IDA-01 (International Symposium on Intelligent Data Analysis), Lisbon; September 2001

Feature Characterization in Scientific Datasets

Elizabeth Bradley,[1] Nancy Collins,[1]⋆ and W. Philip Kegelmeyer[2]

[1] University of Colorado, Department of Computer Science, Boulder, CO 80309-0430
lizb,collinn@cs.colorado.edu,
[2] Sandia National Laboratories, P.O. Box 969, MS 9951, Livermore, CA, 94551-0969
wpk@ca.sandia.gov

Abstract. We describe a preliminary implementation of a data analysis tool that can characterize features in large scientific datasets. There are two primary challenges in making such a tool both general and practical: first, the definition of an interesting feature changes from domain to domain; second, scientific data varies greatly in format and structure. Our solution uses a hierarchical feature ontology that contains a base layer of objects that violate basic continuity and smoothness assumptions, and layers of higher-order objects that violate the physical laws of specific domains. Our implementation exploits the metadata facilities of the SAF data access libraries in order to combine basic mathematics subroutines smoothly and handle data format translation problems automatically. We demonstrate the results on real-world data from deployed simulators.

1 Introduction

Currently, the rate at which simulation data can be generated far outstrips the rate at which scientists can inspect and analyze it. 3D visualization techniques provide a partial solution to this problem, allowing an expert to scan large data sets, identifying and classifying important features and zeroing in on areas that require a closer look. Proficiency in this type of analysis, however, requires significant training in a variety of disciplines. An analyst must be familiar with domain science, numerical simulation, visualization methods, data formats, and the details of how to move data across heterogeneous computation and memory networks, among other things. At the same time, the sheer volume of these data sets makes this analysis task not only arduous, but also highly repetitive. A logical next step is to automate the feature recognition and characterization process so scientists can spend their time analyzing the science behind promising or unusual regions in their data, rather than wading through the mechanistic details of the data analysis. This paper is a preliminary report on a tool that does so.

General definitions of features are remarkably hard to phrase; most of those in the literature fall back upon ill-defined words like "unusual" or "interesting" or "coherent." Features are often far easier to *recognize* than to *describe*, and they are also highly domain-dependent. The structures on which an expert analyst chooses to focus — as well as the manner in which he or she reasons about them — necessarily

⋆ Supported by the DOE ASCI program through a Level 3 grant from Sandia National Laboratories, and a Packard Fellowship in Science and Engineering.

depend upon the physics that is involved, as well as upon the nature of the investigation. Meteorologists and oceanographers are interested in storms and gyres, while astrophysicists search for galaxies and pulsars, and molecular biologists classify parts of molecules as alpha-helices and beta-sheets. Data types vary — pressure, temperature, velocity, vorticity, etc. — and a critical part of the analyst's expert knowledge is knowing which features appear in what data fields.

In this paper, we describe a general-purpose feature characterization tool and validate it with several specific instances of problems in one particular field: finite element analysis data from computer simulations of solid mechanics problems. One of our goals is to produce a practical, useful tool, so we work with data from deployed simulators, in a real-world format: ASCI's SAF, a *lingua franca* used by several of the US national labs to read and write data files for large simulation projects. This choice raised some interoperability issues that are interesting from an IDA standpoint, as discussed in section 2 below. The SAF interface provides access to a geometric description of a computational mesh, including the spatial positions of the mesh points (generally xy or xyz) and the type of connectivity, such as triangles or quads, plus information about the physics variables, such as temperature or velocity. Given such a snapshot, our goal is to characterize the features therein and generate a meaningful report. We began by working closely with domain scientists to identify a simple ontology[1] of distinctive coherent structures that help them understand and evaluate the dynamics of the problem at hand. In finite-element applications, as in many others, there are two kinds of features that are of particular interest to us:

– those that violate the continuity and smoothness assumptions that are inherent in both the laws of physics and of numerical simulation: spikes, cracks, tears, wrinkles, etc. — either in the mesh geometry or in the physics variables.
– those that violate higher-level physical laws, such as the requirement for normal forces to be equal and opposite when two surfaces meet (such violations are referred to as "contact problems").

Note that we are assuming that expert users *can* describe these features mathematically; many of the alternate approaches to automated feature detection that are described in section 5 do not make this assumption. The knowledge engineering process is described in section 3.1 and the algorithms that we use to encapsulate the resulting characterizations, which rely on fairly basic mathematics, are described in section 3.2. We have tested these algorithms on roughly a half-dozen data sets; the results are summarized in section 4.

