

# Case Study: An Environment for Understanding Protein Simulations Using Game Graphics

Donna Gresh, Frank Suits, and Yuk Yin Sham\*  
IBM T.J. Watson Research Center

## Abstract

We describe a visualization system designed for interactive study of proteins in the field of computational biology. Our system incorporates multiple, custom, three-dimensional and two-dimensional linked views of the proteins. We take advantage of modern commodity graphics cards, which are typically designed for games rather than scientific visualization applications, to provide instantaneous linking between views and three-dimensional interactivity on standard personal computers. Furthermore, we anticipate the usefulness of game techniques such as bump maps and skinning for scientific applications.

**CR Categories:** I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques

**Keywords:** visualization, proteins, computational biology, molecular modeling, molecular dynamics, game graphics, DirectX

## 1 Introduction

Proteins are the machinery of life. They are, in the simplest terms, a linear sequence of amino acids. Proteins have the interesting property of self-organization, on a very short time scale, into a particular “native,” or folded configuration. In fact, a protein’s function depends ultimately on its 3-dimensional structure in space with particularly defined motifs and regions.

To date, most protein configurations have been determined by experimental measurements, such as x-ray crystallography, electron crystallography, or NMR techniques. However, these processes require that the protein first be synthesized and isolated in measurable quantities. The visualization environment we have developed is intended to facilitate the study of proteins for researchers in the field of computational biology, where the motion and behavior of proteins and other molecules are studied in the computer rather than in the test tube. As computational power and algorithmic sophistication increases, molecular dynamic simulations of biological systems such as proteins have become an increasingly active area of research. The goal of our system is to provide an environment where the results of these simulations can be interactively studied using a variety of two-dimensional and three-dimensional views. Key to the value of the visualization environment is fast, interactive linking of views enabled by our use of modern commodity graphics technology. Though the computational demands of these simulations are often extreme, the visualization requirements are relatively modest, and the system we have developed runs at interactive speed on standard desktop and laptop personal computers.

---

\*gresh, suits, shamy@us.ibm.com, IBM T.J. Watson Research Center, P.O. Box 704 Yorktown Heights, NY 10598

## 2 Current Trends in Molecular Dynamics Visualization

Molecular visualization has been a focus of the graphics community for many years, and now a large number of high performance viewing environments are available commercially or freely downloadable from the web[9, 12, 4, 10]. These environments comprise two main types, each with its own design requirements: standalone viewers of fixed molecular structures, and visualization front-ends integrated with a molecular dynamics simulation program. In addition, many viewers allow output of the molecular structure in a format compatible with ray-tracing packages such as POV-Ray[11], allowing non-interactive but photorealistic renderings suitable for publication. Since each viewer has its own requirements for appearance and speed, it must strike a balance between rendering quality and interaction performance based on the needs of the application.

Another factor that drives the design of a molecular visualization system is the intended viewing platform, which may range from a high-end graphics workstation to a web browser running on a laptop. Commercial packages intended for the high-end platforms can assume interactive performance with large amounts of geometry, while simple viewers intended for web browsers must be lightweight and able to create compelling results without too much computation or geometry. Furthermore, each platform may have a variety of application programming interfaces (API’s) for three-dimensional programming, such as OpenGL<sup>†</sup>, Java3D<sup>†</sup>, and DirectX<sup>†</sup>. The key factors that determine the API choice are portability among the target platforms and the ability to make optimal use of the graphics capabilities on the target platforms. Viewers intended for the web make additional demands based on functionality within a number of different browsers, and the need for a plug-in.

The standard three- and two-dimensional views of molecules map well to the functionality provided by graphics API’s since the geometry consists of well-structured triangle strips (for ribbons), repeated glyphs (for ball and stick), or triangle manifolds (for molecular surfaces). However, there are many new features available on graphics cards that are either not supported in OpenGL, or are available only as vendor-specific hardware extensions. As a result, molecular visualization software tends to use only a subset of the capabilities of the graphics hardware; otherwise it would need specific code to deal with a variety of hardware possibilities.

