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 dynamics 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 fail 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 renderings into a concise representation. The benefits of our approach are clearer images while retaining relevant data and easier visual tracking of topological changes 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 is the appearance of defects in their atomic structure[4], which have a direct relation to material properties such as resistance, strength, etc. For this purpose, Molecular Dynamics (MD) simulations are used, which reproduce the behavior of atoms and molecules for a period of time. MD generates a great number of components which require complex analysis, and often only statistical measurements are evaluated.

However, the analysis of defects can be restricted to a subset of this data. Defects are local changes in symmetry or regularity of the crystal lattice. Because the atomic structure of compound materials in normal conditions corresponds to crystal lattice structures, it suffices to consider those atoms whose neighborhoods deviate from the regularity established by this lattice. Common lattices include the body-centered cubic (BCC), face-centered cubic (FCC) or hexagonal-closed packed (HCP). The atoms of irregular lattice structures form defects called *dislocations* (a 1-D defect) and *stacking faults* (a 2-D defect). There are special properties from the application domain on which we can rely. Dislocations always form cycles or networks, and never have open, unconnected endpoints. Stacking faults are always bordered by dislocations, which makes it easy to define where they end or start. For FCC lattices, atoms which participate in

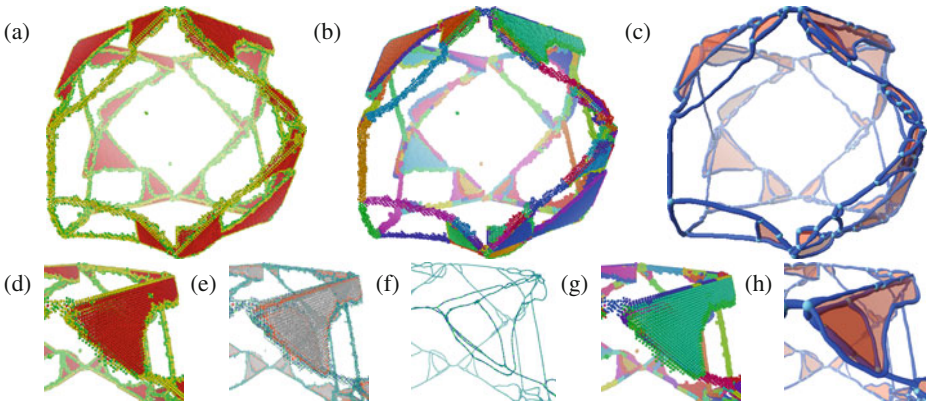


Fig. 1. Upper row: (a) original atom data, (b) after segmentation, (c) extracted defects. Lower row: Detail views - (d) original atom data, (e) relative neighborhood graph, (f) graph after simplification, (g) segmented atom data, (h) extracted defects.

these defects can be identified by testing the geometric properties of a 12-atom neighborhood, which has already been done successfully by domain scientists. For the other crystal types (BCC or HCP) such a classification is not yet known. Therefore, in the remainder of this paper we will only handle FCC crystals.

The topological structure induced by such defects and their interplay helps evaluate the material properties. The identification of dislocations and stacking fault structures from atomistic simulations is similar to 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 MD 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. Since the data is filtered, which is described in section 3, the loaded files only contain between 13,000 and 87,000 atoms, depending on the scene complexity.

2 Background Material and Related Work

Several works in MD discuss the evaluation of material properties under external forces (see Bulatov et al. [4] for a good introductory book on the subject). Since the atomic structure of many solid materials in normal conditions corresponds to crystal lattice structure, defects are observed when changes to the lattice structure occur. Changes in the regularity, symmetry, or ordering of this 12-atom neighborhood create topological defects on the atomic structure[18]. Atoms with irregular neighborhood align in 1-D dislocations surrounding 2-D stacking faults. Geometric measures give one way to

