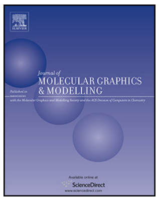
Contents lists available at ScienceDirect

Journal of Molecular Graphics and Modelling

journal homepage: www.elsevier.com/locate/JMGM



MaterialVis: Material visualization tool using direct volume and surface rendering techniques

Erhan Okuyan[a], Uğur Güdükbay[a,*], Ceyhun Bulutay[b], Karl-Heinz Heinig[c]

[a] Department of Computer Engineering, Bilkent University, 06800 Ankara, Turkey
[b] Department of Physics, Bilkent University, 06800 Ankara, Turkey
[c] Helmholtz-Zentrum Dresden – Rossendorf, Bautzner Landstr. 400, 01328 Dresden, Germany

ARTICLE INFO

Article history:
Accepted 23 March 2014
Available online 30 March 2014

PACS:
07.05.Rm
61.72.–y
81.07.Ta
61.43.Dq
61.66.Bi

Keywords:
Material visualization
Embedded nano-structure visualization
Direct volume rendering
Unstructured tetrahedral meshes
Crystal defects

ABSTRACT

Visualization of the materials is an indispensable part of their structural analysis. We developed a visualization tool for amorphous as well as crystalline structures, called *MaterialVis*. Unlike the existing tools, *MaterialVis* represents material structures as a volume and a surface manifold, in addition to plain atomic coordinates. Both amorphous and crystalline structures exhibit topological features as well as various defects. *MaterialVis* provides a wide range of functionality to visualize such topological structures and crystal defects interactively. Direct volume rendering techniques are used to visualize the volumetric features of materials, such as crystal defects, which are responsible for the distinct fingerprints of a specific sample. In addition, the tool provides surface visualization to extract hidden topological features within the material. Together with the rich set of parameters and options to control the visualization, *MaterialVis* allows users to visualize various aspects of materials very efficiently as generated by modern analytical techniques such as the Atom Probe Tomography.

1. Introduction

Extracting the underlying atomic-level structure of natural as well as synthetic materials is vital for materials scientists, working in the fields such as electronics, chemistry, biology, and geology. However, as the topology and other important properties are buried under a vast number of atoms piled on top of one another, this inevitably conceals the targetted information. Without any doubt, the visualization of such embedded materials can help to understand what makes a certain sample unique in how it behaves. However, rudimentary visualization of atoms would fall short because it will not reveal any topological structure or crystalline defects.

In order to visualize the material topology, the data must be represented as a *surface manifold*, whereas, visualization of crystalline defects require extracting and quantifying defects and representing the data *volumetrically*. Current visualization tools lack such features, and hence, they are not very effective for visualizing the material topology and crystalline defects.

Material visualization tools require atomic coordinates of the materials as input. Acquisition of real-space atomic coordinates of a sample has been a major obstacle, until recently mainly restricted to the surfaces. One can call this period as the *dark ages* of material visualization. However, recent techniques, such as *Atom Probe Tomography* [1], can extract atomic coordinates much easier than before. This is also a very active research field, with the promise of many new advances in the near future. Accordingly, as the data acquisition phase for materials gets more efficient and accurate, the necessity for sophisticated material visualization tools becomes self-evident.

Our motivation on *MaterialVis* is to provide such a visualization tool that can reveal the underlying structure and various properties of materials through several rendering modes and visualization options. In this way, we intend to provide a good material analysis tool that will be useful in a wide range of related disciplines. *MaterialVis* supports visualization of both amorphous and crystalline structures. Amorphous structures only present the topological features while crystalline structures present both topological features and defects. The structure of a material can be best visualized using surface rendering methods. The underlying surfaces of the material should be extracted and visualized. On the other hand, defects such as the disposition of some atoms, vacancies or interstitial

* Corresponding author. Tel.: +90 3122901386; fax: +90 3122664047.
E-mail address: gudukbay@cs.bilkent.edu.tr (U. Güdükbay).

http://dx.doi.org/10.1016/j.jmgm.2014.03.007
1093-3263/© 2014 Elsevier Inc. All rights reserved.

impurity atoms in the structure, cannot be visualized by simply drawing the atoms or rendering the surface of the crystal. These defects can be best visualized using *direct volume rendering* techniques. *MaterialVis* supports direct volume rendering and surface rendering, as well as combining them in the same visualization. It provides the functionality-driven visualization of the same structure with several techniques; thus it helps the user to analyze the material structure by combining the output of individual rendering modes.

We tested the tool with three real-world and seven synthetic datasets with various structural properties, sizes and defects. For instance, the sponge dataset [2] is a material produced from silicate, which has interesting nano-technological properties. Very recently, it has been experimentally shown that a silicon-rich oxide film can decay into a silicon nanowire network embedded in SiO_2 by spinodal decomposition during rapid thermal treatment [3], which has also been confirmed by accompanying kinetic Monte Carlo simulations [4]. The underlying goal in such a line of research is to achieve a nano-scale feature control and transfer it to inexpensive large-scale thin-film technology for silicon-based optoelectronics through growth kinetics. However, the direct imaging of such structures through transmission electron microscopy has not been satisfactory due to low contrast between Si and SiO_2 regions. We believe that it forms an ideal candidate for demonstrating the need for a direct volume imaging tool.

The organization of the paper is as follows. First, in Section 2 we discuss related works to address where our contribution lies within the context of existing tools and similar studies. In Section 3 we outline the general framework of *MaterialVis*, followed by two sections on the preprocessing and rendering steps. In these sections the main algorithms are presented in the form of pseudo-codes, leaving technical details to the accompanying Supplementary Materials document. Some of the capabilities of the tool are demonstrated in Section 6 using an embedded quantum dot data set. Even though our primary emphasis in *MaterialVis* is on functionality, but not the speed, nevertheless in Section 7 we provide performance benchmarks for a wide range of datasets. Finally, a brief conclusion is given.

2. Related work

There are many commercial and free crystal visualization tools. *CrystalMaker* [5], *Shape Software* [6], *XtalDraw* [7], *Vesta* [8], *Diamond* [9] and *Mercury* [10] are some examples. There are also some studies on the analysis of crystals that also provide some visualization functionality, such as the work of by Ushizima et al. [11]. These tools are essentially crystal analysis tools, which also provide some visualization functionality. Their visualization capabilities are not very advanced. They mostly offer just atom-ball models with some variations. Some of the tools support primitive surface rendering, which allows examining the crystal on the unit cell level. However, they are not sufficient to examine the underlying topology of a dataset.

There are also general visualization tools such as *AtomEye* [12], *VisIt* [13], and *XCrySDen* [14]. These tools provide sophisticated visualization capabilities but they lack the ability to create volumetric representations of materials, cannot use direct volumetric rendering techniques, and cannot quantify defects of crystal structures.

Iso-surface rendering techniques provide fast surface rendering of the volume data. They are especially useful when the surfaces are the regions of interest for the volumetric data. Doi and Koide [15] propose an efficient method for triangulating equi-valued surfaces by using tetrahedral cells based on the Marching Cubes algorithm [16].

MaterialVis is primarily based on direct volume rendering. There are mainly two types of volume data. The first type is the regular grid representation, which is widely used in medical imaging. Mostly texture-based techniques are used for the visualization of regular grids. Earlier approaches use sampling the volume along the view direction with parallel planes [17,18]. New graphics cards allow storing the volume data as 3D textures in the GPU. Ertl et al. [19,20] use a pre-integration mechanism to render the volume using 3D textures. Regular grid representation can be rendered efficiently, but the datasets using this representation are very large. The second type of data, unstructured grid representation, can be significantly compacted, so it can give much higher detail levels for the same size.

