MDVIZ: A Molecular Dynamics Visualization Toolkit

Wael R. Elwasif *, Jonathan D. Moore †, Peter T. Cummings †‡*, and Robert C. Ward *

Technical Report UT-CS-99-437 [1]

University of Tennessee

December 1999

Abstract. The paper describes a software package called MDVIZ providing the tools necessary to visualize simulations of molecular dynamics systems. The toolkit, which uses a client-server approach for maximum flexibility, enables on-line or post-processing visualization as well as real-time computational steering. The toolkit requires only hardware and software components that are readily available free or for mimimum cost to the research community. Few assumptions about the computing environment are made, making MDVIZ easily portable to different platforms and extendable.

1 Introduction

The explosive growth in computing power that took place over the last two decades has provided the necessary power for simulations of increasingly complicated molecular dynamics systems. Limitations on the number of atoms involved in a particular simulation, the complexity of the model used, and the duration of such a simulation have been constantly removed with the use of more powerful machines and the increasing availability of large cheap memory modules. An important element of the simulation process, interpretation via on-line visualization has been receiving more attention lately. This surge in interest can be attributed in part to the increasing availability of powerful computer graphics systems that can process the large amounts of data involved in constructing visualization frames at a rate compatible with the human eye's perception of continuous motion (30 frames per second).

*Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301
†Department of Chemical Engineering, University of Tennessee, Knoxville, TN 37996-2200
‡Department of Chemistry, University of Tennessee, Knoxville, TN 37996-1600
[1] Available from: http://www.cs.utk.edu/~library/TechReports.html

Large simulations are being increasingly carried out on parallel machines that are optimized for scientific computing and which do not necessarily have the hardware or software libraries to visualize simulation results in real time. Some efforts have gone into developing graphics software libraries for some of the parallel machines most commonly used for scientific computing, e.g. [2–4]. However, the results of such efforts are usually tuned to a particular machine and extremely difficult to port to other architectures.

Another approach to online visualization has been the use of a dedicated visualization server that is connected to the machine performing the simulation via fast networks. This approach is becoming increasingly popular with the development of fast network technologies (e.g. ATM networks and gigabit ethernet). In addition this approach allows visualization to be performed on machines that are particularly optimized for graphics thus freeing computing resources on the simulating machines (which are typically more expensive.)

In this report we describe MDVIZ, a Molecular Dynamics VisualiZation toolkit, that's built using a client-server approach for maximum flexibility. The toolkit is intended for use in a research/educational environment, hence the decision to use only hardware and software components that are readily available for minimal cost to the general research community. The MDVIZ makes few assumptions about the underlying hardware and software environment, which makes it easily portable to many platforms. PVM [5] is used to manage the setup of the client-server system as well as manage inter-process communications.

2 Molecular Dynamics

In a molecular dynamics (MD) simulation, the non-linear ordinary differential equations of motion for the dynamics of the molecules of interest are solved using standard numerical methods for initial-value problems, and the trajectories of the molecules are calculated as a function of time. The text by Allen and Tildesley [6] provides an overview of molecular simulation in general and MD in particular. The input to a MD simulation is typically the set of initial positions and momenta of the molecules as well as the parameters specifying the system's thermodynamic state (e.g. density and temperature). During the course of the simulation, the thermophysical properties (such as pressure, energy, and viscosity) are calculated as time averages from microscopic expressions given in terms of the positions and momenta of the individual molecules. Visualization of the molecules, either during the simulation or upon its completion, can be extremely helpful in terms of interpreting the thermophysical data as well as debugging the simulation code itself. In addition, during the course of the simulation, one may desire to modify the values of certain parameters governing the simulation (e.g. temperature and pressure) and visually monitor the effects such changes impose on the simulated system. MDVIZ is designed to provide these capabilities to the general MD user.

3 System Components

The MDVIZ server runs on a machine equipped with the required graphics rendering resources to manage the actual display of simulation frames. The server uses the OpenGL [1,12] graphics library for rendering. OpenGL is emerging as a de-facto standard for rendering on many platforms with hardware support available on many graphics display cards. In addition, a free software implementation of OpenGL is readily available (however it has been our experience that performance of the software port is not on par with the available hardware implementations in terms of robustness and speed.) The choice to build the server directly on top of OpenGL allows for maximum portability to different platforms that may not support some of the layers built on top of OpenGL to enhance its interface and/or improve overall system performance (e.g. Open Inventor [11]).

The server also uses GLUT [8], a free toolkit built on top of OpenGL to provide encapsulation for some of the functionality in OpenGL and to provide mechanisms for managing display windows and user interactions, which are not part of OpenGL.

Another free library that is used by the server to enhance its functionality is libtiff [9]. This library, which is also built on top of OpenGL, allows for the capture of rendered images into tiff format files for permanent storage. This option allows for generating post-mortem "movies" of simulations deemed important for presentations or archiving. However, use of this facility causes degradation in server's response time and rate of frame rendering.

For purposes of data communication, we adopted PVM because of its support for heterogeneous machines. At time of MDVIZ implementation, the MPI [7] message passing standard did not offer support for heterogeneous environments. Since MDVIZ aims at providing support for possibly many simulation environments communicating with the rendering server, PVM was the natural choice for this reason. As MPI evolves and incorporates more support for process management and heterogeneous environments, the communication subsection of MDVIZ could be easily ported to MPI.