evaluate these defects, such as the Burgers vector [4] 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. [16, 17] discuss ways to track dislocation in colloidal crystals, and techniques to visualize the distribution of Burgers vectors (called Nye tensors). Hartley et al. [11] also discusses 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 meso-scale representations are often used. Dislocation dynamics is an example of such a meso-scale simulation. The work by Lipowsky et al. [13] discusses a way to track and visualize dislocation directly in grain-boundary scars. Bulatov et al. [3, 5] suggests tracking topological features of dislocations, such as junctions or multi-junctions (three or more dislocations in one place). They observe in [6]: "In large-scale dislocation dynamics simulations, multi-junctions present very strong, nearly indestructible, obstacles to dislocation motion and (...) 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 structure from the atomistic data, which identifies features like dislocations and junctions between them. There is a vast literature on skeletonization algorithms, and a good introductory survey can be found in [8]. In this survey there are four classes of skeletonization algorithms: thinning and boundary propagation, distance-field based, geometric methods, and general-field functions. In our work no implicit boundary is formed, but a neighborhood graph [1, 19] that defines atom proximity and allows thinning contraction operations. This is similar to thinning algorithms in distance-field methods, with respect to the approach, but different since no distance field is used. Several possibilities for controlling contraction operations can be defined, and smoothing procedures are often employed [2]. We use a simple mass-spring approach (see Section 3.1). 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 [7, 9, 10]. General field methods such as potential field functions are used in a similar application described in [14]. In their work, crystal dislocation data is given as a potential field function, and a Morse-Smale complex is used to evaluate the structure of dislocations. Since we do not have such a field, we can not employ these methods.

3 Structure Defects Extraction

Visualizations for dislocation dynamics simulations can use the meso-scale data to construct concise representations of the dislocations, which allow easy tracking over time. In atomistic simulations there is no such data description. To generate an analogous visualization, extraction of dislocations from the atomistic data is necessary. We propose an algorithm with the following steps:

1. Reconstruction of the neighborhood graph
2. Extraction of dislocations
   2.1. Contraction of the graph using a simple mass-spring system
   2.2. Edge collapse

2.3. Dislocation junction identification
2.4. Atom data segmentation
3. Extraction of stacking faults
4. Tracking over time

The input data consists of a list of atoms A, each storing its position, a unique ID and a lattice classification c_a, for the atom $a \in A$, based on the nearest neighbors of the atom according to [12]. We will use this classification to identify atoms creating defects. We assume that the undistorted crystal of our material forms an FCC lattice. Atoms with such a neighborhood are classified $c_a = 0$. The atoms which form a HCP lattice have $c_a = 1$. Atoms which have 12 closest neighbors but neither form a FCC lattice nor a HCP lattice have $c_a = 2$. All remaining atoms have $c_a = 3$. Since we are interested in only visualizing the crystal defects, we can completely omit the more than 90 percent of the atoms in class $c_a = 0$. Changes from FCC to HCP ($c_a = 1$) indicate planar displacements and hence 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 a & d: red atoms are $c_a = 1$, green $c_a = 2$, and yellow $c_a = 3$). Using this lattice classification for identifying crystal defects is much easier to compute than Burgers vectors for all atoms.

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 domain knowledge that atom distances are uniform within narrow bounds as shown in figure 2. This is a graph similar to the one used by the MD simulation. We have to recreate this graph since the simulation graph was not stored with the MD data due to data file size considerations.We propose to simplify this graph using a very simple mass-spring system. We define N to be the list of all nodes in the graph, where each node $n \in N$ stores a set of atom indices $a_i \in n$ for all atoms which are represented by this node. Initially we create one node for each atom $n_i = \{a_i\}$ and place the node at the position of that atom $p(n_i) = p(a_i)$. We create edges between nodes of atoms which are within the neighborhood radius. These edges are stored in three lists E_d, E_s, and E_b, depending on the atom classifications c_a, since these edges will be used by different parts of our algorithm. E_d 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). These edges form *dislocations* and will be used by the graph contraction in the dislocation extraction described in section 3.1. E_s holds edges connecting a_3, a_4 with $c_{a_3} = 1 \wedge c_{a_4} = 1$ (shown in gray). These edges form *stacking faults* and will be used in our region growing approach described in section 3.4. E_b (*border*) holds all remaining edges of the neighborhood graph (shown in red). These edges separate dislocations and stacking faults. They will ensure that the graph contraction does not destroy small stacking faults and they will be used to terminate the region growing.