Visibility ordering is an important part of volume rendering algorithms. Cook et al. [21] and Kraus and Ertl [22] propose methods for performing visibility sorting efficiently. Shirley and Tuchman proposed a projected tetrahedra algorithm [23] for visibility sorting. Wylie et al. [24] later extend this algorithm to GPUs using vertex shaders.

Garrity [25] and Koyamada [26] use connectivity information to traverse the mesh efficiently. Weiler et al. [27] extend this approach to GPU. Callahan et al. propose a visibility ordering algorithm, *HAVS* [28], which performs an approximate sorting on the CPU and refines the sorting in the GPU. Silva et al. [29] present an extensive survey of volume rendering techniques.

3. General framework

Fig. 1 illustrates the framework of *MaterialVis* which has two main stages: *preprocessing* and *rendering*. The preprocessing stage takes the raw input and constructs the volumetric representation. For (poly)crystalline structures the preprocessing step further continues and assigns error values to atoms representing crystal defects. The rendering stage visualizes the constructed volume representation. The input reader module reads the volumetric representation and initializes the renderers. At any time, one of five renderers is selected by the user and the visualization is performed. These renderers use the OpenGL-based drawing module to display the volumetric data. The rendering tool is an interactive tool. The user interactively provides various inputs to renderers, such as camera and light information and several renderer-specific parameters.

4. Preprocessing

MaterialVis operates on a very simple input format. For amorphous materials, the types and atomic coordinates of each atom in the material is sufficient. However, for crystalline structures, the tool also requires primitive and basis vector information of the crystal structure. If this information is not readily available, our earlier work, *BilKristal* [30,31], could be utilized to extract the unit cell information from the crystal structure.

MaterialVis construct a volumetric representation using the coordinates of a set of points representing atoms in the material. There are two types of volumetric representations: *regular* and *unstructured* grids. Regular grid representation is widely used in medical imaging fields where the input data is fixed in resolution. For material visualization, interest points are the atoms; crystalline defects are attributed to them and they constitute the surface structure. Because the regular grid representation is defined independent to atoms, a fairly high grid resolution must be used in order to capture crystal defects and surface structures in high detail. On the other hand, unstructured grid representation uses atoms as vertices. Accordingly, despite using the connectivity information, the unstructured grid representation is more compact and suited

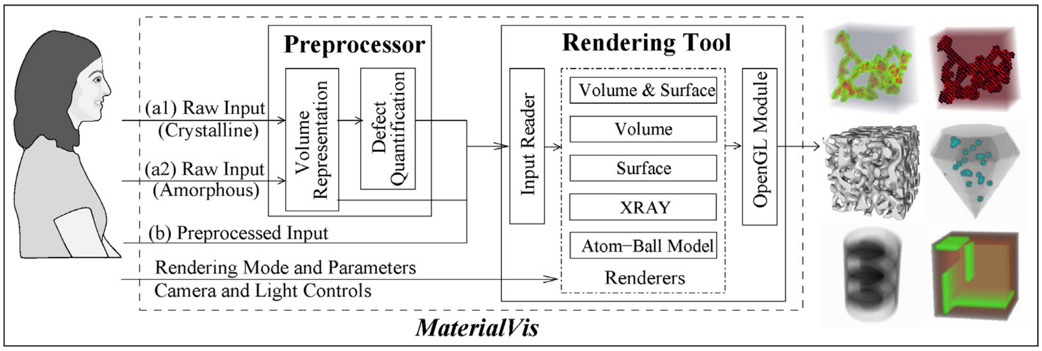


Fig. 1. The overall framework of *MaterialVis*.

better for material visualization. Because the tetrahedra are the simplest 3D primitives, we perform tetrahedralization to convert atomic coordinates into an unstructured volumetric representation.

After tetrahedralization, we extract the surface polygons of the created volume. The surface polygons are required by the surface rendering modes. *MaterialVis* focuses on visualizing crystal defects; thus, for the crystal structures the defects must be quantified for each atom in the crystal. The preprocessing stage performs these tasks and produces a data file storing the volumetric representation of the material. For crystal structures, quantified crystal defects are also included. In our experiments, we observed that the datasets with sizes up to half a million atoms could be preprocessed in less than twenty minutes. The preprocessing stage data flow is summarized in Fig. 2.

4.1. Construction of the volumetric representation

The construction of the volume representation starts with tetrahedralization of atoms. Each atom is represented as a point in 3D space. Tetrahedra cannot overlap with other tetrahedra and all parts of the volume must be covered by exactly one tetrahedra. The generated tetrahedra must be as close to a regular tetrahedron as possible (all sides are equilateral triangles) because volumes containing many sliver tetrahedra do not represent the volume accurately and may cause rendering artifacts. Delaunay tetrahedralization is the approach that generates such tetrahedra and it is the default tetrahedralization scheme in *MaterialVis* because it produces superior results. We adapt *Bowyer-Watson Delaunay triangulation* [32,33] to generate Delaunay tetrahedra. Because Delaunay tetrahedralization is not scalable for data sets containing millions of points, we devised a *pattern-based tetrahedralization* algorithm.

Our pattern-based tetrahedralization algorithm is based on the fact that the crystal structures have regular repeating patterns. The algorithm tetrahedralizes a unit cell of the crystal and searches for the occurrence of this pattern in the actual dataset containing atoms. Hence, it cannot handle arbitrarily unstructured point sets or highly deformed crystals. It does not work on amorphous materials. It can tolerate small deformations, some interstitial impurity atoms and some vacancies. It can handle cavities in the crystal structures, as long as the crystal remains as a single piece. The volumetric representation constructed by the pattern-based tetrahedralization is not as good as the one obtained by the Delaunay tetrahedralization, thus may produce inferior rendering results; but the pattern-based tetrahedralization is much faster for larger input sizes. *MaterialVis* only switches to pattern-based tetrahedralization for very large input datasets, which otherwise would take hours to pre-process. For the details of Delaunay tetrahedralization and pattern-based tetrahedralization, please refer to the supplementary materials provided online.

After the tetrahedralization, the preprocessing stage continues with surface extraction. The surface extraction process simply extracts faces of tetrahedra which are not shared by another tetrahedra. For each face, the normal values are calculated. The face normals are used in flat shading. For smooth shading, the vertex normals should be computed by averaging the normals of the faces sharing the vertex.

4.2. Quantifying crystal defects

We classify crystal defects into three groups. The first group of defects is the positional defects, which are caused by the deviation of atoms from their perfect positions relative to their neighbors. The graphite crystal with slightly shifted layers is an example. Atoms in these shifted layers have positional defects. The second group of defects is caused by vacant positions in crystals where some atoms should exist. The third group of defects is caused by extra (interstitial impurity) atoms where some foreign atoms could be found at off-lattice sites. The majority of crystal defects can be represented as one of these or a combination of them.

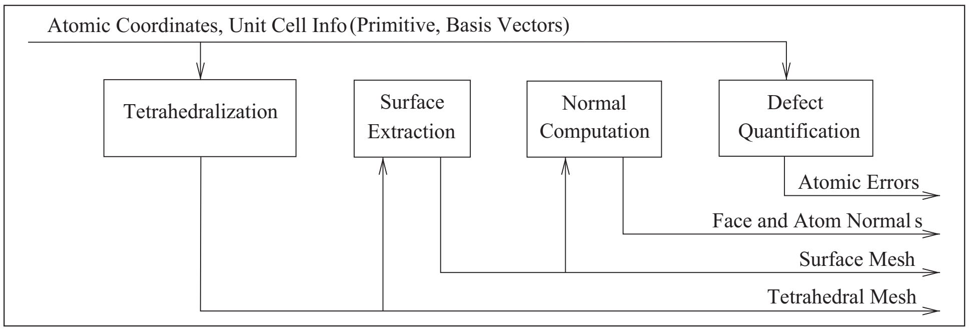


Fig. 2. The preprocessing stage data flow.