The MDVIZ server provides a flexible interface to allow the user to control the simulation from the rendering side. This interface is implemented using the scripting language Tcl/Tk [10], which allows for rapid development and easy customization of the user interface part.

The client side of MDVIZ is not currently implemented as a separate library. The client interacts with the server via PVM according to the protocol described in section 6. The simulation module is responsible for setting up necessary data structures for PVM, composing messages according to the format described in the communication protocol, receiving server responses, interpreting those responses, and carrying out any actions specified therein. This functionality could be consolidated into a separate client-side library as part of future releases for ease of use. The Appendix contains an outline of a generic MD simulation module including the necessary code for interfacing with MDVIZ.

It is the responsibility of the user to make sure PVM is up and running on the two hosts (the simulation client and the visualization server)that are involved in the simulation session. Although PVM allows for dynamic process creation and management, this feature was not used in this early version of MDVIZ. In future releases, this behavior could be modified to allow for an easier user interface to the client-server system.

4 What is being displayed?

The MDVIZ server assumes that a client connecting to the server will display frames that correspond to snapshots of a molecular dynamics simulation involving a collection of molecules over a certain period of time. It is assumed that the user is conducting simulations involving the behavior of molecules contained in a rectangular space. The coordinates of this rectangle are used by the MDVIZ server to position the viewer with respect to the simulation space in a convenient manner. The locations of molecules involved in the simulation at a particular point in time constitute the bulk of frame data that the client sends to the server to render a new frame. MDVIZ currently supports a sphere model of individual atoms rendering. Other models could be supported in future releases (e.g. ball and stick model).

MDVIZ has a database that incorporates physical properties of atoms of 103 elements. These properties, which control the rendering of molecules of any particular element, include the radius and the red, green, and blue color components which combine to give the element a distinct color. A particular element is accessed through its index in the periodic table of elements. In addition, MDVIZ allows the user (client) to define up to 11 extra "special" elements at runtime. These "user-defined" elements could be used to describe a molecule (or a group of molecules) for easy identification at run time. User defined elements are given indices between 110 and 120 (inclusive).

5 Computational Steering

The current implementation of MDVIZ supports a simple mechanism through which the user can remotely control certain aspects of the simulation process from the visualization server. The client (simulation code) sends a list of configurable parameters as part of an initialization message required by the MDVIZ server. This list includes information that defines the name of the configurable parameters, their minimum allowable values, their maximum allowable values, and their initial settings. The server passes this information onto the Tcl/Tk GUI module that constructs an interface (see figures 1- 2) that allows the user to modify the parameters specified in the initialization message. The modified values are then conveyed back to the client for use in its running simulation. It should be noted that the current implementation assumes that certain parameters will not change during the lifetime of the simulation process (e.g. the number of atoms involved).

6 Client-Server Communication Protocol

In this section, we describe the format of the messages exchanged between the client and the server and the actions triggered by such messages. We can broadly divide message exchange into two main stages, initialization and data/parameters exchange. In the following description, some data items exchanged between the simulation client and the visualization server are not currently used (by the

