Chimera: Affordable Desktop Molecular Modeling On Linux Workstations

David Konerding Conrad Huang
Thomas Ferrin
Computer Graphics Laboratory
FAX: (415) 502 1755
University of California San Francisco
San Francisco, CA, 94143
dek@cgl.ucsf.edu, http://www.cgl.ucsf.edu/home/dek

Abstract

Molecular modeling is a well-established tool used in a variety of scientific disciplines, including pharmaceutical chemistry (the development of new drugs) and bioinformatics (the application of information theory to biological sequence analysis) [1]. Molecular modeling is frequently used for planning experiments, analyzing the results of experiments, and for making predictions in situations where technical limitations make experiments difficult or impossible. Recently the Linux/GNU operating system, running on inexpensive desktop PCs using video cards with powerful hardware graphics acceleration, has grown very popular in molecular modeling due to its low cost, low barrier to use, and compatibility with existing UNIX workstations. The Computer Graphics Lab at the University of California San Francisco [2] is developing an advanced modeling application, Chimera [4], to succeed its well-regarded but aging application, MidasPlus [3]. Chimera is written in C++ and Python and emphasizes programmable extensibility as its primary feature. In recognition of the increasing interest in Linux as a molecular modeling platform, we undertook a port of Chimera to Linux. Porting Chimera to Linux represented a number of challenges which had to be overcome; we describe the problems we faced and our solutions.

1 Introduction

For over ten years, the Computer Graphics Lab has distributed a popular UNIX-based molecular modeling application, MidasPlus. The MidasPlus system is used daily in university-level research programs in order to display and manipulate macromolecules such as proteins and nucleic acids. Ancillary programs provide features such as computation of molecular surfaces and electrostatic potentials and generation of publication quality space filling images with multiple light sources and shadows. MidasPlus is distributed with documented source code to serve as a starting point and training tool for others interested in doing their own software development.

Chimera is a new molecular graphics package developed by the UCSF Computer Graphics Laboratory as the next generation of MidasPlus. The primary features of Chimera are high quality real-time molecular graphics and programmable extensibility. The emphasis on extensibility is a direct result of the success of the MidasPlus delegate extension mechanism, which has shown that the ability to incorporate user-written programs greatly enhances the modeling environment. MidasPlus was not originally designed with easy extensibility as a goal, although an extension mechanism allows applications ("delegates") to add functionality to MidasPlus without requiring the developer to recompile the MidasPlus source code.

Chimera has been designed to provide advanced molecular graphics capabilities while retaining the functionality currently available in

MidasPlus. The package has been completely redesigned rather than built upon MidasPlus. Instead of combining the functionality into a single program, we selected a subset of MidasPlus features to form the core of Chimera and then implemented the rest of the features as extensions to the core. The three major groups of core features in Chimera are the graphical display, the user interface, and the extension mechanism.

2 Graphical Display

Molecular modeling applications place a great emphasis on the performance of the workstations graphics subsystem, and Chimera is no exception. Rather than implement a high-performance graphics library from scratch, we chose OpenGL [5], a cross-platform graphics library originally developed by SGI. Chimera supports a number of different methods for displaying molecular structures. Molecular structures are effectively graphs with nodes (atomic positions) and edges (bonds between atoms). Unlike many typical graphs, molecular structures typically occupy 3 dimensions. The default representation is a wireframe model, using OpenGL's line primitive. Chimera can also draw the structure using tubes to represent the bonds, and spheres to represent the atoms. More sophisticated representations include molecular surfaces and volumes. Tubes, spheres, and surfaces are decomposed to triangles. Volumes can be visualized as isocontour surfaces or as true 3D volumes. OpenGL handles line and triangle primitives very efficiently. Recent extensions to OpenGL have added an interface for representing volumes using 3D textures.

The Chimera graphical system is uncoupled from the molecular object management system. Graphical objects are stored in display lists (retained graphical primitives), and the display lists are only rebuilt when the underlying molecular data changes. For example, if the model is rotated but the atom colors are not changed, no molecular data changes (only the viewing transform) and so the display list does not need to be rebuilt. This allows to near-maximal performance from 3D graphics cards, which typically store display lists in card RAM compact form, ready to be written to the graphics pipeline with minimal processing.

