

Data Structures and Workflows for ICME



Sean P. Donegan and Michael A. Groeber

1 Introduction

Integrated computational materials engineering (ICME) represents a grand challenge within materials research and development. Effective ICME involves coupling materials characterization and experimentation with simulation tools to produce a holistic understanding of the materials system, promising to accelerate the materials development enterprise. Under the Center of Excellence on Integrated Materials Modeling (CEIMM), significant strides were made in developing state-of-the-art experimental methods and simulation techniques for interrogating material structure and behavior across multiple scales. In parallel to these method developments, several advances were made in designing data structures and workflow tools that possess the required flexibility and extensibility to operate on the data produced by such advanced methods. Such software tools are a critical enabling component for effective ICME; the National Academy of Sciences noted cyberinfrastructure as a crucial factor for ICME, to include databases, software, and computational hardware [1]. Additionally, these tools enable workflows that properly integrate models and experimentation at each stage of the materials development lifecycle. Figure 1 schematically shows such a workflow for optimization of microstructure and properties in a titanium forging.

S. P. Donegan (✉)

Air Force Research Laboratory, Materials and Manufacturing Directorate, Wright-Patterson Air Force Base, Dayton, OH, USA

e-mail: sean.donegan@us.af.mil

M. A. Groeber

Air Force Research Laboratory, Materials and Manufacturing Directorate, Wright-Patterson Air Force Base, Dayton, OH, USA

The Ohio State University, Department of Integrated Systems Engineering, Columbus, OH, USA

© Springer Nature Switzerland AG 2020

S. Ghosh et al. (eds.), *Integrated Computational Materials Engineering (ICME)*,

https://doi.org/10.1007/978-3-030-40562-5_2

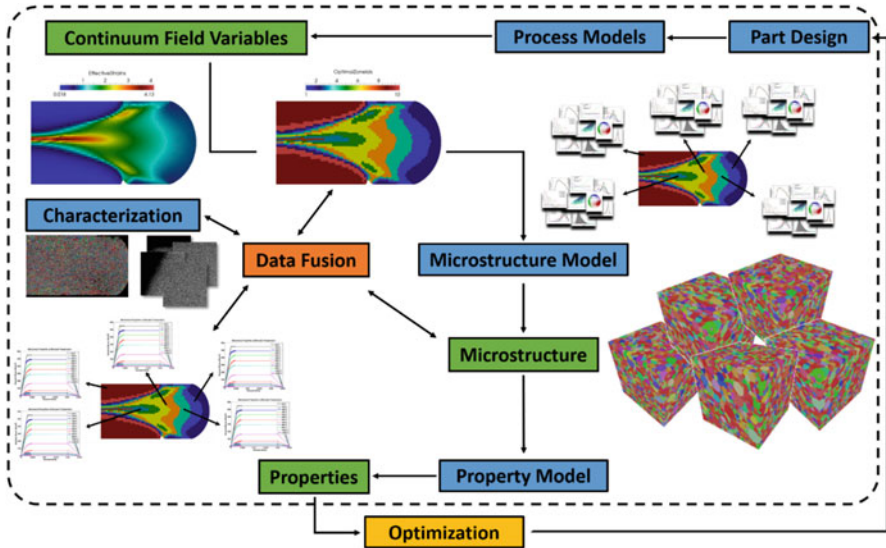


Fig. 1 Schematic of an ICME workflow for optimizing the microstructure and properties in a titanium forging. Blue boxes represent data generation tools, while green boxes represent output information from said tools

In the workflow shown in Fig. 1, an initial part design serves as an envelope for a forging process simulation, which yields *continuum field variables*: materials information, such as temperature or strain, which vary as a function of space and time. These variables feed a data-driven model that *zones* the component geometry, identifying those regions that have undergone a similar process history. Features of the process zones, defined by their constituent continuum field variables, serve as input to a microstructural evolution model. This process yields mean field microstructural measures, such as grain-size distribution and texture, at each zone. In turn, this microstructural information feeds a property model, predicting mechanical behavior for each zone. This mechanical information is finally looped back to the designer, informing modifications of the overall component geometry. Additionally, the model outputs are continuously validated by fusion with characterization measurements. Note the interplay between model and experimental data at each stage of the workflow and the transition of information across length and time scales. The cornerstone of an effective ICME workflow tool is the ability to seamlessly integrate these information streams, allowing an investigator freedom to explore the complex materials design space.

Designing and implementing ICME software tools is complicated by the variety of data streams available for modern materials research. Key features that define the breadth of ICME data include:

- *Geometry*: Simulation and characterization methods are capable of producing spatial data organized on varying topologies. These include unstructured point clouds, surface and volumetric meshes, and image/grid-like geometries.

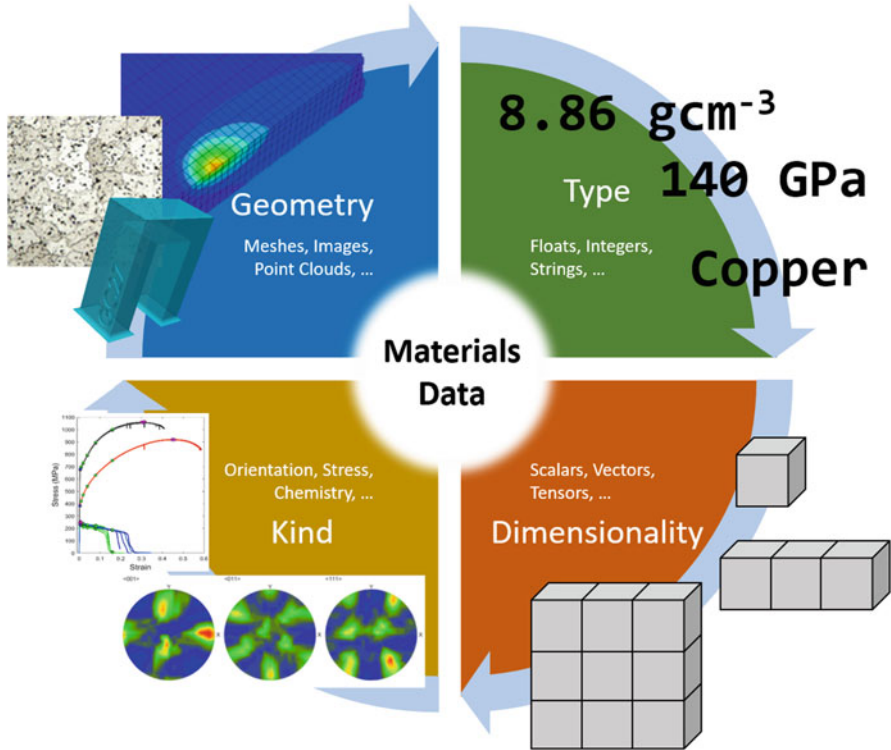


Fig. 2 Schematic of the four key features that define ICME data. Note that a single dataset will often exhibit variation in all four quadrants

- *Type*: Data streams may be of any fundamental data type, such as floating point numbers, integers, or strings, each with various precisions or encodings.
- *Kind*: The data may be representative of various materials phenomena. For example, spectroscopy measurements represent chemical composition, whereas crystal plasticity simulations output localized stress and strain tensors.
- *Dimensionality*: Materials data are inherently multidimensional, both in space, time, and kind. Second-rank tensorial data contains up to nine unique elements, whereas image intensity values are scalar.

Figure 2 graphically shows these four key features with schematic examples.

The challenge of ICME software development is properly generalizing to capture such disparate data streams in a cohesive manner. Once catalogued together, the data can be used to develop analyses that extend beyond the limits of single modalities. For example, validating ICME models requires coupling the model outputs to experimental measurements. Enacting this process robustly is a complex workflow challenge that requires a core structure capable of handling the different data streams. This chapter seeks to elaborate on such challenges in ICME software development. It is organized as follows:

- A select overview of various ICME software tools currently in use, including both commercial and open-source solutions
- A description of the requirements that a successful ICME workflow manager must meet
- An example implementation of such a workflow manager
- Presentation of a case study that highlights the utility of an ICME software infrastructure for solving modern materials problems

This chapter does not discuss details regarding data storage or infrastructure systems, such as Materials Commons [2], The Materials Data Facility (MDF) [3], or the Materials Project [4], or visualization tools, such as ParaView [5]. Instead, the focus is on tools used to generate and analyze materials data in an ICME context.

2 ICME Software Tools

We consider the following general categories for ICME tools: *simulation*, in which a physics-based model is used to generate information about a material process, evolution, or behavior, and *analytics*, where simulation and characterization data are postprocessed to produce additional information streams. A key distinguishing feature in this definition of simulation tools is the use of physics-informed models. Analytics tools may also be used to model materials, but we distinguish these from simulation tools as being *data-driven*. Such data-driven approaches typically use characterization or simulation data to fit surrogate models that approximate the underlying material physics without the need for explicit parameterization.

3 Simulation Tools

Materials modeling and simulation has a rich history that extends beyond the genesis of ICME. Within metals processing, DEFORM[®] has been commercially used since the early 1990s to simulate hot forging processes in both 2D and 3D [6]. Further capabilities include simulation of cold forming, machining, heat treatments, and microstructure evolution [6]. Similarly, ProCAST is a commercially available simulation package for casting processes, with support for die casting, investment casting, and continuous casting [7].

Behavior modeling of structural materials typically consists of solving a set of constitutive equations with supplied boundary conditions using a numerical method. The most commonly used numerical approach is the finite element method (FEM), in which a material volume is discretized into distinct elements on which local solutions are computed. Commercially available FEM packages include Abaqus [8] and ANSYS [9], both of which are used extensively within the aerospace supply chain to simulate material response. Several open-source FEM solutions also exist. Albany is a modular, general FEM solver for partial differential equations built

using reusable libraries [10, 11]. The Multiphysics Object Oriented Simulation Environment (MOOSE) is a similar package that relies on a generic software architecture, utilizing Jacobian-free Newton-Krylov methods [12, 13]. Other approaches besides FEM exist for solving systems of partial differential equations. For example, problems may be recast into convolutional forms, allowing for solutions using spectral (Fourier-based) solvers. Examples include simulating the elastic response of composite materials [14, 15], eigenstrains in thermal barrier coatings [16], and the viscoplastic response of polycrystals [17, 18]. The Düsseldorf Advanced Materials Simulation Kit (DAMASK) implements the spectral approach for solving the polycrystalline elasto-viscoplastic problem in an open-source format [19].

First principle and small-scale simulation tools also have a wide use within research and development. The Vienna Ab initio Simulation Package (VASP) is a broad toolset for electronic structure calculations, with capabilities for computing energy functionals, optical properties, and many-body problems [20]. The Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) is an open-source tool developed by Sandia National Laboratories for molecular dynamics problems [21]. For investigating dislocation dynamics, the open-source ParaDiS tool is available from Lawrence Livermore National Laboratory [22].

While the above software packages are only a small sampling of the toolsets available to a materials researcher, their outputs produce highly disparate data streams. For example, data from an FEM simulation are topologically organized onto a mesh, which may consist of a variety of unit element types (triangles, quadrilaterals, tetrahedra, hexahedra, etc.). However, a spectral solver requires data on a regular grid. Data may even exist on line segments, as in dislocation dynamics, or points, as in molecular dynamics. Additionally, data may be scalar (e.g., temperature fields from a DEFORM[®] forging simulation), vector (e.g., atomic displacement vectors from LAMMPS), or tensorial (e.g., strain tensors from polycrystalline viscoplasticity evaluated using DAMASK). Simulation data are also typically time dependent; this results in an additional dimension, which, in certain models, may also result in a change in geometry. The same variety of data is observed for characterization information. For example, atom probe tomography yields information about points in space (i.e., atoms), while computed tomography produces volumetric images. Electron backscatter diffraction (EBSD) scans yield orientation data on regular grids, which can be represented in only three numbers. However, the original Kikuchi patterns, which are of significant interest in applications such as high-resolution strain imaging, may be images that are upwards of 1024×1024 in dimension. If storage of these original patterns is desired, then the EBSD scan would store a pattern image at each grid location. The diversity of data forms generated by simulation packages and characterization techniques presents a unique integration challenge for downstream analytic tools.

