Groeber and Jackson *Integrating Materials and Manufacturing Innovation* 2014, **3**:5
http://www.immijournal.com/content/3/1/5

Integrating Materials and Manufacturing Innovation
a SpringerOpen Journal

RESEARCH Open Access

DREAM.3D: A Digital Representation Environment for the Analysis of Microstructure in 3D

Michael A Groeber[1]* and Michael A Jackson[2]*

*Correspondence:
michael.groeber@wpafb.af.mil;
mike.jackson@bluequartz.net
[1] Air Force Research Laboratory, 2230 Tenth St, 45433, WPAFB, Ohio, USA
[2] BlueQuartz Software, 400 S. Pioneer Blvd, 45066, Springboro, OH, USA

Abstract

This paper presents a software environment for processing, segmenting, quantifying, representing and manipulating digital microstructure data. The paper discusses the approach to building a generalized representation strategy for digital microstructures and the barriers encountered when trying to integrate a set of existing software tools to create an expandable codebase.

Keywords: Electron back-scatter diffraction; Synthetic microstructure; HDF5; 3D Microstructure reconstruction; Programming library; Open-source

Background

In recent years, two major initiatives have been introduced that promise to affect how the materials science community integrates with the larger system design process. These initiatives, known as Integrated Computational Materials Engineering (ICME) and the Materials Genome Initiative (MGI), are built on the ability to represent materials digitally, both in a structural and performance context. Under the ICME construct [1], materials engineering can be treated as a series of models (empirical or physical) that link a processing history to a suite of properties (mechanical, optical, electromagnetic, etc.). In the most general terms, processing models predict the internal structure of materials under some processing conditions, either directly or through a correlation with continuum state variables like thermal history and strain path. Similarly, property models predict a material's performance under some operating conditions, given a description of its internal structure. Thus, it becomes obvious that the natural link between these models is the internal structure of the material that is output from one and input to the other. The internal structure of nearly all materials is complex, multi-scale and not easily defined by a small number of parameters. As such, there exists an opportunity in materials engineering to advance the quantitative description of internal structure and move further away from ad-hoc, word-based descriptors (i.e. equiaxed, acicular, basket-weave, etc.). Historical efforts have been made to quantify selected aspects of microstructure (ASTM grain size, etc.), but generally the metrics chosen stopped at average quantities, which in part has been driven by the limited description of microstructure in models. The MGI has challenged the materials community to develop a framework for describing materials in a consistent and quantitative way [2], more similar to the approach applied to sequencing the human genome.

Two critical themes in these initiatives are the move to the digital basis and the call for tools with clear and understandable inputs/settings (be they software or hardware). There are 'easy-to-use' software tools that exist in both the processing (ProCast, Deform, etc) and property (Darwin, Abaqus, Deform, etc) modeling regimes. However, there is a lack of easy-to-use software tools that exist to process, quantify and represent microstructure in a general sense, especially in three dimensions (3D). This becomes a problem if one is attempting to validate the predictions of processing models or provide property models with accurate input. The work discussed in this paper is aimed at developing a software architecture that is both open and scalable to address the growing needs for quantitative, digital analysis of microstructural data. The ultimate goal of this effort is to fill the gap in the ICME chain with respect to 'easy-to-use' microstructure quantification and representation tools across all material classes and length scales. Another important goal of this work is to standardize the format of material microstructure data, so that the increasing demand for access to scientific research data can be met [3].

It should be mentioned that the initial focus of DREAM.3D was far less general and pervasive than the ideas discussed in this paper. It was only during this initial development effort that the authors encountered the difficulties that will be discussed here and subsequently broadened the scope and vision of DREAM.3D. This broader vision is in line with efforts in the biological community [4,5] and the authors see a potential for further integration with that community. Many of the examples in this paper reflect the personal experiences of the authors and within this context, we highlight the path needed for the advancement of digital microstructure analysis. Note that microstructure is used throughout this paper as a general term for the internal structure of materials and does not refer to a specific length-scale.

Barriers to integration/development

At the outset of this work, many computational tools existed for treating various aspects of microstructure quantification, post-processing/clean-up, data visualization, etc. However, these tools remained disjointed and generally non-transferable between researchers. It became clear, both to the authors of this work and authors of many of the disjointed tools, that a larger integrated environment was needed to be developed to fully realize the utility of any of the individual tools. Integrating computational codes into a larger ecosystem presents many barriers, for example: storage format of the data, usability of the codes (Graphical User Interface (GUI) vs. Command-Prompt), documentation and Intellectual Property (IP) rights, to name a few. In the typical case, each of these issues needs to be addressed in order for the code to be widely usable by other researchers. A critical component for efficiently solving these issues in a consistent way is having a long-range vision for how the software will be used and any possible growth opportunities. In the following subsections, some of the most critical barriers are highlighted and how the authors addressed them in the development of DREAM.3D will be discussed.

Data structure and storage

One of the critical barriers to integration of software codes is ensuring that downstream algorithms can properly interpret the data produced by upstream algorithms.

Misunderstanding how data is structured can lead to a significant barrier that prevents algorithms from being integrated. During the development process, often the researcher is mainly focused on the correct implementation of the algorithm and gives less time to designing a data and file format that is both efficient from a computational standpoint and shareable with other researchers. This leads to input and output files that are not written in any standardized format. Subsequently, substantial effort may go into manipulating the output files of one algorithm so that the next algorithm can use the data as an input, which is highly inefficient whether done manually or by a computer. A basic example of this problem is an algorithm that stores data as a comma separated list of values and another algorithm that reads values from a space separated list of values. In order for these algorithms to work seamlessly together, one or both of the codes would need to be modified or commonly a third program would be created to 'translate' between the data structures. Developing software this way hinders the reusability of the codes and will present a barrier to the adoption of the codes in the greater community. DREAM.3D aims to use a widely available open-source format to store both archival and processed data. However, this only addresses the external, or resting format of data. Another important issue is how the data is represented internally, which can significantly impact how easily algorithms are able to share data and information. DREAM.3D utilizes a scalable organization to describe data at all dimensionalities.

