Using Tcl for Molecular Visualization and Analysis

Andrew Dalke and Klaus Schulten
Beckman Institute, 405 N. Mathews,
Urbana, IL 61801, USA

Reading and manipulating molecular structure data is a standard task in every molecular visualization and analysis program, but is rarely available in a form readily accessible to the user. Instead, the development of new methods for analysis, display, and interaction is often achieved by writing a new program, rather than building on pre-existing software. We present the Tcl-based script language used in our molecular modeling program, VMD, and show how it can access information about the molecular structure, perform analysis, and graphically display and animate the results. The commands are available to the user and make VMD a useful environment for studying biomolecules.

Introduction

There are many programs for molecular visualization (RasMol [1], Quanta [2], GRASP [3], SETOR [4]) and analysis (Naomi [5], Modeller [6], MD Toolchest [7], STRIDE [8]). They all perform many of the same basic tasks: they read in molecular structure data, perform some analysis, and display the results. Visualization programs usually analyze the data to display some aspects of it, while programs for analysis often return their results as detailed textual information.

As visualization and analysis tasks become more complex and the expectations for programs continues to rise, it becomes harder to build new software from scratch. Fortunately reusable code libraries like PDBlib++ [9] and the MMDB-API [10] have simplified program development. However, such libraries do not provide much assistance to the user, who often wants a high degree of interactivity and the ability to implement new features or combine existing features in new and novel ways without altering the underlying program code.

To address this problem, many programs implement script languages which allow the user to execute various commands and analysis routines. Some of the best known script languages are found in X-PLOR [11] , NAOMI [5] and SETOR [4] . Because the languages were developed *ad hoc*, they are rather incomplete, and do not handle some common programming abstractions. An alternate method is to extend one of the many embeddable scripting languages, such as Tcl [12] , Python [13] or Perl [14] . Over the last few years as scripting languages have matured and people have become more experienced in using them, a new generation of modeling tools have been developed based on them, e.g. Molviewer-ogl [15] , gOpenMol [16] and Chimera [17] .

We have developed several ways to use one such language, Tcl, in our molecular graphics and analysis program, VMD [18] . Originally written to display molecular dynamics trajectories and support interactive molecular dynamics, VMD now includes Tcl language extensions for controlling the graphics display, accessing the internal database of molecular information, analyzing atom selections, displaying user-defined 3D graphics, and acting on specified events. Combined with the base scripting language, these extensions provide a powerful environment for molecular modeling and suggest interesting directions for future developments.

Tcl

Tcl [12] is one of the most popular scripting languages. It is freely available and contains many of the standard language features such as variables, control loops, and procedures as well as more advanced utilities, including file I/O, process spawning, and exception handling. The most interesting aspect of its design, at least from the aspect of software development, is its use as an embeddable and extensible controller.

Embedding Tcl in another program is done through a C library interface. The library contains the interpreter and the core set of Tcl commands. Text commands are passed to the library, which evaluates the string and calls the appropriate functions. The language is extended by adding new functions, either through a Tcl procedure definition or by adding a call-back to a C function. Figure 1 shows how the components are arranged. Because of its popularity and ease of extensibility, many extensions have been built for Tcl (Tk, expect, Dp, Extend). The only thing that connects the different pieces together is the interpreter, which is why Tcl is often called a “glue” language.

VMD Extensions to Tcl

The VMD scripting language uses Tcl along with two of the common extensions, Tcl-Extend (which adds many generally useful functions) and Tcl-Dp (for network communications), and adds many molecular modeling specific extensions. Since the rest of this paper describes the various extensions, it is helpful here to elaborate more on extensions and some of the issues concerning them.