3.1 Graph Contraction

The first step of the graph simplification is a contraction implemented using a simple mass-spring system. Each node n_i manages a speed vector v_i, which is initialized to zero. However, only nodes connected to at least one edge $e \in E_d$ can be moved, while

other nodes are fixed. All edges of E_d and E_b are treated as springs. Edges $e \in E_d$ try to collapse to length of zero and edges $f \in E_b$ try to keep their initial length. These edges f will keep the subgraph of the edges e out of the stacking fault regions and will prevent our method from collapsing small stacking fault regions. This is why we are using a mass-spring approach instead of simple clustering (e. g. shortest-edge-first). This way we can collapse edges of various lengths and still be sure not to loose small features.

In addition to simply moving the nodes in the graph we collapse small edges. The corresponding threshold is based on the neighborhood radius which was used at initialization to build the graph. In our examples we obtained good results using a collapsing threshold which is half of the neighborhood radius. However, this value might be adjusted by the user. When collapsing the edge $e_i \in E_d$ we combine all attributes of the two original nodes $n_{e_i,1}$ and $n_{e_i,2}$ weighted by the relative number of atoms assigned to each node: $n_{e_i,1} = n_{e_i,1} \bigcup n_{e_i,2}$, $p(n_{e_i,1}) = (|n_{e_i,1}| + |n_{e_i,2}|)^{-1}(|n_{e_i,1}|p(n_{e_i,1}) + |n_{e_i,2}|p(n_{e_i,2}))$ (and all other attributes, e. g. speed vector, accordingly). Note that the position of the node $p(n_{e_i,1})$ is no longer directly related to the positions of the atoms it contains. We iterate this procedure until the graph reaches a stable status, which is measured by the maximum overall speed of all nodes (usually after 50 to 80 iterations). A result can be seen in figure 3.

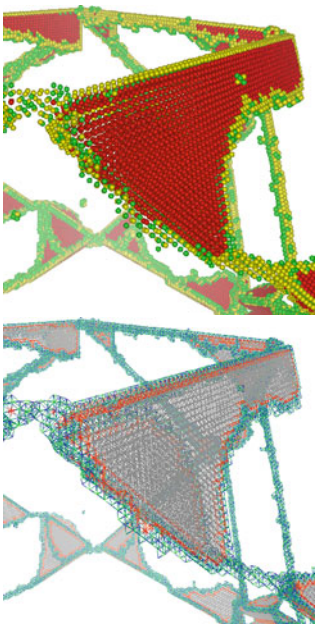


Fig. 2. Construction of the neighborhood graph (bottom) from the original atom data (top). Edges are color-coded: green/blue edges are from E_d, red/orange from E_b, and gray from E_s.

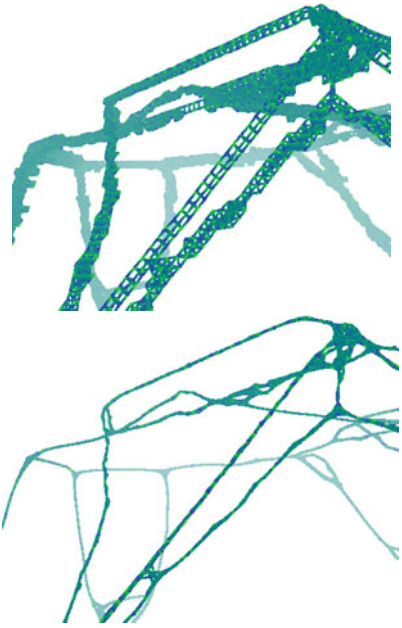


Fig. 3. Top image: initial neighborhood graph (only edges from E_d are shown); bottom: graph after contraction of the mass-spring system (50 iterations). Note the *unclean*, *thicker* regions near junctions.

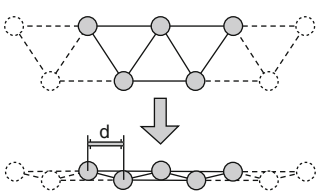


Fig. 4. The mass-spring system contracts the relative neighborhood graph to the lower image. When the distance d between atoms is large, edges do not collapse and the three-edge cycles become stable.

