Topological Extraction and Tracking of Defects in Crystal Structures

Sebastian Grottel[1], Carlos A. Dietrich[2], João L. D. Comba[2], and Thomas Ertl[1]

[1] VISUS - Universität Stuttgart, Germany
[2] Instituto de Informática - UFRGS, Brazil

Abstract. Interfaces between materials with different mechanical properties play an important role in technical applications. Nowadays molecular dynamic simulations are used to observe the behavior of such compound materials at the atomic level. Due to different atom crystal sizes, dislocations in the atom crystal structure occur once external forces are applied, and it has been observed that studying the change of these dislocations can provide further understanding of macroscopic attributes like elasticity and plasticity. Standard visualization techniques such as the rendering of individual atoms work for 2D data or sectional views; however, visualizing dislocations in 3D using such methods usually fails due to occlusion and clutter. In this work we propose to extract and visualize the structure of dislocations, which summarizes the commonly employed filtered atomistic representations into a concise representation, similar to the visualizations used with data from dislocation dynamics. The benefits of our approach are much clearer images still retaining all relevant data and easier visual tracking of the topological changes of these structures over time.

1 Introduction

Compound materials are used in several applications, and it is of uttermost importance to understand their properties and how they react under external forces. An important aspect to consider in this analysis is the appearance of defects in their atomic structure [1], which have a direct relation to material properties such as resistance, conductance, strength, etc. For this purpose, Molecular Dynamics simulations (MD) are used, which reproduce the behavior of atoms and molecules for a period of time. Such simulations generate a great number of components, which lead to a complex analysis. Often only statistical measurements are evaluated.

For the analysis of defects, however, the analysis can be restricted to a subset of this data. Given that the atomic structure of compound materials in normal conditions correspond to crystal lattice structures, it suffices to consider those atoms in which their neighborhood deviate from the regularity established by this lattice. The atoms that have irregular structure align in 1-D and 2-D structures called *dislocations* (a 1-D defect) and *stacking faults* (a 2-D defect).

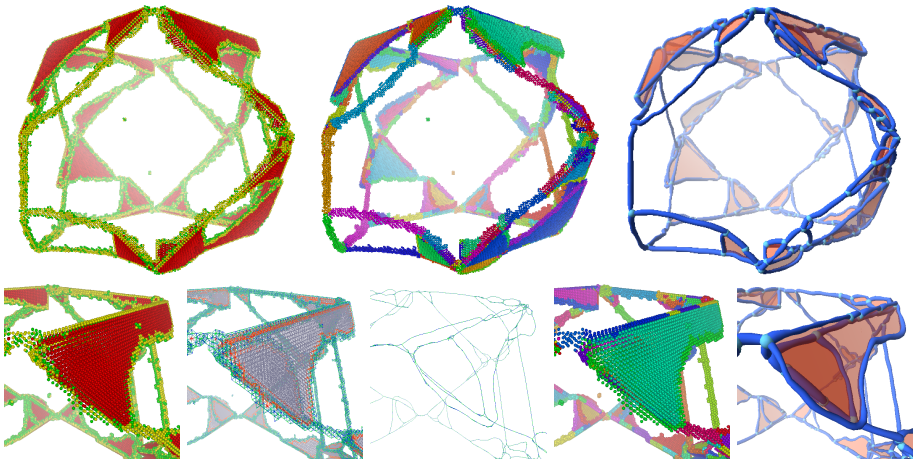


Fig. 1. Upper row shows full view of a data set, from left to right: original atom data; after segmentation; extracted defects. Lower row shows zoomed detail view, from left to right: original atom data; relative neighborhood graph; graph after simplification; segmented atom data; extracted defects.

The topological structure induced by such defects and their interplay helps evaluate the material properties. The identification of dislocations and stacking faults structures from atomistic simulations resemble techniques for skeleton extraction from discrete data, since only particle positions are given. In this paper we present an approach that extracts the topological structure of dislocations and stacking faults from atom positions, and tracks them throughout the simulation. Our proposal is composed of a segmentation algorithm based on simplification of a neighborhood graph that allows defects to be identified and tracked over time. In particular, we identify the formation of junctions among dislocations and stacking faults. In Figure 1 we show different steps of our algorithm using a a molecular dynamic simulation of a block of compound material underlying a stretching force. The two basic materials are Ni and Ni_3Al. The data set originally contains about 1,200,000 atoms. However, because of a first filtering which will be described in section 3 the files loaded only contain between 13,000 and 87,000 atoms, depending on the scene complexity.

2 Background Material and Related Work

Several papers in the literature of molecular dynamics discuss issues regarding the evaluation of material properties under external forces, and a good introductory reference is the book by Bulatov [1]. Since the atomic structure of many solid materials in normal conditions correspond to crystal lattice structures, defects are observed when changes to the lattice structure occur. Common lattices include the body-centered cubic (BCC), face-centered cubic (FCC) or hexagonal-

closed packed (HCP). Observing changes to this structure can be done by inspecting the 12-atom neighborhood of a single atom. In the cases of the FCC and HCP lattices an atom has six neighbors placed in a hexagonal pattern over a plane, plus two sets of three points forming a tetrahedral above and below the atom (see Figure 2). FCC and HCP differ with regard to the orientation of these tetrahedra (opposed for FCC, same for HCP).