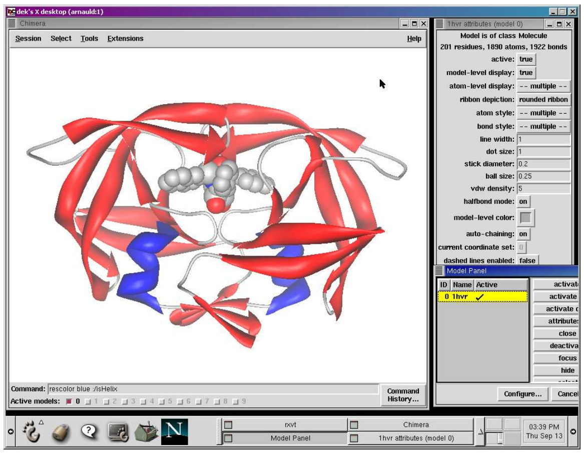

Figure 1: This is an example of a Chimera session. The graphical window is displaying protein model "1hvr" using smooth ribbons and spheres. The ribbons trace the backbone of the protein model while the spheres represent an anti-HIV drug bound at the active site of the protein.

3 User Interface

Chimera is a cross-platform application (Windows and UNIX). To save programmer time and effort, we chose the Tkinter library provided with Python [6] as the base user interface library. Tkinter is a wrapper around the tk [7] library, which provides a cross-platform GUI with native look and feel on the Windows, UNIX and Macintosh platforms. On top of the Tkinter library, we use Pmw [8] and Tix [9], which provide user interface components which can be combined to generate sophisticated dialogs.

4 Extension Mechanism

Molecular modeling has a long tradition of "rapid idea development". Often, a scientist will come up with a new idea, and wants to implement the idea quickly in a scripting language such as Perl or Python to determine whether it is feasible and useful. If the idea is feasible, the implementation is typically recoded in C, FORTRAN, or C++, for performance reasons. We have built a sophisticated extension mechanism which provides support for users to add new features to Chimera without needing to recompile the entire application. The extension mechanism is convenient and can be used to rapidly develop new functionality as needed. Extensions can be written in Python or C++, can access all the molecular data structures provided by Chimera, and use Tkinter to present a GUI for the extension to users. Examples of extensions which have already been contributed to Chimera include the "VolumeViewer" extension [10] for viewing molecular volume data, and the "Collaboratory" [11] which provides synchronized modeling between geographically distant scientists. Although most extensions are written in Python for rapid application development, convenience, and easy cross-platform usage, performance-critical components can be rewritten in C++ and compiled to native object code. An extension which requires C++ is the implementation of the "marching cubes algorithm" [12] which converts molecular volume data into isosurfaces.

Extensions are loaded at runtime using the standard Python mechanism for loading modules: the "import" statement. The Python import statement locates the code which provides the named functionality, either in the form of a Python file or a shared object, and loads it. In the case of a shared object, Python loads the module using the dynamic linker.

Supporting interaction between C++ and Python requires extensive "wrapping" of code. Chimera internally stores all primary molecular data structures as C++ objects, with "shadow" Python objects that have the same attributes and methods as the C++ objects. There are a number of applications which can wrap C functions and C++ objects for use in scripting languages, but our requirement of sophisticated C++ support eliminated all the various wrapping applications available at the time (since that time, better C++ wrappers have become available, and we intend to determine whether they satisfy our requirements). Our wrapping application "wrappy" is part of the freely available OTF ("Object Technology Framework") [17].