Ease of use

As mentioned previously, research grade codes are often developed with little emphasis given to the usability of the codes by other researchers. Generally, this is because the code is never intended for use beyond the author. Many codes typically run from a command prompt or terminal environment and offer little information about the required number and types of input parameters. Worse still is when the author stops actively developing the code and the knowledge of how to use the algorithm and the sensitivity to its input parameters is lost. When this happens, the code becomes effectively unusable. DREAM.3D has tried to mitigate these situations through the use of formal coding protocols. These protocols dictate how the documentation for a filter is written, how the user interface is created and how the filter will interact with the rest of the system. Collectively these formal design patterns are used to ensure that the filter can be employed by researchers in the field simply by reading a documentation file and/or following a simple example. Some of the items that go into integrating a filter into the DREAM.3D system include, but are not limited to, the following items

- Filter is documented including required input parameters and data, output data created and an explanation of the algorithm (including citations if needed).
- Input parameters are written to and read from a native DREAM.3D file.
- Required inputs are enumerated using native DREAM.3D data structures.
- Outputs are clearly defined and relayed to the DREAM.3D internal data structures.

Intellectual property

During the early stages of DREAM.3D a conscious decision was made to structure the codes in such a way as to allow the use of external libraries that may contain proprietary algorithms. This allows academia, industry and government institutions to contribute algorithms and still protect their intellectual property. These various institutions can elect to release the source code to the open community or keep the source private and only release a precompiled library that is compatible with the current release of DREAM.3D. As other institutions begin to contribute to DREAM.3D they can make their own decision as to what is the best model to distribute their specific computational tools.

Long-term maintainability

The maintainability of DREAM.3D has several facets of discussion. From a programmer's perspective the authors strive to use best practices when developing the various algorithms, reusing algorithms and creating reusable software objects that can be applied or adapted to new algorithms. A suite of unit tests are continually developed to ensure the behavior of the public functions is not altered when bug fixes and algorithmic enhancements are added. Currently, all the external libraries that DREAM.3D is built on top of are all open-source, thus giving the development team complete access to the entire code base that is used to build DREAM.3D. Another perspective to consider is the source of funding for DREAM.3D development. The current development of DREAM.3D has been essentially exclusively funded by U.S. Government sources. This funding enabled building the integrated core infrastructure that enabled much of the critical aspects already discussed. However, the core of DREAM.3D should become relatively static, with only minor 'usability' additions in the near future. It will be at this point, which has already begun to occur, that academia and industry will begin to drive the growth and development of DREAM.3D. This growth will likely be focused almost entirely on filter development and expansion. It is the belief of the authors that the materials community, possibly with government support (either directly or through academic funding), will view the core as an enabling tool that will be in the 'best interest' to update as needed with a small overhead on filter development efforts. Finally, it should be noted that since DREAM.3D is currently open-source, the current instance of the core will always be available. Any filter additions that can operate with the current design can always be used. Also, any user can download the current core and extend it to address their research needs.

Methods

Data representation and file format

Material microstructures come in many different sizes and shapes and the features of interest have different dimensionalities. Data describing attributes of microstructure can be obtained from many sources (Scanning Electron Microscopy, Transmission Electron Microscopy, Optical Microscopy, Electron Backscatter Diffraction, Energy Dispersive Spectroscopy, Wavelength Dispersive Spectroscopy, 3D Atom Probe, Atomic Force Microscopy, etc.). Unfortunately, during the development of these experimental methodologies, no common data structure was developed and as such, combining data from multiple sources is difficult. Further, the tendency to link the data with a material class (metal, ceramic, composite, polymer, etc.) has stunted the development of a unified method for describing microstructure data. During development of DREAM.3D, the vision of a unified representation of all digital microstructure data for all material classes and length-scales presented a challenge. As discussed in [6], when writing code or designing data structures that operate on or represent a variety of features and dimensions, it is critical to establish a proper abstraction layer to ensure transferability. In the case of DREAM.3D and microstructure, the authors believe the proper abstraction layer is to work with all features of structure as geometrical objects. By abstracting the materials interpretation of the features and focusing only on how the feature is described digitally, DREAM.3D has been able to institute a general, unified structure for digital data that assumes no prior knowledge of length-scale or material class. The following subsections will discuss this generic data structure and illustrate its direct use in a wide range of materials applications.

Geometric mesh element construct for holding digital microstructure data

Spatially-resolved digital data, of which most material microstructure data is a subset, are simply information or attributes that are associated with discrete geometrical elements. These elements can be pixels in an image, points in a probe scan, line segments in a digital model, etc. At this level, all digital microstructure data can be treated/organized similarly within a computer. Meshes of appropriate dimension can be created and data can sit on the mesh element(s) that they describe. For example, the mass-to-charge ratio of an atom in an atom probe dataset is information associated with a point, while the misorientation across a boundary in a electron backscatter diffraction (EBSD) dataset is information associated with a surface. As such, any given dataset has an associated mesh dimensionality equal to the highest dimension of feature its data describes. It should be noted that the mesh dimensionality may be different from the dimensionality of the dataset. For example, the atom probe dataset consists of 3D locations having (x,y,z) coordinates, but represents microstructure features that are treated as a 0-D point.

DREAM.3D organizes/stores mesh data (and subsequent feature and ensemble data discussed in the next section) in a structure called a “data container”. DREAM.3D uses four types of data containers for the different possible data dimensionalities (Vertex = 0D, Edge = 1D, Surface = 2D, Volume = 3D). Figure 1 illustrates the different data containers and the data they can hold. As Figure 1 shows, lower dimensional geometrical objects bound higher dimensional objects and a given data container can store data on mesh elements of a lower dimension. For example, in a 3D EBSD dataset, the collected orientation data is generally treated as belonging to a cell, but the misorientation between neighboring cells could also be stored on the face shared by the cells and the edges and vertices of the cells could store the coordination number of different features they belong to (i.e. triple line or quadruple point). An example dataset of each type of data container can be found in the supporting material. The examples include a Vienna Ab initio Simulation Package (VASP) input structure (Vertex data container - Additional file 1), a ParaDis output structure (Edge data container - Additional file 2), a grain boundary mesh of a synthetic polycrystalline microstructure (Surface data container - Additional file 3) and a synthetic polycrystalline microstructure (Volume data container - Additional file 4).

The mesh that represents the data locations is unique to the dataset itself. While the mesh can be altered via smoothing, regridding or other processing steps, it is generally defined by the data collection or generation protocol/settings. Furthermore, the mesh itself is not influenced by the material class and can exist at any length-scale. The mesh is solely the physical location of all data elements and their associated attributes.