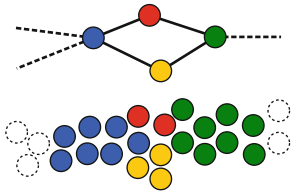


Fig. 5. Four nodes and edges in the graph forming a diamond which represents a single dislocation. Yellow and red atoms are incorrectly separated.

3.2 Graph Simplification

Although the resulting graph visually seems to be the structure we want to extract, it is still too complex for the upcoming segmentation of the atom data. 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 as junctions or non-junctions: based on the degree of the node or based on the positions of the direct neighboring nodes. The first approach is trivial (using only the list of edges). All nodes connected to more than two edges can be considered junctions. However, the graph data created by the mass-spring system still contains structures that are too complex for this method to work as intended, such as *unclean* and *thicker* junction regions (see upper-right region of lower image in figure 3). Therefore we use a second method where we 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 $E_i = \{e_j \in E_d | n_i = n_{e_j,1} \vee n_i = n_{e_j,2}\}$ and N_i the set of nodes directly connected by these edges $N_i = \{n_k \in N | \exists e_j \in E_i : n_k = n_{e_j,1} \vee n_k = n_{e_j,2}\} \setminus \{n_i\}$. Given V_i a set of normalized vectors representing the edges of E_i formed between the positions $p(n_k)$ and a supporting position $p'(n_i) = (|N_i|)^{-1} \sum_{n_k \in N_i} p(n_k)$, where n_k is the second node connected to the edge apart from n_i. The supporting position is the mean of all neighbor positions. It is used to compensate for small curvature in the dislocations. If the dot product between all pairs of vectors in V_i is 1 or -1 (within a small threshold) the vectors form a linear structure and the node n_i is classified as a non-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 E_b. In addition, we increase the collapsing threshold, but only collapse edges between nodes which are both junctions or both non-junctions. The classification is recalculated after each iteration. We terminate this phase of our algorithm when the graph stabilizes (no more edge collapsing). The left image of figure 6 shows the result from the graph in the bottom image of figure 3.

Figure 4 shows a simplified situation in which the graph can reach a stable state, thus preserving cycles inside one linear structure, when all edges of this cycle are too large for the collapsing threshold. 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

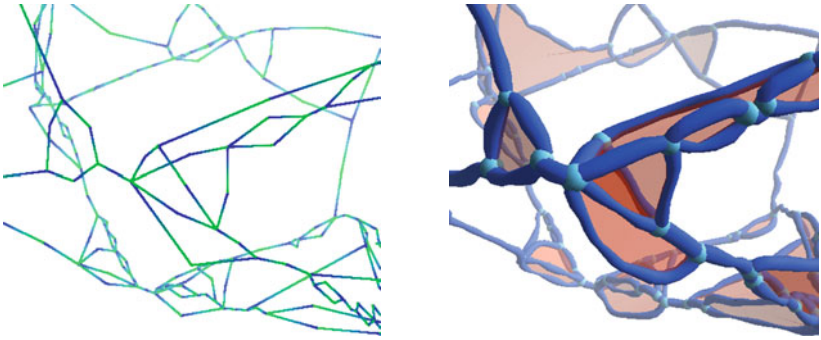


Fig. 6. Left: the graph after all simplification steps; Each edge connects a junction to a non-junction node. All atoms are associated with non-junction nodes. Right: the final result; Junctions are represented by cyan spheres, dislocations by blue tubes, and stacking faults by orange planes.

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 5). We use heuristics to identify these situations and collapse these cycles into single nodes.

3.3 Atom Data Segmentation

The simplified graph represents the topology of the original data set and can be used to segment it into individual dislocations. To use this graph directly for segmentation we further simplify its structure, by collapsing all edges connecting nodes of the same classification (junction or non-junction). 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 non-junction nodes. The segmentation of the atom data is now done by assigning all atoms in the set of each non-junction node the same ID value. For the final representation of dislocations we fit a tube into each atoms segment. Since the junction nodes define the end points of each dislocation, a simple linear distance threshold based fitting of spline tubes can be used. Our final results (figure 6) also show junctions as spheres. These are placed at the positions of the junctions nodes and use a radius depending on the radii of the connected segments.