One of the simplest Tcl extensions in VMD is a set of commands for manipulating vectors and 4x4 matrices. Structural analysis often involves vector

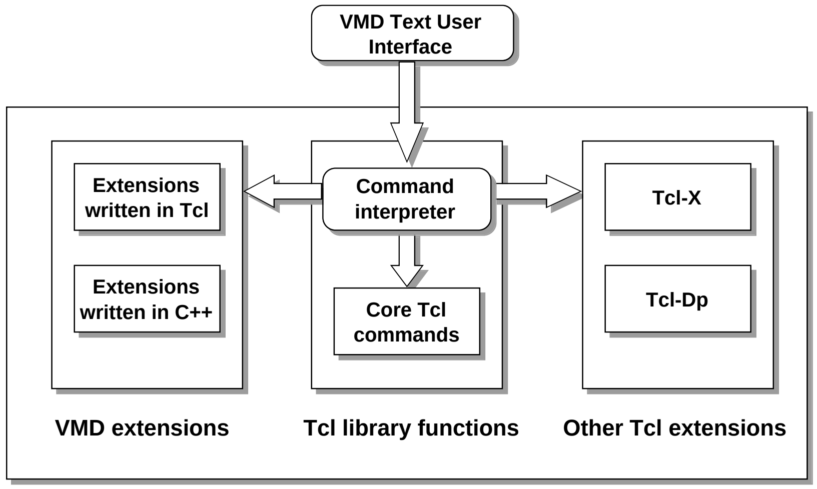

Figure 1: Organization of the Tcl interpreter, basic command library and extensions

algebra, but Tcl is a string-based language with limited provision for mathematical operations and no commands for manipulating vectors and matrices. Thus, the new commands include methods for vector addition and subtraction, dot and cross product evaluation, matrix multiplication, and various methods to construct specific types of matrix transformations.

The commands were originally developed as a set of procedures implemented in Tcl, but the constant interpretation and conversion from Tcl strings to numbers and back to strings made many of the more commonly used commands quite slow. This is especially noticeable in routines like `vecadd` which are likely to be in inner loops. As Tcl can use functions written in C or C++ just as easily as ones in Tcl, several of the most heavily used functions were converted into C, resulting in performance increases ranging from 5- to 40-fold.

Atom Selections

All of the molecular analysis routines in VMD depend on an atom selection, which is both a reference to the atoms which match a given selection text and a way to access the data associated with those atoms. A selection contains a selection text, the molecule from which the selection derives, and a list of atoms meeting the selection criteria. Since a molecule may have multiple animation

Selection Text	Selected Atoms
name N CA C O	backbone atoms in proteins
hydrophobic protein backbone	
x > 0 and y > 0 and z > 0	atoms in the first octant
segname == "PRO1"	same as "segname PRO1"
same residue as name "S.*"	all residues containing an atom name starting with S
resid 5 to 10	residues with index between 5 and 10
sqr(x-1)+sqr(y+2.3)+sqr(z) < sqr(6)	atoms within 6 Å of (1, -2.3, 0)
helix within 5 of resname HEME	atoms on an alpha helix within 5 Å of the heme

Table 1: Examples of atom selection text

frames, the selection also contains a reference to the specific frame to use.

The text used to make a selection consists of four main types of primitives. These can be combined with boolean operators to form more complex selections (see Table 1 for examples). The simplest primitive is of the form "keyword," as in "`backbone`." An atom is selected if it has the given attribute. The next simplest is of the form "keyword *value1 value2* . . . ." If the value of the attribute for an atom matching any of the values in the list, it is selected. Comparison expressions are allowed, so the nonsense selection "`mass * cos(x) / 5.0 < sqrt(abs(charge))`" is acceptable. The operators are the same as those used in the Perl scripting language so the somewhat more useful "`name =~ 'H.*'`" is also acceptable; in this case it finds atoms whose name matches the regular expression which says the first character must be an "H," e.g., hydrogen and mercury atoms. The fourth set of selection primitives include specialized forms for distance searches and set relations.