An example extension, which hides some of the atoms in a protein molecule based on their name, follows. It is clear from the code that Python deserves its description of "executable pseudocode"; a programmer who does not know Python or anything about molecules could quickly pick up the gist of the algorithm.

```
# Import system modules used
# in this example.
import re

# Import Chimera modules used
# in this example.
import chimera

# Define a regular expression
# for matching the names of
# protein backbone atoms (we
# do not include the carbonyl
# oxygens because they tend to
# clutter up the graphics
# display without adding much
# information).


MAINCHAIN = re.compile(\
'^(N|CA|C)\$', re.I)

# Do the actual work of
# setting the display status
# of atoms and bonds.  The
# following 'for' statement
# iterates over molecules.
# The function call
# 'chimera.openModels.list(\
# modelTypes=[chimera.Molecule])'
# returns a list of all open
# molecules; non-molecular
# models such as surfaces and
# graphics objects will not
# appear in the list.  The
# loop variable 'm' refers to
# each model successively.

for m in chimera.openModels.list(\
modelTypes=[chimera.Molecule]):

# The following 'for'
# statement iterates over
# atoms. The attribute
# reference 'm.atoms' returns
# a list of all atoms in model
# 'm', in no particular order.
# The loop variable 'a' refers
# to each atom successively.

    for a in m.atoms:

# Set the display status of
# atom 'a'.  First, we match
# the atom name, 'a.name',
```

```
# against the backbone atom
# name regular expression,
# 'MAINCHAIN'. The function
# call
# 'MAINCHAIN.match(a.name)'
# returns an 're.Match' object
# if the atom name matches the
# regular expression or 'None'
# otherwise.  The display
# status of the atom is set to
# true if there is a match
# (return value is not 'None')
# and false otherwise.

        a.display = \
MAINCHAIN.match(\
a.name) != None

# By default, bonds are
# displayed if and only if
# both endpoint atoms are
# displayed, so therefore we
# don't have to explicitly set
# bond display modes; they
# will automatically "work
# right".
```

5 Challenges associated with Chimera on Linux

Before undertaking the port of Chimera to Linux, we had to satisfy the software requirement restraints presented by Chimera. We selected the Red Hat 6.2 distribution of Linux [13] because it was stable, reliable, well-supported and commonly used. We believe that other distributions such as Debian would also be acceptable candidates for building Chimera, but due to lack of time resources we did not try them. We have since upgraded from Red Hat 6.2 to Red Hat 7.1, which has better built-in support for OpenGL.

Python, OpenGL, and Tcl/Tk are available on Linux so these did not present challenges; we simply compiled and installed these applications from their primary source distributions. The initial port of Chimera to Linux used Python-1.5.2, Mesa-3.2 (for OpenGL, [14], and Tcl/Tk 8.0. More recent ports of Chimera use Python-2.1, Mesa-3.4, and Tcl/Tk 8.3. Because Mesa-3.4 conforms to the Linux OpenGL ABI, any ABI-compliant implementation of OpenGL for Linux will work with Chimera. For example, the nVidia GeForce2 graphics card requires a vendor-supplied OpenGL driver implementation; Chimera works just fine with this driver.

All C++ code in Chimera is written in standard C++, making heavy use of the standard C++ library, and the object-oriented/generic programming features of C++. The primary challenge in porting Chimera to Linux was to find a standard C++ compiler and library. The C++ compiler (egcs) provided with Red Hat 6.2 was not sufficient; it lacked some required C++ template features and did not have a standards-compliant C++ library. This is understandable because the C++ standard library was finalized only recently; egcs complied to a draft version of the standard and significant changes were made in the published standard. Further, templates are a sophisticated (and very recent) feature of C++ which requires extensive compiler and linker support.

After experimentation with several different C++ compilers available for Linux, we settled on GNU g++ [15] version 2.95.3. Version 2.95.3 supports all of the language features we use, although its standard C++ library is not complete. Even where the library calls are available, there are bugs in the implementations which appear only when the library is used extensively. After looking at several different (and expensive) commercial libraries, we found STLport [16], a freely available implementation of the standard C++ library derived from the SGI STL. STLport is a high-quality implementation of the library, and in addition to supporting GNU g++, it supports 25 other C++ compilers. We are currently using STLport-4.5.

Once we had settled on a platform and required toolset and libraries, the port of Chimera to Linux was straightforward. We simply modified our Makefiles to use g++ and STLport, and compiled the entire application with no major modifications. g++ combined with STLport actually found several lines of non-compliant C++ code (lack of std:: namespace qualifier where required and illegal arguments to standard library functions) which were not flagged by the UNIX compilers (Compaq C++ 6.3 and MIPSPro 7.3.1) and Windows compiler (MetroWorks CodeWarrior) we use.