## 3 The Molecular Visualization System

Our system provides a variety of both two-dimensional and three-dimensional views of proteins, all linked dynamically and instantaneously. It is programmed in C++, and uses DirectX<sup>†</sup> to gain direct access to the graphics hardware without the need to write hardware-dependent code. We use splitter windows to build the application from its individual views to maximize screen use and to allow the user to easily expand a view at the expense of other views to focus on a particular area of interest. The application was built using the Microsoft Visual C++<sup>†</sup> development environment and the Microsoft Foundation Class (MFC) library.

At IEEE Visualization 2000[7], there was a growing consensus that games are driving graphics card features, and the way to access those features is via DirectX. We were interested in experimenting with novel ways to use these game features in a scientific visualization application. The rationale and some of the implications of this design decision are discussed in Section 3.4. While our visualization application was developed specifically to study proteins, the viewing infrastructure is generic, and could be easily applied to a variety of visualization applications.

Figure 1 shows the application with a particular protein, green fluorescent protein, 1eme from the Protein Data Bank[2], loaded. Below we will discuss the various views our system provides; however first it is helpful to define a few of the terms common in protein studies. A protein is, fundamentally, a large molecule, consisting of on the order of a few thousand to many thousand atoms, and a few hundred to a few thousand amino acid residues, joined by peptide bonds. While there are approximately 20 different amino acids, each has in common a central carbon atom ( $C_\alpha$ ), to which is attached a hydrogen atom, an amino group ( $NH_2$ ), and a carboxyl group ( $COOH$ ). Each peptide in the chain is essentially a planar unit, that has two degrees of rotational freedom about the *backbone* of the protein, defined by the sequence of  $C_\alpha$  atoms. These two degrees of freedom are described by the angles  $\phi$  and  $\psi$ . A protein is also often described in terms of its primary, secondary and tertiary structure. The primary structure is simply the sequence of amino acid residues. The secondary structure is the description of particular motifs into which the protein locally arranges itself. Examples of secondary structure are  $\alpha$ -helices and  $\beta$ -sheets, which are coils and aligned runs of the backbone, respectively. Tertiary structure is the folding of distant parts of the sequence into associated structures.

### 3.1 Three-Dimensional Views

A protein is a three-dimensional object, and its shape is critical to its functional behavior. Accordingly, an interactive three-dimensional view of the protein is critical to any visualization environment. Since a protein is, in simple terms, just a large molecule, one might expect a ball-and-stick representation to be typical. While we provide such a view as an option, it is more common to display proteins as ribbons[6].  $\alpha$ -helices and  $\beta$ -sheets are usually shown as coiled and flat ribbons, respectively, which follow the backbone of the protein. We use a variation on an algorithm described by [3] to create the three-dimensional geometry of the ribbon model from the atom positions. This algorithm computes the backbone of the protein, following  $C_\alpha$  atoms, and orients the ribbon such that the ribbon normal is always perpendicular to the plane of the peptides. We represent the ribbon as either a flat sheet or as a thickened slab. In addition, we add arrows indicating the ends of the structures with respect to the beginning of the protein chain.

We can color the ribbon using a variety of color maps, including a structure-based map, in which helices, loops, and sheets are each colored differently, and a residue-based map in which each residue is assigned a color using the Shapely scheme (see Figure 2). The left portion of Figure 1 shows our ribbon view, colored by secondary structure type.

### 3.2 Two-Dimensional Views

In addition to the three-dimensional views of the protein described above, we also present three complementary two-dimensional views.

