Available online at www.sciencedirect.com
SCIENCE DIRECT®
Future Generation Computer Systems 21 (2005) 53–59
FGCS FUTURE GENERATION COMPUTER SYSTEMS
www.elsevier.com/locate/future

Rule-based visualization in the Discover computational steering collaboratory☆

Hua Liu[a,*], Lian Jiang[b], Manish Parashar[a], Deborah Silver[b]

[a] *TASSL, Department of Electrical and Computer Engineering, Rutgers, The State University of New Jersey, Piscataway, NJ 08854, USA*
[b] *VIZLAB, Department of Electrical and Computer Engineering, Rutgers, The State University of New Jersey, Piscataway, NJ 08854, USA*

Available online 30 October 2004

Abstract

In this paper, we introduce the concept of rule-based visualization for a computational steering collaboratory and show how these rules can be used to steer the behaviors of the visualization subsystem. Rules define high-level policies and are used to autonomically select and tune the visualization routines based on application requirements and available computing/network resources. Such an autonomic management of the visualization subsystem can significantly improve the effectiveness of computational steering collaboratories in wide-area Grid environments.

Keywords: Rule-based visualization; Computational steering; Feature tracking

1. Introduction

A computational steering collaboratory is an environment in which geographically distributed scientists can collaboratively investigate complex simulations using online remote visualization and computational monitoring/steering techniques. Computational steering not only shortens the period between changes to parameters and the viewing of the results, but enables a what-if analysis, which makes cause–effect relationships more evident [1]. The ability to flexibly manipulate the visualization subsystem in a computational steering collaboratory is important, as visualization is typically the basis for interactive monitoring, steering and multi-user collaboratory. However, for large-scale long-running simulations, it may not be feasible to download an entire data set or even one timestep of the data set to a visualization platform. Therefore, sometimes visualization routines are co-located at the simulation platform, and the methods must be selected a priori, which may not be the most efficient.

☆ This material is based upon work supported by the National Science Foundation under grant numbers ACI-9984357, EIA-0103674, EIA-0120934, CNS-0305495 and 0082634 and by DOE ASCI/ASAP via grant numbers PC295251 and 82-1052856. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

* Corresponding author.
E-mail addresses: marialiu@caip.rutgers.edu (H. Liu), lianjian@caip.rutgers.edu (L. Jiang), parashar@caip.rutgers.edu (M. Parashar), silver@caip.rutgers.edu (D. Silver).

0167-739X/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2004.09.020

In this paper, we present a rule-based visualization system. Rules are decoupled from the system and can be externally injected to manage visualization behaviors at runtime, such as autonomically selecting the appropriate visualization routines and methods, and adjusting the extraction threshold. For example, the scientist may specify the rule "if the number of the extracted regions is greater than 100, then increase the threshold by 5". This rule will inform the visualization system to autonomically change the threshold under different conditions. Further, if the scientist knows from monitoring the first 200 timesteps that the value "5" is not appropriate, he/she can modify the rule at runtime and the modified rule will be applied to the rest of the simulation. Rules can also be used to support collaborative visualization in heterogeneous environments where the collaborators' resources and display capabilities may differ. For example, rendering an isosurface with millions of polygons may be too computation/network-intensive for a thin client like a PDA. Either the polygons can be reduced using straightforward triangle decimation techniques, or a more abstract feature-based representation can be displayed [2]. Such autonomic adaptations can be simply achieved using a rule such as "if there are more than 10 K triangles, then display a higher level of abstraction".

The rule-based visualization subsystem presented in this paper builds on Discover, which is a computational collaboratory for interactive Grid applications and provides the infrastructure for enabling rules to be dynamically and securely composed, injected into the application, and executed so as to autonomically adapt and optimize its behaviors [3]. In this paper, we integrate the rule mechanism into a feature-based visualization subsystem and demonstrate how this can be used to improve monitoring, steering and collaboration.

2. Related work

Computational steering systems have been developed to explore models (e.g., VASE [4], Discover [3], CSE [5] and CUMULVS [6]), experiment algorithms (e.g., VASE) and optimize performance (e.g., CUMULVS and Virtue [7]) [1]. However, few of the existing computational steering systems [3,5–7] support multi-user collaboration. Among the collaborative computational steering systems such as CUMULVS, Virtue, Discover and CSE, only the first three have visualization functionality. Discover and Virtue support WAN-based collaboration, where flexibility and efficiency are critical. Vetter and Reed [7] propose a visualization hierarchy for Virtue. Given a high-level perspective, users can selectively drill down to the lower level for more detail of interest. Feature-based visualization presented in this paper can also be used to similarly drill down to different levels of feature detail.

