Z_j, B_j, E_0, s\right) + C f_j'^{local}\left(Z_j, B_j, E_0, s\right), \qquad (1)$$

where A and C are variable parameters and the first term summarises the non-local contributions, $f_j^{'non\text{-}local}(Z_j, B_j, E_0, s)$, in the form of a Dirac delta function, whilst the second term describes the atom localised contribution, $f_j^{'local}(Z_j, B_j, E_0, s)$, which should be a close approximation to the model of Bird and King, $f_j^{'B\&K}(Z_j, B_j, E_0, s)$, as calculated by their *ATOM* subroutine. The absorption factor for the jth atom in the unit cell depends on the atom's atomic number (Z_j), its Debye-Waller factor at the relevant temperature (B_j), the incident electron energy (E_0) and s ($s = (\sin\theta)/\lambda$).

2. Analysis

An empirical determination of the form of the second term in equation 1 is now described in detail. Considering not only Bird & King (1990) but also previous descriptions of phenomenological absorption (references needed), the inclusion of a Gaussian term is obviated. Even the most primitive approximations used prior to the work of Bird and King suggest this because they set $V_{\mathbf{g}}' = 0.1V_{\mathbf{g}}$ (references needed). Here, $V_{\mathbf{g}}$ is the structure factor of the crystal potential associated with scattering vector $\mathbf{g}$ in units of volts and $V_{\mathbf{g}}'$ is the associated absorption factor. Following this crude approximation, and given

$$V_{\mathbf{g}} = \sum_j f_j(s) e^{-2\pi i \mathbf{g}.\mathbf{r}_j} e^{-B_j s^2}, \quad (2)$$

where $f_j(s)$ is the atomic scattering factor for the jth atom and $\mathbf{r}_j$ is its position in the unit cell, then,

$$V_{\mathbf{g}}' = 0.1V_{\mathbf{g}} = 0.1\sum_j f_j(s) e^{-2\pi i \mathbf{g}.\mathbf{r}_j} e^{-B_j s^2} = \sum_j f_j'(s) e^{-2\pi i \mathbf{g}.\mathbf{r}_j} e^{-B_j s^2}, \quad (3)$$

and so,

$$f_j'(s) = 0.1 f_j(s). \quad (4)$$

Given $f_j(s)$ is very well approximated by a sum of Gaussians in s (reference to IUCr tables), it follows that $f_j^{'local}(Z_j, B_j, E_0, s)$ would include a significant Gaussian term as a function of s.

Bird & King (1990) and a number of other investigators (references needed) suggested that $f_j^{'local}(Z_j, B_j, E_0, s)$ may be Lorentzian in form in its tail region (higher values of s) so the approach adopted in this work is to test sums of Gaussians and Lorentzians in fitting a range of output from the *ATOM* subroutine, remembering that the present aim is not to replace the model of Bird and King, $f_j^{'B\&K}(Z_j, B_j, E_0, s)$, but rather to encapsulated in a single function, $f_j^{'local}(Z_j, B_j, E_0, s)$, that approximates it well, i.e. $f_j^{'local}(Z_j, B_j, E_0, s) \approx f_j^{'B\&K}(Z_j, B_j, E_0, s)$.

Figure 1 plots $f_{Al}^{'B\&K}(Z_{Al} = 13,\ 0.05\ \text{Å}^2 \le B_{Al} \le 2.0\ \text{Å}^2,\ E_0 = 200\ \text{keV},\ 0\ \text{Å}^{-1} \le s \le 6\ \text{Å}^{-1})$, i.e. the absorption factor for aluminium with 200 keV incident electrons over the full range of Debye-Waller factors and s calculable by *ATOM*. Aluminium was chosen as the subject of figure 1 for no other reason than that it is the main test material examined in the sequel paper (Nakashima *et al.*, 2018) which tests

the outcomes of the present paper. The two-dimensional colour plot of $f_{Al}^{'B\&K}$ as a function of both s (horizontal axis) and B_{Al} (vertical axis) is shown in the middle panel and is constructed from a grid of 200 x 200 pixels uniformly spanning the full range of B_{Al} and s stated above. Five different coloured lines (4 horizontal and 1 vertical) are drawn within this plot. The single vertical (grey) line is the locus of s = 0 Å^{-1} from which the plot of $f_{Al}^{'B\&K}$ as a function of B_{Al} at s = 0 Å^{-1} is obtained as shown in the left panel of the figure. The 4 horizontal lines correspond to the loci: B_{Al} = 0.197 Å^2 (lilac), B_{Al} = 0.334 Å^2 (green), B_{Al} = 0.863 Å^2 (blue) and B_{Al} = 1.94 Å^2 (red), and were chosen because these Debye-Waller factors correspond to temperatures attainable by liquid helium cooling (T = 10 K), liquid nitrogen cooling (T = 90 K), ambient conditions (T = 293 K) and *in-situ* annealing (T = 573 K for aluminium) experiments in TEMs. It is along these four loci that $f_{Al}^{'B\&K}$ as a function of s is plotted in the right panel of figure 1. The positions of these lines in terms of B_{Al} are also indicated in the plot of $f_{Al}^{'B\&K}$ as a function of B_{Al} at left.

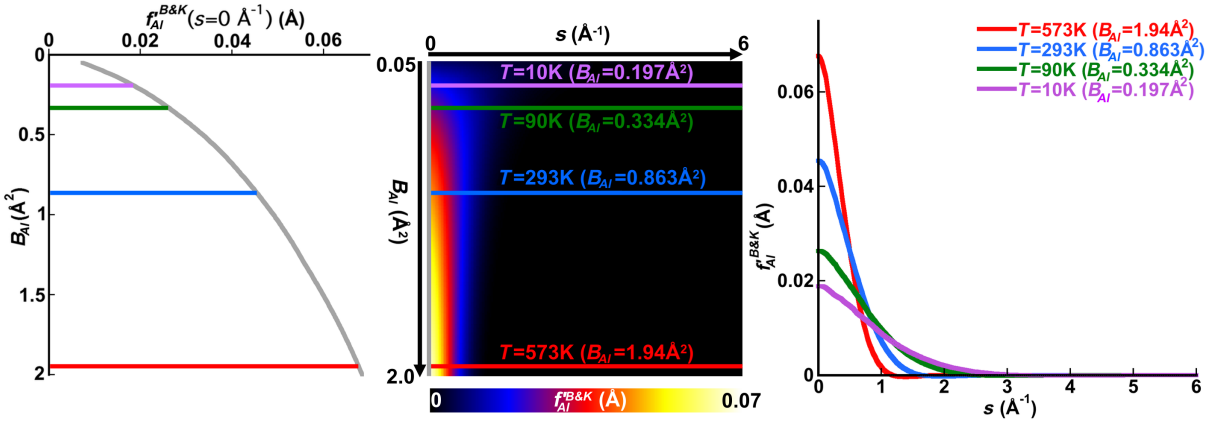


Figure 1 The Bird and King absorption factor for aluminium, $f_{Al}^{'B\&K}$, plotted at E_0 = 200 keV over the range 0 Å^{-1} ≤ s ≤ 6 Å^{-1} and 0.05 Å^2 ≤ B_{Al} ≤ 2.0 Å^2 with s along the horizontal axis and B_{Al} along the vertical axis. The magnitude of $f_{Al}^{'B\&K}$ as computed by the *ATOM* subroutine is given by the colour scale beneath the central panel and this plot is constructed from a grid of 200 x 200 pixels uniformly spanning the full range of B_{Al} and s specified above. Profiles of $f_{Al}^{'B\&K}$ as a function of s are plotted for four different temperatures in the right panel (T = 10 K, T = 90 K, T = 293 K and T = 573 K corresponding to typical temperature attainable with liquid helium cooling, liquid nitrogen cooling, ambient conditions and *in situ* annealing respectively). The left panel shows $f_{Al}^{'B\&K}$ as a function of B_{Al} at s = 0 Å^{-1}. The coloured lines in the left and middle panels show where the corresponding coloured line plots in the right panel are located in these other plots.

The most notable trends from figure 1 are that as the temperature (and Debye-Waller factor) increases, the absorption at s = 0 Å^{-1} increases whilst the tail of $f_{Al}^{'B\&K}$ is very rapidly damped as s

increases. This is readily explained by the increase in TDS with increasing temperature, meaning that the mean absorption will increase and thus, $f_{Al}{}'^{B\&K}(s = 0\ \text{Å}^{-1})$ will increase. The increased thermal displacement of atoms at higher temperatures means that the temperature factor, $e^{-B_j s^2}$ for all atoms, j, becomes rapidly smaller with increasing s due to the larger value of B_j at higher temperatures, thus damping $f_{Al}{}'^{B\&K}$ very rapidly as s increases. Note that *ATOM*-computed absorption factors, $f_{Al}{}'^{B\&K}$, already include the temperature factor (Bird & King, 1990). Worth noting also is that $f_j{}'^{B\&K}(Z_j, B_j, E_0, s) < 0$ for some intermediate values of s. This can be seen on close inspection of the right panel of figure 1, especially for the case of $B_{Al} = 1.94$ Å^2. This is the case for $f_j{}'^{B\&K}(Z_j, B_j, E_0, s)$ in many cases and although the magnitudes of the dips below 0 are small, they are not physically realistic. The development of the function for $f_j{}'^{local}(Z_j, B_j, E_0, s)$ in this paper must avoid situations in which $f_j{}'^{local}(Z_j, B_j, E_0, s) < 0$. From the left panel of figure 1, it is also evident that $f_{Al}{}'^{B\&K}(s = 0\ \text{Å}^{-1})$ is approximately proportional to $B_{Al}{}^{0.5}$. This is informative in the choice of function to be used for $f_j{}'^{local}(Z_j, B_j, E_0, s)$, with the aim of closely approximating $f_j{}'^{B\&K}(Z_j, B_j, E_0, s)$.

The next step is to determine a suitable form for $f_j{}'^{local}(Z_j, B_j, E_0, s)$. By experimenting with different combinations of Gaussians and Lorentzians, it was found that the best fit to the plot of $f_{Al}{}'^{B\&K}(Z_{Al} = 13,\ 0.05\ \text{Å}^2 \le B_{Al} \le 2.0\ \text{Å}^2,\ E_0 = 200\ \text{keV},\ 0\ \text{Å}^{-1} \le s \le 6\ \text{Å}^{-1})$ in figure 1 was obtained if one sets:

$$f_j'^{local}(Z_j, B_j, E_0, s) = e^{-B_j s^2}\left(aB_j^b e^{-cs^d} + \frac{fs^g}{1+hs^{g+2}}\right), \qquad (5)$$

where a, b, c, d, f, g and h are refinable variable parameters. Examining this function closely, the first term in the brackets is the Gaussian in terms of s and this term will dominate when s is small because the Lorentzian second term vanishes as s approaches 0. The multiplier, B_j^b, is included in the Gaussian term as there is an obvious dependence of $f_j{}'^{B\&K}(Z_j, B_j, E_0, s = 0\ \text{Å}^{-1})$ on B_j^b in figure 1 where one might estimate that parameter $b \approx 0.5$ for the case of aluminium with $E_0 = 200$ keV as stated above. The temperature factor, $e^{-B_j s^2}$, modifies the entire function in the brackets.

Figure 2(a) repeats the central plot in figure 1, which shows $f_{Al}{}'^{B\&K}(Z_{Al} = 13,\ 0.05\ \text{Å}^2 \le B_{Al} \le 2.0\ \text{Å}^2,\ E_0 = 200\ \text{keV},\ 0\ \text{Å}^{-1} \le s \le 6\ \text{Å}^{-1})$ as a function of B_{Al} and s. Figure 2(b) shows $f_{Al}{}'^{local}(Z_{Al} = 13,\ 0.05\ \text{Å}^2 \le B_{Al} \le 2.0\ \text{Å}^2,\ E_0 = 200\ \text{keV},\ 0\ \text{Å}^{-1} \le s \le 6\ \text{Å}^{-1})$ as a function of B_{Al} and s after fitting equation 5 to the plot of figure 2(a). From the values of the refined parameters listed in the caption to figure 2, the prediction that $b \approx 0.5$ is satisfied in the present example for aluminium with 200 keV electrons. Furthermore, $g \approx 2$ suggests that the Lorentzian term in equation 5 has even parity, which is a natural consequence of the radial symmetry expected from the absorption processes being accounted for.

Figure 2(c) shows the map of the difference between figure 2(a) and 2(b), i.e. $f_{Al}{}'^{B\&K} - f_{Al}{}'^{local}$. It is worth noting that the mismatch is at least an order of magnitude smaller than the individual

magnitudes of $f_{Al}{}^{'B\&K}$ and $f_{Al}{}^{'local}$. In the process of optimising the fit between $f_{Al}{}^{'B\&K}$ and $f_{Al}{}^{'local}$, a mismatch parameter was minimised, which is defined as

$$\chi_{j,E_0} = \sqrt{\frac{\sum_{i=1}^{n}\left(f'^{B\&K}_{j,i} - f'^{local}_{j,i}\right)^2}{\sum_{i=1}^{n}\left(f'^{B\&K}_{j,i}\right)^2}}. \qquad (6)$$

This is the root-mean-square (RMS) fractional difference between $f_j{}^{'B\&K}$ and $f_j{}^{'local}$ where the sums are over the $i = 1$ to n pixels that make up both images for element j with electrons of energy E_0 in the generalised case. For the present example of aluminium and 200 keV electrons, $\chi_{Al,200\ \mathrm{keV}} = 0.041$ for figure 2(a) and (b) and this is reflected in figure 2(c).

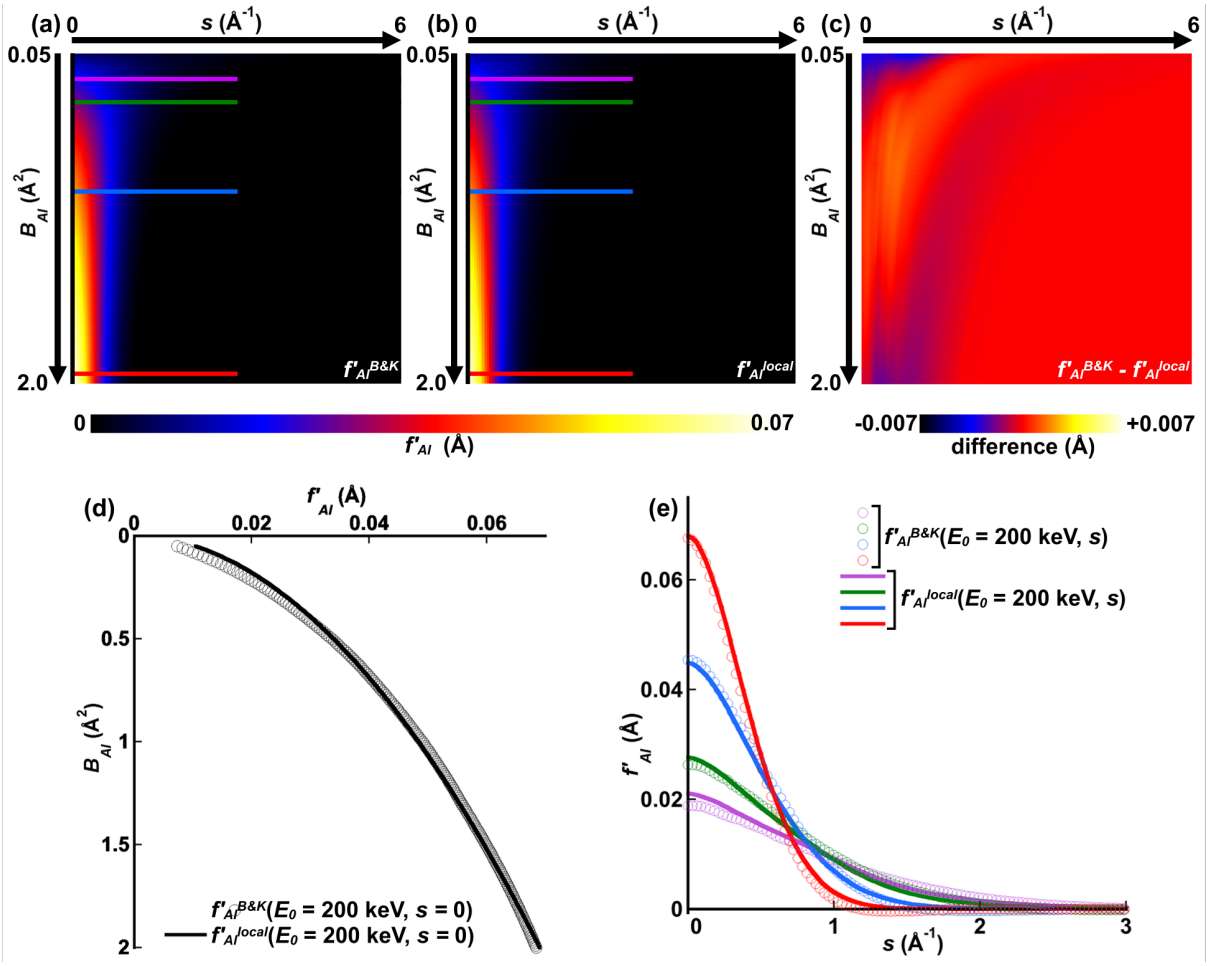


Figure 2 A comparison of (a) $f_{Al}{}^{'B\&K}(Z_{Al} = 13,\ 0.05\ \text{Å}^2 \le B_{Al} \le 2.0\ \text{Å}^2,\ E_0 = 200\ \text{keV},\ 0\ \text{Å}^{-1} \le s \le 6\ \text{Å}^{-1})$ and (b) $f_{Al}{}^{'local}(Z_{Al} = 13,\ 0.05\ \text{Å}^2 \le B_{Al} \le 2.0\ \text{Å}^2,\ E_0 = 200\ \text{keV},\ 0\ \text{Å}^{-1} \le s \le 6\ \text{Å}^{-1})$ after fitting equation 5 to the former. The parameters producing the best fit of equation 5 to $f_{Al}{}^{'B\&K}$ calculated by the *ATOM* subroutine are: $a = 0.048395$, $b = 0.51240$, $c = 1.49948$, $d = 1.69455$, $f = 0.0122095$, $g = 2.00334$, $h = 0.8985707$. The difference map (c) is shown with a colour scale that spans magnitudes 10 times smaller than those in the individual images of $f_{Al}{}^{'B\&K}$ and $f_{Al}{}^{'local}$. Graph (d) compares $f_{Al}{}^{'B\&K}$ and $f_{Al}{}^{'local}$ along the black locus in (a) and (b) at $s = 0$ (as in figure 1) and graph (e) compares $f_{Al}{}^{'B\&K}$ and $f_{Al}{}^{'local}$ along the

coloured loci at B_{Al} corresponding to T = 10 K, T = 90 K, T = 293 K and T = 573 K as in figure 1, but only over the range 0 Å^{-1} ≤ s ≤ 3 Å^{-1} as values beyond s = 3 Å are vanishingly small.

Figure 2(d) plots $f_{Al}{}'^{B\&K}$ and $f_{Al}{}'^{local}$ as functions of B_{Al} along the locus s = 0 Å^{-1} (black) and figure 2(e) shows both $f_{Al}{}'^{B\&K}$ and $f_{Al}{}'^{local}$ plotted as functions of s for the same values of B_{Al} examined in figure 1 but over the reduced range of 0 Å^{-1} ≤ s ≤ 3 Å^{-1} for the sake of magnifying the differences between $f_{Al}{}'^{B\&K}$ and the fitted $f_{Al}{}'^{local}$ at low values of s. From all of these plots, it appears that equation 5 with best-fit refined parameters a to h is a very close approximation to the *ATOM* subroutine and thereby, the Bird and King model for phenomenological absorption localised to each atom.

Using the values of the best-fit parameters in the caption to figure 2, the relative contributions and forms of the terms in equation 5 can be examined. This is done in figure 3 where the blue plot in figure 2(e), corresponding to ambient temperature (T = 293 K and B_{Al} = 0.863 Å^2), is decomposed into its different components as per equation 5.

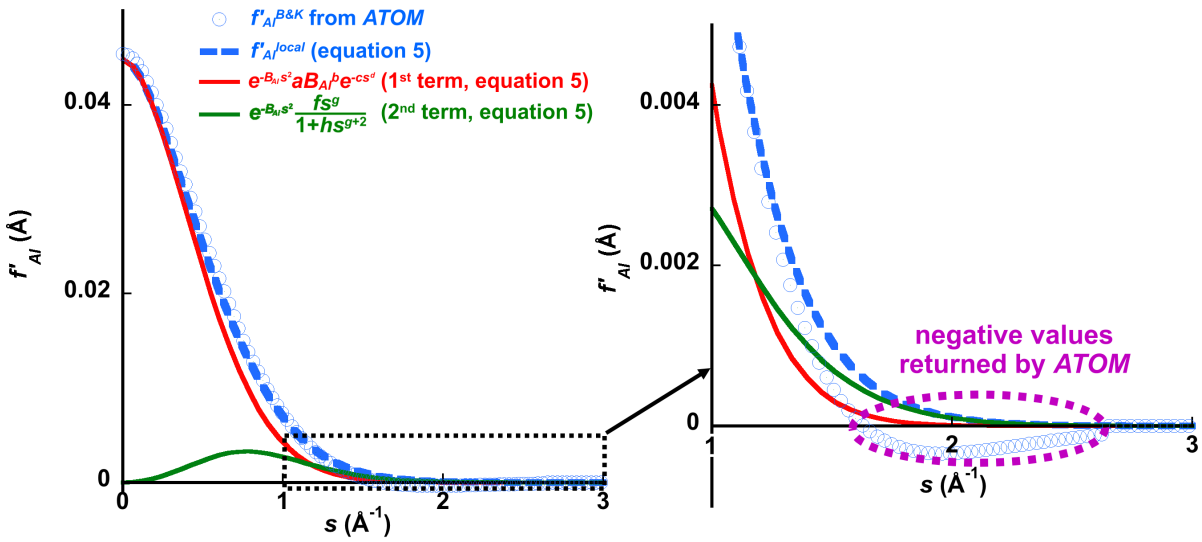


Figure 3 Plots of $f_{Al}{}'^{B\&K}$ and the fitted $f_{Al}{}'^{local}$ showing the contributions of each term in equation 5. Magnifying the region 1 Å^{-1} ≤ s ≤ 3 Å^{-1} shows that the $f_{Al}{}'^{B\&K}$ values calculated by the *ATOM* subroutine are negative at intermediate values of s. Equation 5, which constitutes $f_j{}'^{local}$ and approximates $f_j{}'^{B\&K}$, does not become negative anywhere. In the present case of aluminium (j = Al) and electrons with E_0 = 200 keV, the Lorentzian component dominates $f_{Al}{}'^{local}$ for values of s > 1.2 Å^{-1}.