The first such view is a distance matrix showing the distance between the  $C_\alpha$  atoms for all pairs of residues. Points near the diagonal thus represent distances between adjacent residues along the protein backbone. We color the distance matrix using a gray scale

color map, which is particularly suited to seeing both the overall distance pattern and high frequency fluctuations in distance. High frequency patterns are often associated with  $\alpha$ -helices due to their coiled shape. We use the upper portion of the matrix to show, additionally, information on “backbone contacts” of the protein; that is, pairs of residues for which the backbone nitrogen to oxygen distance is less than 4 Å. Pairs of  $\alpha$ -helices are marked by blue points and pairs of  $\beta$ -sheets by reddish brown points (these colors match those of the secondary structure colors of the three-dimensional plot). Pairs of mixed  $\alpha$ -helices and  $\beta$ -sheets are shown by green points, while pairs that include at least one residue that is part of neither a helix nor a sheet are colored dark gray.

The contact points indicate secondary and tertiary structure. A line of blue points close to the diagonal represents an  $\alpha$ -helix, while a line of red points parallel to the diagonal represents a *parallel*  $\beta$ -sheet (in which the strands of the  $\beta$  sheet are pointing in the same direction). A line of red points orthogonal to the diagonal represents an *antiparallel*  $\beta$ -sheet (in which the strands are pointing in opposite directions). Colored points further away from the diagonal point to tertiary structure (regions of the protein separated in sequence but close in distance). In the case of proteins for which the secondary structure is not well characterized, one can use the contact plot to directly infer the secondary structure.

We chose the colors in the matrix view to be, first, natural, in the sense of matching the use of color in the three-dimensional plot, and second, easily discriminable. Gray is used for the pairs that include at least one non-helix or non-sheet residue, to downplay their prominence, since most tertiary structure arises from helices and sheets. We present a larger version of this matrix view in Figure 3 so that the colors, patterns, and high frequency behavior can be seen more easily.

We use the lower diagonal of the distance matrix view to show “sidechain contacts,” or residues for which the distance between sidechain atoms is less than 4.5 Å. We encode the number of sidechain contacts using color; for example, residue pairs with more than five atom contacts are colored red. This allows the user to see residues which have significant sidechain interaction. Figure 3 illustrates that the sidechain contacts can differ significantly from the backbone contacts. The user has the option to turn off either or both of the contact plot overlays.

The second two-dimensional view is a Ramachandran plot[5], which displays a two-dimensional histogram of the  $\phi$  and  $\psi$  angles of the protein. It is standard to plot the angles from  $-180^\circ$  to  $180^\circ$  in each dimension, with the result that regions of the two-dimensional Ramachandran plot are associated with particular types of secondary structure: the upper left region with  $\beta$ -sheets, and the central left region with  $\alpha$ -helices. We use a blue to yellow color map to indicate the number of residues whose  $(\phi, \psi)$  values fall within each bin of the histogram. Black represents bins into which no residues fall. This view can be seen in the central right portion of Figure 1.

The third two-dimensional view is a plot of  $\phi$  and  $\psi$  as a function of residue number. Runs of similar  $\phi$  and  $\psi$  are strongly indicative of particular secondary structure motifs. This view is shown on the bottom of Figure 1. We use a special presentation of the  $\phi$  and  $\psi$  angles to avoid breaking the  $\beta$ -sheet region across the angle  $\psi=180^\circ$  to  $-180^\circ$ [1]. Thus the plots on the bottom of Figure 1 start at  $0^\circ$ , continue to  $180^\circ$ , then continue from  $-180^\circ$  to  $0^\circ$ . Colors are consistent with the secondary structure colors of the three-dimensional view.

### 3.3 Interaction and Linkage

The three-dimensional view of the protein is completely interactive, with mouse-driven rotation, panning, and zooming. In addition, the plot of  $\phi$  and  $\psi$  vs. residue allows a magnifying “lens” to be dragged over the plot to highlight any desired region. Most importantly, all of the views of the protein are linked, so that when-

