Visualization, Virtual Reality, and Animation within the Data Flow Model of Computing

Richard E. Gillilan and Frank Wood
Cornell University

Introduction

This paper presents our perspective on the utility of data-flow programming in the field of scientific visualization. The Cornell Theory Center (CTC) is an established center of visualization production. Scientists from across the country use the center's resources to explore the data collected from their research. The CTC uses IBM's Visualization Data Explorer™ to do most of its visualization, and maintains a repository of DX extensions that are available free to the public.

The discussion in this paper primarily focuses on the flexibility and speed of development afforded by the use of modular programming. In particular, DX is shown to provide sufficient flexibility to be useful in settings ranging from animation production to Cornell computer science education. Examples from actual work in progress are used in this paper to underpin our advocacy of modular data-flow programming. We begin by examining how data from a specialized application such as chemistry can easily fit within the mathematical model of scientific data representation provided by DX. Included as examples of the modular extensibility of DX are a description of the link between DX and the Electronic Visualization Laboratory's CAVE virtual reality environment and an explanation of the CTC-developed DX interface to RenderMan.™ Finally, as an example of the quick learning curve associated with DX, sections of the curriculum developed for Cornell computer science classes on graphics (CS417 and CS418) are presented.

Researchers not only need to combine existing tools in novel and innovative ways, but need the flexibility to add new tools and new interfaces. Virtual reality, for example, is only beginning to be used in actual research, so existing codes may not yet address the needs of a specific community like chemists. Researchers also need to present their data in a clear and polished form which not only involves higher-quality, more photorealistic rendering but animation as well. With the exception of initial design, these applications are less interactive and more batch-oriented. Education on the other hand, has its own special requirements. In the sections that follow, we address each of these needs, discussing how the data-flow paradigm fits into the application area and how it can be interfaced with existing programs in the discipline.

Data Structures in Chemistry

Interactive graphics has been used extensively in chemistry for many years. With the size of molecules under study becoming ever larger and more complex, stereo graphics displays and hardware rendering engines have become the norm in many laboratories. This early appetite for graphics was, to a great extent driven by the birth of x-ray crystallography, a field in which many thousands of atomic positions must be determined from one or more 3-dimensional electron density distributions. Beyond determining structure, chemists now look to molecular graphics for insight into the mechanisms of interaction, reaction and catalysis of large molecular assemblies.

A number of geometrical constructs have evolved which have become valuable to chemists over the years. True to the ideals of scientific visualization, they are simplified representations that leave out unnecessary complexity while highlighting essential and relevant physics. These constructs, the CPK model, peptide ribbons, molecular surfaces etc. have served well over the years and will continue to be primary visualization tools into the future.

The first step in a scientific visualization is to read in the data and to choose how to organize it into data structures. This section describes how molecular structure data, a specialized application, can be represented with DX's model of scientific data representation.

IBM Visualization Data Explorer is a modular data-flow scientific visualization package based on a client-server execution model [2]. Visual programs, which resemble flow charts, are created using a point-and-click graphical user interface. The boxes in the flow chart represent *modules*, functions that operate on data. Data is passed from module to module in the form of general user-definable data structures. The wires in the flow chart which