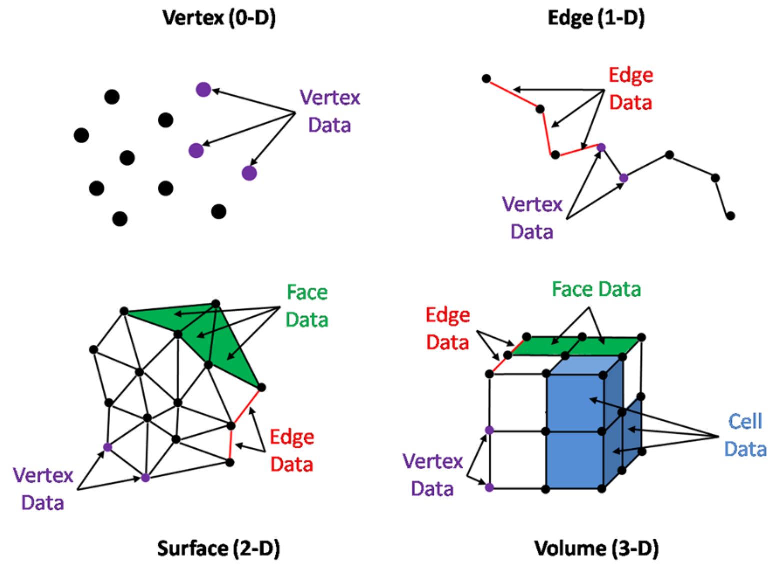


Figure 1 Schematic showing the four types of data container implemented in DREAM.3D and the types of data that can be stored on each mesh. Note that Faces in the Surface data container need not be triangles and Cells in the Volume data container need not be voxels, but have been depicted this way for visual simplicity.

Hierarchical grouping for feature and ensemble representation

A given material's microstructure can be thought of as being constructed using building blocks called "features" such as grains, fibers, pores, magnetic domains, corrosion pits, dislocations, individual atoms and many other possibilities. Though these features are very different in the "real world" material's sense, digitally they are all simply groups of discrete mesh elements. It is the user's prerogative to group the mesh elements in whichever way makes most sense for their uses, which imparts a certain uniqueness to the data set. It is the human interpretation of what the features represent that links the data to a specific material class and/or length-scale. DREAM.3D utilizes a software engineering technique where all of the domain specific groupings can be represented by a generalized data structure. This is commonly referred to as an "Abstraction Layer" in the software engineering field and allows the DREAM.3D system to grow and adapt to new domains.

From the perspective of the computer, the act of assigning elements to a given feature is still material class and length-scale independent. Mesh elements are simply noted to belong to a given feature for a given segmentation/grouping protocol. For each grouping/segmentation protocol, all elements are set to belong to one and only one feature. It is possible that a user would want to group mesh elements by multiple protocols. For example, mesh elements could be grouped by common orientation and then by common chemistry if a data set had both orientation and chemical information. If multiple grouping protocols are used, then each mesh element would have a vector of feature IDs listing which feature it belongs to in each grouping.

After features are defined, attributes such as size, shape, etc. can be calculated and stored associated with each feature. The structure of how these attributes are stored will be discussed in the next section. Also, it may be desirable to the user to group features together to establish "ensembles". Ensembles are groups of features that the user has linked for some reason. Similar to each mesh element having one (or more) feature IDs to list what feature it belongs to, each feature has one (or more) "ensemble IDs". For example, a group of features could be linked because they are all the same phase, because they are the largest 10% of features, etc. Similar to features and individual elements, attributes describing ensembles such as size distribution, average feature curvature, orientation distribution function (ODF), etc. can be calculated and stored associated with each ensemble.

Scalable layout for information storage

At all levels, from the individual mesh elements to features and ensembles, the method of how information is stored must be dynamic. In order to be a flexible software environment that can work with data from multiple sources and treat microstructures from all material classes, it is not reasonable for DREAM.3D to predefine what attributes can be associated with a mesh element, feature or ensemble. As such, a matrix-style container is needed for holding information of this type. For example, in an EBSD scan, each pixel has an Euler angle set, a phase ID, a coordinate in space and a list of values associated with the indexing approach of the commercial software that collected the scan. These attributes, as a set, are called a 'property vector' in DREAM.3D and define the pixel with which they are associated. These property vectors are shown as columns in Figure 2. The rows in Figure 2 are the lists of single attributes for all pixels and are called "attribute array". Given this container structure, it becomes clear that as filters are applied to the data, more attribute arrays are generated and each property vector grows.

At each level (mesh element, feature, ensemble), attribute matrices can exist. Only one matrix exists at the element level because there is no user grouping at that level and as such there is only one definition or instance of the mesh. However, at the feature and ensemble levels, many attribute matrices can coexist. In an attribute matrix, every property vector is the same size and every attribute array is the same size. This is because filters calculate attributes and filters must loop over all members in the attribute matrix for which the attribute is being calculated.

HDF5 File structure

The Hierarchical Data Format Version 5 (HDF5) is an open-source library developed and maintained by "The HDFGroup" [7] that implements a file format designed to be flexible, scalable, highly performant and portable. HDF5 allows each application to organize

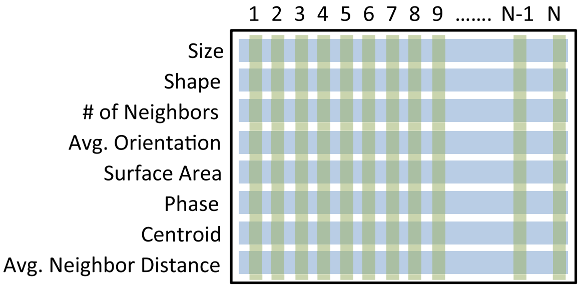


Figure 2 Schematic layout of the container structure to store attribute arrays. Blue represents an "attribute array", where as green represents a "property vector".

its data in a hierarchy that makes sense for the application. Virtually any type of data, from scalar values to complex data structures, can be stored in an HDF5 file. Scalability has been a design consideration from the outset and HDF5 can handle data objects of almost any size or dimensionality. The library has also been designed to be efficient at querying, reading and writing data objects, and including utilizing parallel I/O when needed. One of the most important aspects of HDF5 is its portability across all the major computing operating systems. HDF5 has support for C, C++, Fortran and Java as its native implementations; many higher-level programming languages also have direct support for HDF5, including IDL (Interactive Data Language), MATLAB and python.

