Visualization of sphere packs using a dataflow toolkit

Jeremy Walton

The Numerical Algorithms Group Ltd., Oxford, UK

We describe the construction of a simple application for the visualization of sphere packs, with applications to molecular graphics. Our development environment is IRIS Explorer, one of the new generation of so-called dataflow toolkits. *We emphasize particularly the way in which working in such an environment facilitates the design and construction process, paying special attention to tools which aid the importing of data into the application, the design of the user interface, and the extension or modification of existing tools. Some examples of the use of the application in the field of molecular modeling are presented.*

Keywords: dataflow toolkits, IRIS Explorer, molecular modeling, spheres, visualization

INTRODUCTION

Computer graphics has been used for some time to display configurations of spheres. Typical areas in which this has proved helpful include molecular graphics,[1] the visualization of arrays of hard spheres in the modeling of liquid structure,[2] and volume visualization.[3] However, the main area of interest has traditionally been molecular modeling; indeed, the importance of this field has directly inspired many manufacturers of hardware and software to introduce support for sphere primitives in their display environments. In the molecular modeling field, there currently exist many sophisticated commercial software packages which have a visualization component, or front end. Examples of these molecular display programs include InSight[4] and Quanta.[5]

Many users, however, often find that such off-the-shelf programs are inappropriate or too expensive for their needs, and develop their own display software in-house. The development of in-house software can be advantageous for reasons of flexibility, portability, and economics. Flexibility is important because of the wide variety of sources and formats for the data to be displayed—thus, the configuration could be produced as output from a user's own molecular simulation program, or from an academic package such as GROMOS[6] or MOPAC,[7] or from a commercial routine like Discover[4] or CharmM.[5] Under such circumstances, it is important to have as much control as possible over input to the display program, with the emphasis being on its simplicity.

Traditionally, much development of in-house visualization software has relied on the use of *graphics libraries* to provide the display. Such an environment can be unsatisfactory for many reasons (some of which are outlined below). By contrast, this paper describes the recent construction of an application for the display of sphere packs using a comparatively novel *dataflow toolkit* called IRIS Explorer.

Our paper is arranged as follows. The following section describes the use of a dataflow toolkit, and contrasts it to the way in which graphics libraries are employed in application building. An important, if not indispensable, element of the visualization process (regardless of which environment is employed) is getting the data into the application to start with. The next section presents a simple format for sphere configuration files, and details the construction of a data reader for them. The reader is incorporated into the application in the fourth section, while the fifth section shows how its interface may be tidied up before being passed to another user. Some examples of the use of the application, drawn from the molecular modeling arena, are presented in the following section, and we conclude with some final remarks on future directions in which this application could be extended, and how our environment of choice could assist us in this.

GRAPHICS LIBRARIES AND DATAFLOW TOOLKITS FOR VISUALIZATION

The use of graphics libraries has a history which is almost as long as that of computer graphics itself. In this scenario, a user would write, or modify, a program written in a high-level language such as C or FORTRAN. The program would calculate or read in the configuration and possibly process or filter it in some fashion before making calls to a selection of routines from the library to produce the image on a display device. Examples of graphics libraries (some of which, as noted above, have fairly high-level support for the rendering of spheres) include PHIGS +, Silicon Graphics' GL (or its successor, OpenGL) and Hewlett-Packard's Starbase.

In this way of working, the libraries provide a direct interface to the display device, or can provide graphics functionality in software. The user, on the other hand, is obliged to provide the remainder of the pieces of the pro-

Address reprint requests to Dr. Walton at The Numerical Algorithms Group Ltd., Wilkinson House, Jordan Hill Road, Oxford OX2 8DR, UK.
Received 23 November 1993; revised 24 December 1993; accepted 4 January 1994

gram—the input routine, a memory management system, the computational kernel, the data filtering utilities, and the user interface. Although instructive, this can be time-consuming, and leads to a multiplication of effort as code is duplicated.

At a more fundamental level, however, it is worth noting the iterative, or cyclical nature of the visualization process—the "best" picture is very rarely produced at the first attempt.[8,9] This is particularly true in the case where the subject is a complicated three-dimensional object such as a sphere pack. There are a series of adjustable parameters in the display process such as orientation, lighting and color which the end-user will typically want to play with before they are satisfied that they have obtained the "best-looking" image of their data. A good display program must make all these facilities available to the end-user, which in turn means that a lot of time and effort must be put into designing and constructing the user interface.

