EMsoft
EBSD, ECP and Kossel Simulations

Program Manual, v3.2, September 29, 2017*

Table of Contents

*This set of programs was developed with financial support from two agencies. The Office of Naval Research sponsored the development of the original version 2.0 f90 source code for computation of EBSD patterns in research grant N00014-12-1-0075. The IDL visualization interface, and the complete version 3.0 (including ECP) and beyond were developed with financial support from an AFOSR/MURI grant FA9550-12-1-0458. All modifications in version 3.2 were carried out under a DoD Vannevar Bush Faculty Fellowship grant ONR # N00014-16-1-2821.

1 Introduction

This manual describes a suite of programs for the dynamical simulation of electron backscatter diffraction patterns (EBSD) and electron channeling patterns (ECP): EMMCOpenCL,[1], EMMCfoil, EMEBSDmaster, EMECPmaster, EMTKDmaster, EMEBSD, and EMECP. The output generated by these programs is formatted in the open source HDF5 standard, and can be visualized by the IDL (Interactive Data Language) virtual machine app SEMDisplay.pro.

In this manual, we hope to accomplish the following tasks:

1. Explain briefly the underlying pattern formation theory and the numerical approach followed by the f90 programs (section 5);
2. Document the input files for the f90 programs (sections 6, 7, 9, and 10);
3. Document how to call the EBSD and ECP routines from other programming languages (section 13);
4. Document the IDL interface (section 14);
5. Explain the use of these programs by means of a few basic examples (section 15).

At the time of writing of this manual, these programs have been successfully compiled on the Mac OS X platform and on Linux CentOS using the public domain gfortran compiler. In addition, the code has been built successfully on Windows 10 using the commercial ifort compiler. It would be interesting to see some of this code implemented in a super computer setting, since many of the routines should be quite parallellizable. Where possible, computations are carried out using OpenMP directives, so that multiple cores can be used. The Monte Carlo codes use OpenCL to run on a GPU card, if one is avallable.

The EMsoft package is entirely written in f95 and uses some of the newer features available in the 2003, 2008, and 2013 versions (and therefore the package requires a recent version of gfortran (at least 4.9) to successfully compile). The source code is extensively commented, using regular comment lines, but also using DOxygen documentation generation commands (used only on Mac OS X). Hence, there exists an extensive on-line documentation of all variables, variable types, modules, subroutines, functions, etc. for the latest version of the code. For selected programs, more extensive manual pages are available. If interested, please contact the authors for further information.

The visualization part of the code consists of a series of IDL routines that are available as source code or in the form of a Virtual Machine application. If you have an IDL license, then you can compile and run the IDL source code; alternatively, if you do not have a license, then you can use one of the VM apps to perform the same task. Note that in the VM environment, you will not be able to alter/compile the source code.

[1] Note that all programs in the EMsoft package start with the letters "EM"

2 Version 3.2 features

The version 3.2 release does not have many changes in terms of the capabilities of the EBSD and related modalities series of programs. The most important changes are:

1. The EMsoft package now has an improved ability to compute Transmission Kikuchi Diffraction (TKD) patterns via a new series of Monte Carlo, master pattern, pattern simulation and dictionary indexing programs (the latter is described in a separate manual). Details are described in section 11.

2. Since some of these programs can take quite a while to run, in particular the master pattern simulations, we have added an option to have the program send the user an email message once execution is complete. Alternatively, a Slack notification can be sent. This is described in section 8.

3. The EMEBSD program has a new option to apply a deformation tensor to the crystal lattice before the EBSD pattern is interpolated from the master pattern. In other words, one can now compute approximate EBSD patterns when a small deformation is applied to the crystal lattice. Details are described in section 9.

4. The EMEBSD program has a new option for the user to specify the output format of the patterns. In previous versions, 8-bit grayscale patterns were the only ppossible format; in the present version, bytes, 32-bit floats, and 32-bit integers with various dynamical ranges are introduced.

3 Version 3.1 features

The version 3.1 release once again represents only a subset of the original CTEMsoft 2.0 release; only SEM-related programs are part of release 3.1.

In version 3.1, we have made numerous changes; major changes/additions include:

- EMmkxtal now can be called with the option -W to use Wyckoff positions.
- EMMCOpenCL and EMEBSDmaster now put their output in a single HDF5 file instead of two separate files; this is easier to deal with in terms of transferring data files between computers. This release also provides a file merge program EMmergefiles that can be used to merge Monte Carlo and master files from release 3.0 into the format expected for release 3.1. This is also the case for the EMMCOpenCL and EMECPmaster programs used for ECP master pattern simulations.
- EMEBSD and EMECP remain mostly unchanged, with minor modifications; in EMEBSD a correction factor was introduced to compensate for the fact that the solid angle observed by individual detector pixels depends on the location of each pixel. As a result, the background intensity distribution of EBSD patterns is now more realistic and in better agreement with experimental patterns. The EMEBSD program now also allows for the user to define a barrel/pin-cushion distortion coefficient, which can be used to simulate the optical distortions that are often observed in EBSD patterns acquired on cameras with an optical transfer (lens) from the scintillator to the CCD camera.
- A new pair of programs has been introduced for the simulation of TKD patterns, EMMCfoil and EMTKDmaster. These are still in the experimental stages, but are provided in this release so that users can try them out.
- EMEBSDDI is a new program that implements the dictionary indexing approach; similar programs are made available for ECP and PED dictionary indexing. There are several auxiliary programs that can be used to analyze the data file produced by the dictionary indexing program. There is also an IDL GUI interface, Efit.pro, that can be used to fit the EBSD detector parameters for dictionary-based indexing. An auxiliary program, EMDPfit is also made available for parameter fitting. These latter programs are still undergoing further development, so the user should consider them to be preliminary versions; there is a separate manual that describes their program options.
- Several of the EMsoft programs can now be executed as filters in the DREAM.3D program as part of the EMsoftToolbox. There is a separate manual describing these filters.

4 Version 3.0 features

The version 3.0 release represents only a subset of the original CTEMsoft 2.0 release; only SEM-related programs are part of release 3.0. The other TEM related programs will be added in again with future 3.x releases.

In version 3.0, we have made the following changes:

- EMmkxtal now employs the HDF5 output format for the .xtal files.
- EMMCOpenCL now uses an OpenCL kernel routine to perform the Monte Carlo simulation on a GPU, if one is available. This results in significant acceleration compared to the single CPU or multi-core CPU routines from version 2.0. Speed up factors of between 200× and 300× have been measured with respect to the single CPU code. This program is capable of computing the Monte Carlo histograms needed by both the EBSD and ECP master programs.
- EMEBSDmaster has been rewritten using OpenMP directives and is hence significantly faster than before; despite this rewrite, this program remains the main computational bottleneck. The program now uses both Northern and Southern hemispheres for the representation of the master patterns; this is a change with respect to the older version, but it allows the programs to be valid for all crystallographic symmetries (with the exception of the rhombohedral setting of the trigonal system, which remains to be implemented).
- EMECPmaster is a new program that functions along the same liens as the EMEBSDmaster program; it uses the output from the Monte Carlo program to generate a dynamical simulation of the master ECP pattern.
- EMEBSD, remains mostly unchanged, with minor modifications.
- EMECP, is a new program to compute individual ECPs.
- HDF5 support: all these programs now use HDF5 as their output format.

In terms of input files, there are now two input formats accepted by the above programs; one is the traditional fortran namelist format, with .nml files containing name-value pairs; the other is the json format (JavaScript Object Notation), which is essentially an xml-ized version of the name-value pairs. Most of the programs above will take either format as input, but the template option -t to the program will only generate the namelist version for now; this will be updated to include the .json format in a later release.

5 Electron back-scatter diffraction patterns and electron channeling patterns: the underlying theory

As described in detail in a recent paper,[2] there are three steps in the computation of a realistic (dynamical) EBSP/ECP:

- Monte Carlo simulation of the energy, depth, and directional distributions of back-scattered electrons;
- Dynamical simulation of the EBSD/ECP master pattern (covering all possible directions);
- Simulation of an EBSP/ECP for a given detector geometry and sample (grain) orientation.

The main reason for including ECP and EBSD in the same manual is the fact that an ECP can be regarded, via the reciprocity theorem, as a zero-loss EBSD pattern. TKD patterns appear very similar to EBSD patterns, so we include their simulation in this manual; in the Release 3.1 version of the manual, only selected information is included on TKD Monte Carlo and master pattern simulations.

Before we describe the details of the above three steps, we need to introduce a mapping and interpolation technique that is used extensively in all these programs as well as in the visualization program: *modified Lambert projections*. We also need to discuss the symmetry of the scattering problem in terms of the Laue symmetry groups.

5.1 Modified Lambert Projections

The BSEs that leave a sample that is illuminated by a high energy electron beam typically travel in all possible directions in the hemisphere on the vacuum side of the sample. If we wish to compute and accurately represent this spatial distribution, we must define a sampling grid on the sphere surface for which all bins have very nearly the same shape and area, i.e., a uniform distribution of sampling points on the hemisphere surface. This is not an easy problem, and there is quite a bit of literature on spherical sampling grids. In addition to selecting an efficient sampling scheme, we must also make sure that the scheme is compatible with crystallographic symmetries; if a crystal has hexagonal symmetry, then we should only compute quantities for a 60° wedge of orientations, and copy them into the other symmetrically equivalent wedges. Such a copy action will require interpolations if the numerical grid is based on a square sampling, since it is not possible to accurately represent six-fold symmetry on a square grid.

To accommodate all possible crystal symmetries, we have opted for a mapping scheme between a square grid and a hemisphere derived by D. Roşca.[3] To also accommodate hexagonal symmetry, we have derived a new mapping scheme that was published recently.[4] In this section we briefly discuss both mappings, as well as their implementation, taking into account crystal symmetry.

The main goal of the modified Lambert mappings is to create a bi-directional mapping that preserves the area between either a square grid or a hexagonal grid in the plane, and uniform grids on the surface of a sphere. We start from a square grid with semi edge length $L = \sqrt{\pi/2}$ and grid

[2] P.G. Callahan and M. De Graef, *Dynamical Electron Backscatter Diffraction Patterns. Part I: Pattern Simulations* Microsc. Microanal. **19**, 1255–1265, 2013.

[3] Daniela Roşca, *New uniform grids on the sphere*, Astronomy & Astrophysics, **520**, id. A63 (DOI: 10.1051/0004-6361/201015278).

[4] D. Roşca and M. De Graef, *Area-preserving projections from hexagonal and triangular domains to the sphere and applications to electron back-scatter diffraction pattern simulations*, Modeling and Simulations in Materials Science and Engineering, **21**, 055021 (2013).

coordinates (a,b). This point is mapped onto a point (x,y,z) in the Northern hemisphere (i.e., with $z \geq 0$) of a sphere with unit radius by the following relations:

$$(x,y,z) = \begin{cases} \left(\frac{2a}{\pi}\sqrt{\pi - a^2}\cos\frac{b\pi}{4a}, \frac{2a}{\pi}\sqrt{\pi - a^2}\sin\frac{b\pi}{4a}, 1 - \frac{2a^2}{\pi} \right) & 0 < |b| \leq |a| \leq L; \\ \left(\frac{2b}{\pi}\sqrt{\pi - b^2}\sin\frac{a\pi}{4b}, \frac{2b}{\pi}\sqrt{\pi - b^2}\cos\frac{a\pi}{4b}, 1 - \frac{2b^2}{\pi} \right) & 0 < |a| \leq |b| \leq L. \end{cases} \tag{1}$$

The inverse relation is given by:

$$(a,b) = \begin{cases} \mathrm{sign}(x)\sqrt{2(1-z)}\left(\frac{\sqrt{\pi}}{2}, \frac{2}{\sqrt{\pi}}\arctan\frac{y}{x} \right) & 0 \leq |y| \leq |x|; \\ \mathrm{sign}(y)\sqrt{2(1-z)}\left(\frac{2}{\sqrt{\pi}}\arctan\frac{x}{y}, \frac{\sqrt{\pi}}{2} \right) & 0 < |x| \leq |y|. \end{cases} \tag{2}$$

These mappings are readily implemented and provide a bi-directional transformation between a 2-D square grid and a uniform grid on the surface of a sphere. The triplets (x,y,z) can be regarded as direction cosines (since $x^2+y^2+z^2=1$), so that we can represent a uniform sampling of beam directions by a sampling on a square grid. An additional advantage of the uniform character of the mapping is that we can replace interpolation on the surface of a sphere by simple bi-linear interpolation on a square grid, which is easily implemented numerically. For numerical purposes, we will subdivide the square of edge length $2L = \sqrt{2\pi}$ into $(2N+1)\times(2N+1)$ grid points with a step size $\Delta a = \Delta b = \sqrt{\pi/2}/N$; note that the point $(a,b) = (0,0)$ maps onto the North pole $(0,0,1)$. The origin of the square grid lies at the center of the square; an arbitrary grid point then has coordinates $(i\Delta a, j\Delta b)$, with $-N \leq i,j \leq +N$ $(i,j \in \mathbb{Z})$. Points along the outer perimeter of the square are mapped onto the equatorial circle of the Northern hemisphere.

For the hexagonal grid, we consider the sextants I_k, $k = 0,\ldots,5$, which are defined as follows (see Figure 1):

$$I_0 = \left\{ (x,y) \in \mathbb{R}^2, 0 \leq x,\ -x/\sqrt{3} \leq y \leq x/\sqrt{3} \right\}, \tag{3}$$

$$I_1 = \left\{ (x,y) \in \mathbb{R}^2, 0 \leq x,\ x/\sqrt{3} \leq y \right\}, \tag{4}$$

$$I_2 = \left\{ (x,y) \in \mathbb{R}^2, x \leq 0, -x/\sqrt{3} \leq y \right\}, \tag{5}$$

$$I_3 = \left\{ (x,y) \in \mathbb{R}^2, x \leq 0,\ x/\sqrt{3} \leq y \leq -x\sqrt{3} \right\}, \tag{6}$$

$$I_4 = \left\{ (x,y) \in \mathbb{R}^2, x \leq 0,\ y \leq x/\sqrt{3} \right\}, \tag{7}$$

$$I_5 = \left\{ (x,y) \in \mathbb{R}^2, 0 \leq x,\ y \leq -x/\sqrt{3} \right\}. \tag{8}$$

For $(x,y) \in H_a \cap I_k$ (where H_a is the hexagon) one has $x \leq \alpha$ if $k = 0,1,5$ and $x \geq -\alpha$ if $k = 2,3,4$, where $\alpha = a\sqrt{3}/2$.

