An algorithm for generating Orientation Imaging Maps from LAMMPS output

Sebastian Hütter (sebastian.huetter@st.ovgu.de)

Abstract

An algorithm is presented to generate crystal orientation information for cubic lattices from LAMMPS Dump Files.

Introduction

When working with lattice defects in metals (i.e. stacking faults, twinning), Orientation Imaging Microscopy (or Micrography, OIM) via Electron Backscatter Diffraction (EBSD) patterns can be used to visualise crystal orientation across a sample surface. Orientation changes then point to the location of defects and can used to identify the defect in question.

The same tool would be helpful when working with LAMMPS [1, 2] Dump Files, especially since any plane can be chosen as a "cutting plane", allowing for more complete overview than a single cut could provide on a physical sample. This work describes an algorithm to compute the neccessary information for cubic crystals, in particular base centered cubic (bcc) structure.

Orientation Imaging Micrographs

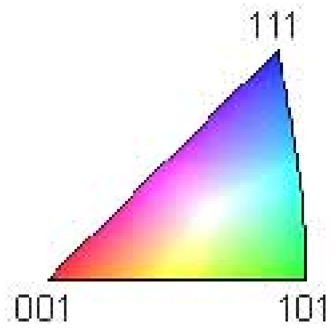

Figure 1. Colour map conventionally used in OIM images (Source: [3]).

Orientation Imaging Micrographs in a laboratory environment are created by capturing the EBSD diffraction pattern and transforming it into coordinate vectors describing the local crystal orientation. These vectors can be projected into an Inverse Pole Figure (IPF), which enables the assignment of colours to vector orientations. Projections are generally placed inside the so-called Standard Triangle of the IPF, giving the assignment in Figure 1. Colours then represent the crystal's local axis that is equivalent to the normal vector of the sample surface. A colour of red would identify a crystal that is axis-aligned to the cutting surface, for example.

Algorithm Description

Generating such maps from LAMMPS Dump Files is based on the mathematical principles of the projection involved behind the experimental routine, and yields the same kind of result.

When working with LAMMPS Dump Files, one must be aware of two main factors: The files are large, often containing data on tens of millions of atoms, each on a line in a text file. There is an obvious need for some form of efficient in-memory representation of the relevant data. Second, no information on inter-atomic relations is given (at least in metals simulations; chemical bounds could very well be simulatied with LAMMPS). Essentially, only X,Y,Z coordinates for each atom are known. From these factors it follows that the file is first read into memory and the atom coordinates are saved in a suitable in-memory representation, essentially an array of location vectors. To facilitate faster neighbour lookups, a three-dimensional binned array is generated from this list, allowing for the quick exclusion of atoms that cannot be neighbours within a cutoff radius.

The actual algorithm is then applied to each atom individually, the results are gathered and written back (along with the original information) to an output file.

Steps 1: Identify Useful Neighbours

First, all neighbour atoms must be collected that lie in cardinal directions from the current/center atom. In a simple cubic (sc) lattice, this means the six nearest neighbours, the six second-nearest neighbours in bcc and the six nearest or third-nearest atoms in fcc lattice structures. To identify the crystal structure, all neighbour atoms within a radius from the current atom are gathered and sorted by their distance. Checking the number of neighbours found and grouping them by distance allows verification against known ratios in the respective lattices and is an easy and fast check for lattice type. This step yields six atoms which will be used in the next step.

Step 2: Compute Local Coordinate Systems

From 6 points in space around a seventh central point, one can form 48 different coordinate systems: one might choose six "up"-vectors, then four associated "front" vectors and two "side" vectors. Since choosing the side vector means choosing a left-hand or right-hand system and only right-hand systems are of interest, the third axis can be computed from the first two by using a cross product, leaving 24 coordinate systems. These can be made the basis vectors of a rotation matrix:

$$\begin{aligned}\mathbf{x} &= (x_1, x_2, x_3)\\ \mathbf{y} &= (y_1, y_2, y_3)\\ \mathbf{z} &= (z_1, z_2, z_3) = \mathbf{x} \times \mathbf{y}\\ M &= \begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{pmatrix}\end{aligned}$$

This matrix M now describes a rotation from the fixed reference system (the unity system formed by $\mathbf{e_x}, \mathbf{e_y}, \mathbf{e_z}$) into a local coordinate system aligned with the current atom's surroundings.