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. Since

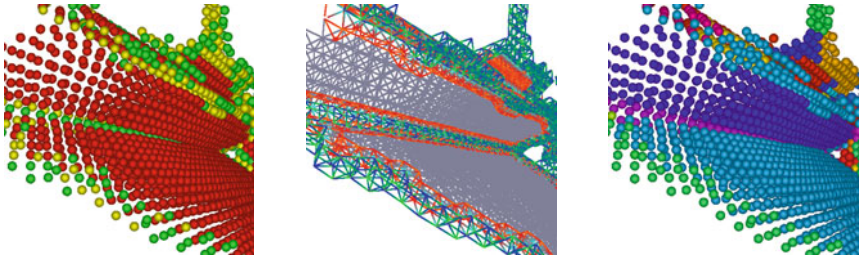


Fig. 7. Shows a problematic situation for the stacking fault segmentation using region growing, from left to right: The original data set; In the relative neighborhood graph atoms from different stacking fault segments are connected; the final segmentation.

stacking faults are always aligned to a crystal plane we can use a flat polygon as representation. Furthermore stacking faults are always bordered by dislocations. Since we already extracted the dislocations as tubes, we can just connect them with a polygon.

The main idea of segmenting the atom data for the stacking faults is to perform a region growing on the neighborhood graph using edges $g_i \in E_s$. 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 7). Simply ignoring edges from E_b is not possible, since some stacking faults are only one atom layer thick and therefore do not have edges from E_s. To fix this, we define a border region of atoms N_B to be a set of all atoms connected by at least one edge $g_i \in E_s$ and at least on edge $e_j \in E_b$. We define N_G to be the set of atoms connected only to edges $g_i \in E_s$, and grow a region outwards from a seeding point chosen from N_G. We assign a new segment ID for this region and repeat this method until all atoms from N_G are segmented. However, atoms from N_B are not guaranteed to be assigned to a segment. To assign these atoms 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, and we assign these IDs all at the same time at the end of an iteration step. By doing so, all segments grow with the same speed (one atom per iteration). Since the border area is equally thick (one atom) we get clean results in the problematic scenarios described above.

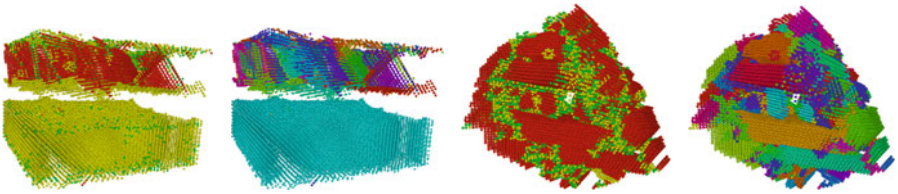
In our second step we collapse all edges $g_i \in E_s$ which connect nodes with the same segment ID by merging atom sets, similar to the approach of the dislocation extraction. Edges $f_i \in E_b$ are used to determine correspondences between the stacking fault segments and the dislocation segments. We sort the list of dislocation segments based on the junction nodes they use to form a cycle surrounding the stacking fault. We then simply fit a polygon into the atom set and stitch it to the surrounding dislocation cycle (e. g. figures 6 and 9). Rendering these polygons using transparency also reduces 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. There have been many publications on

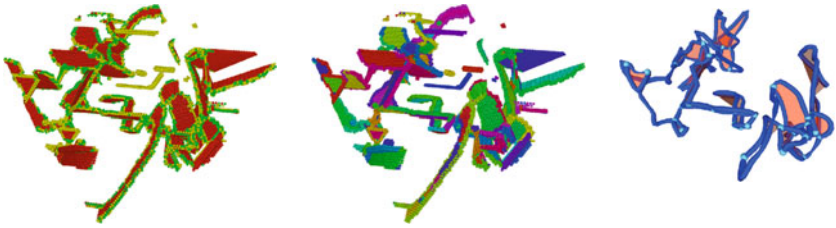
feature tracking in general (e. g. by Samtaney et al. [15]). However, we can use a simple approach by utilizing the fact that all of our features still know the atoms they were created from. Using the unique atom IDs we can easily trace the atoms over time. We perform the segmentation of the atom data independently for each time step. The actual 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 segments in one time step with the sets from the previous time step and pair those segments with the largest overlap. Although we handle dislocations and stacking faults independently in this tracking process, we apply the same method to both feature types. 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.