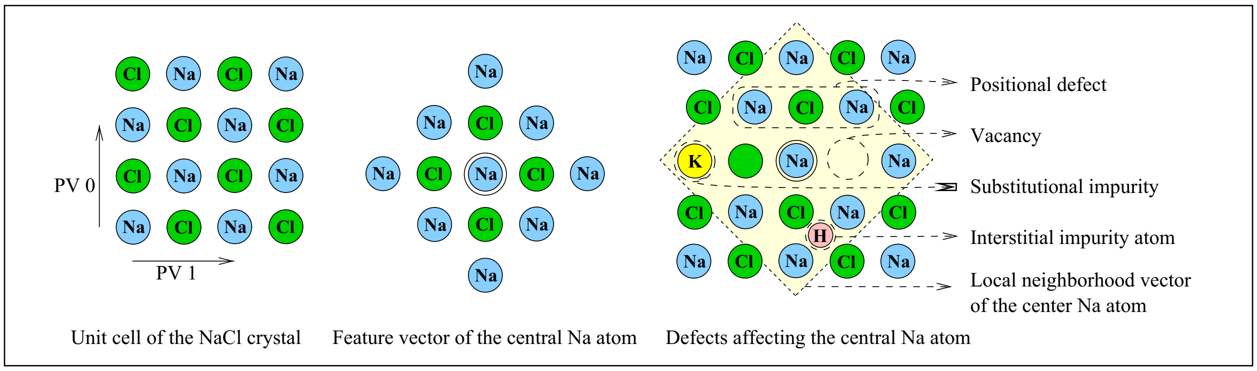

PV 0
PV 1
Cl
Na
K
H
Positional defect
Vacancy
Substitutional impurity
Interstitial impurity atom
Local neighborhood vector of the center Na atom
Unit cell of the NaCl crystal
Feature vector of the central Na atom
Defects affecting the central Na atom

Fig. 3. Illustration of the defect quantification for the *NaCl* crystal.

MaterialVis calculates defect values of atoms for each type of defect. They are calculated using the local neighborhood of atoms; any defect in the local neighborhood of an atom contributes to the atom's defect. In this way, the defects are represented and visualized properly because a large volumetric region is affected.

Fig. 3 illustrates a sample crystal structure with various defects. The unit cell and the primitive vectors of the *NaCl* crystal are shown on the left. Although there are simpler primitive vectors for the *NaCl* structure, we use the given primitive vectors for demonstration purposes. In the middle part, the feature vector of a *Na* atom is given. It includes every atom within the maximum primitive vector length distance to it in a perfect crystal. On the right part of the figure, a sample crystal segment demonstrates various types of crystal defects. The local neighborhood (the yellow background region) vector of the atom is compared with the feature vector of the atom and the error values that will be assigned to the atom are computed accordingly.

The defect quantification process is described in Algorithm 1. Defect quantification is performed for every atom in the crystal. First the local neighborhood vector (*LNV*) of the atom is extracted. *LNV* includes all the atoms within a certain distance to the atom. We used the maximum primitive vector length as the distance, however this value can be tuned by the user. Then the feature vector, which is the local neighborhood vector of the atom in a perfect crystal is computed.

Lastly, the local neighborhood and the feature vectors are compared to quantify the defect value. The comparison process matches each atom in the local neighborhood vector to its corresponding atom in the feature vector. Hence, it finds any positional differences between corresponding atoms and any vacancies or interstitial impurity atoms in the local neighborhood vector. The detailed description of the defect quantification algorithm can be found in the supplementary materials provided online.

4.3. Lossless mesh simplification

In order to capture small material features, like surface topology and crystalline defects, *MaterialVis* use highly detailed tetrahedralization where each atom is represented with a vertex. On the other hand, this representation is usually over-detailed for uniform regions in the material structures. Crystal defects constitute the volumetric features of materials for visualization purposes. *MaterialVis* aims to use volume rendering techniques to visualize such defects. Amorphous materials or perfect crystalline structures do not contain any defects; hence, their structure is mostly uniform. Moreover, many materials containing crystal defects still contain a significant portion of uniform structure. Representing such uniform regions at a low level of detail would reduce the mesh size significantly. We propose a lossless mesh simplification algorithm that would simplify the volumetrically uniform regions in the material improving the rendering performance, without affecting the surface structure and the regions bearing some crystalline defects.

The lossless mesh simplification algorithm is based on edge-collapse-based reduction techniques. This algorithm was first proposed by Hoppe [34] for triangular meshes. We extended the simplification algorithm to tetrahedral meshes [35]. Edge-collapse technique works by repeatedly collapsing edges into new vertices. An edge-collapse would eliminate tetrahedra using the collapsed edge and stretch the tetrahedra using only one vertex of the collapsed edge. We specify the constraints for selecting the edges to collapse in such a way to ensure lossless compression. The details are given in Algorithm 2. In order to preserve surface details, no surface edge can be collapsed. Also, an edge with a vertex on the surface can only be collapsed onto the surface vertex. After an edge collapse, various tetrahedra are affected by either being deleted or being stretched. If any of these affected tetrahedra contain an atom with a non-zero defect value, the edge is not collapsed because it will modify the visual output. The simplification ratio depends highly on the dataset. With the test datasets we used, we achieved simplification ratios of up to 30% of the original size. The detailed description of the lossless mesh simplification algorithm can be found in the supplementary materials provided online.

5. Rendering

MaterialVis provides rendering functionality with various modes and display options, such as lighting and cut-planes. It utilizes graphics acceleration via *OpenGL* graphics application programming interface (API). The rendering tool supports five modes: volume and surface rendering, volume rendering, surface rendering, XRAY rendering, and atom-ball model rendering. Each rendering mode is useful for some aspect of material analysis. A user-friendly graphical interface is provided, allowing users to control the tool easily. For detailed explanation about features and functionalities of the *MaterialVis* tool, please refer to the users manual provided online.

5.1. Volume and Surface Rendering

Volume and surface rendering aims to visualize both the material topology and the crystal defects. It is the slowest but most flexible rendering mode. The user can set many properties of the visualization. The volume rendering is based on the cell-projection algorithm that we used in our earlier work [35]. We extended the mentioned algorithm to handle surfaces. We selected the

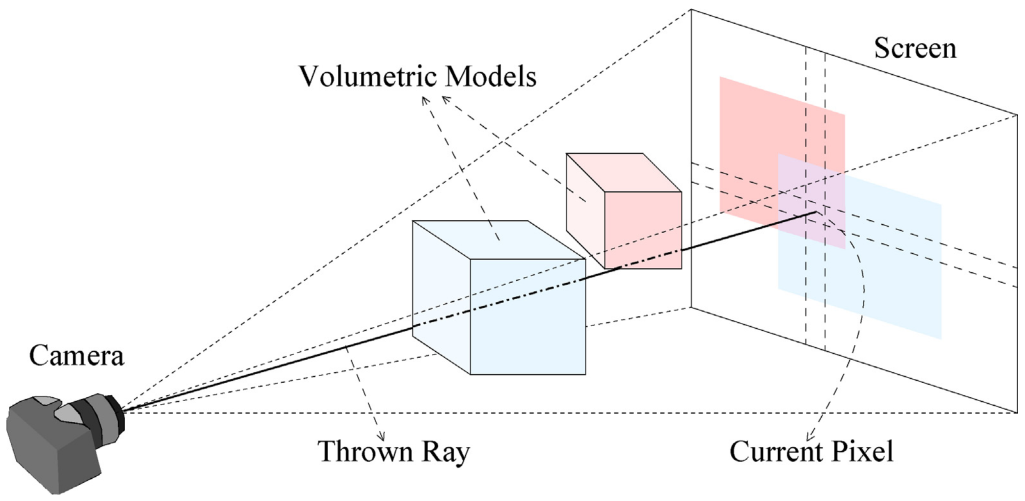


Fig. 4. The raycasting framework.

cell-projection algorithm for several reasons. First of all, cell-projection is a very robust and flexible algorithm. It can be modified to support advanced features easily. It does not require any auxiliary data such as neighboring information. Its execution flow and memory access patterns are mostly uniform, making it ideal for parallel implementations [36]. Our implementation utilizes multi-core CPU hardware. We can achieve almost linear speed-ups [36]; i.e., 3.0- to 3.5-fold speed-ups for quad-core CPUs.