HDF5 files can be thought of as 'a file system within a file'. Data can be stored as datasets (analogous to files) and arranged inside groups (analogous to folders) all within the HDF5 file. This structure is well-suited for storing the organized data from DREAM.3D. The organization of a typical DREAM.3D file is shown in Figure 3. At the 'root directory' or highest level in the file, two groups exist for holding 1) the processing pipeline and 2) all data containers of the dataset. Inside the pipeline group, there are subgroups for each filter and within each subgroup there are datasets for each of the input parameters of the filter. The subgroups are titled as their numerical order in the processing pipeline, but have attributes stored on the group listing the name of the filter and its version number. The datasets inside the subgroups are titled as the name of the input parameter they hold and the contents are the value(s) of the input parameter. Inside the data container group are subgroups for each data container that exists in the dataset. The subgroups are titled as the name the user gave to the data container. Within each subgroup there are multiple groups (the number depending on the dimensionality of the data container).

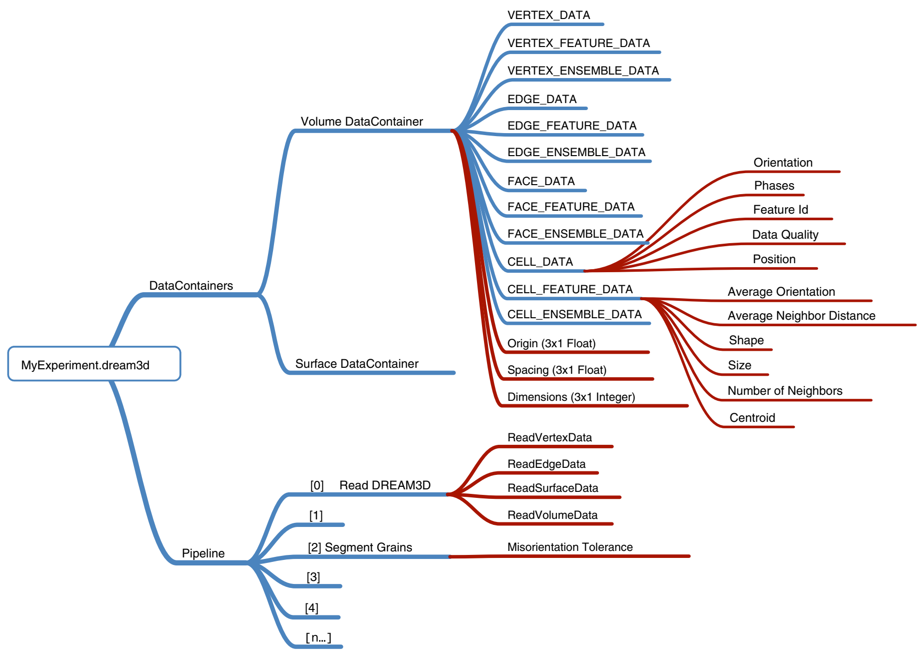


Figure 3 Schematic layout of a native DEAM3D file on disk. Blue nodes represent HDF5 groups and red nodes represent HDF5 data sets.

Each group at this level is associated with an attribute matrix described in the previous section. For example, if the data container was a vertex data container, then there would be a group for the vertex mesh element attribute matrix and there could be multiple groups of feature and ensemble attribute matrices depending on the number of grouping schemes employed by the user. In the example in Figure 3, the dataset contains a single volume data container. Within an attribute matrix group, each dataset represents an attribute array (or row from Figure 2). The name of the dataset is the name of the attribute array and the contents are the entire attribute array in order from object 1 to N.

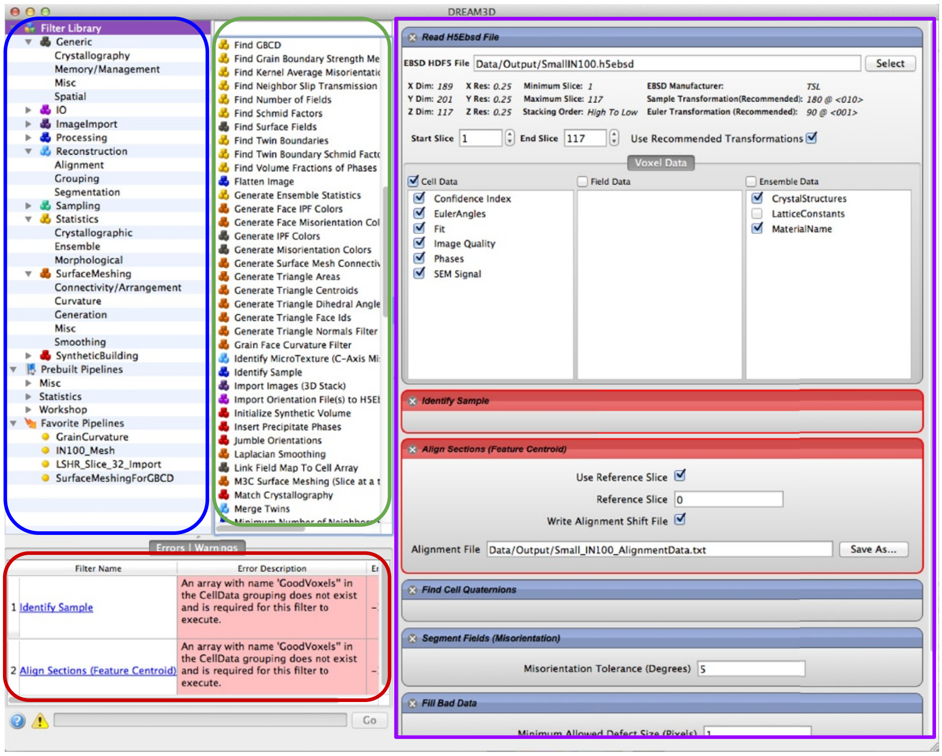
The structured layout of HDF5 and the DREAM.3D file also offer potential for databasing of datasets. The ability of HDF5 to query the existence of datasets and groups without reading the entire file is well-suited for determining if data meets a specified criterion, whether it be a specific processing path, attribute array, etc.