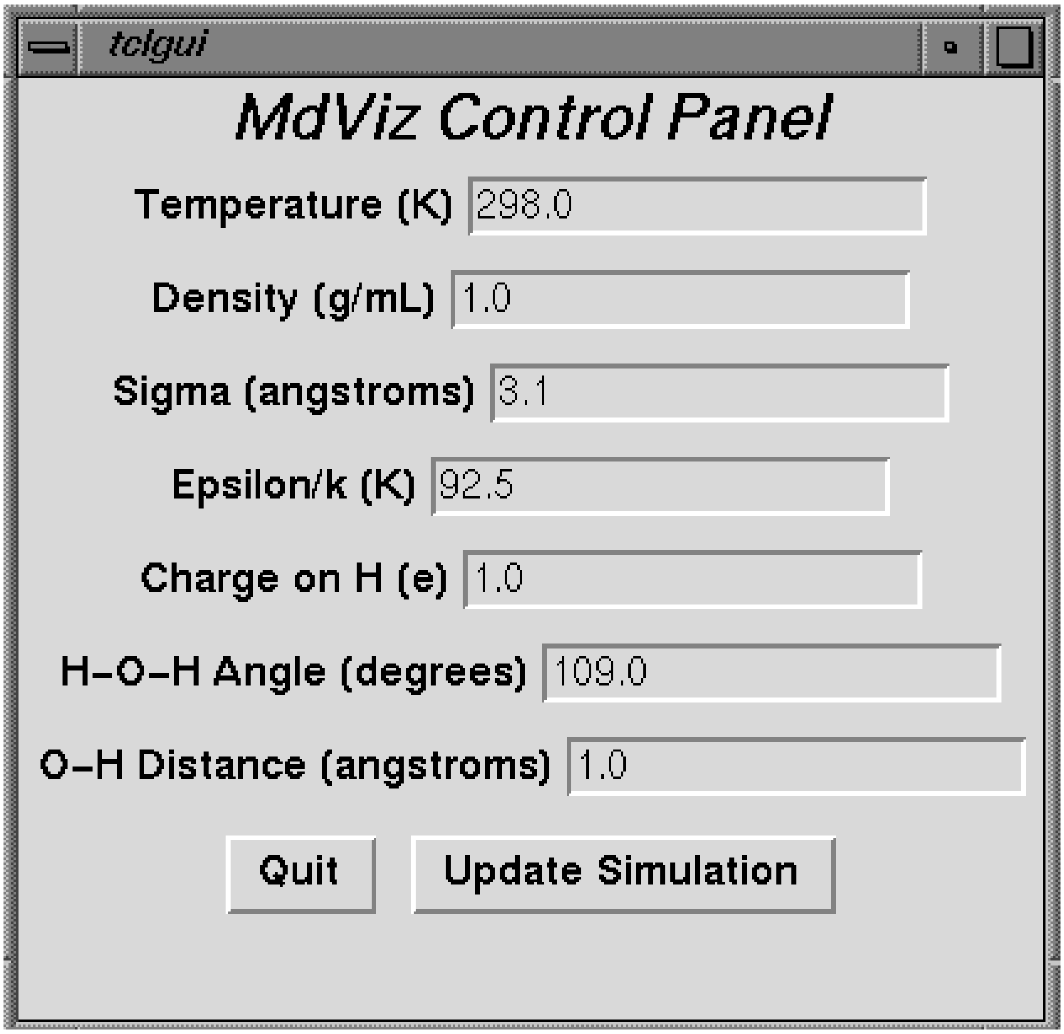

Figure 1: The graphical user interface of MDVIZ used in conjunction with a simulation of water.

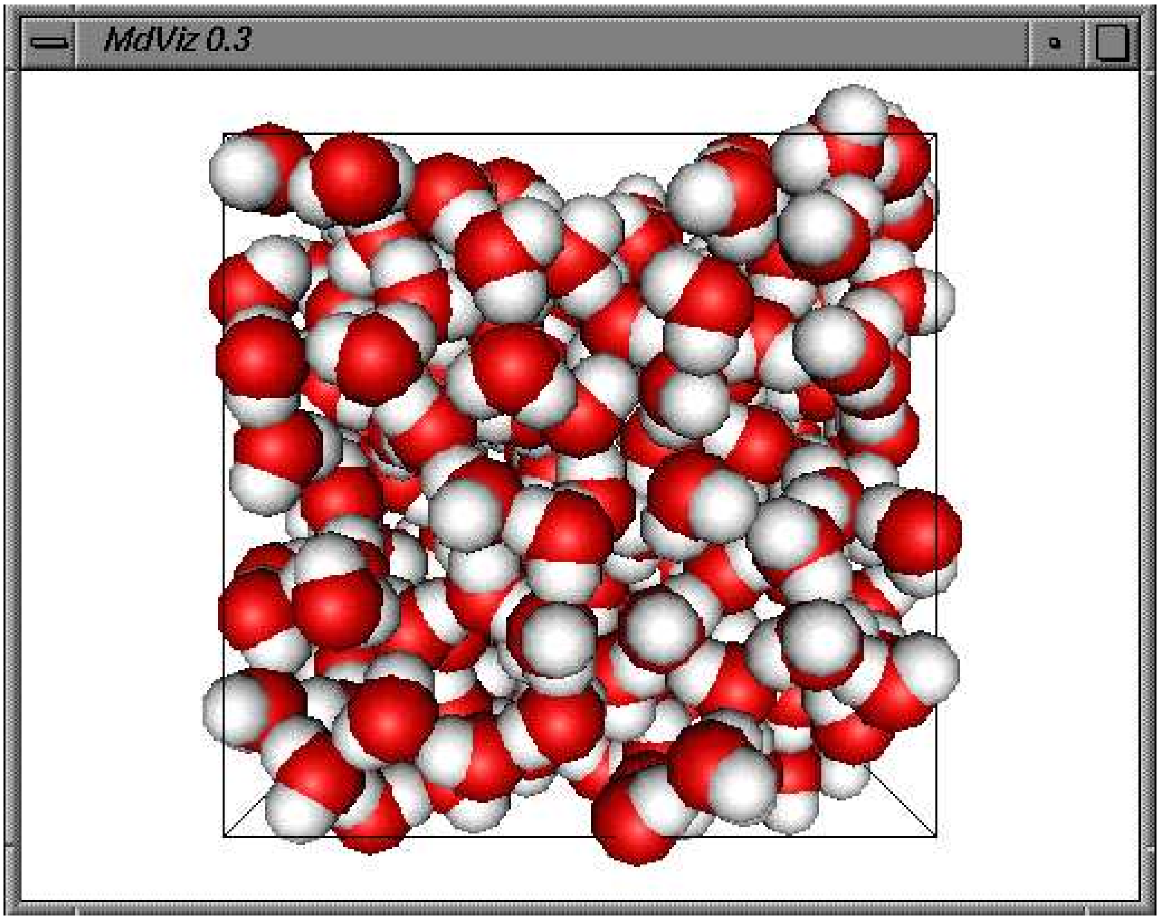

Figure 2: Visualization by MDVIZ of a simulation of water.

server) to affect the visualization process. Those items are included for future releases (particularly of the server side).