3.1 Analytic Tools

Unlike the wealth of tools available for materials simulation, software specifically for materials analytics is a relatively nascent field. Historically, processing data

from materials characterization or simulation was typically accomplished using bespoke solutions tailored to a particular problem type. However, with the advent of relatively inexpensive computational infrastructure, availability of modern statistical and machine learning algorithms, and the popularity of open-source development, several tools for materials data analytics have gained traction within the industrial and research communities.

Since much of the characterization data collected in materials research take the form of n-D images, many analytics tools have been developed specifically tailored for image processing. Avizo™ is a commercial software package that provides image processing and analytics capabilities for materials images, including segmentation, computation of feature statistics such as size and shape, and meshing [23]. Similarly, the commercial GeoDict® software provides solutions for computed tomography processing, fiber analysis, and synthetic composite simulation [24]. For 3D EBSD data, ESPRIT QUBE commercially provides solutions for reconstruction and alignment, misorientation segmentation, and texture analysis [25].

Several open-source tools provide more general capabilities than the commercial products described above. The Materials Knowledge System in Python (PyMKS) is an open-source Python framework intended to provide data science approaches for solving various materials problems [26]. PyMKS has support for a variety of analytics tailored to materials, such as microstructure quantification using 2-point statistics [27] and fitting surrogate convolutional kernels to FEM data, producing highly accelerated elastic models [28]. A similar toolset is the Materials-Agnostic Platform for Informatics and Exploration (Magpie), an open-source Java-based library for fitting various machine learning models to materials data [29]. Specifically for texture analysis, the MATLAB toolbox MTEX provides capabilities for plotting pole figures, segmenting grains, and computing orientation distribution functions [30, 31]. Also leveraging MATLAB, the Materials Image Processing and Automated Reconstruction (MIPAR™) software provides proprietary routines customized for 2D and 3D materials image analysis [32].

3.2 Example Tools from Other Fields

Materials is not the only field that must contend with multiscale, multimodal, hierarchical information. Specifically, the medical and biomedical communities often handle multimodal information streams, however with a focus on n-D images. One of the most widely used tools for scientific biomedical image analysis is ImageJ [34, 35]. Publicly funded by the National Institutes of Health, ImageJ is a Java-based library and application that contains a wide variety of common and advanced image processing methods. Fiji is a popular open-source distribution of ImageJ that contains several additional plugins for advanced image analysis and segmentation [36, 37]. Another popular library for medical image analysis is the open-source Insight Segmentation and Registration Toolkit (ITK) [38, 39]. ITK, by taking advantage of generic template programming techniques in C++, provides highly

flexible image processing techniques applicable to n-D images, including robust approaches for multimodal image registration. ITK utilizes a pipeline construct to build workflows for image processing problems and is particularly suited to processing 3D medical imaging modalities, such as computed tomography and magnetic resonance imaging. ITK on its own is a pure library; the open-source 3D Slicer application provides a graphical front end to many ITK functionalities, including registration, with capabilities for 3D visualization and volume rendering [40, 41]. 3D Slicer leverages the Visualization Toolkit (VTK), which provides a platform-agnostic rendering engine along with a wide variety of geometric processing tools, such as connectivity, smoothing, and mesh fairing [42, 43].

Another open-source software tool for analyzing biomedical information is SCIRun, supported by the Center for Integrative Biomedical Computing [44, 45]. SCIRun provides a graphical programming interface for building simulation and analysis workflows tailored to biomedical data, with a focus on bioelectric fields. This visual programming approach is similar to the interface paradigm adopted by DREAM.3D. For application-agnostic data mining and machine learning tasks, the open-source Orange application provides a visual programming front end built on top of Python's rich set of available analytics libraries [46, 47]. A tool with similar capabilities to Orange is the open-source Java-based Waikato Environment for Knowledge Analysis (Weka) [48, 49]. Weka provides several machine learning functionalities, and it also provides plugin support for data-driven image segmentation in Fiji.

The above examples motivate a more generic need for extensible data processing and handling in scientific analysis. As the materials community begins to broadly adopt the ICME paradigm, it is prudent to take advantage of the strides made in other fields in implementing workflow tools, particularly in medical and biomedical image analysis. Leveraging the lessons learned from these previous tools can accelerate the development of materials applications, allowing for the allocation of development resources toward addressing fundamental materials data problems that are not shared in other fields.

4 Building an Extensible ICME Data Schema and Workflow Tool

We now consider the critical aspects that define a successful ICME workflow tool: a scalable, efficient data structure; modularity and plug-and-play capability for building workflows; and standardized data access and metadata labeling. The primary interest is for processing data that is accessible via a spatiotemporal index. We refer to this sort of data in general as *field data*. These kind of data are naturally generated by many types of materials characterization and simulation approaches. Note that we do not directly consider scalar material properties, such as thermophysical constants. While these constants are integrally important

for automated materials discovery workflows, the present focus is placed on the processing and analysis of materials field data.

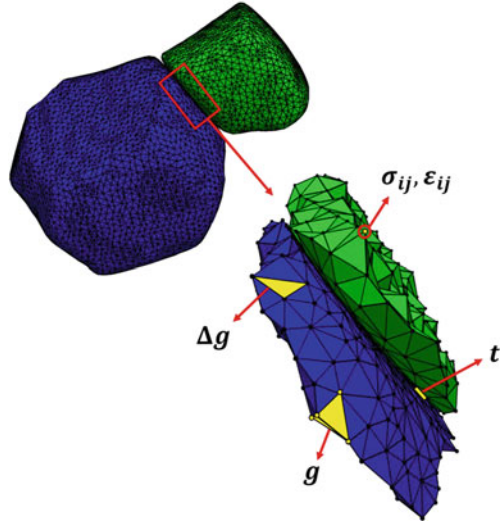
4.1 Data Handling Requirements

Given the wide variety of data types produced by characterization and software tools, a successful implementation of an ICME workflow manager must utilize a flexible data structure to properly ingest and handle the different materials information streams. The previously introduced critical aspects that help define the diversity of materials data are: *geometry*, *type*, *kind*, and *dimensionality*. An ICME data schema must implement a structure capable of handling variety in each of these categories. Practically, this translates to a requirement to represent data on different topology types, including point clouds, 2D and 3D meshes, and regular and irregular rectilinear grids. A crucial component of representing these geometries is a capability to store them together within a consistent spatial reference frame, which enables direct correlation between different datasets. Such correlative workflows are a cornerstone of effective ICME, allowing for direct validation of simulation results using experimental measurements or simultaneous analysis of multimodal information.

A result of this capability is an additional requirement to store data on any *unit element* that composes a geometry. More generally, an effective data structure should be flexible enough to store information on any component of a set of simplicial complexes. Figure 2 showcases a simplified example of this need for output from a crystal viscoplasticity simulation. Output from such a simulation may be geometrically represented as a set of connected tetrahedra that tile the 3D volume. Resulting stress and strain tensors could be stored on the vertices of the tetrahedra, while crystal orientations might be stored on the tetrahedra themselves. Additionally, further connectivity analysis may require information storage on the faces or edges that compose the tetrahedra. For example, it may be advantageous to store information about misorientations across grain boundaries, which would naturally be stored on tetrahedral faces. Similarly, identification of triple lines would be stored on tetrahedral edges where three grains meet. Importantly, what kind of information, and where it should be stored, may not be known a priori for any given problem; thus, the data structure should be extensible enough to handle changing user requirements.

While cursory, the example in Fig. 3 demonstrates the requirement for flexibility in geometric data storage. It also communicates a need for efficient storage of multidimensional information. Triple line identifiers are scalar, while orientations and full misorientations are at least three components. Stress and strain tensors, as symmetric second-rank tensors, require storage of at least six components. In principle, the number and shape of components is arbitrary, tailorable to the specific application or analysis workflow. A common example of this specificity is the number of time steps for a given simulation, which imposes an additional dimension

Fig. 3 A schematic example of two grains from a crystal plasticity simulation, each meshed by a set of tetrahedra. The data structure should be capable of storing information on all unit elements of the mesh: points, edges, faces, and tetrahedra, with support for various component dimensions. The data in this example are stress and strain tensors (σ_{ij} and ϵ_{ij}), triple line identifiers (t), misorientation (Δg), and grain orientation (g)



on the data. Thus, a scalable approach to storing multicomponent data is necessary. Another practical requirement is the fundamental type used to represent the data. Identifiers may be stored as integers, while orientations may be stored as floating point numbers. Precision may also vary (e.g., 32-bit or 64-bit floating point), while integers must consider being signed or unsigned. Data type handling is partly tied to the implementation: certain languages, such as C and C++, utilize strong typing, whereas Python utilizes the weaker duck typing. Regardless, a successful data schema must be capable of representing data of various types, with adjustments as needed for the given language.

A final data handling requirement stems from the natural structure of materials. Materials are inherently hierarchical: physical phenomena couple across multiple length scales to yield observed behavior and performance at the macroscale. Representing this natural hierarchy is critical to fully capturing the space of materials information. Figure 4 shows an example of this hierarchy for a cast Ni-base superalloy blade. Note the inherent coupling and reciprocity across the scale continuum. An ICME data structure must be capable of allowing users to efficiently move across these scales; thus, a mapping scheme is required that shifts reference up and down the hierarchy. For field data, this translates to an ability for arbitrarily grouping the various simplicial complexes that comprise the data geometry, which can then be continuously grouped further until all data are members of a unifying set.

4.2 Modular Workflow Requirements

Beyond efficient storage of materials field data, a user should be capable of interacting with that data through a standardized interface. Ideally, this interface

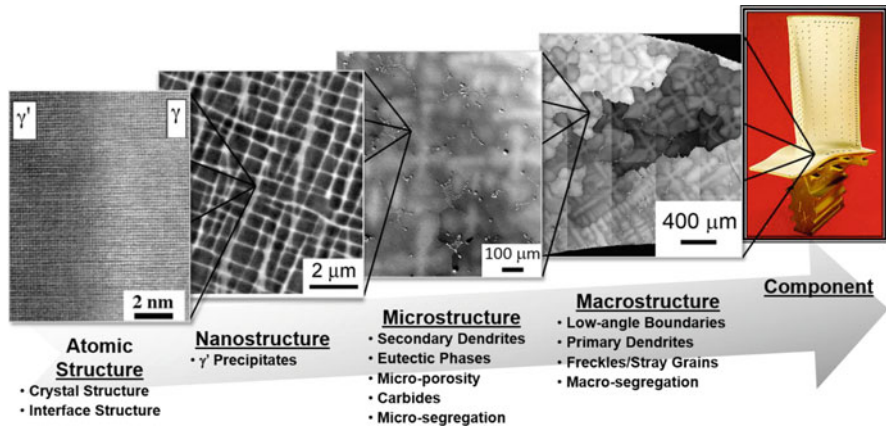


Fig. 4 An example of the hierarchy of materials structure in a cast Ni-base superalloy blade. Note that the physics of one scale are tightly coupled to the phenomena at another scale. Properly composing this multiscale and hierarchical information together requires a flexible and extensible data structure. (Figure reproduced courtesy Dennis Dimiduk and Michael Uchic)

defines the parameters through which the data structure may be accessed, to include: creating new data structure objects, interrogating the properties of existing objects, and modifying existing objects as needed for the present context. To facilitate this interaction, it is desirable to not only standardize the application interface for the data structure itself but also the functional interface by which such interactions are implemented. This functional interface should enable the storage and retrieval of each parameter setting, which is needed for workflow archival and reproducibility. Thus, a workflow for analyzing a collection of materials field data can be conveniently represented as a sequence of these standardized functions.