The selection text parsing is done at the C++ level and uses a keyword symbol table and a lex/yacc grammar. The constructed parse tree is evaluated to produce a boolean array of flags indicating which atoms were selected. The most interesting of these is the symbol table which contains a list of the various keyword terms recognized in a selection. Each element of the table consists of a display string, a regular expression, and accessor function pointers to "get" and "set" the keyword attribute values. The table is used in every stage of the selection evaluation; the graphical browser uses the display string, the lexer uses the regular expression to identify keywords in the selection text, and the evaluation step calls the gettor function to check attribute information. New keywords can be readily added because the symbol table is dynamically built

(a)

```
  set dna_sel [atomselect $molecule_id {segname DNA} ]
  set total_mass 0.0
  foreach mass [$dna_sel get mass] {
    set total_mass [expr $total_mass + $mass]
  }
  puts "The total mass is: $mass"
```

(b)

```
  proc total_mass_proc {selection} {
    eval "vecadd [$selection get mass]"
  }
  puts "The total mass is: [total_mass_proc [atomselect 0 \
    {segname DNA}]]"
```

Figure 2: (a) Script to compute the molecular weight of a segment; (b) rewritten as a function

from function pointers added when the program is run.

The different parts of the evaluation are encapsulated in the VMD script language by the `atomselect` command, which takes as arguments the molecule number, selection text, and optional animation frame number, calls the C++ functions, and creates a new Tcl function. Using one function to construct another one dynamically is one way in Tcl to implement object-oriented programming. The newly created function is specific to the selection and can be called with various parameters to get information about it, such as the original selection text, a list of selected atom indices, and the current frame. More important for analysis, the atom selection object can call the accessor functions from the keyword symbol table to get and set information about the selected atoms, making all the data attributes available to a script.

Analysis of Atom Selections

A major benefit of the VMD script language is that the details of reading the structure data and evaluating its contents are left to the program and the results are made available to the user in a form amenable to analysis. With the atom selections and the standard Tcl commands, a wide range of routines can easily be implemented. As a simple example, Figure 2 (a) computes the molecular weight of the segment named DNA by adding all the mass terms returned by the atom selection. The evaluation of the mass can be written as a function for general use, as shown in Figure 2 (b), though in this case the function was simplified by means of the `vecadd` command.

Several of the more common analysis routines are already built into VMD, such as finding the center of mass of a selection or the RMS deviation between two selections. We plan to include other routines for analysis more specific to molecular modeling, such as finding hydrogen bonds and computing surface and volume terms. Since many specialized analysis programs are developed and made available by other groups, we have found it often easier to use Tcl to communicate with an external process than to rewrite the program. The secondary structure determination used in VMD, for example, is actually performed by calling the program STRIDE [8] . In the future, with the run-time loading feature of Tcl, it will even be possible to add new, compiled routines without recompiling VMD.

User-defined Graphics

The analysis commands described in the previous section use the standard Tcl commands to create text output. Another possibility in VMD is to take advantage of the 3D graphics commands to position various simple geometric primitives such as lines, spheres, cylinders, and triangles in the same display as the visualized molecules. This option was originally included for adding arrows and text to a scene for making images for publication, and as such has been used to draw the unit cell, highlight hinge motions, and display specific bonding patterns. Because of the dynamic versatility of Tcl, the graphics commands have also been used to display the output of other programs including the results of MDToolchest analysis and the surface generated from MS [19] .

When combined with atom selections and the analysis routines, the graphics commands add an exceptionally useful means to test and visualize molecular data in ways not directly implemented in VMD. One proposed visualization method, prototyped using the code in Figure 3 , draws a sphere for each residue with the sphere centered at the residue's center of mass with radius equal to the residue's radius of gyration. The sphere is colored with the color assigned to the residue name.

The combination of routines described have even been used to draw 2D plots. Figure 4 shows the protein bacteriorhodopsin above a C_α-C_α distance plot. The solid protein structure, computations and plot were done in VMD.