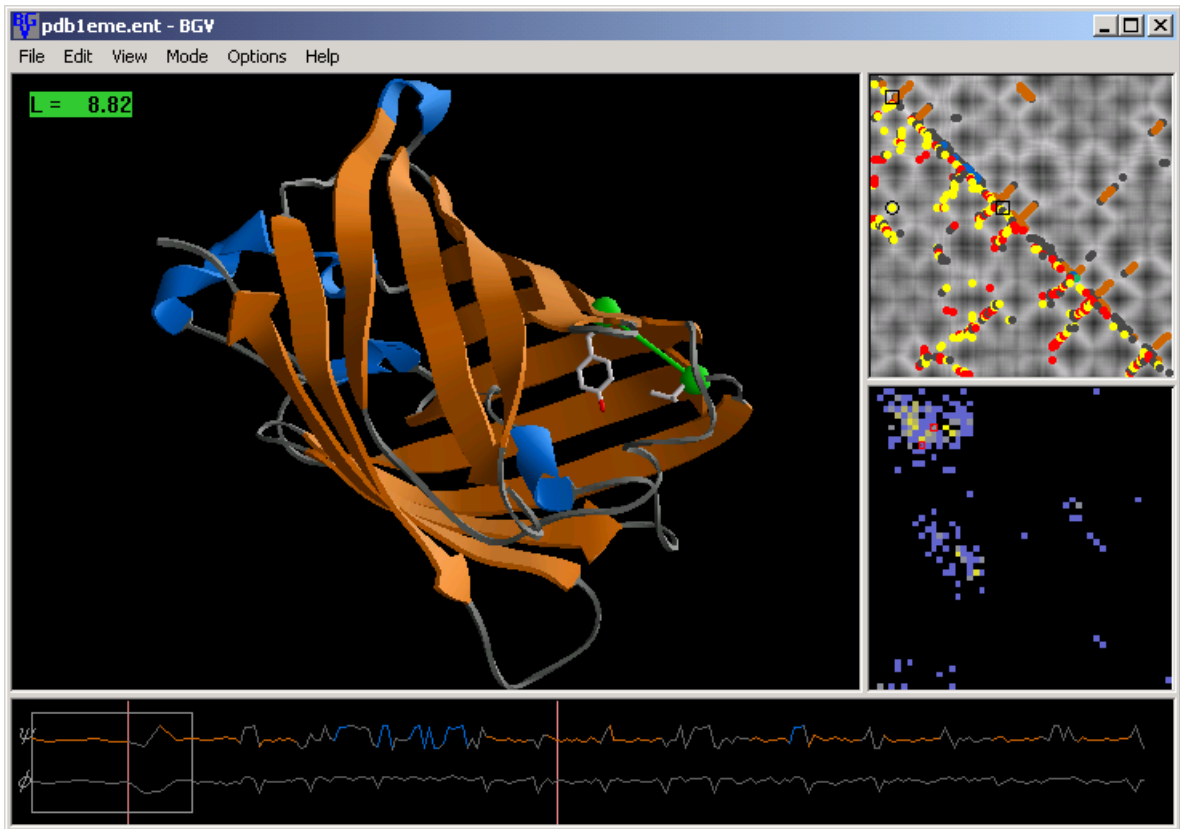


Figure 1: The protein visualization application we have developed. The three-dimensional view displays the protein 1eme from the Protein Data Bank[2] as a ribbon model, colored by secondary structure type. Top right view shows a distance matrix (described more fully in the caption of Figure 3). The bottom right view shows a two-dimensional histogram of  $\phi$  and  $\psi$ , while the plot along the bottom displays  $\phi$  (below) and  $\psi$  (above), with a “lens” expanding the view of a particular stretch of residues. This figure captures a pick of a in the distance correlation plot, which has selected two residues in each view.