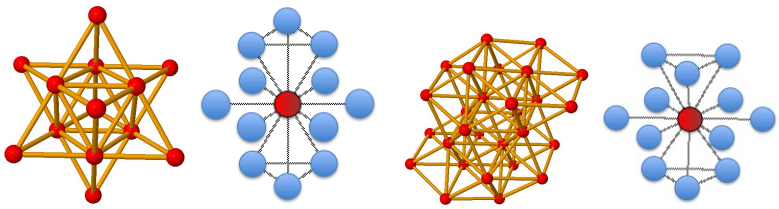


Fig. 2. FCC (a) and HCP (b) lattices and 12-atom neighborhood of a given atom.

Changes in the regularity, symmetry and ordering of this neighborhood create topological defects on the atomic structure [2]. Atoms with irregular neighborhood align in 1-D (dislocations) defects that surround 2-D defects (stacking faults). Geometric measures give one way to evaluate these defects, such as the Burgers vector [1] that represents the magnitude and direction of the lattice distortion of a given dislocation. Several papers discuss how such geometric measures can be used to track the motion of dislocations. For instance, Schall et al has a sequence of papers [3, 4] that discuss ways to track dislocation in colloidal crystals, and techniques to visualize the distribution of Burgers vectors (called Nye tensors). Hartley [5] also discuss a similar approach that measures Nye tensor distributions. Topological analysis of the interplay of dislocations is more elaborate if done at the atomistic level, and therefore analysis on a meso-scale representation is often used. Dislocation dynamics is an example of a simulation running in this meso-scale, and the paper by Lipowsky et al discuss a way to directly track and visualize dislocation in grain-boundary scars [6].

The work of Bulatov [7–9] suggests tracking topological features of dislocation, such as junctions or multi-junctions(common places of three or more dislocations). We quote here an important observation made by Bulatov in [8]: "In large-scale dislocation dynamics simulations, multi-junctions present very strong, nearly indestructible, obstacles to dislocation motion and furnish new sources for dislocation multiplication, thereby playing an essential role in the evolution of dislocation microstructure and strength of deforming crystals simulations". Our proposal is based on extracting a skeleton-like structures from the discrete data given by atomistic simulations, which shows such features like dislocations and junctions of dislocations.

There is a vast literature on extracting skeleton structures, and a good starting point is the survey presented in [10]. In their paper, skeletonization algorithms are classified in four major classes: thinning and boundary propagation, distance-field based, geometric methods and general-field functions. Our skeletonization algorithm works over discrete point sets and resemble a thinning and boundary propagation algorithm, although no implicit boundary is formed, but a neighborhood graph [11,12] that defines atom proximity and allows thinning contraction operations. This is also similar to thinning algorithms in distance-field methods, with respect to the creation of the neighborhood graph, but different from these methods since no distance field is used. Geometric methods are based on proximity structures, such as the Voronoi Diagram, or structures derived from Morse Theory, such as Reeb Graphs or Contour Trees [13–15]. General field methods such as potential field functions are used in a similar application described in [16]. In their work, crystal dislocation data is given as a potential field function, and creates a Morse-Smale complex to evaluate the structure of dislocations.

Several possibilities for controlling contraction operations can be defined, and often smoothing procedures are employed [17]. In this work we use a simple mass-spring contraction based approach, that helps in the creation of a simplified version of the neighborhood graph (described in Section 3.1).

3 Structure Defects Extraction

The visualization for crystal structure defects in atomistic simulations is similar to data from dislocation dynamics (DD) simulations. Unlike their data, which has a clean representation for the structure of the data that allows easy tracking of individual dislocations over time, in atomistic simulations there is no continuous or clean description of the dislocations. Therefore, an extraction based on the atomistic data provided by the molecular dynamic simulations is necessary. This data consists of an unrelated list of atoms A, each storing its position, an unique identifier number and a lattice classification based on the nearest neighbors of the atom accordingly to [18]. This classification is used for a first filtering of the atom. Atoms of class $c_a = 0$ (with c_a being the class of the atom $a \in A$) form a clean FCC lattice with their nearest neighbors, of class $c_a = 1$ form a HPC lattice. Atoms of class $c_a = 2$ form neither a FCC nor a HPC lattice but have 12 closest neighbors. Finally, atoms of class $c_a = 3$ do not have 12 neighbors. Since we are interested in only visualizing the crystal defects, we can completely omit atoms of class $c_a = 0$ (which are most: more then 90 percent). These atoms are not written into the data files to keep their sizes small. Switches from FCC to HPC indicate stacking faults while the other two classes form dislocations (the classification can be seen in all images showing the original data, e. g. figure 1, most left images: red atoms are $c_a = 0$, green atoms are $c_a = 1$, and yellow atoms are $c_a = 2$). Using this lattice classification for identifying crystal defects is much cheaper then calculating burgers vectors for all atoms.