We decided not to use GPU-based implementation for two reasons. First, the conventional GPU based volume rendering algorithms, albeit being fast, cannot support features, such as surface processing, multi-variable visualization, advanced transfer functions, because they rely on limited shader programming techniques. Secondly, although the CUDA or OpenCL based GPU implementations are capable to support required features, they are not very robust and they are highly hardware dependent.

The cell projection algorithm is a ray-casting-based rendering technique. Fig. 4 demonstrates the processing of a single pixel. The visualization parameters are the camera position, orientation and the projection angle. A ray is cast for every pixel on the screen image, traveling the volume and hitting the center of the pixel. The ray starts with full intensity. While the ray traverses the volume, its color is affected by the volume it visited and its intensity is reduced. The final color that the ray assumes after exiting the volume defines the pixel color. Algorithm 3 presents our version of the cell-projection algorithm.

The cell-projection algorithm projects each tetrahedron and face onto the image as the first step. All the pixels that lie under the projections of each face and tetrahedra are found and associated with those faces and tetrahedra. The algorithm constructs the image pixel by pixel. First, the list of tetrahedra and faces associated with the current pixel are extracted. Then intersection contributions are calculated for each face or tetrahedra in the list. While calculating the contributions, tetrahedra and face intersections are treated differently. The intersection contribution structure contains two pieces of data. The first one is the camera distance to the entry point of the tetrahedron or the face which is used in visibility sorting of intersection records. The second piece of data is the color and intensity of a full intensity ray that travels through the tetrahedron or the face.

After the intersection contributions are computed, the results are sorted according to the camera distance. Then starting from near to far, the intersection contributions are composited into a single intensity value, which is assigned as the pixel color.

The calculation of tetrahedron intersection contributions starts by finding the entry and exit points of the ray on the tetrahedron (cf. Fig. 5(a)). It takes several samples on the line segment between the entry and exit points. The color and transparency of each sample is calculated by interpolation. The sampled colors are combined into a single color. While combining the colors, front-to-back alpha-blending is used and the alpha channel value is corrected for each sample. The contribution of each color is proportional to the segment length of the sample. The larger the tetrahedron, the higher its contribution will be. The remaining light intensity is directly proportional to the contribution. For example, for a fully-opaque volume, only the entry color matters because the ray will lose all of its intensity at the beginning.

Volumetric features are generally revealed by the use of appropriate transfer functions. The transfer functions are simply mapping functions that compute the color and intensity values for each set of attributes. They are very critical for the perception. The transfer function should be defined in a way to highlight the features of prime interest. Defects in crystal structures can be an example of such interested features. Usually, interesting features are present in a small fraction of the volume data. In that case, very transparent colors should be assigned to the attributes that one is not interested in and a range of relatively opaque colors should be assigned to interesting features. Thus, the interesting features can be visualized in high detail while the other parts are barely represented. Although general principles can be laid out easily, defining good transfer functions is an important research area.

MaterialVis uses a simple but flexible approach for defining the transfer function. The colors of vertices are determined by the defects associated with the atom defining the vertex. The quantified defect values of an atom a are converted into color values using the defect parameters of the atom as follows:

$$\begin{aligned} a.Color &= BaseColor + a.positionalDefect \times PositionalDefectColor \\ &\times PositionalDefectMultiplier + a.extraAtomDefect \\ &\times ExtraAtomColor \times ExtraAtomMultiplier + a.vacancyDefect \\ &\times VacancyColor \times VacancyDefectMultiplier \end{aligned}$$

The color and error multipliers used in the equation are tunable by the user. The face intersections are used to handle the effects of the surface. The calculation of the face intersection contributions

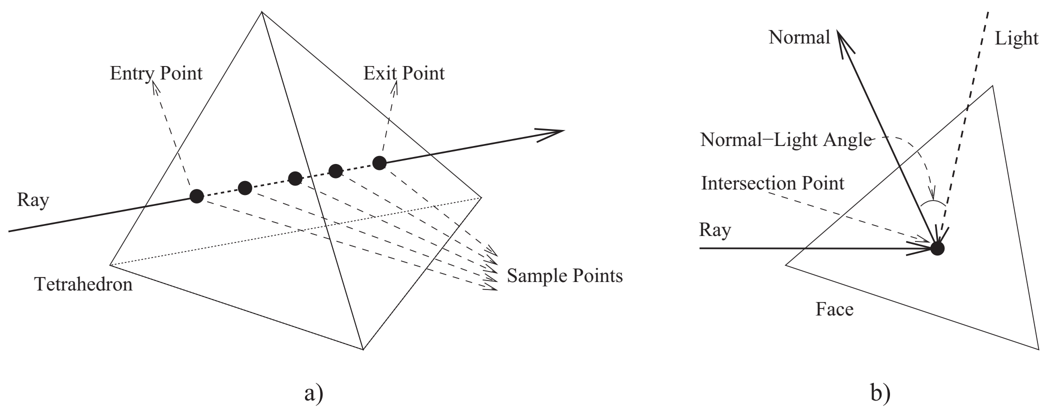


Fig. 5. (a) Tetrahedron-ray intersection and sample points, and (b) face-ray intersection and normal-light angle.

handles the lighting effects that are missing in pure volume renderers. The color and transparency of the faces and the lighting parameters are tunable by the user.

Lighting effects underline the surface structure without hiding the volume visualization. The face intersection contribution calculation starts by finding the intersection point between the face and the ray. The distance from the camera to the intersection point is computed. The color of intersection is computed using interpolation of the colors of face vertices. The normal for the intersection point is calculated. If the shading mode is flat, than the face normal is used. If shading mode is smooth, the vertex normals are interpolated. Fig. 5(b) demonstrates the face ray intersection and the light-normal angle.

We use Phong illumination model for this rendering mode because the specification of an excessive number of lighting parameters used by complex illumination models puts the burden on the user. The main focus in this rendering mode is still the volume rendering part; hence, a simpler lighting model works well and is more user-friendly. More detailed explanations about volume and surface rendering algorithm can be found in the supplementary materials provided online.

Fig. 6 shows the visualization of some material datasets using this mode. We tuned the rendering parameters to focus on the defects in the crystal volume and the surface related parameters to give an impression of the structure itself but not overwhelm the volume visualization. Since the volume and surface rendering mode is flexible, the user can visualize the material in various ways and analyze various aspects of the data efficiently.

5.2. Volume rendering

Volume rendering aims to visualize the defects in the crystal. Since surfaces are not represented, it gives only a very rough idea about the topology of the material. We use Hardware Assisted Visibility Sorting (*HAVS*) for volume rendering [28]. The algorithm performs some of the computations and rendering on the graphics hardware; hence, it is partially GPU accelerated. It is not as fast as surface rendering. Fig. 7 presents the visualization of some datasets with this mode.