The point $(x,y) \neq (0,0)$ in the circle is mapped into $(X,Y)_H$ in the hexagon with coordinates given by

- For $(x,y) \in I_0 \cup I_3$,

$$(X,Y)_H = 3^{\frac{1}{4}}\sqrt{\frac{2}{\pi}}\,x\left(\cos\frac{y\pi}{2\sqrt{3}x}, \sin\frac{y\pi}{2\sqrt{3}x} \right); \tag{9}$$

- For $(x,y) \in I_1 \cup I_4$,

$$(X,Y)_H = \frac{3^{\frac{1}{4}}}{\sqrt{2\pi}}(x+\sqrt{3}y)\left(\sin\frac{2\pi x}{3(x+\sqrt{3}y)},\ \cos\frac{2\pi x}{3(x+\sqrt{3}y)} \right); \tag{10}$$

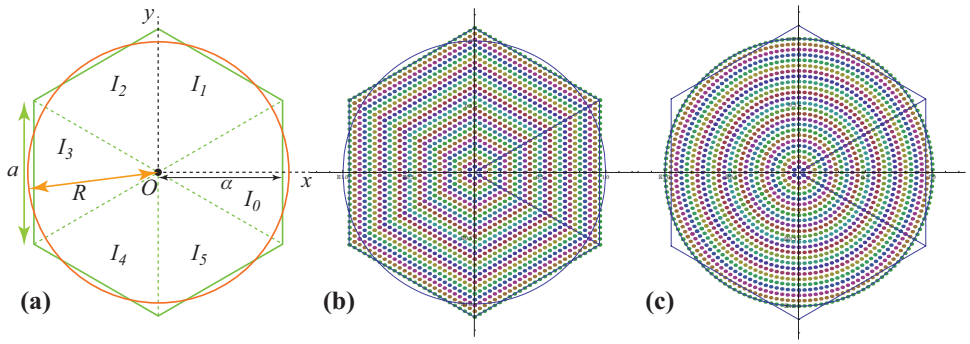

Figure 1: *(a) Subdivision of the hexagon into sextants; the circle has the same area as the hexagon; (b) shows a uniform grid in the hexagon, and (c) shows the same grid after equal-area mapping onto the circle.*

- For $(x, y) \in I_2 \cup I_5$,

$$(X, Y)_H = \frac{3^{\frac{1}{4}}}{\sqrt{2\pi}}(x - \sqrt{3}y)\left(\sin\frac{2\pi x}{3(x-\sqrt{3}y)}, -\cos\frac{2\pi x}{3(x-\sqrt{3}y)}\right). \tag{11}$$

For the origin we define $\mathcal{T}_H(0, 0) = (0, 0)$. The inverse transformation is given by:

- For $(X, Y) \in I_0 \cup I_3$

$$(x, y) = 3^{-\frac{1}{4}}\sqrt{X^2 + Y^2}\,\mathrm{sign}(X)\left(\sqrt{\frac{\pi}{2}}, \sqrt{\frac{6}{\pi}}\arctan\frac{Y}{X}\right); \tag{12}$$

- For $(X, Y) \in I_1 \cup I_4$,

$$(x, y) = \frac{3^{\frac{1}{4}}\sqrt{X^2 + Y^2}}{\sqrt{2\pi}}\,\mathrm{sign}(X)\left(\sqrt{3}\Big(\frac{\pi}{6} - \arctan\frac{Y - \sqrt{3}X}{X + \sqrt{3}Y}\Big), \frac{\pi}{2} + \arctan\frac{Y - \sqrt{3}X}{X + \sqrt{3}Y}\right); \tag{13}$$

- For $(X, Y) \in I_2 \cup I_5$,

$$(x, y) = \frac{3^{\frac{1}{4}}\sqrt{X^2 + Y^2}}{\sqrt{2\pi}}\,\mathrm{sign}(X)\left(\sqrt{3}\Big(\frac{\pi}{6} + \arctan\frac{Y + \sqrt{3}X}{X - \sqrt{3}Y}\Big), -\frac{\pi}{2} + \arctan\frac{Y + \sqrt{3}X}{X - \sqrt{3}Y}\right). \tag{14}$$

The coordinates in the circle can then be transported onto the Northern unit hemisphere by the following inverse Lambert projection:

$$(\widetilde{x}, \widetilde{y}, \widetilde{z}) = \left(\frac{x}{2}\sqrt{4 - (x^2 + y^2)}, \frac{y}{2}\sqrt{4 - (x^2 + y^2)}, 1 - \frac{1}{2}(x^2 + y^2)\right). \tag{15}$$

This relation assumes that the radius of the circle is defined as $R = \sqrt{2}$. The Lambert projection from the hemisphere to the circle is then given by:

$$(x, y) = \sqrt{\frac{2}{1 + \widetilde{z}}}\,(\widetilde{x}, \widetilde{y}); \tag{16}$$

For all these relations to work together, we must also have $a = 2\sqrt{\pi}3^{-3/4}$ for the hexagon edge length, and therefore $\alpha = 3^{1/4}\sqrt{\pi}$ (see Fig. 1). This completes the discussion of the equal area projections used in the EBSD software.

5.2 Master Pattern Symmetry

In the previous version of the EBSD master program, we had implemented the 11 Laue groups as the possible symmetries for the master pattern. Implicitly, however, this meant that we were applying Friedel's Law, which states that all kinematical diffraction patterns are centro-symmetric, hence the use of the Laue groups. For the dynamical EBSD pattern simulations, however, it turns out that this is incorrect. So, in the present version of the program, we have corrected this problem, and now the full point group symmetry is properly taken into account; as a result, the code has become quite a bit more complicated.

There are 32 crystallographic point group symmetries, and each of them requires a dedicated k-space sampling scheme, using the Lambert projections described above. For the crystal systems that map onto the square Lambert projection, things are the same as before, except that we now need to use two Lambert maps, one for the Northern hemisphere, one for the Southern; in the absence of inversion symmetry, both are needed. For the trigonal and hexagonal cases the situation has become quite complicated due to the fact that the trigonal groups can be considered both with a hexagonal and a rombohedral setting. In addition, for the trigonal/rhombohedral case, the three-fold rotation axis does not coincide with the reciprocal $\mathbf{c}^*$ axis, but instead is aligned with the trigonal [111] direction, which causes a bit of a problem. In addition, for both the tetragonal and hexagonal crystal systems, there is one point group that has two settings ($\bar{4}2m/\bar{4}m2$ and $\bar{6}m2/\bar{6}2m$), and this must be taken into account when deciding what the asymmetric unit is for k-space sampling.

Fig. 2 shows the 42 cases for all point group settings; for each case, two circles are shown, representing the stereographic projections of the Northern and Southern hemispheres, labeled NH and SH; the green number between the two circles is the point group number. The black number on the left indicates the case number, whereas the red number to the right of SH indicates the unique case number; several cases end up having the same sampling scheme. So, in total, there are 19 unique sampling schemes; these are implemented in the kvectors module. To determine the proper sampling scheme, the EMEBSDmaster program uses both the space group number and the point group symmetry; this is all done automatically behind the scenes. In cases where the SH projection is blank, this means that all points in this projection are generated from the corresponding NH section by symmetry operations.

Note that for the cubic symmetry groups, we use lower symmetry cases to make sure that the complete square Lambert array is properly filled without interpolations. This means that cubic symmetry takes longer to compute than it should, and we are actively researching alternative approaches that would permit the highest possible resolution Lambert projections for the shortest possible execution time.

For the (obverse) rhombohedral setting of the trigonal space groups, the implementation of the Lambert projections has not yet been worked out; this should not be a problem, since those space groups can always be treated in the hexagonal setting. A full symmetry implementation for the obverse rhombohedral case will become available in one of the next releases of EMsoft. The main issue preventing a simple implementation is the fact that the center axis of the Lambert projection is always taken along $\mathbf{c}^*$, whereas the highest symmetry rotation axis in the rhombohedral case lies along the $[111]_r$ direction, which obviously does not coincide with $\mathbf{c}^*$. This will require special treatment, most likely via the coordinate transformation relating the rhombohedral and hexagonal settings (not difficult, but sufficiently different from the treatment of the other point groups that some special code will be required).

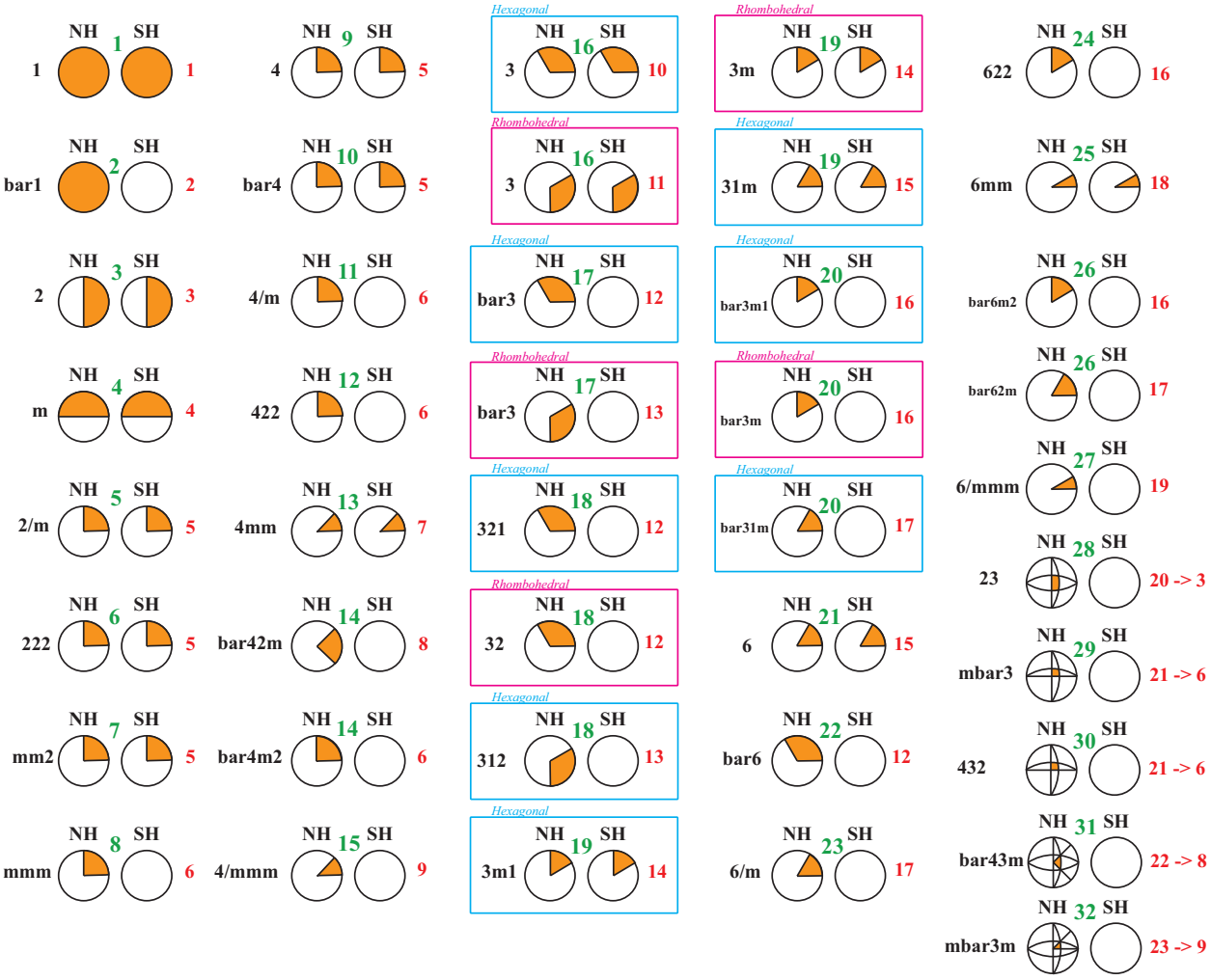

Figure 2: *Asymmetric units for the 32 point groups, drawn on Northern and Southern stereographic projections.*

5.3 Hexagonal Lambert patterns

For the hexagonal and trigonal Laue symmetries, the EMEBSDmaster and ECPmaster programs sample the electron beam directions on a hexagonal grid, so that the symmetry elements can be implemented properly. Subsequently, this grid is transformed to a standard square grid for further processing in the EMEBSD/EMECP programs. This transformation is achieved by going through the square grid, for each point determining the unit vector on the projection hemisphere, and then projecting down into the hexagonal grid using bilinear interpolation. The output of the EMEBSDmaster/EMECPmaster programs is all encoded on the square Lambert grid; the hexagonal grid is only used inside this program and is not visible to the user.

5.4 Monte Carlo Simulations

The current implementation of the Monte Carlo code makes use of the Continuous Slowing Down Approximation (CSDA), which assumes that the incident electron loses energy at a constant rate as it travels inside the sample. This is obviously a rather coarse approximation, since core losses, outer shell excitations, plasmons, phonons, etc are not taken into account explicitly, but replaced by a constant energy loss rate. From a modeling point of view, on the other hand, it is relatively easy to implement, so we have selected this as a first approximate approach over a more involved Monte Carlo model that has been under development as well.

In the Monte Carlo approach, we consider each beam electron separately and follow it on a stochastic trajectory through the sample. The electron travels a random distance scaled by the mean free path length, and then undergoes a scattering event which changes the direction cosines of the trajectory; this process is repeated until either the electron has lost more than a preset amount of energy, or the electron leaves the sample, at which point the direction cosines, energy, and depth of the last scattering event are noted. From this information, the program then generates two different data sets:

- the spatial distribution of electrons, represented on a modified Lambert projection, for each of a series of energy bins;
- and the depth distribution vs. energy and orientation.

This data is then used by the EMEBSDmaster simulation program to determine the depth integration limits and the probability that an electron with a given energy was backscattered at a given depth (note that the depth is the effective distance traveled inside the sample after the last scattering event). For more details on the Monte Carlo approach we refer the interested reader to the literature.[5]

Since version 3.0, the core of this Monte Carlo simulation is performed on a GPU platform, if one is available, using an OpenCL kernel. The kernel source code must be available in the proper folder, EMsoftpathname/opencl.

Note also that this program can be used to perform simulations both for the EBSD case and for the ECP case; this will be discussed in more detail in section 6. For the TKD geometry, a separate program, EMMCfoil, is available for the Monte Carlo simulations. Most of the input parameters for this program are similar to the parameters used for EBSD/ECP MC simulations and will be discussed in a later section.

5.5 Master EBSD and ECP Pattern Simulations

For the master pattern we employ the Bloch wave approach combined with a depth integration. The probability of back-scattered electrons originating from the sample along a specific direction must be integrated over the depth along that direction, which involves an integral over depth. In our approach, this integral is replaced by a summation over the depth bins generated in the EMMCOpenCL program.

The probability of scattering from each subset $\mathcal{S}$ of atomic sites within the unit cell for a given incident beam direction $\mathbf{k}_0$ and an exit energy E can be written as:

$$\mathcal{P}(\mathbf{k}_0, E) = \sum_{\mathbf{g}} \sum_{\mathbf{h}} S_{\mathbf{gh}} L_{\mathbf{gh}}(E), \tag{17}$$

[5] Heinrich, K.H.J., Newbury, D.E. & Yakowitz, H. (Eds.) (1975). *Use of Monte Carlo Calculations in Electron Probe Microanalysis and Scanning Electron Microscopy* Gaithersburg, MD: National Bureau of Standards; Joy, D.C. (1995). *Monte Carlo Modeling for Electron Microscopy and Microanalysis* New York: Oxford University Press.

with

$$S_{\mathbf{gh}} \equiv \sum_n \sum_{i \in \mathcal{S}_n} Z_n^2 \, e^{-M_{\mathbf{h}-\mathbf{g}}^{(n)}} \, e^{2\pi \mathrm{i}(\mathbf{h}-\mathbf{g})\cdot\mathbf{r}_i}; \tag{18a}$$

$$L_{\mathbf{gh}}(E) \equiv \sum_j \sum_k C_{\mathbf{g}}^{(j)*} \alpha^{(j)*} \mathcal{I}_{jk}(E) \alpha^{(k)} C_{\mathbf{h}}^{(k)}. \tag{18b}$$

The first summation in (18a) runs over all the positions in the asymmetric unit of the unit cell; the second sum runs over all equivalent positions in each subset $\mathcal{S}_n$. The parameters $\alpha^{(j)}$ in (18b) are the Bloch wave excitation amplitudes, $C_{\mathbf{g}}^{(j)}$ are the Bloch wave coefficients, and the matrix $\mathcal{I}_{jk}$ is defined by the integral

$$\mathcal{I}_{jk}(E) \equiv \frac{1}{z(E)} \int_0^{z(E)} \lambda(E,z) e^{-2\pi(\alpha_{jk}+\mathrm{i}\beta_{jk})z} \, \mathrm{d}z, \tag{19}$$

where $z(E)$ and $\lambda(E,z)$ are, respectively, the integration depth and the depth- and energy-dependent scattering probability derived from the Monte Carlo program output, and

$$\alpha_{jk} = q^{(j)} + q^{(k)}; \tag{20a}$$

$$\beta_{jk} = \gamma^{(j)} - \gamma^{(k)}. \tag{20b}$$

The complex numbers $\lambda^{(j)} \equiv \gamma^{(j)} + \mathrm{i}q^{(j)}$ are the eigenvalues of the dynamical scattering matrix in the presence of absorption. The integration is performed numerically by simply summing the result for a discrete series of depths. The resulting data is stored as a modified Lambert projection grid (see previous sections).