Representing a workflow in this manner also immediately satisfies an additional requirement for flexibility: since materials problems and the data informing them are constantly evolving, users should not be restricted in building ICME workflows. By composing a workflow from self-contained functions, a user is free to add, delete, swap, and move these functions as appropriate for the given application. This flexibility imposes a complication for testing internal consistency for a constructed workflow. Confirming a workflow is valid for a given set of parameters is trivial if the workflow is constructed a priori. But for on-the-fly development, explicit validation is subject to combinatorial explosion as the number of available functions increases. To solve this issue, the functions within an ICME workflow manager must be capable of performing self-consistency checks, pursuant to the overall application interface of the global data structure. Thus, the overall workflow can be validated by examining the consistency of each individual function.

After construction of a workflow, it is desirable to serialize the workflow. This addresses two needs: the ability to archive workflows for future use and the ability to share workflows with collaborators. An important aspect of this serialization process

is the storage of the parameter settings used in each function of the workflow. Since the constructed workflow may be complex, and itself hierarchical, a sufficiently flexible format must be used for serialization. Ideally, this format would also be human-readable and should be standardized for broad access outside of materials-specific tools.

4.3 Data Access and Metadata Labeling Requirements

After a successful execution of a workflow, the computed data should be serializable into an accessible data format. Since the implemented data structure is highly generic and flexible, the chosen data format must also be equally flexible. Similar to the format for workflow serialization, the data format must also belong to an open standard, allowing access outside of materials toolsets. Ideally, this format should also be size and speed efficient, allowing for fast reading and writing, an enabler for the large datasets inherent in many ICME workflows. Stored information may be either *heavy*, the dense data comprising the bulk of the content in terms of size, or *light*, consisting of metadata such as material name, component dimensions, data type, time step, etc. The format must therefore be capable of storing either heavy or light data. It is desirable to store the processing history of a set of data along with the data itself; thus, the history remains innately coupled with the data, which allows for reproducibility and transparency. Therefore, the data format must enable the workflow to be stored alongside the data and additionally allow for further functions and their parameter settings to be appended to the workflow should future processing be necessary.

5 SIMPL and DREAM.3D: Enabling ICME Workflows

To satisfy the above requirements for an ICME workflow tool, the Air Force Research Laboratory in partnership with BlueQuartz Software developed a software framework known as the Digital Representation Environment for the Analysis of Microstructure in 3D (DREAM.3D). DREAM.3D is an open-source software tool explicitly designed to enable the creation of generic materials analytics workflows that are adaptable to any kind of input, regardless of geometric topology or data type [33]. Specific capabilities include 2D and 3D EBSD reconstruction and analysis, n-D image processing, feature identification and quantification, surface meshing, texture analysis, and synthetic microstructure generation. DREAM.3D's unique capabilities stem from an underlying data structure and management library: The Spatial Information Management Protocol Library (SIMPL). SIMPL is an open-source C++ library that implements an abstract data structure, including a well-defined application programming interface [50]. Additionally, SIMPL defines a functional interface for interacting with the data structure. This interface is

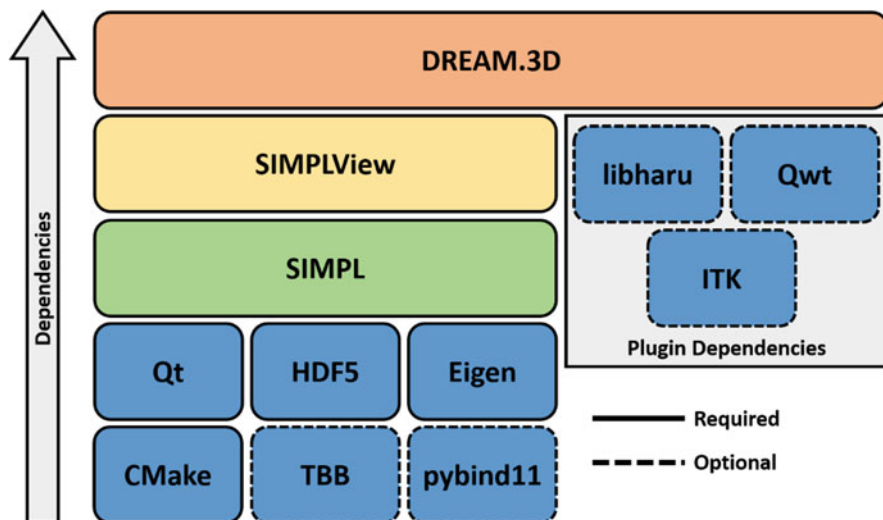


Fig. 5 The dependency tree for DREAM.3D. Items in blue are open-source dependent libraries. Note that this modular design allows for libraries to be added or swapped where necessary, granting flexibility to the overall software architecture

characterized by *filters*, self-contained functions that perform a unit operation on the data structure state. Filters may be sequenced to form a *pipeline*, the fundamental execution unit of a SIMPL workflow. SIMPL also allows for extensions via a *plugin* interface. Users may add their own functionalities to SIMPL by adhering to the plugin architecture. DREAM.3D constitutes an open-source collection of SIMPL plugins tailored for analysis of materials data, along with facilities for processing materials-specific information [33, 51]. Additionally, DREAM.3D utilizes a graphical front end called SIMPLView [52]. All the various projects associated with DREAM.3D are distributed under the permissive 3-clause BSD license. This open-source development has enabled collaborations and contributions across academia, government, and industry.

Figure 5 shows the overall software architecture of DREAM.3D, including dependent libraries. Dependencies generally progress up from the bottom of Fig. 5. SIMPL makes heavy use of the Qt library for various functionalities, such as container objects, string representations, and platform-agnostic file system access [53]. Additionally, Qt provides the facilities for producing the front-end graphical interface in SIMPLView. SIMPL utilizes the HDF5 file format and library for data serialization [54]. Eigen is leveraged for highly efficient linear algebra and matrix manipulations [55]. Optionally, Intel’s Threading Building Blocks provides thread-based parallelism [56], while pybind11 automatically creates Python bindings for SIMPL classes and filters [57]. For all projects, CMake is used to enable easy cross-platform building [58].

A key component of the DREAM.3D software architecture is its modular nature; this allows for dependencies to be added or swapped as needed for a given application. Most commonly, this approach is taken for adding new plugin dependencies. For example, an image processing plugin in DREAM.3D leverages ITK as an underlying dependency, bringing the power and flexibility of a tool originally designed for medical image analysis into the materials domain.

The following sections overview the structure of both SIMPL and DREAM.3D, including: data structure; filter, pipeline, and plugin infrastructure; graphical interface; and analysis capabilities.

5.1 SIMPL Data Structure

The SIMPL data structure is inspired by approaches in other well-known libraries, such as VTK, along with methods in combinatorics and topology. The data structure was designed to directly address those requirements stated in the Data Handling Requirements section above. Principally, the data structure takes the form of a tree; since trees are hierarchical, the data structure is able to naturally conform to the grouping requirements needed for materials data. Items within the data structure are generally termed *objects*, with four primary types of objects available:

- *Data Container Array*: The root node of the data structure. The data container array has access to create and retrieve all descended children objects and check the structure for validity.
- *Data Container*: The direct descendant of data container array, data containers store attribute matrix objects that correspond to a *unique* geometry. Data containers are therefore distinguished by what geometry they represent.
- *Attribute Matrix*: Stored within data containers, attribute matrices contain the objects that store the dense data on each geometry. Attribute matrices are distinguished by a type identifier which signifies at what specific grouping hierarchy the data should be associated. Additionally, attribute matrices define the shape of the underlying dense data.
- *Attribute Array*: Attribute arrays are the leaves of the data structure tree and store the heavy field data for a given dataset.

Objects within the data structure have an associated name; similar to a standard file system, no two objects at the same level of the tree are allowed to have the same name. Additionally, objects deeper within the tree have a unique *path*, the concatenation of all parent object names with the child. An example data structure is shown in Fig. 6. In this example, two data containers are stored in the data container array, one that represents an image and one that represents a surface mesh.

Geometries are a special kind of data structure object, represented by the red boxes in Fig. 6. A data container may only store one geometry, and usually this geometry is unique within the overall data structure. The child attribute matrices

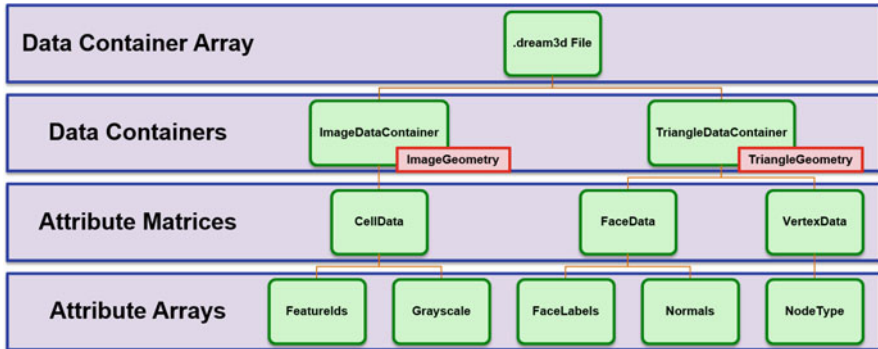


Fig. 6 An example SIMPL data structure, representing storage of an image and a surface mesh. (Figure reproduced from the DREAM.3D user manual)

and attribute arrays for a given data container store information that corresponds to the data container’s geometry object. Geometries are distinguished by the dimensionality of the fundamental unit element that serves as that geometry’s primary building block. There are four primary unit element types: vertices (0-dim), edges (1-dim), faces (2-dim), and cells (3-dim). In principle, the data structure allows storage of higher dimensional simplices; however, for materials data analysis, it is rare for such higher dimensional elements to be needed. For a given geometry, data may be stored on any of the unit elements that comprise the geometry, as shown in Fig. 7.

Data may be stored on any of the unit elements that form a given geometric object. For example, if the fundamental geometric object is a quadrilateral, data may be stored on the vertices, edges, or faces of the polygons, but not cells, since no object within the geometry is volumetric.

SIMPL defines an interface to which implemented geometries must adhere. This abstract interface class enforces that geometries store their connectivity, understand how to compute derivatives, import and export themselves, etc. Filters are able to leverage this generalization, which enables algorithms to operate across different geometries. Currently, SIMPL implements eight geometric classes, along with a special null geometry. These geometries are shown in Table 1.

Similar to the overall data structure, geometries in SIMPL adhere to a hierarchy, as shown in Fig. 8. Geometries may be generally categorized as either *structured*, where explicit definition of point coordinates is not needed, or *unstructured*, where point coordinates must be explicitly stored. The structured geometries are the image and rectilinear geometries, commonly referred to as *grids*. An n-D image is defined implicitly by just three values: its position in space, defined by the origin; the resolution along each dimension; and the number of elements in each dimension. Thus, an n-D image needs only $3n$ numbers to be fully defined. A rectilinear grid, however, may admit variable resolution along each orthogonal direction. For an

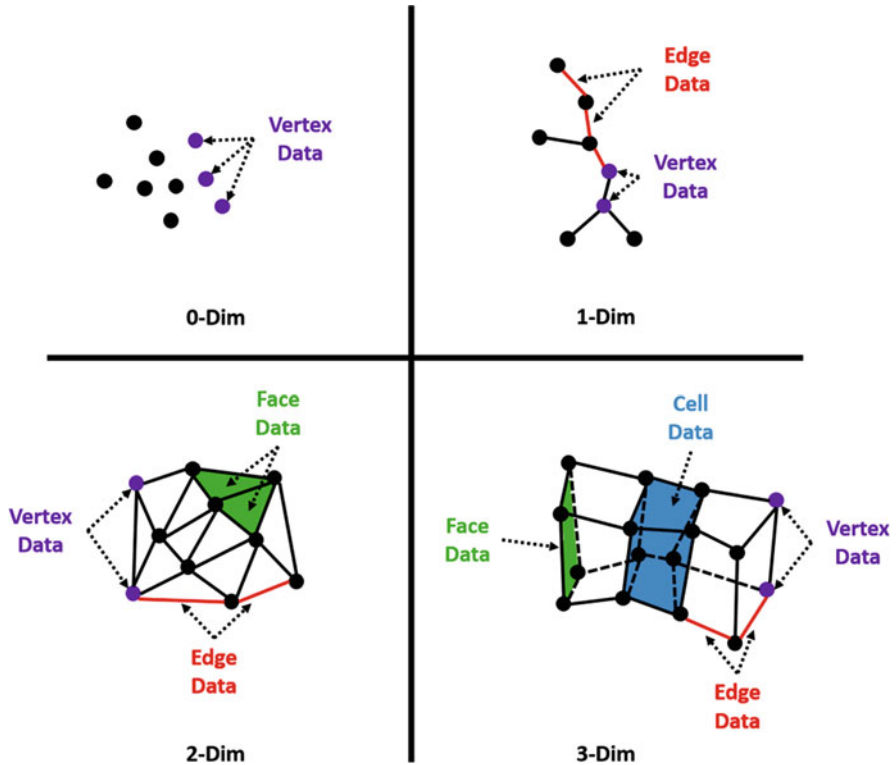


Fig. 7 Schematic showing building-block unit elements for SIMPL geometries. Note how data may be stored on any unit element that comprises a given simplex. (Figure reproduced courtesy Groeber and Jackson [33])

n-D rectilinear grid, the total number of values, v , needed for complete definition is given by the following equation:

$$v = \sum_{i=0}^n N_i + 1$$

where N_i is the number of elements along the i th dimension.