The high performance of the *HAVS* algorithm is due to its use of the graphics hardware. The algorithm converts the volume rendering problem into a simpler version that can be solved on the GPU. Although this approach is fast, it also has drawbacks. The first problem is in visibility sorting. *HAVS* performs a rough but fast visibility sorting on the CPU, which may have errors. The algorithm relies on a shader program running in the GPU to correct these errors before rendering. Due to the limitations in the graphics hardware, all of the errors might not be corrected, leading to visual artifacts. This situation is very particular for irregular tetrahedralizations. Luckily, material structures have fairly regular tetrahedralization, thus *HAVS* work well with *MaterialVis*.

The second problem is the limitations on color computations. *HAVS* use a pre-integration table in terms of 3D textures to compute the contributions of tetrahedra. This brings a restriction on color computations so that the visualization attribute of the volume, the quantified defect value in our case, can only be a scalar. In the defect quantification stage, we assign three defect attributes

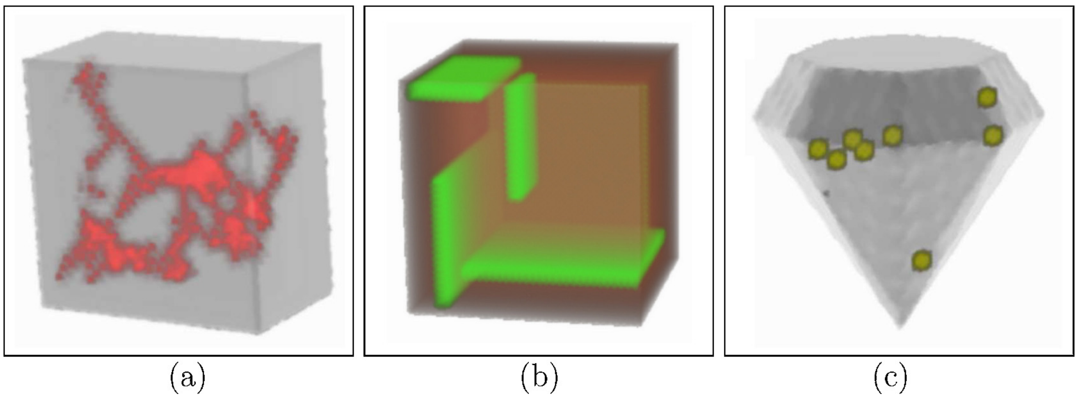


Fig. 6. Rendered images of various dataset: (a) NaCl cracked, (b) Cu line defect, (c) A centers (substitutional nitrogen-pair defects) in diamond [42].

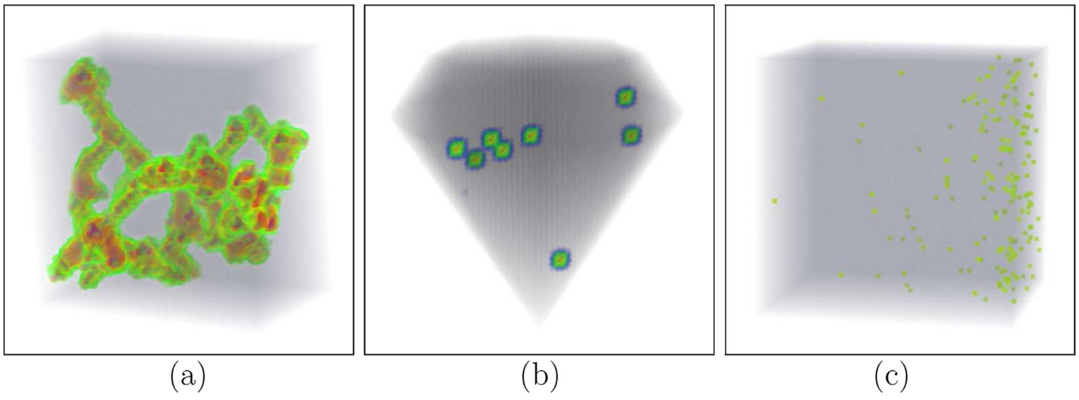


Fig. 7. Volume rendering mode: (a) NaCl cracked, (b) A centers (substitutional nitrogen-pair defects) in diamond, (c) Palladium with hydrogen.

to each vertex: *positional*, *vacancy*, and *extra (interstitial impurity) atom defects*. *HAVS* cannot handle three attributes; thus these defect values must be merged as a single defect. We compute a weighted sum using the user-specified weights: *positional defect multiplier*, *extra atom multiplier*, and *vacancy defect multiplier*. We calculate the scalar defect value of atom *a* using the defect parameters of the atom as follows:

$$\begin{aligned} a.scalar &= a.positionalDefect \times PositionalDefectMultiplier \\ &+ a.extraAtomDefect \times ExtraAtomMultiplier \\ &+ a.vacancyDefect \times VacancyDefectMultiplier \end{aligned}$$

After all defect values are computed, they are normalized to the range [0, 1].

The scalar-to-color conversions are performed using a simple color map specified by the user. The color map is a set of entries mapping a certain scalar value to a certain color and intensity. The colors and intensities of intermediate scalar values are found using linear interpolation between the color map entries. Fig. 8 shows a sample color map where five entries are defined and the whole scalar range is computed from these entries. The example map focuses on the scalar range [0.4, 0.6]; thus, it can distinguish scalar values in this range much better than the other parts.

5.3. Surface rendering

Surface rendering aims to visualize the topological structure of the material and is suited to visualize datasets with an underlying topological structure. The sponge dataset is one example. Fig. 9(a) presents the rendered output of sponge dataset with this mode. For regular datasets without any specific shape, this mode cannot provide much information.

We can easily render the surface of the material because the surface data is present in the volume representation. Cut-planes change the surface structure but with the surface reconstruction algorithm, the current surface data is maintained. The rendering is performed using *OpenGL* rendering functionality. The triangular mesh that represents the surface is rendered by *OpenGL* directly.

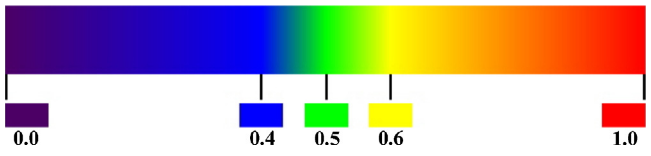


Fig. 8. An example color map. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Vertex or face normals are fed to the shaders, depending on the selected shading model being *smooth* or *flat*, respectively. The color and the shininess of the surface material can be specified by the user. Because surface data is directly rendered with *OpenGL*, surface rendering is GPU accelerated. The surface data is only a small portion of the volume data; hence, surface rendering is a fast rendering mode, compared to the other rendering modes.

5.4. XRAY rendering

XRAY rendering mode can be considered as a simplified volume visualization technique. Its output resembles the XRAY images, hence it is named after it. Fig. 9(b) presents a material rendered in this mode. This rendering mode is particularly useful for visualizing the internal structure of crystals. It is aimed to fill a small gap that other rendering modes cannot address well. XRAY rendering mode does not visualize the errors in the structure of a crystal. Similar to the surface rendering mode, it focuses on the topology. However, unlike the surface rendering mode, it does not just visualize the outer surface but visualizes the volume.

The algorithm is a simplified version of volume and surface rendering algorithm. Basically, for each thrown ray, the faces it intersects with are found and sorted with intersection order. The odd numbered faces would be the entry faces, where ray enters inside the material and even numbered faces would be the exit faces. These faces are used to calculate the distance that the ray travels inside the material. The calculated distance is then used as the opacity coefficient for the pixel that ray is thrown for. Because the algorithm uses surface polygons to visualize the volume, the input size is much smaller than the modes that use tetrahedra. This mode is relatively fast even though the implementation is not GPU accelerated.