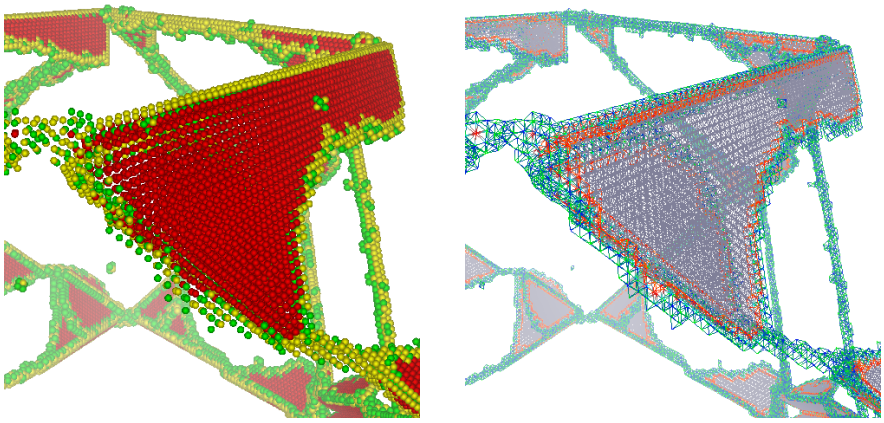


Fig. 3. Construction of the neighborhood graph (right) from the original atom data (left). Edges are color-coded: green/blue edges are from E, red/orange edges are from F, and gray edges are from G.

However, to extract the topological structure, we first need to define neighborhood information between atoms. We create a neighborhood graph using a cutoff radius automatically chosen, based on the mean distance between two atoms in the data set as shown in figure 3. For the extraction of the topology we propose to simplify this graph while keeping relations between graph nodes and atoms from the original data. We therefore define N to be the list of all nodes in the graph, where each node $n \in N$ stores a set of atom indices $a_i \in n$ referencing all atoms which are represented by this node. The initial graph creates one node for each atom ($n_i = \{a_i\}$ with $n_i \in N$ and $a_i \in A$) and places the node on the position of that atom $\overrightarrow{p}(n_i) = \overrightarrow{p}(a_i)$. E, F, and G are the lists of edges in the graph, where E holds edges connecting two nodes which are initialized with two atoms a_1, a_2 with $c_{a_1} > 1 \wedge c_{a_2} > 1$ (shown in green), G holds edges connecting a_3, a_4 with $c_{a_3} = 1 \wedge c_{a_4} = 1$ (shown in gray), and F holds all remaining edges of the neighborhood graph (shown in red). Note that the edges in E form dislocations, the edges in G form stacking faults, and the edges in F separate both. Based on this data structure we perform several simplifications, presented in the following section.

3.1 Graph Contraction

The first step of the graph simplification a contraction implemented using a simple mass-spring system. Each node n_i manages a speed vector v_i, which is initialized to zero, and a mass, which is modeled by a simple attenuation of v_i. All edges of E and F are considered as springs, where edges $e \in E$ try to collapse to length of zero and edges $f \in F$ try to keep their initial length. Edges from G and all nodes which are connected to at least one edge $g \in G$ are not considered in this calculation step. Also only nodes connected to at least one edge $e \in E$

can be moved. All other nodes are fixed. This way we can ensure that we do not collapse dislocations cycling around a stacking fault into a single position.

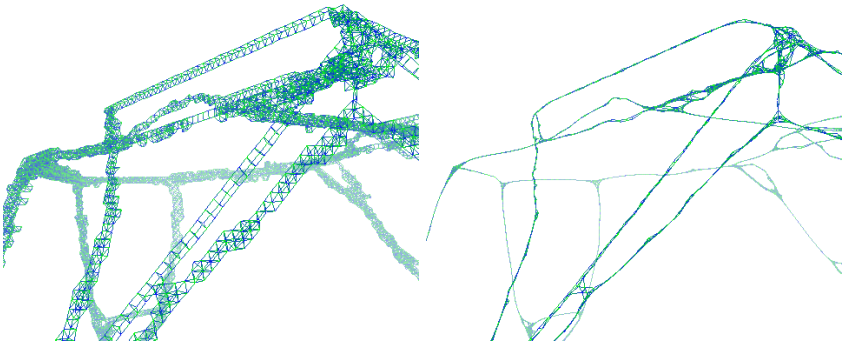


Fig. 4. Left image: the initial neighborhood graph (only edges from E are shown); right image: graph after the contraction of the mass-spring system (50 Iterations). Note the *unclean*, *thicker* regions near junctions.