The only difference between the EBSD and ECP cases is the fact that for the ECPs there is only one energy to consider, whereas the EBSD patterns require integration over an energy range; this means that the $z(E)$ and $\lambda(E,z)$ functions are different for EBSD and ECP. Otherwise, the two simulations approaches are nearly identical.

[In a later release, we will replace the Bloch wave approach by an equivalent scattering matrix approach, so that we can use the GPU platform to compute the necessary matrix exponentials. The Bloch wave approach requires an OpenCL version of the Lapack routines which we have not been able to get to work thus far. The derivation proceeds as follows.

Application of the scattering matrix approach to the EBSD modality requires a different "Ansatz" for the wave function $\Psi(\mathbf{r})$; we write:

$$\Psi(\mathbf{r}) = \sum_{\mathbf{g}} \psi_{\mathbf{g}}(\mathbf{r}) \mathrm{e}^{2\pi\mathrm{i}(\mathbf{k}_0+\mathbf{g})\cdot\mathbf{r}}; \tag{21}$$

this expression implicitly assumes that the scattered electron can only travel along the directions predicted by the Bragg equation, $\mathbf{k}' = \mathbf{k}_0 + \mathbf{g}$. After substitution in the perfect crystal Schrödinger equation, application of the high energy approximation, and the substitution:

$$\psi_{\mathbf{g}}(\mathbf{r}) = S_{\mathbf{g}}(\mathbf{r}) \mathrm{e}^{\mathrm{i}\theta_{\mathbf{g}}}, \tag{22}$$

where $\theta_{\mathbf{g}}$ is the phase of the Fourier coefficient $U_{\mathbf{g}}$, we obtain the following system of coupled first order differential equations:

$$\frac{\mathrm{d}S_{\mathbf{g}}(z)}{\mathrm{d}z} = 2\pi\mathrm{i}s_{\mathbf{g}} S_{\mathbf{g}}(z) + \mathrm{i}\pi \sum_{\mathbf{g}'} \frac{1}{q_{\mathbf{g}-\mathbf{g}'}} S_{\mathbf{g}'}(z), \tag{23}$$

where

$$\frac{1}{q_{\mathbf{g}}} \equiv \frac{1}{\xi_{\mathbf{g}}} + \mathrm{i}\frac{\mathrm{e}^{\mathrm{i}(\theta'_{\mathbf{g}}-\theta_{\mathbf{g}})}}{\xi'_{\mathbf{g}}}; \tag{24}$$

the extinction distance $\xi_{\mathbf{g}}$ and anomalous absorption length $\xi'_{\mathbf{g}}$ are defined as:

$$\frac{1}{\xi_{\mathbf{g}}} \equiv \frac{|U_{\mathbf{g}}|}{|\mathbf{k}_0+\mathbf{g}|\cos\alpha}; \qquad \frac{1}{\xi'_{\mathbf{g}}} \equiv \frac{|U'_{\mathbf{g}}|}{|\mathbf{k}_0+\mathbf{g}|\cos\alpha}; \tag{25}$$

$U_{\mathbf{g}} = (2me/h^2)V_{\mathbf{g}}$, α is the angle between the beam direction and $\mathbf{k}_0+\mathbf{g}$, and $\mathbf{k}_0$ is the incident wave vector corrected for refraction. The absorption potential Fourier coefficients are represented by $U'_{\mathbf{g}} = (2me/h^2)V'_{\mathbf{g}}$.

This set of equations can be represented in matrix form as:

$$\frac{\mathrm{d}\mathbf{S}(z)}{\mathrm{d}z} = \mathrm{i}\mathcal{A}(\mathbf{r})\mathbf{S}(z), \tag{26}$$

where the structure matrix $\mathcal{A}$ contains the excitation errors and the normal absorption coefficient $1/q_{\mathbf{0}}$ along its diagonal, and the interaction parameters $1/q_{\mathbf{g}}$ on the off-diagonal positions:

$$\mathcal{A}_{\mathbf{g},\mathbf{g}} = 2\pi s_{\mathbf{g}} + \frac{\pi}{q_{\mathbf{0}}};$$
$$\mathcal{A}_{\mathbf{g},\mathbf{g}'} = \frac{\pi}{q_{\mathbf{g}-\mathbf{g}'}}.$$

The formal solution for a crystal of thickness ϵ is then given by:

$$\mathbf{S}(\epsilon) = e^{\mathrm{i}\mathcal{A}\epsilon}\mathbf{S}(0) = \mathcal{S}(\epsilon)\mathbf{S}(0); \tag{27}$$

the matrix exponential $\mathcal{S}(z)$ is commonly known as the *scattering matrix*, and can be computed numerically by means of the Padé approximation [**moler2003a**].

Substitution of eq. (21) into the BSE yield equation results in the following expression for the BSE yield:

$$\mathcal{P}(\mathbf{k}_0) = \frac{1}{z_0}\sum_n \sum_{i\in\mathcal{S}_n} \sum_{\mathbf{g}} \sum_{\mathbf{h}} \int_0^{z_0} \mathrm{d}z\, Z_n^2\, \mathrm{e}^{-M^{(n)}_{\mathbf{h}-\mathbf{g}}}\, \mathrm{e}^{2\pi\mathrm{i}(\mathbf{h}-\mathbf{g})\cdot\mathbf{r}_i} S^*_{\mathbf{g}}(z) S_{\mathbf{h}}(z). \tag{28}$$

This expression can be simplified as for the Boch wave case to read:

$$\mathcal{P}(\mathbf{k}_0) = \sum_{\mathbf{g}} \sum_{\mathbf{h}} S_{\mathbf{gh}} L^S_{\mathbf{gh}}, \tag{29}$$

where the matrices are defined as

$$S_{\mathbf{gh}} \equiv \sum_n \sum_{i\in\mathcal{S}_n} Z_n^2\, \mathrm{e}^{-M^{(n)}_{\mathbf{h}-\mathbf{g}}}\, \mathrm{e}^{2\pi\mathrm{i}(\mathbf{h}-\mathbf{g})\cdot\mathbf{r}_i}; \tag{30a}$$

$$L^S_{\mathbf{gh}} \equiv \frac{1}{z_0}\int_0^{z_0} \mathrm{d}z\, S^*_{\mathbf{g}}(z) S_{\mathbf{h}}(z). \tag{30b}$$

The superscript S in $L^S_{\mathbf{gh}}$ refers to the scattering matrix approach. Note that the matrix $S_{\mathbf{gh}}$ is identical to that for the Bloch wave approach in the previous section (eq. (18a)). As in the Bloch wave case, we must replace the integral above by an energy-dependent depth-weighted integration, as follows:

$$L^S_{\mathbf{gh}}(E) \equiv \frac{1}{z(E)}\int_0^{z(E)} \mathrm{d}z\, \lambda(E,z) S^*_{\mathbf{g}}(z) S_{\mathbf{h}}(z). \tag{31}$$

The entire computation of the matrix L can then be performed on the GPU, potentially resulting in a tremendous acceleration compared to the OpenMP CPU version. Each OpenCL work-item performs an entire matrix computation, with the limitation that all matrices must have the same size (same number of beams); this requires a rather comprehensive rewrite of the fortran-90 code in the main EMEBSDmaster program, which is currently underway.]

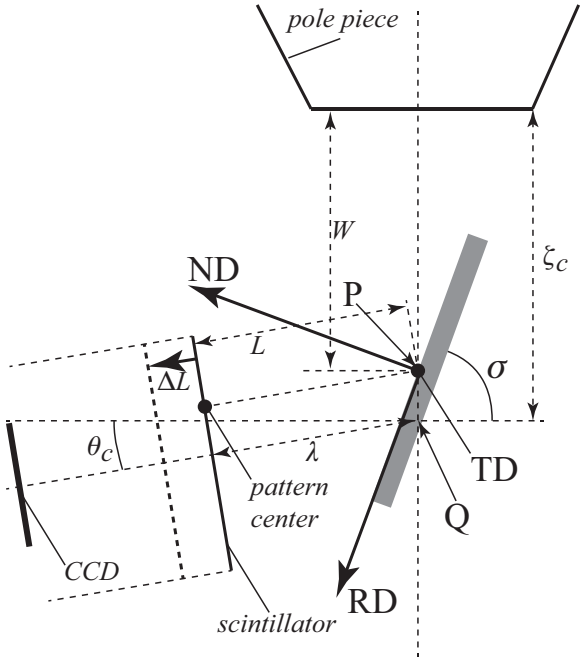

Figure 3: *Typical geometry of sample and scintillator for an EBSD experiment.*

5.6 Final EBSD Pattern Simulations

Consider the geometry in Fig. 3, which shows the sample-scintillator configuration for a typical EBSD experiment. The diffraction geometry is fully determined if we can express the direction cosines of the line connecting an arbitrary point on the scintillator screen to the interaction point P on the sample surface, expressed with respect to the (RD, TD, ND) triad. We measure the scintillator coordinates with the x_s axis in the direction opposite to TD, and the y_s axis pointing up towards the objective lens. The z_s axis is then normal to the screen and forms a second right-handed unit triad. These coordinates are measured with respect to the center of the screen and can be positive or negative (except for z_s which is always positive). In the scintillator reference frame, a single point on the screen has coordinates $(x_s, y_s, 0)$ and the point Q has coordinates $(0, 0, \lambda)$. The interaction point P has coordinates (x_{pc}, y_{pc}, L). For simplicity, we will measure all coordinates in units of micrometers (μm).

The arbitrary point $(x_s, y_s, 0)$ on the scintillator screen corresponds to a set of three direction cosines with respect to the sample reference frame. The angle between ND and the scintillator normal is $\alpha = \frac{\pi}{2} - \sigma + \theta_c$. If the pattern center were located on the ND axis, by rotating the detector assembly by a clockwise angle α around the TD direction, then its (RD, TD, ND) coordinates (keeping track of the definition of this reference frame) would be $(y_{pc} - y_s, x_{pc} - x_s, L)$. After counterclockwise rotation to the actual camera orientation we find for the screen coordinates in the sample reference frame:

$$\mathbf{r}_g = [(y_{pc} - y_s)\cos\alpha + L\sin\alpha, x_{pc} - x_s, -(y_{pc} - y_s s)\sin\alpha + L\cos\alpha] \,.$$

The length of this vector is simply:

$$|\mathbf{r}_g| \equiv \rho_s = \sqrt{L^2 + (y_{pc} - y_s)^2 + (x_{pc} - x_s)^2}$$

so that the direction cosines of a screen pixel $(x_s, y_s, 0)$ in the (RD,TD,ND) reference frame are

given by:

$$\hat{\mathbf{r}}_g(x_s, y_s) = \frac{\mathbf{r}_g}{\rho_s}. \tag{32}$$

For a given crystallographic grain orientation, it is straightforward to convert these direction cosines to an electron channeling direction (all rotations are implemented by means of quaternion arithmetic in our implementation). This transformed direction is then further converted to coordinates (X, Y) on the square grid of the master EBSD pattern, which is used as a look-up table (with appropriate bi-linear interpolation) for the EBSD intensity at the scintillator position $(x_s, y_s, 0)$. Repeating this for all scintillator pixels then provides the complete EBSD signal for a given grain orientation.

We represent the energy-dependent master EBSD pattern by the symbol $\mathcal{M}(\kappa, X, Y)$, where the first component labels the energy bins, and (X, Y) indicates the position on the square sampling grid. The energy histogram from the Monte Carlo simulation is represented by the symbol $\mathcal{E}(\kappa, i, j)$, where (i, j) references the scintillator pixel $(x_s, y_s) = (i, j)\delta$. The number of BSE electrons incident on that pixel can then be written as a weighted sum over all n_E energy bins:

$$I_{\mathrm{BSE}}(i, j) = \eta \sum_{\kappa=1}^{n_E} s(\kappa)\mathcal{E}(\kappa, i, j)\mathcal{M}\left(\kappa, X(x_s, y_s, \tilde{q}), Y(x_s, y_s, \tilde{q})\right), \tag{33}$$

where $\tilde{q}$ is a unit quaternion that represents the grain orientation (derived from a triplet of Euler angles), and the coordinates (X, Y) are computed from (x_s, y_s) and $\tilde{q}$ by the sequential application of equation (32), the quaternion rotation of the resulting direction cosines, and the inverse Lambert projection equations. The prefactor η in (33) is defined as:

$$\eta = \frac{n_A I_0 \Delta t}{N_{MC}} \tag{34}$$

where I_0 is the incident probe current in Amperes, Δt the dwell time in seconds, N_{MC} is the total number of incident electrons used for the Monte Carlo simulation, and $n_A = 6.241 \times 10^{18}$ electrons per Coulomb. The resulting units for I_{BSE} are then number of electrons incident on the scintillator. The factor $s(\kappa)$ represents the energy conversion efficiency of the scintillator and is currently set equal to unity for all energies.

This is not the final EBSD pattern as acquired by the CCD camera. One must still take into account the point spread function of the optics that projects the photons onto the CCD chip, Poisson noise, as well as any binning and contrast/brightness scaling that can be applied to the pattern. These final steps are currently not available, but the simulated patterns already have a satisfactory agreement with experimental observations.

5.7 Final ECP Pattern Simulations

The geometry for the ECP case is quit different from the EBSD case. First of all, the sample is typically nearly horizontal for ECP, and, secondly, the detector is an integrating detector, not a position sensitive detector. Therefore, the ECP is obtained by scanning the incident beam inside a conical volume with apex as the rocking point on the sample surface, and then measuring the integrated detector signal as a function of the incidence direction. Simulation of this process requires knowledge of the detector acceptance angles, which involves the working distance, W, and the detector inner and outer radii, R_i and R_o. Furthermore, the sample may be inclined slightly with respect to the horizontal plane (in particular for ECCI observations), so this must be taken into account in the detector model. Finally, the incident beam cone opening angle, θ_c, must be specified.

5.8 Final TKD Pattern Simulations

The most important differences between TKD and EBSD are the sample thickness, which must be specified as an input parameter to EMMCfoil, and the sample tilt angle, which is a negative angle, typically in the $-20°$ to $-30°$ range. All other geometrical parameters are identical to those used for EBSD patterns, and the EMEBSD program can be used with a TKD master pattern input file to compute TKD patterns.

5.9 Kossel Pattern Simulations

Kossel patterns represent the probability that an incident beam electron survives as part of the (coherent) beam up to a certain depth in the sample. They are computed using the same Bloch wave code that is used for the EBSD master pattern simulation. The intensity level in a Kossel pattern is determined from the Bloch wave eigenvalues and excitation amplitudes by the following expression:

$$P(z, \mathbf{k}_0) = \sum_j |\alpha^{(j)}|^2 e^{-4\pi q^{(j)} z}. \tag{35}$$

This computation is then repeated for each of the incident beam directions, and the results are collected and displayed as an electron Kossel pattern. The EMKosselMaster program can be used to compute the master pattern in the same way as for the EMEBSDmaster and EMECPmaster programs. The IDL SEMDisplay routine discussed in a later section has resources for the display of the master pattern as well as the computation of specific Kossel patterns for arbitrary crystal orientation.