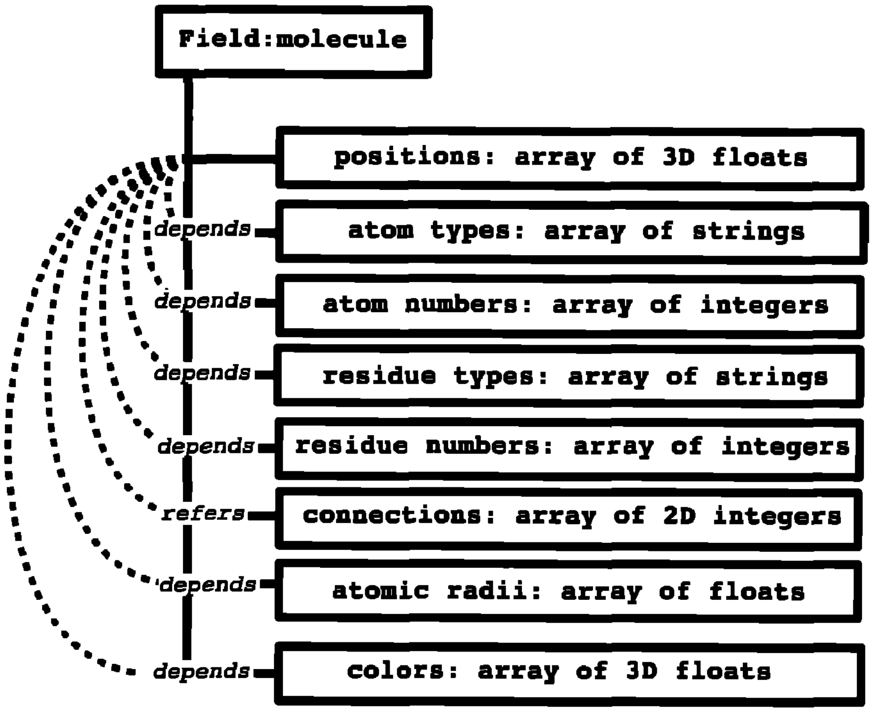
Field:molecule
positions: array of 3D floats
depends
atom types: array of strings
depends
atom numbers: array of integers
depends
residue types: array of strings
depends
residue numbers: array of integers
refers
connections: array of 2D integers
depends
atomic radii: array of floats
depends
colors: array of 3D floats

Figure 1: Diagram of a molecular data structure expressed within IBM Visualization Data Explorer's model of scientific data representation. The field is a group of arrays related by user-specified dependencies. When the atomic radii component of the field, for example, depends on positions, it is understood that each position 3-vector in the field has a corresponding atomic radius. A component like connections refers to positions since each element of the integer connections array is an index into the positions array.

connect modules can be thought of as data pipelines. Tabs on modules represent possible connection points. All information necessary for geometry, interaction, rendering and even final images are passed as DX data structures into these tabs. A detailed description can be found elsewhere [3].

For molecular structure data, coordinates are stored as a DX field, a group of arrays that map onto each other in user-defined ways. In our case, a minimal molecule field consists of positions (the atomic coordinates), atom types, indices and other information regarding larger scale or so-called secondary structure. Members of fields can be added or transformed by modules as the data structure flows down the network. *Figure 1* gives a diagrammatic representation listing the data type for each member commonly encountered in a molecular structure field along with how it relates to other members of the field.

Modules and Extensibility

There are a number of graphics constructs unique to chemistry. The simpler models of molecules, the CPK and ball-and-stick representations for example, can trivially be created using the existing tools provided with DX. They can then encapsulated as single boxes called macros; more specialized models with nontrivial algorithms, however, call for the creation of modules. From the point of view of the end user, there is no distinction between macros and modules, although modules enable developers to encapsulate their own code.

We present a simple example recently encountered while working with a team of chemists studying how molecules bind to a newly discovered protein. The researchers had found two very similar proteins, call them *A* and *B*, with over a 100-fold difference in ability to bind a particular molecule. They wanted a graphical way to compare the binding sites for the two proteins to find the reason for the difference. It was natural to compute and superimpose the so-called solvent-accessible or molecular surfaces first. The highly efficient code developed by Amitab Varshney [4,5] creates a triangle-mesh realization of the surface. Creation of a molecular surface module from this code required only some additional hashing to remove redundant vertices. The input data structure must contain fields at its lowest level, each having atomic positions and radii components. The output retains the same data structure with each original field replaced by a field containing the positions and connections of the triangle mesh. The resulting visualization clearly shows which parts of one surface protrude beyond the other but gives no indication of relative distance. We decided that the best solution was to color code the first surface *A* according the the signed distance of closest approach to the other *B*.

An algorithm was developed based on formulae presented in Ref [6]. A simple hashing scheme was used to reduce the computational effort. We called the resulting module *CompareSurfaces*. *Figure 2* shows the DX visual program which computes the two surfaces and feeds them into CompareSurfaces. The output data structure is the same as the input data structure for the first tab (surface) but with two additional components. The "data" component, which depends on the vertex positions, is the signed distance from the vertices to the surface *B*. The "segs" component is the actual vector to the surface. The Image window in *Figure 2* shows the surface *A*, color-coded by signed distance to surface *B*. The darker the color, the large the separation between the two. We also superimpose surface *B* as a grey wire mesh.