In addition to simply moving the nodes in the graph we collapse edges which get too short. The corresponding threshold is based on the neighborhood radius used to initially build up the graph (in our examples we used a collapsing threshold which is half of the neighborhood radius). When collapsing the edge $e_i \in E$ we combine all attributes of the two original nodes $n_{e_i,1}$ and $n_{e_i,2}$ (e. g. by incorporating all attributes of node $n_{e_i,2}$ into $n_{e_i,1}$): $n_{e_i,1} = n_{e_i,1} \bigcup n_{e_i,2}$, $p(n_{e_i,1}) = (|n_{e_i,1}| + |n_{e_i,2}|)^{-1}(|n_{e_i,1}|p(n_{e_i,1}) + |n_{e_i,2}|p(n_{e_i,2}))$ (and all other attributes accordingly). Edges connected to $n_{e_i,2}$ will be changed to point to $n_{e_i,1}$ and duplicated edges will be removed. Note that the set of the node $n_{e_i,1}$ now contains more than one atom, but the position of the node $p(n_{e_i,1})$ is no longer directly related to the positions of these atoms. We iterate this procedure until the graph reaches a stable status, which is measured by the maximum overall speed of a nodes (usually after 50 to 80 iterations). A result can be seen in figure 4.

3.2 Graph Simplification

Although the graph resulting from this method visually seems to be very close to the structure we want to extract, the topology is not as clean as it has to be for the upcoming task of segmenting the atom data. To perform this segmentation we need to separate nodes in the graph which represent dislocations from nodes which represent junctions of dislocations. We propose two different approaches to classify nodes in the graph to be junctions or not: based on the degree of the node or based on the positions of the direct neighboring nodes.

The first approach, classification using only the list of edges is trivial, because all nodes connected to more than two edges can be considered to be junctions.

This clean method is clearly the goal for our final structure. However, the graph data created by the mass-spring system still contains too complex structures (*unclean* and *thicker* junction regions in figure 4) for this method to work. We therefore propose for the upcoming simplification tasks a second method.

Fig. 5. Classification of nodes to be junctions or not. Left: edges to the projected node position $p'(n_i)$ are all placed on a line (within a threshold), therefore n_i is not a junction. Right: edges do not form a line (red edge), therefore in this case n_i is a junction.

The idea is to check the directions of the edges connected to the node. When classifying the node n_i, we define E_i to be the set of connected edges $E_i = \{e_j \in E | n_i = n_{e_j,1} \vee n_i = n_{e_j,2}\}$ and N_i the set of nodes directly connected by these edges $N_i = \{n_k \in N | \exists e_j \in E_i : n_k = n_{e_j,1} \vee n_k = n_{e_j,2}\} \backslash \{n_i\}$. To take a small curvature of the dislocation structures into account (figure 5) we define a supporting position $p'(n_i) = (|N_i|)^{-1} \sum_{n_k \in N_i} p(n_k)$. Given V_i a set of normalized vectors representing the edges of E_i formed between the position $p(n_k)$ and $p'(n_i)$, where n_k is the second node connected to the edge apart from n_i. We chose WLOG $v_1 \in V_i$ to be our reference vector, and we now compute the dot products between all remaining vectors v_j and v_1. If all results are 1 or -1, within a given threshold (we use 0.01), all edges form a linear structure and thus this node n_i is classified not to be a junction.

Using this classification we continue the simplification of the graph by again employing a mass-spring system, similar to the one presented in section 3.1, but without considering edges from F. In addition, we double the collapsing threshold, but only collapse edges between nodes which have been classified similar using the classification described above. The classification is recalculated after each iteration. We terminate this phase of our algorithm when the graph stabilizes (no more edges collapsing).

In addition, we remove edge cycles of length three from the graph when their overall length is small. Figure 6 shows a simplified situation in which the graph can reach a stable state, thus preserving such cycles inside one linear structure, when all edges of this cycle are too large foe the collapsing threshold. We collapse these cycles into a single node by choosing one node and incorporating the attributes of the other two into it (similar to collapsing an edge). This heuristic of removing these cycles yields to very good results.

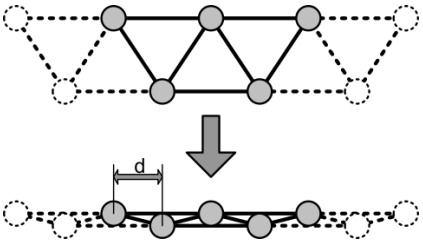


Fig. 6. Schematic view of a problematic situation. The upper image shows the relative neighborhood graph. The mass-spring system contracts the graph to the lower and stable image. However, the distance between the atoms d might be too large, so the edges do not collapse and the three-edge cycles become stable.

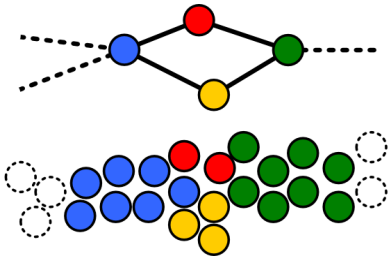


Fig. 7. Four nodes and edges in the graph forming a diamond which only represents a single dislocation. The yellow and red atoms are separated by mistake.

There is also a very rare case where cycles of four edges are formed within a single dislocation. This happens due to the varying size of the original atom data, and thus the varying size of the neighborhood graph near this location. The four edges form a diamond structure connecting two junctions and two non-junctions, where the two non-junctions only hold very few atoms in their sets (figure 7). We simply remove this structue by merging the two non-junctions, although they are not directly connected.