At least for this case (aluminium at room temperature with 200 keV electrons), it appears that the Gaussian term is dominant for the lower values of s, which, in terms of elastic scattering, includes the components of the electrostatic potential distribution associated with bonding. This suggests that techniques like quantitative convergent-beam electron diffraction (QCBED) which measure the

bonding-sensitive elastic structure factors of the crystal potential can reasonably approximate $f_j^{'local}$ with just a Gaussian. In fact, QCBED using small Laue circle geometries is unlikely to be sensitive to the Lorentzian component of $f_{Al}^{'local}$ given in equation 5.

Accepting equation 5 as a suitable and likely form for $f_j^{'local}(Z_j, B_j, E_0, s)$, it is important to establish whether the number of refineable variables can be reduced by replacing them with functions of the variables Z_j, B_j, E_0 and s. As a first test, in the same manner as was done for aluminium and 200 keV electrons in figure 2, the fitting of equation 5 to $f_j^{'B\&K}(Z_j, B_j, E_0, s)$ from the *ATOM* subroutine was repeated for Al (Z = 13), Cu (Z = 29), Ag (Z = 47), Nd (Z = 60), Au (Z = 79) and U (Z = 92) and in each of these cases, with electron energies ranging from 1 keV $\leq E_0 \leq$ 1 MeV. In other words, this meant that optimised sets of parameters and their associated RMS misfits $\{a, b, c, d, f, g, h, \chi_{j,E_0}\}$ were obtained as a function of both Z_j and E_0 from comparisons of $f_{Al}^{'B\&K}(Z_j, 0.05\ \text{Å}^2 \leq B_{Al} \leq 2.0\ \text{Å}^2, E_0, 0\ \text{Å}^{-1} \leq s \leq 6\ \text{Å}^{-1})$ and $f_{Al}^{'local}(Z_j, 0.05\ \text{Å}^2 \leq B_{Al} \leq 2.0\ \text{Å}^2, E_0, 0\ \text{Å}^{-1} \leq s \leq 6\ \text{Å}^{-1})$ for each of the elements given above and for 1 keV $\leq E_0 \leq$ 1 MeV. The only difference in procedure for these refinements was that a 100 x 100 grid of pixels spanning 0.05 Å$^2 \leq B_j \leq$ 2.0 Å^2 and 0 Å$^{-1} \leq s \leq$ 6 Å^{-1} was used instead of 200 x 200 pixels as in figures 1 and 2.

In performing these fits, it transpired that the parameters b, c, d, g and h are completely independent of E_0 and only dependent on Z_j, remaining constant during each refinement of equation 5 in fitting $f_{Al}^{'B\&K}(Z_j, 0.05\ \text{Å}^2 \leq B_{Al} \leq 2.0\ \text{Å}^2, E_0, 0\ \text{Å}^{-1} \leq s \leq 6\ \text{Å}^{-1})$ for any one element Z_j. Only the parameters a and f are dependent on E_0 and their dependence on electron energy is plotted for each of the six elements given above, in figure 4.

As becomes clear from figure 4, parameters a and f in equation 5 have the same form when plotted as a function of E_0, independent of atomic number. Their magnitudes increase with increasing atomic number. A variety of asymptotic functions were tested in fitting a and f versus E_0 with the following form yielding perfect fits in every case:

$$a \text{ and } f = p\left(q + \frac{e^{-rE_0^t}}{1-e^{-rE_0^t}}\right), \tag{7}$$

Where p, q, r and t are introduced as refineable variables in fitting equation 7 to the points in the plots of a and f versus E_0 for each of the elements as shown in figure 4. Thus, 12 fits were carried out and in all cases, the parameters q, r and t were found to be constant between different elements and between a and f versus E_0. It turned out that $q = t = 0.5$ and $r = 0.0041225$ in all cases and that only p changed from fit element to element and between a and f. One can therefore express a and f as follows:

$$a = m\left(0.5 + \frac{e^{-0.0041225E_0^{0.5}}}{1-e^{-0.0041225E_0^{0.5}}}\right) \tag{8}$$

and

$$f = n\left(0.5 + \frac{e^{-0.0041225E_0^{0.5}}}{1-e^{-0.0041225E_0^{0.5}}}\right), \tag{9}$$

where m and n are retained as variable parameters that simply allow for different magnitudes of these functions as observed in figure 4.

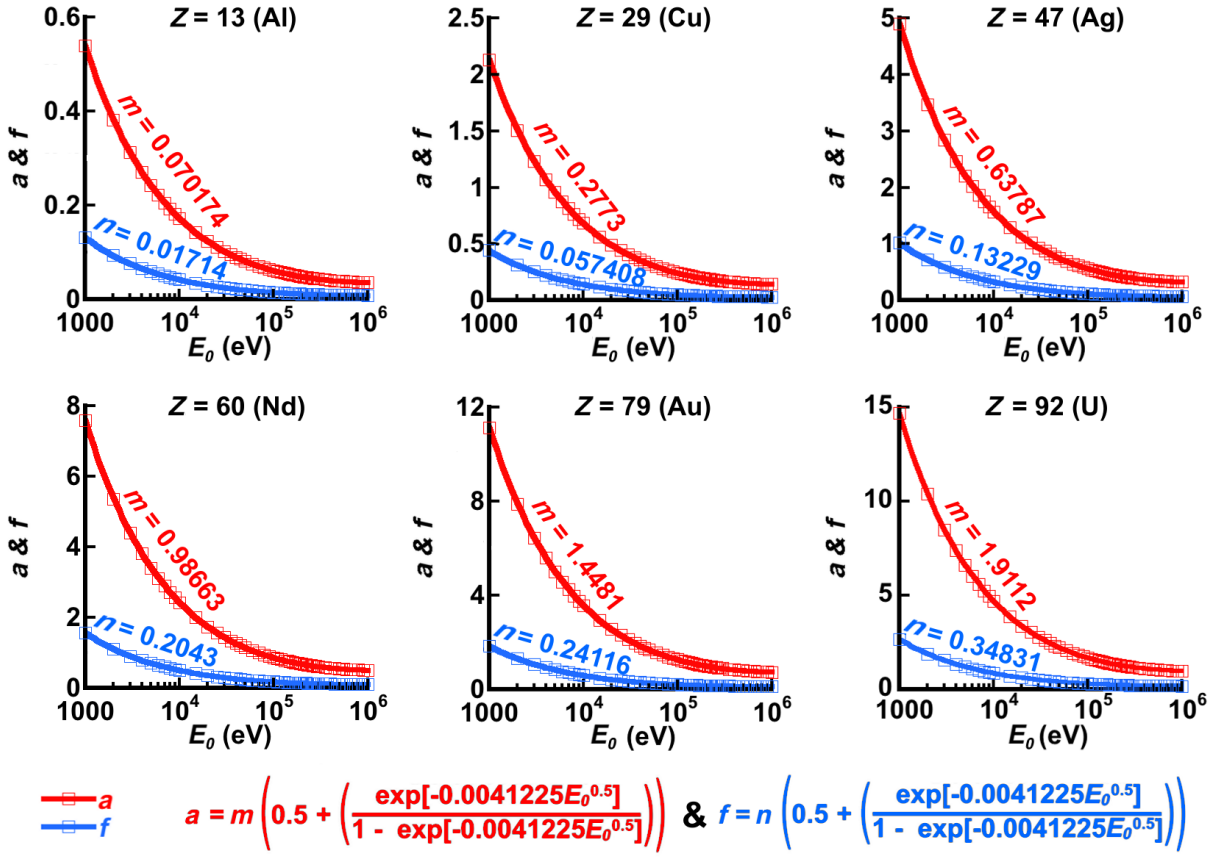


Figure 4 Graphs of parameters a and f in equation 5 as a function of the electron energy, E_0, after fitting equation 5 to $f_{Al}^{'B\&K}(Z_j,\ 0.05\ \text{Å}^2 \le B_{Al} \le 2.0\ \text{Å}^2,\ E_0,\ 0\ \text{Å}^{-1} \le s \le 6\ \text{Å}^{-1})$ for Al (Z = 13), Cu (Z = 29), Ag (Z = 47), Nd (Z = 60), Au (Z = 79) and U (Z = 92). The fitted functions for a versus E_0 and f versus E_0 all have the same form (given explicitly below the plots), independent of the element and independent of the parameter a or f. The only difference is in the relative magnitudes of the functions so these are assigned new parameters m and n for the functions describing a and f respectively. Note: the functions for a and f versus E_0 fit the plotted points perfectly without any mismatch at all.

Substituting equations 8 and 9 into equation 5 builds in the energy dependence of $f_j^{'local}(Z_j,\ B_j,\ E_0,\ s)$ and equation 5 becomes:

$$f_j^{'local}(Z_j, B_j, E_0, s) = e^{-B_j s^2}\left(0.5 + \frac{e^{-0.0041225\sqrt{E_0}}}{1-e^{-0.0041225\sqrt{E_0}}}\right)\left(mB_j^b e^{-cs^d} + \frac{ns^g}{1+hs^{g+2}}\right). \quad (10)$$

The next step is to determine the dependence of the variable parameters m, n, b, c, d, g and h in equation 10 on atomic number, Z_j. Given the dependence on of $f_j^{'local}(Z_j,\ B_j,\ E_0,\ s)$ on E_0 has already been dealt with completely, the electron energy for all future analyses is set to E_0 = 200 keV. The same approach as in figure 2 is now taken for all elements accommodated by the *ATOM* subroutine resulting

in a comparison between $f_j^{'B\&K}(Z_j,\ 0.05\ \text{Å}^2 \le B_j \le 2.0\ \text{Å}^2,\ E_0 = 200\ \text{keV},\ 0\ \text{Å}^{-1} \le s \le 6\ \text{Å}^{-1})$ and (b) $f_j^{'local}(Z_j,\ 0.05\ \text{Å}^2 \le B_j \le 2.0\ \text{Å}^2,\ E_0 = 200\ \text{keV},\ 0\ \text{Å}^{-1} \le s \le 6\ \text{Å}^{-1})$ for Z_j = 1 to 98 (H to Cf). Again, as with the examination of the dependence of $f_j^{'local}(Z_j,\ B_j,\ E_0,\ s)$ on E_0, the only difference in procedure for these comparisons from that shown in figure 2 was that a 100 x 100 grid of pixels spanning $0.05\ \text{Å}^2 \le B_j \le 2.0\ \text{Å}^2$ and $0\ \text{Å}^{-1} \le s \le 6\ \text{Å}^{-1}$ was used instead of 200 x 200 pixels as was used for figures 1 and 2.

For each element j, an optimal set of parameters and associated RMS misfit were returned, i.e. $\{m_j,\ n_j,\ b_j,\ c_j,\ d_j,\ g_j,\ h_j,\ \chi_{j,E_0=200\ \text{keV}}\}$. An overall assessment of the ability of equation 10 to approximate $f_j^{'B\&K}(Z_j,\ 0.05\ \text{Å}^2 \le B_j \le 2.0\ \text{Å}^2,\ E_0 = 200\ \text{keV},\ 0\ \text{Å}^{-1} \le s \le 6\ \text{Å}^{-1})$, in other words, the ability of equation 10 to approximate the entirety of the *ATOM* subroutine, can be gained by considering the mean RMS misfit, $\bar{\chi}$, averaged over all of the elements considered. With all seven parameters, *m, n, b, c, d, g* and *h*, allowed to vary, $\bar{\chi} = 0.0407 \pm 0.0007$. This figure can be regarded as the benchmark RMS misfit with all degrees of freedom in equation 10 available. As each of the parameters, *m, n, b, c, d, g* and *h*, is replaced, the performance of equation 10 in approximating $f_j^{'B\&K}(Z_j,\ B_j,\ E_0,\ s)$ is expected to deteriorate.

The optimal parameters $\{m_j,\ n_j,\ b_j,\ c_j,\ d_j,\ g_j,\ h_j\}$ for each element are plotted as a function of Z_j in figure 5. The parameters are grouped into each of the graphs based on similar behaviours with respect to *Z* and functions have been found that give the best fit to the plot of each parameter.

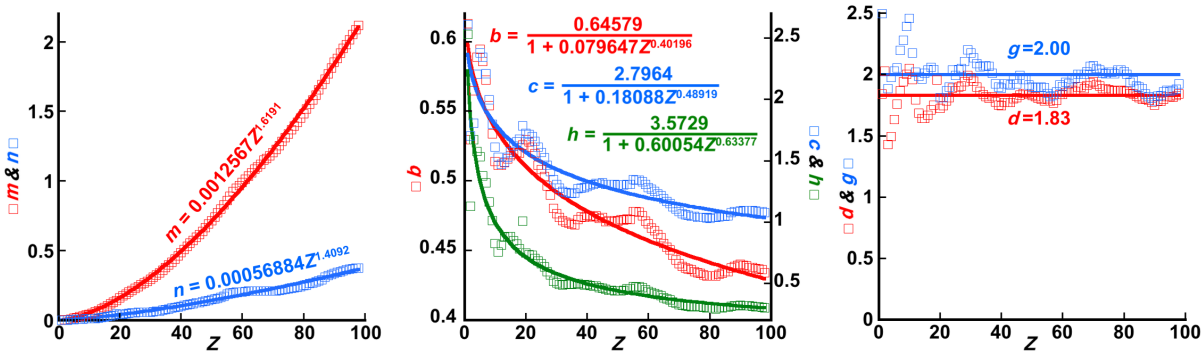


Figure 5 Plots of the variable parameters *m, n, b, c, d, g* and *h* as a function of *Z* after fitting equation 10 to $f_j^{'B\&K}(Z_j,\ 0.05\ \text{Å}^2 \le B_j \le 2.0\ \text{Å}^2,\ E_0 = 200\ \text{keV},\ 0\ \text{Å}^{-1} \le s \le 6\ \text{Å}^{-1})$ obtained from the *ATOM* subroutine. Parameters have been grouped into plots based on similar behaviours with respect to *Z*. In every case, functions have been found that best fit the refined values of each of the parameters as a function of *Z* and these are written explicitly into each plot.

Parameters *m* and *n* are fitted quite well by the function:

$$l = t_l Z^{u_l}, \qquad (11)$$

where $l = m$ or n and t_l and u_l are refined in the fit of equation 11 to *m* and *n* versus *Z*. Parameters *b, c* and *h* in equation 10 have general trends in terms of *Z* that can be fitted with:

$$l = \frac{t_l}{1+u_l Z^{v_l}}, \tag{12}$$

where l = *b, c* or *h* and the refinable fit parameters are t_l, u_l and v_l. The result of each fit of equation 11 to *m* and *n* versus *Z* is written into the first graph in figure 5 for the red and blue sets respectively, whilst the results of each fit of equation 12 to *b, c* and *h* versus *Z* is written into the second graph in figure 5 for the red, blue and green sets respectively. Parameters *d* and *g* from equation 10 were fitted with constants, d = 1.83 and g = 2.00, as shown in the third graph in figure 5. That *g* is 2 is rather unsurprising because this confirms the expected even parity of the Lorentzian component in $f_j^{'local}(Z_j, B_j, E_0, s)$ as a function of *s*.

The plots of the individual points determined by fitting equation 10 to $f_j^{'B\&K}(Z_j,\ 0.05\ \text{Å}^2 \le B_j \le 2.0\ \text{Å}^2,\ E_0 = 200\ \text{keV},\ 0\ \text{Å}^{-1} \le s \le 6\ \text{Å}^{-1})$ for each element *j* show varying degrees of oscillation in addition to the general trends fitted by equation 11 in the cases of *m, n, b, c* and *h* and constants in the cases of *d* and *g*. These oscillations are much more pronounced for *b, c, h, d* and *g* than for *m* and *n*. An additional improvement in the fits might be obtained if a sine component were incorporated into each of the fits; however, given the periods of the oscillations seem to change with increasing *Z* as well and that the oscillations are not particularly regular in form, it was decided that the replacement functions for all seven parameters *m, n, b, c, h, d* and *g* should be kept as simple as possible if this does not result in a large deterioration in the fit of equation 10 to $f_j^{'B\&K}(Z_j,\ 0.05\ \text{Å}^2 \le B_j \le 2.0\ \text{Å}^2,\ E_0 = 200\ \text{keV},\ 0\ \text{Å}^{-1} \le s \le 6\ \text{Å}^{-1})$ averaged over all elements *j*.

Replacing *m, n, b, c, h, d* and *g* in equation 10 with the fitted functions detailed above results in:

$$f_j^{'local}(Z_j, B_j, E_0, s) = k e^{-B_j s^2}\left(0.5 + \frac{e^{-0.0041225\sqrt{E_0}}}{1-e^{-0.0041225\sqrt{E_0}}}\right) \tag{13}$$

$$\left(0.0012567 Z_j^{1.6191} B_j^{\frac{0.64579}{1+0.079647 Z_j^{0.40196}}} e^{-\frac{2.7964 s^{1.83}}{1+0.18088 Z_j^{0.48919}}} + \frac{0.00034161\left(1.66517 Z_j^{1.4092} + Z_j^{2.04297}\right) s^2}{1+0.60054 Z_j^{0.63377} + 3.5729 s^4}\right).$$

A proportionality constant, *k*, has been introduced as a means of fitting equation 13 to $f_j^{'B\&K}(Z_j,\ 0.05\ \text{Å}^2 \le B_j \le 2.0\ \text{Å}^2,\ E_0 = 200\ \text{keV},\ 0\ \text{Å}^{-1} \le s \le 6\ \text{Å}^{-1})$ for each element *j* in order to check of how well the function above approximates the entire *ATOM* subroutine. Repeating the fitting process for all elements returned $\bar{\chi}$ = 0.044±0.006, which is only very slightly increased in overall mismatch compared to the performance of equation 10 where $\bar{\chi}$ = 0.0407±0.0007 with seven variable parameters. Although the value of *k* was expected to be constant and close to 1 for all elements, this proportionality parameter added for the sake of the fitting process was seen to vary with respect to atomic number *Z*. This is shown in figure 6. This parameter is likely to be absorbing errors incurred by the substitution of parameters *m* and *n* in equation 10 with the fitted functions whose form in given in equation 11 because the fits (as seen in figure 5) were by no means perfect.

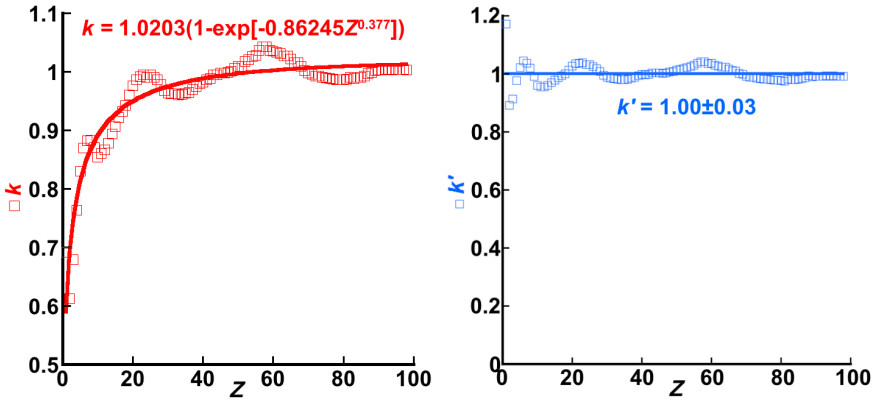


Figure 6 A plot of the proportionality constants *k* in equation 13 and *k'* in equation 15 as a function of *Z*. The general trend in *k* with respect to *Z* is fitted by equation 14, yielding the values of the parameters written into the equation in the plot at left. In contrast *k'* is seen to oscillate about 1.