Virtual Reality

Data Flow in the CAVE

The CAVE, or Cave Automatic Virtual Environment, was developed at the Electronic Visualization Laboratory of the University of Illinois at Chicago (EVL). It is a room in which the floor and two adjoining walls are stereo projections systems [7,8]. Viewers with head tracking experience computer-generated 3D objects as if they were actually present in the room. The CAVE library, developed at the EVL, provides all the initialization routines required for the graphics hardware and automatically does all stereo perspective calculation. Graphical function calls are made in native GL (Silicon Graphics Graphics Library™ and are passed to the Reality Engine hardware rendering engine.

We have created an inboard module, Cave, that allows users of DX to create and pass geometries through shared memory to the CAVE. The module extracts polygons, normals, colors, lines, points, and lights from the data structure and parses them into GL and CAVE library function calls. With this configuration, the full power of DX is available to control calculations on large geometries. The benefits of parallel computation can be harnessed by DX's inherent support for parallelism. DX also provides the flexibility necessary to link different hardware and software systems, capturing the best of both.

The match is by no means optimal. DX (version 2.1) does not currently support data feedback loops, and thus the data path from DX to the CAVE is one way. For example, a

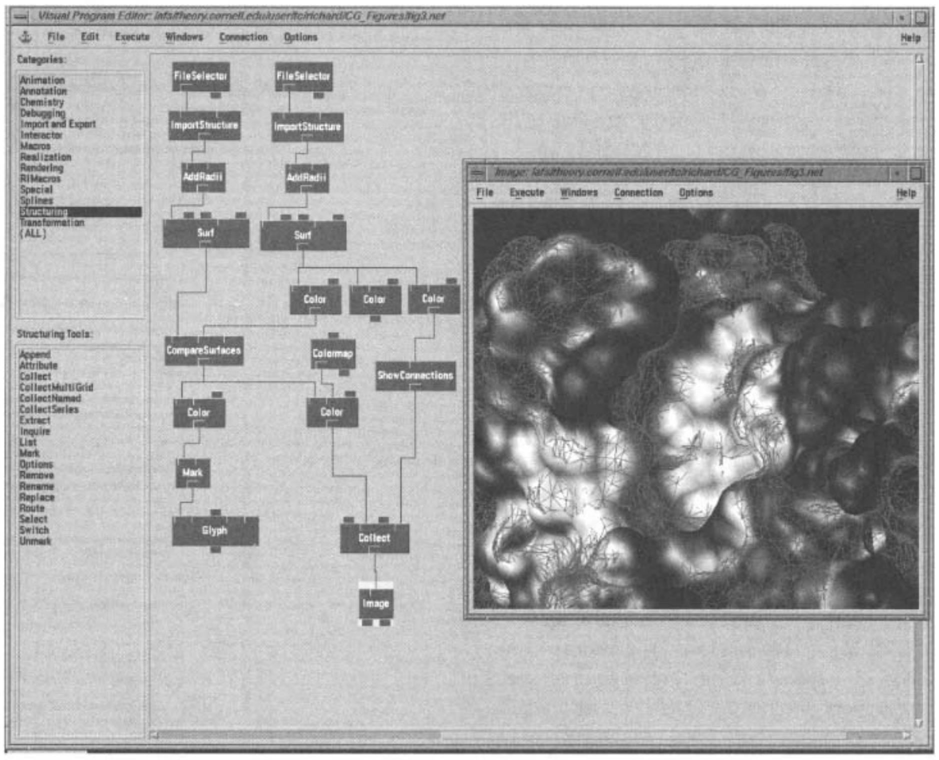

Figure 2: *IBM Visualization Data Explorer user interface. The block diagram is a visual program that generates two molecular surfaces and compares them using our custom module CompareSurfaces. The first surface is shown in the Image window color-coded by signed distance to the second surface. The second surface is also shown as a grey wire mesh. Darker regions have higher separations. The close similarity of the two surfaces is easily seen in the central pocket.*