For these and other reasons,[8,9] dissatisfaction with the traditional method of writing visualization applications has recently led to the employment of the so-called *dataflow model* in the development of a variety of *desktop toolkits.* In this approach, the user interactively creates the application in the form of a network of modules. Each module is a software routine that operates on its input data and produces some output. The network controls the way in which data flows between modules, i.e., how data appearing on the output port from one module is connected to the input port of another. Typical networks will only contain a small number of modules, because each is designed to perform a substantial amount of data processing.

The creation of the network is performed *interactively* using a point-and-click interface where the user first selects the modules that are to compose the network from a palette and drags them onto a work area. The user then makes or breaks connections between the modules using the same type of interface. Each module usually has a set of parameters which control some aspect of its behavior. These are the "control knobs" for the module (and so, for the whole application), and the user is able to alter these while the application is running via a familiar set of widgets (dials, buttons, sliders, file browsers, etc.). Examples of dataflow toolkits which are currently available include Khoros,[10] apE,[11] AVS,[12] Data Explorer,[13] and IRIS Explorer.[14]

An important feature of the module toolkit is that it is extensible: users have the facility to create their own, supplementary, modules which can be added to the default set. This is especially important when it comes to getting the data into the network to begin with. A large number of modules are usually shipped with the default set which read data written in various popular formats (an example from the molecular modeling area is the Brookhaven Protein Database, or PDB format), but it is more or less unlikely, depending on one's degree of optimism, that a reader will exist for every format required by the user. (Where the user has devised their own data format, it is of course inevitable that a reader module will have to be created.)

In the special case of IRIS Explorer,[14,15] an interactive tool called DataScribe is provided which facilitates the creation of a module for importing data into the network. Another tool, the Module Builder,[14,15] assists in the more general task of transforming a user's routine (as C, C++, or FORTRAN 77 sources, or even as a pure executable) into a module. Very recent work by Casher et al.[16] has exploited this latter aspect of the toolkit to produce EyeChem, a new suite of IRIS Explorer modules for molecular visualization. EyeChem 1.0 includes readers for molecular formats such as MOPAC, Gaussian 92, and Sybyl, together with a number of useful geometric display modules. The following section illustrates the use of DataScribe by showing how the reader module for the present application was constructed.

CONSTRUCTING A DATA READER

The design of a module to read in data of a specific format starts with a consideration of the input format itself. In the present example, this had been previously specified by existing user-written applications—doubtless, a very common situation. To be specific, the structure of the sphere configuration file is:

$$\begin{array}{lllll}
NA & & & & \\
X(1) & Y(1) & Z(1) & R(1) & L(1) \\
X(2) & Y(2) & Z(2) & R(2) & L(2) \\
 & & \vdots & & \\
X(NA) & Y(NA) & Z(NA) & R(NA) & L(NA)
\end{array}$$

Here, NA is the number of spheres to be read, while $X(I)$, $Y(I)$, and $Z(I)$ are the coordinates of sphere number $I.R(I)$ is its radius (measured in the same units as its position), and $L(I)$ is a label which is used to assign a color to the sphere when it is displayed. Although this appears to be sensible, it is not intended to be the definitive format for this type of data, even if such a thing existed. However, it serves to illustrate the problems of data formats which are incompatible with those used by the display program, and the way in which such problems may be overcome in a modern visualization environment.

The second thing to consider in the design of a data reader is the format used by the application, i.e., that into which the data is to be translated. IRIS Explorer has six built-in data types,[14] of which the most widely used is the *lattice.* This is essentially a multidimensional array with two parts: *data,* the physical variables stored at the nodes of the lattice, and *coordinates,* the way in which the computational space is mapped to physical space.

The coordinates part gives rise to the three basic types of lattice: uniform (an orthogonal grid with constant spacing, specified by its extent in each direction), perimeter (an orthogonal grid with variable spacing, specified by vectors of coordinate locations in each direction), and curvilinear (a nonorthogonal grid with variable spacing, specified by the coordinate values for each node). The structure of a lattice is specified[17] in terms of the following components:

- *nDim,* number of spatial dimensions.
- *dims,* vector containing the number of nodes in each dimension.
- *nDataVar,* number of data variables at each node.
- *primType,* primitive variable type (i.e., byte, short, long, float, etc.).
- *coordType,* type of physical mapping (i.e., uniform, perimeter, or curvilinear).
- *nCoordVar,* number of coordinate variables at each node.