The form of the function fitted to *k* versus *Z* is:

$$k = t\left(1 - e^{-uZ^v}\right), \tag{14}$$

where *t, u* and *v* are fitting parameters whose final values are given in the left-most plot in figure 6. Substitution of this result into equation 13 returns:

$$f_j'^{local}\left(Z_j, B_j, E_0, s\right) = k' e^{-B_j s^2}\left(0.5 + \frac{e^{-0.0041225\sqrt{E_0}}}{1-e^{-0.0041225\sqrt{E_0}}}\right)\left(1 - e^{-0.86245 Z_j^{0.377}}\right) \tag{15}$$

$$\left(0.0012822 Z_j^{1.6191} B_j^{\frac{0.64579}{1+0.079647 Z_j^{0.40196}}} e^{-\frac{2.7964 s^{1.83}}{1+0.18088 Z_j^{0.48919}}} + \frac{0.00034854\left(1.66517 Z_j^{1.4092} + Z_j^{2.04297}\right)s^2}{1+0.60054 Z_j^{0.63377} + 3.5729 s^4}\right),$$

with the value of $t = 1.0203$ absorbed into the coefficients of the two terms in the main brackets of equation 15. Again, a proportionality constant, *k'*, is retained to test equation 15 against $f_j'^{B\&K}(Z_j,$ 0.05 Å$^2 \leq B_j \leq$ 2.0 Å^2, $E_0 = 200$ keV, 0 Å$^{-1} \leq s \leq 6$ Å^{-1}) from the *ATOM* subroutine. This time, the graph of *k'* versus *Z* simply oscillates about *k'* = 1 as seen in the plot on the right in figure 6. As with previous parameters fitted with equations, the oscillations are not dealt with and it is at this point that the determination of $f_j'^{local}(Z_j, B_j, E_0, s)$ concludes.

3. Discussion

Revisiting equation 1, where the full expression for phenomenological absorption separates the components that are non-local and local to each atom, substitution of equation 15 for the local component gives:

$$f_j'\left(Z_j, B_j, E_0, s\right) = A f_j'^{non-local}\left(Z_j, B_j, E_0, s\right) + C e^{-B_j s^2}\left(0.5 + \frac{e^{-0.0041225\sqrt{E_0}}}{1-e^{-0.0041225\sqrt{E_0}}}\right)\left(1 - e^{-0.86245 Z_j^{0.377}}\right)$$

(15)

$$\left(0.0012822Z_j^{1.6191}B_j^{\frac{0.64579}{1+0.079647Z_j^{0.40196}}}e^{-\frac{2.7964s^{1.83}}{1+0.18088Z_j^{0.48919}}}+\frac{0.00034854\left(1.66517Z_j^{1.4092}+Z_j^{2.04297}\right)s^2}{1+0.60054Z_j^{0.63377}+3.5729s^4}\right), \quad (16)$$

where the parameter *k'* in equation 15 is replaced by parameter *C* from equation 1. At this point, the non-local component is replaced by a Dirac delta function in *s*:

$$f_j'^{non-local}(Z_j, B_j, E_0, s) = ke^{-10^{12}s^2}, \quad (17)$$

where the multiplier of 10^{12} in the exponent is arbitrarily chosen just because it is a large number with respect to the range of *s* over which $f_j'(Z_j, B_j, E_0, s)$ is considered significant. The expression involves a proportionality constant, *k*, which can be absorbed into the parameter *A* in equation 16. Using a Dirac delta function for $f_j'^{non-local}(Z_j, B_j, E_0, s)$ makes the approximation that the sum of all of the non-local contributions to the phenomenological absorption can be considered as adding a constant to $f_j'(Z_j, B_j, E_0, s = 0)$. This is equivalent to saying that the non-local contributions are uniformly distributed at all distances from the atoms in real space. Testing this against experimental data using QCBED (reference paper II) is an opportunity for verifying how well this approximation holds in reality.

Substituting equation 17 into equation 16 yields:

$$f_j'(Z_j, B_j, E_0, s) = Ae^{-10^{12}s^2} + Ce^{-B_j s^2}\left(0.5+\frac{e^{-0.0041225\sqrt{E_0}}}{1-e^{-0.0041225\sqrt{E_0}}}\right)\left(1-e^{-0.86245Z_j^{0.377}}\right)$$

$$\left(0.0012822Z_j^{1.6191}B_j^{\frac{0.64579}{1+0.079647Z_j^{0.40196}}}e^{-\frac{2.7964s^{1.83}}{1+0.18088Z_j^{0.48919}}}+\frac{0.00034854\left(1.66517Z_j^{1.4092}+Z_j^{2.04297}\right)s^2}{1+0.60054Z_j^{0.63377}+3.5729s^4}\right), \quad (18)$$

with *k* from equation 17 absorbed into parameter *A* in this equation. This represents a two-parameter model for all phenomenological absorption that can be refined by an experimental technique such as QCBED. However, if one were to apply equation 18 in its present form, the values of *A* and *C* obtained from refinements would not provide an immediate indication of the relative contributions of $f_j'^{non-local}(Z_j, B_j, E_0, s)$ and $f_j'^{local}(Z_j, B_j, E_0, s)$. This can be seen if one considers $f_j'(Z_j, B_j, E_0, s = 0)$:

$$f_j'(Z_j, B_j, E_0, s = 0) =$$

$$A + 0.0012822C\left(0.5+\frac{e^{-0.0041225\sqrt{E_0}}}{1-e^{-0.0041225\sqrt{E_0}}}\right)\left(1-e^{-0.86245Z_j^{0.377}}\right)Z_j^{1.6191}B_j^{\frac{0.64579}{1+0.079647Z_j^{0.40196}}}. \quad (19)$$

The parameter *C* is modified by a factor that is dependent on Z_j, B_j and E_0, whilst *A* is not. Therefore equation 18 must be factorised in a way that both *A* and *C* are modified by the same factors that are independent of *s*. This factorisation transforms equation 18 into:

$$f_j'(Z_j, B_j, E_0, s) = 0.0012822e^{-B_j s^2}\left(0.5+\frac{e^{-0.0041225\sqrt{E_0}}}{1-e^{-0.0041225\sqrt{E_0}}}\right)\left(1-e^{-0.86245Z_j^{0.377}}\right)Z_j^{1.6191}B_j^{\frac{0.64579}{1+0.079647Z_j^{0.40196}}}$$

$$\left(Ae^{-10^{12}s^2} + C\left(e^{-\frac{2.7964s^{1.83}}{1+0.18088Z_j^{0.48919}}} + \frac{\left(0.27183Z_j^{0.42387} + 0.45264Z_j^{-0.2099}\right)s^2}{\left(1+0.60054Z_j^{0.63377}+3.5729s^4\right)B_j^{\frac{0.64579}{1+0.079647Z_j^{0.40196}}}}\right)\right). \quad (20)$$

In this form, at $s = 0$,

$$f_j'(Z_j, B_j, E_0, s=0) = 0.0012822(A+C)\left(0.5 + \frac{e^{-0.0041225\sqrt{E_0}}}{1-e^{-0.0041225\sqrt{E_0}}}\right)\left(1 - e^{-0.86245Z_j^{0.377}}\right)Z_j^{1.6191}B_j^{\frac{0.64579}{1+0.079647Z_j^{0.40196}}}, \quad (21)$$

and therefore, the magnitudes of A and C obtained experimentally are directly comparable and representative of the relative magnitudes of the non-local and local contributions to phenomenological absorption respectively.

In principle, it would also be easy to further segment the contributions into Gaussian and Lorentzian contributions to the total phenomenological absorption. If this is desired, then equation 20 could be re-written as:

$$f_j'(Z_j, B_j, E_0, s) = 0.0012822e^{-B_j s^2}\left(0.5 + \frac{e^{-0.0041225\sqrt{E_0}}}{1-e^{-0.0041225\sqrt{E_0}}}\right)\left(1 - e^{-0.86245Z_j^{0.377}}\right)Z_j^{1.6191}B_j^{\frac{0.64579}{1+0.079647Z_j^{0.40196}}}$$

$$\left(Ae^{-10^{12}s^2} + C_G e^{-\frac{2.7964s^{1.83}}{1+0.18088Z_j^{0.48919}}} + C_L\frac{\left(0.27183Z_j^{0.42387} + 0.45264Z_j^{-0.2099}\right)s^2}{\left(1+0.60054Z_j^{0.63377}+3.5729s^4\right)B_j^{\frac{0.64579}{1+0.079647Z_j^{0.40196}}}}\right), \quad (22)$$

where C_G and C_L are variable coefficients for the Gaussian and Lorentzian contributions respectively. This is taken to be the most general form of the function for fully describing phenomenological absorption as empirically derived in the present work. Equation 20 is simply the case where $C = C_G = C_L$.

4. Conclusion

Using Bird's and King's *ATOM* subroutine, the present work has empirically developed a functional approximation to the atom-localised contribution to phenomenological absorption (see equation 15) and followed this up with an equation that approximates all contributions to absorption (see equations 20 and 22). This functional approximation can be used to replace the *ATOM* subroutine with a single line of code, but more importantly, it is written in a form where all contributions are separated such that their individual relative magnitudes could be refined using experimental data – in a QCBED experiment for example.

With recent developments that allow TEM data collected without electron-optical energy filtering to be used quantitatively in techniques like QCBED, the availability of a fully flexible phenomenological absorption model where the non-local and local contributions can be refined as independent components becomes important. Incorporating equation 22 into the calculations of

scattered intensities reduces the number of assumptions about what can and cannot be absorbed into the normalisation process (Nakashima & Muddle, 2010b). The application of the presently determined phenomenological absorption function in differential QCBED without energy filtering removes errors incurred by using the Bird and King model via the *ATOM* subroutine because the normalisation process no longer needs to compensate for the unaccounted components. Furthermore, as was pointed out in the example of aluminium, the *ATOM* subroutine often returns negative values for the absorption factor at intermediate values of $s = (\sin\theta)/\lambda$. This is not the case for the present function as its form prohibits negative values of $f_j'(Z_j, B_j, E_0, s)$.

In the sequel paper (Nakashima *et al*., 2018) equation 22 and its more constrained equivalent, equation 20, are used to examine the effects of electron energy (E_0), temperature (B), crystal orientation (s), atomic number (Z), specimen thickness, energy filtering and not energy filtering, on the refined magnitudes of each of the components that contribute to the total phenomenological absorption.

Acknowledgements TL thanks the Faculty of Engineering, the Department of Materials Science and Engineering and the Monash Centre for Electron Microscopy for funding his Ph.D. candidature and scholarships. PN is grateful to Prof. P.C. O'Gamez for providing critical perspectives in the development of this work.

References

Allen, L. J., D'Alfonso, A. J. & Findlay, S. D. (2015). *Ultramicroscopy* **151**, 11-22.

Allen, L. J. & Rossouw, C. J. (1990). *Phys. Rev.* B**42**, 11644-11654.

Allen, L. J., Faulkner, H. M. L., Oxley, M. & Paganin, D. (2001). *Ultramicroscopy* **88**, 85-97.

Anstis, G. R. (1996). *Acta Cryst.* A**52**, 450-455.

Bird, D. M. & King, Q. A. (1990). *Acta Cryst.* A**46**, 202-208.

Bird, D. M. & Saunders, M. (1992). *Acta Cryst.* A**48**, 555-562.

Buxton, B. F. & Loveluck, J. E. (1977). *J. Phys.* C**10**, 3941-3958.

Cosgriff, E. C. & Nellist, P. D. (2007). *Ultramicroscopy* **107**, 626-634.

Deininger, C., Necker, G. & Mayer, J. (1994). *Ultramicroscopy* **54**, 15-30.

Friis, J., Jiang, B., Spence, J. C. H., Marthinsen, K. & Holmestad, R. (2004). *Acta Cryst.* A**60**, 402-408.

Gajdardziska-Josifovska, M., McCartney, M. R., DeRuijter, W. J., Smith, D. J., Weiss, J. K. & Zuo, J. M. (1993). *Ultramicroscopy* **50**, 285-299.

Hall, C. R. & Hirsch, P. B. (1965). *Proc. R. Soc. Lond.* A**286**, 158-177.

Hashimoto, H., Howie, A. & Whelan, M. J. (1962). *Proc. R. Soc. Lond.* A**269**, 80-103.

Holmestad, R., Birkeland, C. R., Marthinsen, K., Høier, R. & Zuo, J. M. (1999). *Microsc. Res. Techn.* **46**, 130-145.

Hosokawa, F., Shinkawa, T., Arai, Y. & Sannomiya, T. (2015). *Ultramicroscopy* **158**, 56-64.

Humphreys, C. J. & Hirsch, P. B. (1968). *Philos. Mag.* **18**, 115-122.

Ichimiya, A. (1985). *Jpn. J. Appl. Phys.* **24**, 1579-1580.

Ichimiya, A. & Lehmpfuhl, G. (1988). *Acta Cryst.* A**44**, 806-809.

Ishizuka, K. (2002). *Ultramicroscopy* **90**, 71-83.

Jansen, J., Tang, D., Zandbergen, H. W. & Schenk, H. (1998). *Acta Cryst.* A**54**, 91-101.

Midgley, P. A. & Saunders, M. (1996). *Contemp. Phys.* **37**, 441-456.

Mobus, G. & Rühle, M. (1993). *Optik* **93**, 108-118.

Nakashima, P. N. H. (2007). *Phys. Rev. Lett.* **99**, 125506.

Nakashima, P. N. H. & Muddle, B. C. (2010a). *J. Appl. Cryst.* **43**, 280-284.

Nakashima, P. N. H. & Muddle, B. C. (2010b). *Phys. Rev.* B**81**, 115135.

Nakashima, P. N. H., Shao, Y. T., Liu, T., Bourgeois, L. & Zuo, J. M. (2018). *Acta Cryst.* A**XX**, xxx-yyy.

Neish, M. J., Lugg, N. R., Findlay, S. D., Haruta, M., Kimoto, K. & Allen, L. J. (2013). *Phys. Rev.* B**88**, 115120.

Palatinus, L., Petricek, V. & Correa, C. A. (2015). *Acta cryst.* A**71**, 235-244.

Peng, L. M., Dudarev, S. L. & Whelan, M. J. (1998). *Phys. Rev.* B**57**, 7259-7265.

Peng, L. M. & Whelan, M. J. (1992). *Surf. Sci.* **268**, L325-L329.

Pennington, R. S., Coll, C., Estrade, S., Peiro, F. & Koch, C. T. (2018). *Phys. Rev.* B**97**, 024112.

Pannycook, S. J. & Jesson, D. E. (1992). *Acta Metall. Mater.* **40**, S149-S159.

Radi, G. (1970). *Acta Cryst.* A**26**, 41-56.

Rossouw, C. J., Miller, P. R., Drennan, J & Allen, L. J. (1990). *Ultramicroscopy* **34**, 149-163.

Rossouw, C. J. & Miller, P. R. (1993). *Philos. Mag.* B**67**, 733-745.

Sang, X. H, Kulovits, A. & Wiezorek, J. M. K. (2010). *Acta Cryst.* A**66**, 685-693.

Saunders, M., Fox, A. G. & Midgley, P. A. (1999). *Acta Cryst.* A**55**, 471-479.

Shao, Y. T. & Zuo, J. M. (2017). *Acta Cryst.* B**73**, 708-714.

Streltsov, V. A., Nakashima, P. N. H. & Johnson, A. W. S. (2003). *Microsc. Microanal.* **9**, 419-427.

Tabira, Y., Withers, R. L., Minervini, L. & Grimes, R. W. (2000). *J. Sol. St. Chem.* **153**, 16-25.

Tsuda, K., Mitsuishi, H., Terauchi, M. & Kawamura, K. (2007). *J. Electron Microsc.* **56**, 57-61.

Tsuda, K. & Tanaka, M. (1999). *Acta Cryst.* A**55**, 939-954.

Twesten, R. D., Gibson, J. M. & Hellman, O. C. (1997). *Surf. Rev. & Lett.* **4**, 245-269.

Zuo, J. M. (2004). *Rep. Prog. Phys.* **67**, 2053-2103.

Zuo, J. M. (1993). *Acta Cryst.* A**49**, 429-435.

Zuo, J. M. & Spence, J. C. H. (1993). *Philos. Mag.* A**68**, 1055-1078.

D.2. Paper 2

Electron Bonding Distribution in Copper (Draft)

Tianyu Liu, Andrew E. Smith, Xiahan Sang, Laure Bourgeois, Jorg M. Wiezorek, Philip N. H. Nakashima

Abstract

Copper is one of the most anisotropic metals. Therefore, the bonding electron distribution in copper will have a significant trend, which cannot be simply described by the free electron gas model. However, results from previous experimental measurements and theoretical calculations had poor agreements. Here we show our findings of the bonding electron distribution in copper yielded by quantitative convergent beam electron diffraction and density function theory calculation.

Introduction

As taught in high school chemistry, strong chemical bonds are conventionally classified into three bonding types: covalent bond, ion bond and metal bond, by electronegativity. Nevertheless, this categorisation is questioned by some researchers [1-4] who suggest metallic bond being a subset of covalent bond. They indicate that the difference of electronegativity properties is more typical between ion bond and covalent/metal bond, while that between covalent and metallic bond is less significant, and many other aspects of electronic structure show no distinguishing difference between covalent bond and metallic bond. Apart from electronegativity that used to differ bonding types, however, due to limited resolution of the available techniques, detailed characteristics like interatomic bonding positions, shapes and charge density distribution of metallic bond is still very limited to date, which is mostly based on the classic free-electron cloud model that proposed in 1900 which is also taught in high school.

Copper, being the first metal and the first alloy (incorporated with tin) used by human, is still one of greatest production metals and of great interests in the research community. Inspired by the successful bonding characterisation of aluminium [5], this work will try to reveal certain bonding characteristics of copper aluminium via quantitative convergent beam election diffraction (QCBED) and density function theory (DFT)

calculation. By comparing the bonding difference between copper and aluminium, this work will also explain their difference in elastic isotropy.

The concept of bonding is defined by the deformation charge density $\Delta\rho$

$$\Delta\rho = \rho_{\text{real}} - \rho_{\text{IAM}} \qquad (1)$$

where ρ_{real} is the experimentally measured or theoretically calculated electron density distribution of real-life materials, and ρ_{IAM} is the electron density distribution of the theoretical ideally independent atom model (IAM) where bonding is missing. For simple and convenient representation of the three dimensional distribution of the electron density in a crystal, it can be expressed as a Fourier sum,

$$\rho(\mathbf{r}) = \sum_{\mathbf{g}} F_{\mathbf{g}} \cdot \exp(-2\pi i \mathbf{g} \cdot \mathbf{r}) \qquad (2)$$

where $F_{\mathbf{g}}$ is the Fourier coefficients of the electron distribution, also known as (x-ray) structure factors. $\mathbf{g}$ is the reciprocal lattice vector and $\mathbf{r}$ is the real space vector in the unit cell. By combining Equations 1 and 2, we get

$$\Delta\rho(\mathbf{r}) = \sum_{\mathbf{g}} (F_{\mathbf{g}}^{real} - F_{\mathbf{g}}^{IAM}) \cdot \exp(-2\pi i \mathbf{g} \cdot \mathbf{r}) \qquad (3)$$

Hence, the bonding in a crystal can be expressed as structure factors differences between its real structure factors and ideal IAM structure factors.

For copper, most of the bonding information is contributed by the three lowest-order structure factors, F_{111}, F_{200} and F_{220}, while the information lies in typically less than 1% differences between the real and IAM structure factors. For higher-order structure factors the charge deformation contribution become insignificant. Therefore, the key to solve the interatomic bonding distribution is to determine the low-order structure factors with high accuracy.

Conventional x-ray diffraction, Pendellösung x-ray, neutron diffraction, critical voltage and QCBED were used to measure the structure factors of copper in the last 80 years [6-10]. Experiments like conventional x-ray and neutron diffractions that based on the Kinematical (single scattering) theory suffer from extinction and anomalous scattering errors in low-order reflections, and critical voltage can only measure the ratio between two structure factors, thus potentially Pendellösung x-ray and QCBED, based on the Dynamical (multiple scattering) theory, have great advantages to measure low-order structure factors accurately for bonding density distribution in copper. In actual practices, however, the Pendellösung method is limited by its severe specimen requirements which need large perfect single crystals to yield accurate results. While for QCBED, the electron beam is focused into a small spot on the specimen, which makes selecting a defect-free (e.g. single grain and no dislocation) region on a specimen to produce perfect diffractions possible.

Experiments and DFT calculation

In this work, we conducted experiments with two different methods of QCBED that based on the Bloch wave theory and the multislice theory respectively, and also used WIEN2K to perform a density function theory calculation. Being the two main steams of QCBED, the Bloch wave theory is preferred for analysis of perfect single crystals with a small unit cell due to its flexibility and accuracy, while the multislice theory has higher efficiency in computing time in a many-beam situation.

Multislice

For the multislice experiments, 118 diffraction patterns were collected with accelerate voltage of 200 kV and 120 kV at room temperature (293 K), near three different major zone-axe (<100>, <110> and <112>, sensitive to target low-order structure factors), from different regions (thickness ranges from 600 to 2000 Å) of multiple polycrystalline samples. The samples were made of 99.9999% (6N) pure copper, and annealed at 530 °C for 8 hours and then 150 °C for 4 hours to remove dislocations. Point spread function correction, noise characterisation and angular differential were used to eliminate point spread effect, digital signal noise and thermal diffuse scattering signals of collected CBED patterns. A modified version of the angular differential QCBED program was used for pattern-matching. The lattice parameter used for refinement is 3.61496 Å, and the Debye-Waller factor is 0.544 $Å^2$.