6.1 Initialization

The initialization process can be further divided into two main steps

1. **PVM initialization:** In this step, the client and the server are connected to form a virtual machine. This is accomplished by joining a PVM group, conveniently named *MDVIZ* and synchronizing with each other afterwards. It is the responsibility of the user to construct a virtual machine that incorporates the host running the simulation and the host where the MDVIZ visualization module is to be run. In addition, the user is also responsible for initiating the simulating process and the visualization server manually, each on its respective host.

Parameter	**type**	**Description**	**count**
textLineLength†	Integer4	Length of text line	3
textLine1†	Byte	Text line	
numFrames	Integer4	Total number of frames to be displayed	1
numSteps †	Integer4	Number of frames to be skipped	1
simCorner1X	Real4	Coordinates of simulation cell lower left front corner	1
simCorner1Y	Real4		
simCorner1Z	Real4		
simCorner2X	Real4	Coordinates of simulation cell upper right back corner	1
simCorner2Y	Real4		
simCorner2Z	Real4		
numAtoms	Integer4	Total number of atoms in simulation	1
numFixedBonds†	Integer4	Number of fixed bonds between atoms	1
fixedBondsFrom†	Integer4	Indices of atoms at one end of fixed bonds	numFixedBonds
fixedBondsTo†	Integer4	Indices of atoms at the other end of fixed bonds	numFixedBonds
numVarPar	Integer4	Number of steering parameters	1
parNameLength	Integer4	Length of parameter name	numVarPar
parName	Byte	Parameter name	

continued on next page

continued from previous page

Parameter	type	Description	count
parMinVal	Real4	Parameter minimum value	
parMaxVal	Real4	Parameter maximum value	
parInitVal	Real4	Parameter initial value	
numExtraElem	Integer4	Number of user-defined elements	1
elemIndex	Integer	Element identifying index	numExtraElem
elemRed	Real4	Red component in element atom color [0-1]	
elemGreen	Real4	Green component in element atom color [0-1]	
elemBlue	Real4	Blue component in element atom color [0-1]	
elemNameLen	Integer4	Length of element name	
elementName	Byte	Element name	
elemradius	Real4	Element atom radius (in Angstrom)	

† *Parameter not used in current implementation*

Table 1: Initialization message format

2. **MDVIZ initialization:** In this step, the MDVIZ visualization server configures itself for the client that had just joined the PVM group. This step is initialized by the client, which sends an initialization message to the server. This message contains information regarding the simulation environment. The server uses this information to initialize its internal state in anticipation of receiving subsequent frame data. The initialization message is given a PVM message ID of 100. The layout of this message is outlined in table 1. It should be noted that types given in table 1 correspond to PVM types used in the actual implementation.

Upon receipt of the initialization message, the MDVIZ visualization modules spawns a GUI process configured with the computational steering parameters specified in the initialization message. In addition, the visualization module initializes its internal state (including the graphics subsystem). As part of the state initialization, an initial setting of the viewer's relative position with respect to the simulation box is chosen for future use in rendering. This position can be adjusted through keyboard interface to allow the user to navigate through the simulation scene as will be explained in section 7.

6.2 Exchanging frame data

Upon completion of the initialization process, the MDVIZ server enters into a loop that controls data exchange with the client as well as possible user input either by controlling the display environment or by modifying the steering parameters. Frames are displayed asynchronously, where the server signals to the client its readiness to display further frames after rendering the previous frame. This signal takes the form of an acknowledgment message that instructs the client to send a new frame to display and/or update one or more steering parameter. The decision as to which frame to send depends on the simulation and visualization requirements.

The client may choose to continue with the simulation process up to the point where it needs to display a new frame, at which time it may go into a waiting loop which periodically probes the PVM subsystem for server messages (this would be the case if regularly spaced frames are required.) Another alternative would be for the client to continue with the simulation process, while periodically probing for the server response. This scenario may result in frames being displayed at possibly highly irregular intervals, since the user at the visualization end can interactively freeze a particular frame for further inspection for an extended period of time before requesting further frames as will be explained in section 7.

The following tables outline the format of these two types of messages exchanged between the client and the server. The format of the message containing frame data is shown in Table 2, while the format of the acknowledgment message is shown in Table 3. Frame data message is given a PVM message id of 200, while the acknowledgment message has a PVM message id of 300.

Parameter	**type**	**Description**	**count**
frameTime†	Real4	Frame time (in μSec)	1
simCorner1X	Real4	Coordinates of simulation cell lower left front corner	1
simCorner1Y	Real4		
simCorner1Z	Real4		
simCorner2X	Real4	Coordinates of simulation cell upper right back corner	1
simCorner2Y	Real4		
simCorner2Z	Real4		
numTransBonds†	Integer4	Number of transient bonds between atoms	1
atomIndex	Integer4	List of atom indices	numAtoms
atomXPos	Real4	$atomXPos[i]$ = x-position of atom $atomIndex[i]$	numAtoms
atomYPos	Real4	$atomYPos[i]$ = y-position of atom $atomIndex[i]$	numAtoms
atomZPos	Real4	$atomZPos[i]$ = z-position of atom $atomIndex[i]$	numAtoms
continued on next page			

continued from previous page			
Parameter	**type**	**Description**	**count**
atomType	Integer4	$atomType[i]$ = Index of element to which atom i belongs	numAtoms
atomMol†	Integer4	$atomMol[i]$ = Index of molecule to which atom i belongs	numAtoms
transBondsFrom†	Integer4	Indices of atoms at one end of transient bonds	numTransBonds
transBondsTo†	Integer4	Indices of atoms at the other end of transient bonds	numTransBonds
transBondsColor†	integer4	Color indices for respective transitional bonds	numTransBonds