Our configuration of spheres can be viewed as a list of *NA* coordinates in three-dimensional space, with two values (radius and label) stored at each coordinate. Since the coordinates are not, in general, regularly positioned in space, an appropriate internal format for the configuration would seem to be a one-dimensional curvilinear lattice (*nDim* = 1, *dims* = *NA*, *coordType* = *curvilinear*), with the radius and label for each sphere at each node (*nDataVar* = 2, *nCoordVar* = 3). As it turns out, translating the configuration into this format has another advantage: there exists a module which produces a set of spheres (in IRIS Explorer's geometry data type) from a configuration given as input in precisely this format. That this module already exists could be viewed as either serendipitous (i.e., confirming that our choice for the internal format is a sensible one), or as the only necessary justification for using this format to begin with.

The DataScribe is an interactive data conversion utility for scalars and arrays which can be used to convert data from an external source (such as a file, or existing application) into an IRIS Explorer lattice.[18] The user builds a *script* in DataScribe which consists of input and output *templates*. Each template describes the abstract structure of a file, internal constant, parameter or lattice, built up in terms of icons (or *glyphs*) which give a visual representation of each data type. Finally, the user defines the data conversion to be done in the script by selecting individual data components (contained within the glyphs) from the input template and connecting them to selected components in the output template, using the same point-and-click interface as in IRIS Explorer itself.

Figure 1a shows the DataScribe window containing the script designed to read our sphere configuration files and to convert the data into an IRIS Explorer curvilinear lattice. The upper part of the window shows the input and output templates on the left and right sides; the lower part shows the way in which they have been connected together. The first template on the left (named QMfile, see Figure 1B) describes the structure of the file: a scalar glyph of type *long* named *NA*, followed by a two-dimensional array glyph of type *float* named *spheres*. The dimensions of the array are given as 5 by *NA*, according to the symbol at the left of the glyph—note that the array is sized when the *NA* variable is referenced at run-time earlier in the traversal of the script.

This suffices to read in the full contents of the data file, but a little more work must be done to separate out the data part (radii and labels) from the coordinate part so that they can be stored in the appropriate places in the lattice. We do not describe here how this is done, apart from noting that DataScribe has a powerful mechanism for selecting portions of arrays to facilitate this separation.[18]

The template on the right (named *Configuration*, see Figure 1c) defines the output lattice. It contains just one glyph: that for the one-dimensional curvilinear lattice. The correspondence between the appearance of this glyph and the components of the IRIS Explorer lattice described above can clearly be seen. Values have been entered for some of these components (*nDim*, *nDataVar*, and *nCoordVar*), and the remaining parts of the lattice (*dims*, *data*, and *coord*) come from the connection to the appropriate parts of the input template. For technical reasons (basically, that *dims* is a vector while *NA* is a scalar) the connection to *dims* has to