The rule-based visualization presented in this paper can be used for post-processing visualization, in which the data set is downloaded to a special visualization machine and is interactively interrogated using visualization techniques, as well as for in situ visualization, which runs with the simulation and bypass the need to download the data, allowing the scientist to get a first look at the data.

3. The Discover computational collaboratory

Discover is a virtual, interactive and collaborative PSE that enables geographically distributed scientists and engineers to collaboratively monitor, and control high-performance parallel/distributed applications using web-based portals [3]. As shown in Fig. 1, Discover provides a 3-tier architecture composed of detachable thin clients at the front-end, a peer-to-peer network of servers in the middle, and the Distributed Interactive Object Substrate (DIOS++) [8] at the back-end.

DIOS++ enables rule-based autonomic adaptation and control of distributed scientific applications. It is composed of three key components: (1) autonomic objects that extend computational objects with sensors (to monitor the state of the objects), actuators (to modify the state of the objects), access policies (to control accesses to sensors and actuators) and rule agents (to enable rule-based autonomic self-management); (2) mechanisms for dynamically and securely composing, deploying, modifying and deleting rules; and (3) a hierarchical control network that is dynamically configured to enable runtime accesses to and management of the autonomic objects and their sensors, actuators, access policies and rules.

Discover is currently operational and is being used to provide interaction capabilities to a number of scientific and engineering applications. Furthermore, the Discover middleware substrate provides interoperabil-

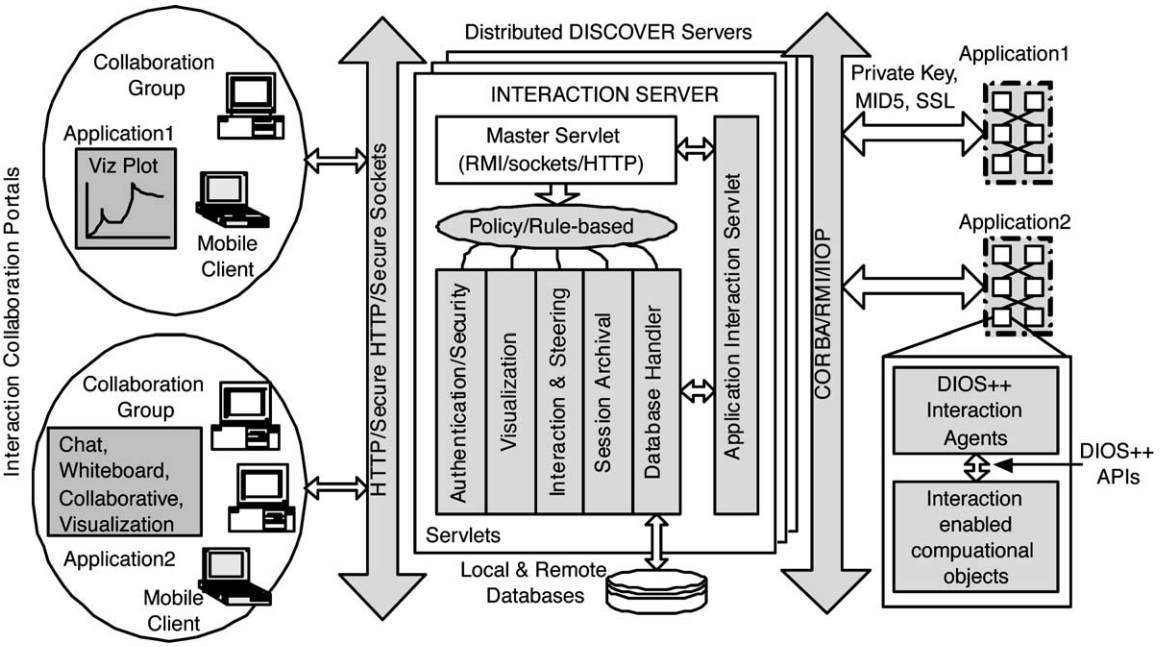

Fig. 1. The architecture of the Discover computational collaboratory.

ity between Discover interaction and collaboration services and Globus Grid services.