5.5. Atom-ball model rendering

Atom-ball model rendering mode visualizes the material as a group of atoms. It does not consider the volumetric properties and the surface structure of the material. This mode is useful to understand the relations between atoms and to examine small datasets. It is the only mode that distinguishes between different types of atoms in the material because it treats the material as a set of atoms, rather than as a volume or a surface. Atoms are drawn as spheres. The user can set the colors of each atom type. The atom radii given in the input file are used as the radii of the spheres representing atoms. However, the user is allowed to set a parameter, which scales down the radii. In this way, the user can visualize the crystal with actual atom radii in a very compact form, or scale down

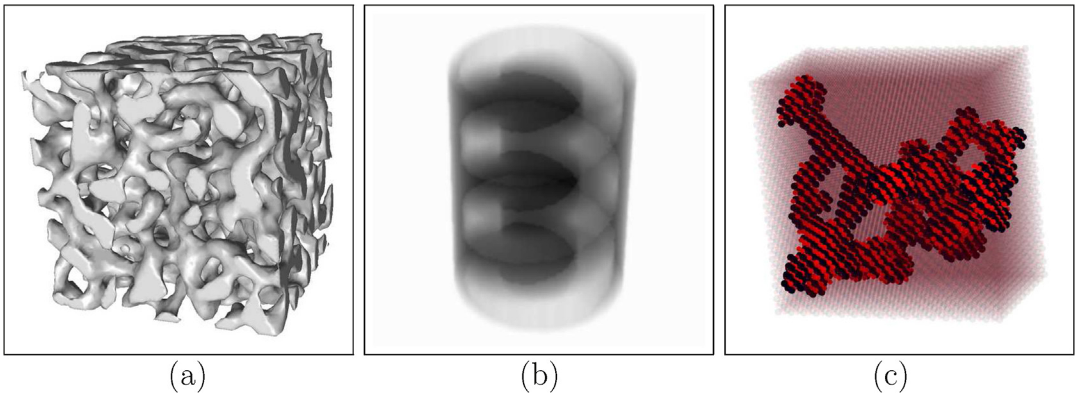


Fig. 9. Examples: (a) Surface rendering mode – Sponge dataset, (b) XRAY rendering mode – $CaCuO_2$ spiral dataset, (c) Atom-ball model rendering mode – NaCl cracked dataset.

the radii to obtain a spacious version where individual atoms can be distinguished easily.

Atom-ball model rendering can visualize the crystal defects in a restricted way. The user can set the transparency of atoms that do not contain any defects, which makes the atoms with defects distinguished easily. However, this mode cannot help to assess the magnitude of defects and differentiate different defect classes. Fig. 9(c) depicts the visualization of NaCl cracked dataset with this rendering mode.

The rendering is done using basic *OpenGL* functionality to draw spheres representing atoms. However, in order to handle transparency, the atoms should be sorted in visibility order. This mode is also GPU accelerated; it is a fast mode and can be used interactively.

6. Demonstration: embedded quantum dot datasets

In order to demonstrate the usage and various capabilities of *MaterialVis*, we describe the steps of how we have used the tool for the structural analysis of two real-world quantum dot datasets that we have been working on. Quantum dots are semiconductors with built-in structural irregularities. Such irregularities provide the semiconductor unique electrical properties. Quantum dots have possible uses in various areas such as quantum computing, solar cells, medical imaging, LEDs and transistors. Biasiol and Heun [37] and Ulloa et al. [38] present in-depth information about the structure and physical properties of quantum dots.

We used two InGaAs type quantum dot datasets, one with random alloying among the cations and one without. The base semiconductor is the GaAs compound. The quantum dot is grown layer by layer. The atoms belonging to each layer are deposited onto existing layers. Deposited atoms use the existing lattice structure to bind. When the quantum dot layers are to be grown, indium atoms are deposited instead of gallium atoms at certain regions. Although the indium atoms are larger than the gallium atoms, they still fill the binding sites for gallium atoms. The resulting crystal structure becomes highly stressed. Eventually, indium atoms cause deformations in the crystal structure, relaxing to stable positions. The crystal regions with such deformations have significantly different electrical properties. By managing the deposition of indium atoms, building quantum dots with various shapes and properties is possible.

Both of the quantum dot datasets contain just under 1.5 million atoms. Due to the deformations in the crystal structure, pattern-based tetrahedralization cannot be used for quantum dot datasets. They must be treated as amorphous materials where Delaunay tetrahedralization must be used; hence, it is crucial to keep input sizes low. However, in order to simplify our task, we can mask the Arsenic atoms from the dataset. Arsenic is the common atom that is found throughout the whole material more or less homogeneously. What we are really interested in is the distribution of gallium and indium atoms. If Arsenic atoms are included, they will have significant effect on the volume visualization, reducing the effects of interested properties of the material. Secondly, masking the Arsenic atoms reduces the size of the datasets significantly. This helps to keep pre-processing times low.

We can also employ another input simplification technique. Volume rendering techniques mainly visualize the gallium and indium distributions in the material. It does not depend on the density of atoms in a certain region. For example, in InGaAs quantum dots, certain parts of the material will be made of just regular GaAs alloy and certain parts will be made of just InAs alloy. Because we masked the Arsenic atoms, those parts will be composed of just single type of atoms. For volume rendering purposes, it does not matter if we represent such regions with all the atoms or just a fraction of them; hence, we can reduce the input size significantly.

We employed a simple data size reduction technique. First, we included the atoms belonging to the surface of the material. Because our datasets have rectangular prism shape, determining the boundary atoms was straightforward. Secondly, we uniformly sampled the whole material and included the sampled atoms, which helps to keep the tetrahedralization regular. Finally, we included every atom that has another atom of different type within a certain distance. With this technique, we can capture the regions with gallium–indium transitions with high detail. We also reduced the sizes of our two datasets to 5.8% and 8.5% to their original sizes, without losing any information regarding the visualization.

The next step is scalar assignment. Because we are only interested in gallium–indium transitions, we assigned *0.0* to gallium atoms and *1.0* to indium atoms. However, users can assign any scalar values depending on the properties they want to visualize. After scalar assignments, the datasets are ready to be pre-processed. Because the data sizes are kept low, pre-processing takes just a few minutes. After pre-processing, we tuned the rendering parameters. We used volume and surface rendering. We set the surface lighting parameters so that the material surfaces are just identifiable. We assigned a green, high transparency color as the base color. This color represents the gallium atoms bearing *0.0* scalar value. The scalar values are used as the positional defect. We used a high opacity red color to positional defect. Accordingly, we observed the indium atoms in red. Fig. 10 depicts the rendered images of our sample datasets.

7. Benchmarks

Minimum hardware requirements of *MaterialVis* are rather modest. We tested the tool without any problems on various low

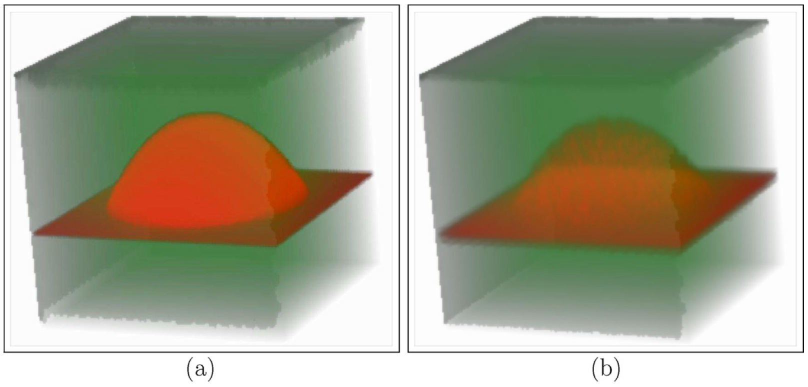


Fig. 10. *InGaAs* quantum dots: (a) without random alloying, (b) with random alloying. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 1
Preprocessing and rendering times of each dataset (in ms).