The unstructured geometries require explicit definition of point coordinates. Additionally, for any geometry other than vertex, the connectivity between points must be stored. SIMPL represents unstructured geometries using *shared element lists*. In the shared element list schema, only unique unit elements are stored; for example, if two triangles share an edge, then the two vertices that comprise that edge are shared, and need not be stored twice. Despite their simplicity, shared element lists offer a number of benefits: highly efficient storage of the geometric

Table 1 Currently implemented SIMPL geometries, along with their topology and associated data types

NAME	Topology	Associated data	Description
Unknown	N/A	Any	The null geometry, used when the underlying data have no spatial indices
Vertex	0	Vertex	A set of points, or point cloud
Edge	1	Edge/vertex	A set of edges, forming lines
Triangle	2	Face/edge/vertex	A set of triangles, forming a surface mesh
Quadrilateral	2	Face/edge/vertex	A set of quadrilaterals, forming a surface mesh
Image	3	Cell	A structured rectilinear grid composed of pixels/voxels of constant resolution
Rectgrid	3	Cell	An unstructured rectilinear grid, composed of pixels/voxels of variable resolution
Tetrahedral	3	Cell/face/edge/vertex	A set of tetrahedra, forming a volume mesh
Hexahedral	3	Cell/face/edge/vertex	A set of hexahedra, forming a volume mesh

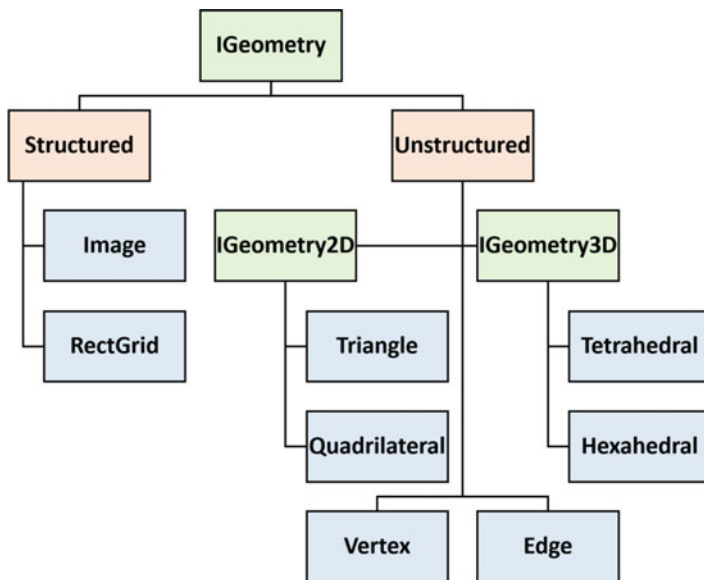


Fig. 8 The inheritance hierarchy of SIMPL geometries. By providing abstract interfaces, represented in green, SIMPL allows for generic algorithm programming, reducing code replication

information, fast static access of the geometry, and the ability to store nonmanifold simplices. However, shared element lists suffer from inefficient manipulations of the geometry. For example, a shared element architecture is not suited toward mesh refinement or decimation.

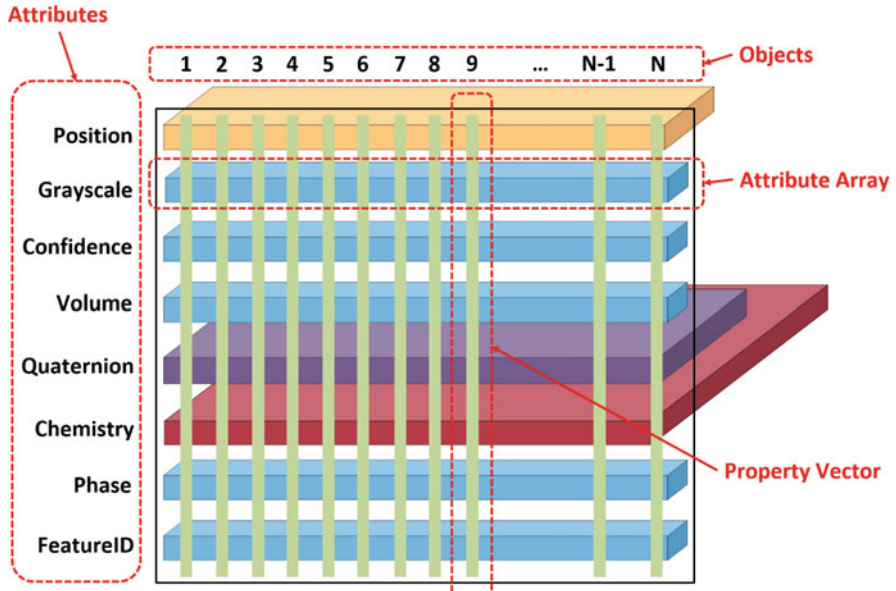


Fig. 9 Schematic of an $N \times 1$ attribute matrix. Objects may be elements, features, or ensembles. (Figure reproduced from the DREAM.3D user manual)

SIMPL represents hierarchy within a given geometric dataset through the concepts of *features* and *ensembles*. Features are groups of geometric elements. For example, an EBSD scan may have its pixels grouped into grains under some threshold for misorientation. Data may then be associated with these features, such as size, shape, and average orientation. Features may also be grouped into ensembles. In the EBSD example, grains may be grouped together based on their crystal structure. Note that ensembles may also be grouped together recursively; all subsequent groupings are also referred to as ensembles. Data are mapped between levels of the scale hierarchy using an identifier array; this array denotes to which feature or ensemble an object lower in the hierarchy belongs.

Associating data with geometries, features, and ensembles is organized through attribute matrices. Attribute matrices themselves do not store heavy data; instead, they serve to define the type of data being stored and its shape. Dense data are stored in attribute array objects, which are contained within attribute matrices. There are three general types of attribute matrices: *element*, which store data associated to the unit elements of a geometry; *feature*, which store data for groups of elements; and *ensemble*, which store data for groups of features. There are four types of element attribute matrices, corresponding to the four basic unit elements: vertex, edge, face, and cell. Other than a type, an attribute matrix also has a shape; in the SIMPL ontology, this shape is referred to as the *tuple dimensions*. Figure 9 shows an example attribute matrix of N objects, where the tuple dimensions are $N \times 1$.

For a raveled attribute matrix with scalar dimensions, the rows are comprised of specific attribute arrays, while the columns represent particular objects. This storage scheme generalizes to higher dimensions, where attribute arrays are stored in hyper-rows and objects are denoted by the hyper-columns. Note that an attribute matrix is extensible: new attribute arrays may simply be appended to the matrix without the need for resizing. Attribute matrices may be interrogated in either direction: obtaining arrays along (hyper)-rows, which return information about a given attribute for each object, or property vectors along each (hyper)-column, yielding a list of attributes for a specific object.

Attribute arrays, the (hyper)-rows of attribute matrices, are the final leaves of the overall data tree. These arrays store dense, heavy data. Arrays may be *multicomponent*, defining a depth dimension at each tuple. The overall array shape, tuple dimensions, is inherited from the dimensions of its parent attribute matrix. SIMPL allows for any fundamental data type to be stored within an attribute array, including various precision integers and floating point numbers. Attribute arrays are stored compactly within attribute matrices, even if the arrays do not share the same component dimensions: therefore, an attribute matrix is sparse in its depth dimension, as shown in Fig. 9.

The SIMPL data structure is highly flexible and customizable. In order to serialize it to storage, a data format must be used that is similarly flexible. SIMPL utilizes the Hierarchical Data Format, or HDF5, as its data format [54]. HDF5 is a binary file format whose data model allows for explicit hierarchy by organizing information into *groups*, similar to folders on a file system, while dense data are stored in *datasets*. SIMPL takes advantage of this model by mapping its data container arrays, data containers, and attribute matrices to groups in an HDF5 file, with attribute arrays being stored in datasets. An example mapping for a SIMPL data file is shown in Fig. 10. HDF5 is an open standard which enables easy cross-platform data sharing and data transfer to toolsets other than SIMPL or DREAM.3D.

Similar to SIMPL, HDF5 allows for dense data to have arbitrary shape and component dimensions and store any fundamental data type. Critically, the analysis workflow used to generate the SIMPL data structure is stored along with the data within the SIMPL file, enabling reproducibility and archival.

5.2 *Filters, Pipelines, and Plugins*

SIMPL defines a standard interface for interacting with the data structure through the concept of *filters*. A filter is simply a self-contained function that performs some operational interaction with the data structure, such as creating a new object (i.e., computing some new information) or modifying an existing object. Filters adhere to a standardized interface defined in an abstract base class. A critical feature of filters is their ability to request parameters from a user and translate these requests into queries of the data structure. For example, a user may select an attribute array by supplying a filter with a path; the filter interface will then utilize this path to

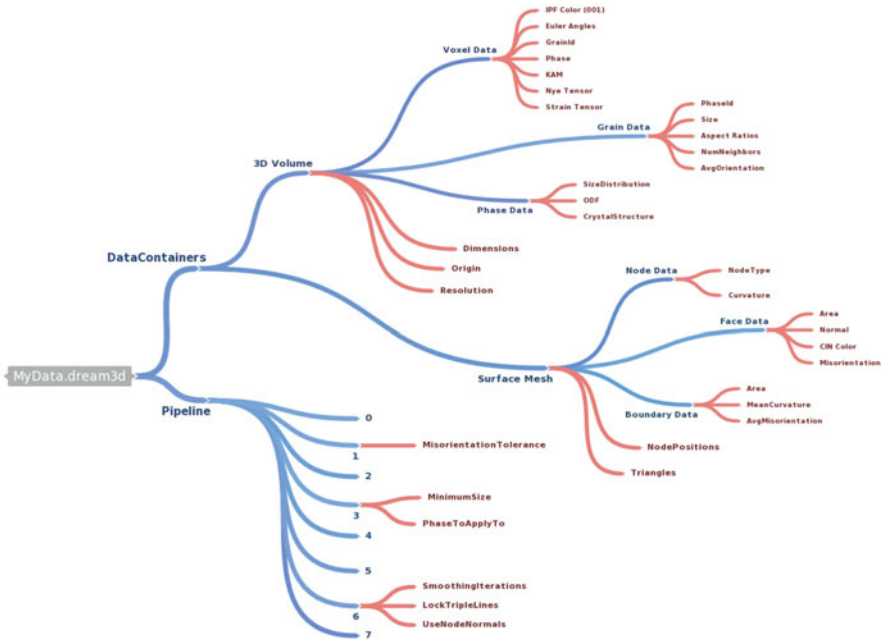


Fig. 10 Schematic of a SIMPL data structure storage scheme in an HDF5 file

access the array in the data structure and make it available for computation when the filter runs. Importantly, filters are also capable of performing validity checks when processing their parameters. This validation procedure is referred to as a *preflight* state. During preflight, a filter performs the necessary validation checks, modifying the data structure as needed *by proxy*: memory is not allocated and computations are not run during preflight, but any object creation or removal is represented in the data structure.