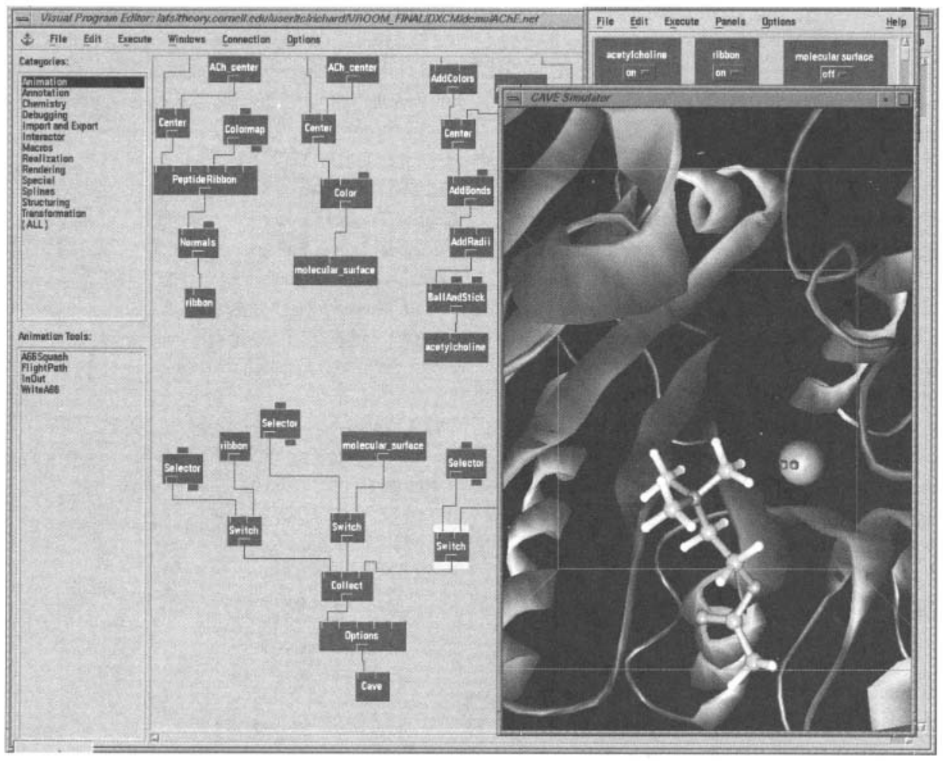

Figure 3: The DX-to-CAVE module transfers geometries generated in the modular visualization environment into the virtual reality environment. The CAVE is an immersive virtual reality system developed at the University of Illinois Electronic Visualization Laboratory. The image on the right is created using the simulator version of the code and enables developers to see the CAVE graphics on a single workstation screen.

Sequencer module can be used to update CAVE geometries, but information from the CAVE's 3D wand is isolated to the CAVE module's internal code. It is possible, however, for the CAVE's 3D wand to be used to access DX's control panels. To accomplish this, we have encorporated code which warps the X Windows cursor and issues button-press X events from the wand. Users can push buttons on control panels and even edit their visual program from inside the CAVE. *Figure 3* shows the a visual program in the DX user interface with a CAVE module. In simulator mode, the CAVE module produces an ordinary graphics window with lines delimiting the CAVE walls. The window in *Figure 3* shows a ribbon representation of a protein with a sphere representing the user's location in the CAVE.