The following steps are simply applied to each of the 24 possible systems in sequence, until one yields a verifiable result.

Step 3: Identify Reference Vector

When looking back at the original definition, we can see that OIM colours describe the direction normal to the sample surface in terms of the local coordinate system. Since we already have a transformation matrix, using $\mathbf{v_{ref}}$ as the reference direction, this vector can be easily computed as

$$\mathbf{p} = M^{-1} \cdot \mathbf{v_{ref}}$$

Step 4: Decompose Vector Into Parts

This vector is now decomposed into a linear combination of the three base vector forms at the corners of the Standard Triangle which are $\langle 1,0,0 \rangle$, $\langle 1,1,0 \rangle$ and $\langle 1,1,1 \rangle$. Since possible directions of one form are always identical to each other by rotations of exactly 90° along one or two axes and this algorithm iterates over all 24 orientations, the actual direction is irrelevant as long as the correct form is chosen. This means that the following colour mapping to values of **r**ed, **g**reen and **b**lue may be used:

$$(x, y, z) = r \cdot (0,0,1) + g \cdot (0,1,1) + b \cdot (1,1,1)$$

Of all the orientations examined by the algorithm, only one will yield a vector which can be projected into the standard triangle and thus has all three components between 0.0 and 1.0. A certain value of excess outside these limits needs to be allowed, since the transformation matrix will have been not a pure rotation matrix but contain a small shear part that has little to no influence on the direction represented, but can lead to values as low as -0.05 for one colour component.

Once a passing vector has been identified, processing for that atom is done and the next can be examined. It would at this point be possible to do other computations based on the orientation matrix or save it in the atom data.

Applications

The resulting information may be used most directly to create OIM visualisations of simulation data (cf. Figure 2).

Since the detection of crystal orientations does not work on grain boundaries (because of the very fact that the lattice ends there), picking out the atoms that have no vector attached to them can be useful in detecting grain boundaries as well as defect boundaries.

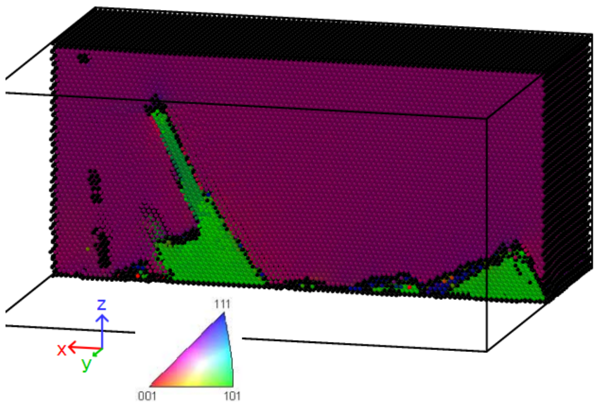

Figure 2. OIM image generated from a simulation of shock-induced twinning.

Implementation

An implementation has been created in the shape of the `oim`-tool included in the mklattice collection of programs related to LAMMPS.

Without further optimizations, this implementation is able to process 500k Atoms in about 10 seconds and scales linearly in time (and memory) required: neighbour binning leads to a constant number of atoms that have to be checked for proximity for each atom, and there are at most 24 coordinate systems to verify. Parallelisation across multiple CPU cores is relatively straightforward, since there is no information to share between atoms.

References

1. Plimpton, S.: Fast Parallel Algorithms for Short-Range Molecular Dynamics. In: J Comp Phys, 117 (1995) pp. 1–19

2. LAMMPS WWW Site. URL http://lammps.sandia.gov/, (2014-02-22)

3. Stanford Nanocharacterization Laboratory: Introduction to Orientation Imaging Microscopy. URL http://web.stanford.edu/group/snl/SEM/OIMIntro.htm, (2014-02-2)