By creating a sequence of filters, a user instantiates a workflow that creates, modifies, and saves a SIMPL data structure. This sequence of filters is referred to as a *pipeline*. Pipelines orchestrate the task of requesting filters to preflight themselves, ensuring that the overall pipeline is in a valid state before allowing execution. Additionally, pipelines may be serialized using the JSON file format. JSON enables a human-readable transfer format for saving user-defined pipelines; the constituents of a JSON pipeline file are simply the sequence of filters with their explicit parameter settings. Note that a JSON file can be encoded as a string; this capability is leveraged to store the pipeline within the HDF5 SIMPL file schema as a string dataset.

SIMPL provides an interface for defining self-contained groups of filters called *plugins*. Programmatically, plugins are dynamically loaded libraries that comprise a collection of SIMPL filters, along with any additional support code necessary for the filters’ operation. DREAM.3D itself is simply a collection of SIMPL plugins

with capabilities customized for materials data analysis. Since all interaction with the SIMPL data structure is governed by a standardized interface, filters are able to leverage functionalities across plugins, which results in less code duplication. A canonical example is computing the size of features: this problem essentially reduces to summing the volumes of each constituent element that belong to a set of features. This computation is the same, regardless if the features define a set of grains or a collection of pores. Thus, when adding new filters via plugins, a developer can avoid re-implementing such a generic algorithm as size computation and instead focus on designing those functions that are specific to the problem space.

5.3 *SIMPLView: The Standard SIMPL Graphical Interface*

SIMPL on its own does not require a user interface. SIMPL may be used as a library, in which the data structure, filters, and pipelines would be accessible via code; or using a command-line interface. SIMPL does contain a set of pre-defined graphical widgets that allow developers to rapidly generate user interfaces. The most widely used implementation of this feature is SIMPLView [52]. SIMPLView is a basic interface to the functionalities of SIMPL, allowing users to construct pipelines using a visual programming style. SIMPLView is most widely recognized as the interface of DREAM.3D. Note, however, that SIMPLView may be swapped with any other graphical interface implementation as needed; therefore, the user experience is highly customizable to the particular application. SIMPLView is under active development; the current incarnation of the interface is shown in Fig. 11.

The major components of SIMPLView are comprised of *panes*: the toolbox pane, pipeline view, filter parameters, data structure, and pipeline issues and output. The toolbox lists all available filters, both alphabetically and categorized by functionality. Additionally, the toolbox stores *bookmarks*, which are the locations of saved JSON pipeline files. The pipeline view is the main focus window of the interface: here the user constructs a pipeline by building a sequence of filters. The filter parameters pane shows the user what variables may be set for a selected filter in the pipeline view. By using the data structure pane, the user may view the status of the current SIMPL data structure, inspecting attributes such as geometry types and array dimensions. Finally, any pipeline issues or output are shown in their corresponding panes. When constructing a pipeline, SIMPLView communicates the current pipeline state to SIMPL, which orchestrates the preflight procedure to validate the settings provided by the user. If issues arise, the user is presented with an error message in the pipeline issues pane. Recall that during preflight, no actual computations are undertaken; the user must finish constructing a valid pipeline that completes preflight without errors before execution is allowed. While SIMPLView exposes only the basic operations of SIMPL, it does offer customization features: all panes can be hidden or moved, and support is available for skinning the interface.

SIMPLView does not have present capabilities for visualization. Instead, SIMPL leverages the open-source ParaView application to provide visualization

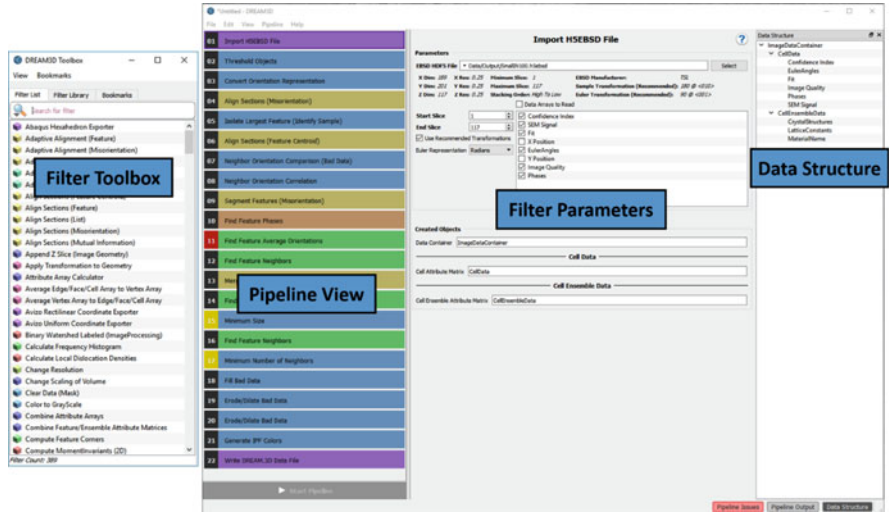


Fig. 11 The current version of SIMPLView, as implemented in the DREAM.3D distribution, with major components of the interface labeled. In this rendering, the pipeline issues and output are hidden

capabilities. When exporting a SIMPL data file, the user may elect to write a companion XDMF file, which is essentially an XML document that explains how the data are organized in the HDF5 file [59]. ParaView has facilities to import this XDMF file, allowing for efficient volumetric visualization of the data processed by SIMPL.

5.4 DREAM.3D: An ICME Workflow Tool

DREAM.3D is essentially a set of SIMPL plugins customized to process materials data. As such, it inherits all the benefits of the SIMPL data structure described above. DREAM.3D contains a number of capabilities, including:

- Import and export of a variety of file formats, including many image formats, FEM file formats, generic ASCII and binary files, HDF5 files, and the SIMPL file format
- Feature identification approaches, such as expectation maximization, connected components segmentation including metrics such as misorientation, and clustering approaches
- Texture analysis, including pole figure plotting, orientation distribution function sampling, disorientation and average orientation computations, and fundamental zone reductions
- Reconstruction of 3D data from 2D slices, for both images and crystallographic EBSD measurements

- Data processing and cleanup functionalities, including robust image processing courtesy ITK
- Statistical computations such as feature size and shape distributions, histograms, and distribution fitting
- Surface and volumetric meshing
- Instantiation of synthetic microstructures from morphological and crystallographic statistics, provided either from generative statistical models or experimental measurements

Leveraging the above functionalities in concert with the flexibility of SIMPL, a user can construct arbitrarily complex workflows for difficult ICME problems. In the following section, such a problem is introduced as a case study to demonstrate the utility of DREAM.3D.

6 Case Study: Ti-6242Si Pancake Forging

This section presents a case study for an ICME workflow concerned with quantitatively relating processing parameters in a titanium disk forging to measured microstructure characteristics. Unless otherwise noted, all processing and analysis steps presented were performed using DREAM.3D. This problem is a subset of the workflow shown in Fig. 1. Specifically, we are interested in those procedures shown in Fig. 12. A cylinder of Ti-6Al-2Sn-4Zr-2Mo-0.1Si (Ti-6242Si) with diameter 25.4 mm and height 38.1 mm was forged into a pancake with an average true height strain of 1.07. After forging, the pancake was cross-sectioned radially and characterized using both backscatter electron (BSE) imaging and EBSD. Concurrently, the forging process was simulated using DEFORM[®]. This forging was part of a larger study in which additional cylinders were excised from the same parent billet and isothermally compressed. Specific experimental and simulation details may be found in Pilchak et al. [60].

This work was motivated to develop quantitative relationships between process history and resulting microstructure. Specifically, microstructure features of interest are microtexture regions (MTRs). MTRs are relatively large (i.e., millimeters, to centimeters) regions of similar crystallographic orientation that form in near-alpha titanium alloys [61]. These regions have been identified as prime factors implicated in dwell fatigue debits of titanium forgings [62, 63]. In order to understand the impact the process state has on MTRs, the model output from DEFORM[®] was fused with the characterization data. Then, the model data was zoned using an approach from unsupervised machine learning, partitioning the forging geometry into discrete regions of self-similar processing history. With the characterization information colocated with these zones, an assessment can be made concerning the types of microstructure expected for a given process. This zoning procedure has the additional benefit of signifying to a designer which regions of a component

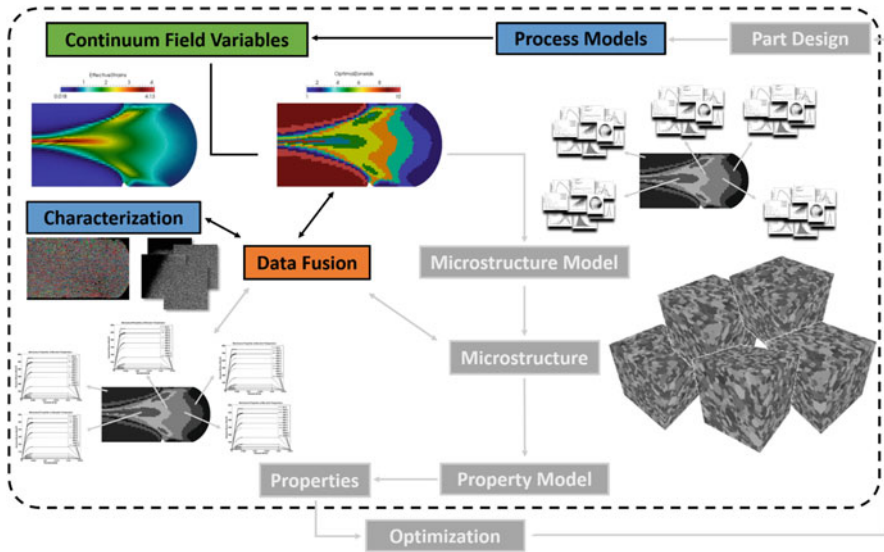


Fig. 12 The specific workflow items for the titanium disk case study. Model data from a process simulation produce continuum field variables, which are then reduced to zones. These data are fused to characterization measurements, allowing for microstructure to be assessed per zone

are most distinct, serving as an indication for where additional measurements may be prudent. For example, a zoning scheme may indicate where samples should be excised for further mechanical testing. Here, we focus on the problem of relating the zoned geometry to microstructure, instead of determining an optimal component cutup.

6.1 Zoning Process Histories

The pancake forging process was simulated using DEFORM[®]. Since the geometry is radially and axially symmetric, a single half cross-section was simulated in 2D. The underlying quadrilateral FEM mesh was dynamically remeshed as appropriate as the component was strained. DEFORM[®] optionally allows for tracking the initial material points until the final time step. The total evolution of an example pancake forging model is shown in Fig. 13.