6 The EMMCOpenCL.f90 and EMMCfoil.f90 programs

For EBSD and ECP simulations, the Monte Carlo program EMMCOpenCL must be executed first; this program takes the following name list as input:

```
 &MCCLdata
! only bse1, full or Ivol simulation
 mode = 'full' ! 'bse1' or 'full', 'Ivol',
! name of the crystal structure file
 xtalname = 'undefined',
! for full mode: sample tilt angle from horizontal [degrees]
 sig = 70.0,
! for bse1 mode: start angle
 sigstart = 0.0,
! for bse1 mode: end angle
 sigend = 30.0,
! for bse1 mode: sig step size
 sigstep = 2.0,
! sample tilt angle around RD axis [degrees]
 omega = 0.0,
! number of pixels along x-direction of square projection [odd number!]
 numsx = 501,
! number of incident electrons per thread
 num_el = 10,
! GPU platform ID selector
 platid = 1,
! GPU device ID selector
 devid = 1,
! number of work items (depends on GPU card; leave unchanged)
```

```
 globalworkgrpsz = 150,
! total number of incident electrons and multiplier (to get more than 2^(31)-1 electrons)
 totnum_el = 2000000000,
 multiplier = 1,
! incident beam energy [keV]
 EkeV = 30.D0,
! minimum energy to consider [keV]
 Ehistmin = 15.D0,
! energy binsize [keV]
 Ebinsize = 1.0D0,
! maximum depth to consider for exit depth statistics [nm]
 depthmax = 100.D0,
! depth step size [nm]
 depthstep = 1.0D0,
! should the user be notified by email or Slack that the program has completed its run?
 Notify = 'Off',
! output data file name; pathname is relative to the EMdatapathname path !!!
 dataname = 'MCoutput.h5'
 /
```

The program uses OpenCL coding to carry out the simulations on a GPU card if one is available. The input parameters include the crystal file name (to compute the theoretical density and average atomic number), the sample tilt angle with respect to the horizontal direction, a potential tilt around the RD axis, the number of pixels along the modified Lambert projection edge (should be "a multiple of 10 plus one"), the number of incident electrons per computational thread, the incident beam energy, the minimum energy to be considered, and the energy interval, as well as the maximum penetration depth and depth step size to be considered. The data is then stored in an HDF5 file. It should be noted that the execution time of the following program, EMEBSDmaster, is directly determined by how many depth steps and energy bins are selected in the Monte Carlo computation; the more depth steps/energy bins, the longer the dynamical simulation will take.

It should be noted that this program can provide statistics that are used by two other programs: EMEBSDmaster.f90 and EMECPmaster.f90. To generate output for the former, the variable mode should be set to 'full', and the sample tilt angle from horizontal (sig) should be specified. For the latter program, the mode should be set to 'bse1', and the user should provide the parameters sigstart, sigend, and sigstep. In the bse1 mode, the program expects the sample to be oriented normal to the electron beam (sigstart=0.0) initially, and then the computation is repeated for increasingly larger sample tilt angles (equivalent to beam tilts). For each tilt value, the OpenCL computation is relatively fast since only BSE1 electrons are considered; the total computation time depends on the number of sample tilt values that need to be considered. For the EBSD case, with mode='full', the computation considers all electrons, both BSE1 and BSE2, since both types contribute to the EBSD pattern; in the ECP case, only the BSE1 electrons carry diffraction information, since the BSE2 electrons are fully incoherent with respect to the incident beam, and the integrating annular detector averages any channeling effects of the electrons on their way out of the sample.

The HDF5 output file contains a number of program parameters that need to be passed on to the next program, as well as two large arrays that contain the electron distribution in the hemisphere on the vacuum side of the sample surface, formatted as a modified Lambert projection. Each energy bin has an individual Lambert projection. The second large output array represents the distribution of electron exit depth as a function of energy bin and orientation, but sampled on a smaller Lambert projection (with one-tenth the number of points along each axis). This second output array is used in the EMEBSDmaster program, whereas the first one is primarily used by the EMEBSD program.

The default number of electrons to be considered is set to 2×10^9; the actual number may be

slightly different, due to the way the GPU distributes the work to multiple work-items. In our test runs on the latest generation Mac Pro platform, a typical Monte Carlo run takes about 25 minutes. On a high-end GPU card, such as the NVidia GTX 1080 card, a similar run may take less than five minutes, depending on the parameters settings.

It is important to note that in release 3.1, the Monte Carlo output file will also contain the master pattern data, so that there is only one file to be considered by any subsequent program. This has a practical consequence in terms of the file naming convention. Starting in release 3.1, we recommend that the output file from the Monte Carlo program have a file name with the following structure: compound-name-master-XXkV.h5, where XX is the accelerating voltage in kV, and compound-name is the name of the .xtal file that contains the structural information. So, for Silicon at 20 kV, the following file name is suggested: Si-master-20kV.h5. Note that all filenames (except for the crystal structure file name) are relative to the EMdatapathname path. So, if your EMdatapathname equals /Users/me/EMdata and you wish the Monte Carlo output file to have the absolute path /Users/me/EMdata/test/Si-master-20kV.h5, then only the portion test/Si-master-20kV.h5 should be entered as the file name in the name list file. Relative pathnames are used so that data files can be transferred between computers with different directory structures.

Monte Carlo and master pattern files created with Release 3.0 can be converted to the single file format by means of the new EMmergefiles utility program. This program works for both EBSD and ECP programs and automatically merges a Monte Carlo file with a master file. If the existing files are Si-MC.h5 and Si-master.h5, and the microscope voltage is 25 kV, then a new file Si-master-25kV.h5 file can be created as follows:

```
EMmergefiles -EBSD Si-MC.h5 Si-master.h5 Si-master-25kV.h5
```

Note the order of the three arguments. To determine the syntax for this program, simply execute the following command: EMmergefiles -h.

For TKD pattern simulations, one must first execute the EMMCfoil program, which takes the following namelist as input:

```
 &MCCLfoildata
! name of the crystal structure file
 xtalname = 'undefined',
! for full mode: sample tilt angle from horizontal [degrees]
 sig = -20.0,
! sample tilt angle around RD axis [degrees]
 omega = 0.0,
! number of pixels along x-direction of square projection [odd number!]
 numsx = 501,
! number of incident electrons per thread
 num_el = 10,
! GPU platform ID selector
 platid = 1,
! GPU device ID selector
 devid = 1,
! number of work items (depends on GPU card; leave unchanged)
 globalworkgrpsz = 150,
! total number of incident electrons and multiplier (to get more than 2^(31)-1 electrons)
 totnum_el = 2000000000,
 multiplier = 1,
! incident beam energy [keV]
 EkeV = 30.D0,
! minimum energy to consider [keV]
 Ehistmin = 10.D0,
```

```
! energy binsize [keV]
 Ebinsize = 1.0D0,
! max depth [nm] (this is the maximum distance from the bottom foil surface to be considered)
 depthmax = 100.0D0,
! depth step size [nm]
 depthstep = 1.0D0,
! total foil thickness (must be larger than depth)
 thickness = 200.0,
! output data file name; pathname is relative to the EMdatapathname path !!!
 dataname = 'MCoutput.h5'
 /
```

Most of the parameters have the same meaning as for the EBSD case. The main differences are that the sample tilt angle sig must be negative since the sample must be tilted such that its bottom plane faces the EBSD detector (for the conventional detector geometry); furthermore, there is a new parameter, thickness, which must be set to the foil thickness. The depthmax parameter represents the maximum distance from the exit (bottom) surface for which electron statistics is recorded; most of the TKD signal originates in the bottom of the sample. For thick samples, the electrons that exit the sample will have lost a large amount of energy, resulting in a downward shift of the exit energy distribution compared to thinner foils. *For this reason, the TKD Monte Carlo program must be executed for each desired sample thickness.*

7 The EMEBSDmaster.f90, EMECPmaster.f90, and EMTKDmaster.f90 programs

The second program performs the computation of the EBSD master pattern (or the ECP master pattern) on a square modified Lambert projection. The input name list file for the EMEBSDmaster program is formatted as follows:

```
 &EBSDmastervars
! smallest d-spacing to take into account [nm]
 dmin = 0.05,
! number of pixels along x-direction of the square master pattern  (2*npx+1 = total number)
 npx = 500,
! name of EMMCOpenCL output file to be used to copy the MC data from for this master pattern run;
! This can be used to perform multiple master pattern runs starting from the same MC data set without
! having to rerun the MC computation.  Leave this variable set to 'undefined' if not needed.
 copyfromenergyfile = 'undefined',
! name of the energy statistics file produced by EMMCOpenCL program; relative to EMdatapathname;
! this file will also contain the output data of the master program
 energyfile = 'MCoutput.h5',
! number of OpenMP threads
 nthreads = 1,
! do you wish to receive a notification (Email or Slack) when the program completes ?
 Notify = 'Off',
! restart computation ?
 restart = .FALSE.,
! create output file with uniform master patterns set to 1.0 (used to study background only)
 uniform = .FALSE.,
 /
```

For the EMECPmaster program, the input file is nearly identical:

```
 &ECPmastervars
! no. of pixels in the X and Y direction
 npx = 500,
! smallest d-spacing to take into account [nm]
 dmin = 0.05,
! name of EMMCOpenCL output file to be used to copy the MC data from for this master pattern run;
! This can be used to perform multiple master pattern runs starting from the same MC data set without
! having to rerun the MC computation.  Leave this variable set to 'undefined' if not needed.
 copyfromenergyfile = 'undefined',
! name of the energy statistics file produced by EMMCOpenCL program; relative to EMdatapathname
! this file will also contain the output of the master program
 energyfile = 'undefined',
! set to 'On' if you want Slack or Email notification at the end of program run
 Notify = 'Off',
! number of threads to run the master pattern
 nthreads = 1
 /
```

And for the TKD master pattern we have:

```
 &TKDmastervars
! smallest d-spacing to take into account [nm]
 dmin = 0.05,
! number of pixels along x-direction of the square master pattern  (2*npx+1 = total number)
 npx = 500,
! name of the energy statistics file produced by EMMCfoil program; relative to EMdatapathname
! this file will also contain the output data of the master program
 energyfile = 'MCoutput.h5',
! number of OpenMP threads
 nthreads = 2,
! restart computation ?
 restart = .FALSE.,
! create output file with uniform master patterns set to 1.0 (used to study background only)
 uniform = .FALSE.,
 /
```

The only differences are the restart and uniform parameters described below.

These short input files only require the definition of the smallest d-spacing to be taken into account in the dynamical simulation, the number of pixels along the output Lambert projection, the name of the output file from the EMMCOpenCL or EMMCfoil program; starting in this release 3.1, the name of an HDF5 output file is no longer needed, since the Monte Carlo output file will be used to also store all the master pattern data.

The EMEBSDmaster program can take a very long time to run, despite the fact that the current version is a multi-threaded OpenMP code. For each energy bin, a complete EBSD master pattern is computed, using the appropriate asymmetric unit according to the crystal symmetry. The output HDF5 file contains a stack of modified Lambert projections, one for each energy bin, along with some other parameters. The size of the Lambert projections does not need to be the same as the size of the Monte Carlo Lambert projections. The projections can be visualized using the IDL program discussed in section 14. Note that for complex crystal structures, with multiple atom sites in the asymmetric unit, the program will store the master patterns separately for each entry in the asymmetric unit. This means that one can, in principle, study the effect of elemental substitutions on EBSD patterns. The EMEBSDmaster program can be interrupted at any time, since the master patterns will be written to the HDF5 output file for each energy bin. The restart parameter (set to

.FALSE. by default) must then be set to .TRUE. and the computation will resume with the pattern for the next energy bin. It is therefore possible to carry out a few energy bins on one computer, and then continue the computation on another one.

The parameter dmin determines a truncation value in direct space, which is effectively a truncation in Fourier space; decreasing this value below the default will dramatically increase the execution time. For complex crystal structures, the simulation time will also increase, since the larger unit cell will automatically generate more reciprocal lattice points inside the truncation sphere.

The -t option to the EMEBSDmaster, EMECPmaster, and EMTKDmaster programs will also create a file named BetheParameters.template in your folder. Rename this file to BetheParameters.nml and set the following parameters, which have been found to be reasonable values for a number of crystal structures and typical SEM accelerating voltages:

```
 &Bethelist
! Do not change the line above !
!
! Details about the particular implementation of the Bethe potentials can be
! found in the recent paper: A. Wang & M. De Graef, "Modeling dynamical electron
! scattering with Bethe potentials and the scattering matrix" Ultramicroscopy,
! vol 160, pp. 35-43 (2016)
!
! strong beam cutoff
 c1 = 4.0,
! weak beam cutoff
 c2 = 8.0,
! complete cutoff
 c3 = 50.0,
! double diffraction reflections: maximum excitation error to include [nm^{-1}]
 sgdbdiff = 1.00
 /
```

increasing c_1 and c_2 will increase the computation time, but will also increase the accuracy of the result. For TEM simulations, to be released at a later point in time, the values of c_1 and c_2 will likely need to be increased. Note that this name list file will also be copied into the master output file.

8 Sending program completion notifications

Some of the programs in the EMsoft package can take quite a while to run. In version 3.2, we have added an option to have those programs automatically notify the user when the run has completed. The notification can be either in the form of an email message, or via a message push to a slack.com channel. Access to this new feature only requires a few additional lines in the EMsoftConfig.json configuration file.

8.1 Email notifications

Email notification is very easy to set up and only requires one additional configuration parameter:

"EMNotify": "Email",

In addition, you must make sure that the UserEmail parameter has the correct email address to which the notifications should be sent. The computer that you execute the program on must be configured to send email messages.

8.2 Slack notifications

Slack notifications require that the user set up a slack.com message channel. To do so, first obtain a team Slack URL of the form your-team-url.slack.com; even if you are just working by yourself, you will still need to set up a team. You could, for instance, name your team in the following form: institution-EMsoft.slack.com, where institution is the abbreviation for your university or research entity. Then you provide your email address and a password to complete the set up of your slack account (this is the free account).

Once you have a team URL, you will find that you have two default message channels, #general and #random; we suggest that you create an #EMsoft channel as well. At this point, you may want to also install the desktop and/or mobile slack app which you can download from the slack.com site or via the Apple Store for Mac OS X users. You can access your slack.com account via a web browser or via the app. You will also need to allow slack to push notifications to your desktop/mobile.

Next, you will need to set up incoming webhook integration. Point your browser to

> https://api.slack.com/custom-integrations

Select the Incoming Webhooks menu item. In the first section titled *Send data into Slack in real-time*, click on the incoming webhook integration link in the paragraph starting with *Start by setting up* .... You will get a page titled Incoming Webhooks; select the name of the channel that you wish to use for messaging from the pull down menu in the *Post to Channel* form. Then click the green button, which will get you to an integration settings page. From this page you will need to copy the Webhook URL entry into you EMsoftConfig.json configuration file; this is a long entry with several code strings in it that uniquely define your message channel, so you should not share this URL with anyone else (unless you want that person to be able to post messages to your channel).

In the EMsoftConfig.json file, add the following three lines:

> "EMNotify": "Slack",
> "EMSlackWebHookURL": "https://hooks.slack.com/services/T6.../B6...",
> "EMSlackChannel": "channelname",

where channelname must be replaced by the name of the channel you want messages to be posted to (without the # character). On the *Integration Settings* page, check the other items and customize as needed, then Save the Settings. You should now have a functioning EMsoft message channel! Try it by running the EMsoftSlackTest program from the command line...

8.3 Adding notifications to your own f90 programs

If you are developing EMsoft code, you can easily add notification support to your code. Here is a simple example of code that you could insert into your own subroutine or main program:

```
! at the top of your subroutine
use notifications

! in the variable declaration section
character(fnlen),ALLOCATABLE      :: MessageLines(:)
integer(kind=irg)                 :: NumLines
character(fnlen)                  :: MessageTitle
character(100)                    :: c
```

```
! at the end of your program
if (trim(EMsoft_getNotify()).ne.'Off') then
  if (trim(namelist%Notify).eq.'On') then
! allocate the message string array
    NumLines = 2
    allocate(MessageLines(NumLines))
! get the hostname
    call hostnm(c)
! initialize the message strings
    MessageLines(1) = 'ProgramName program has ended successfully'
    MessageLines(2) = 'Data stored in '//trim(outname)
! define the title of the message (will appear in bold face at the top of your message)
    MessageTitle = 'EMsoft on '//trim(c)
! and post the message to Slack
    i = PostMessage(MessageLines, NumLines, MessageTitle)
  end if
end if
```

In this code, the variable "outname" is a full path file name. The variable *namelist%Notify* is part of a namelist that was read from a namelist parameter file; see the EMMCOpenCL.template file for an example. Obviously, you can add more output lines by increasing the *NumLines* parameter and entering appropriate text in the *MessageLines* string array. You could print out the values of certain program variables, for instance the execution time, or the final results of the calculation.