A third heuristic is to remove leaf nodes from the graph. This is valid due to the nature of dislocations, since these structures always form cycles or complex graphs, but can never end or emerge within an material. Therefore leaf nodes are only created by outlier atoms in the input data. We perform these actions iteratively until the graph stabilizes. After this simplification the graph is small enough that we can use the easier edge classification based on the degree of the node, since both methods now create the same results. (See also the discussion on this heuristics in the conclusion of section 4).

3.3 Atom Data Segmentation

The simplified graph data represents the topology of the original data set and can therefore be used to segment this data set into the individual dislocations. To directly use this graph for the segmentations we further simplify its structure, by collapsing all edges connecting nodes of the same classification (junction or non-junction). Since we understand junctions as points where dislocation meet we want them to be ideal points without any size. Therefore we remove all atoms $a_j \in n_j$ from the sets of junction nodes $n_j \in N$, by reassigning these atoms to the set of the connected non-junction node which contains the atom which is closest to the atom to be reassigned $n_i \in N$ with $MIN_{a_i \in n_i} |a_i - a_j|$. After this

operation all junction nodes have empty atom sets. However their position is calculated using the atoms which they contained before reassignment.

After this operation is finished, each edge in the graph connects a junction node to a non-junction node, and all atoms are assigned to one non-junction node. To perform the segmentation of the atom data set we can now simply assign all atoms in the set of each non-junction node the same ID value.

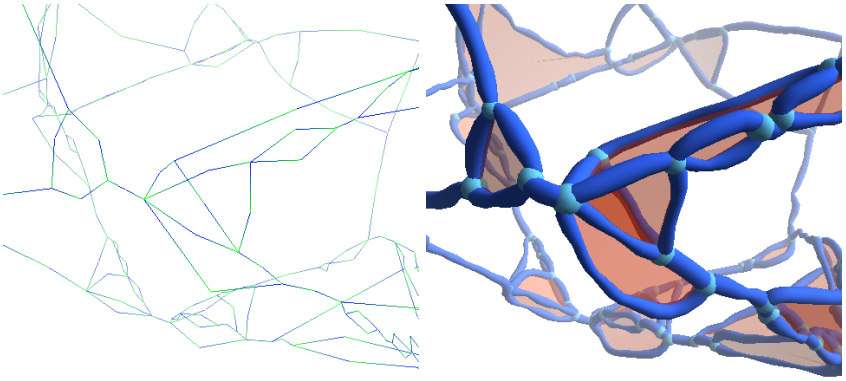


Fig. 8. Left: the graph after all simplification steps; Each edge connects a junction to a non-junction node. All atoms are associated with non-junction nodes. Right: the final result; Junctions are represented by cyan spheres, dislocations by blue tubes, and stacking faults by orange planes.

To generate a clearer representation of the dislocations we fit a tube into each atoms segment. This is trivial because we kept the positions of junctions nodes, and they are used to define the start and an end points for the tube, and we simply interpolate a straight line between them. Whenever the distance of atoms to the line gets too big, we split the line in two and continue recursively with both line segments. This way we not only find a simple line through the segment, but we also calculate a radius for a tube, by defining a plane perpendicular to a line segment going through the midpoint of that line segment and projecting all atoms near onto this plane. By fitting a circle into these points we get a useful radius.

Our final results (figure 8) also show spheres as representatives of the junctions. These are simply placed at the positions of the junctions nodes and use a radius depending on the maximum radius of the line segments connected to this junction.

3.4 Stacking Fault Extraction

After we extracted the dislocation structures and created a concise representation using tubes, we can now focus on creating a similar visualization on the

stacking faults. Stacking faults are always aligned to a crystal plane, so it is sufficient to perform a segmentation on the atom sets and then fitting a single polygon into the data. Furthermore stacking faults are always bordered by dislocations, because they also cannot emerge or disappear within a normal crystal. We propose only to determine the existence of a stacking fault and to find their surrounding dislocations. Since we already extracted the dislocations as tubes, we can simply connect their lines with a polygon.

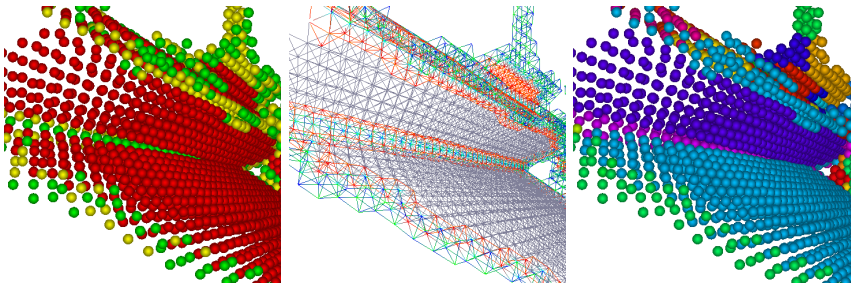


Fig. 9. Shows a problematic situation for the stacking fault segmentation using region growing, from left to right: The original data set; In the relative neighborhood graph atoms from different stacking fault segments are connected; the final segmentation.