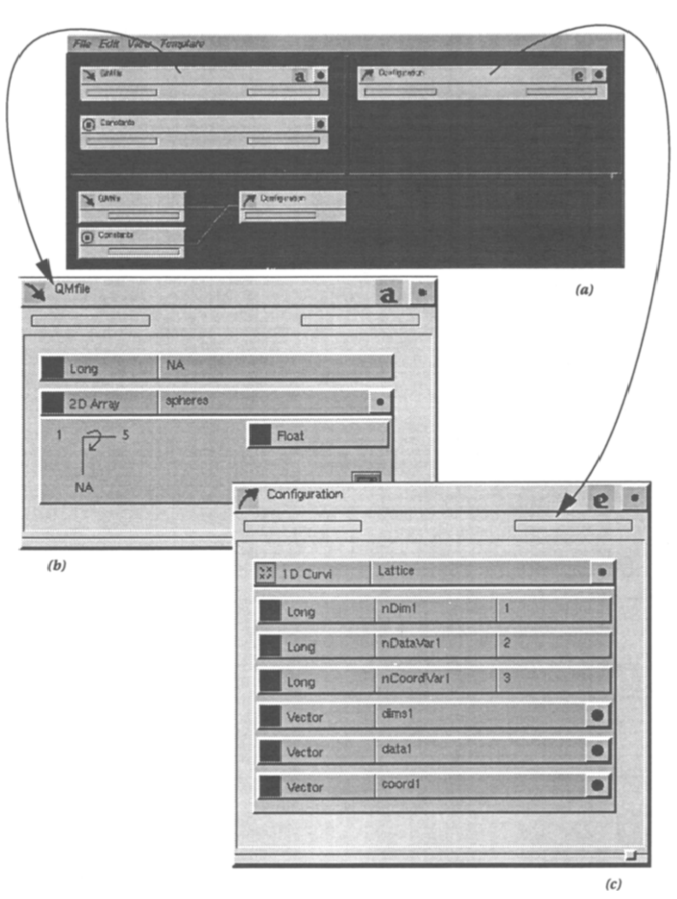

Figure 1. Using DataScribe to construct a reader for the sphere configuration files. (a) The DataScribe window, with input and output templates on the upper left, and the output template on the upper right. Templates and glyphs in DataScribe have an open *and* closed *form;[16] the templates are in the closed form here. The template interconnections are shown in the lower pane. (b) Open form of the input file template, describing the structure of the file. (c) Open form of the output template, defining the structure of the lattice.*

be made using the set glyph from within a constants template.[18]

Having built the script, the user can interactively design the control panel for the module (in the present case, this just amounts to selecting a suitable widget—text typein or file browser for the input file parameter) and then begin to use it in an application. That for the visualization of sphere packs within IRIS Explorer is described in the following section.

CONSTRUCTING A NETWORK

As noted above, our application has been built around a preexisting IRIS Explorer module, called *Ball*, which produces a set of spheres from a configuration given as input in the form of a curvilinear lattice. *Ball* is part of the default library of modules which ships with the toolkit, although an earlier version was available from a public domain repository for contributed modules. The source code (which is supplied along with the executable) is written in C, with an internal structure which is reasonably straightforward. This is due to the rather high level of abstraction exhibited by the IRIS Explorer API (for example, it contains a function *cxGeoSpheresDefine* which defines a set of spheres from their coordinates and radii). Thus, even if Ball hadn't already existed, it would have been fairly easy for a user to create.

Figure 2 shows the work area (or so-called *map editor*) in IRIS Explorer where modules are connected together to produce applications. The structure of our application can be seen to be very simple; at the top left is our data reader (created using the DataScribe, as previously noted), whose output goes to *Ball.* The other input to *Ball* is a color map, which it uses to color each sphere according to the value of its label. Color maps are lattices in IRIS Explorer; ours is created using the *LatFunction* module and passed via *GenerateColormap* (which provides a convenient visual reference for the color map) to *Ball.* Finally, the output from *Ball* is passed to the *Render* module for display.

The advantages of using this type of toolkit are already apparent—having constructed a module to read in the data, we are able directly to visualize it with little extra work. Moreover, since the *Render* module has a rich set of built-in functionality for changing aspects of the drawing style (e.g., switching between a shaded and wireframe representation), as well as the more general manipulation, lighting, and editing of three-dimensional objects, we are immediately able to interact with the image in a physically meaningful way. We return to this point below.

The development environment we have been using up to this point, while powerful and flexible, is not necessarily the most appropriate from the end-user's perspective. Faced with a complicated network of modules, each with their own parameters, the end-user may be unable to identify the significant or useful control parameters for the application, or may be tempted to modify the network, with possibly disastrous results. So, having established the form and content of our network, the final step in the creation of our application involves hiding the development environment, if required. This is the subject of the following section.

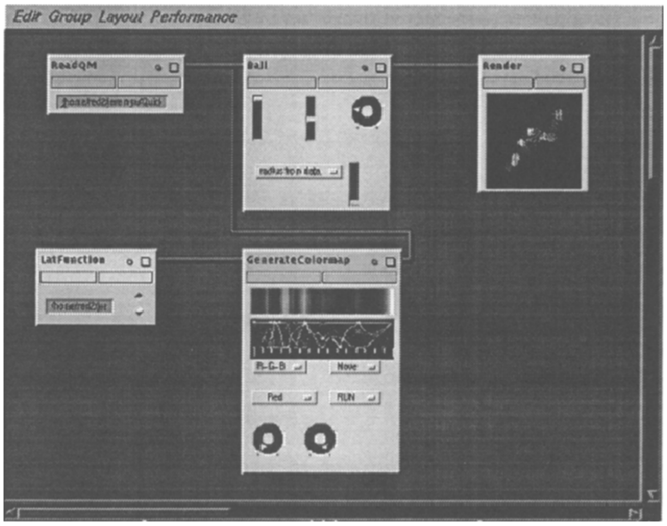

Figure 2. The map editor in IRIS Explorer. Modules are selected from module librarian (not shown) and dragged onto the map editor where they are connected together to form the application. The widgets which control the modules actually appear on the representation of the modules themselves (a smaller and a larger representation is available).