† *Parameter not used in current implementation*

Table 2: Frame data message format

Parameter	**type**	**Description**	**count**
numChgPar	Integer4	Number of changed steering parameters	1
chgParIdx	Integer4	Index of changed steering parameter	numChgPar
chgParVal	Real4	New value of steering parameter	

Table 3: Acknowledgment message format

7 Interaction with running visualizations

The MDVIZ server allows the user to control not only the configurable parameters conveyed by the client as part of the initialization message, but also the visual aspects of the displayed frames themselves. The user can freeze a particular frame for further inspection, navigate through the displayed frame for different angles of view, and save frames to permanent storage for future compression into a movie using a variety of available software packages.

In this release of MDVIZ, user interaction is done mainly through keyboard entries. This feature could be changed in future releases to allow for other types of controls (e.g. mouse control). In what follows we describe the actions that can be initiated by the user at startup and during the visualization process itself.

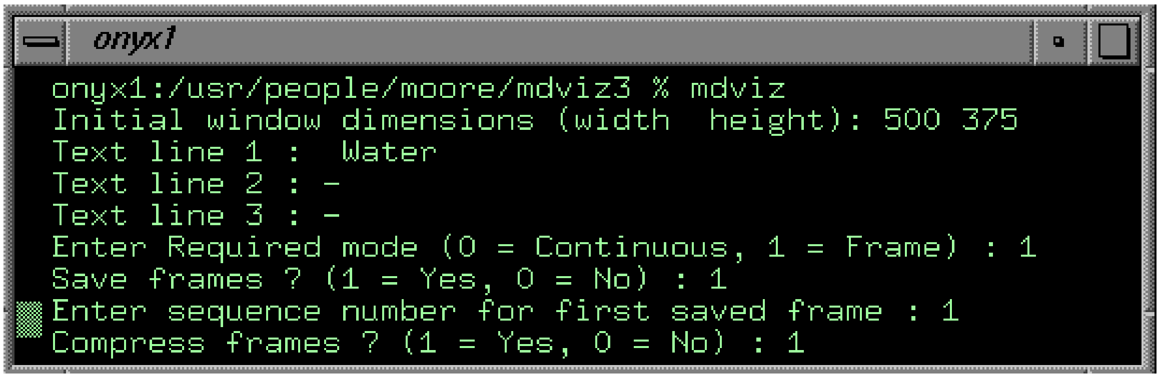

Figure 3: Command line window where MDVIZ is invoked and the mode of operation is specified.

7.1 Initialization

As part of the startup process, the user is given the ability to specify the initial mode of the visualization process (see figure 3). Currently two modes are supported, frame mode; in which the user needs to explicitly request every subsequent frame, and continuous mode; in which a new frame is fetched and displayed as soon as the server is done displaying the preceeding frame. In addition, the user is given the choice of saving displayed frames to permanent storage or not. If the user chooses to save the visualization session, each frame will be saved in a file with the name *framei.tif*, where i is the frame sequence number.

7.2 Frame display control and navigation

The user can navigate through the displayed frame by controlling the *target point* in the display. The target point is defined to be the point affected by changes initiated by the user during a visualization session. Currently MDVIZ supports two target points, the viewer's position and the viewing target position. By changing the viewer's position, the user is able to move about the displayed scene while keeping the focus of view on the same spot. On the other hand, by changing the viewing target, the user directs his attention at different zones in the display while maintaining a fixed position. The ability to control both these points give users the flexibility to explore interesting zones in the simulation cell easily. The different actions that can be applied to target points as well as other controls available to the user are outlined in table 4. The capability of viewing a simulation from different points of view is illustrated in figure 4 containing visualizations of a confined alkane lubricant.

Key	Action	Description
q Q	Quit	Terminate visualization server
m M	Switch mode	Change between single frame and continuous display mode
n N	Next frame	Display next frame (in frame mode)
r	Rotate counterclockwise	Rotate the viewer around the point of view
R	Rotate clockwise	Rotate the viewer around the point of view
x X	Move along x-axis	Increase x-axis coordinate of target point Decrease x-axis coordinate of target point
y Y	Move along y-axis	Increase y-axis coordinate of target point Decrease y-axis coordinate of target point
z Z	Move along z-axis	Increase z-axis coordinate of target point Decrease z-axis coordinate of target point
f F	Move forward	Move target point forward
b B	Move backward	Move target point backwards
s S	Switch target point	Switch target point between viewer position and point of view

Table 4: Frame display control actions

8 Future work

This release of MDVIZ lays the required framework for further enhancements in the functionality of the visualization system. One immediate direction of work would be to encapsulate the comunication calls on the client side in a separate library that exports an intuitive API to the user of the simulation code. This step should make it easier for users of the simulation code to use MDVIZ with little or no knowledge of the details of PVM. Another improvement to the ease-of-use aspect of the package would be to improve the frame display control mechanism offered in MDVIZ to provide easier user interface that might employ menu-driven actions and/or mouse control.