When executed from a workstation with IP name mycomputer.edu, this code will produce a Slack message of the following form:

```
EMsoft on MYCOMPUTER.EDU APP [1:50 PM]
ProgramName program has ended successfully
Data stored in /Users/.../Files/.../testfile.h5
```

Finally, we should note that a free slack.com account entitles you to a maximum of $10,000$ messages; you can manually delete messages one by one, but to remove that restriction you will have to register with a paid account. Slack makes it nearly impossible to perform bulk deletes of messages by limiting how many individual delete commands can be requested per second.

9 The EMEBSD.f90 program

The final program takes as input a series of detector characteristics as well as orientations, and uses the output from both EMMCOpenCL and EMEBSDmaster to compute one or more actual EBSD patterns. The easiest way to execute this program is via the IDL interface introduced later, but command line execution is also possible, as is execution as a filter in DREAM.3D. The input name list file contains the following entries:

```
 &EBSDdata
! template file for the EMEBSD program
!
! distance between scintillator and illumination point [microns]
 L = 15000.0,
! tilt angle of the camera (positive below horizontal, [degrees])
 thetac = 10.0,
```

```
! CCD pixel size on the scintillator surface [microns]
 delta = 50.0,
! number of CCD pixels along x and y
 numsx = 640,
 numsy = 480,
! pattern center coordinates in units of pixels
 xpc = 0.0,
 ypc = 0.0,
! angle between normal of sample and detector
 omega = 0.0,
! transfer lens barrel distortion parameter
 alphaBD = 0.0,
! energy range in the intensity summation [keV]
 energymin = 5.0,
 energymax = 20.0,
! name of angle file (euler angles or quaternions); path relative to EMdatapathname
 anglefile = 'testeuler.txt',
! 'tsl' or 'hkl' Euler angle convention parameter
 eulerconvention = 'tsl',
! name of EBSD master output file; path relative to EMdatapathname
 masterfile = 'master.h5',
! name of Monte Carlo output file; path relative to EMdatapathname
 energyfile = 'MC.h5',
! name of output file; path relative to EMdatapathname
 datafile = 'EBSDout.h5',
! bitdepth '8bit' for [0..255] bytes; 'float' for 32-bit reals; '##int' for 32-bit integers with ##-bit dynamic ran
! e.g., '9int' will get you 32-bit integers with intensities scaled to the range [ 0 .. 2^(9)-1 ];
! '17int' results in the intensity range [ 0 .. 2^(17)-1 ]
 bitdepth = '8bit',
 ! incident beam current [nA]
 beamcurrent = 150.0,
! beam dwell time [micro s]
 dwelltime = 100.0,
! binning mode (1, 2, 4, or 8)
 binning = 1,
! should we perform an approximate computation that includes a lattice distortion? ('y' or 'n')
! This uses a polar decomposition of the deformation tensor Fmatrix which results in
! an approcimation of the pattern for the distorted lattice; the bands will be very close
! to the correct position in each pattern, but the band widths will likely be incorrect.
 applyDeformation = 'n'
! if applyDeformation='y' then enter the 3x3 deformation tensor in column-major form
! the default is the identity tensor, i.e., no deformation
 Ftensor = 1.D0, 0.D0, 0.D0, 0.D0, 1.D0, 0.D0, 0.D0, 0.D0, 1.D0,
! intensity scaling mode 'not' = no scaling, 'lin' = linear, 'gam' = gamma correction
 scalingmode = 'not',
! gamma correction factor
 gammavalue = 1.0,
! should a circular mask be applied to the data? 'y', 'n'
 maskpattern = 'n',
! number of threads (default = 1)
 nthreads = 1,
 /
```

The program reads the name list file, then the master pattern file; then the detector geometry is used to compute the energy distribution with respect to the scintillator. This is then used along with the master pattern and the orientation information to compute the final EBSD patterns. For a detailed description of all detector parameters we refer the interested reader to both the IDL

description in section 14 and our paper on the EBSD forward model.[6] The output file contains all EBSPs in full resolution, and can be read with the IDL visualization program; at that point, the camera point spread function can be included (currently in a future release), as well as binning and brightness/contrast scaling. See section 14 for details.

Note that this program can also be used to create so-called EBSD dictionaries. This is a much more advanced functionality that we will not fully detail in the present manual, since it is still under development. Dictionary-type computations were first introduced with release 3.1, and are described in a separate manual.

The outputformat variable should be kept at the value 'bin' for manual program runs; it will take on the value 'gui' for runs that are controlled by the IDL routine described in section 14.

The EMEBSD program has three new options in version 3.2:

1. bitdepth: Before this release, the only output format for the EBSD patterns was 8-bit grayscale with intensities between 0 and 255. The new bitdepth parameter is a character string that can take on the following values:
   - '8bit': this is the default and corresponds to the original output format.
   - 'float': this produces output in 32-bit floating point format.
   - '##int': this produces output in 32-bit integer format; the characters ## stand for a number between 1 and 32 and indicate the dynamic range to which the output numbers will be scaled. For instance, a bit depth of '9int' will produce integer output with the intensities scaled between 0 and $2^9 - 1$; '12int' will produce the intensity range $[0 \dots 2^{12} - 1]$, and so on.

   Note that these latter options will produce an output file that is significantly larger (by about a factor of four) than the 8bit output format.

2. applyDeformation: The default value is 'n' (no deformation is applied to the unit cell); when deformation is desired, then this parameter should be set to 'y', and the Ftensor parameter should be present in the name list file.

3. Ftensor: This is a 3×3 deformation tensor (double precision) that should be entered in column-major format as a series of 9 numbers. The default value is the identity matrix. *This option has only been tested on cubic crystal structures and needs further analysis to make sure the results are correct for all lattice types.*

   The deformation tensor is defined as the transformation matrix from the undistorted crystallographic basis vectors $\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ to the new basis vectors $\{\mathbf{a}', \mathbf{b}', \mathbf{c}'\}$:

$$\begin{pmatrix} \mathbf{a}' \\ \mathbf{b}' \\ \mathbf{c}' \end{pmatrix} = \begin{pmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{pmatrix} \begin{pmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \end{pmatrix}$$

   The numbers needs to be entered in column-major order, i.e., in the order $F_{11}, F_{21}, F_{31}, F_{12}, \dots$. The program performs a polar decomposition of this tensor and applies both the unitary (rotation) and symmetric stretch parts of the decomposition to the master pattern interpolation process. As a result, the EBSD patterns for the distorted unit cell will have all bands and zone axes in the correct location, but the band widths will continue to be those for the undistorted

[6] P.G. Callahan and M. De Graef, "Dynamical EBSD Patterns Part I: Pattern Simulations," Microscopy and Microanalysis, vol. 19, pp. 1255-1265 (2013)

structure, i.e., changes in the lattice spacing do not result in changes of the Kikuchi band widths. The error is relatively small for small deformations.

The deformation tensor can also be used to achieve a change of the reference frame by entering a unitary (rotation) matrix for the components of Ftensor. In that case, the stretch part of the decomposition will be the identity matrix.

10 The EMECP.f90 program

The EMECP program uses the following template file:

```
 &ECPlist
! The line above must not be changed
!
! The values below are the default values for this program
!
! crystal structure filename
 xtalname = 'undefined',
! size of output pattern in pixels (image is always square npix x npix)
 npix = 256,
! half angle of cone for incident beams (degrees)
 thetac = 5.0,
! mask pattern or not; 'y' or 'n'
 maskpattern = 'n',
! monte carlo input file; path relative to EMdatapathname
 energyfile = 'undefined',
! master pattern input file; path relative to EMdatapathname
 masterfile = 'undefined',
! input file for list of angles
 anglefile = 'undefined',
! euler angle convention in input angle file; 'hkl' or 'tsl'
 eulerconvention = 'hkl',
! gamma scaling value
 gammavalue = 1.0,
! output format; 'gui' for IDL and 'bin' for dictionary
 outputformat = 'gui',
! output file ; path relative to EMdatapathname
 datafile = 'undefined',
! number of threads for parallel execution
 nthreads = 1,
! tilt of the sample; only one angle incorporated for now
 sampletilt = 0.D0,
! detector geometry parameters
! working distance [in mm]
 workingdistance = 13.0,
! inner radius of annular detector [in mm]
 Rin = 2.0,
! outer radius of annular detector [in mm]
 Rout = 6.0,
!
!=============================================================
! the nml parameters only used in case of two layered structure
! ignore them for normal ECP calculation
!=============================================================
! foil normal of film
 fn_f = 0,0,1,
! foil normal of substrate
```

```
 fn_s = 0,0,1,
! second (substrate) crystal structure filename
 xtalname2 = 'undefined',
! plane normal in Film
 gF = 0,0,0,
! plane normal in Substrate
 gS = 0,0,0,
! direction in Film
 tF = 0,0,0,
! direction in Substrate
 tS = 0,0,0,
! smallest d-spacing to take into account [nm]
 dmin = 0.025,
! film thickness if present  (0.0 if not present)
 filmthickness = 0.0,
! output file for film signal
 filmfile = 'undefined',
! output file for substrate signal
 subsfile = 'undefined'
 /
```

Note that the bottom set of parameters can be completely ignored in this version of the software. Most of the parameters are similar to the ones in the EBSD case, except for several of the detector parameters. The EMECP program can be called from the command line with an input file of this form, or it can be called directly from within the IDL GUI described in section 14.

11 The EMTKD program

The EMTKD program can be used to generate a dictionary of TKD patterns and has options that are basically identical to those of the EMEBSD program. It is the user's responsibility to enter the correct detector parameters; all other options function in exactly the same way. The new options regarding bit depth and unit cell deformation described in section 9 have not yet been implemented in the TKD program; this will happen in the next release (3.3).

12 The EMKosselMaster program

The EMKosselMaster program does not require any Monte Carlo simulations, and is controlled via the following namelist template file:

```
 &Kosselmasterlist
! do not change the first line in this file !
!
! Note: All parameter values are default values for the program
!
!--------------
! the name of the crystal input file MUST be specified (full pathname)
 xtalname = 'undefined',
! output goes to screen (6) or to a file (any number >10)
 stdout = 6,
! microscope accelerating voltage [kV]
 voltage = 200.,
! minimum d-spacing to be taken into account [nm]
 dmin = 0.025,
```

```
! output will be diffraction disks inside a square of area (2*npx+1)^2
 npx = 500,
! number of thicknes values to be used in the output
 numthick = 10,
! starting thickness value [nm]
 startthick = 10.0,
! thickness increment value [nm]
 thickinc = 10.0,
! thickness fraction parameter [dimensionless], used when Kosselmode = 'thicks'
 tfraction = 0.1,
! output mode 'normal' or 'thicks'
 Kosselmode = 'normal',
! output will be stored here (full pathname)
 outname = '/some/folder/on/my/disk/KosselMaster.data'
! this last line must be present!
 /
```

Most of the parameters should be clear from the previous sections. The Kosselmode parameter is set to 'normal' to compute a thickness-dependent Kossel master pattern; when the option is set to 'thicks', the output is a single master pattern (as always in HDF5 format) that represents the depth at which the probability defined in section 5.9 is equal to the value set by the tfraction parameter. The Kossel master pattern can be visualized by the SEMDisplay program.

13 Calling pattern generation routines from another program

13.1 Release 3.0 externally callable routines (IDL)

In release 3.0, we made three of routines available for external calls using IDL: SingleEBSDPatternWrapper, SingleECPatternWrapper and SingleKosselPatternWrapper. Each of these routines takes two arguments, argc and argv, where argc is the number of arguments and argv contains a series of pointers to each of the arrays that need to be passed on to the wrapper routines.[7]

The wrapper routines have the following function declarations:

```
function SingleEBSDPatternWrapper(argc, argv) bind(c, name='SingleEBSDPatternWrapper')

use,INTRINSIC :: ISO_C_BINDING

IMPLICIT NONE

INTEGER(c_size_t), VALUE, INTENT(IN)            :: argc
type(c_ptr), dimension(argc), INTENT(INOUT)     :: argv
REAL(c_float)                                   :: SingleEBSDPatternWrapper

function SingleECPatternWrapper(argc, argv) bind(c, name='SingleECPatternWrapper')

use,INTRINSIC :: ISO_C_BINDING

IMPLICIT NONE
```

[7] Note that this has only been tested on Mac OS X with the gfortran 5.3.0 compiler and IDL's call_external functionality. There are no guarantees that this will work with other configurations, but we would appreciate any feedback, error messages, and so forth for other platforms and environments so that we can work towards making these routines more generally available for direct linking between programs. For details on the fortran-C bindings, we recommend the fortran-2008 language definition at http://j3-fortran.org/doc/year/10/10-007r1.pdf.

```
INTEGER(c_size_t), VALUE, INTENT(IN)            :: argc
type(c_ptr), dimension(argc), INTENT(INOUT)     :: argv
REAL(c_float)                                   :: SingleECPatternWrapper
```

and

```
function SingleKosselPatternWrapper(argc, argv) bind(c, name='SingleKosselPatternWrapper')

use,INTRINSIC :: ISO_C_BINDING

IMPLICIT NONE

INTEGER(c_size_t), VALUE, INTENT(IN)            :: argc
type(c_ptr), dimension(argc), INTENT(INOUT)     :: argv
REAL(c_float)                                   :: SingleKosselPatternWrapper
```

and use the fortran 2003 ISO_C_BINDING module; each function returns a single 4-byte real which is equal to 1.0 when the function properly returns. The computed EBSD, ECP or Kossel pattern is returned via a pointer to the corresponding array. Each function takes a list of c_ptrs in the argv input array. The order of the pointers is roughly the same for each function but the structure of the arrays that they point to may be different.

13.1.1 EBSD pattern calculation

The pointers that are passed on to the EBSD pattern calculation wrapper routine must point to the following arrays:

1. ipar: an array of 8 integers of fortran type c_size_t (8-byte integers) that represent the array dimensions of the other input arrays. The components of ipar are:

ipar(1) if 0, then the complete detector geometry is computed; if 1, then the stored detector geometry is used. This is useful (i.e., speeds things up a little) when the geometry does not change between consecutive pattern simulations.

ipar(2) number of scintillator pixels s_x along horizontal direction

ipar(3) number of scintillator pixels s_y along vertical direction

ipar(4) number of energy bins n_E in Monte Carlo accum_e array

ipar(5) number n_{mc} of x or y pixels in Monte Carlo accum_e array (total $2n_{mc}+1$)

ipar(6) number n_{mp} of pixels in modified Lambert projected EBSD master pattern (total $2n_{mp}+1$)

ipar(7) number of special positions n_{set} for which the master patterns have been computed

ipar(8) number of orientations n_q (quaternions) for which patterns must be computed

2. fpar: an array of 9 floating point numbers of the type c_float (4-byte reals) that represent several detector geometry parameters. The components of fpar are:

fpar(1) pattern center x-coordinate [in units of scintillator pixels]

fpar(2) pattern center y-coordinate

fpar(3) scintillator pixel size [microns]

fpar(4) sample tilt angle from Monte Carlo simulation [degrees]

fpar(5) sample rotation angle around the RD direction [degrees]

fpar(6) detector inclination angle with respect to horizontal [degrees; positive below horizontal]

fpar(7) distance between illuminated point and pattern center [micron]

fpar(8) beam current [nA]

fpar(9) dwell time [micro s]

3. EBSDpattern: a floating point array of type c_float with dimensions $s_x \times s_y \times n_q$; memory must be allocated by the calling program.

