Acta Crystallographica Section D
Biological Crystallography
ISSN 0907-4449

The *CCP*4 molecular-graphics project

Elizabeth Potterton,[a]* Stuart McNicholas,[a] Eugene Krissinel,[b] Kevin Cowtan[a] and Martin Noble[c]

[a]Structural Biology Laboratory, Department of Chemistry, University of York, Heslington, York YO10 5DD, England, [b]European Bioinformatics Institute, Hinxton, Cambridge CB10 1SD, England, and [c]University of Oxford, England

Correspondence e-mail: lizp@ysbl.york.ac.uk

Received 29 May 2002
Accepted 28 August 2002

This new package will provide easy-to-use access to crystallographic structure solution, model building and structure analysis. It will be possible for any developer to integrate scientific software into the system.

1. Introduction

The Collaborative Computational Project, Number 4 (CCP4) is a collaboration for developing and distributing software for macromolecular crystallography (Collaborative Computational Project, Number 4, 1994). The products of the collaboration cover the full range of crystallographic procedures from initial data analysis to structure refinement, but excluding any molecular-graphics-based model building. Programs such as *O* (Jones *et al.*, 1991), *Xtalview* (McRee, 1999) and *QUANTA* (Accelerys) provide this functionality.

In the context of a meeting on high-throughput crystallography, is molecular graphics, which assumes a non-automated user, relevant? Molecular graphics does have its uses: appropriate visualization tools can occasionally help difficult structure solutions, suitable visual tools certainly help in model building and model correction, visualization of analysis and comparison information is essential for any biological interpretation of a structure and presentation of results is greatly aided by graphics.

*CCP*4 distributes a library of basic software tools to assist scientific developers with reading and writing the common file formats, parsing command input and handling symmetry. These software libraries are currently being revised and extended (Winn *et al.*, 2002). One view of the molecular-graphics project is that it will be a library to assist developers who wish to present their functionality to users *via* a molecular-graphics interface. Access to a graphics system can also be helpful in the process of developing more automated procedures, as it enables developers to visualize test systems even if they do not use the graphics in the same way in their final released software.

Our objective is to provide an open-source package which integrates well with *CCP*4 and other crystallographic software. Tools for analysis and comparison of the resultant structures will be incorporated and smooth interfacing with popular web-based tools and databases will be essential. The package should also be useful for dissemination of information, providing for generation of pictures and presentations, and it should be easy to use for those outside the crystallographic field. It should be possible for any programmer to integrate their scientific functionality into the molecular-graphics system.

2. Program design

There are several key principles for the program design. Firstly, modularity of the software, which simplifies writing and maintaining the software and makes it feasible to make major revisions to individual modules when necessary. Secondly, provision of libraries of commonly used functionality. Thirdly, object-oriented programming languages can simplify handling complex hierarchical data such as a protein structure, the full results of a crystallographic data collection or the range of graphical objects displayed in molecular graphics. Finally, scripting languages can be more appropriate than compiled languages for some areas of the code.

2.1. The data-handling libraries

The core of the molecular-graphics package handles two key types of data: crystallographic experimental data and macromolecular structure data. There have been major initiatives within *CCP*4 to develop libraries to hold these data and provide basic tools to manipulate and analyse the data. These libraries will simplify and speed up development of scientific software. They have been written in C++ and the object-oriented programming approach has simplified the handling of the complex hierarchical data. The libraries are already available to developers and have extensive documentation and examples. The libraries are being used to develop the molecular-graphics package and the molecular-structure library is being used within the European Bioinformatics Institute to support their macromolecular-structure database.

The Clipper library (Cowtan, 2002) is designed to handle consistently all experimental data including multicrystal and multiwavelength data and to handle maps and interconversion between real and reciprocal space. It also provides sophisticated data-analysis tools and mechanisms to import and export data from external files.