2 Data Formats and Issues

DMF[15] is a joint interoperability project involving several US national labs. Its goal is to coordinate the many heterogeneous data handling libraries and analysis tools that are used by these organizations, and to produce standards and libraries that will allow others to exploit the results. This project is motivated by the need to perform

[1] Formally, an ontology seeks to distill the most basic concepts of a system down into a set of well defined nouns and verbs (objects and operators) that support effective reasoning about the system.

simulations at the system level, which requires formerly independent programs from various disciplines to exchange data smoothly. The attendant interoperability problems are exacerbated by the growing sophistication and complexity of these tools, which make it more difficult to adapt them to new data formats, particularly if the new format is richer than the old. The specific DMF data interface that we use, called SAF[11], exploits *metadata* — that is, data about the data — to solve these problems. Used properly, metadata can make a dataset *self-describing*. SAF, for example, captures not only the data values, but also the geometry and topology of the computational grid, the interpolation method used inside each computational element, and the relationships between various subsets of the data, among other things. Its interface routines can translate between different data formats automatically, which confers tremendous leverage upon tools that use it. They need only handle one type of data and specify it in their metadata; SAF will perform any necessary translation. In our project, this is important in both input and output. Not only must we handle different kinds of data, but we must also structure and format the results in appropriate ways. As discussed at length in the scientific visualization literature, different users need and want different data types and formats, so reporting facilities must be flexible. Moreover, the consumer of the data might not be a person, but rather another computer tool in a longer processing pipeline. For example, output generated by the characterization routines developed in this paper might be turned into a simple ascii report or formatted into an html page for viewing with a browser by a human expert, and simultaneously fed to a visualization tool for automatic dataset selection and viewpoint positioning. For all of these reasons, it is critical that data be stored in a format that supports the generation and use of metadata, and SAF is designed for exactly this purpose.

Metadata is a much broader research area, and SAF was not the first data model to incorporate and use it. Previous efforts included PDBlib, FITS, HDF, netCDF, VisAD, and DX, among others[4,16,5,8,9] — data formats that enabled analysis tools to reason about metadata in order to handle the regular data in an appropriate manner. While metadata facilities are of obvious utility to the IDA process, they are also somewhat of a Pandora's Box; as simulation tools increase in complexity, effective analysis of their results will require a corresponding increase in the structure, amount, and complexity of the metadata. This raises a host of hard and interesting ontology problems, as well as the predictable memory and speed issues, which are beyond the scope of the current paper.

The SAF libraries are currently in alpha-test release[2]. Because of this, few existing simulation, analysis, and visualization tools understand SAF's native interface. Our early development prototypes, for instance, used the SAF library directly for data access, but had to convert to the OpenDX file format for visualization: the very kind of translation that SAF is intended to obviate. Because visualization is so critical to data analysis, there has been some recent progress in adapting existing visualization tools to parse SAF input. In the first stages of our project, however, such tools did not exist, so we used OpenDX for visualization. We recently began converting to a SAF-aware visualization tool called EnSight[1], but this has not been without

[2] We are a designated alpha-test group, and a secondary goal of this project is to provide feedback to the DMF developers, based on our experiences in designing an intelligent data analysis tool around this format.

problems. Data interface libraries are subject to various chicken-and-egg growing pains. The tools need not understand a format until an interesting corpus of data exists in that format; scientists are understandably unwilling to produce data in a format for which no analysis tools exist. Intelligent data analysis tools that take care of low-level interoperability details can remove many barriers from this process.

3 Intelligent Analysis of Simulation Data

3.1 Knowledge Engineering

In order to automate the feature characterization process, we first needed to understand how human experts perform the analysis. We spent several days with various project analysts at Sandia National Laboratories, observing as they used existing tools on different kinds of data. We focused in on what they found important, how they identified and described those features, how they reasoned about which data fields to examine for a given stage of the process, and how the entire process changed if they were trying to prove or disprove a particular hypothesis. Most of the features of interest to these experts, we found, are clued from local geometry of the simulation mesh; inverted elements with non-positive volume, spikes, wrinkles, dimples, and so on. A smaller set of features of interest are extrema in the physics variables: hot spots and the like. We used this information to specify a simple ontology: that is, a set of canonical features (spikes, tears, cracks, etc.), together with mathematical descriptions of each — the statistical, geometric, and topological properties that define them. We also studied how the experts wrote up, reported, and used their results.