4. Feature extraction and feature tracking

Feature-based visualization allows scientists to extract regions of interests (the features), and then visualize, track and quantify the evolution of these features. In this work, we show how feature-based visualization can be effectively used in a rule-based computational steering collaboratory.

Features are tracked from one timestep to the next to capture how the features evolve [2]. Feature tracking allows events to be catalogued and enables complex queries to be performed on the data set. Queries include data-mining-type exploration, such as "how many new regions appear in timestep t_i?" or "in which timesteps do large regions merge?". The framework of feature-based visualization is shown in Fig. 2. The first step in feature-based framework is defining the feature of interests. There are many different definitions and we utilize the most basic one here, i.e., features are defined as connected regions which satisfy some thresholds (such as isosurfaces or volume rendering) [2]. After the features are extracted, the feature attributes, such as isosurface, mass, volume and centroid, can be computed. The features can also be abstracted using a simpler shape. One such reduced representation is an ellipsoid that provides an iconic abstraction to blob-like regions [9], as shown in Fig. 2. Other types of abstractions include more line-like regions such as skeletons [10] and vortex cores [11], and critical points/curves for vector

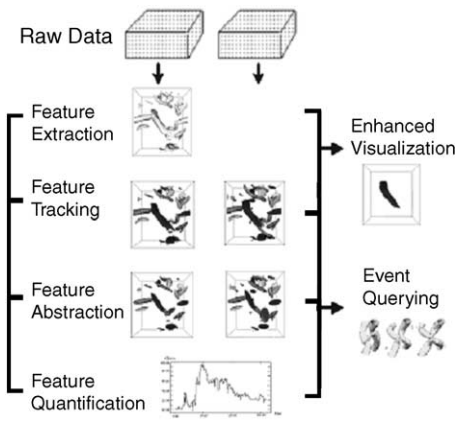

Fig. 2. A feature-based visualization.

fields [12]. For scalar fields, contour trees [13] can also provide an abstraction.

In previous work, feature extraction and tracking has been implemented as a post-process distributed visualization routine [14]. In this paper, we build upon this distributed routine and demonstrate an integrated approach within Discover using the rule-based mechanism. This allows scientists to control aspects of the feature extraction process for greater flexibility. In our previous version, the visualization parameters had to be set at the start of the simulation, but they can now be autonomically controlled during the simulation.

5. Rule-based autonomic visualization

Ftrack, the feature-based visualization system integrated with Discover, is capable of autonomic behaviors based on the rules defined by users at runtime. The rules are categorized as: (1) *steering rules* that enable intra-function management, e.g., changing the runtime behaviors of the visualization functions by dynamically altering their parameters and (2) *configuration rules* that enable inter-function management, e.g., organizing the visualization capabilities by selecting the appropriate routines to be executed. Examples of rules and the related sensors, actuators applicable to *Ftrack* are presented below.

Steering rule. The rule for choosing an appropriate threshold mentioned in Section 1 is an example of steering rule. Another example given here is a rule for changing the color schemes. The level-highlighting color scheme gives each level a different hue and assigns different saturation to each feature. The time-highlighting color scheme assigns different hues to the feature at different timesteps.

```
IF getAverageNumberofFeatures()<3 THEN useLevelHighlightingScheme()
                                  ELSE useTimeHighlightingScheme()
```

This rule informs *Ftrack* to highlight time tracking when the number of features is very small (3 in this example), otherwise highlight refinement levels. This rule is composed of one sensor, *getAverageNumberofFeatures*, and two actuators, *useLevelHighlightingScheme* and *useTimeHighlightingScheme*, exposed by *Ftrack*.

Configuration rule. Scientists may choose to vary the visualization routines based on the computing/network resources available at the time of visualization. For example, when the number of grid cells in the data set exceeds a threshold, the scientist at a thin client may want to display ellipsoids instead of isosurfaces, which is specified as Rule1 below. The scientist can modify Rule1 at runtime to change 10–50 K or switch THEN and ELSE statements. If the scientist is working on a PDA, which typically has poor graphics resolution as well as limited memory capacity, Rule2 can be defined to dynamically modify the visualization behavior.

```
Rule1: IF getNumCells()>10K
       THEN {setEllipsoidFitting(true); setIsosurface(false)}
       ELSE {setEllipsoidFitting(false); setIsosurface(true)}
Rule2: IF isPDA()==TRUE
       THEN {setEllipsoidFitting(true); setIsosurface(false)}
```

6. Rule-based visualization using Discover

The visualization subsystem, *Ftrack*, which performs the distributed feature extraction and tracking, is designed as a DIOS++ object. As shown in Fig. 3, the *Ftrack* object consists of three actors, each managing a part of the visualization task.