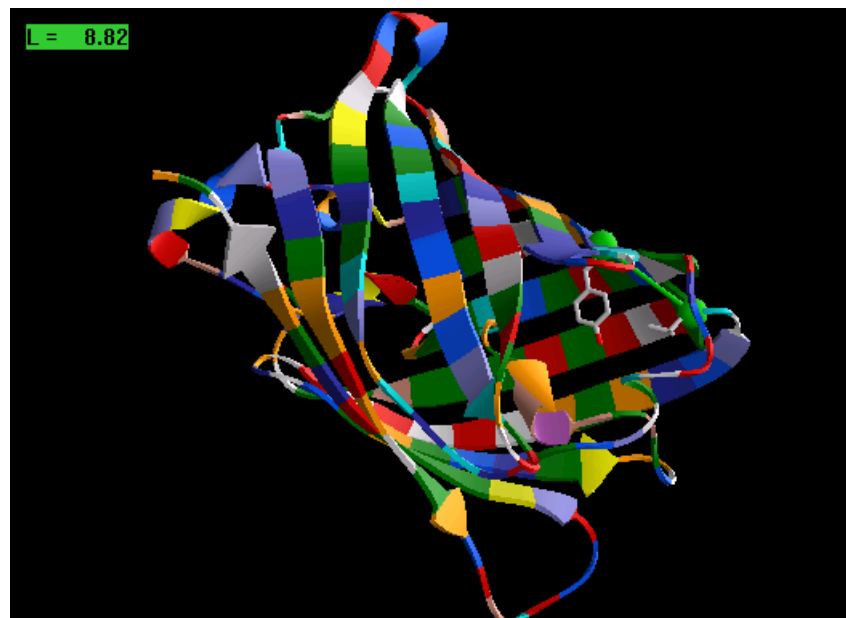


Figure 2: Ribbon diagram of Figure 1, coloring residues individually using the Shapely color scheme. This (as well as the structure-based coloring) is implemented using a one-dimensional texture map.

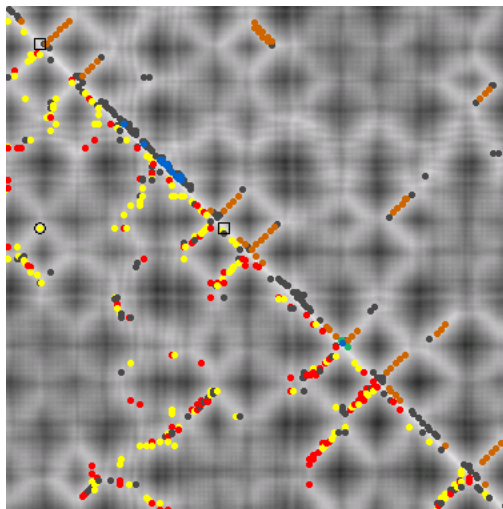


Figure 3: An enlarged view of the distance matrix plot shown in Figure 1. We use a gray scale color map throughout to highlight high-frequency differences in distances, typically indicating  $\alpha$ -helical areas. Above the diagonal we additionally encode the secondary structure type whenever two residues have a backbone to backbone distance of less than 4 Å. Blue indicates a pair of  $\alpha$ -helices, reddish-brown a pair of  $\beta$ -sheets, and green a pair of mixed  $\alpha$ -helix and  $\beta$ -sheet. Dark gray is used to mark a pair that includes at least one residue which is part of neither a helix nor a sheet. Below the diagonal, marks indicate residues with sidechain to sidechain contacts of less than 4.5 Å; red indicates more than 5 contacts, yellow between 2 and 4 contacts, and gray 1 contact. A yellow sidechain contact pair has been picked (left side).

ever the mouse simply pauses at a particular point in the frame, all views are updated, with no mouse clicks required. The linkage is contextually appropriate, depending on the plot the mouse is residing in and its location in the plot. Thus, for example, if one dwells in the three-dimensional ribbon view, an indication of the residue name and number is drawn at the cursor position, and at the same time, the residue is indicated by a vertical line at the appropriate residue in the  $\phi$  and  $\psi$  line plot, by a square around the appropriate histogram bin in the Ramachandran plot, and by an indicator along the diagonal in the distance matrix. Similar behavior results from a dwell in the line plot, or a dwell on the diagonal of the distance matrix. However, since selecting an off-diagonal point in the distance matrix is equivalent to selecting *two* residues, two vertical lines are drawn in the  $\phi$  and  $\psi$  plot, the bin locations of both residues are shown in the Ramachandran plot, and both residues are indicated in the three-dimensional plot, along with a line connecting them and a display of the length of the line, representing the distance between the residues. The user also has the option of “snapping” to a nearby contact. This mitigates the problem of distance plots which may have fewer pixels than residues, allowing picking at subpixel resolution. A dwell in the two-dimensional histogram selects all residues within the selected bin and each residue selected is shown in each of the other plots.