Pipeline concept

DREAM.3D's pipeline workflow is designed around the concept of signal processing. In this analogy, the signal is the 'raw' data and the individual algorithms/programs in DREAM.3D are filters that process the signal. It is for this reason that DREAM.3D refers to each individual program as a filter. It should be noted that unlike typical signal or image processing, many of the filters in DREAM.3D do not change data/attributes existing on each element, feature or ensemble, but rather create new data/attributes to be stored. The intent of modular pipeline workflows is to separate the two critical aspects of data processing: algorithms and order of operations. When designing a pipeline, the user is solely focused on the latter while using an existing set of algorithms. Each algorithm can be treated as a module that can be modified or replaced if it is not generating the desired results. The following subsections will discuss the user interface of the workflow and how it is linked to the data.

Visual programming workflow

In a typical high level programming environment, such as MATLAB or IDL, the user must manually type in the proper commands to build the desired pipeline/workflow and ensure that all data is available during the execution. Many times missing data can cause the systems to crash at worst or give a cryptic error message in the best case. With DREAM.3D a visual approach to designing the workflow was engineered. Each filter can still be thought of as a pre-packaged subroutine like the functions in MATLAB or IDL, but in DREAM.3D a visual linking of the filters/subroutines is more analogous to programming environments like LabView. Each filter has the knowledge of every piece of data that is required before it will execute. As each filter is placed into the pipeline area the workflow is executed in a "preflight" step where each filter dynamically checks to make sure it will have the required input data to operate (also similar to LabView). If any inputs are not correct or there is missing data an error message is displayed for the user to correct. Once all the errors are corrected the pipeline will be allowed to be executed. In this respect DREAM.3D presents a very high level and simple programming model that is easy and straight forward to learn. An example of the DREAM.3D GUI with a pipeline containing errors is shown in Figure 4 to illustrate the layout of the software.



EBSD HDF5 File Data/Output/SmallIN100.h5ebsd

X Dim: 189 X Res: 0.25 Minimum Slice: 1 EBSD Manufacturer: TSL
Y Dim: 201 Y Res: 0.25 Maximum Slice: 117 Sample Transformation(Recommended): 180 @ <010>
Z Dim: 117 Z Res: 0.25 Stacking Order: High To Low Euler Transformation (Recommended): 90 @ <001>

Start Slice 1 End Slice 117 Use Recommended Transformations ☑

☑ Cell Data
☑ Confidence Index
☑ EulerAngles
☑ Fit
☑ Image Quality
☑ Phases
☑ SEM Signal

☐ Field Data

☐ Ensemble Data
☑ CrystalStructures
☐ LatticeConstants
☑ MaterialName

Use Reference Slice ☑
Reference Slice 0
Write Alignment Shift File ☑
Alignment File Data/Output/Small_IN100_AlignmentData.txt

Misorientation Tolerance (Degrees) 5

Minimum Allowed Defect Size (Pixels) 1

	Filter Name	Error Description	Er
1	Identify Sample	An array with name 'GoodVoxels" in the CellData grouping does not exist and is required for this filter to execute.	–
2	Align Sections (Feature Centroid)	An array with name 'GoodVoxels" in the CellData grouping does not exist and is required for this filter to execute.	–

Figure 4 DREAM.3D GUI with example pipeline. The region highlighted in blue contains the Filter Library, Prebuilt Pipelines and Favorite Pipelines, where filters are grouped and common pipelines are stored. The region highlighted in green contains the Search Window, where filters can be searched or viewed by group. The region highlighted in purple contains the Pipeline Area, where filters are inserted and ordered to build a pipeline. The region highlighted in red contains the Errors and Warnings Window, where messages regarding the feasibility of the pipeline are displayed to the user.

Working file format and complete data provenance

Continuing the analogy of signal processing, one can think of the digital data, raw or otherwise processed, as existing in a discrete processing-step domain. Between all filter steps in the processing pipeline, the data exists as a 'snapshot'. DREAM.3D terms these points as a 'digital instance' of the microstructure. Any digital instance should be reproducible by beginning with the raw data and following the same processing pipeline to the point the instance was captured. It is this realization that led to the differentiation between a working file and an archival file in DREAM.3D. An archival file, which is discussed in more detail in (Jackson, Groeber, Rowenhorst, Uchic and DeGraef: "h5ebsd: An archival data format for electron back-scatter diffraction data sets.", submitted), contains the unaltered data from the collection instrument along with meta data that may help infer the inherent artifacts of digitizing the true 'analog' microstructure. DREAM.3D terms the unaltered data as step = 0 in the processing-step domain. A working file implies the contained microstructure is beyond step = 0 in the processing-step domain and some filtering has been applied to the data. During the execution of a processing pipeline in DREAM.3D, the user can export/save an instance of the microstructure at any point by writing all of the in-memory data arrays to a data file using the HDF5 format. These files are organized by each data container that is being used within the active pipeline. In addition to saving the complete set of in-memory data arrays to a file, the complete processing history of that data is also saved in the form of the user's pipeline up to the point of saving the instance. If the user selects to start a pipeline by reading a DREAM.3D file, the previous pipeline stored in the starting file is transferred to any newly written DREAM.3D file, thus keeping the entire provenance of the data safely stored in the same file. The pipeline information is written such that the specific version of DREAM.3D that was used during the processing is attached to each filter's inputs values. This type of meta-data attachment can help researchers independently recreate results of past experiments and is fast becoming an important aspect of scientific publishing.

Growth and scalability of DREAM.3D

For DREAM.3D to realize the vision of a material class and length-scale independent analysis environment, careful thought had to be given to the creation of a method for implementing new filters with a low barrier to entry. No one researcher, group or laboratory has the knowledge diversity or time to implement all useful processing or analysis filters for even one material class, let alone all. For this reason, an environment where collaboration and competition of ideas/algorithms/implementations can occur on a common basis is critical to the development of standards for microstructure analysis.