The Macromolecular Database (MMDB; Krissinel, 2002) holds structure data and provides tools to perform common tasks. MMDB can read and write data files in PDB or mmCIF format and stores the data in a hierarchical structure with four levels for the model, chain, residue and atom. The model level is necessary to handle the multiple models derived from NMR experiments. MMDB has tools to select subsets of atoms (or any other object in the hierarchy) based on geometric criteria such as closeness to a defined atom or based on a selection command specified by the user. The selection language supported by MMDB is terse but very flexible; for example, a Ca atom in residue 27 of chain *A* is written as 'A/27/CA'. The idea of using a forward-slash field separator is based on computer-file systems which should be familiar to most crystallographers. Further examples of the syntax are given in Table 1. Atom selection is very important within a molecular-graphics system and ideally the user should be able to control what is displayed *via* simple options on a GUI. The MMDB selection tools provide a powerful generic mechanism to support the popular options presented by the GUI and the user only needs to use this syntax for entering specific customized selections.

Table 1
Examples of atom-selection syntax supported by MMDB.

A specific atom	A/27/CA
A range of residues	A/27-30
All atoms in a specified residue type	A/(GLY)/*
A list of atom names	A/27/CA,N,C,O
All atoms of a given element type	[N]

In order to refine macromolecular structures, programs such as *REFMAC* (Murshudov *et al.*, 1997) need to have constraints on the internal geometry of the structure. The ideal internal geometry (bond lengths, bond angles, torsion angles, atoms constrained to a plane) and estimated standard deviations are saved in a reference file which contains data for amino acids, nucleic acids, sugar monomers and ligands. The same information can also be useful in molecular graphics; for example, the definition of ideal bond lengths tells you which atoms are bonded and this knowledge can be used to derive and display the correct bonds. This is more reliable than the alternative approach, which is to assume that atoms closer than some cutoff distance are bonded. The MMDB library maintains the ideal internal geometry information and provides tools for the application programmer to query: for example, the ideal bond lengths within a given residue in an imported structure. In order to use the internal geometry data, the residues and ligands within an imported structure must be recognized and cross-referenced to the database; this is straightforward for commonly occurring amino acids and nucleic acids which have standard residue names. Identifying non-standard amino acids or non-standard nucleic acids or any ligands within an imported structure is harder, but an efficient mechanism is being developed. The method uses graph-theory techniques which represent each residue or ligand as a graph with nodes representing atoms (with the property of element type) and edges representing bonds. There is a very efficient fast algorithm to match this representation of an unknown residue or ligand in an imported structure against a database of known residues and ligands.

2.2. The scripting language component

The data-handling libraries in the molecular-graphics system are written in C++, but the framework of the system is written in the scripting language Python which is generally faster to code and provides easy access to operating-system tools such as threads and sockets. The overall management of the loaded data and the derived graphical objects is implemented in Python and uses an object-oriented approach. The definition of the content of the graphical user interface (GUI) and handling of all user input is performed in the Python layer. It is possible to access all of the functionality of the C++ libraries from Python, but this requires a thin layer of interface code which can be generated automatically by packages such as *SWIG*. Possibly the main benefit of Python will be in enabling faster prototyping of scientific functionality.

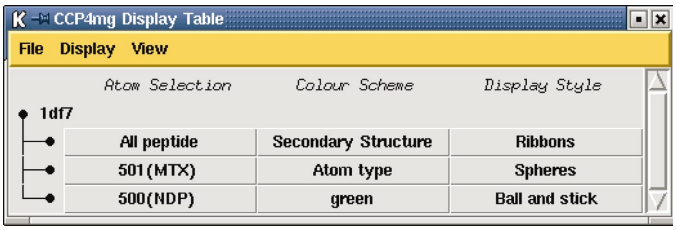

Figure 1
The display table after the user has read data from the file 1df7.pdb and set up a display with three graphical objects: the peptide, coloured by secondary structure and displayed as a ribbon, the ligand MTZ, coloured by atom type and displayed as space-filling spheres, and the cofactor NDP, coloured green and displayed as ball-and-stick.

2.3. The graphics and graphical user interface

The molecular-graphics package uses two major external tools: the OpenGL graphics library for three-dimensional graphics and Tcl/Tk for the GUI. The interface to both of these tools is intentionally minimal and *via* an abstract representation of the objects to be drawn. This approach will mean that the package is not strongly tied to either OpenGL or Tcl/Tk. In the case of the three-dimensional graphics the object to be drawn is first defined in terms of graphical primitives such as vectors, spheres and cylinders. An interpreter, which only needs to be a relatively small piece of code, converts this abstract representation to the appropriate OpenGL library calls to display the object. With this approach, it will be relatively straightforward to provide alternative interpreters to other graphics libraries such as the Mac OS X native library or to a Postscript library to generate Postscript output.