The Sandia analysts view the mechanical modeling process in two stages. The first is model debugging, wherein they ensure that the initial grid is sound, that the coupling is specified correctly between various parts of the model, and that the modeling code itself is operating correctly. The second is the actual simulation, where they examine the data for interesting physical effects: vibrational modes, areas that exceed the accepted stress tolerances, etc. We found that features play important roles in both phases, and that the sets of features actually overlapped. A spike in the results, for instance, can indicate either a numerical failure or a real (and interesting) physical effect. In some cases, reasoning about features let analysts identify model errors that were undetectable by traditional numerical tests like overflow, divide-by-zero, etc. One scientist described a simulation of an automobile engine compartment, including the front bumper. Due to a numerically innocuous error, one of the grid points moved to a location well beyond the back end of the entire car. This obviously non-physical situation — which was immediately visible to the analyst as a feature — flagged the model as faulty, even though no numerical fault occurred.

Note that features can involve the mesh coordinates, the physics variables, and sometimes both. Vertical relief, for instance, is a property of surface geometry, not the value of the physics variables upon that surface. Conversely, calculation of the highest temperature on a surface depends solely on the physics variables. Often, analysts are interested in features that involve both: say, the temperature or wind speed at the highest point on the landscape, or the position of the hottest point. Often, too, their underlying assumptions about geometry and about physics are similar, which can lead to some terminology confusion. A spike in temperature and a spike on the surface are similar in that both violate smoothness assumptions, but the mathematics of their characterization is quite different. This is actually a symptom of a deeper and more interesting property of features: like data analysis itself, they are hierarchical. All surfaces, whether numerical or physical, are generally continuous and smooth, so tears and spikes are likely to be considered to be features in *any* domain. If one knows more about the physics of the problem, other features become interesting as well. In contact problems, for instance — where two surfaces touch one another — the normal forces at the intersection of the two surfaces should be equal and opposite and surfaces should certainly not interpenetrate. Violations of these physical realities are interesting features. To capture these layers of meaning, our feature ontology is hierarchical. It contains a baseline set of features that rest on assumptions that are true of *all* physical systems, together with layers of higher-order features that are specific to individual domains (and sub-domains and so on). Currently, we have finished implementing two such layers: the baseline one mentioned above (spikes *et al.*) and a contact-problem one, which defines deviation from equal-and-opposite as a feature. Both are demonstrated in section 4.

3.2 Algorithms

Given the feature ontology described in the previous section, our next task was to develop algorithms that could find instances of those features in DMF data snapshots and generate meaningful reports about their characteristics. In order to make our work easily extensible, we structured the overall design so as to provide a general-purpose framework into which characterization routines specific to the features of a given domain can be easily installed. In particular, we provide several basic building-block tools that compute important statistical, geometrical, and topological information — about the mesh itself and about the values of the physics variables that are associated with each point in the mesh. Their results are stored using the SAF library format, complete with metadata that allow them to be combined in different ways to assess a wide variety of features in a range of domains. Often, there is more than one way to find a single feature; a surface spike, for instance, can be characterized using statistics (a point that is several σ away from the mean) or geometry (a point where the slope changes rapidly).

Our current set of basic building blocks is fairly straightforward:

- `normals()`, which takes a DMF dataset and computes the unit-length normal vector to each mesh element.
- `topological-neighbors()`, which takes a DMF dataset and an individual mesh element m and returns a list of mesh elements that share an edge or a vertex with m.
- `geometric-neighbors()`, which takes a DMF dataset, an individual mesh element m and a radius r, and returns a list of mesh elements whose vertices fall entirely within r of the centroid of m.
- `statistics()`, which takes a DMF dataset and a specification of one variable (one of the mesh coordinates or physics variables) and computes the maximum, minimum, mean, and standard deviation of its values.

– `displacements()`, which takes a DMF dataset, finds all neighboring[3] pairs of vertices, measures the *xyz* distance between them, and reports the maximum, minimum, mean, and standard deviation of those distances

In addition, we provide various vector calculus facilities (e.g., dot products) and distance metric routines.