Feature extractor extracts the features of interest, and computes the geometry attributes (e.g., isosurface) and quantification attributes (e.g., volume, mass, tensor) for each feature. Furthermore, it can calculate global statistics such as the average number of features, the variation in the number of features, etc.

Feature tracker tracks the extracted features. The input is the feature information from feature extractor and the steering commands defined by the users via the

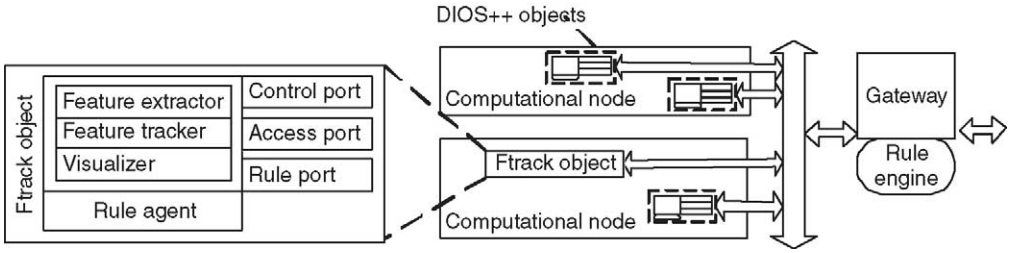

Fig. 3. Ftrack within Discover/DIOS++.

portals or by the rule engine. The output is the tracking information represented as a set of graphs.

Visualizer includes various visualization modules supported by the feature extractor and feature tracker. It utilizes the results from the other two actors to create visualizations of the features (e.g., an isosurface rendering displaying quantification) and sends the images and/or data to the Discover portals.

The *control port* exposes sensors and actuators to allow the state of the visualization subsystem to be externally monitored and controlled by users. The *access port* controls access to these sensors/actuators based on users' privileges and capabilities. The *rule port* contains the rules evaluated and executed by the rule agent (RA) to autonomically monitor, adapt and control the *Ftrack* object. The rule-based visualization process is summarized below.

Initialization and interaction. During initialization, the application uses the DIOS++ APIs to register its objects, export their sensors/actuators, functional interfaces and access policies to the local computational nodes, and further to the Gateway, which then updates its registry. The rule engine is co-located with Gateway, and thus has access to the Gateway's registry. The Gateway interacts with the external access environment and coordinates accesses to the application's sensors/actuators, policies and rules as a broker.

At runtime, the Gateway may receive incoming interaction or rule requests from users. The Gateway first checks the user's privileges based on the user's role, and refuses any invalid access. It then forwards valid interaction requests to destination objects and forwards valid rule requests to the rule engine. The rule engine analyzes the rules, decomposes them if necessary, and injects them to the corresponding objects. In our implementation, visualization rules are injected into *Ftrack*. *Ftrack* invokes the sensors and functional interfaces exposed by computational objects to get real-time data, performs feature-tracking computations based on the rules and sends the visualization data to the portals for display.

Rule deployment and execution. The rule engine dynamically creates rule agents for *Ftrack* and other objects if they do not already exist. It then composes a script for each agent that defines the rule agent's lifetime and rule execution sequence based on rule priorities.

While typical rule execution is straightforward (actions are issued when their required conditions are fulfilled), the application dynamics and user interactions make things unpredictable. As a result, rule conflicts must be detected at runtime. In DIOS++, rule conflicts are detected at runtime and are handled by simply disabling the conflicting rules with lower priorities. This is done by locking the required sensors/actuators. For example, configuration rules Rule1 and Rule2 conflict if *getNumCells()* is less than 10 K while *isPDA()* is TRUE. Assuming Rule1 has higher priority, the script will inform the rule agent to fire Rule1 first. After Rule1 is executed, interfaces *setEllipsoidFitting()* and *setIso-*

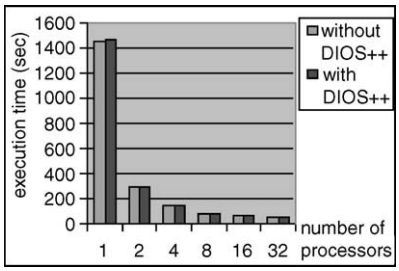

Fig. 4. Runtime overheads introduced in the minimal rule execution mode.