4 Results and Future Work



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

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



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

Fig. 8. Visualization examples. Each sequence shows the original data set with the lattice classification (left) and the results of the atom data segmentation (right; center in figure 8(c)). Right image in figure 8(c) shows the results of the structure extraction.

We presented an approach for generating simplified representations of MD data sets containing crystal structure defects, especially dislocations and stacking faults. We thereby created 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 structure of the crystal lattice defect to be extracted) for the Ni-Ni_3Al data set mainly used in this paper. These

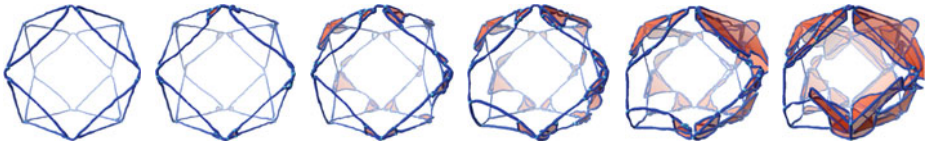


Fig. 9. Extracted crystal defects of time steps 10, 250, 350, 400, 500 and 595.

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 than 10 FPS) up to millions of atoms. However, we only have this many 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 than 60 FPS. The structure extracted is stored as a triangle mesh. 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, we tested our approach with several other data sets, all from MD simulations and all containing crystal structure defects (Figure 8). Figure 8(a) shows a simulation of a probe consisting of one layer of Ni_3Al and one layer of Ni. The two layers are pulled apart. However, the feature extraction cannot be performed because the lower layer has a BCC lattice (see discussion below). The other two of these data sets are simulations of clean materials (Ni_3Al in figure 8(b) and Ni in figure 8(c), each pulled in one direction). In the Ni_3Al data set the stacking faults group together in big blocks, thus making our approach of using single flat polygons inapplicable. Nevertheless, just segmenting the atoms allows a clearer view of the crystal planes in complex regions of the material. The Ni data set shown in figure 8(c) also works with our feature extraction (right image). However, our current implementation misses some of the features due to some problems with tracking the atom segments over the periodic boundary conditions of the simulation area (truncated octahedron). Figure 9 shows the crystal structure evolution of the Ni-Ni_3Al data set in six relevant time steps. The data sets does not change significantly until about time step 300. Then dislocations start to split up and change their form and position (e. g. in the lower left area, in front) and large stacking faults emerge (e. g. in the upper right area).

Our approach still has room for improvements. In particular, the heuristics applied to simplify the neighborhood graph are not completely robust, although they work very well on all data sets we used and they are based on knowledge from the application domain. We want to enhance their quality by taking the coherence between different time steps into account. The result of the atom data segmentation could be used in a following time step as starting point. Our initial tests are encouraging.

Another problem already mentioned earlier is that our approach currently only works for FCC crystals, because we rely on the atom neighborhood classification in the input data for identifying atoms forming dislocations and junctions (see [12]). We want to extend our approach based on a similar classification to BCC crystals. However, it is not clear at the moment how such classification can be made.

Domain scientist partners in this project gave us very positive feedback on our visualization. The clear representation of the crystal planes is really beneficial, as well as the tracking of the dislocation. The visualization aids them in perceiving the structure of the data and its evolution over time. The semi transparent rendering of the stacking faults not only makes the occlusion less problematic, it also allows the user to identify the crystal planes more easily on which they occur. Since all features and the tracking method rely only on the original atom data, they believe that this visualization is valid. There is currently no visualization tool available which is capable of representing atomistic data sets from MD simulations in the described manner. For future work we want to optimize our approach, and integrate our visualization in a framework allowing better user interaction, making our tool more usable for the domain experts.

Acknowledgements

This work is partially funded by Deutsche Forschungsgemeinschaft (DFG) as part of SFB 716. The work of J. Comba and C. Dietrich is supported by CNPq grant 485853/2007-0.