As an example of how these tools work, consider Fig. 1. The vectors computed

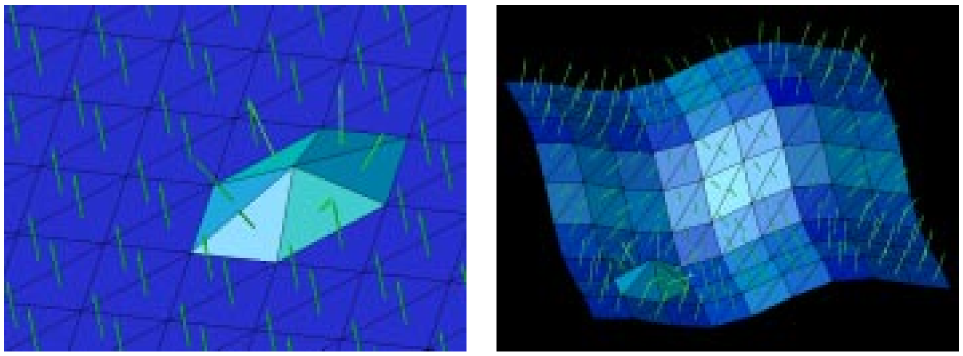

Fig. 1. 3D surface mesh examples, showing the vectors computed by the `normals()` function.

by `normals()` are shown emanating from the center of each mesh face. In a regular mesh, finding topological neighbors could be trivial. SAF, however, is designed to be able to represent irregular and adaptive meshes as well, so the current version of SAF only provides neighbor information implicitly. For this reason, we preprocess the DMF data at the beginning of the characterization run and place it in a data structure that makes the topological information explicit. Our current design maintains a single list of vertices, including *xyz* position and the values of any associated physics variables. Three other lists point into this vertex list — a face list, an edge list, and a normal list — making it easy to look for shared edges or vertices and deduce neighbor relationships. In the examples in Fig. 1, each triangle has three "face neighbors" and at least three other "vertex neighbors," all of which are returned by `topological-neighbors`. The `geometrical-neighbors` function is a bit more complicated; it calls `topological-neighbors`, measures the Euclidean distances between the vertices of the resulting triangles and the centroid of the original element, discards any element whose vertices do not all fall within the specified distance, and iteratively expands on the others. The `statistics()` and `displacements()` routines use simple traditional methods. The left-hand surface in Fig. 1, for instance, is completely flat, with the exception of the bump in the foreground, and the `statistics()` results reflect the appropriate mean height of the surface and a very small standard deviation. The right-hand surface fluctuates somewhat, so the standard deviation is larger. In both cases, the `displacements()` results would likely be uninformative because the edge lengths of the elements are fairly uniform.

[3] *Topologically* neighboring

There are a variety of ways, both obvious and subtle, to improve on the toolset described above. We are currently focusing on methods from computational geometry[12] (e.g., Delaunay triangulation) and computational topology, such as the α-shape[7], and we have developed the theoretical framework and some preliminary implementations of these ideas[13, 14]. Since features are often easier to *recognize* than to *describe*, we are also exploring the use of machine learning techniques to discover good values for the heuristic parameters that are embedded in these computational geometry and topology algorithms.

4 Results and Evaluation

We have done preliminary evaluations of the algorithms described in the previous section using half a dozen datasets. For space reasons, only two of those datasets are discussed here; please see our website[4] for further results, as well as color versions of all images in this paper. The first dataset, termed `irregular-with-spike`, is shown in Fig. 2. It consists simply of an irregular surface mesh; no physics variables are