The main idea of segmenting the atom data for the stacking faults is to perform a region growing on the neighborhood graph using edges $g_i \in G$. However, the data we used have crystal planes meeting under sharp angles (60 degree), which results in neighboring stacking faults meeting at one dislocation and being connected in the graph (figure 9). Simply ignoring edges from F is not possible, since some stacking faults are only one atom layer thin and therefore do not have edges from G. To fix this, we define a border region of atoms N_B as set of all atoms connected by at least one edge $g_i \in G$ and at least on edge $e_j \in E \bigcup F$. We define N_G to be the set of atoms only connected to edges $g_i \in G$, and chose an atom from N_G as seeding point for a region growing by assigning a new segment ID, repeating this until all atoms from N_G are segmented. However, it might happen that atoms from N_B will not be assigned to any segment. To assign atoms of N_B to a segment we look at all neighbor atoms of N_B which are $a_i \in N_B \bigcup N_G$ and have already a segment ID. We save the segment ID which is assigned to most neighbors for the atom we want to segment, but we assign these IDs all at the same time at the end of an iteration step. Since the border area is equally thick (one atom) with this approach all segments grow with the same speed (one atom per iteration), we will get clean results in the problematic scenarios described above.

As second step we collapse all edges $g_i \in G$ which connect nodes with the same segment ID by merging there atom sets, similar to the approach of the dislocation extraction. Eedges $f_i \in F$ are used to determine correspondences between the stacking fault segments and the dislocation segments. We sort the

list of dislocation segments based on the junctions nodes they use to form a cycle surrounding the stacking fault. We then simply span a polygon inside this cycle to represent the stacking fault (e. g. figures 8 and 11). Rendering these polygons using transparency also lessens the occlusion problem compared to the original atomistic representation.

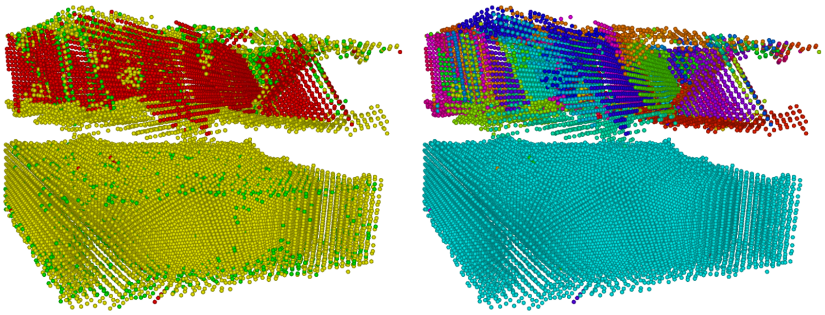
3.5 Segment Tracking

The last remaining feature of our visualization is the tracking of the extracted features (dislocations and stacking faults) over time. Although we handle dislocations and stacking faults independently in this tracking process, we apply the same method to both data structures. Because of this we will not differentiate between these. We perform the segmentation of the atom data independently for each time step. The tracking is done by relabeling the extracted segments with IDs from the last time step. We compute the overlap of the atom sets of the nodes of the neighborhood graph which represent the segments in one time step with the sets from the previous time step and pair those segments with the largest overlap. We order the segments by their size decreasingly and start with the largest segment performing this action. Due to our graph simplification, our results are a bit unsteady at the junction regions, but the tracking shows that even in these regions the main segments are quite stable (see the attached video material).

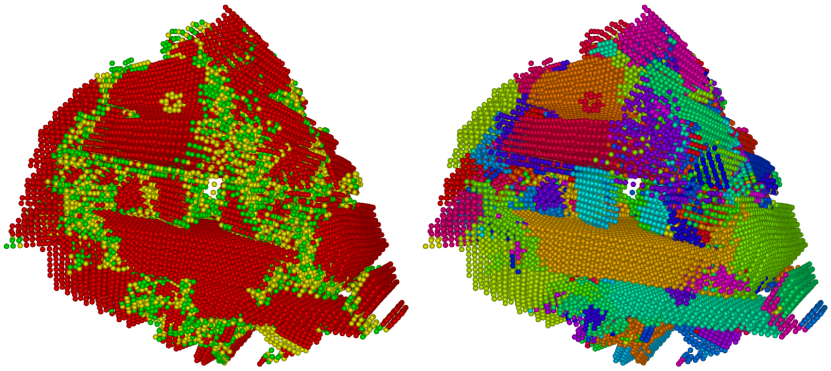
4 Results and Future Work

We presented an approach for generating simplified representations molecular dynamics data sets containing crystal structure defects, especially dislocations and stacking faults. We thereby create a connection between visualizations of atomistic data and data from dislocation dynamics. The proposed feature extraction and tracking is performed in a preprocessing step. On a common desktop computer (Intel Core2Duo 6600@2.40 GHz, 2 GB RAM, NVidia GeForce 7900 GT) these calculations take an average time of about 20 seconds per time step (depending on the complexity of the contained structure) for the Ni-Ni_3Al data set mainly used in this paper. These preprocessing results can then be visualized and explored interactively on the same machine. The rendering of the atoms is done using point-sprites, so we have no problems interactively rendering (more then 10 FPS) up to millions of atoms. However, we only have this large number of atoms in artificial test data sets. The real world data sets we worked with contained between 13,000 and 87,000 atoms, and can be rendered with far more then 60 FPS. The extracted structure is stored as triangle mesh and uploaded using vertex arrays. Our data sets resulted in meshes containing between 28,000 and 93,000 triangles. Interactive exploration of the time-dependent data set can be done employing an out-of-core streaming approach.