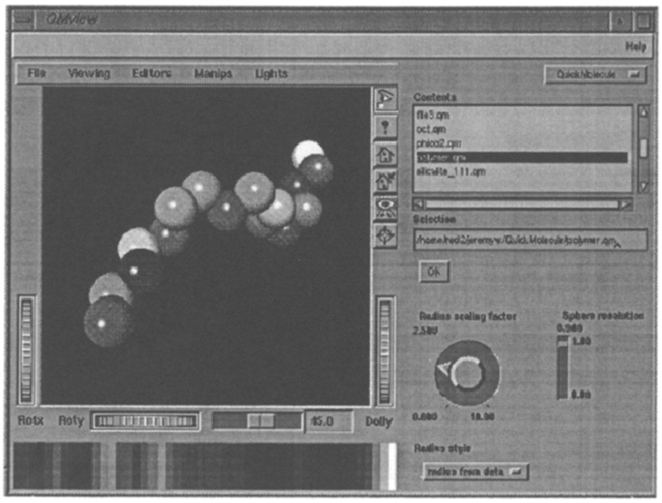

Figure 3. Grouped form of our application. The file browser widget from the reader module is at top right, while three useful widgets from the Ball *module are collected together at bottom right. The key from* GenerateColormap *has been placed under the display window from the* Render *module. Note the thumbwheels and buttons around the window, which facilitate control over the orientation of the object; other options are available from the menus above the window.*

CONSTRUCTING AN APPLICATION

IRIS Explorer provides a mechanism for *grouping* related modules together into (what appears to the end-user as) a single module, with a single control panel.[18] The Group Editor is then used to select the ports and parameters which are to appear at the group level. Since most of the controls and all of the ports can usually be hidden, the result is an application with a simpler interface. Once again, the control panel for this can be interactively designed, with widgets being selected and positioned using a drag-and-drop interface. In addition, values can be explicitly set for widget limits or initial values, and labels for widgets can be edited, if necessary.

The final form for our application is displayed in Figure 3. We have collected together the most useful parameters from the modules in the network, for example, the input file name from the data reader, the key from *GenerateColormap* and the radius scaling factor from *Ball* (note that some of these widgets have been given more meaningful labels). As a final step, we note that the application can be run in a stand-alone mode, without the appearance of the map editor and other supporting IRIS Explorer windows.

EXAMPLES

In this section, we present a few examples of the use of our application. Figure 4 shows a configuration from a molecular simulation of a fluid of spherical molecules confined to a slit-like pore.[19] The results from this type of simulation have provided a useful starting point for the development of models of adsorption of light gases in graphite pores.

The slit is modeled by two adsorbing walls (not shown) that are in the plane normal to the figure. In this configuration, the density of fluid in the centre of the pore is low, corresponding to a gaslike phase, although it can be seen that there is a region of enhanced density on the walls of the pore. The nature of these adsorbed layers has been rather extensively investigated, and images such as Figure 4 have been very useful in this work.

Figure 5 shows an image of the ethyl acetate molecule. This confirmation has been taken from the output of an energy minimization calculation on the molecule using MOPAC,[7] the semiempirical quantum mechanical package. The viewing facilities made available by IRIS Explorer are particularly helpful in examining conformational changes in small molecules of this type, because of the ease with which the molecule can be rotated and examined from different angles.

Finally, Figure 6 shows the unit cell from a zeolite, which is a microporous crystalline material. Compounds such as this are widely used in separation and catalytic processes because of their characteristic shape-selective properties. These are due to the fact that their internal channels and cages have sizes which are similar to that of a typical small molecule. Molecular modeling methods for predicting the adsorption and diffusion of gases in zeolites have recently become active areas of research,[20] and the display of the complex three-dimensional structure of the zeolite (together