Plugin Architecture

In order to allow DREAM.3D to grow organically through the addition and integration of new algorithms, a plugin architecture has been designed and implemented. This allows researchers with some programming experience to expand and enhance the capabilities of DREAM.3D. The plugin architecture allows entities such as government labs and commercial businesses to create DREAM.3D compatible binary plugins and retain full rights to their specific sources. Plugin developers can distribute their tools in a number of different ways. First, the developer(s) can contribute their filters directly to DREAM.3D and have them compiled with the core. The core of DREAM.3D is simply the previously discussed internal data management classes, macros to facilitate filter-to-filter communication, the GUI and a set of libraries for common operation like math and I/O. This path is only possible if the developer(s) release their plugin as open-source, as the core of DREAM.3D is open-source. A second option is for the developer(s) to compile their plugin themselves and then distribute their plugin as a library. Under this second option, the developer(s) retain multiple avenues for dissemination. The plugin can be offered as freeware, can be licensed or can be provided with the source (albeit disjointed from the DREAM.3D core). Offering these options is intended to help drive adoption of DREAM.3D across a diverse set of materials science domains and industries. This type of programming model leverages contributions of different organizations to allow the entire DREAM.3D system to grow larger, thus spreading the development cost among all of those different organizations.

Documented interface protocol and common libraries

DREAM.3D has been developed with a concerted focus on lowering the barrier for future developers to contribute codes. Common libraries for math, I/O and internal data management are supplied with the core of DREAM.3D. DREAM.3D also supplies libraries for dynamically creating the visual presence of a filter, provided a relatively simple interface required by the filter. This limits the amount of 'low-level' computer science knowledge the developer must have. Furthermore, DREAM.3D can be compiled to build an accompanying program that will generate all necessary files for a user that is making a new plugin. The shell files created contain all the required functions of new filters along with examples of how to add input parameter calls to the user and how to request and add data to the scalable attribute matrices. This allows the developer to focus on their algorithm, while operating in a 'put-your-algorithm-here, list-your-input-requirements-here, list-your-output-details-here' type of environment.

Interface with external software

Future growth of DREAM.3D is also likely to be tied to the ability to integrate DREAM.3D with other software packages. For example, the authors made a conscious decision early in the development of DREAM.3D to not invest time and effort into generating a visualization package within DREAM.3D. Instead, a link was built to interface with ParaView [8], an open-source visualization environment developed by Kitware with Department of Energy (DoE) funding. ParaView is a powerful visualization package with many, many man-years of development already invested. Using the HDF5 file structure already discussed, coupled with an XML description (Xdmf format), DREAM.3D files can be opened and viewed within ParaView. As such, new developments to ParaView are indirectly developments to DREAM.3D.

Results and Discussion

Case studies

This section will demonstrate the workflow and data structure of DREAM.3D in a set of case studies. Due to the historical focus of the software tools that evolved to become DREAM.3D, many of the filters currently in DREAM.3D are related to processing and analysis of polycrystalline metal datasets with 3D EBSD data. The case studies presented here show a subset of current DREAM.3D functionalities, but should not be viewed as an exhaustive list of current or future capabilities. Furthermore, the final results of the various pipelines may not be different than previous codes or similar analysis software packages. The major differentiating factor with DREAM.3D is the time and manual effort to get results. The use of a visual representation of the workflow reduced the learning curve greatly, which makes DREAM.3D an approachable software suite for all levels of user.

Reconstruction and Meshing (3D EBSD)

A dataset consisting of 117 serial sections through a polycrystalline Ni-based superalloy with EBSD data on each section was collected in [9]. DREAM.3D was used to reconstruct, segment, clean-up and mesh the features of the dataset. The pipeline used to accomplish these tasks, which is listed in Table 1, will be discussed briefly. In the interest of brevity, the details of each individual filter will not be discussed here, but can be found in the documentation of DREAM.3D. Many of the steps in the pipeline are also discussed in [9] and [10]. It should be noted that the results presented in [9] and [10] were generated prior to the existence of DREAM.3D. The total processing/analysis time in the previous work

Table 1 3D EBSD reconstruction pipeline

Filter #	Filter name	Reason for use
0	Read H5EBSD File	Loads raw EBSD data
1	Multi Threshold (Cell Data)	Allows user to define which voxels are 'good'
2	Find Cell Quaternions	Converts voxel Euler angles to Quaternions
3	Align Sections (Misorientation)	Rough alignment of sections by minimizing misorientation between sections
4	Identify Sample	Adjusts the 'good' voxels assuming one contiguous block of 'good' data
5	Align Sections (Feature Centroid)	Secondary alignment of sections assuming sample is parallelepiped
6	Neighbor Orientation Comparison	Checks orientation of 'bad' voxels against neighboring 'good' voxels
7	Neighbor Orientation Correlation	Second check of 'bad' voxels against neighboring 'good' voxels
8	Segment Features (Misorientation)	Identifies features of similar orientation
9	Find Feature Phases	Determines the phase of each feature
10	Find Feature Average Orientations	Calculates average orientation of each feature
11	Find Feature Neighbors	Determines list of neighbors for each feature
12	Merge Twins	Merges features misoriented by 'special' sigma3 relationship
13	Minimum Size Filter	Removes small features and fills gaps with neighboring features
14	Find Feature Neighbors	Determines neighbors after removing features
15	Minimum Number of Neighbors Filter	Removes features with few neighbors
16	Fill Bad Data	Fills in 'bad' data with neighboring 'good' data if 'bad' data regions are small
17	Erode/Dilate Bad Data	Shrinks any remaining 'bad' data regions
18	Erode/Dilate Bad Data	Grows back any remaining 'bad' data regions
19	Write DREAM.3D File	Writes attribute matrices and pipeline to file

Table listing the filters in the reconstruction pipeline.

took approximately 24 hours and involved moderate manual interaction between multiple software codes. The current processing/analysis time was reduced to approximately 5 minutes and required effectively no user interaction (beyond setting up the pipeline). In both cases, the times quoted reflect use of a standard desktop PC. The resulting digital instance is shown in Figure 5 and is attached as supporting material in the form of a DREAM.3D file (Additional file 5).