References

1. Pankaj K. Agarwal and Jiří Matoušek. Relative neighborhood graphs in three dimensions. In *SODA '92: Proceedings of the 3rd annual ACM-SIAM symposium on Discrete algorithms*, pages 58–65, Philadelphia, USA, 1992. SIAM.
2. Oscar Kin-Chung Au, Chiew-Lan Tai, Hung-Kuo Chu, Daniel Cohen-Or, and Tong-Yee Lee. Skeleton extraction by mesh contraction. *ACM Trans. Graph.*, 27(3):1–10, 2008.
3. Vasily Bulatov, Farid F. Abraham, Ladislas Kubin, Benoit Devincre, and Sidney Yip. Connecting atomistic and mesoscale simulations of crystal plasticity. *Nature*, 391:669–672, 1998.
4. Vasily Bulatov and Wei Cai. *Computer Simulation of Dislocations*. Oxford University Press: Oxford, New York, 2006.
5. Vasily Bulatov, Wei Cai, Jeff Fier, Masato Hiratani, Gregg Hommes, Tim Pierce, Meijie Tang, Moono Rhee, Kim Yates, and Tom Arsenlis. Scalable Line Dynamics in ParaDiS. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 19, Washington, DC, USA, 2004. IEEE Computer Society.
6. Vasily V. Bulatov, Luke L. Hsiung, Meijie Tang, Athanasios Arsenlis, Maria C. Bartelt, Wei Cai, Jeff N. Florando, Masato Hiratani, Moon Rhee1, Gregg Hommes, Tim G. Pierce, and Tomas Diaz de la Rubia. Dislocation multi-junctions and strain hardening. *Nature*, 440:1174–1178, 2006.
7. Hamish Carr, Jack Snoeyink, and Ulrike Axen. Computing contour trees in all dimensions. *Comput. Geom. Theory Appl.*, 24(2):75–94, 2003.
8. Nicu D. Cornea, Patrick Min, and Deborah Silver. Curve-Skeleton Properties, Applications, and Algorithms. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):530–548, 2007.
9. Herbert Edelsbrunner, John Harer, Ajith Mascarenhas, Valerio Pascucci, and Jack Snoeyink. Time-varying reeb graphs for continuous space–time data. *Comput. Geom. Theory Appl.*, 41(3):149–166, 2008.

10. Attila Gyulassy, Vijay Natarajan, Valerio Pascucci, Peer-Timo Bremer, and Bernd Hamann. A Topological Approach to Simplification of Three-Dimensional Scalar Functions. *IEEE Transactions on Visualization and Computer Graphics*, 12(4):474–484, 2006.
11. C.S. Hartley and Y. Mishin. Characterization and visualization of the lattice misfit associated with dislocation cores. *Acta Materialia*, 53:1313–1321, 2005.
12. J. Dana Honeycutt and Hans C. Andersen. Molecular dynamics study of melting and freezing of small Lennard-Jones clusters. *Journal of Physical Chemistry*, 91(19):4950–4963, 1987.
13. Peter Lipowsky, Mark J. Bowick, Jan H. Meinke, David R. Nelson, and Andreas R. Bausch. Direct visualization of dislocation dynamics in grain-boundary scars. *Nature Materials*, 4:407–411, 2005.
14. Vijay Natarajan, Prof Herbert Edelsbrunner, Prof Lars Arge, Prof John Harer, and Prof Xiaobai Sun. *Topological Analysis of Scalar Functions for scientific Data Visualization*. PhD thesis, Duke University, 2004.
15. Ravi Samtaney, Deborah Silver, Norman Zabusky, and Jim Cao. Visualizing features and tracking their evolution. *Computer*, 27(7):20–27, 1994.
16. Peter Schall, Itai Cohen, David A. Weitz, and Frans Spaepen. Visualization of Dislocation Dynamics in Colloidal Crystals. *Science*, 24:1944–1948, 2004.
17. Peter Schall, Itai Cohen, David A. Weitz, and Frans Spaepen. Visualizing dislocation nucleation by indenting colloidal crystals. *Nature*, 440:319–323, 2006.
18. James P. Sethna. *Statistical Mechanics: Entropy, Order Parameters and Complexity*. Oxford University Press: Oxford, New York, 2006.
19. Kenneth J. Supowit. The Relative Neighborhood Graph, with an Application to Minimum Spanning Trees. *J. ACM*, 30(3):428–448, 1983.