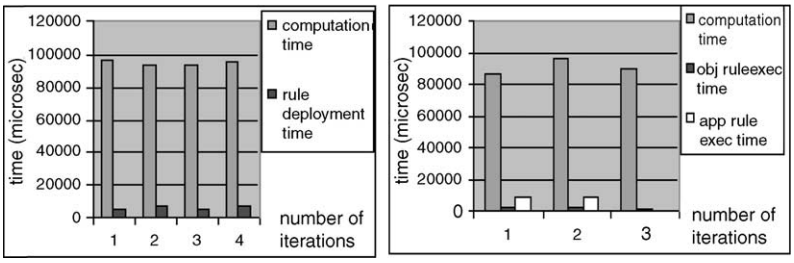

Fig. 5. Left: Comparison of computation and rule deployment time. Right: Comparison of computation, object rule execution and application rule execution time.

surface() are locked during the period when getNumCells() is less than 10 K. When Rule2 is issued, it cannot be executed as its required interfaces are locked. The two interfaces will be unlocked when *getNumCells()* becomes greater than 10 K. By modifying the rules in the rule base through the Discover portal, the scientist can increase the priority of Rule2 higher than that of Rule1 so that the visualization subsystem always displays ellipsoids if *isPDA()* is TRUE.

7. Experiments and evaluation

This section summarizes experimental evaluations using the IPARS reservoir simulation on a 32-node beowulf cluster. IPARS is a Fortran-based parallel/distributed reservoir simulator. Using DIOS++/Discover and Ftrack, engineers can interactively feed in rules to dynamically determine the parameters such as water/gas injection rates and well bottom hole pressure, and visualize the water/oil ratio or the oil production rate.

Experiment 1 (Fig. 4). This experiment measures the runtime overhead introduced to the application in the minimal rule execution mode. In this mode, the application autonomically updates the rule engine, *Ftrack*, Discover server and its connected clients with current state of autonomic objects and rules, and explicit interaction and rule execution are disabled. The application's runtime with and without DIOS++/Ftrack are plotted in Fig. 4. It can be seen that the runtime overheads are very small and within the error of measurements.

Experiment 2 (Fig. 5, left). This experiment compares the computation time and rule deployment time for successive iterations. At each of the four iterations, a new rule is injected into the DIOS++ and deployed by the rule engine to the destination objects. It shows that the rule deployment time is much less than the computation time in each iteration.

Experiment 3 (Fig. 5, right). This experiment compares the computation time, object rule execution time and application rule execution time for successive application iterations. Object rules only affect one object, while application rules may manage several objects. Therefore, application rules are more expensive than object rules.

8. Conclusion

This paper presented a rule-based visualization system that improves the flexibility of visualization in a WAN-based computational steering collaboratory. Rules can be used to steer in situ visualization and aid in data mining. In a heterogeneous environment, rules can help the scientists specify the type of interaction and visualization desired based on system capabilities. The rule-based visualization system built on the Discover/DIOS++ computational collaboratory. An experimental evaluation of Discover/DIOS++ was presented in the paper.

References

[1] J.D. Mulder, J.J. van Wijk, R. van Liere, A survey of computational steering environments, Future Gen. Comput. Syst. 15 (1) (1999) 119–129.

[2] D. Silver, X. Wang, Tracking and visualizing turbulent 3D features, IEEE Trans. Visualization Comput. Graph. 3 (2) (1997) 129–141.