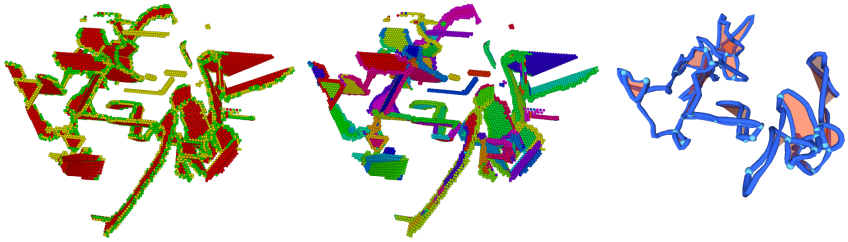
In addition to the Ni-Ni_3Al data set mainly used in this paper and used for the video supplied with this paper, we tested our approach with several other data



(a) Data set 2 containing two layers (upper one Ni_3Al; lower one Ni) pulled apart.



(b) Data set 3 containing only Ni_3Al pulled in one direction.



(c) Data set 4 containing Ni atoms pulled in one direction.

Fig. 10. Three datasets visualized with our method. All show the original data set with the lattice classification (left) and the results of the atom data segmentation (right; center in figure 10(c)). Right image in figure 10(c) shows the results of the structure extraction.

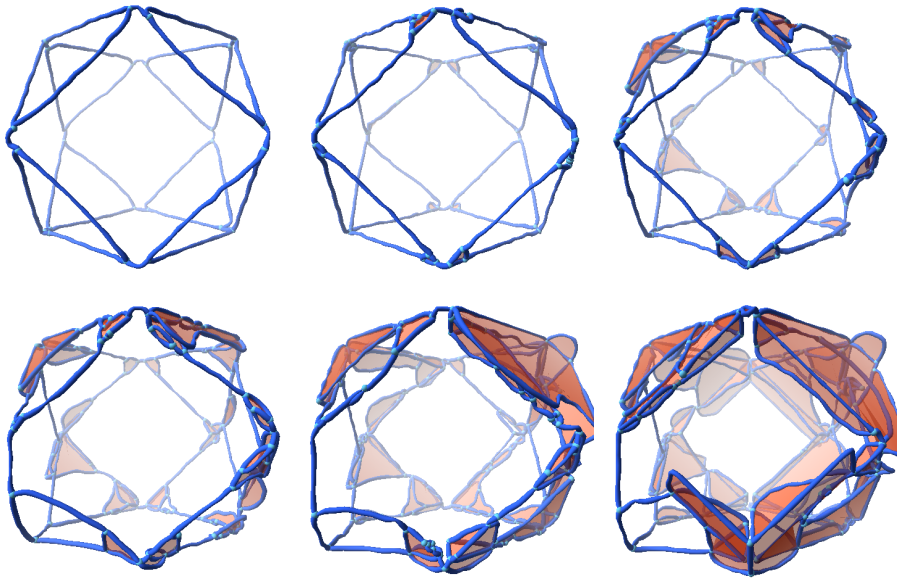


Fig. 11. Extracted crystal defects of time steps 10, 250, 350, 400, 500 and 595.

sets, all from molecular dynamic simulations and all containing crystal structure defects (Figure 10). Figure 10(a) shows a simulation of a probe consisting of one layer Ni_3Al and one layer Ni. The two layers are pulled apart. Although the feature extraction cannot be performed because of the lower layer has a BCC lattice and we don't have a suitable neighborhood classification for atoms in this lattice (see discussion below), the results of the segmentation (right image) are very good for the FCC lattice. The other two of these data sets are simulations of clean materials (Ni_3Al in figure 10(b) and Ni in figure 10(c) pulled in one direction. The segmentation of the atoms is good in both cases. In the Ni_3Al data set the stacking faults group together in big blocks, thus making our approach of using single polygons unapplicable. Nevertheless, the segmentation allows a clearer view of the crystal planes in complex regions of the material. The Ni data set shown in figure 10(c) also works with our feature extraction (right image). However, our current implementation is missing some of the features due to some problems with tracking the atom segments over the periodic boundary conditions (Note for reviewers: we are positive to fix this problem for the final version).