Although our experience with libstdc++ (the standard C++ library implementation included with g++) was somewhat negative, we are glad to report that the libstdc++ developers fixed every bug that we reported (we always isolated the bugs to simple testcases, typically only several lines long) and prospects of having a complete, reliable libstdc++ in the future are promising.

An interesting problem that we only noticed on Linux was related to the usage of dynamic loading of modules with "dlopen" when the main application (for example, Python) is linked with the C compiler and the modules are compiled with the C++ compiler. When Python imports a second module linked against the standard C++ library and uses certain features (such as exceptions combined with ostringstreams), the application will core dump. Close inspection of debugging output from the dynamic linker as well as the linker's source code showed that when the second library was loaded, the standard C++ library was loaded a second time, rather than having its symbols resolved by the first module's linked standard library. This caused problems with the internal state of the library which caused a core dump when a floating point number was output to a stringstream. There are several "work-arounds", none of which is completely satisfactory. One is to modify the arguments to dlopen() in the Python module loading code to allow global symbol resolution, but the authors of Python highly discredit this approach. Another workaround is to force the standard C++ shared library to be loaded before the main program starts using the LD_PRELOAD environment variable; further loads of the library always use the initially loaded library. A third option is to explicitly link Python against the standard C++ library when compiling. We are currently using the second option because this allows Chimera to use the distribution-installed python without any modifications The second option was causing core dumps on application exit). We have determined that the core dump was actually due to mistaken linking of one of Chimera's subsidiary libraries against the STLport static library. Later, the chimera shared library was linked against the subsidiary library as well as the dynamic STLport library. Therefore, there were two copies of the standard library resident in the program and when objects were passed between the main Chimera application and the subsidiary library some memory was likely corrupted (the standard library maintains some internal state for its objects, and there were two copies of the internal state). When the proper linking was applied, the segmentation fault on exit went away.

6 Performance of Chimera on PC Workstations with Hardware Accelerated Graphics

Over the past few years, PC hardware has improved tremendously in performance and is now reaching the level of performance which normally would only be available in high-end Unix workstations. This performance increase is particularly visible in the 3D graphics area. In the past, graphics cards for PCs provided a "dumb 2D frame buffer" and possibly a few primitive accelerated operations for drawing rectangles and filled regions. These cards were sufficient for accelerating windowing systems like Microsoft Windows and the X Window System. However, demanding 3D applications such as "first person shooters" have driven the graphics card industry to add support for hardware accelerated 3D graphics ("HA3G"). HA3G has existed for some time on high-end Unix workstations from SGI, HP, Digital, and IBM. Early HA3G on PCs was very limited; initial cards like the Voodoo mainly provided fast rasterization (writing of texture pixels into the frame buffer) but did not provide support for transformation and lighting in hardware. Whenever a 3D point is to be plotted on a 2D screen its 3D coordinates must be transformed into the appropriate 3D space and then projected to 2 dimensions. These calculations are normally done in single precision floating point. For models with very large numbers of points, this puts an excessive burden on the FPU of the system's CPU, leaving much less time for the application. For some time, high-end graphics cards in the PC have had hardware transformation and lighting. New low-cost HA3G cards, such as the nVidia GeForce2 and ATI Radeon, have made hardware transformation and lighting a standard feature. This allows for significantly higher polygon counts for model objects, which equates to greater on-screen realism.

Molecular modeling typically represents a molecule as a collection of sticks (the chemical bonds) and balls (the atoms). While small molecules such as ethanol (9 atoms) do not place a large demand 3D graphics system, “interesting” molecules such as proteins and DNA are much, much larger (often 10,000 or more atoms). To model these systems in real-time absolutely requires HA3G, and the better the HA3G, the larger the system that can be modeled. Further, enhanced representations such as backbone ribbons, spheres, and surfaces can be used.