Figure 1 shows the state after a dwell in a region indicating a pair of residues with a significant number of sidechain contacts (a yellow mark in the lower diagonal region). The user has also elected to show the actual sidechains of the picked pair in the three-dimensional view. This picked pair is also indicated in the Ramachandran and the  $\phi$  and  $\psi$  plots.

### 3.4 Use of Game Technology

We chose to use DirectX for our three-dimensional views to explore the capabilities of modern commodity graphics cards and their applicability to scientific visualization. There is much debate on the merits of DirectX vs. OpenGL, and our choice in this application was motivated by a combination of research and utilitarian interests rather than superior performance or preferred API design. The immediate users of this application are scientists who tend to use Windows laptops and desktops (even though their simulations are running on large parallel computers), and writing in DirectX allowed one code-base to work on all their personal computers and take advantage of the available hardware without, for example, having to code to vendor-specific OpenGL extensions required for different graphics cards. While DirectX has the advantage of a common API and feature set across multiple game card platforms under the Windows operating system, we note that DirectX is not available on Linux or Unix systems. For our application, this was not an issue, as the laptop platform was able to deliver acceptable interactivity and performance.

Protein simulations are ideally suited to commodity graphics cards due to the relatively small amount of geometry required and the well-defined structure of the triangles and glyphs. For ball-and-stick views we can store single instances of the needed glyphs (a sphere and a cylinder) in memory and use transforms to create the full molecule; we assign colors on the glyphs with small, one-dimensional texture maps corresponding to the possible atom pairs in each bond. For protein ribbons with under 1000 residues we can use compact one-dimensional textures to color the geometry based on residue-dependent values. For both visualization modes, we create vertex buffers corresponding to the ribbons and glyphs with fixed texture coordinates corresponding to the atom or residue of the ball-and-stick or ribbon geometry, respectively. We can thus create the geometry and dynamically change color maps without rewriting to the vertex buffers; we write the geometry and textures into memory once, and apply different color maps by changing the applied texture, not the texture coordinates. Even when the geometry does change, for example during a ribbon view of a trajectory run (sequence of steps in a simulation), the number of vertices remains constant. Thus the vertex buffer does not need to be recreated; the new vertex positions simply need to be loaded into it. DirectX not only allows us to create these texture maps and vertex buffers, but we can assign their location in memory based on size and access needs: Since many of the textures are small and do not have to be read by the CPU after creation, we can place them directly into local video memory on the graphics card.

One technique common to games is vertex blending or “skinning,” which we have used to create a visualization of the molecular surface, shown in Figure 4. Computation of a molecular surface[8] is relatively expensive, and one efficient visual approximation is to draw large spheres around each atom based on its van der Waals radius. This technique is easy to implement but loses much of the surface shape due to the sphere approximation and lack of coupling between separated atoms (see Figure 5). We have created the first, to our knowledge, approximation of a molecular surface based on hardware vertex blending, a technique commonly used in games to create realistic motion of a smooth surface covering multiple articulated segments. In our implementation, we first find all atom pairs that are near each other (within six Angstroms) and we use 2- or 3-matrix vertex blending of a single sphere to create a stretched surface corresponding to the molecular surface of that pair. (Three-matrix blending allows the surface to have a closer approximation to the characteristic curved shape between the atoms.) Figure 4 shows all such pair surfaces rendered together, and the resulting molecular surface is evident. Although we never calculate the actual surface geometry, the image is a good visual compromise between a space-filling spheres and a full surface calculation.