Parallel Animation and the RenderMan™ Interface

Although DX was never designed as an animation choreography program, it has proven to be a highly productive tool for producing short scientific animations. The Sequencer module module provides visual programs with a stream of integers controlled by a VCR-like control panel. Each time an integer is generated, the program is reexecuted. Although DX does provide basic rendering capabilities, polished animations often require high-quality images. The commercial RenderMan package (Rman) from PIXAR fills this need. We have written a DX-to-RenderMan module which allows DX users to access nearly all of RenderMan's sophisticated functionality. In this example, researchers wanted a symbolic representation of the acetylcholine receptor. Our crude model is represented by a surface of revolution primitive with atoms on the post-synaptic membrane represented by sphere primitives. The surface of revolution was created by scanning in a black and white illustration and then using DX's ProbeList module to trace a cross section. DX is not capable of rendering either type of primitive, but we have written an outboard module which produces RenderMan rib files. The DX data structure is sufficiently flexible to support almost all RenderMan functionality. When the RMan module encounters a field with positions and data, it looks for what is called an attribute, a named tag that can be attached to any DX data structure. In RenderMan the states of the graphics environment are also called attributes. DX attributes can be conveniently used to set RenderMan attributes such as the lighting model being used, etc. When a DX attribute called "geometry" has a string value of "sphere," RMan looks for a array of positions and corresponding radii and places spheres at each position. Similarly, a string value of "surface of revolution" will assume that an array of positions in a field defines the cross section of a surface of revolution about the origin. If RMan encounters a field with a geometry attribute that has another data structure as a value, that data structure is assumed to define a "glyph" that should be treated as a RenderMan object and replicated at each position of the field. This is a much more memory-efficient scheme than using the conventional DX Glyph module that replicates the entire geometry in memory. *Figure 4* shows the final image.

Animation at the Cornell Theory Center is currently done in parallel on 10 128MB thin nodes of an IBM SP2. Both rib files (2-3MB on average) and final rendered images (0.7 MB)

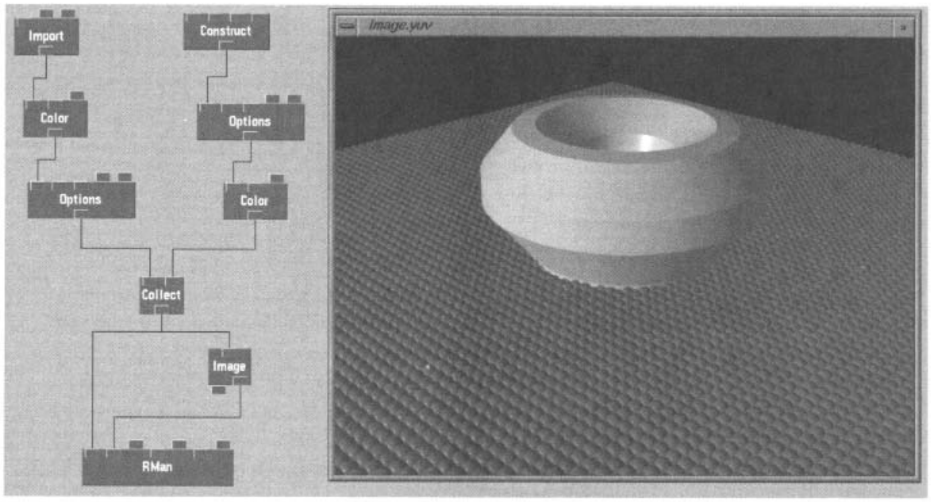

Figure 4: The commercial code RenderMan available from PIXAR easily interfaces to IBM's Data Explorer. The advanced data structures of the modular visualization environment enable users to access features such as geometric sphere and surface-of-revolution primitives. Options modules shown here notify our RMan module that the incoming data structures should be interpreted as the appropriate geometric primitives.

are written to our mass storage system so that any frame of a previous animation can be retrieved in digital form or re-rendered from the rib file at higher resolution if needed.

Education

Although DX is primarily a scientific data visualization tool, it is successfully used in the instruction of Cornell computer science classes 417 and 418 on computer graphics and visualization. The course exercises range from explicit polygon listing to Z-buffering and antialiasing. DX is used in these classes because of the quick learning curve associated with modular programming. Students are able to experiment with aspects of computer graphics which individually would take months of programming to explore. For young students, a lack of practical programming experience can often cause problems in a traditional computer graphics course. Instead of wasting time learning huge libraries of functions, students can pick modules and customized macros from a list and then assemble them into short visual programs to test principles. Macros can be opened to see the various visual subroutines employed within. Those subroutines can then be enhanced or "turned off" by replacing them with student developed algorithms. Comments on DX's modularity and ease of use have appeared in the positive student feedback. In them, DX is shown to be an easy medium for investigating computer graphics principles because of its quick learning curve. Detailed information including student generated images is available through the Cornell Theory Center's World Wide Web server, the URL of which is given below.