The server-side features implemented in the current release of MDVIZ are but a sub-set of those features that are envisioned to be in future releases. Support for dynamic inter-molecular bonds, support for different bond representations (e.g. ball and stick representation), support for different shapes of the simulation cell are but a few of the features that could significantly add to the existing functionality in MDVIZ.

One important direction of work would be to improve the performance of the visualization server to reduce frame rendering time. This enhancement is necessary for efficient use of MDVIZ to render frames containing a large number of molecules at relatively close time intervals. This direction of work will ultimately require the use of parallel rendering techniques (in space or time) to achieve the desired improvement in performance.

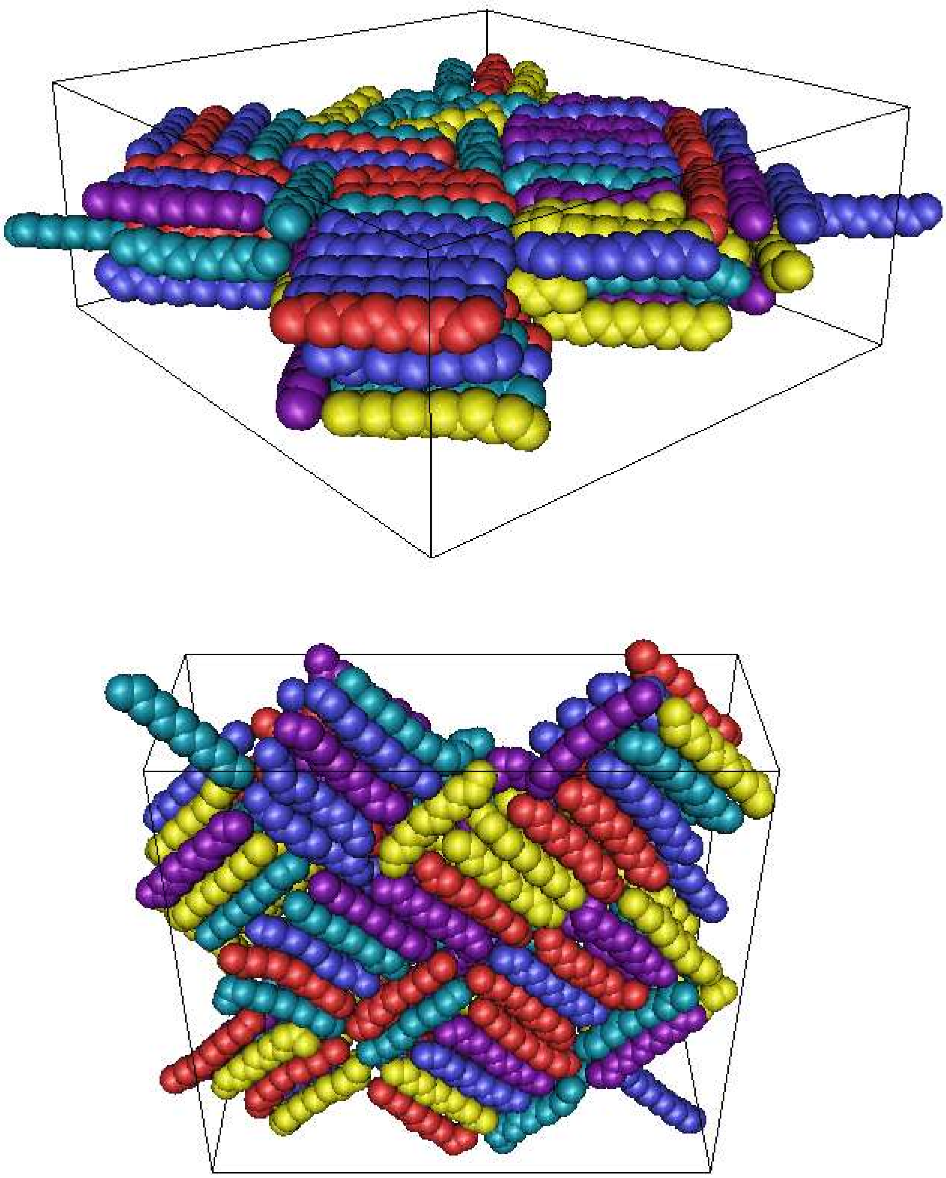

Figure 4: A simulation of a confined alkane lubricant as seen from two different points of view. User-defined elements are utilized to distinguish between chains that are all chemically identical.

9 Conclusion

In this report, we presented the design and implementation details of a distributed molecular dynamics visualization toolkit that provides support for on-line visualization of dynamic molecular behavior in a parallelpipe-shaped simulation cell. The toolkit offers a flexible computational steering capability that allows for visualization server-side control of user-defined simulation parameters. This tool should prove useful for researchers who use molecular dynamics simulation techniques, in particular for rapid change in simulation parameters and for generation of simulation videos that capture a possibly long simulation interval.

References

[1] Opengl Architecture Review Board, Chris Frazier (Editor), and Renate Kempf. *Opengl Reference Manual : The Official Reference Document to Opengl, Version 1.1.* Addison-Wesley, 1997.

[2] Thomas W. Crockett. PGL- a parallel graphics library for distributed memory applications. `http://www.icase.edu/reports/interim/29/PGL.html`.

[3] Thomas W. Crockett. Beyond the renderer: Software architecture for parallel graphics and visualization. Technical Report 96-75, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, 1994.