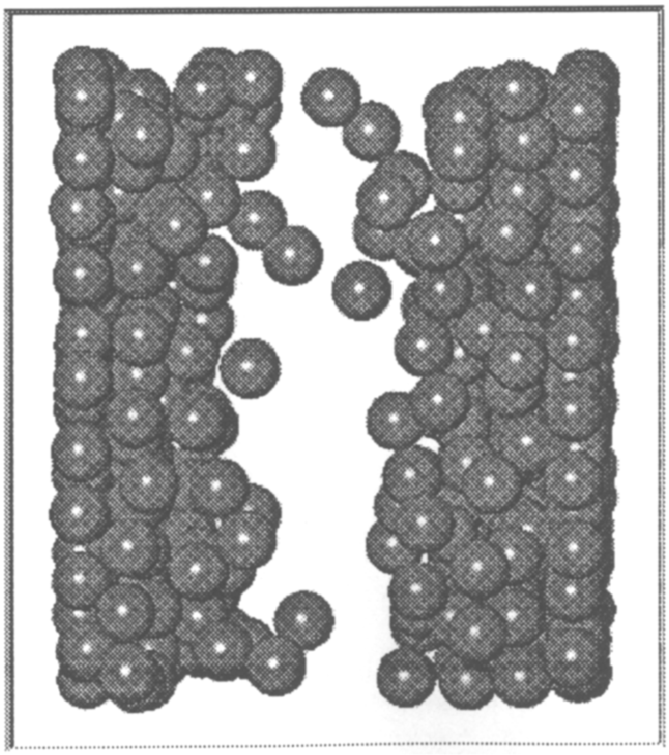
Figure 4. Modeling the behavior of a fluid in a slit using molecular simulation. This snapshot from the simulation clearly shows the layering as the fluid molecules pack against the walls of the pore, and the region of low density in the middle of the pore.

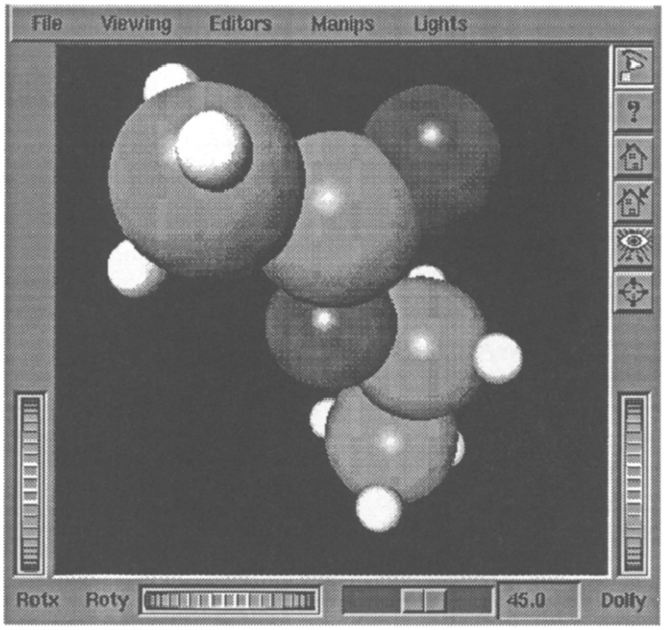

Figure 5. Ethyl acetate, $CH_3COOC_2H_5$. The conformation corresponds to a minimum in the molecule's potential energy, as calculated using MOPAC.

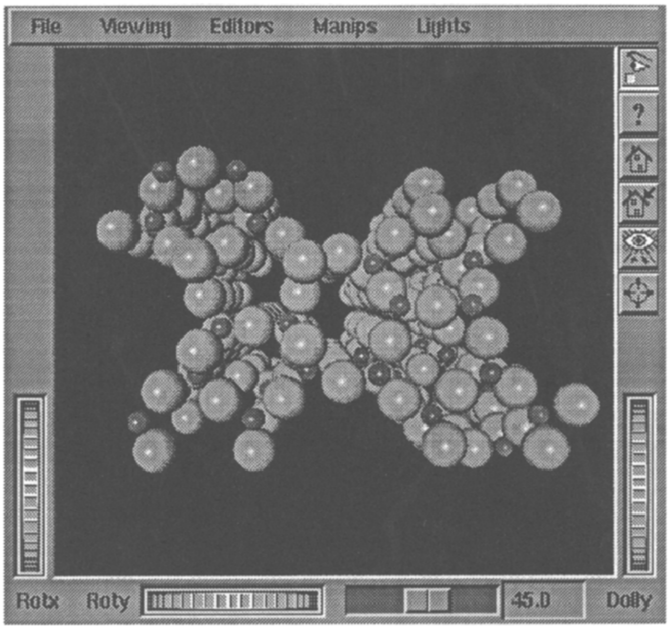

Figure 6. The unit cell of silicalite, one of the microporous crystalline materials known as zeolites. The crystal is composed of silicon and oxygen atoms, here displayed as larger and smaller spheres. Some of the channels which lie parallel to the viewing direction can be seen in this orientation.

with the adsorbate molecules) is of great assistance in this type of study.