For the present analysis, several field values were computed, including effective strains, full stress and strain tensors, and damage accumulation. Together, these fields yield a high-dimensional description of the forging process history. We wish to develop a scheme that effectively zones this process history into discrete categories. In the parlance of machine learning, this procedure can be considered an application

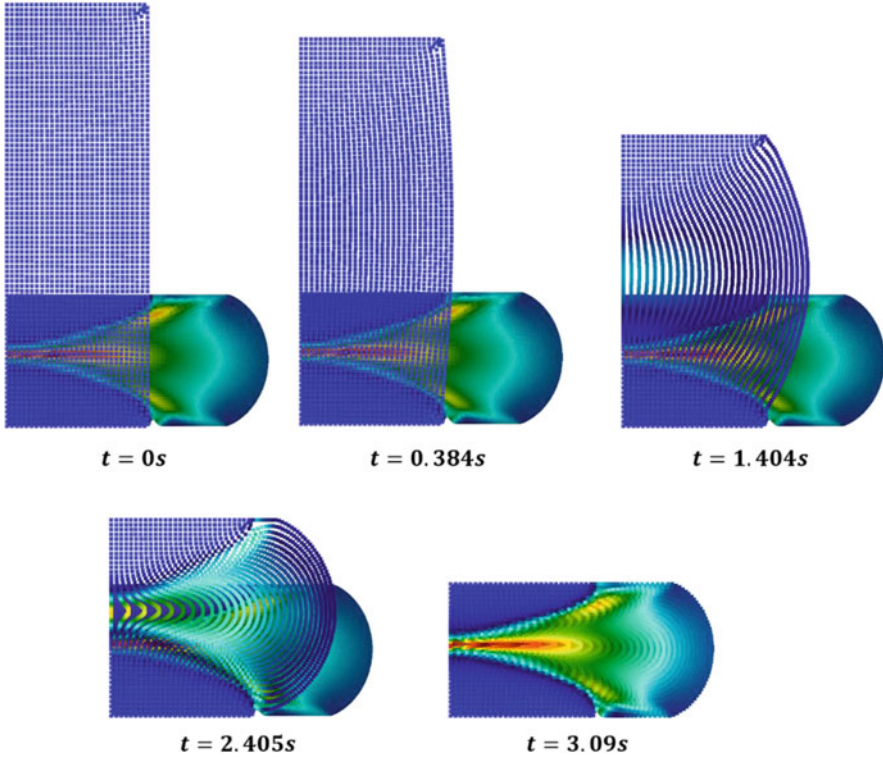


Fig. 13 Evolution of an example 2D DEFORM[®] pancake forging model. The points represent the tracked material points across all time steps, while the underlying mesh is for the final time step. Elements are colored by their effective strain

of cluster analysis. Cluster analysis seeks to classify an input space such that points that are most similar, according to some metric, are categorized in the same cluster [64]. In the context of zoning, the clustering procedure would ideally be performed on a latent space that best captures the process history relevant for the given response. For this example, that response is the representation of microstructure obtained from the EBSD and BSE measurements. While the relevant process history may be some linear combination of the various time-dependent fields produced from the DEFORM[®] simulation, we demonstrate a zoning procedure using only the strain tensors from the final time step for simplicity. For the relevant simulation, these strains are shown in Fig. 14. While the overall sample volume is conserved, individual element volumes may not be preserved; hence, the 2D strain tensor at a given element may not be uniquely symmetric. Thus, for this example, all four components of the 2D strain tensor are used.

To produce a zoning of the strain tensors, we utilize the k -medoids algorithm from cluster analysis [65]. k -medoids labels a set of points into k classes such that each datum is placed into the cluster with the closest medoid value, where the

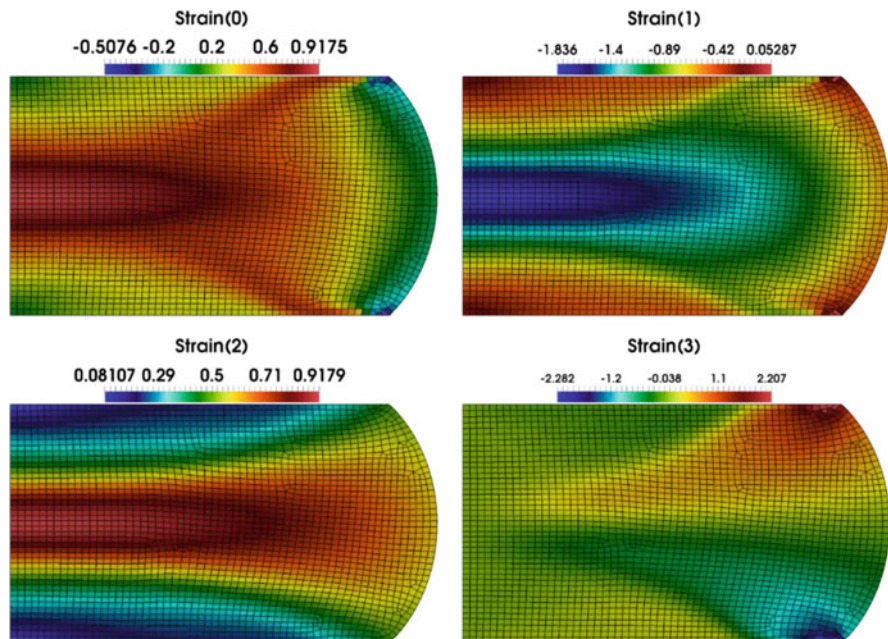


Fig. 14 The strain components from the final time step of the DEFORM[®] simulation. The underlying mesh has been emphasized to better show the quadrilateral geometry

medoid is a representative datum for that cluster. This approach is reminiscent of the classic k -means algorithm, in which data are placed in clusters with the closest cluster mean [66]. For a set of d -dimensional data points $\mathbf{X} = \{x_i\}$ and k clusters $\mathbf{C} = \{c_k\}$, k -medoids attempts to solve the following minimization:

$$\min_{\mathbf{C}} \sum_{i=1}^k \sum_{x_i \in c_k} d(x_i, m_k)$$

where m_k is the medoid of cluster c_k and $d(a, b)$ is some distance metric between points a and b . A benefit of k -medoids is the ability to customize the choice of metric, which is useful for problems where the standard l_2 norm may be inappropriate. This optimization is computationally intensive; however, several heuristic algorithms exist that perform well in practice. We utilize the partition around medoids algorithm, which iteratively minimizes the total distances within each cluster by recursively checking medoid candidates for a given partition, reassigning points to new clusters as medoids are moved [65]. k -medoids has the advantage of being *unsupervised*: the clustering model does not require training data other than the input. However, the choice of k is problem dependent, and imprudent choices of k may lead to spurious results. Various quality metrics exist

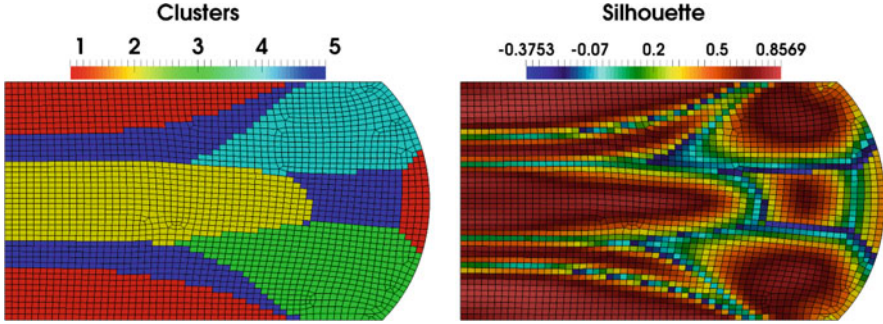


Fig. 15 The clustering of the 4-dimensional strain space with $k = 5$ (left) and the corresponding silhouette (right)

for determining the fitness of a particular choice for k . For a given set of clusters $\mathbf{C} = \{c_k\}$ and points $\mathbf{X} = \{x_i\}$, we define the following quantities:

$$a_i = \frac{\sum_{x_j \in c_{k,i}} d(x_i, x_j)}{n_{k,i}}$$

where $n_{k,i}$ is the number of points in cluster k to which x_i belongs and

$$b_i = \min_{\mathbf{C}} \frac{\sum_{x_j \notin c_{k,i}} d(x_i, x_j)}{n_{k,j}}$$

Thus, a_i represents the average distance of datum x_i to all other points in their parent cluster, and b_i is the minimum average distance of x_i to all other points in any other cluster. The silhouette metric is then defined as follows [67]:

$$s_i = \frac{b_i - a_i}{\max\{a_i, b_i\}}$$

The range of possible silhouette is thus $-1 \leq s_i \leq 1$. For a well-clustered datum, $a_i \ll b_i$ and $s_i \approx 1$, whereas a datum that has been placed in an incorrect cluster will have $s_i \approx -1$. Using k -medoids with a squared l_2 norm, we cluster the 4-dimensional strain space with $k = 5$ and compute the corresponding silhouette, as shown in Fig. 15.

From the silhouette map in Fig. 15, we see that most data are effectively grouped in their parent cluster. The data that are poorly clustered tend to lie along the boundaries of zones, which is reasonable given the continuous nature of the strain field.

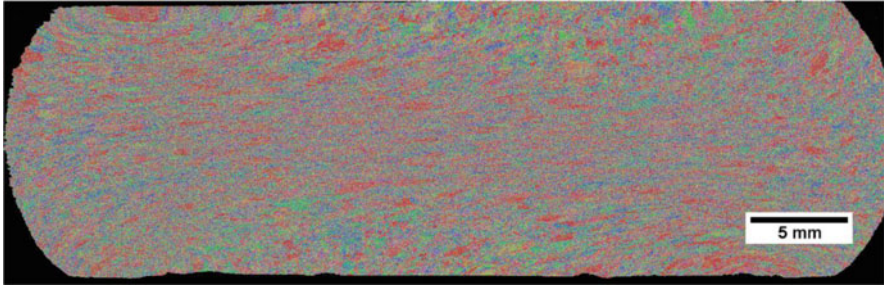


Fig. 16 The stitched cross-sectional EBSD montage of the pancake forging, colored using 001 IPF colors

6.2 Processing Characterization Data

In order to relate the zoned process history to microstructure, the collected characterization data must be processed and relevant statistics extracted. The physical pancake forging corresponding to the DEFORM[®] simulation in the above section was cross-sectioned and imaged using both EBSD and BSE. Due to the size of the specimen, both modalities required montage collections. Individual EBSD tiles were collected with a step size of 15 μm and stitched together using the AnyStitch software [68]. The stitched EBSD montage is shown in Fig. 16. After stitching, alpha particles within the EBSD data were identified by segmenting using a 5° misorientation. These alpha particle orientations were then clustered into five zones using *k*-medoids, and the resulting partition was spatially segmented to identify individual MTRs. Additionally, several statistics about the MTR features were computed, including areas, axis lengths, and morphological orientations.

The BSE imaging produced 979 2048 \times 2048 image tiles with a pixel resolution of 0.5 μm , collected with roughly 20% tile overlap. The total BSE montage was constructed using the image stitching plugin in Fiji [69]. After stitching, the two-phase structure was segmented by applying a simple threshold. An example BSE tile and its segmented counterpart are shown in Fig. 17.

6.3 Registration and Fusion

In order to quantitatively assess the relationship between the process zones and the resulting microstructure, the DEFORM[®] simulation must be registered and fused with the EBSD and BSE characterizations. First, the EBSD and BSE montages are cropped to only the right half of the images, since the DEFORM[®] simulation was only run for one symmetric half of the forging. The DEFORM[®] and BSE montages were then resampled onto image grids with 15 μm pixel spacing, the same as the EBSD. The DEFORM mesh was resampled using nearest neighbor interpolation.

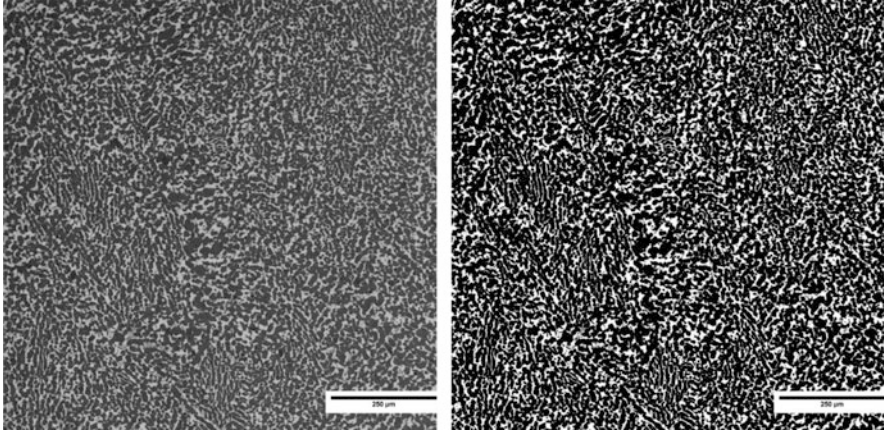


Fig. 17 Original BSE tile (left) and corresponding binary segmented image (right)



Fig. 18 The mask of the identified component in each of the resampled datasets: EBSD (left), BSE (middle), and DEFORM[®] (right)

For the BSE montage, a $15\ \mu\text{m}$ window was passed over the segmented high-resolution montage and the average value of the binary segmentation was computed within this window. This procedure yields an alpha area fraction at each $15\ \mu\text{m}$ pixel.