[3] V. Mann, V. Matossian, R. Muralidhar, M. Parashar, Discover: an environment for web-based interaction and steering of high-performance scientific applications, Concurrency – Pract. Exp. 13 (8–9) (2001) 737–754.
[4] D.J. Jablonowski, J.D. Bruner, B. Bliss, R.B. Haber, VASE: the visualization and application steering environment, in: Proceedings of the Supercomputing'93, Portland, OR, 1993, pp. 560–569.
[5] R. van Liere, J.J. van Wijk, CSE: a modular architecture for computational steering, in: Virtual Environments and Scientific Visualization, Springer Verlag, Vienna, 1996, pp. 257–266.
[6] G.A.I. Geist, J.A. Kohl, P.M. Papadopoulos, CUMULVS: providing fault tolerance, visualization, and steering of parallel applications, Int. J. High Perform. Comput. Appl. 11 (3) (1997) 224–236.
[7] J.S. Vetter, D.A. Reed, Real-time performance monitoring, adaptive control, and interactive steering of computational grids, Int. J. High Perform. Comput. Appl. 14 (4) (2000) 357–366.
[8] H. Liu, M. Parashar, DIOS++: a framework for rule-based autonomic management of distributed scientific applications, in: Proceedings of the Ninth International Euro-Par Conference, Lecture Notes in Computer Science, Springer-Verlag, Klagenfurt, Austria, 2003, pp. 66–73.
[9] T. van Walsum, F.H. Post, D. Silver, F.J. Post, Feature extraction and iconic visualization, IEEE Trans. Visual. Comput. Graph. 2 (2) (1996) 111–119.
[10] F. Reinders, M.E.D. Jacobson, F.H. Post, Skeleton graph generation for feature shape description, in: Proceedings of the Symposium on Data Visualisation, Springer, 2000, pp. 73–82.
[11] D.C. Banks, B.A. Singer, A predictor–corrector technique for visualizing unsteady flow, IEEE Trans. Visual. Comput. Graph. 1 (2) (1995) 151–163.
[12] J. Helman, L. Hesselink, Representation and display of vector field topology in fluid flow data sets, IEEE Comput. 22 (8) (1989) 27–36.
[13] S. Sural, G. Qian, S. Pramanik, Segmentation and histogram generation using the HSV color space for image retrieval, in: Proceedings of the IEEE International Conference on Image Processing, IEEE, Rochester, 2002, pp. 589–592.
[14] J. Chen, D. Silver, L. Jiang, The feature tree: visualizing feature tracking in distributed AMR data sets, in: Proceedings of the IEEE Symposium on Parallel and Large-data Visualization and Graphics, IEEE, Seattle, WA, 2003, pp. 103–110.

Hua Liu is a Ph.D. candidate of the Center for Advanced Information Processing in the Department of Electrical and Computer Engineering at Rutgers University. Her research interests include autonomic computing, parallel, distributed and Grid computing.

Lian Jiang is a Ph.D. candidate of the Center for Advanced Information Processing at Rutgers University, New Jersey, USA. His research interests include computer graphics and scientific visualization, parallel and distributed computing.

Manish Parashar is an Associate Professor of Electrical and Computer Engineering at Rutgers University, where he also is Director of the Applied Software Systems Laboratory. He received a B.E. degree in Electronics and Telecommunications from Bombay University, India, in 1988, and M.S. and Ph.D. in computer engineering from Syracuse University in 1994. He has received the NSF CAREER Award (1999) and the Enrico Fermi Scholarship from Argonne National Laboratory (1996). His current research interests include autonomic computing, parallel, distributed and Grid computing, networking, scientific computing, and software engineering. He is a member of the executive committee of the IEEE Computer Society Technical Committee on Parallel Processing (TCPP), part of the IEEE Computer Society Distinguished Visitor Program (2004–2006), and a member of ACM. He is also the co-founder of the IEEE International Conference on Autonomic Computing (ICAC). He has co-authored over 130 technical papers in international journals and conferences, has co-authored/edited five books/proceedings, and has contributed to several others in the area of parallel and distributed computing.

Deborah Silver is an Associate Professor in the Department of Electrical and Computer Engineering at Rutgers, The State University of New Jersey. She received a B.S. from Columbia University School of Engineering in 1984 and an M.S. (1986) and a Ph.D. (1988) from Princeton University in Computer Science. Her area of research is scientific visualization and computer graphics and she is the PI of the VIZLAB, which is part of the CAIP Center at Rutgers University since joining the faculty in 1988. She has taught courses in computer graphics, visualization, data structures, software engineering and robotics. She has over 80 publications in visualization and computer graphics. She has worked on many aspects of visualization and is currently involved in distributed visualization, volume graphics, oceanographic visualization and multimedia research efforts. She has been co-chair of the papers session and program co-chair of the yearly IEEE Visualization Conference, Vice Chair of operations for the IEEE Technical Committee on Computer Graphics (1993–2000), and a member of editorial committee of the IEEE Transaction on Visualization and Computer Graphics (1995–2000).