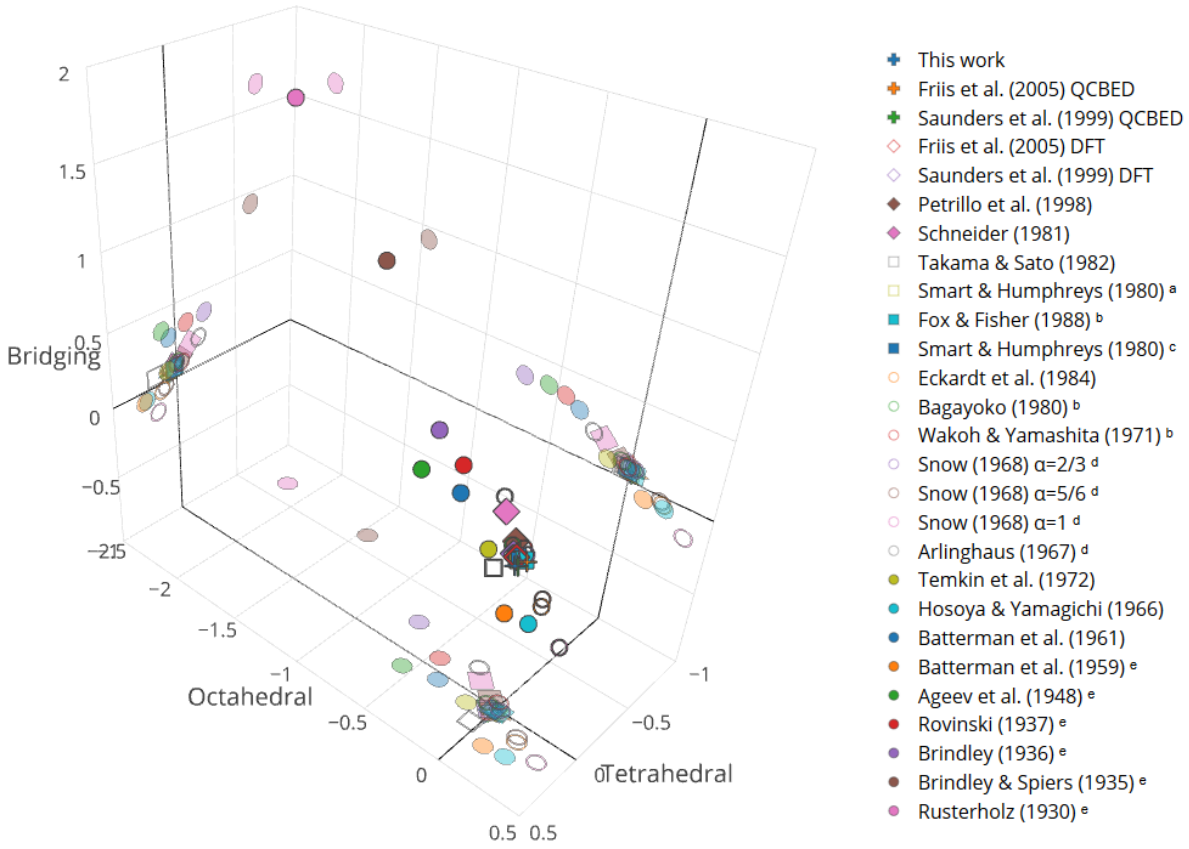
Bloch wave

(The Bloch wave method is used by our collaborators, Jorg Wiezorek's group. This paragraph will be written by them.)

DFT calculation

(The DFT calculations are done by Andrew Smith, and the detailed simulation setup is still being modifying.)

Current Results and comparison with previous studies



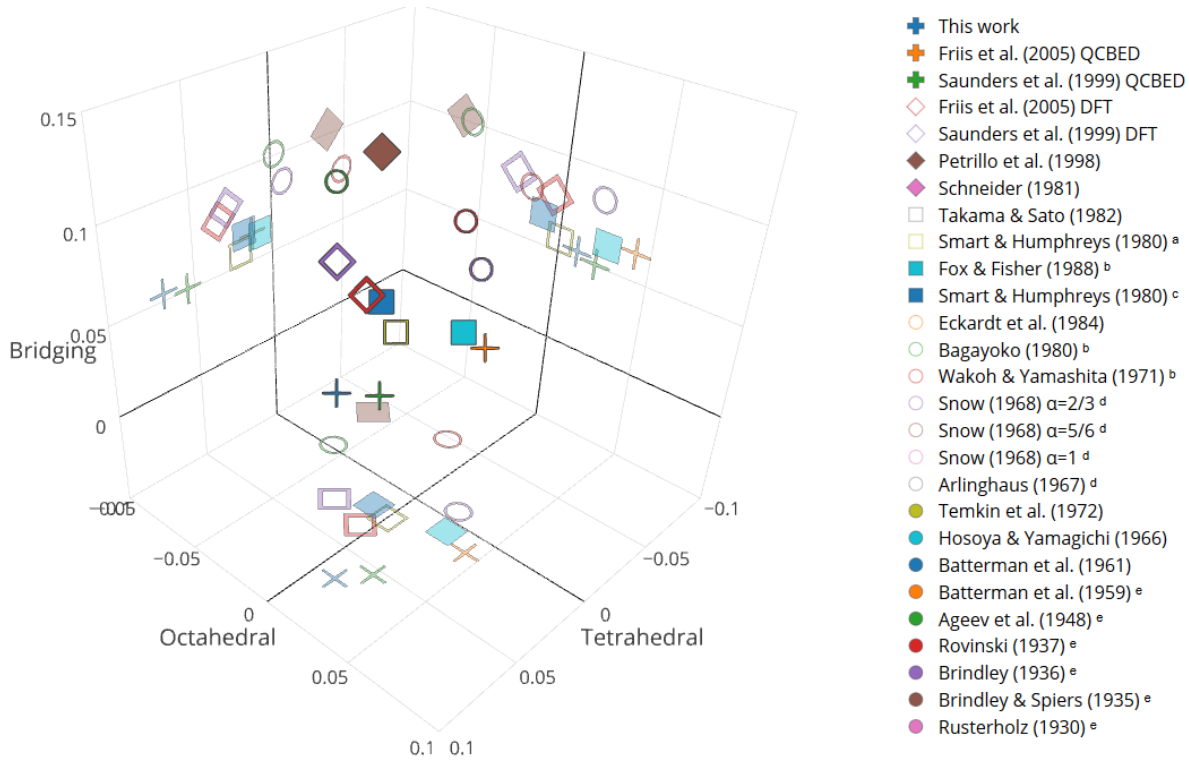


Figure 1 Charge density results of copper from studies since 1930 are summarised and plotted by their bonding density ($e^-/Å^3$) at the octahedral centre, tetrahedral centre and bridging centre in 3-dimension. (Top) Overview of all studies, and (bottom) zoom in view of clustered modern and recent study results. Noted values are taken from: [a] Friis et al., 2003 [11], [b] Tabbernor et al., 1990 [8], [c] Saunders et al., 1999 [7], [d] Temkin et al., 1972 [9] and [e] Sirota, 1969 [10].

In Figure 1 (top), it is shown that previous charge distribution studies in copper have yielded diverse bonding characteristics in both absolute charge intensity and charge distribution. Despite of their absolute intensities, results from more than half of the studies show that the majority of the bonding densities locate between the nearest neighbour atoms; most of these studies further suggest that there are anti-bondings at the tetrahedral and/or octahedral sites. Tetrahedral dominant is also supported by a few studies. Only few band theory studies suggest the major bonding locates at the octahedral site.

Relatively, as shown in Figure 1 (bottom), modern and recent studies have more consistent conclusions: QCBED and DFT researches by Friis et al. in 2005 [6] and Saunders et al. in 1999 [7], corrected neutron diffraction results of Schneider (original experiment in 1981) [12] by Petrillo et al. [13], Pendellösung x-ray experiment by Smart and Humphreys in 1980 [11], the two available critical voltage results in the 1980s [7,8], band theory calculation by Bagayoko in 1980 [14] and QCBED and DFT results of this work have close agreement about the absolute bonding intensity. QCBED results of this work indicate that the dominant bonding in copper locates

at the tetrahedral site, which is in good agreement to Saunders et al. QCBED results [7]. DFT results of this work agree to previous DFT calculations done by Saunders et al. [7] and Friis et al. [6] respectively, and suggest that copper has a bridging-type bonding, and anti-bonding at the octahedral centre. QCBED study by Friis et al. [6] and rescaled neutron diffraction results of Schneider [13] also concluded bridge-type bonding similar to the DFT calculations.

			XRD	Band Theory	Pendellösung	γ-ray	DFT		QCBED		
hkl	s (Å^{-1})	IAM*	Temkin et al. (1972)	Bagayoko (1980)	Takama & Sato (1982)	Schneider et al. (1981)	Saunders et al. (1999)	Friis et al. (2005)	Saunders et al. (1999)	Friis et al. (2005)	This work
111	0.24	22.03	21.93(15)	21.68(35)	21.80(6)	21.51(5)	21.71	21.70	21.78(2)	21.69(3)	21.79(4)
200	0.28	20.64	20.36(15)	20.35(29)	20.28(11)	20.22(4)	20.37	20.38	20.44(2)	20.44(2)	20.42(4)
220	0.39	16.77	16.70(16)	16.62(15)	16.75(8)	16.45(5)	16.66	16.67	16.72(11)	16.68(2)	16.73(6)

Table 1 Selected theoretical and experimental low-order structure factors of copper from key and recent studies.

By comparing the low-order structure factors of our QCBED experiment and QCBED results of Friis et al. [6] and Saunders et al. [7], it is shown that all QCBED experiments have approximate F_{200} and F_{220}, while the structure factor F_{111} of Friis et al. [6] is significantly different from Saunders et al. [7] and our work. The main difference between Friis et al. [6] QCBED experiment and other QCBED experiments is they used the systematic row approach which involves only 1-dimensional line profile fitting, while Saunders et al. [7] and we applied 2-dimensional zone-axis approaches that have more spatial constraint.

Structure factors calculated by DFT tend to be lower compared to QCBED results and more different from the IAM values. This means that DFT calculations suggest a more significant charge deformation than QCBED experimental results.

Discussion

Conclusion

Reference

1 Schön, J. C. Does the Death Knell Toll for the Metallic Bond? *Angewandte Chemie International Edition in English* **34**, 1081-1083, doi:10.1002/anie.199510811 (1995).

2 Allen, L. C. & Burdett, J. K. The Metallic Bond—Dead or Alive? A Comment and a Reply. *Angewandte Chemie International Edition in English* **34**, 2003-2003, doi:10.1002/anie.199520031 (1995).

3 Anderson, W. P., Burdett, J. K. & Czech, P. T. What Is the Metallic Bond? *Journal of the American Chemical Society* **116**, 8808-8809, doi:10.1021/ja00098a050 (1994).

4 Allen, L. C. & Capitani, J. F. What is the Metallic Bond? *Journal of the American Chemical Society* **116**, 8810-8810, doi:10.1021/ja00098a051 (1994).

5 Nakashima, P. N. H., Smith, A. E., Etheridge, J. & Muddle, B. C. The Bonding Electron Density in Aluminum. *Science* **331**, 1583-1586, doi:10.1126/science.1198543 (2011).

6 Friis, J., Jiang, B., Marthinsen, K. & Holmestad, R. A Study Of Charge Density In Copper. *Acta Crystallographica Section A* **61**, 223-230, doi:10.1107/s0108767305001315 (2005).

7 Saunders, M., Fox, A. G. & Midgley, P. A. Quantitative Zone - Axis Convergent - Beam Electron Diffraction (Cbed) Studies Of Metals. I. Structure - Factor Measurements. *Acta Crystallographica Section A* **55**, 471-479, doi:10.1107/S0108767398012604 (1999).

8 Tabbernor, M. A., Fox, A. G. & Fisher, R. M. An Accurate Reappraisal Of The Elemental Form-Factors And Charge-Density Of Copper. *Acta Crystallographica Section A* **46**, 165-170, doi:10.1107/s0108767389011141 (1990).

9 Temkin, R. J., Raccah, P. M. & Henrich, V. E. Experimental Charge-Density Of Copper. *Physical Review B* **6**, 3572-3581, doi:10.1103/PhysRevB.6.3572 (1972).

10 Sirota, N. N. Survey Of Results In The Determination Of X - Ray Intensities And Structure Factors For Metals, Alloys And Covalent Compounds. *Acta Crystallographica Section A* **25**, 223-243, doi:10.1107/S0567739469000386 (1969).

11 Friis, J., Jiang, B., Spence, J. C. H. & Holmestad, R. Quantitative Convergent Beam Electron Diffraction Measurements Of Low-Order Structure Factors In Copper. *Microscopy and Microanalysis* **9**, 379-389, doi:10.1017/s1431927603030319 (2003).

12 Schneider, J. R., Hansen, N. K. & Kretschmer, H. A Charge-Density Study Of Copper By Gamma-Ray Diffractometry On Imperfect Single-Crystals. *Acta Crystallographica Section A* **37**, 711-722, doi:10.1107/s0567739481001599 (1981).

13 Petrillo, C., Sacchetti, F. & Mazzone, G. Relevance of Charge-Density Measurements for High-Precision Calculations. *Acta Crystallographica Section A* **54**, 468-480, doi:10.1107/S0108767398001548 (1998).

14 Bagayoko, D., Laurent, D. G., Singhal, S. P. & Callaway, J. Band structure, optical properties, and compton profile of copper. *Physics Letters A* **76**, 187-190, doi:http://dx.doi.org/10.1016/0375-9601(80)90609-X (1980).

D.3. Paper 3

A FORTRAN function for exact Doyle and Turner X-ray/electron neutral atom scattering factors using cubic spline interpolation

Authors

Tianyu Liu[a], Andrew E. Smith[b], Laure Bourgeois[ac] and Philip N.H. Nakashima[a]*

[a]Department of Materials Science and Engineering, Monash University, Melbourne, VIC, Australia

[b]School of Physics and Astronomy, Monash University, Melbourne, VIC, Australia

[c]Monash Centre for Electron Microscopy, Monash University, Melbourne, VIC, Australia

Correspondence email: philip.nakashima@monash.edu

Synopsis One or two sentences suitable for the Journal contents listing (style: IUCr synopsis).

Abstract Atomic scattering factors are the very fundamental key to electron bonding computations as a pseudo-wavefunctions solution. During these computations, scattering factors at specific scattering angles are required and practically derived from interpolation to the tabulated values published in the International Tables for Crystallography (Brown *et al.*, 2006; Colliex *et al.*, 2006). This paper provides a FORTRAN function of an alternative method, which uses cubic spline for interpolation. Unlike other existing popular parameterisation interpolations, this new interpolation features scattering factor curves with exact fittings to the tabulated values. Simulations by quantitative convergent beam electron diffraction pattern-matching with different interpolation methods are also provided to show the improvement on electron bonding measuring with the new interpolation.

Keywords: X-ray scattering factor; electron scattering factor; scattering factor parameterisation.

1. Introduction

Computational solutions to high energy electron microscopy diffraction and electron bonding calculations of crystalline materials are based on crystal potentials expressible in terms of atomic scattering factors. The most widely used atomic scattering factors were published by Doyle and Turner (1968). Their scattering factors with parameterisation interpolations have provided standard values for comparison between calculations and experiments. In particular, they provide a framework for the study of bonding by a treatment of the isolated atom/ion bonding and potential numerical calculation. Their publication (Doyle & Turner, 1968) covers 76 common atoms and ions, giving tables with 27 values of X-ray and electron scattering factors respectively for each atom and ion from 0 to 6 Å^{-1} of s ($\sin\theta/\lambda$). X-ray scattering factors from 0 to 2 Å^{-1} of s for other atoms and ions were calculated by Cromer and Waber (1968), and later the neutral atoms in these tables were extended to 6

Å^{-1} of s by Fox *et al.* (1989). This collection of X-ray and electron scattering factors was later published in the International Tables for Crystallography (Brown *et al.*, 2006; Colliex *et al.*, 2006), forming tables that cover atoms and ions from number 1 to 98. The neutral atoms tables hold up to 62 values from 0 to 6 Å^{-1} of s for both X-ray and electron scattering factors; while the ion tables hold up to 56 values from 0 to 2 Å^{-1} of s. Note that for atoms and ions that are not published by Doyle and Turner (1968), their electron scattering factors were converted from their X-ray values using the Mott-Bethe formula.

In practice, these scattering factor tables will be interpolated to return either X-ray or electron scattering factors for desired s. There are mainly two solutions: the original and more popular method called parameterisation (Doyle & Turner, 1968; Fox *et al.*, 1989; Weickenmeier & Kohl, 1991; Peng *et al.*, 1996; Kirkland, 1998; Lobato & Van Dyck, 2014), and polynomial interpolation (Bird & King, 1990).

Doyle and Turner (1968) also provided their parameterisation along with their 76 tabulated atoms/ions scattering factors, which uses four sets of Gaussian summations, for both X-ray and electron. Parameterisation then became the most popular interpolation method for scattering factors. Improved high angle electron scattering factor parameterisation was later published by Fox *et al.* (1989) for $2 \leq s \leq 6$ Å^{-1}, while for $s < 2$ Å^{-1}, it is suggested to use the Mott formula to derive electron scattering factors from the X-ray values. Similarly, Weickenmeier and Kohl (1991) published their 6-Gaussian-summations parameterisation for electron scattering factors, intending to give better fitting at high scattering angles, covering neutral atoms from number 1 to 98. A FORTRAN subroutine *FSCATT* with their parameterisation implemented is also provided. The two parameterisations, one aims for $0 \leq s \leq 2$ Å^{-1} and the other for $0 \leq s \leq 6$ Å^{-1}, with 5 Gaussian summations by Peng *et al.* (1996) for electron scattering factors of the 98 neutral atoms and their ions is currently included in the latest International Table of Crystallography (Colliex *et al.*, 2006) as an updated standard. Note that only the $0 \leq s \leq 6$ Å^{-1} parameterisation will be discussed for Peng *et al.* (1996) in this paper. There are also parameterisations using functions other than pure Gaussian. Kirkland (1998) has used 3 Lorentzian and 3 Gaussian summations to fit the electron scattering factors. As the most recent publication, Lobato and Van Dyck (2014) have provided a 5-Lorentzian summations and concluded the best fitting to the electron scattering factor table.

Apart from parameterisation, a less popular interpolation method is polynomial interpolation. Bird and King (1990) published a FORTRAN subroutine, *ATOM*, which primarily aims to return both the real and imaginary components of electron atomic scattering factors, for neutral atoms included in the original publication of Doyle and Turner (1968). For the real component, *ATOM* uses Lagrange interpolating cubic to interpolate the electron scattering factors at specific s. Due to the nature of polynomial interpolation, *ATOM* can return a curve of electron scattering factors with perfect fitting to its embedded tabulated values.

Although all interpolation methods mentioned above intend to interpolate and fit either the X-ray or electron scattering factors by Doyle and Turner (1968), they still tend to yield significantly different scattering factors during interpolation, even for scattering factors at certain s that have tabulated values. If electron bonding and potential calculations involve such inaccurate scattering factors, the introduced errors may accumulate and vastly alter the calculated results. For example, in the routine of quantitative convergent beam electron diffraction (QCBED) of bonding density calculation, interpolated scattering factors of Doyle and Turner (1968) are used to derive bonding and potentials for ideal isolated atom models (IAM). During QCBED pattern-matching, only the several lowest order scattering factors of the IAM will be refined, while hundreds of higher order scattering factors are fixed to the interpolated values. The main reason is because major bonding information in real materials concentrates in the several lowest-order scattering factors, while higher-order scattering factors are theoretically identical to that of the IAM. Another reason is increased refining variables will vastly increase the computation time. If the interpolated scattering factors are not sufficiently accurate, the unchanging higher order scattering factors will have the lower order scattering factors refinement compromised, yielding incorrect results. Moreover, in the final stage of bonding density determination, the QCBED refined scattering factors will be subtracted by the IAM scattering factors to conclude the deformation density caused by the presence of bonding. This implies even if it is possible to refine all the scattering factors during QCBED pattern-matching, the final bonding deformation density cannot be measured correctly without an accurate IAM.

In this paper, a cubic spline interpolation method and its FORTRAN function will be introduced. The biggest advantage of our cubic spline interpolation, compared to the popular parameterisation methods, is guaranteed to result perfect fitting to tabulated scattering factors as it is a polynomial interpolation. Comparing to the FORTRAN subroutine *ATOM* by Bird and King (1990), our function: 1) uses cubic spline to interpolate, hence continuous derivatives, 2) also returns X-ray scattering factors, and 3) uses International Tables for Crystallography (Colliex *et al.*, 2006) values, which has denser tabulated values and covers atom numbers from 1 to 98. Comparison of simulations by QCBED pattern-matching with the new cubic spline and other existing interpolations will be provided to demonstrate the reliability of this new method.

2. Algorithm logic

Our FORTRAN function is designed to return desired X-ray/electron scattering factors with inputs of atomic number (1-98) and s ($0 – 6$ Å^{-1}). Our function covers the tabulated neutral atom X-ray and electron scattering factors in the International Tables for Crystallography (Brown *et al.*, 2006; Colliex *et al.*, 2006), which have 62 values for $0 \leq s \leq 6$ Å^{-1} for most atoms from number 1 to 98 for both X-ray and electron scattering factors. Some obvious errors in the table (*e.g.* mismatch to Doyle and Turner (1968), or local minimum/maximum that breaks curve trend) are corrected or set to be

neglected in the function. For a given s, our function will use the nearest four (two on both sides, except near s = 6 Å^{-1}) tabulated values to interpolate and return the corresponding scattering factor, which guarantees exact matching of the tabulated values and provide a smooth curve with continuous derivatives. While s is close to 0, the spline will assume the curve is symmetrical at s = 0 Å^{-1}.

For atoms whose tabulated electron scattering factors are derived from their X-ray values, our function will first interpolate their tabulated X-ray values and then convert with the Mott formula when their electron scattering factors are required.