Event Call-Backs

Saying that VMD uses a script language is somewhat of a misnomer. Scripts are usually thought of as a sequence of commands which are executed one after another, but Tcl can be used to call functions based on asynchronous events, such as using the mouse to pick an atom. There are two ways to implement these actions in Tcl; we chose to use the `trace` command, which associates a function call-back with specified Tcl variable being read, modified, or deleted. For example, every time an animation frame changes for a molecule the Tcl variable "vmd_frame($molecule_id)" is set to the new frame number, where the molecule identifier is denoted by $molecule_id. The interpreter looks up the functions to be called when that array element changes, and executes each of them.

This type of call-back is used in viewing trajectories where the user-defined graphics may change during the course of the playback. A notable example computes and draws the best-fit line through each of two protein helices that move during a molecular dynamics simulation. A new call-back function was set up so that when the animation frame changed, the best-fit lines were updated, and the angle between the two printed. In that way, the researcher could see that the calculations were correct, visualize the motions of the helices, and get an analytical value for each time step.

Another trace variable, and the only event directly related to the graphical user interface, occurs when an atom is selected with the mouse. Two variables are set, which define the atom index and molecule identifier of the selected atom. The associated user function can act on the event, for instance, by printing extra information about the selected atom, computing some property of the molecule, or modifying the secondary structure of the residue. It has even be used to develop hyperlinks in a molecule: a group of atoms can be associated with a URL, and when one of the atoms is selected, a web browser will be updated accordingly.

Program Availability

The complete, documented source code for VMD and a precompiled binary for IRIX 5.x is available via anonymous ftp from ftp.ks.uiuc.edu. A detailed User's Guide is also available as is a Programmer's Guide for those interested in modifying the code. Please see the VMD home page at
http://www.ks.uiuc.edu/Research/vmd/ for more information.

Conclusion

Molecular modeling software can require the use of many different types of features. Since it is impossible to predict all the combinations of options a user may want, the dynamic nature of an embedded scripting language like Tcl is a useful way to increase the utility and interactivity of a program. Complex code and functions that demand intensive calculations can be written in C or C++ and organized by Tcl so components which are orthogonal in design, such as making atom selections and adding graphical primitives, are tied together to form a synergistic combination.

More importantly, the flexible nature of Tcl along with the modeling extensions added by VMD has provided a way for users to implement new features that were not considered when VMD was developed. With commands to access the internal structure database, new analysis routines are easier to write, especially since the details of reading and determining the inital structure are automatically performed. The results can be presented as text, or displayed as 3D graphics; allowing a wide range of new display methods to be constructed and even animated.

The next stage in VMD development will add more analysis routines, provide support for viewing kinemage files and electron density maps, and develop methods for user interactions. One of the most popular Tcl extensions is the Tk library for writing graphical menus, and adding Tk to VMD will allow others to design menus to handle new tasks, such as graphically modifying the secondary structure definitions. The mouse controls will be expanded to allow for selecting user-defined objects in the graphics display. A major goal of VMD is to develop methods for interactive molecular dynamics, where VMD is used to visualize and alter a system undergoing simulation. The basic interface exists, but there are very few definite ideas on how it should be expanded so the highly extensible and configurable nature of VMD will play a key role in testing the various possibilities.

Acknowledgments

We thank Bill Humphrey for his original design and development of VMD, and everyone else in the Theoretical Biophysics group for offering suggestions and testing out new features. This work is supported by the National Institutes of Health (grant number PHS 5 P41 RR05969-04), and the National Science Foundation (grant number BIR-9318159).