4. quats: a floating point array of type c_float with dimensions $4 \times n_q$, holding the individual quaternions corresponding to the crystal orientations for which the EBSD patterns must be computed (note that quaternions have their scalar part as the first element, followed by the three-component vector part).

5. accum_e: a floating point array with the output from the Monte Carlo program; array dimensions $n_E \times n_{mc} \times n_{mc}$.

6. mLPNH: a floating point array with the Northern hemisphere portion of the EBSD master pattern; array dimensions $n_{mp} \times n_{mp} \times n_E \times n_{set}$.

7. mLPSH: a floating point array with the Southern hemisphere portion of the EBSD master pattern; array dimensions $n_{mp} \times n_{mp} \times n_E \times n_{set}$.

Note that all seven arrays must be properly defined by the calling program, and the pointers must be passed in the stated order in the argv array. It is the responsibility of the calling program to fill the quats array with the appropriate list of quaternions, and to read the accum_e, mLPNH and mLPSH arrays from the appropriate HDF5 files; nearly all of the array dimensions needed for the ipar array can be extracted from the HDF5 variables. After the routine has completed its calculations, the desired EBSD patterns can be found in the array EBSDpattern.

13.1.2 ECP pattern calculation

The pointers that are passed on to the ECP pattern calculation wrapper routine must point to the following arrays:

1. ipar: an array of 8 integers of fortran type c_size_t (8-byte integers) that represent the array dimensions of the other input arrays. The components of ipar are:

ipar(1) if 0, then the complete detector geometry is computed; if 1, then the stored detector geometry is used. This is useful (i.e., speeds things up a little) when the geometry does not change between consecutive pattern simulations.

ipar(2) number of ECP pattern pixels s_x along horizontal direction

ipar(3) number of ECP pattern pixels s_y along vertical direction

ipar(4) number of sample tilt angles n_s in Monte Carlo accum_e array

ipar(5) number n_{mc} of x or y pixels in Monte Carlo accum_e array (total $2n_{mc}+1$)

ipar(6) number n_{set} of independent atom positions for which a master pattern is available

ipar(7) number n_{mp} of pixels in modified Lambert projected ECP master pattern (total $2n_{mp}+1$)

ipar(8) number of orientations n_q (quaternions) for which patterns must be computed

2. fpar: an array of 8 floating point numbers of the type c_float (4-byte reals) that represent several detector geometry parameters. The components of fpar are:

fpar(1) illumination cone semi-opening angle [degrees]

fpar(2) sample tilt angle around RD axis [degrees]

fpar(3) working distance [mm]

fpar(4) inner radius of BSE detector [mm]

fpar(5) outer radius of BSE detector [mm]

fpar(6) start angle from Monte Carlo input file [degrees]

fpar(7) end angle from Monte Carlo input file [degrees]

fpar(8) angle step size from Monte Carlo input file [degrees]

3. ECpattern: a floating point array of type c_float with dimensions $s_x \times s_y \times n_q$; memory must be allocated by the calling program.

4. quats: a floating point array of type c_float with dimensions $4 \times n_q$, holding the individual quaternions corresponding to the crystal orientations for which the EBSD patterns must be computed (note that quaternions have their scalar part as the first element, followed by the three-component vector part).

5. accum_e: a floating point array with the output from the Monte Carlo program; array dimensions $n_s \times n_{mc} \times n_{mc}$.

6. mLPNH: a floating point array with the Northern hemisphere portion of the ECP master pattern; array dimensions $n_{mp} \times n_{mp} \times n_{set}$.

7. mLPSH: a floating point array with the Southern hemisphere portion of the ECP master pattern; array dimensions $n_{mp} \times n_{mp} \times n_{set}$.

Note that all seven arrays must be properly defined by the calling program, and the pointer must be passed in the stated order in the argv array. It is the responsibility of the calling program to fill the quats array with the appropriate list of quaternions, and to read the accum_e, mLPNH and mLPSH arrays from the appropriate HDF5 files; nearly all of the array dimensions needed for the ipar array can be extracted from the HDF5 variables. After the routine has completed its calculations, the desired electron channeling patterns can be found in the array EBSDpattern.

13.1.3 Kossel pattern calculation

The pointers that are passed on to the Kossel pattern calculation wrapper routine must point to the following arrays:

1. ipar: an array of 6 integers of fortran type c_size_t (8-byte integers) that represent the array dimensions of the other input arrays. The components of ipar are:

ipar(1) if 0, then the complete detector geometry is computed; if 1, then the stored detector geometry is used. This is useful (i.e., speeds things up a little) when the geometry does not change between consecutive pattern simulations.

ipar(2) number of Kossel pattern pixels s_x along horizontal direction

ipar(3) number n_{mp} of pixels in modified Lambert projected Kossel master pattern (total $2n_{mp}+1$)

ipar(4) number of orientations n_q (quaternions) for which patterns must be computed

ipar(5) number of sample depths n_d for which the master pattern was computed

ipar(6) integer identifying the depth for which the pattern(s) need to be computed.

2. fpar: an array of 1 floating point number of the type c_float (4-byte reals) that represent several detector geometry parameters. The components of fpar are:

fpar(1) semi-opening angle of the Kossel pattern [degrees]

3. Kosselpattern: a floating point array of type c_float with dimensions $s_x \times s_y \times n_q$; memory must be allocated by the calling program.

4. quats: a floating point array of type c_float with dimensions $4 \times n_q$, holding the individual quaternions corresponding to the crystal orientations for which the EBSD patterns must be computed (note that quaternions have their scalar part as the first element, followed by the three-component vector part).

5. mLPNH: a floating point array with the Northern hemisphere portion of the ECP master pattern; array dimensions $n_{mp} \times n_{mp} \times n_d$.

6. mLPSH: a floating point array with the Southern hemisphere portion of the ECP master pattern; array dimensions $n_{mp} \times n_{mp} \times n_d$.

Note that all six arrays must be properly defined by the calling program, and the pointer must be passed in the stated order in the argv array. It is the responsibility of the calling program to fill the quats array with the appropriate list of quaternions, and to read the mLPNH and mLPSH arrays from the appropriate HDF5 files; nearly all of the array dimensions needed for the ipar array can be extracted from the HDF5 variables. After the routine has completed its calculations, the desired Kossel patterns can be found in the array Kosselpattern.

13.2 Release 3.1 C/C++ callable routines

Starting with Release 3.1, we introduce an additional four routines that can be called from a C/C++ program. It is assumed that the C/C++ program will (1) read all the necessary input arrays from the HDF5 files generated by other programs, and (2) allocate memory for the resulting pattern array. In this section we provide a detailed description of these routines. The C/C++ header file can be found in the include/EMsoftLib/EMsoftLib.h file of the installed version of this package, or in the Source/EMsoftLib/EMsoftLib.h file of the development version. This file reads as follows:

```
#ifndef _emsoft_lib_H_
#define _emsoft_lib_H_


#ifdef __cplusplus
extern "C" {
#endif


typedef void (*ProgCallBackType)(size_t, int);

typedef void (*ProgCallBackType2)(size_t, int, int, float);
```

```
typedef void (*ProgCallBackType3)(size_t, int, int, int, int);


/**
* EBSD pattern calculations:
* @param ipar array with integer input parameters
* @param fpar array with float input parameters
* @param EBSDpattern output array
* @param quats quaternion input array
* @param accum_e array with Monte Carlo histogram
* @param mLPNH Northern hemisphere master pattern
* @param mLPSH Southern hemisphere master pattern
* @param callback callback routine to update progress bar
* @param object unique identifier for calling class instantiation
* @param cancel boolean to trigger cancellation of computation
*/

void EMsoftCgetEBSDPatterns
        (int32_t* ipar, float* fpar, float* EBSDpattern,
         float* quats, int32_t* accum_e, float* mLPNH, float* mLPSH,
         ProgCallBackType callback, size_t object, bool* cancel);


/**
* ECP calculations:
* @param ipar array with integer input parameters
* @param fpar array with float input parameters
* @param ECpattern output array
* @param quats quaternion input array
* @param accum_e array with Monte Carlo histogram
* @param mLPNH Northern hemisphere master pattern
* @param mLPSH Southern hemisphere master pattern
* @param callback callback routine to update progress bar
* @param object unique identifier for calling class instantiation
* @param cancel boolean to trigger cancellation of computation
*/

void EMsoftCgetECPatterns
        (size_t* ipar, float* fpar, float* ECpattern,
         float* quats, float* accum_e, float* mLPNH, float* mLPSH,
         ProgCallBackType callback, size_t object, bool* cancel);

/**
* Monte Carlo calculations:
* @param ipar array with integer input parameters
* @param fpar array with float input parameters
* @param atompos atom position, site occupations and Debye-Waller factors
* @param atomtypes atom numbers
* @param latparm lattice parameters
* @param accum_e array with Monte Carlo energy histogram
* @param accum_z array with Monte Carlo depth histogram
* @param callback callback routine to update progress bar
* @param object unique identifier for calling class instantiation
* @param cancel boolean to trigger cancellation of computation
*/


void EMsoftCgetMCOpenCL
        (int32_t* ipar, float* fpar, float* atompos, int32_t* atomtypes,
```

```
        float* latparm, int32_t* accum_e, int32_t* accum_z,
        ProgCallBackType2 callback, size_t object, bool* cancel);

/**
* EBSD master pattern calculations:
* @param ipar array with integer input parameters
* @param fpar array with float input parameters
* @param atompos atom position, site occupations and Debye-Waller factors
* @param atomtypes atom numbers
* @param latparm lattice parameters
* @param accum_z array with Monte Carlo depth histogram
* @param mLPNH modified Lambert projection northern hemisphere
* @param mLPSH modified Lambert projection southern hemisphere
* @param callback callback routine to update progress bar
* @param object unique identifier for calling class instantiation
* @param cancel boolean to trigger cancellation of computation
*/


void EMsoftCgetEBSDmaster
        (int32_t* ipar, float* fpar, float* atompos, int32_t* atomtypes,
        float* latparm, int32_t* accum_z,  float* mLPNH, float* mLPSH,
        ProgCallBackType3 callback, size_t object, bool* cancel);


/**
 * @brief HiPassFilterC
 * @param rdata real data to be transformed
 * @param dims dimensions of rdata array
 * @param w width of Gaussian profile
 * @param init (optional) initialize without computing anything
 * @param destroy (optional) destroy fft plans
 * @param fdata
 */
void HiPassFilterC(double* rdata, int32_t* dims, double* w, bool* init, bool* destroy, double* fdata);

#ifdef __cplusplus
}
#endif



#endif /*_EMSOFTLIB_H_*/
```

The first three declarations are for progress callback functions, which allow the C/C++ side to receive updates on the progress of the fortran-90 computation. The other four routines are: EMsoftCgetEBSDpatterns, EMsoftCgetECPpatterns,[8] EMsoftCgetMCOpenCL, and EMsoftCgetEBSDmaster. The parameters for each routine are described in the comment blocks. ipar and fpar are two arrays of 40 entries each, 32-bit integers in ipar and 32-bit floats in fpar; these variables contain all the information needed by the fortran routines to properly set the array dimensions and a few other things. The fortran-90 side of things can be found in the Source/EMsoftLib/EMdymod.f90 module. The ipar and fpar definitions are as follows (note that f90 arrays start with index 1, whereas C/C++ arrays start with index 0):

[8] Note that EMsoftCgetECPpatterns is a temporary version that is still being developed; external call to it may cause unpredictable returns.

```
! general information: the ipar and fpar arrays for all the routines that are C-callable
! are identical, so we document here their component definitions; to allow for future
! expansion, each array has 40 entries, of which about half are currently used.
!
! integer(kind=irg) :: ipar(40)  components
! ipar(1) : nx  = (numsx-1)/2  - number of points along semi-axis of Monte Carlo pattern
! ipar(2) : globalworkgrpsz  - GPU work group size parameter
! ipar(3) : num_el  - number of electrons per GPU work item
! ipar(4) : totnum_el  - total number of electrons
! ipar(5) : multiplier  - multiplier for totnum_el when more than 2^31-1 electrons are needed
! ipar(6) : devid  - GPU device id number
! ipar(7) : platid - GPU platform id number
! ipar(8) : CrystalSystem  - crystal system number (1-7)
! ipar(9) : Natomtypes  - number of atom types in asymmetric unit
! ipar(10): SpaceGroupNumber  - space group number
! ipar(11): SpaceGroupSetting  - space group setting (1 or 2)
! ipar(12): numEbins  - number of energy bins in MC file
! ipar(13): numzbins  - number of depth bins in MC file
! ipar(14): mcmode  ( 1 = 'full', 2 = 'bse1' )  - MC mode (EBSD or ECP)
! ipar(15): numangle  - number of angles for ECP mode
! ipar(16): nxten = nx/10  - number of points along semi-axis of MC depth array
! the following are only used in the master routine
! ipar(17): npx  - number of points along semi-edge of master pattern
! ipar(18): nthreads  - number of OpenMP threads to be used for master pattern computation
! the following are only used in the EBSD pattern routine
! ipar(19): numx - number of detector pixels along x
! ipar(20): numy - number of detector pixels along y
! ipar(21): nquats - number of orientation in quaternion set
! ipar(22): binning - pattern binning factor (0-3)
! ipar(23): binx - binned x-dimension
! ipar(24): biny - binned y-dimension
! ipar(25): anglemode - 0 for quaternions, 1 for Euler angle triplets (in radians)
! ipar(26:40) : 0 (unused for now)


! real(kind=dbl) :: fpar(40)  components
! fpar(1) : sig - sample tilt angle around TD (degrees)
! fpar(2) : omega - sample tilt angle around RD (degrees)
! fpar(3) : EkeV - beam energy in keV
! fpar(4) : Ehistmin - minimum energy to consider in MC computation (keV)
! fpar(5) : Ebinsize - energy bin size (keV)
! fpar(6) : depthmax - maximum depth to consider in MC computation (nm)
! fpar(7) : depthstep - depth step size (nm)
! fpar(8) : sigstart - starting value of sample tilt (ECP mode)
! fpar(9) : sigend - end value of sample tilt (ECP mode)
! fpar(10): sigstep - tilt angle step size (ECP mode)
! parameters only used in the master pattern routine
! fpar(11) : dmin - minimum d-spacing to consider in master pattern computation
! fpar(12) : Bethe  c1 - Bethe potential cutoff parameter
! fpar(13) : Bethe  c2 - Bethe potential cutoff parameter
! fpar(14) : Bethe  c3 - Bethe potential cutoff parameter
! parameters only used in the EBSD pattern routine
! fpar(15): pcx - pattern center x
! fpar(16): pcy - pattern center y
! fpar(17): delta - scintillator pixel size
! fpar(18): thetac - detector tilt angle
! fpar(19): L - sample-scintillator distance
! fpar(20): bc - beam current
! fpar(21): dt - dwelltime
```

```
! fpar(22): gamma - gamma intensity scaling value
! fpar(23:40): 0 (unused for now)
```

All of the above parameters must be read by the C/C++ program from the Monte Carlo-master pattern file.

13.2.1 EMsoftCgetMCOpenCL

In addition to the ipar and fpar arrays, the following variables are passed to this routine (all parameters that have the qualifier INTENT(IN) are input parameters; the qualifier INTENT(OUT) implies data returned by the function):

```
integer(c_int32_t),PARAMETER  :: nipar=40
integer(c_int32_t),PARAMETER :: nfpar=40
integer(c_int32_t),INTENT(IN) :: ipar(nipar)
real(kind=sgl),INTENT(IN) :: fpar(nfpar)
real(kind=sgl),INTENT(IN) :: atompos(ipar(9),5) - from .xtal file
integer(kind=irg),INTENT(IN) :: atomtypes(ipar(9)) - from .xtal file
real(kind=sgl),INTENT(IN) :: latparm(6) - from .xtal file
integer(kind=irg),INTENT(OUT) :: accum_e(ipar(12),-ipar(1):ipar(1),-ipar(1):ipar(1)) - energy accumulator
integer(kind=irg),INTENT(OUT) :: accum_z(ipar(12),ipar(13),-ipar(16):ipar(16),-ipar(16):ipar(16)) - depth accumulat
TYPE(C_FUNPTR), INTENT(IN), VALUE :: cproc - progress callback function
integer(c_size_t),INTENT(IN), VALUE :: objAddress - object identifier from calling program
character(len=1),INTENT(IN) :: cancel - cancel computation flag
```

Note that f90 arrays can be defined with index ranges that span from negative to positive values; on the C/C++ side, the array should be defined starting at index 0.