[4] Thomas W. Crockett. Design considerations for parallel graphics libraries. Technical Report 94-49, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, 1994.

[5] A. Geist, A. Beguelin, J. Dongarra, R. Manchek, W. Jaing, and V. Sunderam. *PVM — A Users' Guide and Tutorial for Networked Parallel Computing.* MIT Press, Boston, 1994.

[6] M. Allen and D. Tildesley. *Computer Simulation of Liquids.* Oxford University Press, Oxford, 1987.

[7] W. Gropp, E. Lust, and A. Skjellum. *Using MPI: Portable parallel programming with the message passing interface.* MIT Press, Boston, 1994.

[8] Mark Kilgard. *OpenGL Utility Toolkit (GLUT).* `http://reality.sgi.com/opengl/opengl-links.html#glut`.

[9] Sam Leffler. *LIBTIFF - TIFF Software Library* . `http://www.sgi.com/Fun/tiff/tiff-v3.4beta018/html/`.

[10] John K. Ousterhout. *Tcl and the Tk Toolkit.* Addison-Wesley, 1994.

[11] Josie Wernecke. *The Inventor Mentor : Programming Object-Oriented 3d Graphics With Open Inventor, Release 2*. Addison-Wesley, 1994.

[12] Mason Woo, Jackie Neider, Tom Davis, , Paula Womack, and Opengl Architecture Review board. *Opengl Programming Guide : The Official Guide to Learning Opengl, Version 1.1*. Addison-Wesley, 1997.