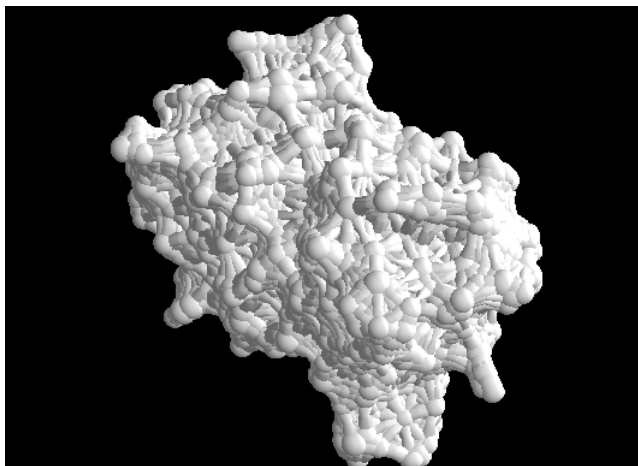


Figure 4: Approximate solvent-accessible surface for protein shown in Figure 1, using DirectX three-matrix vertex blending.

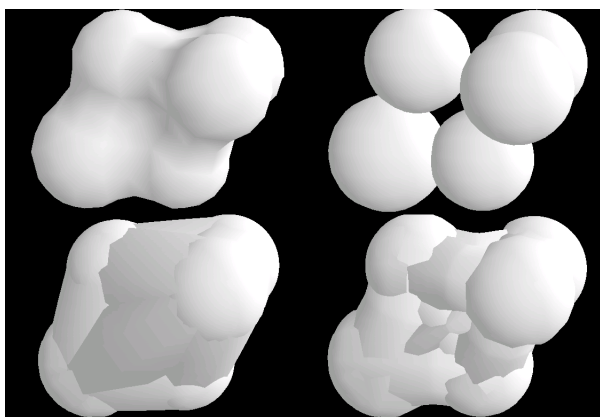


Figure 5: Solvent accessible surface of a small (5 atom) molecule using: (top left) SURF algorithm[8]; (top right) spheres of the van der Waals radius; (bottom left) two matrix vertex blending; (bottom right) three matrix vertex blending.

The interactive linkage so crucial to our merged 3D and 2D visualization environment requires fast queries of the geometry under the mouse pointer, and since the protein is already colored using a texture map, we are able to do the query by rendering to a background buffer using a sequentially colored texture that encodes the residue number along the protein. With lighting and shading disabled, the pixel color at the pointer location on the unseen background buffer uniquely identifies the residue without any geometric hit-detection. This relies on direct readback of the unadulterated RGB value from memory, which is provided by DirectX since it allows direct reads from buffers in video memory. Although the bandwidth of such reads may not be optimized for reading large areas, the readback expense is minimal for the one pixel, and we experience no noticeable latency.

## 4 Conclusions and Future Directions

Simple animations of moving ball-and-stick or ribbon representations are limited in being able to provide insight and intuition about proteins. We have developed an environment where multiple, linked views of the protein allow a user to interactively probe and

query the protein to see patterns and relationships. Key to the value of the system we have developed is the ability to generate immediate results to queries that result simply from letting the mouse dwell for a fraction of a second, even in the three-dimensional views. The results of these picks can then be further investigated by zooming or rotating to obtain a more convenient viewpoint. This leads to a very natural environment for investigating relationships.