13.2.2 EMsoftCgetEBSDmaster

In addition to the ipar and fpar arrays, the following variables are passed to this routine:

```
integer(c_int32_t),PARAMETER :: nipar=40
integer(c_int32_t),PARAMETER :: nfpar=40
integer(c_int32_t),INTENT(IN) :: ipar(nipar)
real(kind=sgl),INTENT(IN) :: fpar(nfpar)
real(kind=sgl),INTENT(IN) :: atompos(ipar(9),5) - from .xtal file
integer(kind=irg),INTENT(IN) :: atomtypes(ipar(9)) - from .xtal file
real(kind=sgl),INTENT(IN) :: latparm(6) - from .xtal file
integer(kind=irg),INTENT(IN) :: accum_z(ipar(12),ipar(13),-ipar(16):ipar(16),-ipar(16):ipar(16)) - depth accumulato
real(kind=sgl),INTENT(OUT) :: mLPNH(-ipar(17):ipar(17),-ipar(17):ipar(17),1:ipar(12),1:ipar(9))
- modified Lambert projection Northern Hemisphere
real(kind=sgl),INTENT(OUT) :: mLPSH(-ipar(17):ipar(17),-ipar(17):ipar(17),1:ipar(12),1:ipar(9))
- modified Lambert projection Southern Hemisphere
TYPE(C_FUNPTR), INTENT(IN), VALUE :: cproc  - progress callback function
integer(c_size_t),INTENT(IN), VALUE :: objAddress - object identifier from calling program
character(len=1),INTENT(IN) :: cancel - cancel computation flag
```

13.2.3 EMsoftCgetEBSDpatterns

In addition to the ipar and fpar arrays, the following variables are passed to this routine:

```
integer(c_int32_t),PARAMETER :: nipar=40
integer(c_int32_t),PARAMETER :: nfpar=40
integer(c_int32_t),PARAMETER :: nq=4
integer(c_int32_t),INTENT(IN) :: ipar(nipar)
real(kind=sgl),INTENT(IN) :: fpar(nfpar)
```

```
real(kind=sgl),INTENT(IN) :: quats(nq,ipar(21)) - quaternions for which to compute patterns
integer(c_int32_t),INTENT(IN) :: accum_e(ipar(12),-ipar(1):ipar(1),-ipar(1):ipar(1)) -energy accumulator
real(kind=sgl),INTENT(IN) :: mLPNH(-ipar(17):ipar(17), -ipar(17):ipar(17), ipar(12), ipar(9))
- modified Lambert projection Northern Hemisphere
real(kind=sgl),INTENT(IN) :: mLPSH(-ipar(17):ipar(17), -ipar(17):ipar(17), ipar(12), ipar(9))
- modified Lambert projection Southern Hemisphere
real(kind=sgl),INTENT(OUT) :: EBSDpattern(ipar(23),ipar(24),ipar(21)) - output EBSD patterns
TYPE(C_FUNPTR), INTENT(IN), VALUE :: cproc  - progress callback function
integer(c_size_t),INTENT(IN), VALUE :: objAddress - object identifier from calling program
character(len=1),INTENT(IN) :: cancel - cancel computation flag
```

Note that the value of ipar(25) encodes whether the quats array contains quaternions (ipar(25)=0) or Euler angle triplets in radians (ipar(25)=1). The quats array must have the first dimension equal to 4, regardless of the rotation angle type. This will likely be changed in a future version of the code, so that an arbitrary orientation representation can be passed to the routines.

13.2.4 EMsoftCgetECPpatterns

In addition to the ipar and fpar arrays, the following variables are passed to this routine:

```
integer(c_int32_t),PARAMETER :: nipar=40
integer(c_int32_t),PARAMETER :: nfpar=40
integer(c_int32_t),PARAMETER :: nq=4
integer(c_int32_t),INTENT(IN) :: ipar(nipar)
real(kind=sgl),INTENT(IN) :: fpar(nfpar)
real(kind=sgl),INTENT(IN) :: quats(nq,ipar(21)) - quaternions for which to compute patterns
integer(c_int32_t),INTENT(IN) :: accum_e(ipar(12),-ipar(1):ipar(1),-ipar(1):ipar(1)) -energy accumulator
real(kind=sgl),INTENT(IN) :: mLPNH(-ipar(17):ipar(17), -ipar(17):ipar(17), ipar(12), ipar(9))
- modified Lambert projection Northern Hemisphere
real(kind=sgl),INTENT(IN) :: mLPSH(-ipar(17):ipar(17), -ipar(17):ipar(17), ipar(12), ipar(9))
- modified Lambert projection Southern Hemisphere
real(kind=sgl),INTENT(OUT) :: EBSDpattern(ipar(23),ipar(24),ipar(21)) - output EBSD patterns
TYPE(C_FUNPTR), INTENT(IN), VALUE :: cproc  - progress callback function
integer(c_size_t),INTENT(IN), VALUE :: objAddress - object identifier from calling program
character(len=1),INTENT(IN) :: cancel - cancel computation flag
```

14 The IDL SEMDisplay.pro program

The SEMDisplay.pro program[9] is a graphical interface that provides the user with the ability to handle the output from the fortran-90 programs, EMMCOpenCL, EMMCfoil, EMEBSDmaster, EMECPmaster, EMTKDmaster, and EMKosselMaster. The main interface window is shown in Fig. 4, and consists of the following regions:

- Master pattern data file name;
- Information window: this shows program messages about file loading, changed parameters, etc.
- Button row: allows the user to load an EBSD/ECP/TKD/Kossel master file, which will also automatically load the corresponding MC file (except for the Kossel mode). The Define detector

[9] The EMsoft executable package comes with several virtual machine apps (VMAPPs), i.e., pre-compiled IDL routines that can be executed without an IDL license. These apps are provided without support but they do work on Mac OS X and Linux; further development will be needed for Windows. Note that the IDL source code for all apps is also available from the source code repository, but this will require an IDL license to be able to compile and run the codes.

Figure 4: *Main user interface for the SEMDisplay program.*

button will be described in more detail below. The user also has the option to create a log-file that will store all the entries shown also in the information window.

14.1 Monte Carlo and Master Pattern visualization interface

For EBSD, TKD, and ECP pattern simulations, this interface will show both Monte Carlo and master pattern data; for the Kossel mode, only the master pattern will be available. The interface controls available are slightly different for each case.

When the user selects the Display button next to the Master Pattern file name widget, the display widget shown in Fig. 5 will appear, with the Monte Carlo results on the left and the master pattern on the right. Each of the graphics areas has a separate set of min/max indicators as well as a Save button. Note that this widget has a pull-down menu next to the energy slider; initially, this menu will display the selection SUM all sites. For a crystal structure with more than one atom in the asymmetric unit, the pull-down menu will display a list of all atoms along with the square of the atomic number (recall that the back-scattering cross section varies approximately as the square of the atomic number). When the user selects one of the atoms from the asymmetric unit, the graphics window on the right will display only the contribution from those sites (the one in the asymmetric unit and all equivalent positions). The SUM all sites option displays the sum over all atomic sites in the unit cell. Separating the BSE contributions by element (asymmetric unit site) is useful, because it reveals which pattern is the dominant one. This will obviously depend on the atomic number for each site; for crystal structures with low elemental contrast[10], the net master pattern may have approximately equal contributions from two or more elements. For structures with high elemental contrast, usually one element (the one with the largest atomic number) will

[10] We refer to the difference between the atomic numbers as "elemental contrast;" in $BaTiO_3$, for instance, we have $Z^2_{Ba} = 3,136$, $Z^2_{Ti} = 484$, and $Z^2_O = 3 \times 64 = 192$, so that the EBSD pattern is dominated by the contributions from Ba. In $CaTiO_3$, on the other hand, we have $Z^2_{Ca} = 400$, so the contrast between Ca and Ti is low, and the pattern will have contributions from both elements. In both cases, the O contribution would be rather small.

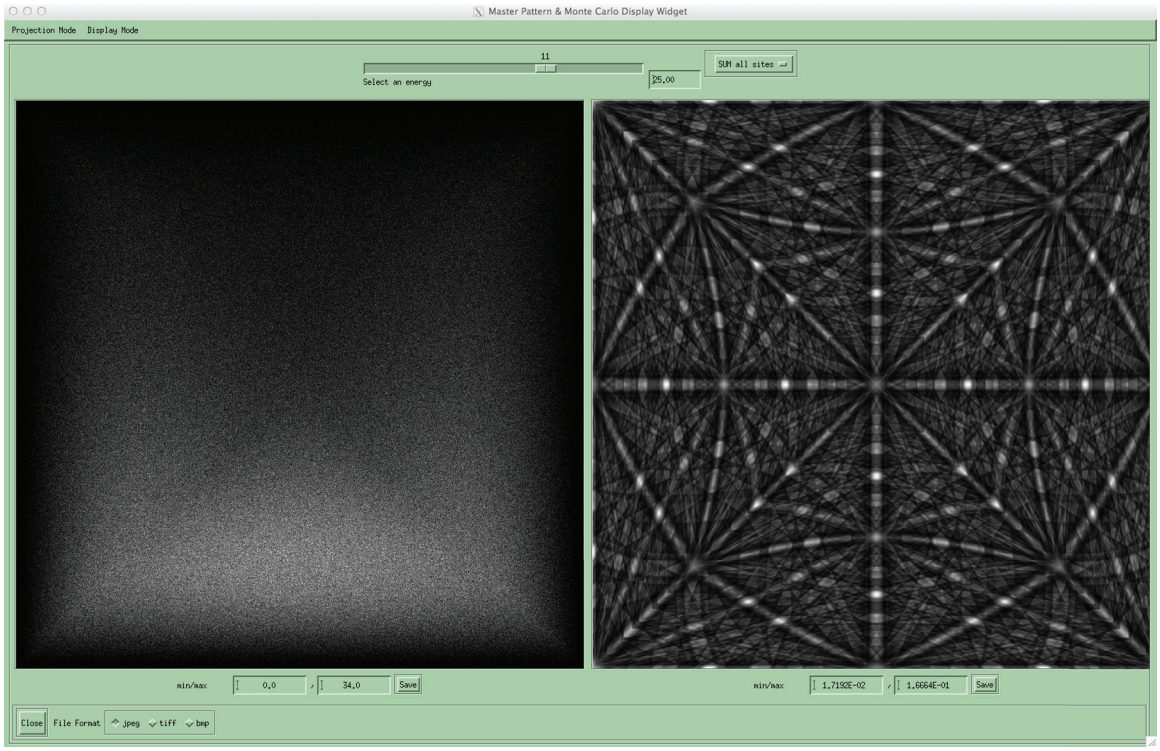

Figure 5: *Monte Carlo and Master Pattern interface.*

dominate the EBSD pattern. For ECP master patterns, the interface functions in the same way, but there are fewer display options available.

- Projection Mode menu: this menu has three items, Lambert [square], Lambert [circle], and Stereographic P., corresponding to the Lambert projection explained in section 5.1, the regular equal-area Lambert projection, and the equal-angle stereographic projection, respectively. The data in the Monte Carlo file is stored in the square Lambert projection format.

- Display Mode menu: there are three entries in this menu, Individual Energy Bin, Simple Energy Sum, and Simple Energy Sum RGB.
  - Individual Energy Bin: in this display mode, the graphics window will show the spatial distribution of BSEs for a single energy value. The energy can be selected using the slides above the graphics window. Note that the label above the slider indicates the sequential number of the energy in the data file, not the actual energy which is shown in the small text box to the right once the slider is released.[11]
  - Simple Energy Sum: in this mode, the slider bar is grayed out, and the graphics window will show the sum pattern over all energy values.
  - Simple Energy Sum RGB: this is similar to the previous mode, except that a color code (from Green to Blue to Red) is used to indicate the various energy levels.

Note that all three Display Modes work for either of the Projection Modes; in a later version of the program, stereographic projection mode and 3D sphere mode will be added to the Projection Mode

[11] This is not an optimal arrangement and it will be changed in a later version.

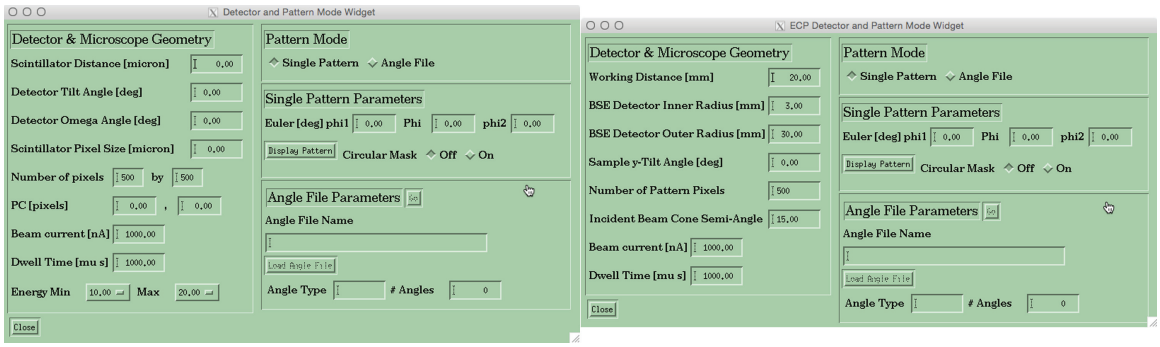

Figure 6: *Detector interfaces for EBSD/TKD and ECP modes.*

menu. The pattern that is displayed in the graphics window can be saved by pressing the Save button; the file format must be set first using the File Format selector (jpeg, tiff, or bmp). The minimum and maximum value of the displayed array are also shown below the image region.

If the Monte Carlo data was generated with the bse1 option for the program mode, then the energy slider will be replaced by a beam tilt slider. For the Kossel master pattern, the slider controls the sample depth for which the pattern was generated.

14.2 Detector interface

Clicking on the Define detector button in the main program widget generates a new widget with several groups of user-definable parameters, as shown in Fig. 6. Depending on the master file type (EBSD or ECP) the interface offers different users adjustable parameters to define the detector geometry and sample orientation. On the left, the user can define the detector geometry (scintillator-sample distance, detector tilt, scintillator pixel size, number of pixels, and pattern center location), the incident beam current, and the beam dwell time. The energy range over which the EBSD patterns should be integrated can be set by means of the Min and Max droplists.

On the right side of the interface there are two display modes for EBSD patterns: *Single Pattern* and *Angle File*.

- Single Pattern Parameters: The user can set an Euler angle triplet for which the EBSP should be displayed (using all the detector and microscope parameters set in other parts of the interface). Clicking on the Display Pattern button will then bring up another interface that displays the actual EBSP (Fig. 7).

- Angle File Parameters: In this mode, the user selects an existing angle file (format described below); once the file has been selected, the display will list the angle type (Euler or quaternion) and the number of angles in the file. The Go button next to the title of this region will then be highlighted. Clicking on this button will execute a dynamically linked library routine using all the parameters that are currently set, and an EBSP or ECP will be computed for each entry in the angle file. Once the computation is complete, an interface similar to that for the individual pattern will pop up (see Fig. 8); the main difference is that in this interface, there is no possibility of modifying the imaging parameters; the program will use the parameters determined by the user in the *Single Pattern* mode. The display interface does offer the possibility of stepping through the entries in the angle file and displaying individual EBSPs or ECPs along with their Euler angle triplets or quaternions. There is also an option to generate separate output files for individual patterns or for the whole set.