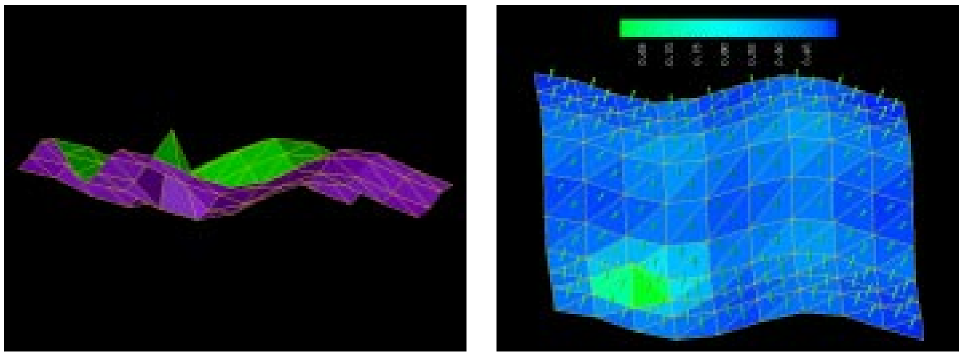

Fig. 2. A 3D surface mesh dataset that contains a spike. By dotting the normals of neighboring faces and comparing the result to the local average of the surface normals, we can detect anomalies in the slope of the surface. Results of this algorithm are used to shade the mesh elements in the right-hand image. Lighter elements are members of a *surface spike* feature.

involved. Such a dataset might, for instance, represent the surface of a mechanical part. As rendered, this surface contains an obvious feature — a vertical spike — to which the eye is immediately drawn. Such a feature may be meaningful for many domain-dependent and -independent reasons: as an indicator of numerical problems or anomalies in the physics models, or perhaps a real (and surprising) physical effect. All of these reasons essentially boil down to an assumption of continuity and smoothness in the surface.

The task of our spike detection algorithm is to find places where that smoothness assumption is violated. To accomplish this, we begin by using the `normals()` and

[4] `http://www.cs.colorado.edu/~lizb/features.html`

`topological-neighbors()` functions to find the normal vector to each mesh face and dot it with its neighbors' normals. While this does allow us to detect sudden variations in slope, it is inadequate for evaluating the results because anomalies are always *relative to a whole*. A 1 cm bump in a surface whose mean bumpiness is 5cm, for instance, may not be interesting; a 1 cm bump in the Hubble Space Telecope mirror, however, is most definitely an issue. Moreover, it is impossible to characterize a feature — e.g., to report the *size* of a bump — without having a baseline for the measurement. For these reasons, one must incorporate scale effects. We do so using the `geometric-neighbors()` function. In particular, we compute the *average* difference between neighboring face normals over some[5] region and compare the *individual* differences to that average. (This is essentially a spatial equivalent to the kinds of moving average or low-pass filtering algorithms that are used routinely in time-series analysis.) The right-hand image in Fig. 2 depicts the results. The normal vectors to each mesh element are shown as before, and each mesh element is grey-shaded according to how much the difference between its normal and those of its nearest neighbors differs from the local average of the surface normals. The cluster of lighter mesh elements in the bottom left corner of the right-hand image are part of a feature — a "surface spike" — whose distinguishing characteristic is lack of smoothness in slope.

Fig. 3 shows the `chatter` dataset, a simulation of a hard cylindrical pin pushing into a deformable block. Each point in this dataset gives the xyz position of a

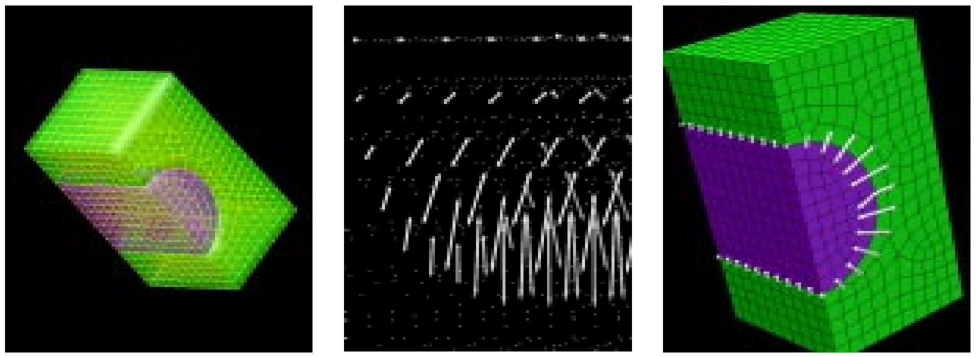

Fig. 3. A simulation of a hard cylindrical pin pushing into a deformable block. The left-hand figure shows the geometry; at the center is a 3D closeup view of the forces at a collection of grid points in and near the arch. In this rendering, the forces are difficult to see, let alone interpret; if the pin and block are shown in a view that suppresses information that is unrelated to the feature, however — as in the right-hand image — it is easy to identify places where the normal forces do not balance.

vertex and the force at that vertex. The issues that arise in this example are more complicated. Because there are physics variables involved, we are not only interested in features that violate mesh smoothness and continuity. In this particular case, we