References

1. Roger A. Sayle and E. J. Milner-White. *RasMol: Biomolecular graphics for all*, TIBS 20(Sept):374-376 (1995).
http://www.umass.edu/microbio/rasmol/
2. Quanta, Polygen Corporation. Waltham, MA.
http://www.msi.com/marketing/life/ls_software.html#Quanta

3. A. Nicholls, K. A. Sharp and B. Honig. *Protein Folding and Association: Insights From the Interfacial and Thermodynamic Properties of Hydrocarbons*, Proteins: Structure, Function and Genetics 11:282-290 (1991). http://tincan.bioc.columbia.edu/grasp/
4. Stephen Evans. *SETOR: Hardware-lighted three-dimensional solid model representations of macromolecules*, J. Mol. Graph. 11:134-138(1993).
5. Simon M. Brocklehurst and Richard N. Perham. Protein Science 2:626-639 (1993). http://www.ocms.ox.ac.uk/~smb/Software/N_details/naomi.html
6. A. Sali and T. L. Blundell. *Comparative protein modelling by satisfaction of spatial restraints*, J. Mol. Biol. 234:779-815 (1993). http://guitar.rockefeller.edu/modeller/modeller.html
7. G. Ravishankar and David L. Beveridge. Molecular Dynamics Analysis Toolchest, Department of Chemistry, Wesleyan University.
8. Dmitri Frishman and Patrick Argos. *Knowledge-based secondary structure assignment*, Proteins: structure, function and genetics, 23:566-579 (1995). http://www.embl-heidelberg.de/stride/stride_info.html
9. W. Chang, I. N. Shindyalov, C. Pu, and P. E. Bourne. *Design and Application of PDBlib, a C++ Macromolecular Class Library*, CABIOS 10(6):575-586 (1994). http://www.sdsc.edu/CompSci/pb/pdblib/pdblib.html
10. Christopher W. V. Hogue, Hitomi Ohkawa, and Stephen H. Bryant. *A dynamic look at structures: WWW-Entrez and the Molecular Modeling Database*, TIBS 21: 226-229 (1996). See ftp://ncbi.nlm.nih.gov/pub/mmdb/README for API information.
11. Axel Brünger. **X-PLOR, Version 3.1, A System for X-ray Crystallography and NMR**, Yale University (1992). http://xplor.csb.yale.edu/xplor-info/xplor-info.html
12. John Ousterhout. **Tcl and the Tk Toolkit**, Addison-Wesley (1994). http://www.sunlabs.com/research/tcl/
13. The Python home page is at http://www.python.org/
14. The Perl home page is at http://www.perl.org/perl/
15. Molviewer-ogl is described at http://www.yorvic.york.ac.uk/~mjh/molviewer-ogl/about-molviewer-ogl.html
16. gOpenMol is described at http://laaksonen.csc.fi/gopenmol/gopenmol.htmls
17. Chimera is described at http://www.cgl.ucsf.edu/home/chimera/introduction/
18. W. F. Humphrey, A. Dalke and K. Schulten. *VMD – Visual Molecular Dynamics*, J. Molec. Graphics 14(1):33-38 (1996).

http://www.ks.uiuc.edu/Research/vmd/

19. M. L. Connolly. *Analytical Molecular Surface Calculation*, J. Appl. Crystal. 16:548-558 (1983).

```
proc find_rgyr {sel} {
  # get the center of mass
  set com [measure center $sel weight mass]
  # compute the numerator
  set I 0 ; set M 0
  foreach m [$sel get mass] pos [$sel get {x y z}] {
    set I [expr $I + $m * [veclength2 [vecsub $pos $com]]]
    set M [expr $M +  $m]
  }
  # compute and return the radius of gyration
  return [expr sqrt($I/$M)]
}

### find the sphere parameters for every residue except waters
set sel [atomselect top "not water"]
set molid [$sel molindex]
foreach residue [luniq [$sel get residue]] {
  set ressel [atomselect $molid "residue $residue"]
  ## find the center of mass and radius of gyration
  set com [measure center $ressel weight mass]
  set radius [find_rgyr $ressel]
  ### draw the sphere with the correct color assigned to the residue
  lassign [$ressel get resname] resname
  draw color [colorinfo category Resname $resname]
  draw sphere $com radius $radius
}
```