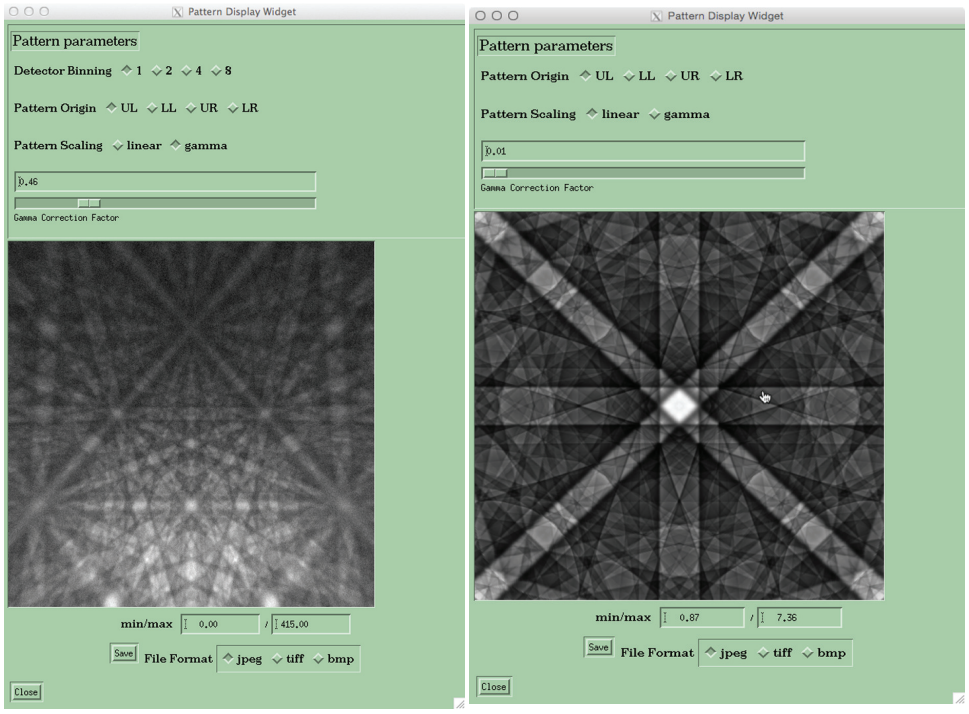

Figure 7: *EBSP/TKD/ECP display interface (single pattern mode).*

The format for the angle file is a simple text file, structured as follows:

```
eu
15
34.5 25.9 183.0
22.9 65.3 112.0
...
```

On the first line we have a two-character angle type identifier ('eu' for Euler angles, 'qu' for quaternions); the second line lists the number of angle entries in the file, and each subsequent line lists either the three Euler angles (in degrees, without commas) or the four quaternion components.

The interfaces for ECP and Kossel modes are conceptually similar to that for the EBSD mode, but the detector geometrical parameters are different for each case. The user is encouraged to experiment with the controls to see what effect each has on the simulated pattern.

It should be noted that this program stores all variables in a preferences file located in the .config/EMsoft folder in the user's home folder. This is useful since the preferences are written each time the program is quit, and read whenever the program is started. The preferences file can also be edited using any text editor; variable names should not be changed.

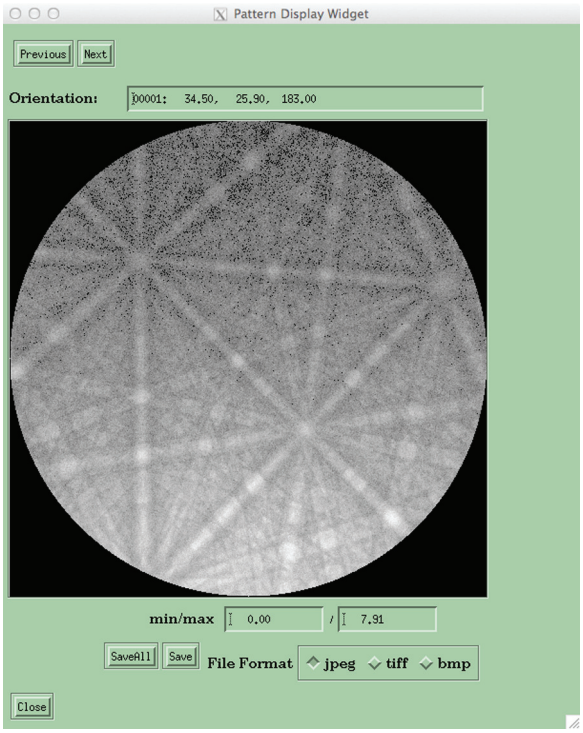

Figure 8: *EBSP display interface (angle file mode); a similar interface is available for the ECP mode.*

15 A worked example

In this final section, we describe a simple example of an EBSD pattern simulation for pure aluminum. We begin by using the EMmkxtal program to enter the following data:

```
crystal system ---> 1
a [nm] = 0.405
Enter space group number : 225
Atomic number : 13
Fractional coordinates etc. : 0.0, 0.0, 0.0, 1.0, 0.00746
Another atom ? (y/n)  n
Enter output file name : Al.xtal
```

The room temperature Debye-Waller factor above comes from a paper by Peng et al.[12]

The EBSD pattern simulation then consists of three consecutive program runs, described in the following subsections.

15.1 Monte Carlo simulation

In the Examples/EBSDPatterns folder you will find the following namelist file (EMMCexample.nml):

Copy this file into your working folder inside the EMdatapathname path, and update the dataname in the name list file to reflect the correct relative path.

This input file will produce an output file with energy statistics for the range $[10, 25]$ keV in 1 keV steps on an 501×501 square Lambert grid. The program can be executed in your work folder by the following command line:

```
EMMCOpenCL EMMCexample.nml
```

Note that your computer's screen updates may become extremely sluggish in response time while running this program on the GPU (unless you have a separate GPU from the one that drives the screen); this is normal, and you will just have to be really patient. After about 12 minutes,[13] this run produced an Al-master-25kV.h5 HDF5 file of size $32,909,120$ bytes, along with the following output:

```
 EMsoft version 3_1_beta, Copyright (C) 2001-2016 Marc De Graef Research Group/CMU
 EMsoft comes with ABSOLUTELY NO WARRANTY.
 This is free software, and you are welcome to redistribute it
 under certain conditions; see License.txt file for details.

 Program name : EMMCOpenCL.f90
 Purpose      : Monte Carlo backscattered electron simulation
 Platform     : Darwin
```

[12] L.-M. Peng, G. Ren, S.L. Dudarev, and M.J. Whelan, "Debye-Waller factors and absorptive scattering factors of elemental crystals," Acta Cryst. (1996) A52, 456-470.

[13] iMac 4 GHz Intel Core i7 with 8 cores, 32Gb RAM and AMD Radeon R9 M395 Compute Engine GPU card; note that this is a medium range GPU. On more modern graphics cards, a 2 billion electron Monte Carlo run make take as little as five minutes (Tesla K80). On a laptop, a similar run make take more than an hour.

```
 Dec  3 2016  12:23:55.840 PM

-->Crystal Structure Information<--
  a [nm]              :  0.40500
  b [nm]              :  0.40500
  c [nm]              :  0.40500
  alpha [deg]         : 90.00000
  beta  [deg]         : 90.00000
  gamma [deg]         : 90.00000
  Volume [nm^3]       :  0.06643013
  Space group #       : 225
  Space group symbol : F m 3 m
  Generator String   :16aODDaDODbOOOcOOOdOOOeOOOO
   Structure is centrosymmetric

  Number of asymmetric atom positions           1
  General position / atomic number / multiplicity :   1/13/  4 (Al)
   Equivalent positions  (x y z  occ  DWF)
         >    0.00000,  0.00000,  0.00000,  1.00000,  0.00746
         >    0.00000,  0.50000,  0.50000,  1.00000,  0.00746
         >    0.50000,  0.00000,  0.50000,  1.00000,  0.00746
         >    0.50000,  0.50000,  0.00000,  1.00000,  0.00746

Mean inner potential [V] 0.1777E+02
 Wavelength corrected for refraction
Relativistic correction factor [gamma] 0.1049E+01
Relativistic Accelerating Potential [V] 0.2563E+05
Electron Wavelength [nm] 0.7661E-02
Interaction constant [V nm]^(-1) 0.1678E-01
 Range of reflections along a*, b* and c* =           9           9           9
 Normal absorption length [nm] =   155.646851

Generating Fourier coefficient lookup table ...Done

Density, avZ, avA =   2.69781  13.00000,  26.98154
 OpenCL source file set to : EMMC.cl
Kernel source length (characters) :                13628

Program Build Successful... Creating kernel
Monte Carlo mode set to full. Performing full calculation...

 Total number of electrons incident =      131072000
 Number of BSE electrons =       60743958
 Total number of electrons incident =      262144000
 Number of BSE electrons =      121482345
 Total number of electrons incident =      393216000
 Number of BSE electrons =      182219538
```

```
.......
 Total number of electrons incident =     1966080000
 Number of BSE electrons =      911089781
Total number of incident electrons =     2000000000
Total number of BSE electrons =      926887393
Backscatter yield =       0.463444
Total execution time [s] =                   709
```

The last lines of output indicate that the overall BSE yield is 46.3%; this is not the yield on the detector, but the total yield in the Northern hemisphere with respect to the sample surface (i.e., all BSEs are counted, regardless of the direction in which they escape the surface).

15.2 EBSD master pattern simulation

The next step is the computation of the EBSD master pattern, using the EMEBSDmaster program. The example input file EMEBSDMexample.nml contains the following entries:

One should also set the Bethe potential parameters as described in an earlier section. The program is executed as follows:

```
EMEBSDmaster EMEBSDMexample.nml
```

and, after about 5 minutes on the same system used in the previous section, increases the size of the file Al-master-25kV.h5 to 197, 182, 732 bytes. A partial program output is as follows:

```
 EMsoft version 3_1_beta, Copyright (C) 2001-2016 Marc De Graef Research Group/CMU
 EMsoft comes with ABSOLUTELY NO WARRANTY.
 This is free software, and you are welcome to redistribute it
 under certain conditions; see License.txt file for details.

 Program name : EMEBSDmaster.f90
 Purpose      : EBSD Energy-dependent Master Pattern Simulation
 Platform     : Darwin

 Dec  3 2016  12:39:52.642 PM

-->Crystal Structure Information<--
  a [nm]             :  0.40500
  b [nm]             :  0.40500
  c [nm]             :  0.40500
  alpha [deg]        : 90.00000
  beta  [deg]        : 90.00000
  gamma [deg]        : 90.00000
  Volume [nm^3]      :  0.06643013
  Space group #      : 225
  Space group symbol : F m 3 m
  Generator String   :16aODDaDODbOOOcOOOdOOOeOOOO
   Structure is centrosymmetric
```

```
  Number of asymmetric atom positions           1
  General position / atomic number / multiplicity :   1/13/  4 (Al)
   Equivalent positions  (x y z  occ  DWF)
         >    0.00000,  0.00000,  0.00000,  1.00000,  0.00746
         >    0.00000,  0.50000,  0.50000,  1.00000,  0.00746
         >    0.50000,  0.00000,  0.50000,  1.00000,  0.00746
         >    0.50000,  0.50000,  0.00000,  1.00000,  0.00746

Mean inner potential [V] 0.1777E+02
 Wavelength corrected for refraction
Relativistic correction factor [gamma] 0.1049E+01
Relativistic Accelerating Potential [V] 0.2563E+05
Electron Wavelength [nm] 0.7661E-02
Interaction constant [V nm]^(-1) 0.1678E-01
 Range of reflections along a*, b* and c* =           9           9           9
 Normal absorption length [nm] =   155.646851

Generating Fourier coefficient lookup table ...Done

 -> completed reading MC/Al-master-25kV.h5

 Range of reflections along a*, b* and c* =           9           9           9

Generating Sgh coefficient lookup table ...Done

Starting computation for energy bin  16; energy [keV] = 25.00

# independent beam directions to be considered =   80602
 Attempting to set number of threads to   6
  completed beam direction    5000
  completed beam direction   10000
  completed beam direction   15000
  completed beam direction   20000
  completed beam direction   25000
  ... many lines removed ...
  completed beam direction   75000
  completed beam direction   77500
  completed beam direction   80000
 -> Average number of strong reflections =   17
 -> Average number of weak reflections   =    5
Final data stored in file MC/Al-master-25kV.h5
```

Note that this program computes a master pattern for each energy bin produced by the Monte Carlo program; in the present case, there are 16 energy bins, so the program executes the full dynamical computation that many times. Output is stored on the square Lambert projection, and can be read by the EMEBSD program as well as the standard HDFView program, and the IDL visualization interface SEMDisplay.pro.

15.3 EBSD pattern simulation

When the Monte Carlo and master pattern files have been created, then one can start up the SEMDisplay visualization program, or, if all imaging parameters are known, one can run the EMEBSD program directly with a namelist file. Consider the EMEBSDexample.nml file in the EMsoft/Examples/EBSDpatterns folder:

We leave it up to the reader to experiment with the namelist file approach. For the IDL visualization interface, start up the program, then click on the Load master file button to load both the master file and the Monte Carlo file; using the Display button one can then examine both the electron distribution vs. energy and the master patterns. Use the Define detector button to bring up the detector interface; select single pattern mode and fill in the detector parameters. Select an output file, then enter an Euler angle triplet, click on the Display pattern button and you will see the corresponding EBSP. Set the intensity scaling to gamma, and use the slider to select an appropriate pattern contrast.

To get a feel for how the various detector parameters affect the EBSP, you should try changing the parameters one by one. For instance, decreasing either the beam current or the dwell time, the patterns will become more noisy. Note that these patterns do not include the point spread function of the detector; that option will become available in a next release of the software.

15.4 Accessing the HDF5 data files

The HDF5 files in the EMsoft package all have the same internal structure. You can use the HDFView program to access and/or export any variables. Fig. 9 shows a screenshot from a fully expanded internal structure of a Si-master-20kV.h5 file generated by the EMMCOpenCL and EMEBSDmaster programs. The main components of the file are:

- EMData: This is where the actual program output data are stored in separate groups of numbers, strings, or arrays.
- EMheader: This contains information on the version, data, time, etc...
- NMLfiles: contains verbatim all the namelist files that were used by this program.
- NMLparameters: contains a subgroup for each namelist file, with all the entries of the file parsed out and assigned to individual variables.

The reason for including the namelist files explicitly is so that you will always have a record of all the parameters used for this particular run.

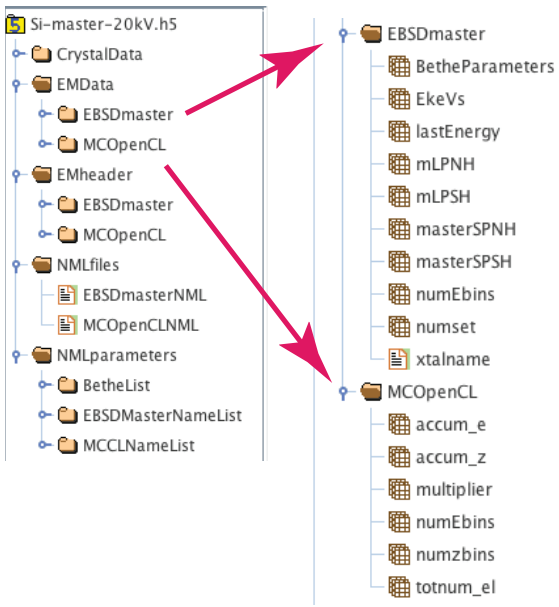

Figure 9: *Screen shot of the HDFView program with an example Monte Carlo output file. The individual entries on the left can be displayed as images or arrays by right-clicking on them and selecting "Open As".*