[5] The size and shape of this region are an obvious research issue; our current solution chooses an arbitrary square, and we are investigating how best to improve this.

are also looking for *contact problems*: places on the surface between the two objects where the normal forces are not equal and opposite. Finding and describing contact problems with the tools described in the previous section is somewhat more involved than in the previous example, but still quite straightforward. We first find all mesh faces that lie on the contact surface between the two objects and determine which faces in the pin and the block touch one another. (Each object is a separate mesh, so this amounts to traversing the boundary of each, checking for xyz proximity of vertices and opposing faces, and building the appropriate association table.) We then compute the normals $\hat{n_i}$ to these faces, project the force vector $\boldsymbol{f}_i$ at each face along the corresponding $\hat{n_i}$ in order to eliminate its tangential[6] component, and finally compare the normal force vectors of adjacent faces to see if they are indeed of equal magnitude and in the opposite direction. The right-hand image shows the results, including a contact problem at one vertex, indicated by the bent vector in the middle of the arch.

Unlike the previous example, this dataset is complex and very hard to visualize: parts of the object obscure other parts, and it can be difficult or impossible to make sense of the geometry, as is clear from the center image in Fig. 3. In situations like this, automated feature characterization is critical, as it can find and highlight geometry that is effectively invisible to a human user — and even choose display parameters based on that investigation, in order to best present its results to that user (e.g., focusing in on the area around a feature and choosing a view that brings out its characteristic geometry, as in the right-hand image of the pin/block system).

These methods are not only very effective, but also quite extensible; one can detect a variety of other features using different combinations of these same basic tools. Tears and folds, for instance, can be flagged when geometric neighbors are not topological neighbors. Because we use the SAF format for the inputs and outputs of our toolkit routines, it is very easy to generate, modify, and use new combinations of those tools. The detection method for a specific feature, of course, is not uniquely defined; the `displacements()` and `statistics()` routines can also be useful in finding spikes, but geometric methods that rely on normal vectors are more precise. They are also more expensive, however, and when we begin working with the truly immense datasets that are the target applications of this project, we will likely use the former as a first pass on larger datasets, in order to identify areas for deeper study with the latter.

5 Related Work

This work draws upon ideas and techniques from a wide variety of disciplines, ranging from mathematics to artificial intelligence. Space requirements preclude a thorough discussion here; rather, we will just summarize a few of the methods and ideas that most closely relate to this paper. Many tools in the intelligent data analysis literature[2] focus on assessing different kinds of analysis tools and using that knowledge to build toolkits that adapt the analysis to the problem at hand. Our work is similar in spirit to many of these; indeed, our SAF-based framework solves the pernicious interoperability problems that motivate many of these toolkits. A handful of groups

[6] Tangential forces also play roles in different kinds of contact problems; see our website for more details.

in the IDA, pattern recognition, and machine learning communities specifically target reasoning about features in scientific data. Notable instances are the spatial aggregation framework of Yip and Zhao[17], Mitchell's GENIE program[6], and the AVATAR pattern recognition tool[3, 10], which invisibly watches as a user investigates a given 4D physics simulation dataset and deduces what s/he finds "interesting" in that data. Like these algorithms, our tool is designed to be both powerful and general, rather than domain-specific. Unlike GENIE and AVATAR, however, we assume that features can be described in closed form, and we are very interested both in those descriptions and in the process of discovering them. (Indeed, section 3.1 is essentially a chronicle of that knowledge engineering procedure.)

6 Conclusion

The goal of the intelligent data analysis tool described here is to distill a succinct description of the interesting and important features out of a massive simulation dataset. An automated tool like this, which produces a compact, useful description of the dynamics, couched in the language of the domain, frees human experts to devote their time and creative thought to other demanding tasks. It can not only classify the features in a data set, but also signal places where the expert analyst should take a closer look, and even aid in the presentation of the view — a critical prerequisite to effective visualization of a spatially complex datasets, where anything but a selective, narrowed focus will overwhelm the user with information. Equally important, it allows scientists to interact with their data at the *problem* level, encapsulating the details of and changes to the underlying infrastructure tools. Of course, this automated feature characterization tool will never *replace* the human expert. It does not do anything that cannot already be done by hand; it simply automates many of the more onerous, repetitive, and/or detailed parts of the analysis process.