Statistical analysis

The previous section discussed the reconstruction and segmentation of a polycrystalline Ni-based superalloy dataset with 3D EBSD data. Upon reconstructing and segmenting the data to obtain features, those features and ensembles of those features can be measured and statistically described. Table 2 lists the pipeline used to calculate a number of morphological and crystallographic attributes of the features and ensembles within the dataset. The list of features and all attributes calculated to describe them can be found in a comma separated value (.csv) file in the supporting material (Additional file 6). Some of these results were also presented in [10] (using tools prior to DREAM.3D). After determining the attributes of the individual features, distributions of those attributes can be calculated for ensembles of the features. For this dataset, the material was treated as single-phase and all grains were said to belong to a single ensemble. The distribution of sizes, shapes, numbers of neighbors, orientations and misorientations

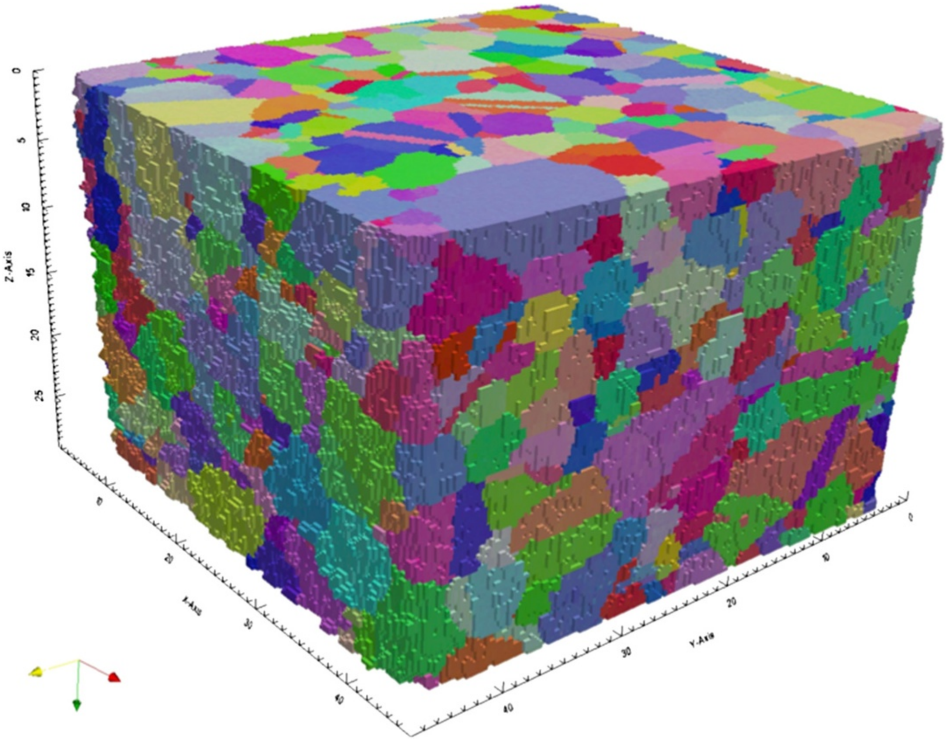


Figure 5 Visualization from ParaView with Inverse Pole Figure (IPF) coloring of the polycrystalline Ni-based superalloy reconstructed and segmented by the DREAM.3D pipeline described in Table 1.

Table 2 Statistics pipeline

Filter #	Filter name	Reason for use
0	Read DREAM.3D File	Loads reconstructed and segmented dataset
1	Find Feature Centroids	Determines the centroid locations of each feature
2	Find Feature Sizes	Determines the volume of each feature
3	Find Feature Shapes	Determines aspect ratios and omega3 of each feature
4	Find Feature Neighbors	Determines the number and list of contiguous neighbors for each feature
5	Find Feature Neighborhoods	Determines the number and list of features within one diameter of each feature
6	Find Euclidean Distance Map	Determines the distance each voxel is from the nearest grain boundary, triple line and quadruple point
7	Find Feature Average Orientations	Second check of 'bad' voxels against neighboring 'good' voxels
8	Find Feature Average Orientations	Calculates average orientation of each feature
9	Find Feature Neighbor Misorientations	Determines the misorientation for each contiguous neighbor of each feature
10	Find Schmid Factors	Determines the Schmid factors of each feature
11	Find Feature Reference Misorientations	Determines the misorientation between each voxel and a reference orientation for the feature it belongs to
12	Find Kernel Average Misorientations	Determines the average misorientation between each voxel and its neighbor voxels
13	Write Feature Data As CSV	Outputs attributes of features to CSV file
14	Write DREAM.3D File	Writes out all attribute matrices and pipeline to file

Table listing the filters in the statistical analysis pipeline.

Table 3 Synthetic structure generation pipeline

Filter #	Filter name	Reason for use
0	Initialize Synthetic Volume	Loads goal statistics and creates empty volume
1	Pack Primary Phases	Generates set of grains and places them inside of volume
2	Find Feature Neighbors	Determines the number and list of contiguous neighbors for each feature
3	Find Number of Features	Determines the number of features in the volume
4	Match crystallography	Assigns orientations to match the ODF and MDF
5	Write DREAM.3D File	Writes out all attribute matrices and pipeline to file

Table listing the filters in the synthetic structure pipeline.

were all calculated. The following section will discuss one use case for applying this information.

Sythetic structure generation

DREAM.3D has filters to generate synthetic digital microstructures with a goal set of statistics as input. The synthetic generation process is discussed in detail in [11]. Using the statistics calculated by the pipeline in the previous section, a ‘statistically-equivalent’ microstructure was generated using DREAM.3D. The pipeline used to generate this microstructure is listed in Table 3 and the DREAM.3D file corresponding to the synthetic volume is attached as supporting material (Additional file 4). A visualization of the resultant synthetic microstructure is shown in Figure 6. It should be noted that the synthetic microstructure generated was created using statistics calculated without the twin features