Appendix

This Appendix contains an outline of a generic MD simulation module including the necessary code for interfacing with MDVIZ.

```
      PROGRAM MD

      integer     N, NSTEP, NMOL, NUMPERMBND

      parameter   ( N = 768 )
      parameter   ( NMOL = 256 )
      parameter   ( NSTEP = 1 )
      parameter   ( NUMPERMBND = 1 )

C N is the number of atoms to be visualized
C NMOL is the number of molecules
C     (NMOL=N/(# of atoms per molecule))
C NSTEP number of timesteps in simulation
C NUMPERMBND is the number of permanent bonds

      integer lnonelen, lntwolen, lnthreelen
      integer elemNamelen

C Lengths of various character variables

      parameter   ( lnonelen = 7 )
      parameter   ( lntwolen = 1 )
      parameter   ( lnthreelen = 1 )
      parameter   ( elemNamelen = 4)

C Various character variables declared

      character*lnonelen line1
      character*lntwolen line2
      character*lnthreelen line3
      character*elemNamelen elemName

C Variables for PVM

      integer mytid, me, info
      integer  myinst, target, bufid

C Variables related to computational steering

C pnumbr is the number of steered parameters
      integer pnumbr, steer1len, steer2len
      parameter   ( pnumbr = 2 )
      parameter   ( steer1len = 15 )
      parameter   ( steer2len = 14 )

C Character variables naming the steered parameters

      character*steer1len steer1
      character*steer2len steer2

      integer pindex(2)

C Min and max allowed values of steered parameters

      real    pmin(pnumbr), pmax(pnumbr)

C Array containing updated steered parameters

      real  value(pnumbr)

C Bond-related variables

      integer bonds, tranzbnd
      integer permbndfrom(NUMPERMBND)
      integer permbndto(NUMPERMBND)

C Other variables

      integer i, istep, j, be, jmol, natom
      integer atomnum(N), typenum(N), ack, narray
      integer molenum(N), nframes, framerate, nskip
      real    coor(6), timesend, timecon, timestep
      real    RX(N), RY(N), RZ(N), temp, delt


C VARIABLES FOR USER-DEFINED ELEMENTS.

      integer numXelem, elemi
      real    RGB(11,3), elemR

C ** JOIN THE PVM GROUP **

      call pvmfmytid(mytid)

      call pvmfjoingroup('MDVIZ',myinst)
      if ( myinst .lt. 0) then
              call pvmfexit ( info )
              stop
      endif

      call pvmfbarrier('MDVIZ',2,info)

C ** Find out the tid of MDVIZ **

      if (myinst.eq.0) then

              call pvmfgettid('MDVIZ',1,target)
      else
              call pvmfgettid('MDVIZ',0,target)
      endif

C ** Set up MDVIZ parameters **


C Text lines to display

      line1 = ' Water '
      line2 = '-'
      line3 = '-'
```

```
C Text description of first steering parameter

      steer1 = 'Temperature (K)'

C Text description of second steering parameter

      steer2 = 'Density (g/mL)'

C Max and min values of the steering paramters

      pmin(1) = 298.0
      pmax(1) = 600.0
      pmin(2) = 0.01
      pmax(2) = 1.0

C Define indices of steering parameters

      pindex(1)=1
      pindex(2)=2

C The current release of MDVIZ does not use the
C permanent and transiet bond information, but
C future releases will.

C Initialize the number of transient bonds

      tranzbnd = 0

C The number of permanent bonds to be visualized.

      bonds=0

C The agreed upon standard is to send time to MDVIZ
C in units microseconds.  To do so, timecon should
C be set to the proper value so that
C istep*timstep*timecon has units of microseconds.

      timestep = 0.001
      timecon = 1.0

C Send frame to MDVIZ every <framerate> timesteps.

      framerate = 1

C After MDVIZ displays a frame, it should skip
C <nskip> frames it receives before displaying
C another frame.

      nskip = 0

C The coordinates of the front bottom lefthand
C and top rear righthand corners of the
C simulation cell.

      coor(1) = 0.0
      coor(2) = 0.0
      coor(3) = 0.0
      coor(4) = 19.7
      coor(5) = 19.7
      coor(6) = 19.7

C Total number of frames MDVIZ will be visualizing

      nframes = nstep / framerate

C Assign atomnum, molenum, and typenum values



C Number of elements the user will define.

      numXelem = 0

C Radius in angstroms of the defined elements.
C Make this an array if you want to use more than
C one radius value.

C     elemR = 0.5

C Names for user-defined elements can be assigned.


      elemNameLen = 4
      elemName = 'same'

C The send is intiated and data are packed.  Two of the
C pvmfpack lines are commented out since I have set
C bonds=0 temp and delt are the variables that are
C steered in this example.

      call pvmfinitsend( PVMDEFAULT, bufid )
      call pvmfpack(integer4,lnonelen,1,1,info)
      call pvmfpack(byte1,line1,lnonelen,1,info)
      call pvmfpack(integer4,lntwolen,1,1,info)
      call pvmfpack(byte1,line2,lntwolen,1,info)
      call pvmfpack(integer4,lnthreelen,1,1,info)
      call pvmfpack(byte1,line3,lnthreelen,1,info)
      call pvmfpack(integer4,nframes,1,1,info)
      call pvmfpack(integer4,nskip,1,1,info)
      call pvmfpack(real4,coor,6,1,info)
      call pvmfpack(integer4,N,1,1,info)
      call pvmfpack(integer4,bonds,1,1,info)
C     call pvmfpack(integer4,permbndfrom,bonds,1,info)
C     call pvmfpack(integer4,permbndto,bonds,1,info)
      call pvmfpack(integer4,pnumbr,1,1,info)
      call pvmfpack(integer4,pindex(1),1,1,info)
      call pvmfpack(integer4,steer1len,1,1,info)
      call pvmfpack(byte1,steer1,steer1len,1,info)
      call pvmfpack(real4,pmin(1),1,1,info)
      call pvmfpack(real4,pmax(1),1,1,info)
      call pvmfpack(real4,temp,1,1,info)
      call pvmfpack(integer4,pindex(2),1,1,info)
      call pvmfpack(integer4,steer2len,1,1,info)
      call pvmfpack(byte1,steer2quit,steer2len,1,info)
      call pvmfpack(real4,pmin(2),1,1,info)
      call pvmfpack(real4,pmax(2),1,1,info)
```

```
       call pvmfpack(real4,delt,1,1,info)
       call pvmfpack(integer4,numXelem,1,1,info)

C These lines are used if using user-defined elements

C       do i = 1, numXelem
C          elemi = i+109
C          call pvmfpack(integer4,elemi,1,1,info)
C          call pvmfpack(real4,RGB(i,1),1,1,info)
C          call pvmfpack(real4,RGB(i,2),1,1,info)
C          call pvmfpack(real4,RGB(i,3),1,1,info)
C          call pvmfpack(integer4,elemNameLen,1,1,info)
C          call pvmfpack(byte1,elemName,elemNameLen,1,info)
C          call pvmfpack(real4,elemR,1,1,info)
C       enddo

C Send the initialization data to MDVIZ

       call pvmfsend( target, 100, info )

C Enter main simulation loop

       do 245 istep = 1 , nstep

C Simulation code generates position data (RX,RY,RZ)
C for the next timestep.  This code is not shown
C here and is specific to each user.  This data
C could also be read from a file for a
C ``post-mortem'' visualization.

       if ( mod(istep,framerate) . eq . 0 ) then

       timesend=timestep*real(istep)*timecon

C **** Communicate data to mdviz

       call pvmfinitsend( PVMDEFAULT, bufid )
       call pvmfpack(real4,timesend,1,1,info)
       call pvmfpack(real4,coor,6,1,info)
       call pvmfpack(integer4,tranzbnd,1,1,info)
       call pvmfpack(integer4,atomnum,N,1,info)
       call pvmfpack(real4,RX,N,1,info)
       call pvmfpack(real4,RY,N,1,info)
       call pvmfpack(real4,RZ,N,1,info)
       call pvmfpack(integer4,typenum,N,1,info)
       call pvmfpack(integer4,molenum,N,1,info)

C If tranzbnd>0, then you would have lines here
C to pack the arrays of [transient bonds from],
C [transient bonds to], and [bond color]

       call pvmfsend( target, 200, info )

C ** Receive an acknowledgement from MDVIZ **

       call pvmfrecv( target, 300, bufid )
       call pvmfunpack(integer4,ack,1,1,info)

C The value of ack returned by MDVIZ specifies how
C many steered parameters have been changed by the
C user through the gui.

       if (ack.gt.0) then

       do 24 i=1,ack

          call pvmfunpack(integer4,index(i),1,1,info)
          call pvmfunpack(real4,value(i),1,1,info)

          if (index(i).eq.1) then

             temp=value(i)

          elseif (index(i).eq.2) then

             dens=value(i)

          endif

 24    continue

       endif

 245 continue


C Exit PVM

       call pvmfexit(info)

stop
end
```