CONCLUSIONS AND FURTHER WORK

Our intent in this paper has been to illustrate the use of a toolkit like IRIS Explorer in the construction of a molecular graphics application. As noted above, the advantage of using a dataflow toolkit (as opposed to a graphics library) is the reduced time between getting the data read in and getting it displayed. This is because of all the functionality that is thrown in "for free" with a toolkit, the wheel (or sphere-rendering algorithm, in the present case) doesn't have to be reinvented each time around.

One area in which the toolkit application could be at a disadvantage when compared to a more traditional application is that of run-time performance. It is possible that the overhead associated with starting up the toolkit and passing the data between modules could have an adverse effect on its speed, although the designers of modern toolkits such as IRIS Explorer have put a lot of effort into optimizing these aspects of its design—for example, shared memory (if the hardware architecture supports its use) and named pipes are used to transfer data between modules, which keeps data copying and memory transfers to a minimum.[15] However, even if the performance of the toolkit application is found to be insufficient, the toolkit will still be invaluable in the design and prototyping of the application. Using a toolkit means that a developer can quickly try out a variety of different solutions, techniques, and interface designs (possibly in collaboration with an end-user) before focusing on one of them and, if necessary, reimplementing it in a more traditional development environment.

In the remainder of this section, we shall look at further work which could be done with the present application. It is often illuminating to be able to add other geometric objects, such as surfaces (as dots, in wire frames, or as solid sheets) to the images of sphere packs. For example, Figure 7 shows an isosurface of potential energy in the space between three atoms which are interacting via the pairwise Lennard–Jones potential function.[21] This function, which gives a reasonable account of many of the properties of liquids and gases,[21,22] is characterized by a well depth ϵ and collision diameter σ; the surface in Figure 7 goes through all points in the box where the potential has a value of -0.24ϵ, while the atoms are

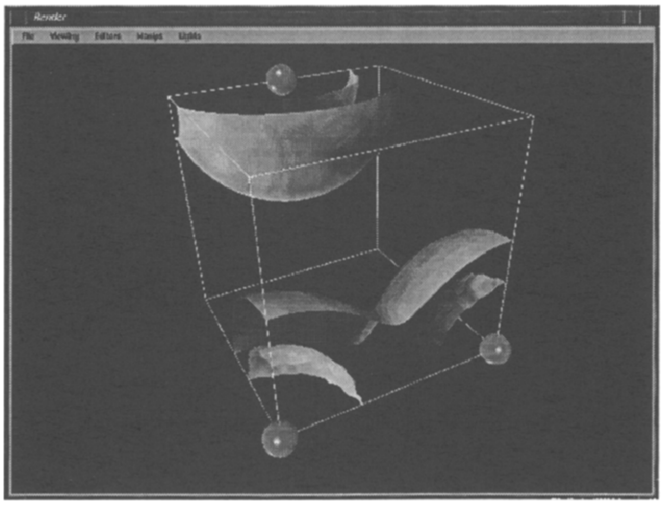

Figure 7. Compositing the representation of spheres with other geometric data—in this case, an isosurface of potential energy in the space between three interacting Lennard–Jones atoms.

represented as spheres of radius 0.2σ. The cubic box is 3σ on each side.

The main point of interest here is that the incorporation of such extra objects into the display is very easy in a toolkit environment. In the present example, all that was required was the addition of two extra modules from the default set to the network of Figure 2: one module (*ReadLat*) to read in the field data (again, as a lattice), and one to calculate the isosurface. Another variation, where *ReadLat* is replaced with a user-written module which calculates the potential from the atoms' configuration, could be imagined and quickly implemented, while an extension to this might see the user being able to enter or modify the location of individual atoms directly via pointing and clicking in the rendered image, and seeing the effect on the shape of the potential field. The latter mode involves the use of a feedback loop between *Render* and the calculation module to pass the information about the selected point in the image upstream in the network—again, this is easily done in a visual programming environment such as IRIS Explorer.

Another way in which the application could be extended is to the display of molecules. To a chemist, molecules are atoms plus bonds; to a visualizer, they are spheres plus lines, or cylinders. It would be simple to add the display of bonds (whether generated from an explicit list to be read in, or from some distance criteria), which would enhance the appearance of displays such as Figure 5, for example.