The resampling procedure conveniently brings all datasets onto the same geometric topology; however, they are still misaligned relative to one another. To register the datasets together, the component was first identified in each of the modalities. Figure 18 shows these component masks.

Since the goal is to determine correspondence between process zones and microstructure, we do not wish to a priori assume any relationships that would bias the registration. Instead, we identify the component geometry in each modality since we expect it to be relatively invariant between each dataset. Thus, the component geometry itself can be used as a registration datum. To obtain points for registration, the exterior of the masks shown in Fig. 18 are regularly sampled. These sampled points are shown in Fig. 19.

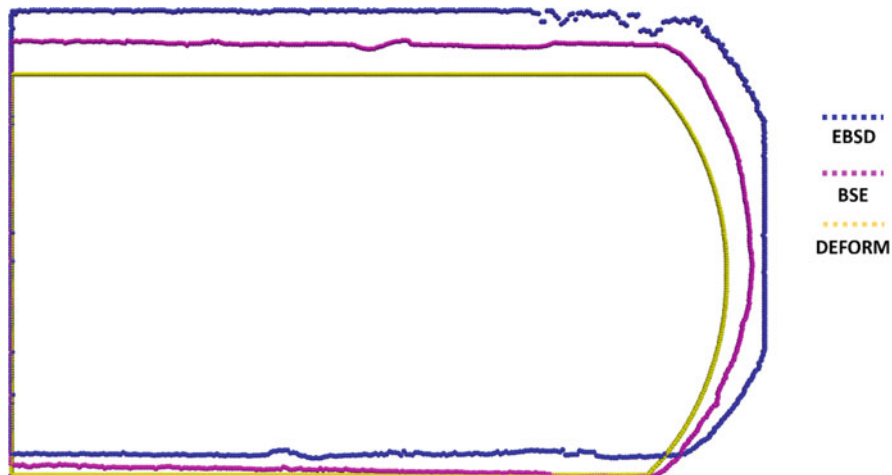


Fig. 19 Sampled exterior points, used for registration, from each of the modalities

We compute the transform that best brings sampled points into alignment using a least-squares approach that is robust to noise [70]. The goal is to estimate the rotation R , translation t , and scaling s that best minimize the squared error between two sets of points, $\mathbf{X} \in \mathbb{R}^d$ and $\mathbf{Y} \in \mathbb{R}^d$:

$$\varepsilon^2(R, t, s) = \frac{1}{2} \sum_{i=1}^n \|y_i - (sRx_i + t)\|^2$$

The above minimization is possible for solutions in R , t , s from the following equations:

$$R = USV^T$$

$$t = \mu_y - sR\mu_x$$

$$s = \frac{1}{\sigma_x^2} tr(DS)$$

where UDV^T is the singular value decomposition of \mathbf{XY}^T and

$$S = \begin{cases} I, & \det(\mathbf{XY}^T) \geq 0 \\ \text{diag}(1, 1, \dots, 1, -1), & \det(\mathbf{XY}^T) < 0 \end{cases}$$

μ_x and μ_y are the average positions of \mathbf{X} and \mathbf{Y} , and σ_x^2 is the variance of \mathbf{X} . Using the above approach, the EBSD registration points were first transformed to

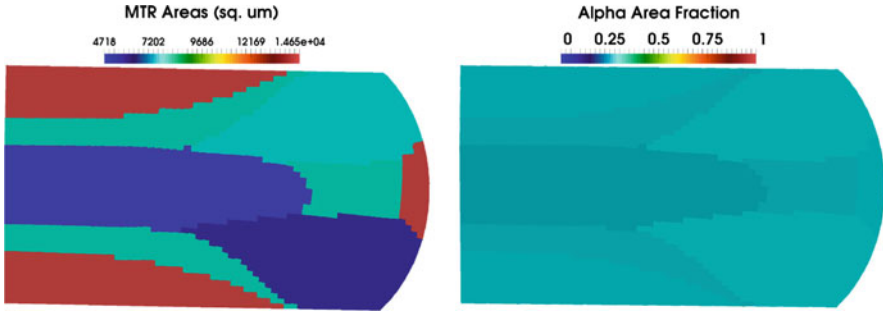


Fig. 20 Zoned regions of process history colored by MTR areas (left) and alpha area fraction (right)

the DEFORM[®] reference frame, with both datasets followed by being transformed to the BSE reference frame. After applying the transformations on the resampled image geometries, the resulting aligned images were fused on the same grid using nearest neighbor interpolation. On this new resampled geometry, the strain tensors from the DEFORM[®] simulation were rezoned using $k = 5$. After performing this fusion, it is possible to assess microstructure characteristics per zone. Figure 19 shows the process zones colored by different aspects of the microstructure. In Fig. 20, the average MTR area, as measured from the EBSD, and alpha area fraction, determined from the segmentation of the BSE image montage, are shown for each zone. Note that since the data have been fused onto the same geometry, this approach presents a direct comparison between the zoned process variables and the resulting microstructure. The average MTRs are much larger in the zones that correspond to regions of large strain, as compared to the zones central to the forging. The alpha area fraction, however, does not vary substantially with the strain zones.

This visualization demonstrates the power of a flexible ICME tool: the ability to simultaneously represent various geometries (i.e., images, meshes, and points), data shapes (i.e., tensorial strains, vector orientations, and scalar image intensities), and complex hierarchy (i.e., zoned process variables, identified MTRs, and segmented BSE images) allows for novel analyses to be conducted. DREAM.3D, by leveraging SIMPL, is able to effectively manage these disparate data streams and orchestrate their fusion to produce actionable information. Thanks to the reusability of filters via plugins, characterization steps such as computing sizes of features or finding average values within features did not require reimplementing, freeing the development time to be spent on devising a robust zoning and registration procedure. Additionally, since SIMPL archives pipeline information along with the raw data, researchers are able to confidently store data and reproduce workflows as needed. Indeed, the authors greatly benefited from this functionality in constructing figures for this use case.

7 Summary

We have presented an overview of the ICME software applications available to a modern materials researcher, with a focus on simulation and analytics tools that generate and process field data. In order to properly handle the myriad kinds of information these tools produce, we have sketched an outline of the requirements that an ICME workflow manager should aspire to address. We define requirements on such a tool's data structure, modularity, data access, and workflow capabilities. As an example of one tool that satisfies these requirements, we showcased SIMPL, the Spatial Information Management Protocol Library, along with its most prominent user, the Digital Representation Environment for the Analysis of Microstructure in 3D, or DREAM.3D. To demonstrate how DREAM.3D can leverage SIMPL's features, a case study was outlined that shows how to address a common problem in ICME workflows: quantitatively relating process simulation to measure microstructure. This worked example showed how process variables from a forging simulation of a Ti-6242Si cylinder could be directly coupled with resulting microstructure characterized using EBSD and BSE imaging.

As a community, materials research and development must still make progress on developing easily shareable toolsets for analysis, driven by the continuing adoption of ICME techniques and data-driven methods. Other fields, such as bioinformatics, have made this transition successfully; the materials community should heed the lessons learned from these other sister fields and seek to grow tools that foster development on those problem spaces unique to materials. Additionally, teaching the next generation of materials researchers how to reason through materials problems with an ICME lens is paramount. Developing robust, standardized, and open tools for the growing community ensures that the goals of ICME are achievable.

Acknowledgments The authors would like to acknowledge Mike Jackson, for his vision and programming expertise in enabling the implementation of the SIMPL architecture; Dennis Dimiduk, for his consistent support and fruitful discussions; Adam Pilchak, for motivating the demonstrated ICME use case and providing the data and material; Mike Uchic, for providing characterization support and contribution to the vision of SIMPL; and Chris Woodward, for his unyielding support in the early stages of designing DREAM.3D.

References

1. National Research Council, *Integrated Computational Materials Engineering: A Transformational Discipline for Improved Competitiveness and National Security* (The National Academies Press, Washington, DC, 2008). <https://doi.org/10.17226/12199>.
2. B. Puchala, G. Tarcea, E.A. Marquis, M. Hedstrom, H.V. Jagadish, J.E. Allison, The materials commons: a collaborative platform and information repository for the global materials community. *JOM* **68**(8), 2035–2044 (2016). <https://doi.org/10.1007/s11837-016-1998-7>
3. B. Blaiszik, K. Chard, J. Pruyne, R. Ananthakrishnan, S. Tuecke, I. Foster, The materials data facility: data services to advance materials science research. *JOM* **68**(8), 2045–2052 (2016). <https://doi.org/10.1007/s11837-016-2001-3>

4. A. Jain, S.P. Ong, G. Hautier, W. Chen, W.D. Richards, S. Dacek, S. Cholia, D. Gunter, D. Skinner, G. Ceder, K.A. Persson, Commentary: the materials project: a materials genome approach to accelerating materials innovation. *APL Mater* **1**(1) (2013). <https://doi.org/10.1063/1.4812323>
5. J. Ahrens, B. Gevecki, C. Law, ParaView: an end-user tool for large data visualization, in *Visualization Handbook*, (Elsevier, Amsterdam, 2005)
6. Applications. (Scientific Forming Technologies Corporation) [Online]. Available: <https://www.deform.com/applications/>. Accessed Mar 2019
7. Casting Applications. (ESI Group) [Online]. Available: <https://www.esi-group.com/software-solutions/virtual-manufacturing/casting/applications>. Accessed Mar 2019
8. Abaqus Unified FEA. (Dassault Systemes) [Online]. Available: <https://www.3ds.com/products-services/simulia/products/abaqus/>. Accessed Mar 2019
9. ANSYS. (ANSYS) [Online]. Available: <https://www.ansys.com/>. Accessed Mar 2019
10. Albany. (Sandia National Laboratories, Center for Computing Research) [Online]. Available: <https://cfwebprod.sandia.gov/cfdocs/CompResearch/templates/insert/project.cfm?proj=28>. Accessed Mar 2019
11. A.G. Salinger, R.A. Bartlett, A.M. Bradley, Q. Chen, I.P. Demeshko, X. Gao, G.A. Hansen, A. Mota, R.P. Muller, E. Nielsen, J.T. Ostien, R.P. Pawlowski, M. Perego, E.T. Phipps, W. Sun, I.K. Tezaur, Albany: using component-based design to develop a flexible, generic multiphysics analysis code. *Int J Multiscale Comput Eng* **14**(4), 415–438 (2016). <https://doi.org/10.1615/IntJMultCompEng.2016017040>
12. MOOSE: Multiphysics Object Oriented Simulation Environment. (Idaho National Laboratory) [Online]. Available: <https://mooseframework.org/>. Accessed Mar 2019
13. D. Gaston, C. Newman, G. Hansen, D. Lebrun-Grandie, MOOSE: A parallel computational framework for coupled systems of nonlinear equations. *Nucl. Eng. Des.* **239**(10), 1768–1778 (2009)
14. H. Moulinec, P. Suquet, A numerical method for computing the overall response of nonlinear composites with complex microstructure. *Comput. Methods Appl. Mech. Eng.* **157**(1–2), 69–94 (1998). [https://doi.org/10.1016/S0045-7825\(97\)00218-1](https://doi.org/10.1016/S0045-7825(97)00218-1)
15. J.C. Michel, H. Moulinec, P. Suquet, A computational scheme for linear and non-linear composites with arbitrary phase contrast. *Numer Methods Eng* **52**(1–2), 139–160 (2001). <https://doi.org/10.1002/nme.275>
16. S.P. Donegan, A.D. Rollett, Simulation of residual stress and elastic energy density in thermal barrier coatings using fast Fourier transforms. *Acta Mater.* **96**, 212–228 (2015). <https://doi.org/10.1016/j.actamat.2015.06.019>
17. R. A. Lebensohn, N-site modeling of a 3D viscoplastic polycrystal using Fast Fourier Transform. *Acta Mater.* **49**(14), 2723–2737 (2001). [https://doi.org/10.1016/S1359-6454\(01\)00172-0](https://doi.org/10.1016/S1359-6454(01)00172-0)
18. P. Eisenlohr, M. Diehl, R.A. Lebensohn, F. Roters, A spectral method solution to crystal elasto-viscoplasticity at finite strains. *Int. J. Plast.* **46**, 37–53 (2013). <https://doi.org/10.1016/j.ijplas.2012.09.012>
19. F. Roters, M. Diehl, P. Shanthraj, P. Eisenlohr, C. Reuber, S.L. Wong, T. Maiti, A. Ebrahimi, T. Hochrainer, H.-O. Fabritius, S. Nikolov, M. Friak, N. Fujita, N. Grilli, K.G.F. Janssens, N. Jia, P.J.J. Kok, D. Ma, F. Meiner, E. Werner, M. Stricker, D. Weygand, D. Raabe, DAMASK – The Dusseldorf advanced material simulation kit for modeling multi-physics crystal plasticity, thermal, and damage phenomena from the single crystal up to the component scale. *Comput. Mater. Sci.* **158**, 420–478 (2019)
20. What is VASP? (VASP Software GmbH) [Online]. Available: <https://www.vasp.at/index.php/about-vasp/59-about-vasp>. Accessed Mar 2019
21. LAMMPS Molecular Dynamics Simulator. (Sandia National Laboratories) [Online]. Available: <https://lammps.sandia.gov/>. Accessed Mar 2019
22. ParaDiS. (Lawrence Livermore National Laboratory) [Online]. Available: <http://paradis.stanford.edu/site/home>. Accessed Mar 2019