This case study has two classes of users: The designers of the application, who learned DirectX and wrote code to take advantage of its features, and the end-users of the application who experimented with the viewer to gain insight into proteins involved in their molecular dynamics simulations. We found the DirectX API to be somewhat cumbersome to learn, partly because the viewer is a multi-windowed MFC application and most of the DirectX examples are game-oriented, non-MFC, and written at a much lower level. Once the framework of the application was in place, however, new features were straightforward to implement. Part of the programming difficulty was due to our use of DirectX version 7, rather than version 8, which was released during the application development. Although version 8 addresses some of the programming complexity issues, particularly for people new to the API, the fact that the API is changing significantly with each release may be an issue more palatable to game programmers than those writing scientific visualization systems. Writing to version 8 also would have forced users to download the latest version of DirectX, which would have reduced the portability and ease-of-use of the application.

The end-users of the application, who provided feedback and feature requests during its development, were pleased with the linked multiple representations of the proteins and high performance achieved on a wide range of platforms, particularly their laptops (typical configuration: Pentium III, 733 MHz, 128MB, S3 Savage/IX graphics adapter). Having the high degree of interactivity conveniently on a laptop was particularly important when loading and animating long trajectories containing multiple time steps. The end-users also appreciated the consistent color scheme and use of mouse hovers rather than mouse clicks. The negative aspects of the application were related to features not yet implemented rather than graphical limitations involving the use of DirectX: Inability to export data such as contact maps, inability to load multiple pdb files, etc. These features, and other graphical extensions using more DirectX capabilities, are expected as the application evolves.

In conclusion, while commodity graphics hardware cards are designed for game applications, rather than for scientific visualization, this doesn't mean that the features of the technology can not be useful for scientific applications. While we have taken advantage of a number of the features of the DirectX API in our application, such as vertex buffers and vertex blending, there are a number of features for which we can see potential value. For example, bump maps or procedural textures on generated geometry can indicate other information; as a simple example, the many local properties calculated during a molecular dynamics simulation could be indicated by a bump map whose frequency profile depended on the value of the property. As another example, a large system of atoms could be represented by point sprites, which are implemented in DirectX to simulate particle systems but also might work well for molecular dynamics simulations. The value of interaction in developing intuition and understanding has long been appreciated by the visualization community; the advent of a market for fast graphics has brought the capability of such interactive applications into reach on standard desktop and laptop machines.

## Acknowledgements

We thank James Klosowski for his suggestion of the three-dimensional picking technique and Bruce D'Amora for discussion

of DirectX issues. We thank the entire IBM Blue Gene team for their input and advice.

† Trademark or registered trademark of Silicon Graphics, Microsoft Corporation, or Sun Microsystems.

## References

- [1] O. M. Becker. Representing protein and peptide structures with parallel-coordinates. *Journal of Computational Chemistry*, 18:1893–1902, 1997.
- [2] H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I. Shindyalov, and P. Bourne. The protein data bank. *Nucleic Acids Research*, 28:235–242, 2000.
- [3] M. Carson and C. E. Bugg. Algorithm for ribbon models of proteins. *J. Mol. Graphics*, 4:121–122, 1986.
- [4] Kinemage, <http://kinemage.biochem.duke.edu>.
- [5] G. N. Ramachandran and V. Sasisekharan. Conformation of polypeptides and proteins. *Adv. Protein Chem.*, 23:283–438, 1968.
- [6] J. S. Richardson. The anatomy and taxonomy of protein structure. *Adv. Protein Chem.*, 34:167, 1981.
- [7] The impact of computer games on scientific and information visualization: If you can't beat them, join them. Panel at IEEE Visualization, 2000. Theresa-Marie Rhyne, Chair.
- [8] A. Varshney, J. Frederick P. Brooks, D. C. Richardson, W. V. Wright, and D. Manocha. Defining, computing, and visualizing molecular interfaces. In *Proceedings of IEEE Visualization*, pages 36–43, 1995.
- [9] Vmd, [www.ks.uiuc.edu/Research/vmd](http://www.ks.uiuc.edu/Research/vmd).
- [10] Chime, [www.mdlchime.com/chime](http://www.mdlchime.com/chime).
- [11] Povray, [www.povray.org](http://www.povray.org).
- [12] Protein explorer, [www.proteinexplorer.org](http://www.proteinexplorer.org).