A more adventurous extension would be away from the display of static configurations, and towards the visualization of a sequence dynamically evolving in time. One example of this is already in EyeChem,[16] which displays animations of normal vibrational modes from a MOPAC FORCE calculation. Another could be based on the molecular simulation technique known as molecular dynamics.[23] This produces trajectories for assemblies of interacting molecules, and has made many valuable contributions to the development of new theories, for example, of liquids and liquid surfaces. In many cases, the animated display of the molecules themselves has been of interest, and it would be fairly straightforward to modify the present application to this end. A further goal would be its use (or, more broadly, the use of our toolkit environment) to modify the controlling parameters of the simulation as it proceeded (the so-called *interactive steering* visualization scenario). This is currently under investigation.

ACKNOWLEDGMENTS

I thank Omer Casher and Henry Rzepa for helpful discussions. IRIS Explorer is a registered trademark of Silicon Graphics, Inc. All other trademarks are acknowledged.

REFERENCES

1 Brodlie, K.W., Carpenter, L.A., Earnshaw, R.A., Gallop, J.R., Hubbold, R.J., Mumford, A.M., Osland, C.D., and Quarendon, P. *Scientific Visualization Techniques and Applications.* Springer-Verlag, Berlin, 1992, 144
2 Finney, J.L. Ph.D. Thesis, University of London, 1968
3 Laur, D. and Hanrahan, P. *Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering.* Proceedings of SIGGRAPH '91. In *Comp. Graphics.* 1991, **25,** 285

4 Biosym Technologies Inc., San Diego, CA, USA
5 MSI, Burlington, MA, USA
6 van Gunsteren, W.F. and Berendsen, H.J.C. *Groningen Molecular Simulation (GROMOS) library manual.* Biomos b.v., Groningen, 1988
7 Stewart, J.J.P., *MOPAC—A General Molecular Orbital Package.* Quantum Chemistry Program Exchange No. 455, QCPE, Indiana
8 Walton, J.P.R.B. Get the picture—new directions in data visualization. In *Animation and Scientific Visualization* (D. Watson and R.A. Earnshaw, Eds.) Academic Press, London, 1993, 29
9 Walton, J.P.R.B. Now you see it—interactive visualization of large datasets. In *Applications of Supercomputers in Engineering III.* (C.A. Brebbia and H. Power, Eds.), Computational Mechanics Publications/Elsevier Applied Science, 1993, 139
10 Rasure, J. and Young, M. *An Open Environment for Image Processing Software Development.* Proceedings of 1992 SPIE/IS&T Symposium on Electronic Imaging, 1992, **1659**
11 Dyer, D.S. *A Dataflow Toolkit for Visualization.* IEEE Computer Graphics and Applications, 1990, **10,** 60
12 Upson, C., Faulhaber Jr., T., Kamins, D., Laidlaw, D., Schlegel, D., Vroom, J., Gurwitz, R., and van Dam, A. *The Application Visualization System: A Computational Environment for Scientific Visualization.* IEEE Computer Graphics and Applications, 1989, **9,** 30
13 Lucas, B., Abram, G.D., Collins, N.S., Epstein, D.A., Gresh, D.L., and McAuliffe, K.P. *An Architecture for a Scientific Visualization System.* Proceedings of Visualization '92, IEEE Computer Society Press, 1992, 107
14 Edwards, G. *The design of a second generation visualization environment.* Proceedings of Computer Graphics 1991, Blenheim Online, 1991, 261
15 *IRIS Explorer 2.0 Technical Report.* Silicon Graphics Computer Systems, 1992
16 Casher, O., Green, S.M., and Rzepa, H.S. EyeChem 1.0: A Modular Chemistry Toolkit for Collaborative Molecular Visualization. *J. Mol. Graphics* 1994, **12,** 226
17 *IRIS Explorer Module Writer's Guide.* Silicon Graphics Computer Systems, 1993
18 *IRIS Explorer User's Guide.* Silicon Graphics Computer Systems, 1993
19 Walton, J.P.R.B. and Quirke, N. Capillary Condensation: A Molecular Simulation Study. *Molec. Sim.* 1989, **2,** 361
20 Goodbody, S.J., Watanabe, K., MacGowan, D., Walton, J.P.R.B., and Quirke, N. Molecular Simulation of Methane and Butane in Silicalite. *J. Chem. Soc. Faraday Trans.* 1991, **87,** 1951
21 Maitland, G.C., Rigby, M., Smith, E.B., and Wakeham, W.A. *Intermolecular Forces: their origin and determination,* Oxford University Press, 1981
22 Allen, M.P., and Tildesley, D.J. *Computer Simulation of Liquids,* Oxford University Press, 1987