	Number of atoms	Pre-processing	Volume and surface rendering	Volume rendering	Surface rendering	XRAY rendering	Atom-ball rendering
NaCl cracked	25,725	9254	707	66	11	275	28
Cu line defect	173,677	171,559	1329	269	15	302	187
Diamond vacancy defect	44,982	17,824	891	109	14	266	49
A Centers (Substitutional Nitrogen-pair Defects) in Diamond	45,005	18,072	879	112	15	265	49
Graphite slided	66,576	140,563	1405	138	13	284	72
Palladium with hydrogen	137,549	103,399	1471	254	14	298	148
$CaCuO_2$ spiral	199,764	114,221	1484	305	16	337	216
Sponge	534,841	602,869	2748	1015	22	471	578
Quantum dot without random alloying	86,338	84,376	611	145	16	175	114
Quantum dot with random alloying	125,595	1,161,757	730	196	16	182	181

end computers. On the other hand, the rendering times heavily depend on available computational power. The performance of the *volume and surface rendering* and the *XRAY rendering* modes depend on the CPU power. They can also benefit from multi-core CPUs. Other rendering modes are GPU bound modes; high-end graphics cards will increase the performance significantly. The minimal configuration should have a graphics card with *OpenGL 1.5* support. Stand-alone graphics cards with private memory is recommended. Memory requirements heavily depend on the input size. In our tests, we barely reached 1GB of memory usage. A standard personal computer with a stand-alone graphics card could run the tool without any significant latency.

We tested the tool with various datasets. In the sponge dataset [2], which was already mentioned in the introduction section, we tackled the volumetric imaging of a highly complicated structure. In the dataset we used, the stoichiometry of SiO_x was fixed to $x = 1$, i.e., SiO by setting the silicon excess to 30 vol.%. There are more than half a million atoms in total.

The quantum dot represents a self-assembled InGaAs quantum dot embedded in a GaAs matrix. It contains a lens-shaped quantum dot placed on an InAs half-monolayer-thick wetting layer. The random alloy variant has 20% indium and 80% gallium compositional alloying between the cation atoms. Both structures are first prepared in the zinc blende sites of the GaAs crystal, followed by strain relaxation using molecular statics as implemented in the LAMMPS code [39]. Here, the interatomic force fields are described by the Abell-Tersoff potentials [40,41]. The sponge and quantum dot datasets are real-world datasets that are researched actively.

The *NaCl Cracked* dataset represent a *NaCl* crystal with some positional defects. The atoms with defects represent a crack. The datasets *Cu Line Defect*, *Diamond Vacancy Defect*, *A Centers (Substitutional Nitrogen-pair Defects) in Diamond*, and *Graphite Slided* represent crystals with some well-known defects. The *Palladium with Hydrogen* dataset represents a block of palladium metal absorbing hydrogen from one of its faces. The *$CaCuO_2$ Spiral* dataset presents a cylinder-shaped crystal with a spiral sculptured from inside. These datasets are synthetic datasets and they are specifically designed to showcase various crystal defects and interesting topological structures using the features and capabilities of our rendering tool.

Table 1 presents the preprocessing and rendering times of each dataset on a middle-end PC with 3.2 GHz quad-core CPU and nVidia GTX560 GPU. The longest preprocessing time is less than 20 min. Despite the high computational cost of volume and surface rendering mode, the highest rendering time is 2.7 s for tested datasets. With other rendering modes, interactive rendering rates were achieved for all tested datasets.

8. Conclusions

MaterialVis is a functional visualization tool, which can easily process million-atom datasets. It supports many rendering modes to accentuate both the topology and the defects within the nanostructures. What distinguishes *MaterialVis* from other visualization tools is that it can handle the materials as a volume or a surface manifold, as well as a set of atoms. We believe that *MaterialVis* will be an instrumental software for crystallographers, polymer and macromolecule researchers, solid state physicists, or more generally material scientists in need to analyze nanostructures embedded within a matrix of atoms. Although only a small part of its visualization capabilities could be demonstrated throughout this work, the user can easily tune the rendering parameters with the user-friendly interface to obtain custom visual representations of materials. The tool with source codes, executables, datasets and user manual is provided as a supplementary material for online publication.

Algorithm 1. Defect quantification algorithm.

```
DefectQuantification(Atoms $A$)
begin
  foreach (Atom $a$ in $A$) do
    //Extract all atoms within a certain distance to atom $a$
    $LNV$=extractLocalNeighborhoodVector($a$);
    //Extract all atoms within a certain distance to atom $a$
  in a perfect crystal
    $FV$=computeFeatureVector($a.type$);
    //Assign defect upon feature comparisons
    $a$.defect=compareFeatures($FV$, $LNV$);
end
```

Algorithm 2. Lossless mesh simplification algorithm.

```
LosslessMeshSimplification(Atoms $A$, Tetrahedra $T$)
begin
  //Extract and sort all non-surface edges with no defect
  $EdgeList$=ExtractEdgeList($T$);
  while $EdgeList$ is not empty do
    $e$=$EdgeList$.getShortestEdge()
    if No tetrahedron with a vertex having non-zero defect will be affected
  from the collapse of edge $e$ then
      //Collapse edge $e$ into newly created vertex $v'$
      $v'$=collapse($e$);
      //Delete tetrahedra that use edge $e$ and update
  tetrahedra that use a vertex of edge $e$ to use $v'$ instead
      UpdateTetrahedra($T$, $e$, $v'$);
      //Update the edge list upon tetrahedral changes
      UpdateEdgeList($EdgeList$, $e$, $v'$);
end
```

Algorithm 3. The cell-projection algorithm.

```
VolumeAndSurfaceRenderer()
begin
  //Associate the tetrahedra and the faces with the pixels
  that they are projected onto
  ProjectTetrahedraOntoImageSpace();
  ProjectFacesOntoImageSpace();
  //Process pixel by pixel
  foreach Pixel $p$ do
    //Extract the faces and tetrahedra that are projected
  upon $p$
    $list$=getProjectedFacesAndTetrahedra($p$);
    foreach Face or Tetrahedra $fot$ in $list$ do
      //Compute the contibution of $fot$ on the ray cast from $p$
      CalculateIntersectionContributions($fot$,$p$);
    SortByEyeDistance($list$);
    $p$.color={0,0,0,0};
    //Combine the intersection contributions with alpha
  blending and alpha correction to compute $p$'s color
    foreach Face or Tetrahedra $fot$ in $list$ do
      CompositeColor($p$.color,$fot$);
end
```

Appendix A. Supplementary Data

Supplementary data associated with this article can be found, in the online version, at http://dx.doi.org/10.1016/j.jmgm.2014.03.007.