Summary

We have shown applications of DX to illustrate the ease of use and flexibility afforded by modular data-flow programming. The flexibility of the DX data structure allows optimal integration of unique applications, either forefront technology or specialized algorithms. We have demonstrated that modular programming, especially when conjoined with a graphical user interface such as that in DX, can bring sophisticated and powerful computer tools within reach of the non-programmer. Currently, many research groups are accustomed to doing their computation and visualization in a piecemeal fashion from disparate programs, some public domain, others written by former group members or companies. In all, we think that modular visualization environments will play a major role in the future of research and teaching by providing a unified interface that brings together disparate applications, promotes exchange, and simplifies computing. For more information regarding visualization at the Cornell Theory Center, open the URL http://www.tc.cornell.edu/Visualization/. Cornell's DX repository is located at ftp.tc.cornell.edu in directory pub/Data.Explorer.

Acknowledgements

Many thanks to Dr. Bruce Land for his contributions, especially in the domain of education. We are also grateful to Jun Liang and Jon Clardy for the use of their molecular data. The Cornell Theory Center receives major funding from the National Science Foundation and New York State, with additional support from the Advanced Research Projects Agency, the National Center for Research Resources at the National Institutes of Health, IBM Corporation and members of the Corporate Research Institute.

References

1. Gillilan and B. Land. "Scientific Visualization of Chemical Systems." *Proc. IEEE Supercomputing '93*, pp.296-301, Los Alamitos, CA, 1993.
2. B. Lucas, G. D. Abram, N. S. Collins, D. A. Epstein, D. L. Gresh, and K. P. McAuliffe. "An architecture for a scientific visualization system." *Proc. IEEE Visualization '92*, pp.107-113, Los Alamitos, CA, 1992.
3. R. Haber, B. Lucas, and N. Collins. "A Data Model for Scientific Visualization with Provisions for Regular and Irregular Grids." *Proc. IEEE Visualization '91*, pp.298-305, Los Alamitos, CA, 1991.
4. A. Varshney, Jr. F. P. Brooks, and W. V. Wright. "Linearly Scalable Computation of Smooth Molecular Surfaces." *IEEE Comp. Graph. Appl.*, 14:19-25, 1994.
5. A. Varshney and F. P. Brooks. "Fast Analytical Computation for Richard's Smooth Molecular Surface." *Proc. IEEE Visualization '91*, pp.300-307, San Jose, CA, 1993.
6. D., Badouel. "An Efficient Ray-Polygon Intersection." *In Andrew S. Glassner, editor, Graphics Gems*, pp.390-393. Academic Press Inc., San Diego, 1990.
7. C. Cruz-Neira, J. Leigh, C. Barnes, S. M. Cohen, S. Das, R. Engelmann, R. Hudson, M. Papka, T. Roy, L. Siegel, C. Vasilakis, T. A. DeFanti, and D. J. Sandin. "Scientists in wonderland: A Report on Visualization Applications in the Cave Virtual Reality Environment." *IEEE 1993 Symposium on Research Frontiers in Virtual Reality*, pp.59-66, San Jose, 1993.
8. C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. V. Kenyon, and J. C. Hart. "The Cave: Audio Visual Experience Automatic Virtual Environment." *Communications of the ACM*, 35:65-72, 1992.
9. B. Land. "Teaching Computer Graphics and Scientific Visualization Using the Dataflow, Block Diagram Language Data Explorer." *University Education Uses of Visualization in Scientific Computing*, pp.33-36, New York, 1994.

Richard E. Gillilan
Cornell Theory Center Engineering & Theory Center Building
Ithaca, NY 14853-3801
Tel: 607-254-8757
Fax: 607-254-8888
Email: richard@tc.cornell.edu

Frank Wood
Cornell Theory Center Engineering & Theory Center Building
Ithaca, NY 14853-3801
Tel: 607-254-8345
Fax: 607-254-8888
Email: fwood@tc.cornell.edu