Similarly the definition of the GUI content is independent of the underlying graphical toolkit. The GUI interpreter uses Tcl/Tk and is based partly upon CCP4i, the graphical user interface to the *CCP*4 package. The GUI interpreter runs as a separate process connected to the main process *via* sockets. The actual content of each interface window is defined in general terms which are independent of Tcl/Tk, in the format of nested lists within the Python language. The definition of the window is sent to the GUI interpreter, which transforms the generic definition into Tcl/Tk commands and displays the window. User input to the GUI triggers sending a command in Python format to the main process.

3. Current status

The present early version of the graphics program will read in and display multiple PDB files. By default, one graphical object is created and displayed for one imported molecule. The three key properties of the graphical object are the atom selection, the colour scheme and the display style. By default the atom selection is 'all atoms', the colour scheme is to colour according to 'atom type' and the display style is 'bonds'. The imported molecules and its child graphical objects are listed in the GUI display table. The display table enables the user to quickly change the properties of the graphical object *via* menus, or the user can create new objects with different properties, so the display table shown in Fig. 1 will give the display shown in Fig. 2.

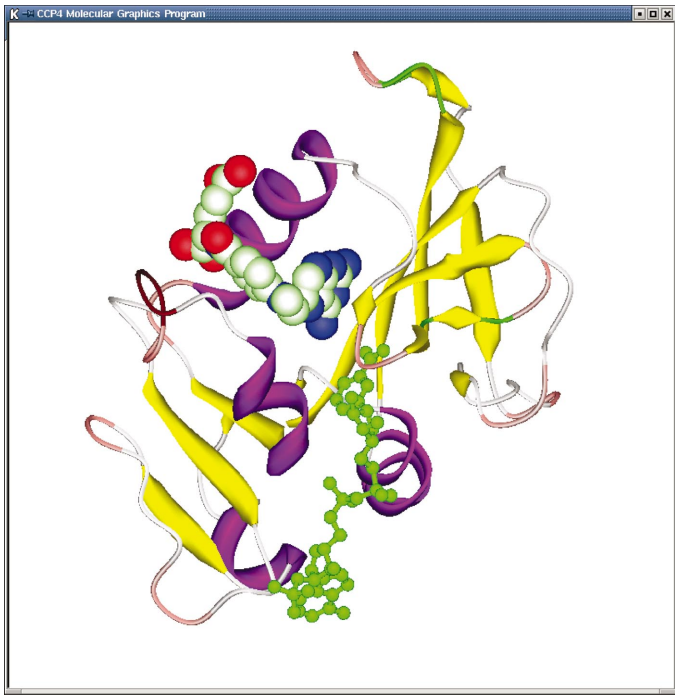

Figure 2
The molecule-viewer display corresponding to the display-table definition given in Fig. 1.

The content of the imported PDB file has been analysed and the information on peptide chains, nucleic acid strands, solvent and small molecules used to customize the display table options to be appropriate for this molecule.

It is intended in the short term to add display of electron-density maps and some simple functionality to show the result of analysis after a cycle of structure refinement using *REFMAC*, particularly to highlight poorly fitted regions. Following on from this, tools to correct the poorly fitted regions will be developed and the ultimate objective is to automate the whole refine, analyse and rebuild cycle.

References

Collaborative Computational Project, Number 4 (1994). *Acta Cryst.* D**50**, 760–763.
Cowtan, K. D. (2002). *Clipper Libraries*, http://www.ysbl.york.ac.uk/~cowtan/clipper/clipper.html.
Jones, T. A., Zou, J.-Y., Cowan, S. W. & Kjeldgaard, M. (1991). *Acta Cryst.* A**47**, 110–119.
Krissinel, E. (2002). *CCP4 Coordinate Library Project*. http://www.ebi.ac.uk/~keb/cldoc/.
McRee, D. E. (1999). *J. Struct. Biol.* **125**, 156–165.
Murshudov, G. N., Vagin, A. A. & Dodson, E. J. (1997). *Acta Cryst.* D**53**, 240–255.
Winn, M., Ashton, A. W., Briggs, P. J., Ballard, C. C. & Patel, P. (2002). *Acta Cryst.* D**58**, 1929–1936.