Figure 3: VMD script to compute the radius of gyration for a given atom selection. $R_{gyr} = (\sum_{all\ atoms} mass(i) \cdot \|\vec{r}(i) - \vec{r}_{com}\|^2)/total\ mass)$

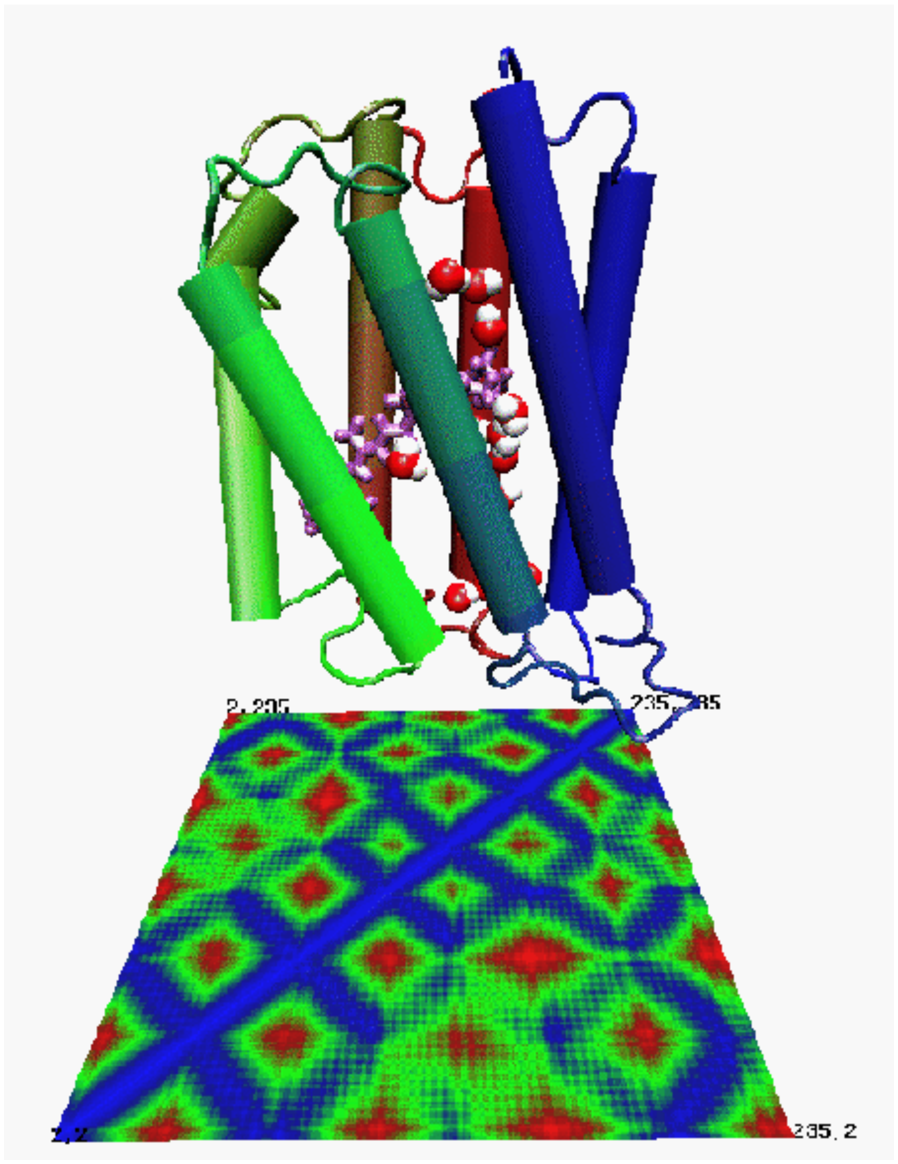

Figure 4: C_α-C_α distance plot of the bacteriorhodopsin protein. The protein visualization, distance calculations, and 2D plot were made in VMD