There is a great deal of interest in replacing older Unix workstations with modern PCs with HA3G. One important decision when replacing these systems with new PCs is what operating system will run. This decision is driven by several forces, including what software will be run, and what level of performance is required. Although porting legacy Unix applications to Windows is far more work than porting to Linux, if the same software runs on both Microsoft Windows and Linux, the decision will be based on the performance of the software on the given platform. With this in mind, we undertook a study to measure the performance of Chimera on the same machine (AMD 1GHz, 256MB DDR-SDRAM, nVidia GeForce2 MX 32MB DDR RAM) running both Windows 2000 Profession and Red Hat 7.1. The GeForce2 MX graphics card has hardware transformation and lighting and is very inexpensive- less than $100. It has drivers for both Linux and Windows 2000, and these drivers share much of the same code. Essentially, the drivers share all the card-driving code but have different interface code for the operating system. In this study we controlled for the system hardware but allowed the operating system/window display code to vary. Also, because Chimera was compiled with Visual C++ on Windows and g++ on Linux that code varies as well. One possible improvement on the study would be to compile Chimera with g++ and STLport on Windows, which would be possible using MingWin, which would control for the generated code.

The question we asked was not “what is the highest frame rate achievable with a given workload”, but rather, “can we achieve a target frame rate with a given workload”. The minimum acceptable frame rate in molecular modeling is about 20 frames per second, below which interactive manipulation becomes noticeably jerky. The workload in this case is a moderately sized protein (HIV protease, PDB model “1hvr”). We varied the representation between wireframe, ball and stick, surfaces, and spheres. Wireframe is typically the fastest while spheres are the slowest, based on the number of polygons to produce the representation.

Repr	Windows	Linux
wireframe	50	45
ball/stick	12	26
surface (d=2)	12	17
surface (d=10)	6	5
sphere	3	7

Table 1: Relative Performance, in frames per second of Windows 2000 Professional and Red Hat Linux 7.1 on a Chimera Benchmark. The hardware was a 1GHz Athlon, A7M266 motherboard, with 256MB DDR-SDRAM and a nVidia GeForce2MX (32MB SDRAM). Performance is measured in frames/second for the given representation. The PDB (Protein Data Bank) “1hvr” (HIV Protease) crystal structure was used.

As can be seen by the table, in most cases the performance was fairly similar between Windows and Linux, but in certain cases, Linux did significantly better. We do not know the actual cause of this performance disparity, since it does not appear to be systematic across all representations. Further, we note that except for wireframe and ball and stick representations the performance was less than acceptable. This is not too surprising given the fact that the card used for the experiments is designed for games and costs less than $100. Better cards such as the nVidia Quadro2 and FireGL2, both of which are supported under Linux, give significantly better performance than the GeForce2 MX. The model used in this study was not particularly large, and recently, structural biologists have determined biologically significant structures 100 times larger. For facile manipulation of these molecules with wireframe representations, high-end graphics cards are an absolute necessity.

7 Conclusions

A modern Linux distribution represents a powerful development platform, capable of compiling and executing a sophisticated scientific application developed using a modern software language. Minimal changes were required to compile Chimera, allowing the Chimera developers to focus their efforts on producing their software, rather than debugging their build environment. We especially appreciated that g++ and STLport helped us find programming errors which were not flagged by other C++ compilers. When developing sophisticated C++ applications that make use of dynamic linking, it is important to understand the behavior of the runtime linker, especially with respect to static constructors.

References

[1] T.E. Ferrin and T.E. Klein, "Computer Graphics and Molecular Modeling," in: The Encyclopedia of Computational Chemistry, P.v.R. Schleyer, N.L. Allinger, T. Clark, J. Gasteiger, H.F. Schaefer III, and P.R. Schreiner, eds., pp. 463-474, John Wiley and Sons, 1998.

[2] http://www.cgl.ucsf.edu

[3] http://www.cgl.ucsf.edu/Outreach/midasplus/index.html

[4] http://www.cgl.ucsf.edu/chimera

[5] http://www.opengl.org

[6] http://www.python.org

[7] http://dev.ajubasolutions.com

[8] http://pmw.sourceforge.net

[9] http://tixlibrary.sourceforge.net

[10] http://www.cgl.ucsf.edu/Research/chromosome

[11] http://www.cgl.ucsf.edu/Research/collaboratory

[12] William Lorensen, Harvey Cline. Computer Graphics, Volume 21, Number 4, July 1987, pp. 163-169.

[13] http://www.redhat.com

[14] http://www.mesa3d.org

[15] http://gcc.gnu.org

[16] http://www.stlport.org

[17] http://www.cgl.ucsf.edu/Research/otf