While the feature ontology and the characterization algorithms described in this paper are specific to finite-element simulation data, the general ideas, the notion of a layered ontology, the mathematics routines that we use to implement the characterization process, and the compositional structure of the framework within which they are combined are far more broadly applicable. The results described here are only preliminary, of course; full assessment of the strengths and weaknesses of this approach will only be possible with much more experience and testing. To this end, we have begun working on turbulent convection problems, both numerical and experimental, where experts begin by reasoning about three basic structures in the 3D vorticity data: *tubes*, *sheets*, and *cells*, which play well-defined roles in fluid transport, and which can easily[7] be described using the same simple tools that we describe in section 3.2. The metadata facilities of the SAF libraries are an important part of what makes this work easily generalizable; to apply our tool to a new kind of data, we simply need to write the appropriate transliteration routine and pass it to SAF. This interoperability also makes the *results* of our tool's analysis more broadly applicable: our output contains not only raw data, but also metadata that describes the structure, format, and content of that data. This allows consumers of these results, whether human experts or other computer tools, to understand and use them. Incorporating the data analysis tool described here into an existing scientific computing

[7] to the great dismay of the project analyst

environment would further streamline this process, and we plan to investigate this when the next Ensight release — which will allow such modifications — becomes available.

Acknowledgments: Andrey Smirnov and Stephanie Boyles contributed code and ideas to this paper as well.

References

1. `http://www.ensight.com/`.
2. M. Berthold and D. Hand, editors. *Intelligent Data Analysis: An Introduction*. Springer-Verlag, 2000.
3. K. Bowyer, L. Hall, N. Chawla, T. Moore, and W. Kegelmeyer. A parallel decision tree builder for mining very large visualization datasets. In *Proceedings of the IEEE 2000 Conference on Systems, Man, and Cybernetics*, October 2000.
4. S. Brown and D. Braddy. PDBLib user's manual. Technical Report M270, Rev. 2, Lawrence Livermore National Laboratory, January 1993.
5. S. A. Brown, M. Folk, G. Goucher, and R. Rew. Software for portable scientific data management. *Computers in Physics*, 7(3):304–308, 1993.
6. S. P. Brumby, J. Theiler, S. Perkins, N. Harvey, J. J. Szymanski, J. J. Bloch, , and M. Mitchell. Investigation of image feature extraction by a genetic algorithm. In *Proceedings of SPIE*, 1999.
7. H. Edelsbrunner and E. Muecke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72, 1994.
8. W. H. *et al.* A lattice model for data display. In *Proceedings IEEE Visualization*, pages 310–317, 1994.
9. R. Haber, B. Lucas, and N. Collins. A data model for scientific visualization with provisions for regular and irregular grids. In *Proceedings IEEE Visualization*, pages 298–305, 1991.
10. L. Hall, K. Bowyer, N. Chawla, T. Moore, and W. P. Kegelmeyer. AVATAR — adaptive visualization aid for touring and recovery. Sandia Report SAND2000-8203, Sandia National Laboratories, January 2000.
11. M. C. Miller, J. F. Reus, R. P. Matzke, W. J. Arrighi, L. A. Schoof, R. T. Hitt, and P. K. Espen. Enabling interoperation of high performance, scientific computing applications: Modeling scientific data with the sets & fields (SAF) modeling system. In *International Conference on Computational Science (ICCS-2001)*, 2001.
12. F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, 1985.
13. V. Robins, J. Meiss, and E. Bradley. Computing connectedness: An exercise in computational topology. *Nonlinearity*, 11:913–922, 1998.
14. V. Robins, J. Meiss, and E. Bradley. Computing connectedness: Disconnectedness and discreteness. *Physica D*, 139:276–300, 2000.
15. L. Schoof. The ASCI data models and formats (DMF) effort: A comprehensive approach to interoperable scientific data management and analysis. In *4th Symposium on Multidisciplinary Applications and Interoperable Computing*, Dayton, OH, August 2000.
16. D. Wells, E. Greisen, and R. Harten. FITS: A flexible image transport system. *Astronomy and Astrophysics Supplement Series*, 44:363–370, 1981.
17. K. Yip and F. Zhao. Spatial aggregation: Theory and applications. *Journal of Artificial Intelligence Research*, 5:1–26, 1996.