References

[1] B. Gault, M.P. Moody, J.M. Cairney, S.P. Ringer, Atom probe crystallography, Mater. Today 15 (9) (2012) 378–386.
[2] J. Kelling, G. Ódor, M.F. Nagy, H. Schulz, K.-H. Heinig, Comparison of different parallel implementations of the 2+1-dimensional KPZ model and the 3-dimensional KMC model, Eur. Phys. J. – Special Top. 210 (1) (2012) 175–187.
[3] D. Friedrich, B. Schmidt, K.H. Heinig, B. Liedke, A. Mücklich, R. Hübner, D. Wolf, S. Kölling, T. Mikolajick, Sponge-like Si–SiO_2 nanocomposite-morphology studies of spinodally decomposed silicon-rich oxide, Appl. Phys. Lett. 103 (13) (2013) (Article no. 131911).
[4] B. Liedke, K.-H. Heinig, A. Mcklich, B. Schmidt, Formation and coarsening of sponge-like Si-SiO_2 nanocomposites, Appl. Phys. Lett. 103 (13) (2013) (Article no. 133106).
[5] CrystalMaker Software Ltd., CrystalMaker, 2013 http://www.crystalmaker.com/crystalmaker/index.html
[6] Shape Software, Shape Software, 2012 http://www.shapesoftware.com/00_Website_Homepage
[7] R.T. Downs, et al., XtalDraw, 2004 http://www.geo.arizona.edu/xtal/group/software.htm
[8] K. Momma, VESTA – JP-Minerals, 2011 http://jp-minerals.org/vesta/en
[9] Crystal Impact, Diamond crystal and molecular structure visualization, 2012 http://www.crystalimpact.com/diamond
[10] C.F. Macrae, P.R. Edgington, P. McCabe, E. Pidcock, G.P. Shields, R. Taylor, M. Towler, J. van de Streek, Mercury: visualization and analysis of crystal structures, J. Appl. Crystallogr. 39 (3) (2006) 453–457.
[11] D. Ushizima, D. Morozov, G.H. Weber, A.G. Bianchi, J.A. Sethian, E.W. Bethel, Augmented topological descriptors of pore networks for material science, IEEE Trans. Visual. Comput. Graph. 18 (12) (2012) 2041–2050.
[12] J. Li, AtomEye: An efficient atomistic configuration viewer, Model. Simulat. Mater. Sci. Eng. 11 (2) (2003) 173–177.
[13] Dept. of Energy and Advanced Simulation and Computing Initiative, VisIt, 2013 https://wci.llnl.gov/codes/visit/home.html
[14] A. Kokalj, XCrySDen – a new program for displaying crystalline structures and electron densities, J. Mol. Graph. Model. 17 (3-4) (1999) 176–179.
[15] A. Doi, A. Koide, An efficient method of triangulating equi-valued surface by using tetrahedral cells, IEICE Trans. Inf. Syst. E74-D (1) (1991) 214–224.
[16] W.E. Lorensen, H.E. Cline, Marching cubes: A high resolution 3D surface construction algorithm, in: Proceedings of ACM SIGGRAPH'87, ACM, New York, NY, USA, 1987, pp. 163–169.
[17] A.E. Lefohn, J.M. Kniss, C.D. Hansen, R.T. Whitaker, A streaming narrow-band algorithm: interactive computation and visualization of level sets, IEEE Trans. Visual. Comput. Graph. 10 (4) (2004) 422–433.
[18] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, T. Ertl, Interactive volume on standard PC graphics hardware using multi-textures and multi-stage rasterization, in: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware, HWWS'00, ACM, New York, NY, USA, 2000, pp. 109–118.
[19] K. Engel, M. Kraus, T. Ertl, High-quality pre-integrated volume rendering using hardware-accelerated pixel shading, in: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware, HWWS'01, 2001, pp. 9–16.
[20] S. Roettger, S. Guthe, D. Weiskopf, T. Ertl, W. Strasser, Smart hardware-accelerated volume rendering, in: Proceedings of the Symposium on Data Visualisation, VISSYM'03, 2003, pp. 231–238.
[21] M. Kraus, T. Ertl, Cell-projection of cyclic meshes, in: Proceedings of IEEE Visualization, 2001, pp. 215–559.
[22] R. Cook, N.L. Max, C.T. Silva, P.L. Williams, Image-space visibility ordering for cell projection volume rendering of unstructured data, IEEE Trans. Visual. Comput. Graph. 10 (6) (2004) 695–707.
[23] P. Shirley, A. Tuchman, A polygonal approximation to direct scalar volume rendering, Proc. Workshop Vol. Visual. 24 (5) (1990) 63–70.
[24] B. Wylie, K. Moreland, L.A. Fisk, P. Crossno, Tetrahedral projection using vertex shaders, in: Proceedings of the IEEE Symposium on Volume Visualization and Graphics, VVS'02, 2002, pp. 7–12.
[25] M.P. Garrity, Raytracing irregular volume data, Proc. Workshop Vol. Visual., ACM SIGGRAPH Comput. Graph. 24 (5) (1990) 35–40.
[26] K. Koyamada, Fast traversal of irregular volumes., in: Visual Computing – Integrating Computer Graphics with Computer Vision, Springer, Berlin, 1992, pp. 295–312.
[27] M. Weiler, M. Kraus, M. Merz, T. Ertl, Hardware-based ray casting for tetrahedral meshes, in: Proceedings of the 14th IEEE Visualization, VIS'03, IEEE Computer Society, 2003, pp. 333–340.
[28] S.P. Callahan, M. Ikits, J.L.D. Comba, C.T. Silva, Hardware-assisted visibility sorting for unstructured volume rendering, IEEE Trans. Visual. Comput. Graph. 11 (3) (2005) 285–295.
[29] C.T. Silva, J.L.D. Comba, S.P. Callahan, F.F. Bernardon, A survey of GPU-based volume rendering of unstructured grids, RITA 12 (2) (2005) 9–30.
[30] E. Okuyan, U. Güdükbay, O. Gülseren, Pattern information extraction from crystal structures, Comput. Phys. Commun. 176 (7) (2007) 486–506.
[31] E. Okuyan, U. Güdükbay, BilKristal 2.0: a tool for pattern information extraction from crystal structures, Comput. Phys. Commun. 185 (1) (2014) 442–443.
[32] A. Bowyer, Computing Dirichlet tessellations, Comput. J. 24 (2) (1981) 162–166.
[33] D.F. Watson, Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes, Comput. J. 24 (2) (1981) 167–172.
[34] H. Hoppe, View-dependent refinement of progressive meshes, in: Proceedings of ACM SIGGRAPH, 1997, pp. 189–198.
[35] E. Okuyan, U. Güdükbay, V. İşler, Dynamic view-dependent visualization of unstructured tetrahedral volumetric meshes, J. Visual. 15 (2012) 167–178.
[36] E. Okuyan, U. Güdükbay, Direct volume rendering of unstructured tetrahedral meshes using CUDA and OpenMP, J. Supercomput. 67 (2) (2014) 324–344.
[37] G. Biasiol, S. Heun, Compositional mapping of semiconductor quantum dots and rings, Phys. Rep. 500 (4–5) (2011) 117–173.

[38] J. Ulloa, P. Offermans, P. Koenraad, InAs quantum dot formation studied at the atomic scale by cross-sectional scanning tunnelling microscopy, in: M. Henini (Ed.), Handbook of Self Assembled Semiconductor Nanostructures for Novel Devices in Photonics and Electronics, Elsevier, Amsterdam, 2008, pp. 165–200.
[39] S. Plimpton, Fast parallel algorithms for short-range molecular dynamics, J. Comput. Phys. 117 (1995) 1–19.
[40] D. Powell, M. Migliorato, A. Cullis, Optimized tersoff potential parameters for tetrahedrally bonded III–V semiconductors, Phys. Rev. B 75 (11) (2007) (Article no. 115202).
[41] C. Bulutay, Quadrupolar spectra of nuclear spins in strained In_xGa_{1-x} as quantum dots, Phys. Rev. B 85 (2012) (Article no. 115313).
[42] G. Davies, The A nitrogen aggregate in diamond – its symmetry and possible structure, J. Phys. C: Solid State Phys. 9 (1976) L537–L542.