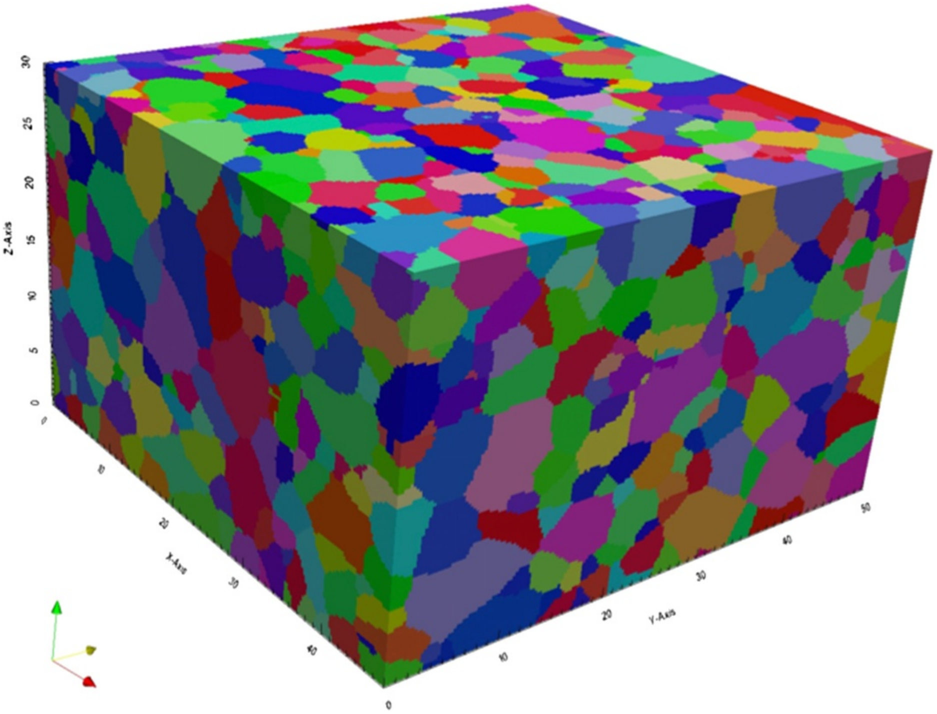


Figure 6 Visualization from ParaView with Inverse Pole Figure (IPF) coloring of the synthetic microstructrue generated by the DREAM.3D pipeline described in Table 3.

in the microstructure and as a result may look slightly different that the experimental microstructure in Figure 5.

Conclusion

DREAM.3D is an open-source software package focused on creating a high-level programming environment to process, segment, quantify, represent and manipulate digital microstructure data. DREAM.3D's central goal is to enable the move of microstructure quantification to a digital basis with easy-to-use software tools. The core of DREAM.3D implements a standardized approach to working with and storing digital microstructure data. Additionally, protocols are included to allow independently-developed filters and plugins to interface with one another. The DREAM.3D environment is constructed in a way that small research groups, government laboratories, start-up companies and major industrial corporations can collaborate and leverage each other's work. It is the belief of the authors that DREAM.3D will reduce the time and cost to conduct microstructural characterization, due to the ability to leverage community-wide developments and bring disjointed research areas into a common environment for development.

Additional files

Additional file 1: Example vertex data container. This file is an example of a vertex data container containing a Vienna Ab initio Simulation Package (VASP) input structure.

Additional file 2: Example edge data container. This file is an example of an edge data container containing a ParaDis output structure.

Additional file 3: Example surface data container. This file is an example of a surface data container containing a grain boundary mesh of a synthetic polycrystalline microstructure.

Additional file 4: Example volume data container. This file is an example of a volume data container containing a synthetic polycrystalline microstructure.

Additional file 5: Polycrystalline ni-based superalloy 3D EBSD reconstruction. This file contains a reconstructed and segmented experimentally measured polycrystalline Ni-based superalloy microstructure.

Additional file 6: Grain statistics. This file contains the attributes calculated in the Statistics pipeline for all features identified in the 3D EBSD reconstruction pipeline.

Competing interests

The authors declare that they have no competing interests.

Authors' contributions

The authors contributed to this paper equally and would like to be considered as joint First Authors. MJ generally contributed more of the computer science vision and MG generally contributed more of the materials engineering vision. All authors read and approved the final manuscript.

Acknowledgements

MAJ would like to acknowledge financial support from the U.S. Air Force Research Laboratories contracts # FA8650-07-D-5800 and # FA8650-10-D-5210 and U.S. Naval Research Laboratories contract # N00173-07-C-2068.
The authors would like to acknowledge the following people:

- Chris Woodward for his strong belief in the need to integrate software tools that initiated this effort
- Marc DeGraef, Sukbin Lee, Greg Rohrer, Anthony Rollett, David Rowenhorst, and Joseph Tucker for sharing their codes and knowledge to help create the initial functionality of DREAM.3D and for setting a standard for what collaboration should be in these types of ventures
- Michael Uchic and Dennis Dimiduk for their daily discussions and consistent belief and support
- Joseph Kleingers for his efforts to improve the usability of DREAM.3D
- Jeff Simmons for his vision beyond the materials community that helped crystallize the views now central to DREAM.3D
- Paul Ret for his unwavering support for DREAM.3D's vision
- The early adopters of DREAM.3D for their patience and suggestions that have helped to grow and strengthen the software.

Received: 15 November 2013 Accepted: 27 February 2014
Published: 1 April 2014

References

1. Committee on Integrated Computational Materials Engineering (2008) Integrated Computational Materials Engineering: a Transformational Discipline for Improved Competitiveness and National Security. National Research Council, The National Academies Press, Washington D.C.
2. National Science and Technology Council (2011) Materials Genome Initiative for Global Competitiveness. Executive Office of the President, Washington D.C
3. Holder J (2013) Increasing Public Access to the Results of Scientific Research. https://petitions.whitehouse.gov/response/increasing-public-access-results-scientific-research.
4. OME. http://www.openmicroscopy.org/site/.
5. OSS. http://loci.wisc.edu/software/oss/.
6. Pietzsch T, Preibisch S, Tomancak P, Saalfeld S (2012) ImgLib2 - generic image processing in Java. Acta Materialia 28(22): 3009–3011
7. HDF5. http://www.hdfgroup.org/HDF5/.
8. Kitware. http://www.paraview.org/.
9. Groeber M, Haley BK, Uchic MD, Dimiduk DM, Ghosh S (2006) 3d reconstruction and characterization of polycrystalline microstructures using a fib-sem system data set. Mater Charac 57: 259–273
10. Groeber M, Ghosh S, Uchic MD, Dimiduk DM (2008) A framework for automated analysis and simulation of 3d polycrystalline microstructures. part 1: Statistical characterization data sets. Acta Materialia 56: 1257–1273
11. Groeber M, Ghosh S, Uchic MD, Dimiduk DM (2008) A framework for automated analysis and simulation of 3d polycrystalline microstructures. part 2: Synthetic structure generation. Acta Materialia 56: 1257–1273

doi:10.1186/2193-9772-3-5
Cite this article as: Groeber and Jackson: DREAM.3D: A Digital Representation Environment for the Analysis of Microstructure in 3D. *Integrating Materials and Manufacturing Innovation* 2014 **3**:5.