3. Results and Discussion

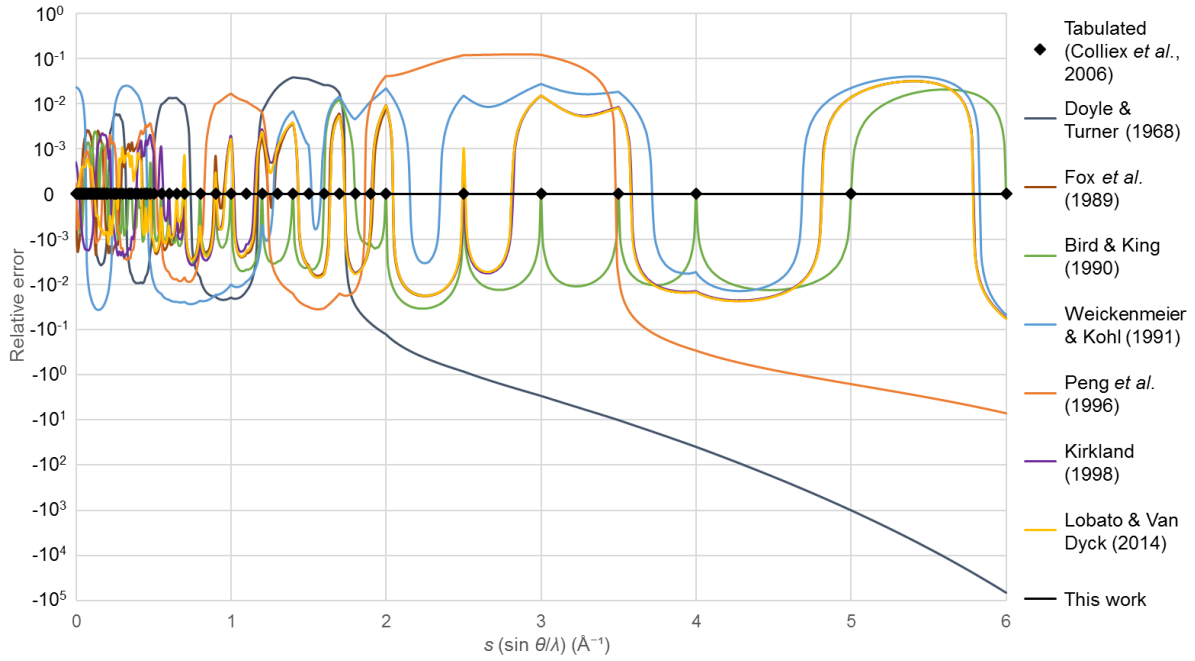


Figure 1 Relative errors of electron scattering factor curves by other methods for aluminium, using our cubic spline scattering factors as the reference. Since the curves of Fox *et al.* (1989), Kirkland (1998) and Lobato & Van Dyck (2014) have close values, they are mostly overlapping each other at higher scattering angles.

As mentioned in the introduction, different parameterisation methods have various performance on fitting the tabulated X-ray and electron scattering factors as the popular solution. Generally, parameterisation with Gaussians have reasonably good fitting to X-ray scattering factors, while Lorentzian and hybrid parameterisation have better fitting to electron scattering factors. However, small errors remain even with the best parameterisation. Another solution to further reduce the curve-fitting errors is to use polynomial interpolation. As shown in Figure 1, our cubic spline curve has perfect fitting to all the tabulated values and Bird and King's Lagrange cubic interpolation has perfect fitting to tabulated values published by Doyle and Turner (1968), because polynomial interpolations are used.

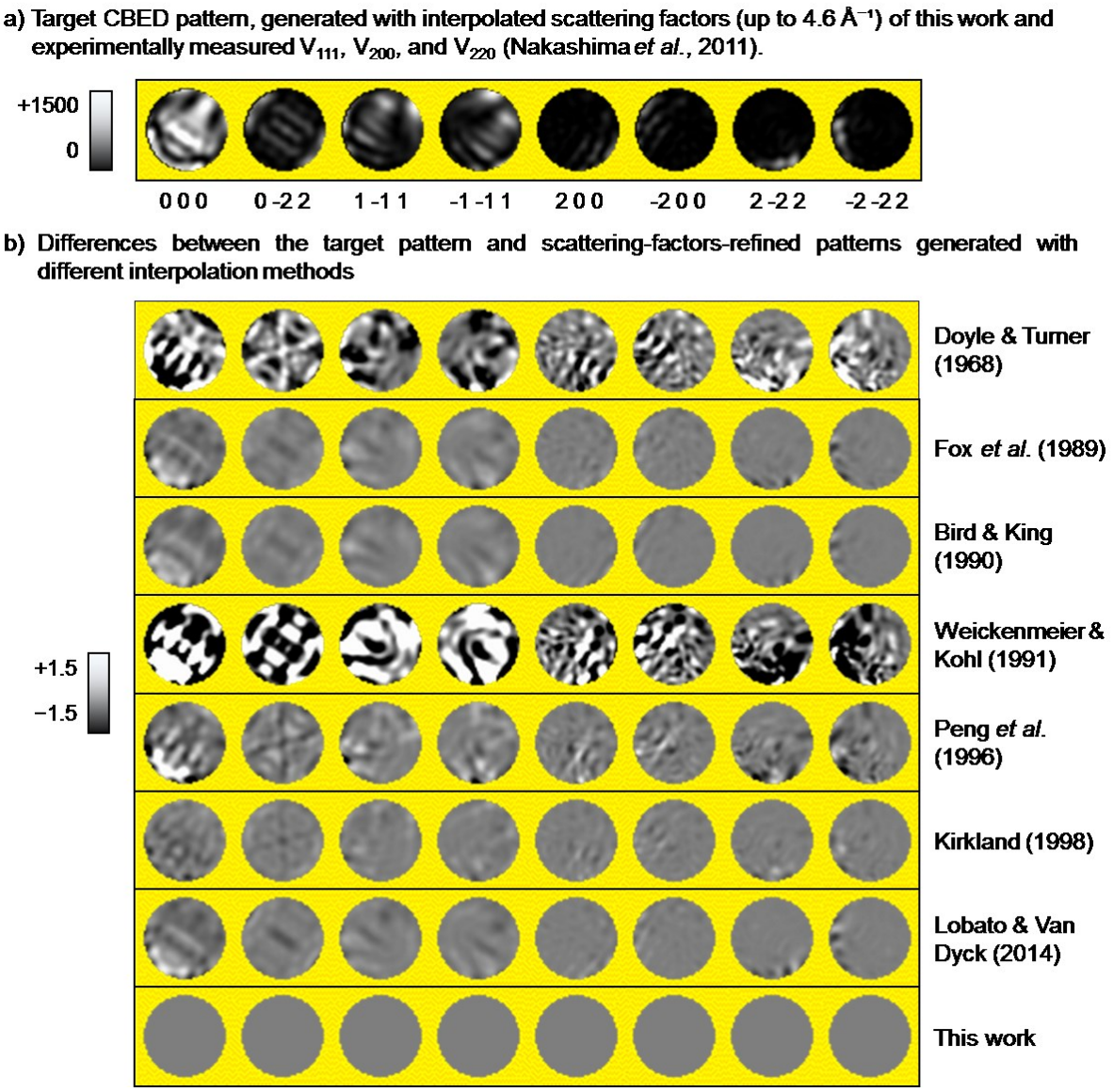


Figure 2 Comparison of CBED pattern-matching results with different scattering factor interpolation methods. a) A noise-free CBED pattern (pure aluminium, near the [110] axis) generated with our cubic spline and measured V_{111}, V_{200}, and V_{220} (Nakashima *et al.*, 2011), which is used as the pattern-matching target, and b) differences between the target pattern and patterns generated by different methods with the three lowest-order scattering factor refined. Due to restrictions of computation time, CBED pattern simulations and pattern-matching were conducted with scattering factors only up to 4.6 $Å^{-1}$.

To further demonstrate the differences of various methods and examine their reliabilities, QCBED simulations of a pure aluminium crystal generated with electron scattering factors of different methods were compared. The target CBED pattern to be refined was generated with our cubic spline and experimentally measured V_{111}, V_{200}, and V_{220} (Nakashima *et al.*, 2011). The QCBED simulations

were conducted with the same routine as Nakashima *et al.* (2011). The three lowest-order structure factors (key to face-centred cubic metal) V_{111}, V_{200}, and V_{220} were refined for each method during pattern-matching. A phenomenological absorption curve was used and also refined. The target and refined CBED pattern calculations only involved scattering factors up to 4.6 $Å^{-1}$ due to computational time restriction. The target and refined CBED patterns are shown in Figure 2. Structure factor refinement results are summarised in Table 1 and Figure 3. a) of Figure 2 is a simulated pure aluminium CBED pattern generated with our cubic spline, which was used as the target pattern for pattern-matching. In b) of Figure 2, it shows that with various methods and their scattering factors, the final calculated patterns can have obvious differences, which demonstrate the significance of even slightly different scattering factors. Although the lowest-order structure factors were refined, higher-order structure factors were fixed while they are known to have different values to that of the target patterns. Therefore, mismatches between refined patterns of different methods and the target pattern will occur, resulting different refined structure factors to the known values. Goodness-of-fit of different methods' pattern-matching are denoted as χ^2 in Table 1. χ^2 is defined as:

$$\chi^2 = \sum_i \left(I_i^{target} - I_i^{calc}\right)^2$$

where . I_i^{target} is the target CBED intensity and I_i^{calc} is the calculated CBED intensity to match the target for the i^{th} pixel. When relating the goodness-of-fit of different methods' scattering factor curve-fitting (in Figure 1) and their CBED patterns structure factor refinements (in Table 1 and Figure 3), it is shown that methods with better fitting to the tabulated scattering factors also have better structure factor refinement results and lower mismatch between the target patterns and refined patterns (except Peng *et al.* (1996), because its scattering factors have good fitting at small *s*, while these simulations involved limited high scattering angle information due to computational time restriction), which can justify the reliability of our cubic spline interpolation.

Table 1 Refinement results of different methods upon a simulated aluminium CBED pattern (shown in Figure 2) generated by our cubic spline scattering factors; electron structure factors in unit of volts.

	Doyle & Turner (1968)	Fox et al. (1989)	Bird & King (1990)	Weicken -meier & Kohl (1991)	Peng et al. (1996)	Kirkland (1998)	Lobato & Van Dyck (2014)	This work	Known value
V_{111}	6.2295	6.2309	6.2303	6.2214	6.2311	6.2318	6.2311	6.2340	6.2340
V_{200}	5.2408	5.2410	5.2405	5.2300	5.2415	5.2420	5.2415	5.2416	5.2416
V_{220}	3.2032	3.2161	3.2143	3.2227	3.2177	3.2166	3.2161	3.2151	3.2151
χ^2	1.5123	0.0321	0.0210	13.7627	0.1322	0.0158	0.0275	0.0000	-

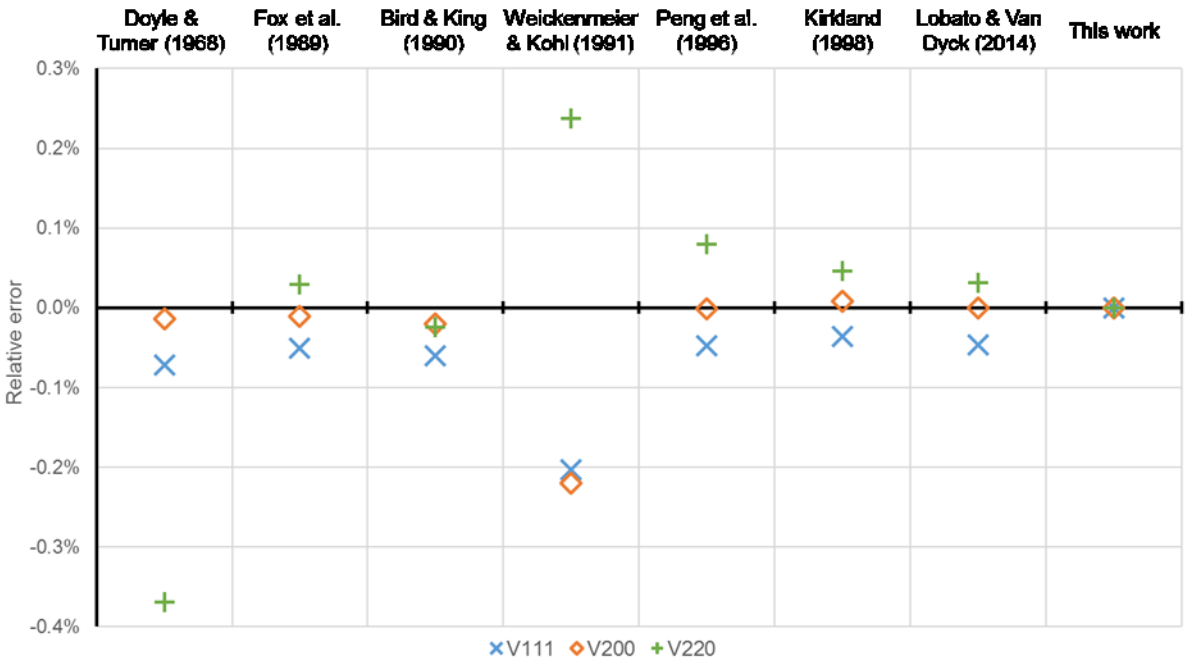


Figure 3 Refinement results of different methods on an aluminium CBED patterns generated by our cubic spline scattering factors (shown in Figure 2).

Moreover, refined low-order scattering factors may compromise the mismatching higher-order scattering factors in order to follow the pattern-matching iteration, resulting in inaccurate scattering factor refinement, which is observed in Table 1 and Figure 3. Such problems will also exist in practical bonding experiments by QCBED. In QCBED bonding experiments, the bonding information of a real material is measured only within the lowest-order structure factors. Higher-order structure factors are identical to that of the un-bonded isolated-atom model that calculated by interpolation of the tabulated scattering factors. Therefore, without an accurate interpolation, the refined low-order scattering factors may be affected by the inaccurate higher-order scattering. Increasing the number of refining structure factors may reduce the effect of an inaccurate interpolation, however, it will multiple the computational resources and time. Eventually, for a bonding experiment, the resulting bonding will require the subtraction of the IAM from the measured experimental result, which makes the accuracy of the IAM vital.

4. Concluding remarks

Our FORTRAN function is capable of interpolating both X-ray and electron scattering factors of neutral atoms from atomic number 1 to 98 tabulated in the International Tables for Crystallography. Noting that for atoms with tabulated electron scattering factors converted from X-ray values by the Mott formula, their electron interpolation will start from interpolating their X-ray scattering factors and then be finished by the Mott formula conversion. Compared to existing popular parameterisations that use Gaussian/Lorentzian-like summations, our method returns identical scattering factors to the

tabulated values, while the continuity of the resulting curves and their derivatives is guaranteed. In addition to the perfect fitting to tabulated scattering factors, our QCBED pattern-matching simulations also confirm the reliability of our new interpolation.

Acknowledgements The acknowledgements should be in a single paragraph (style: IUCr acknowledgements; this style applies the heading).

References

Bird, D. M. & King, Q. A. (1990). *Acta Crystallographica Section A* **46**, 202-208.

Brown, P. J., Fox, A. G., Maslen, E. N., O'Keefe, M. A. & Willis, B. T. M. (2006). *Intensity of diffracted intensities, International Tables for Crystallography*, Vol. C*, Interpretation of diffracted intensities,* 1st online ed.*,* edited by E. Prince, ch. 6.1, pp. 554-595. Chester: International Union of Crystallography. [10.1107/97809553602060000600]

Colliex, C., Cowley, J. M., Dudarev, S. L., Fink, M., Gjønnes, J., Hilderbrandt, R., Howie, A., Lynch, D. F., Peng, L. M., Ren, G., Ross, A. W., Smith, V. H., Spence, J. C. H., Steeds, J. W., Wang, J., Whelan, M. J. & Zvyagin, B. B. (2006). *Electron diffraction, International Tables for Crystallography*, Vol. C*, Production and properties of radiations,* 1st online ed.*,* edited by E. Prince, ch. 4.3, pp. 259-429. Chester: International Union of Crystallography. [10.1107/97809553602060000103]

Cromer, D. T. & Waber, J. T. (1968). Unpublished work.

Doyle, P. A. & Turner, P. S. (1968). *Acta Crystallographica Section A* **24**, 390-397.

Fox, A. G., O'Keefe, M. A. & Tabbernor, M. A. (1989). *Acta Crystallographica Section A* **45**, 786-793.

Kirkland, J. (1998). *Advanced Computing in Electron Microscopy*. Springer US.

Lobato, I. & Van Dyck, D. (2014). *Acta Crystallographica Section A* **70**, 636-649.

Nakashima, P. N. H., Smith, A. E., Etheridge, J. & Muddle, B. C. (2011). *Science* **331**, 1583-1586.

Peng, L.-M., Ren, G., Dudarev, S. L. & Whelan, M. J. (1996). *Acta Crystallographica Section A* **52**, 257-276.

Weickenmeier, A. & Kohl, H. (1991). *Acta Crystallographica Section A* **47**, 590-597.

Appendix A. First level appendix heading (style: IUCr appendix heading 1)

IUCr body text

A1. Second level appendix heading (style: IUCr appendix heading 2)

IUCr body text

A1.1. Third level appendix heading (style: IUCr appendix heading 3)

IUCr body text

Supporting information

Supporting information (such as experimental data, additional figures and multimedia content) that may be of use or interest to some readers but does not form part of the article itself will be made available from the IUCr archives and appropriate databases. If possible, please include supporting material here; otherwise, separate supporting files may be uploaded upon submission of your article.

Supplementary sections, tables and figures should be numbered with a leading 'S'. The styles 'IUCr sup heading 1', 'IUCr sup table caption', 'IUCr sup figure caption', etc. will apply the numbering automatically.

S1. First-level heading [Style: IUCr sup heading 1]

IUCr body text

S1.1. Second-level heading [Style: IUCr sup heading 2]

IUCr body text

S1.1.1. Third-level heading [Style: IUCr sup heading 3]

IUCr body text

Table S1 This is a supplementary table heading [style: IUCr sup table caption; this style applies table numbering]. Please use the **IUCr tables** (toolbar button) to create experimental and geometry tables when reporting crystal structure data.

This is a table headnote (style: IUCr table headnote)

IUCr table text	IUCr table text	IUCr table text	IUCr table text
IUCr table text	IUCr table text	IUCr table text	IUCr table text

This is a table footnote (style: IUCr table footnote)

Figure S1 Supplementary figure [Style: IUCr sup figure caption]

D.4. Paper 4

Please note that this paper is unrelated to the work of this thesis.

research paper

JOURNAL OF APPLIED CRYSTALLOGRAPHY

ISSN 1600-5767

Mechanisms of void shrinkage in aluminium

Zezhong Zhang,[a] Tianyu Liu,[a] Andrew E. Smith,[b] Nikhil V. Medhekar,[a]* Philip N. H. Nakashima[a]* and Laure Bourgeois[a,c]*

[a]Department of Materials Science and Engineering, Monash University, Victoria 3800, Australia, [b]School of Physics and Astronomy, Monash University, Victoria 3800, Australia, and [c]Monash Centre for Electron Microscopy, Monash University, Victoria 3800, Australia. *Correspondence e-mail: nikhil.medhekar@monash.edu, philip.nakashima@monash.edu, laure.bourgeois@monash.edu

Received 8 January 2016
Accepted 30 June 2016

Edited by G. Kostorz, ETH Zurich, Switzerland

Keywords: nanovoids; transmission electron microscopy (TEM); vacancies; diffusion; aluminium.

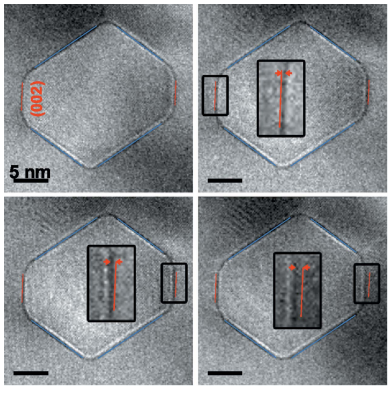
Supporting information: this article has supporting information at journals.iucr.org/j

Voids can significantly affect the performance of materials and a key question is how voids form and evolve. Voids also provide a rare opportunity to study the fundamental interplay between surface crystallography and atomic diffusion at the nanoscale. In the present work, the shrinkage of voids in aluminium from 20 to 1 nm in diameter through *in situ* annealing is imaged in a transmission electron microscope. It is found that voids first shrink anisotropically from a non-equilibrium to an equilibrium shape and then shrink while maintaining their equilibrium shape until they collapse. It is revealed that this process maximizes the reduction in total surface energy per vacancy emitted. It is also observed that shrinkage is quantized, taking place one atomic layer and one void facet at a time. By taking the quantization and electron irradiation into account, the measured void shrinkage rates can be modelled satisfactorily for voids down to 5 nm using bulk diffusion kinetics. Continuous electron irradiation accelerates the shrinkage kinetics significantly; however, it does not affect the energetics, which control void shape.

1. Introduction

A void is a volume of empty space inside matter, and as such it is considered a fundamental defect (Smallman & Bishop, 1999). In solids, voids can form relatively easily under a wide variety of conditions (Kiritani & Yoshida, 1963; Clarebrough *et al.*, 1967; Epperson *et al.*, 1974; Nakajima, 1997; Kassner & Hayes, 2003; Bourgeois *et al.*, 2010; Xu *et al.*, 2013). Voids tend to have a deleterious effect on the mechanical and electrical performance of materials (Nutt & Needleman, 1987; Thomason, 1993; Gungor & Maroudas, 1999; Li *et al.*, 2004; Morgeneyer *et al.*, 2008; Pande, 2012). There has also been a growing interest in using voids as 'negative particles' to design new materials for catalysis (Yin *et al.*, 2004), electronics (Yankovich *et al.*, 2012) and plasmonics (Sigle *et al.*, 2013).