23. Avizo for Materials Science. (ThermoFisher Scientific) [Online]. Available: <https://www.thermofisher.com/us/en/home/industrial/electron-microscopy/electron-microscopy-instruments-workflow-solutions/3d-visualization-analysis-software/avizo-materials-science.html>. Accessed Mar 2019
24. GeoDict – The Digital Material Laboratory. (Math2Market GmbH) [Online]. Available: <https://www.math2market.com/Solutions/aboutGD.php>. Accessed Mar 2019
25. ESPRIT QUBE – Advanced 3D analysis of EBSD/EDS Data. (Bruker Corporation) [Online]. Available: <https://www.bruker.com/products/x-ray-diffraction-and-elemental-analysis/eds-wds-ebds-sem-micro-xrf-and-sem-micro-ct/quantax-ebds/esprit-qube.html>. Accessed Mar 2019
26. D. Wheeler, D. Brough, T. Fast, S. Kalidindi, A. Reid, *PyMKS: Materials Knowledge System in Python* (2014). <https://doi.org/10.6084/m9.figshare.1015761>
27. S.R. Niezgodna, D.T. Fullwood, S.R. Kalidindi, Delineation of the space of 2-point correlations in a composite material system. *Acta Mater.* **56**(18), 5285–5292 (2008). <https://doi.org/10.1016/j.actamat.2008.07.005>
28. T. Fast, S.R. Kalidindi, Formulation and calibration of higher-order elastic localization relationships using the MKS approach. *Acta Mater.* **59**(11), 4595–4605 (2011)
29. Magpie. (Wolverton Research Group) [Online]. Available: <https://bitbucket.org/wolverton/magpie>. Accessed Mar 2019
30. MTEX Toolbox [Online]. Available: <http://mteX-toolbox.github.io/>. Accessed Mar 2019
31. F. Bachmann, R. Hielscher, H. Schaeben, Texture analysis with MTEX – Free and open source software toolbox. *Solid State Phenom.* **160**, 63–68 (2010)
32. J.M. Sosa, D.E. Huber, B. Welk, H.L. Fraser, Development and application of MIPAR: A novel software package for two- and three-dimensional microstructural characterization. *Integr Mater Manuf Innov* **3**(10), 123 (2014). <https://doi.org/10.1186/2193-9772-3-10>
33. M.A. Groeber, M.A. Jackson, DREAM.3D: A digital representation environment for the analysis of microstructure in 3D. *Integr Mater Manuf Innov* **3**(5), 56 (2014). <https://doi.org/10.1186/2193-9772-3-5>
34. ImageJ: Image Processing and Analysis in Java. (National Institutes of Health) [Online]. Available: <https://imagej.nih.gov/ij/index.html>. Accessed Mar 2019
35. C.A. Schneider, W.S. Rasband, K.W. Eliceiri, NIH image to ImageJ: 25 years of image analysis. *Nat. Methods* **9**, 671–675 (2012). <https://doi.org/10.1038/nmeth.2089>
36. Fiji [Online]. Available: <https://fiji.sc/>. Accessed Mar 2019
37. J. Schindelin, I. Arganda-Carreras, E. Frise, V. Kaynig, M. Longair, T. Pietzsch, S. Preibisch, C. Rueden, S. Saalfeld, B. Schmid, J.-Y. Tinevez, D.J. White, V. Hartenstein, K. Eliceiri, P. Tomancak, A. Cardona, Fiji: An open-source platform for biological image analysis. *Nat. Methods* **9**, 676–682 (2012). <https://doi.org/10.1038/nmeth.2019>
38. ITK. (Kitware, Inc.) [Online]. Available: <https://itk.org/>. Accessed Mar 2019
39. T.S. Yoo, M.J. Ackerman, W.E. Lorensen, W. Schroeder, V. Chalana, S. Aylward, D. Metaxas, R. Whitaker, Engineering and algorithm design for an image processing Api: A technical report on ITK – The insight toolkit. *Stud. Health Technol. Inform.*, 586–592 (2002). <https://doi.org/10.3233/978-1-60750-929-5-586>.
40. 3D Slicer [Online]. Available: <https://www.slicer.org/>. Accessed Mar 2019
41. R. Kikinis, S.D. Pieper, K.G. Vosburgh, 3D slicer: A platform for subset-specific image analysis, visualization, and clinical support, in *Intraoperative Imaging and Image-Guided Therapy*, (2014), pp. 277–289. https://doi.org/10.1007/978-1-4614-7657-3_19
42. VTK (Kitware, Inc.) [Online]. Available: <https://vtk.org/>. Accessed Mar 2019
43. W. Schroeder, K. Martin, B. Lorensen, *The Visualization Toolkit*, 4th edn. (Kitware, 2006)
44. SCIRun. (The NIH/NIGMS Center for Integrative Biomedical Computing) [Online]. Available: <http://www.sci.utah.edu/cibc-software/scirun.html>. Accessed Mar 2019
45. S.G. Parker, C.R. Johnson, SCIRun: a scientific programming environment for computational steering, in *Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*, San Diego, 1995. <https://doi.org/10.1109/SUPERC.1995.241689>

46. Orange – Data Mining Fruitful & Fun. (University of Ljubljana) [Online]. Available: <https://orange.biolab.si/>. Accessed Mar 2019
47. J. Demsar, T. Curk, A. Erjavec, C. Gorup, T. Hocevar, M. Milutinovic, M. Mozina, M. Polajnar, M. Toplak, A. Staric, M. Stajdohar, L. Umek, L. Zagar, J. Zbontar, M. Zitnik, B. Zupan, Orange: data mining toolbox in Python. *J. Mach. Learn. Res.* **14**, 2349–2353 (2013)
48. Weka 3: Data Mining Software in Java. (University of Waikato) [Online]. Available: <https://www.cs.waikato.ac.nz/ml/weka/index.html>. Accessed Mar 2019
49. E. Frank, M.A. Hall, I.H. Witten, The WEKA workbench, in *Data Mining: Practical Machine Learning Tools and Techniques*, (Morgan Kaufmann, 2016)
50. SIMPL. (BlueQuartz Software, LLC) [Online]. Available: <https://github.com/BlueQuartzSoftware/SIMPL>. Accessed Mar 2019
51. DREAM3D. (BlueQuartz Software, LLC) [Online]. Available: <https://github.com/BlueQuartzSoftware/DREAM3D>. Accessed March 2019
52. SIMPLView. (BlueQuartz Software, LLC) [Online]. Available: <https://github.com/BlueQuartzSoftware/SIMPLView>. Accessed Mar 2019
53. QT | Cross-platform software development for embedded & desktop [Online]. Available: <https://www.qt.io/>. Accessed Mar 2019
54. Hierarchical Data Format, version 5. (The HDF Group, 1997–2019) [Online]. Available: <http://www.hdfgroup.org/HDF5/>
55. G. Guennebaud, B. Jacob, Eigen v3. (2010) [Online]. Available: <http://eigen.tuxfamily.org>. Accessed March 2019
56. Intel Threading Building Blocks. (Intel Corporation) [Online]. Available: <https://www.threadingbuildingblocks.org/>. Accessed Mar 2019
57. W. Jakob, J. Rhineland, D. Moldovan, pybind11 – Seamless operability between C++11 and Python. (2019). [Online]. Available: <https://github.com/pybind/pybind11>. Accessed Mar 2019
58. CMake. (Kitware, Inc.) [Online]. Available: <https://cmake.org/>. Accessed Mar 2019
59. Xdmf [Online]. Available: http://xdmf.org/index.php/Main_Page. Accessed Mar 2019
60. A.L. Pilchak, J. Shank, J.C. Tucker, S. Srivatsa, P.N. Fagin, S.L. Semiatin, A dataset for the development, verification, and validation of microstructure-sensitive process models for near-alpha titanium alloys. *Integr Mater Manuf Innov* **5**(14), 259 (2016). <https://doi.org/10.1186/s40192-016-0056-1>
61. A.P. Woodfield, M.D. Gorman, R.R. Corderman, J.A. Sutliff, B. Yamrom, Effect of Microstructure on Dwell Fatigue Behavior of Ti-6242, in *Titanium '95: Science and Technology*, (Birmingham, 1996)
62. A.L. Pilchak, A. Huston, W.J. Porter, D.J. Buchanan, R. John, Growth of small and long fatigue cracks in Ti-6Al-4V subjected to cyclic and dwell fatigue, in *Proceedings of the 13th World Conference on Titanium*, Warrendale, 2016.
63. A.L. Pilchak, A simple model to account for the rolw of microtexture on fatigue and dwell fatigue lifetimes of titanium alloys. *Scr. Mater.* **74**, 68–71 (2014). <https://doi.org/10.1016/j.scriptamat.2013.10.024>
64. A.K. Jain, M.N. Murty, P.J. Flynn, Data clustering: A review. *ACM Comput. Surv.* **31**(3), 265–323 (1999). <https://doi.org/10.1145/331499.331504>
65. L. Kaufman, P.J. Rousseeuw, Clustering by means of medoids, in *Proceedings of Statistical Data Analysis Based on the L1 Norm*, Neuchatel, 1987
66. A.K. Jain, Data clustering: 50 years beyond K-means. *Pattern Recogn. Lett.* **31**(8), 651–666 (2010). <https://doi.org/10.1016/j.patrec.2009.09.011>
67. P.J. Rousseeuw, Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.* **20**, 53–65 (1987). [https://doi.org/10.1016/0377-0427\(87\)90125-7](https://doi.org/10.1016/0377-0427(87)90125-7)
68. A.L. Pilchak, A.R. Shiveley, J.S. Tiley, D.L. Ballard, AnyStitch: A tool for combining electron backscatter diffraction data sets. *J. Microsc.* **244**(1), 38–44 (2011). <https://doi.org/10.1111/j.1365-2818.2011.03496.x>

69. S. Preibisch, S. Saalfeld, P. Tomancak, Globally optimal stitching of tiled 3D microscopic image acquisitions. *Bioinformatics* **25**(11), 1463–1465 (2009). <https://doi.org/10.1093/bioinformatics/btp184>
70. S. Umeyama, Least-squares estimation of transformation parameters between two point patterns. *IEEE Trans Pattern Anal Mach Intell* **13**, 376–380 (1991). <https://doi.org/10.1109/34.88573>