Figure 11 shows the crystal structure evolution in the Ni-Ni_3Al data set by showing six relevant time steps. The data sets does not change significantly until about time step 300. Then dislocations start to split up and large area of stacking faults start to form (e. g. in the upper right area) and dislocations also start to change their form and position (e. g. in the lower left area, in front). The semi transparent rendering of the stacking faults not only makes the occlusion less

problematic, it also allows to identify the crystal planes more easily on which they occur.

Our approach still has some problems. Especially the heuristics applied to simplify the neighborhood graph are not very robust. Although they work very well on all data sets we used, and although they are based on knowledge from the application domain, we want to enhance their quality by taking the coherence between different time steps of the data set into account. First basic tests are encouraging. The information we get by tracking the cluster segments could be used for further cleaning the results and additionally the results of the atom data segmentation could be used in the next time step as starting condition.

Another problem already mentioned earlier is that our approach currently only works for FCC crystals, because we relay on the atom neighborhood classification in the input data for identifying atoms forming dislocations and junctions (as described in [18]). We want to extend our approach based on a similar classification to BCC crystals, however, it is not clear at the moment how such an classification can be made.

According to the feedback we got from the domain experts we were working with, the clear representation of the crystal planes is really beneficial, as well as the tracking of the dislocation. There is no currently available visualization tool which is capable of handling atomistic data sets from molecular dynamic simulations in the described manner. For future work we want to optimize our approach, eliminate all problems described above, and integrate our visualization in a framework allowing better user interaction, making our tool more usable for the domain experts.

References

1. Bulatov, V., Cai, W.: Computer Simulation of Dislocations. Oxford University Press, USA (2006)
2. Sethna, J.P.: Statistical Mechanics: Entropy, Order Parameters and Complexity. Oxford University Press (2006)
3. Schall, P., Cohen, I., Weitz, D.A., Spaepen, F.: Visualization of dislocation dynamics in colloidal crystals. Science **24** (2004) 1944–1948
4. Schall, P., Cohen, I., Weitz, D.A., Spaepen, F.: Visualizing dislocation nucleation by indenting colloidal crystals. Nature **440** (2006) 319–323
5. Hartley, C., Mishin, Y.: Characterization and visualization of the lattice misfit associated with dislocation cores. Acta Materialia **53** (2005) 1313–1321
6. Lipowsky, P., Bowick, M.J., Meinke, J.H., Nelson, D.R., Bausch, A.R.: Direct visualization of dislocation dynamics in grain-boundary scars. Nature Materials **4** (2005) 407–411
7. Bulatov, V., Cai, W., Fier, J., Hiratani, M., Hommes, G., Pierce, T., Tang, M., Rhee, M., Yates, K., Arsenlis, T.: Scalable line dynamics in paradis. In: SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing, Washington, DC, USA, IEEE Computer Society (2004) 19
8. Bulatov, V.V., Hsiung, L.L., Tang, M., Arsenlis, A., Bartelt, M.C., Cai, W., Florando, J.N., Hiratani, M., Rhee1, M., Hommes, G., Pierce, T.G., de la Rubia, T.D.: Dislocation multi-junctions and strain hardening. Nature **440** (2006) 1174–1178

9. Bulatov, V., Abraham, F.F., Kubin, L., Devincre, B., Yip, S.: Connecting atomistic and mesoscale simulations of crystal plasticity. Nature **391** (1998) 669–672
10. Cornea, N.D., Min, P.: Curve-skeleton properties, applications, and algorithms. IEEE Transactions on Visualization and Computer Graphics **13**(3) (2007) 530–548 Member-Deborah Silver.
11. Supowit, K.J.: The relative neighborhood graph, with an application to minimum spanning trees. J. ACM **30**(3) (1983) 428–448
12. Agarwal, P.K., Mataušek, J.: Relative neighborhood graphs in three dimensions. In: SODA '92: Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics (1992) 58–65
13. Gyulassy, A., Natarajan, V., Pascucci, V., Bremer, P.T., Hamann, B.: A topological approach to simplification of three-dimensional scalar functions. IEEE Transactions on Visualization and Computer Graphics **12**(4) (2006) 474–484
14. Edelsbrunner, H., Harer, J., Mascarenhas, A., Pascucci, V.: Time-varying reeb graphs for continuous space-time data. In: SCG '04: Proceedings of the twentieth annual symposium on Computational geometry, New York, NY, USA, ACM (2004) 366–372
15. Carr, H.: Efficient generation of 3-d contour trees. Master's thesis, University of British Columbia (2000)
16. Natarajan, V., Edelsbrunner, P.H., Arge, P.L., Harer, P.J., Sun, P.X., Natarajan, V., Edelsbrunner, P.H., Arge, P.L., Harer, P.J., Sun, P.X.: by (2004)
17. Au, O.K.C., Tai, C.L., Chu, H.K., Cohen-Or, D., Lee, T.Y.: Skeleton extraction by mesh contraction. ACM Trans. Graph. **27**(3) (2008) 1–10
18. Honeycutt, J.D., Andersen, H.C.: Molecular dynamics study of melting and freezing of small lennard-jones clusters. Journal of Physical Chemistry **91**(19) (1987) 4950–4963