In order to control void formation – whether to suppress or promote it – it is crucial to understand the stability of voids and their evolution under different conditions. Such an understanding requires a detailed structural characterization of voids and how they evolve during growth or shrinkage. Particularly important is the regime corresponding to void diameters at the lower end of the nanoscale, *i.e.* less than 20 nm, as this range is associated with the nucleation and early stages of growth (Smallman & Bishop, 1999). In recent years, there has been much interest in simulating the structure and evolution of voids at this scale. For a seminal set of simulations, see Woo *et al.* (1996), Marian *et al.* (2004, 2005), Uberuaga *et al.* (2007), Sun *et al.* (2009) and Nguyen & Warner (2012). In contrast, experimental studies on the evolution of



nanoscale voids have been surprisingly scarce (Ono & Kino, 2001; Kovács *et al.*, 2011; Xu *et al.*, 2013).

The structure of a void is defined by the surface that encloses it. In crystalline materials, that surface consists of lattice planes. The process of void growth or shrinkage involves the capture or emission of vacancies across those planes (Volin & Balluffi, 1968; Westmacott *et al.*, 1968; Smallman & Bishop, 1999; Xu *et al.*, 2013). Voids formed by quenching a very pure metal will progressively lose vacancies into the matrix and shrink because a void is not thermodynamically stable compared to the matrix with dispersed vacancies at the equilibrium vacancy concentration. Void shrinkage as a function of temperature was studied many years ago for quenched aluminium and found to obey the kinetics of vacancy-mediated bulk diffusion (Volin & Balluffi, 1968; Westmacott *et al.*, 1968), at least for voids with a diameter larger than 10 nm, which was the smallest size that could be measured with confidence in those early investigations. A subsequent study (Ono & Kino, 2001) found evidence that vacancy emission can be affected by the geometry of voids. More recently, the formation of voids through the aggregation of vacancies generated by an electron beam in magnesium was determined to follow specific crystallographic stages (Xu *et al.*, 2013). To date, however, the mechanisms of vacancy emission from a void and their relationship with crystal structure remain unknown. Additionally, the shrinkage kinetics of sub-10 nm voids have not been examined until now.

In the present work, we investigate the shrinkage of voids in ultra-pure aluminium using *in situ* annealing in a transmission electron microscope. Aluminium was chosen as a typical face-centred cubic (f.c.c.) metal that, whether in its pure form or alloyed with other elements, is commonly affected by voids (Kassner & Hayes, 2003; Li *et al.*, 2004). Void-containing aluminium has also shown much promise as a substrate for surface-enhanced Raman scattering (Sigle *et al.*, 2013). Our approach of characterizing the structure and evolution of individual voids by transmission electron microscopy (TEM) differs from that more commonly used where the average size and shape of a large collection of voids is determined by small-angle X-ray scattering (Haubold & Martinsen, 1978; Liu *et al.*, 1978; Fischer *et al.*, 2010) or small-angle neutron scattering (Saegusa *et al.*, 1978). High-resolution TEM imaging *in situ* enables the determination of structural changes of a single void at the sub-nanoscale as a function of time and temperature during shrinkage. In particular, through an analysis of the thermodynamics and kinetics of the process, we show that void shrinkage is more complex than hitherto considered and, significantly, an accurate explanation of the shrinkage process must include a description of the truncated octahedral geometry of the void. We also demonstrate that void shrinkage is quantized *via* single crystallographic facets and that this quantization has a very significant effect on the shrinkage kinetics of sub-10 nm voids. Electron irradiation during the entire shrinkage process was found to greatly accelerate the shrinkage kinetics through a substantial reduction in the diffusion energy barrier. However, the energetics, which control the void shape, do not change.

2. Experimental procedures

The samples were in the form of discs 3 mm in diameter and 1 mm in thickness, punched from a sheet of 99.9999+ at.% pure aluminium. They were heat treated at 823 K for 30 min in a nitrate salt bath and quenched in water at room temperature. This heat treatment led to the formation of voids. The TEM specimens were made by mechanically grinding the discs and electro-polishing them in a 67% methanol–33% nitric acid mixture at 248 K and 13 V. These were examined by TEM within days of being made.

Void shrinkage under thermal annealing was examined *in situ* in a JEOL JEM 2100F field-emission gun transmission electron microscope with 200 and 160 keV electrons. These two values were chosen to explore both sides of the knock-on damage threshold electron energy of 170 keV for bulk aluminium (Hobbs, 1987). The *in situ* annealing was performed using a Gatan 652 double-tilt heating holder at various temperatures. Experiments using 200 keV electrons consisted of isothermal annealing at 95, 293, 323, 373, 403 and 423 K. An additional experiment with 160 keV electrons involved increasing the temperature from 293 to 443 K. In all cases the specimens were continuously subjected to the electron beam with the same current density of $\sim$3 A cm^{-2}, except for two experiments. In these two experiments, performed at 200 keV, in order to identify the effect of electron irradiation on the shrinkage mechanisms, we conducted annealing at 373 and 403 K with the beam off, apart from 10 s for image acquisition every 30 min.

All specimens were examined in bright-field TEM mode, along a $\langle 110 \rangle$ direction. *In situ* annealing of a specimen will tilt the crystal slightly away from its zone axis owing to thermal expansion caused by heating. This meant that, occasionally, we had to adjust the specimen orientation to compensate for this effect. Relatively thick regions (>70 nm) were selected to ensure the voids were sufficiently far from the surface of the sample in order to minimize possible surface oxidation as well as enhanced diffusion due to proximity to a vacancy source/sink. At each temperature the evolution of one void was followed, except at 293 and 373 K, where the evolution of a second void was also characterized. All images were recorded on a Gatan Ultrascan 1000 CCD camera. Individual frames were acquired at high resolution (2048 $\times$ 2048 pixels) and movies were generated by continuous acquisition at lower resolution (512 $\times$ 512 pixels) with exposure times of 0.5–1 s to generate movies through the digital video streaming plug-in of the Gatan *DigitalMicrograph* software (http://www.gatan.com/products/tem-analysis/gatan-microscopy-suite-software) in association with Corel *VideoStudio* (http://www.videostudiopro.com/en/).

3. Results

The shapes and sizes of voids observed before any annealing or prolonged electron irradiation were consistent with previous reports (Kiritani & Yoshida, 1963; Clarebrough *et al.*, 1967; Volin & Balluffi, 1968; Westmacott *et al.*, 1968; Ono &

Kino, 2001), namely truncated octahedra ~20 nm in diameter. Two examples are shown in Fig. 1(*a*). One of the voids is shown at higher magnification in Fig. 1(*b*), revealing its characteristic {002} and {111} facets. No other crystallographic facets are visible. It can clearly be seen that the {111} facets are larger than the {002} facets. We use the distances between parallel facets, $D_{(\bar{1}11)}$, $D_{(1\bar{1}1)}$ and $D_{(002)}$, as marked on the TEM image (see Fig. 1*b*), to characterize the size and aspect ratio of the voids. The aspect ratio, r, is defined as the ratio of the distances between parallel {111} and {002} facets, *i.e.* $r = D_{\{002\}}/D_{\{111\}}$. We found that all voids observed (about 50) exhibited initial aspect ratios ranging from 1.1 to 1.5. These are significantly larger than that corresponding to the equilibrium void shape, $r = 1.06$ (see §3.1.1). Fig. 1(*c*) presents a schematic drawing of a truncated octahedron approximating the shape of the voids observed experimentally: the largest facets are the {111} facets (shown in blue) and the {002} facets (red). We now present the results of our *in situ* annealing experiments.

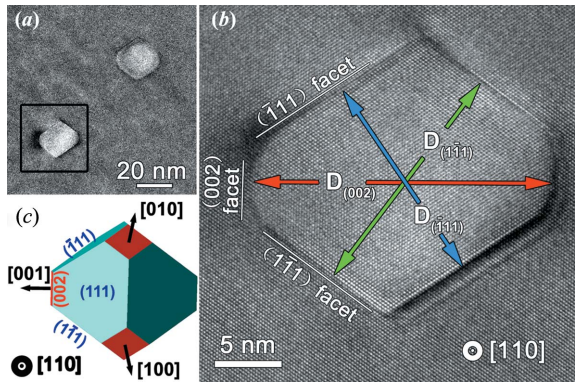


Figure 1
(*a*) TEM image showing two voids. The void framed is shown in (*b*) at high resolution together with the notation used for characterizing the shape and size of voids throughout the present study. (*c*) A truncated octahedron with an aspect ratio typical of the voids observed. The images and void model are all viewed along [110].

3.1. Direct observation of void shrinkage by transmission electron microscopy

3.1.1. A two-stage process. All our annealing experiments (293, 323, 373 and 423 K, as well as the experiment with the electron beam mostly off at 403 K under 200 keV electrons and with ramping temperature under 160 keV electrons) revealed that void shrinkage follows two well defined stages. This is illustrated in Fig. 2. This figure presents TEM snapshots of a void at different stages of its shrinkage when annealed at 373 K. During Stage I, the void shrinks anisotropically, by shrinking in the ⟨001⟩ direction, and undergoes a reduction in aspect ratio to a specific value, $r_0 = 1.06 \pm 0.04$, as shown in Figs. 2(*a*)–2(*c*). During Stage II, the void reduces its size equally in all directions while keeping its aspect ratio constant at $r_0 = 1.06 \pm 0.04$, as shown in Figs. 2(*d*) and 2(*e*), until it is fully dissolved into the matrix. The last moments of a void's life are shown in a movie and still pictures extracted from the movie in the supporting information (see Fig. S1 and Supplementary Movie 1). The void can be clearly observed down to a diameter of 1.2 nm. The same behaviour was observed for the other temperatures considered (see Fig. S2 in the supporting information). In the annealing experiment at 403 K with the beam mostly off, the void also experienced two-stage shrinkage, and the equilibrium shape at Stage II (see Fig. S3 in the supporting information) is consistent with all other cases investigated. The physical meaning of r_0 will be explained in §3.2.1.

The aspect ratio of a void determines the shrinkage mode. This is explained in Fig. 3, which plots the measured distances between the parallel facets as well as the deduced aspect ratios for the same void shown in Fig. 2. It shows $D_{(\bar{1}11)} < D_{(1\bar{1}1)}$ initially, and therefore $D_{(002)}/D_{(\bar{1}11)} > D_{(002)}/D_{(1\bar{1}1)}$. $D_{(\bar{1}11)}$ does not change during Stage I of the shrinkage process (left of the dashed line in Fig. 3). In contrast, $D_{(1\bar{1}1)}$ reduces once $D_{(002)}/D_{(1\bar{1}1)}$ reaches the equilibrium value of 1.06 ± 0.04, which occurs well before the other ratio, $D_{(002)}/D_{(\bar{1}11)}$, equals r_0. Only when all aspect ratios reach this equilibrium value can Stage II be said to have begun.

Throughout their evolution under annealing, voids have their shape dominated by {111} and {002} facets. The {002} facets were found to flatten [compare Fig. 2(*a*) with Figs. 2(*b*)–2(*e*)]. Therefore we used these {111} and {002} facets to characterize the geometry of the voids. Minor facets such as

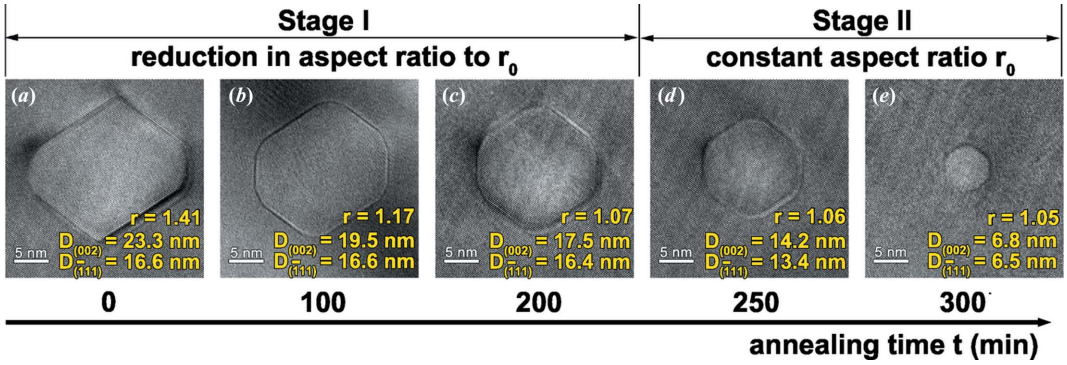


Figure 2
Evolution of a void shrinking at 373 K, as imaged during *in situ* TEM annealing. The shrinkage process follows two stages: first (*a*)–(*c*) a reduction in aspect ratio to a specific value $r_0 = 1.06 \pm 0.04$, and second (*d*)–(*e*) a reduction in size with a constant aspect ratio $r_0 = 1.06 \pm 0.04$. These two stages were observed to take place at all temperatures considered.

{112} and {110} were also observed [this is particularly visible in Fig. 2(*b*)] but were disregarded in the following analysis for the sake of simplicity.

Two additional observations are worth mentioning regarding the overall evolution of voids subjected to annealing. The first one is, in some cases (293 and 373 K), the formation and growth of a dislocation loop a few seconds after the void's disappearance from view (see Supplementary Movie 1). We believe that this is an electron irradiation effect resulting in the clustering of the vacancies left in high concentration shortly after the complete dissolution of the void. The second observation is the rare presence of voids that do not anneal out. This is the case for the upper void displayed in Fig. 1(*a*) and Fig. S4 in the supporting information. This is likely to be the result of TEM sample preparation and the formation of a thick oxide shell at the surface of voids close to the specimen surface. This shows that a void's stability is strongly affected by the chemistry of its surface.

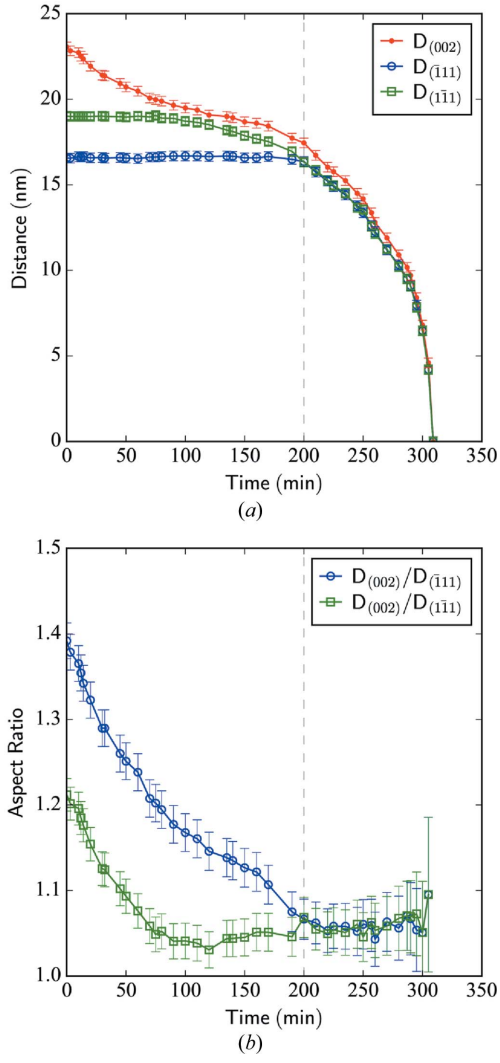


Figure 3
(*a*) Plots of the distance between parallel facets, $D_{(\bar{1}11)}$, $D_{(1\bar{1}1)}$ and $D_{(002)}$, and (*b*) the deduced aspect ratios $D_{(002)}/D_{(\bar{1}11)}$ and $D_{(002)}/D_{(1\bar{1}1)}$, as a function of time for a void shrinking at 373 K (the same void as shown in Fig. 2) The two-stage process is clearly apparent, as indicated by the dashed line separating the two stages.

3.1.2. A monolayer-by-monolayer process. Our *in situ* annealing TEM study uncovered atomic level details regarding the mechanisms of void shrinkage, and in particular how atoms fill the void. We first consider Stage I, or the reduction in the aspect ratio of the void. Fig. 4 shows six snapshots taken from *in situ* annealing TEM movies spanning ∼12 min. During this period, the void can be seen to shrink five times by the same amount (<0.5 nm) through the movement of two of its visible {002} facets, as shown in the magnified insets. The four {111} facets viewed edge on did not move in the present sequence. Owing to the limited spatial resolution of the movie frames whose field of view was used to image an entire void in its early stages, the quantum by which the void shrinks could not be determined accurately in most cases. However, a higher-magnification image of part of the void (see Fig. S5 in the supporting information) clearly indicates that this quantum is 0.2 nm, which is the {002} interplanar spacing. This reduction in void aspect ratio by taking one monolayer at a time was observed to occur systematically during the ∼3 h taken for the void to reach the start of Stage II. The movement of the {002} facets one monolayer at a time was too rapid for its origin to be determined.

We now examine Stage II, where void shrinkage takes place in equal amounts in all directions. Fig. 5 shows a set of three consecutive snapshots taken from an *in situ* annealing TEM movie. An additional set of consecutive snapshots is shown in Fig. S6 in the supporting information. Supplementary Movie 2 is the entire movie sequence from which these snapshots were taken. In the images, the crystal lattice projected down the ⟨110⟩ axis is clearly resolved. In most cases the void surface can also be determined to within one lattice plane, be it a {111} plane or a {002} plane. The movement of the six facets projecting parallel to the viewing direction was tracked, as indicated by the blue and red lines for the {111} and {002} facets, respectively. As for Stage I, void shrinkage during Stage II is found to take place one monolayer at a time: Fig. 5 shows this for a {111} plane. Other monolayer decreases in void size are shown in Fig. S6 in the supporting information. The temporal resolution of our movies was not sufficient to allow determination of the origin of the new atomic planes inside the void. However, there is a clear temporal separation between the appearance of a new atomic plane for different facets. This suggests that facets move one at a time rather than all facets moving in concert.

In order to determine whether there were any qualitative changes in the mechanisms of void shrinkage due to electron irradiation, we conducted one *in situ* experiment with 160 keV electrons. We found that, as for 200 keV electrons, the shrinkage takes place one atomic layer and one facet at a time (see Fig. S7 in the supporting information).

3.2. Analysis of TEM observations

One key result of our TEM observations is the importance of void geometry: the increase in the surface area of the {002} facets during Stage I will inevitably lead to an increase in the surface energy for these facets. This means a substantial

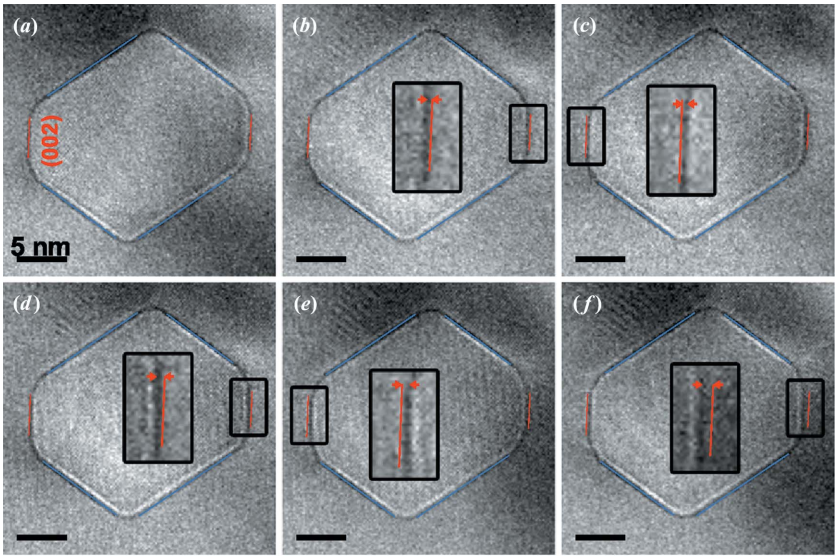


Figure 4
Snapshots from an *in situ* annealing TEM movie at 373 K (the same void as shown in Fig. 2), showing that, during Stage I, shrinkage takes place in a quantized fashion. The blue lines indicate the {111} facets edge on, and the red lines the {002} facets during the first snapshot (*a*). The {002} facets move towards the centre of the void by a distance <0.5 nm in each subsequent frame (*b*)–(*f*).

breakdown of the spherical approximation made in previous studies (Volin & Balluffi, 1968; Westmacott *et al.*, 1968; Ono & Kino, 2001). Therefore, in order to investigate the changes in total energy during void shrinkage across the two observed stages, it is necessary to explicitly take into account the truncated octahedral shape. This is shown in the next section. We show how this treatment provides an explanation for the two-stage process of void evolution and how it can lead to a measure of the vacancy emission rate. The vacancy emission rates deduced from our experiments are then compared with values calculated from bulk diffusion kinetics. Significant differences are observed, which show the strong effect of electron irradiation.

3.2.1. Energetics of the two-stage shrinkage process. We ignore the minor facets, only considering the {002} and {111} facets, and assume that the void is a regular truncated octahedron in order to simplify the analytical derivation of the energetics (see the supporting information for a detailed description of void geometry). An arbitrarily truncated octahedron is also considered in the supporting information and implemented in our simulation program (Zhang & Liu, 2015).

The total energy E of a void consists of a surface energy term and an elastic strain energy term. For voids larger than 1 nm, the elastic strain energy is negligibly small (Westmacott *et al.*, 1968) and thus we only consider the surface energy. The total energy E is therefore

$$E = \gamma_{(002)} S_{\{002\}} + \gamma_{(111)} S_{\{111\}}, \qquad (1)$$

where $\gamma_{(002)}$ and $\gamma_{(111)}$ are the surface energies per unit area for the {002} and {111} surfaces of aluminium, and $S_{\{002\}}$ and $S_{\{111\}}$ are the surface areas of the corresponding facets. We used the surface energies per unit area obtained experimentally at 300 K from the measurements of Allen *et al.* (2003) based on Al–Xe interfacial tensions, $\gamma_{(111)} = 6.6 \pm 0.3$ eV nm^{-2} and $\gamma_{(002)} = 6.9 \pm 0.3$ eV nm^{-2}. The surface energies per unit area are assumed constant throughout the void shrinkage process.

The total energy E as well as the total surface energies $\gamma_{(002)}S_{\{002\}}$ and $\gamma_{(111)}S_{\{111\}}$ for the {002} and {111} facets, respectively, are plotted as a function of time in Fig. 6(*a*) for a void annealed at 373 K. When the distances between the two sets of parallel {111} facets measurable on the images, $D_{(\bar{1}11)}$ and $D_{(1\bar{1}1)}$, differed, $D_{\{111\}}$ was taken as the average of these two distances. One important consequence of the reduction mainly in $D_{\{002\}}$ in the early part of Stage I is that the surface area of these {002} facets and their corresponding surface energies increase with time. However, the total energy of the void actually decreases throughout the shrinkage, as one would expect. This is because the increase in the surface area of the {002} facets is more than compensated by the decrease in the surface area of the {111} facets.

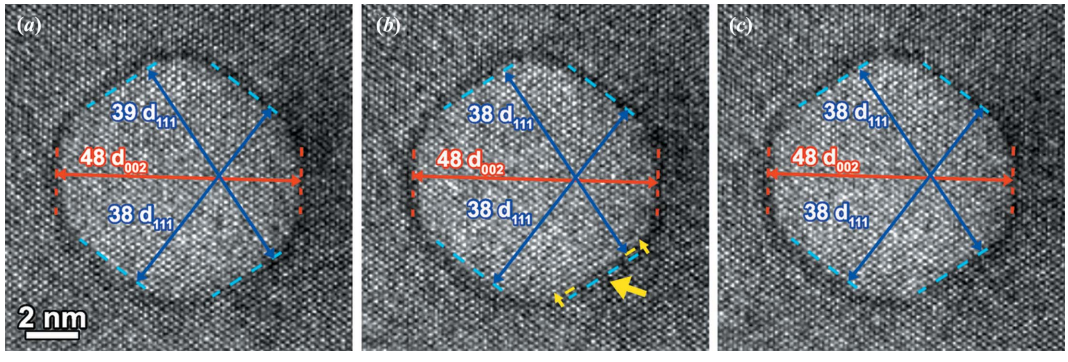


Figure 5
Snapshots from an *in situ* annealing TEM movie at 373 K (the same void as shown in Fig. 2), showing that, during Stage II, void shrinkage takes place one monolayer at a time (see yellow arrows). The three consecutive snapshots span a duration of 3 s and show the movement of a {111} facet into the void. The void dimensions are indicated in terms of the number of {111} and {002} layer spacings, d_{111} and d_{002}, respectively.

In order to estimate the number of vacancies n that make up a void, and therefore determine the vacancy emission rate, we begin with the following equation that always holds exactly:

$$n = V_{\text{void}}/\Omega, \tag{2}$$

where V_{void} is the exact volume of the void and Ω is the volume of a single vacancy, which is equivalent to the atomic volume of 0.0166 nm^3 for aluminium. In practice, we have no way of knowing V_{void}, so we make the following approximation:

$$V_{\text{void}} \simeq V_{\text{reg.t.oct.}}, \tag{3}$$

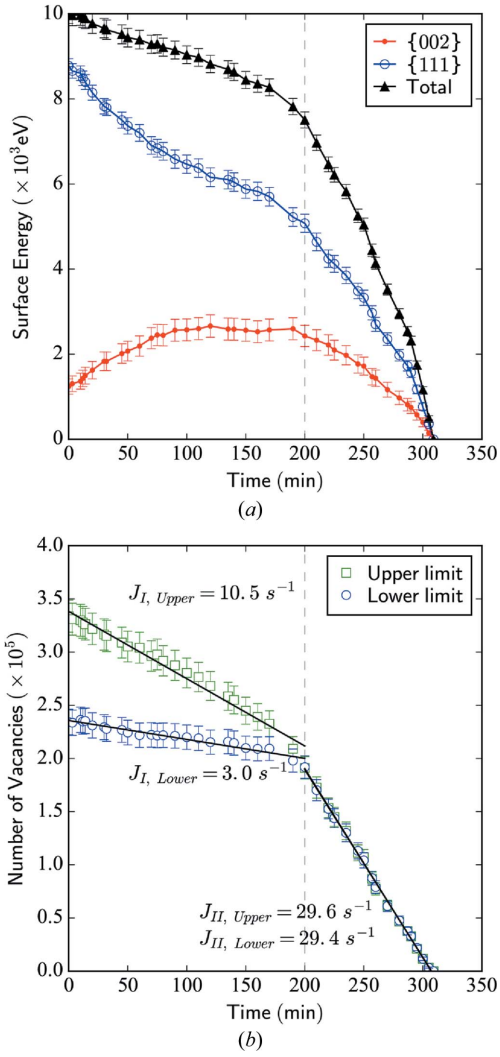


Figure 6
(*a*) Total surface energy and surface energies of {002} facets and {111} facets of a void (the same void as shown in Fig. 2) as a function of annealing time at 373 K. (*b*) Number of vacancies within the void as a function of annealing time at 373 K. The vacancy emission rates J are indicated for each segment of the straight lines with an uncertainty of ±0.5 vacancy per second. The upper and lower limits deduced from the TEM experiments are shown to indicate the uncertainty associated with the extrapolation from the two-dimensional TEM image to a three-dimensional object, as a result of unequal $D_{\{111\}}$ during Stage I. The dashed line represents the separation between Stages I and II.

where $V_{\text{reg.t.oct.}}$ is the volume of the regular truncated octahedron with dimensions that most closely fit those of the void. Equation (2) now becomes

$$n \simeq V_{\text{reg.t.oct.}}/\Omega. \tag{4}$$

This approximation is reasonable for voids of diameter larger than about 10 nm, but degrades for voids below this size. This is because this approximation does not correctly account for the true volume of vacancies at the void surface. In addition, the number of vacancies in the void becomes so small that the irregularities associated with additional lattice planes along particular facets as the void evolves make up a significant perturbation in volume and shape from that of a regular truncated octahedron. In other words, the quantization of void volume with successive lattice planes when voids are small (< 10 nm) cannot be ignored. In order to take this quantization of void volume into account, we use a truncated octahedron that is allowed to vary from a regular shape by single lattice planes in each of the facets and count the lattice sites accurately, including those at the surface. We call this the quantized model, and in the following analyses, we present results from both the continuum approximation [equation (4)] and the quantized model.

The evolution of the number of vacancies inside the void as a function of annealing time at 373 K is shown in Fig. 6(*b*). Two sets of calculated data are shown in order to reflect the uncertainty associated with unequal $D_{\{111\}}$ for Stage I (see Fig. 3). The data align along straight lines, demonstrating that the vacancy emission rate is constant for each of the two stages. The deduced vacancy emission rates J are indicated in each case. The vacancy emission rate is between three and ten times larger during Stage II than during Stage I. This difference can be explained qualitatively by our observations that during Stage I vacancies appear to leave the void only from certain facets, the dominant ones being {002}, whereas during Stage II vacancies leave from all facets.

To explain the two-stage shrinkage process, we now calculate the change in energy per vacancy emitted as a function of the aspect ratio, $(\frac{\partial E}{\partial r}\frac{\partial r}{\partial n})_{D_{\{111\}}}$, and as a function of the distance between {111} facets, $(\frac{\partial E}{\partial D_{\{111\}}}\frac{\partial D_{\{111\}}}{\partial n})_r$, using the continuum approximation for the void volume [equations (4) and (S8) in the supporting information]. The curves for the two energy changes are plotted in Fig. 7 as a function of r.

We can determine which energy change is larger at a given aspect ratio, r, and a given $D_{\{111\}}$ by examining the difference M:

$$\begin{aligned} M &= \left(\frac{\partial E}{\partial r}\frac{\partial r}{\partial n}\right)_{D_{\{111\}}} - \left(\frac{\partial E}{\partial D_{\{111\}}}\frac{\partial D_{\{111\}}}{\partial n}\right)_r \\ &= [\gamma_{(111)} r - \gamma_{(002)}]B, \end{aligned} \tag{5}$$

where the factor B (see the supporting information) is always positive within the observed aspect ratio range of $(1, 3^{1/2})$. Thus,

$$M\begin{cases} > 0 & \text{if } r > \gamma_{(002)}/\gamma_{(111)}, \\ = 0 & \text{if } r = \gamma_{(002)}/\gamma_{(111)}, \\ < 0 & \text{if } r < \gamma_{(002)}/\gamma_{(111)}. \end{cases} \quad (6)$$

It is the aspect ratio that characterizes the shrinkage stage. As shown in Fig. 7, for aspect ratios, r, larger than the equilibrium ratio, $r_0 = \gamma_{(002)}/\gamma_{(111)}$, the void will shrink so as to reduce its aspect ratio to maximize its surface energy reduction per vacancy emitted. Since an increase in $D_{\{111\}}$ will increase the void energy, the void will achieve a reduction in aspect ratio by shrinking mainly along ⟨001⟩. The value of the aspect ratio r_0 is, of course, that corresponding to the Wulff construction (Wulff, 1901), also known as Herring's equilibrium shape (Herring, 1951). Experimentally we found that this equilibrium aspect ratio value fluctuated around 1.06 ± 0.04 during Stage II (see Fig. 3*b*). Measurements at various temperatures give essentially the same value (see Fig. S2 in the supporting information). This is consistent with the measured aspect ratios of 1.05 for Xe-containing bubbles (Allen *et al.*, 2003) and 1.03 for He-containing bubbles (Nelson *et al.*, 1965) in Al after high-temperature annealing to reach equilibrium. On the other hand there is variable agreement with first-principles calculations of Al surfaces at 0 K, which yield values of 1.12 (Jacobs *et al.*, 2002), 1.09 (Kuznetsov *et al.*, 1998) and most recently 1.08 (Gupta *et al.*, 2016).

We built a computer program to simulate an arbitrarily truncated octahedron in an f.c.c. lattice and simultaneously visualize the atomic layer-by-layer shrinkage of a void based on our quantized model (Zhang & Liu, 2015). It assumes that monolayer shrinkage always takes place at the facet that maximizes the total energy reduction per vacancy emitted. The simulation reveals that a void will first shrink anisotropically to its equilibrium shape, as shown above analytically. This holds true for any starting truncated octahedral void shape, including voids with unequal distances between parallel facets such as those experimentally observed. For Stage II, however, the aspect ratio is found to fluctuate instead of maintaining a constant value, in contrast to the prediction of the continuum approximation (Fig. 7). As shown in Fig. 8(*a*), this fluctuation reveals a 'wave' pattern with amplitude and mean value both increasing as the void size decreases. This is because monolayer shrinkage results in increasing departures from the equilibrium aspect ratio as a void becomes smaller, as demonstrated in Fig. 8(*b*). This effect is significant for void diameters below ∼10 nm or ∼40 {111} atomic planes. This also means that such small voids will not necessarily adopt the exact equilibrium shape defined in equation (6). Quite interestingly, a similar behaviour called the 'magic shape effect' is exhibited by Pb nanoprecipitates in Al in order to minimize elastic strain (Hamilton *et al.*, 2007). Hamilton *et al.*'s and our present work show in different ways that the shape and size relationship in nano objects can be understood *via* an energy analysis.

3.2.2. Vacancy emission rates from bulk diffusion kinetics. As shown above, we have demonstrated experimentally that the vacancy emission rate from a void is distinctly different for Stage I and Stage II (see Fig. 6*b*). The phenomenon cannot be

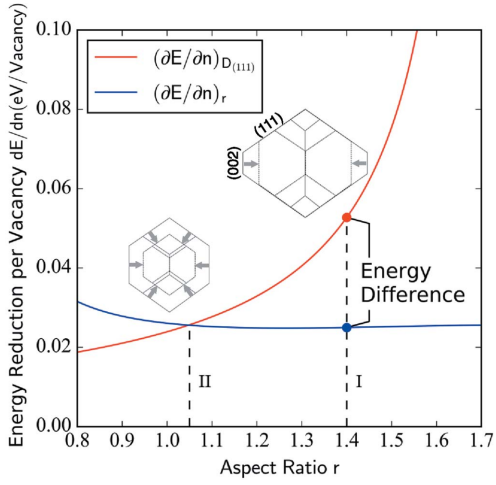


Figure 7
Plots of the change in surface energy per vacancy emitted at 373 K for a void starting with $D_{\{111\}}$ of 17 nm, for the cases of changing aspect ratio (blue) and size (red) according to equation (5). This was calculated using the continuum model. The insets are schematic diagrams illustrating void shrinkage in terms of shape and size to maximize energy reduction per vacancy emitted. (I) $r > \gamma_{(002)}/\gamma_{(111)}$ and the void reduces in aspect ratio with constant $D_{\{111\}}$. (II) $r = \gamma_{(002)}/\gamma_{(111)}$ and the void reduces in $D_{\{111\}}$ whilst maintaining its equilibrium aspect ratio. The grey arrows indicate the change in void shape for Stage I and Stage II.

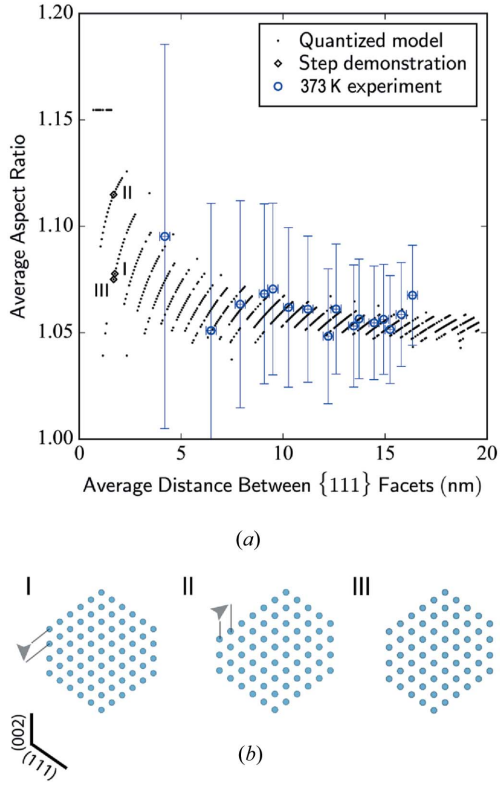


Figure 8
(*a*) Average aspect ratio as a function of average distance between {111} facets in Stage II of void shrinkage: data points are obtained from the simulation of quantized shrinkage as well as the 373 K annealing experiment of Fig. 3. (*b*) Atomic models corresponding to three adjacent quantized states (I, II and III) and viewed along ⟨110⟩. Each arrow indicates which monolayer will be removed in the next state.

explained by the spherical model (Volin & Balluffi, 1968; Westmacott *et al.*, 1968; Crank, 1979). We now use a diffusion equation similar to that developed earlier but apply it to the truncated octahedral void geometry as follows (see derivation in the supporting information). The vacancy emission rate can be written as

$$\frac{\mathrm{d}n}{\mathrm{d}t} = \frac{2SD_\mathrm{s}}{\xi D_{\{hkl\}}\Omega}\left[\exp\left(\frac{\mathrm{d}E/\mathrm{d}n}{k_\mathrm{B}T}\right) - 1\right], \tag{7}$$

where D_s is the self-diffusion coefficient to be measured and ξ is the correlation factor for self-diffusion. For f.c.c. structures, ξ = 0.781 (Volin & Balluffi, 1968). $D_{\{hkl\}}$ represents the distance between major facets $\{hkl\}$ and S is the activated surface area for vacancy emission, which is mainly $S_{\{002\}}$ for Stage I, but both $S_{\{002\}}$ and $S_{\{111\}}$ for Stage II for a regular truncated octahedron. Likewise, the change in energy per vacancy emitted $\mathrm{d}E/\mathrm{d}n$ is different for the two stages (see the supporting information). k_B is the Boltzmann constant and T the temperature.

By fitting equation (7) to the experimental curves of void size *versus* time, we calculate the self-diffusion coefficient for each temperature. An example for 373 K is shown in Fig. 9. The calculated self-diffusion coefficient as a function of temperature is found to display an almost perfect Arrhenius behaviour (see Fig. 10), as should be expected for mono-vacancy mediated diffusion:

$$D_\mathrm{s} = D_0 \exp\left(\frac{-Q}{k_\mathrm{B}T}\right), \tag{8}$$

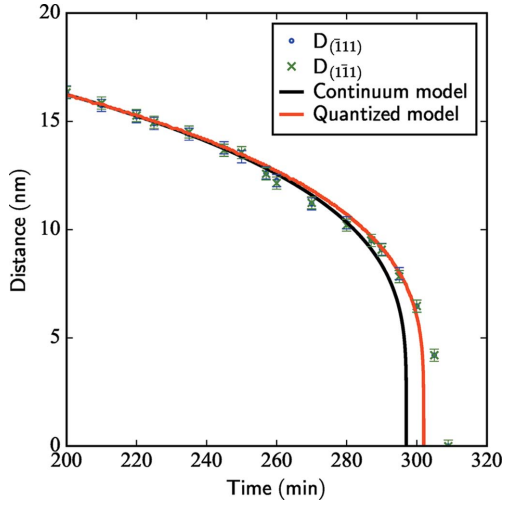


Figure 9
Calculated distance between parallel {111} facets, $D_{\{111\}}$, as a function of time for a void shrinking at 373 K under the electron beam. The calculated curves are for the continuum approximation (black) and a more accurate model taking into account the quantized evolution of monolayer shrinkage and an irregular octahedral geometry (red). The fitting of the continuum model gives D_s = 0.0018 nm^2 s^{-1} and the void is predicted to collapse after 297 min, while the quantized model yields D_s = 0.0020 nm^2 s^{-1} and the void is predicted to collapse after 302 min. The experimental data from Fig. 3(*a*), where the void was observed to vanish after 309 min, are also shown.

where D_0 is the temperature-independent pre-factor and Q is the activation energy for diffusion. Our measured activation energy for diffusion (0.2 eV) is substantially lower than literature values (1.3 eV) (Volin & Balluffi, 1968; Westmacott *et al.*, 1968). The pre-factor D_0 = 2 nm^2 s^{-1} is also much lower, differing from the accepted value (1.76 × 10^{13} nm^2 s^{-1}) by 13 orders of magnitude. This is the well known effect of radiation-enhanced diffusion (Dienes & Damask, 1958; Hobbs, 1987; Zhang *et al.*, 2005). Here it is due to the high-energy electron beam (200 keV) causing knock-on damage in the aluminium crystal and, in particular, generating both vacancies and interstitials, thus enhancing diffusion. The knock-on damage threshold electron energy for aluminium is 170 keV (Hobbs, 1987). The irradiation effect is most significant at low temperatures, where the number of bulk thermal vacancies is much smaller than the number of electron-generated vacancies.

By turning off the electron beam and measuring the change in void diameter during that time, we confirmed that the deduced diffusivity is much closer to the accepted value for bulk diffusion (see Fig. 10). Examination of almost the entire shrinkage process at 403 K with the electron beam mostly off confirmed that bulk diffusion controls the process. In the 373 K experiment with the beam mostly off, a void (diameter ∼10 nm) reduced its size by one atomic layer within 30 min, showing that the layer-by-layer shrinkage is a general mechanism that operates with or without an electron beam. Our experiment at 160 keV (below the knock-on damage threshold for aluminium) and 433 K also yields a value for D_s much closer to the bulk diffusion value of Volin & Balluffi (1968).

With the diffusion coefficients fitted to take electron irradiation into account, we find that the calculated void size

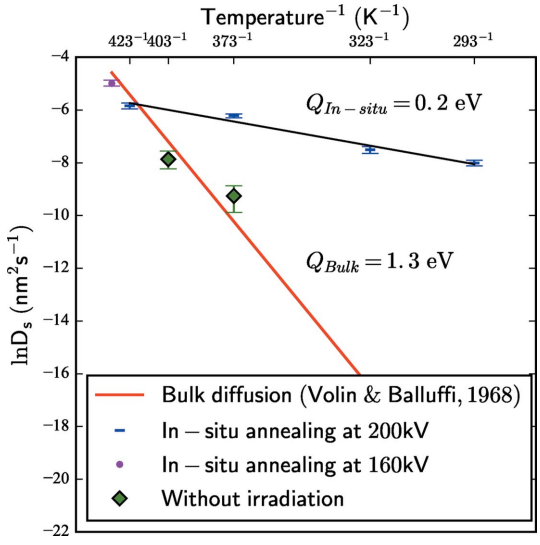


Figure 10
Self-diffusion coefficient of aluminium as a function of the reciprocal of the absolute temperature. The data consist of *in situ* annealing at 293, 323, 373, 423, 433 K under the electron beam and *in situ* annealing at 373 and 403 K with the electron beam mostly off. A comparison is made with the accepted value of bulk diffusion in aluminium (Volin & Balluffi, 1968).

versus time fits the experimental curve quite well, as shown in Fig. 9. Nevertheless, the calculations assuming a continuous change in void size exhibit a notable departure from the experimental data for void diameters less than 10 nm: void shrinkage is observed to be remarkably slower than predicted, by a very significant 12 min, compared with the total duration of 109 min for Stage II. This characteristic was observed in all our *in situ* annealing experiments. This cannot be explained by uncertainties in measuring void dimensions, as voids remain clearly measurable down to a diameter of 2 nm (see Fig. S1 in the supporting information). Two reasons can be invoked to explain this discrepancy: (1) the number of vacancies is underestimated by equation (4); (2) the aspect ratio fluctuates around the equilibrium value and has a shape–size relationship, whereas the continuum approximation assumes a constant aspect ratio. Both factors will lead to a greater energy change per vacancy emitted $\mathrm{d}E/\mathrm{d}n$ for the continuum model [see equation (7)] and thus faster kinetics compared with experimental observations. The quantized model, in contrast, matches experimental observation well down to 5 nm and also reduces the time discrepancy. However, for voids smaller than 5 nm, the calculations based on the quantized model still differ from experiments. We briefly discuss the possible reasons for this discrepancy in §4.

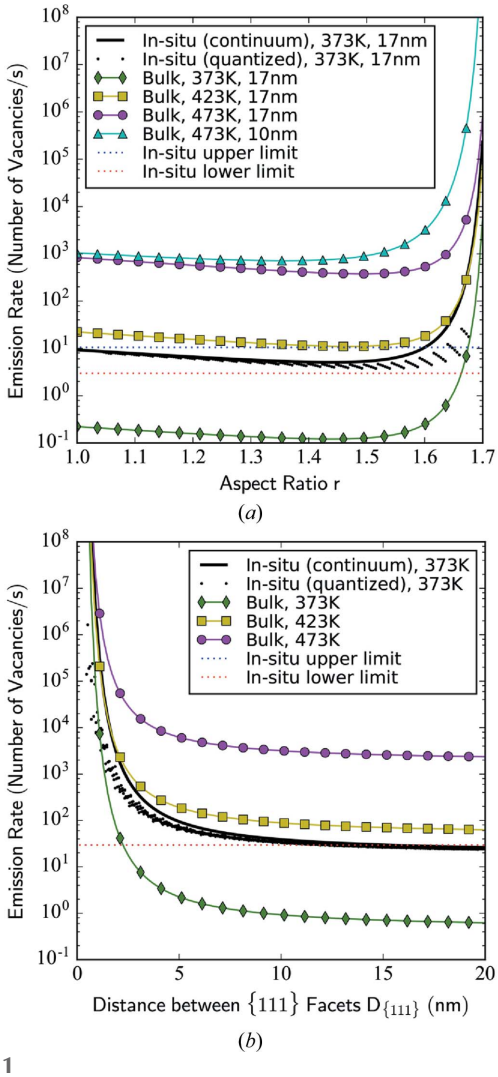


Figure 11
Vacancy emission rates as a function of (*a*) aspect ratio during Stage I and (*b*) distance between {111} facets $D_{\{111\}}$ during Stage II. Plots are displayed for different $D_{\{111\}}$ distances [17 and 10 nm in (*a*)], different temperatures, the continuum model, the quantized model, and the diffusion coefficients corresponding to the *in situ* experimental conditions and the accepted values for bulk aluminium. The upper and lower limits are those deduced from experiment, as shown in Fig. 6(*b*). Note that in the present figure (*b*) these two limits overlap.

The vacancy emission rates calculated in equation (7) provide information about void shrinkage under more general conditions, as shown in Fig. 11. The conditions investigated are higher temperatures, different void sizes and a diffusion coefficient value for bulk diffusion unaffected by electron irradiation. During Stage I, the curves exhibit a minimum at a specific aspect ratio, or tipping point (see Fig. 11*a*). For a typical void diameter of 17 nm, the minimum occurs at an aspect ratio of ∼1.5. Above that value, the vacancy emission rate increases dramatically with increasing aspect ratio. This is because the energy reduction per vacancy emitted is high for $r > 1.5$, whereas below that value, the emission rate decreases only moderately with increasing aspect ratio. This may provide an explanation for our observation that all voids had an aspect ratio of no more than 1.5 before annealing. Our calculated vacancy emission rate also shows that the tipping point is strongly affected by temperature and void size. This may explain other long-standing observations regarding void formation, such as the fact that it is favoured by a relatively slow quenching rate to a moderately high temperature (Martin *et al.*, 1997). It is also known that, under such conditions, voids in copper will tend to display much larger sizes and higher aspect ratios (Clarebrough *et al.*, 1967). These observations are consistent with our calculations: larger void sizes and higher temperatures will lead to a tipping point corresponding to larger aspect ratios.

The calculated vacancy emission rate for Stage II is relatively constant for void diameters above ∼10 nm, but increases exponentially for voids less than ∼7 nm in diameter (see Fig. 11*b*). This differs markedly from the experimentally observed vacancy emission rate, which is constant throughout Stage II, including for voids as small as 1 nm (see Fig. 6*a*).

The quantized model has a noticeably lower vacancy emission rate for voids of high aspect ratios in Stage I or small sizes in Stage II. This difference yields a slower shrinkage rate (see Fig. 9). However, for a sufficiently large void close to its equilibrium aspect ratio, the two models converge.

4. Discussion

Our *in situ* annealing experiments in the transmission electron microscope enabled the shrinkage of voids in pure aluminium to be characterized accurately in terms of size, shape and crystallography, for voids 20–1 nm in diameter. We found that void shrinkage follows two stages. Stage I is distinctly anisotropic, involving the movement of certain facets only. A similar observation was made by Clarebrough *et al.* (1967) for

voids in copper quenched in carbon monoxide: upon annealing voids exclusively bound by {111} facets will develop {110} and {002} facets, thus approaching an equiaxed shape. A more extreme form of anisotropy was observed by Westmacott *et al.* (1968) in so-called 'sausage-shaped voids', elongated voids that had nucleated on impurity inclusions in otherwise pure aluminium. The voids were observed to shrink only in their longitudinal dimension, with little change in their diameter. Calculations based on bulk diffusion indicated that the shrinkage kinetics of such voids were consistent with vacancies only leaving the hemispherical cap of the voids. We have shown that this anisotropic behaviour is simply the void moving to its equilibrium shape in the most efficient way, *i.e.* in the way that maximizes the decrease in energy per vacancy emitted. This is therefore a general phenomenon that holds true for any void that does not exhibit its equilibrium shape, whether an inclusion is associated with it or not.

Our results clearly show that, before being subjected to annealing (apart from the low-temperature annealing during storage before the TEM observations), voids do not exhibit their equilibrium shape. The reason for this is most likely to be found in the growth process of voids, a process which remains poorly understood. However, our study combining a detailed experimental examination and a kinetics model taking into account the truncated octahedral shape suggests an explanation for the surprising uniformity of the aspect ratio of voids prior to annealing. Vacancies are emitted from the {002} facets at high rates until the aspect ratio reaches the commonly observed value of ∼1.5. On the basis of our calculated vacancy emission rates (see Fig. 11*a*), it is reasonable to suggest that this phenomenon takes place during void formation, which incorporates vacancy absorption and emission as competing non-equilibrium processes, rather than in the short time the TEM samples are stored before the examination. Indeed, the emission rate changes significantly with aspect ratio only at high temperatures (>423 K).

In previous studies (Volin & Balluffi, 1968; Westmacott *et al.*, 1968) bulk diffusion was found to fit the overall kinetics of void shrinkage quite well. This is also supported by our experiments, both under the electron beam and with the beam turned off (see Fig. 10). A continuum model of the transport of vacancies away from the void through bulk diffusion could describe the observed void shrinkage kinetics satisfactorily for a void diameter as small as 10 nm, provided the diffusion parameters are modified to take into account the enhancement caused by electron irradiation. For smaller voids (diameter <10 nm), the agreement between calculations and experiments improved when the quantization of void size and shape was taken into account. However, the shrinkage kinetics seems to be further slowed for void diameters smaller than 5 nm. This may be caused by the presence of molecules (*e.g.* H_2) trapped within the void. Such gas molecules may lower the nucleation energy barrier of voids (Shimomura & Yoshida, 1967; Shimomura & Moritaki, 1981). For example, voids formed by quenching in carbon monoxide or hydrogen atmospheres were found in significantly higher number densities compared with voids quenched in vacuum and did not shrink following prolonged annealing at elevated temperatures (Clarebrough *et al.*, 1967). In the present work, however, where water quenching was performed, it is unlikely that the samples will contain noticeable levels of hydrogen, as reported by previous authors (Shimomura & Yoshida, 1967). In addition, the fact that the kinetics are observed to slow down for sub-5 nm-diameter voids only suggests the gas molecules are very few, if any. Therefore even if such gas molecules were present in very small numbers and influenced the kinetics for sub-5 nm voids, it is highly unlikely that they would change the observed surface-driven anisotropy and monolayer shrinkage. The discrepancy may also be the result of other nanoscale effects: (1) a change in the formation energy of vacancies generated at the surface; (2) a change in the surface energy per unit area for a sub-5 nm void. Finally, the effect of electron irradiation cannot be discounted, even though the discrepancy in the kinetics of small voids was observed for all temperatures and voltages.

While electron irradiation does not change the two-stage nature of the shrinkage of voids and the energy associated with the equilibrium shape, the shrinkage kinetics are greatly accelerated. The radiation-enhanced diffusion activation barrier was determined to be 0.2 eV in our *in situ* experiments, a value substantially less than 1.3 eV for bulk diffusion in the absence of radiation. The total activation barrier for diffusion comprises the activation energies for vacancy formation and vacancy migration, each ∼0.65 eV (Mehrer, 2007). It may seem surprising that electron irradiation can enhance vacancy diffusion to such a large extent; however, this can be understood by noting that vacancy formation is initiated at surfaces, including the surface of a void, and that it will be greatly enhanced through the process of electron-beam-induced sputtering (Egerton *et al.*, 2010). Similarly, vacancy migration can be strongly accelerated by electron irradiation (Banhart, 1999). We can estimate the sputtering rate s for our experimental conditions. According to Egerton *et al.* (2010), $s = (J_e/e)\sigma a_0^{1/3}$, where J_e is the current density of the electron beam (∼3 A cm^{-2} for our experiments), e is the charge of the electron, a_0 is the volume per atom and σ is the sputtering cross section. For aluminium and for 200 keV electrons, $\sigma \simeq$ 400 barn (1 barn = 10^{-28} m^2) (Bradley & Zaluzec, 1989). Using an atomic volume of 0.01661 nm^3 for aluminium, we obtain a sputtering rate of 0.002 nm s^{-1}, or ∼3 atoms per second (*i.e.* 3 vacancies per second leaving the void) for each 5 nm-wide {002} facet. For stage I, assuming only the {002} facets emit vacancies and assuming the electron beam sputters atoms at the same rate in all ⟨001⟩ directions, this yields a sputtering rate of 18 vacancies per second. This is probably an overestimation given that the electron beam will also generate interstitials in the bulk (Egerton *et al.*, 2010), which by recombining with vacancies will slow down the effective vacancy emission rate (Dienes & Damask, 1958). But the beam-generated interstitials can be expected to be far fewer than the beam-generated vacancies at the void surface. This can explain why this rough estimate is of the same order of magnitude as the vacancy emission rate measured experimentally at 373 K, where electron-beam-generated vacancies

are expected to dominate thermal vacancies (3–13 vacancies per second; see Fig. 6). Similarly, for Stage II our estimate for the sputtering rate is ∼50 vacancies per second based on the model above, which is also larger but of the same order of magnitude as the measured rate of 29 vacancies per second. The discrepancy between the calculated and the measured vacancy emission rate from a void may also be partly attributed to the additional activation barrier associated with vacancy migration away from a void, although it can be expected to be very low under the electron beam. These simple calculations show that electron beam sputtering can explain the dramatic enhancement in vacancy diffusion observed.

Unfortunately, there is no way of avoiding the effect of irradiation when performing such *in situ* annealing studies of voids in pure Al, as the knock-on sputtering damage threshold for Al is ∼65 keV (Egerton *et al.*, 2010), and the threshold for the acceleration of kinetics *via* vacancy migration is ∼10 keV. These values correspond to the lowest electron beam energy required to sputter or displace an Al atom at a free surface or next to a vacancy (Hobbs, 1987), assuming the displacement energy required is ∼3 eV for an Al atom at a surface and 0.65 eV for an Al atom next to a vacancy. The latter value is the activation energy for vacancy migration. Nevertheless, it is important to note that, although the electron beam accelerates the kinetics of void shrinkage, it does not modify its mechanisms. As mentioned earlier, a void not exposed to an electron beam follows a distinct two-stage shrinkage behaviour as well as monolayer-by-monolayer shrinkage. It also adopts the same equilibrium shape as a void subjected to electron irradiation. Finally, electron irradiation may be mitigated by the use of lower current density beams and more sensitive detectors.

Alloys affected by voids in technological applications will contain many more chemical elements than just aluminium. Solute elements can be expected to interact with the surfaces of voids and hence affect void growth and shrinkage behaviour. The present study should thus constitute a useful reference for studies of voids in more complex systems.

5. Conclusion

We performed *in situ* transmission electron microscopy annealing of voids in pure aluminium and found that void shrinkage is a two-stage process. Voids first shrink anisotropically before reaching their equilibrium shape, then shrink isotropically until full dissolution into the matrix. This process maximizes the decrease in energy per vacancy emitted. The void reduction in size is quantized, taking place one atomic layer and one facet at a time. With an explicit consideration of geometry, modelling of the energetics and kinetics of void shrinkage reproduced the experimentally observed changes in shape and size quite well, for void diameters down to 5 nm, provided the effect of electron irradiation was taken into account. Electron irradiation greatly accelerates the shrinkage kinetics but does not affect energetically controlled phenomena such as void shape.

Acknowledgements

The authors acknowledge the financial support of the Victorian State Government and Monash University for instrumentation, and use of the facilities within the Monash Centre for Electron Microscopy. PNHN thanks the Australian Research Council for his Future Fellowship (FT110100427). ZZ is grateful for a Monash Graduate Scholarship and a Monash International Postgraduate Research Scholarship. TL is grateful for a Monash University Faculty of Engineering International Postgraduate Research Scholarship. LB and NVM acknowledge the financial support of the Australian Research Council (DP150100558). ZZ is indebted to Dr Jicun Li for his help with computer programming.

References

Allen, C. W., Birtcher, R. C., Donnelly, S. E., Song, M., Mitsuishi, K., Furura, K. & Dahmen, U. (2003). *Philos. Mag. Lett.* **83**, 57–64.
Banhart, F. (1999). *Rep. Progr. Phys.* **62**, 1181–1221.
Bourgeois, L., Bougaran, G., Nie, J. F. & Muddle, B. C. (2010). *Philos. Mag. Lett.* **90**, 819–829.
Bradley, C. R. & Zaluzec, N. J. (1989). *Ultramicroscopy*, **28**, 335–338.
Clarebrough, L. M., Humble, P. & Loretto, M. H. (1967). *Acta Metall.* **15**, 1007–1023.
Crank, J. (1979). *The Mathematics of Diffusion*, Oxford Science Publications. Oxford: Clarendon Press.
Dienes, G. J. & Damask, A. C. (1958). *J. Appl. Phys.* **29**, 1713–1721.
Egerton, R. F., McLeod, R., Wang, F. & Malac, M. (2010). *Ultramicroscopy*, **110**, 991–997.
Epperson, J. E., Hendricks, R. W. & Farrell, K. (1974). *Philos. Mag.* **30**, 803–817.
Fischer, S., Diesner, T., Rieger, B. & Marti, O. (2010). *J. Appl. Cryst.* **43**, 603–610.
Gungor, M. R. & Maroudas, D. (1999). *J. Appl. Phys.* **85**, 2233–2246.
Gupta, S. S., van Huis, M. A., Dijkstra, M. & Sluiter, M. H. F. (2016). *Phys. Rev. B*, **93**, 085432.
Hamilton, J. C., Léonard, F., Johnson, E. & Dahmen, U. (2007). *Phys. Rev. Lett.* **98**, 236102.
Haubold, H.-G. & Martinsen, D. (1978). *J. Appl. Cryst.* **11**, 592–596.
Herring, C. (1951). *Phys. Rev.* **82**, 87–93.
Hobbs, L. W. (1987). *Introduction to Analytical Electron Microscopy*, edited by J. Hren, I. Goldstein & D. C. Joy, pp. 437–480. New York: Springer.
Jacobs, P. W. M., Zhukovskii, Y. F., Mastrikov, Y. & Shunin, Y. N. (2002). *Solid State Phys.* **6**, 7–28.
Kassner, M. E. & Hayes, T. A. (2003). *Int. J. Plasticity*, **19**, 1715–1748.
Kiritani, M. & Yoshida, S. (1963). *J. Phys. Soc. Jpn*, **18**, 915.
Kovács, A., Sadowski, J., Kasama, T., Domagała, J., Mathieu, R., Dietl, T. & Dunin-Borkowski, R. E. (2011). *J. Appl. Phys.* **109**, 083546.
Kuznetsov, V. M., Kadyrov, R. I. & Rudenskii, G. E. (1998). *J. Mater. Sci. Technol.* **14**, 320–322.
Li, B., Sullivan, T. D., Lee, T. C. & Badami, D. (2004). *Microelectron. Reliab.* **44**, 365–380.
Liu, S., Moteff, J., Hendricks, R. W. & Lin, J. S. (1978). *J. Appl. Cryst.* **11**, 597–602.
Marian, J., Knap, J. & Ortiz, M. (2004). *Phys. Rev. Lett.* **93**, 165503.
Marian, J., Knap, J. & Ortiz, M. (2005). *Acta Mater.* **53**, 2893–2900.
Martin, J. W., Doherty, R. D. & Cantor, B. (1997). *Stability of Microstructure in Metallic Systems*, Cambridge Solid State Science Series. Cambridge University Press.

Mehrer, H. (2007). *Diffusion in Solids*, Springer Series in Solid-State Sciences. Berlin, Heidelberg: Springer.
Morgeneyer, T. F., Starink, M. J. & Sinclair, I. (2008). *Acta Mater.* **56**, 1671–1679.
Nakajima, H. (1997). *JOM*, **49**, 15–19.
Nelson, R. S., Mazey, D. J. & Barnes, R. S. (1965). *Philos. Mag.* **11**, 91–111.
Nguyen, L. D. & Warner, D. H. (2012). *Phys. Rev. Lett.* **108**, 035501.
Nutt, S. R. & Needleman, A. (1987). *Scr. Metall.* **21**, 705–710.
Ono, K. & Kino, T. (2001). *Philos. Mag. A*, **81**, 2565–2575.
Pande, C. S. (2012). *Fatigue of Materials II: Advances and Emergences in Understanding*, edited by T. S. Srivatsan, M. Ashraf Imam & R. Srinivasan, pp. 1–15. Hoboken: John Wiley and Sons.
Saegusa, T., Weertman, J. R., Cohen, J. B. & Roth, M. (1978). *J. Appl. Cryst.* **11**, 602–604.
Shimomura, Y. & Moritaki, Y. (1981). *Jpn. J. Appl. Phys.* **20**, 1787–1790.
Shimomura, Y. & Yoshida, S. (1967). *J. Phys. Soc. Jpn*, **22**, 319–331.
Sigle, D. O., Perkins, E., Baumberg, J. J. & Mahajan, S. (2013). *J. Phys. Chem. Lett.* **4**, 1449–1452.
Smallman, R. E. & Bishop, R. J. (1999). *Modern Physical Metallurgy and Materials Engineering*, pp. 84–124. Oxford: Butterworth-Heinemann.
Sun, Z., Zhou, J., Blomqvist, A., Johansson, B. & Ahuja, R. (2009). *Phys. Rev. Lett.* **102**, 075504.
Thomason, P. F. (1993). *Acta Metall. Mater.* **41**, 2127–2134.
Uberuaga, B. P., Hoagland, R. G., Voter, A. F. & Valone, S. M. (2007). *Phys. Rev. Lett.* **99**, 135501.
Volin, T. E. & Balluffi, R. W. (1968). *Phys. Status Solidi*, **25**, 163–173.
Westmacott, K. H., Smallman, R. E. & Dobson, P. S. (1968). *Met. Sci.* **2**, 177–181.
Woo, C. H., Singh, B. N. & Semenov, A. A. (1996). *J. Nucl. Mater.* **239**, 7–23.
Wulff, G. (1901). *Z. Kristallogr. Cryst. Mater.* **34**, 449–530.
Xu, W., Zhang, Y., Cheng, G., Jian, W., Millett, P. C., Koch, C. C., Mathaudhu, S. N. & Zhu, Y. (2013). *Nat. Commun.* **4**, 1–6.
Yankovich, A. B., Puchala, B., Wang, F., Seo, J.-H., Morgan, D., Wang, X., Ma, Z., Kvit, A. V. & Voyles, P. M. (2012). *Nano Lett.* **12**, 1311–1316.
Yin, Y., Rioux, R. M., Erdonmez, C. K., Hughes, S., Somorjai, G. A. & Alivisatos, A. P. (2004). *Science*, **304**, 711–714.
Zhang, L. H., Johnson, E. & Dahmen, U. (2005). *Acta Mater.* **53**, 3635–3642.
Zhang, Z. & Liu, T. (2015). *Void Evolution in Pure Aluminum*, http://tianyu-liu.github.io/purealvoid/.