research papers

Developments in the *CCP*4 molecular-graphics project

Liz Potterton,[a]* Stuart McNicholas,[a] Eugene Krissinel,[b] Jan Gruber,[c] Kevin Cowtan,[a] Paul Emsley,[a] Garib N. Murshudov,[a] Serge Cohen,[d] Anastassis Perrakis[d] and Martin Noble[c]

[a]Department of Chemistry, University of York, England, [b]European Bioinformatics Institute, Cambridge, England, [c]Department of Biochemistry, University of Oxford, England, and [d]NKI, Amsterdam, The Netherlands

Correspondence e-mail: ccp4mg@ccp4.ac.uk

Progress towards structure determination that is both high-throughput and high-value is dependent on the development of integrated and automatic tools for electron-density map interpretation and for the analysis of the resulting atomic models. Advances in map-interpretation algorithms are extending the resolution regime in which fully automatic tools can work reliably, but at present human intervention is required to interpret poor regions of macromolecular electron density, particularly where crystallographic data is only available to modest resolution [for example, $I/\sigma(I) < 2.0$ for minimum resolution 2.5 Å]. In such cases, a set of manual and semi-manual model-building molecular-graphics tools is needed. At the same time, converting the knowledge encapsulated in a molecular structure into understanding is dependent upon visualization tools, which must be able to communicate that understanding to others by means of both static and dynamic representations. *CCP4mg* is a program designed to meet these needs in a way that is closely integrated with the ongoing development of *CCP*4 as a program suite suitable for both low- and high-intervention computational structural biology. As well as providing a carefully designed user interface to advanced algorithms of model building and analysis, *CCP4mg* is intended to present a graphical toolkit to developers of novel algorithms in these fields.

1. Introduction

The first implementations of interactive molecular graphics have been central to the development of macromolecular crystallography. Pioneering work in *FRODO* (Jones, 1978) and its successor *O* (Jones *et al.*, 1991) have to a large extent established the protocols for macromolecular model building and have defined the way that generations of crystallographers build molecular models. Innovative implementations of interactive model-building software, such as *XtalView* (McRee, 1999), *MAIN* (Turk, 2001), *QUANTA* (Accelrys, San Diego, CA, USA) or *TURBO FRODO*, have provided powerful tools and have gained a faithful audience.

Once a structure is finalized, the need for convincing illustrations has powered the development of software such as *Molscript* (Kraulis, 1991) and *Ribbons* (Carson, 1997) that, in addition to implementing publication-quality graphics, have introduced many concepts on how a structure is illustrated. Other software, such as *Raster3D* (Merritt & Bacon, 1997), *Bobscript* (Esnouf, 1999), *Spock* (Christopher *et al.*, 1996) and *Dino* (http://www.dino3d.org), have contributed their share of ideas within powerful implementations. Ideas such as the representation of surfaces and the mapping of residue properties onto them have been pioneered in *GRASP* (Nicholls *et al.*, 1991). There are more recent modern implementations of

macromolecular visualization in packages such as *PyMol* (DeLano, 2002).

Parallel to the development of such crystallographically oriented software, other graphic packages have addressed the need for analysis tools. While naturally providing capabilities for good-quality illustration, their focus has been on the ease of use for the non-expert. *RASMOL* pioneered this philosophy, while initiatives such as *Swiss-PdbViewer* (Guex & Peitsch, 1997), *Chime* (Martz, 2002), the *AstexViewer* (Hartshorn, 2002) and others have also made a significant contribution.

However, the crystallographer's toolkit for visualization, manipulation and analysis of molecules remains diverse: apart from the existence of personal preferences, most crystallographers will use different software for initial model building, rebuilding during refinement, illustration of the fold and illustration of atomic details with or without electron-density maps. Moreover, the time-consuming model-rebuilding process that by nature is coupled to structure refinement has never been tightly integrated with refinement software. The *CCP4mg* project was set up with the ambitious goal of addressing the need for an all-inclusive package that will suit both the needs of the molecular biologists who want to look at and analyse and illustrate macromolecular structures, and crystallographers, novice and expert, who want to manipulate and build models using electron-density maps. Finally, given the dramatic increase in computing power and the birth of automated building algorithms, *CCP4mg* aims to provide an ensemble of semi-automatic tools while keeping manual functionality.

2. General design and implementation

2.1. The data-handling software libraries

The core of *CCP4mg* is written in a combination of C++ and Python. C++ libraries handle the data and provide the computer-intensive tools for applications. Python is used to provide the high-level program control and to define and handle input from the GUI. To handle model and experimental data, *CCP4mg* uses two libraries which have been developed for general use in *CCP*4 programs and which are freely available for use by any developer.

CCP4mg is heavily based on MMDB (MacroMolecular DataBase), a C++ library for handling model data (Krissinel *et al.*, 2004). This library provides tools to read and write PDB or mmCIF format files with systematic tools for the application programmer to access and edit the data. It also handles generic structure-based data and provides powerful atom-selection tools.

The experimental data is handled by the Clipper library (Cowtan, 2003), which provides tools to either read electron-density maps or to read reflection data and then calculate maps.

Creating a Python interface to C and C++ libraries requires extensive code to interface every element of functionality. Fortunately, there are several packages that will autogenerate the required code. For *CCP4mg* the interface between C++ and Python is autogenerated by the *SWIG* package (D. Bleazley; http://www.swig.org). The exported interface to a C++ library is a Python module that can be imported into any Python script. Our approach to using this is that the bulk of the data are held in C++ library objects, with Python generally only holding minimal handles to the top-level data objects and calling the appropriate C++ methods to apply the required operations. We have observed that moving large quantities of data through the interface between C++ and Python is slow, but we normally only need to move the small amounts of data needed for presentation on the GUI or for setting algorithm options.

2.2. Program control and the GUI

The *CCP4mg* program consists of three main threads. One thread is the core section of the program, which holds the main data structures and applications. This thread also responds to input from the various other subprocesses and dispatches commands to them. Secondly, there is a thread that draws the graphics window. This thread does no more than draw objects and receive input from the graphics window, which is then sent to the main thread. Finally, there is a thread that responds to GUI input.

The task of developing a GUI or a graphics system can be greatly simplified by building it using a third-party toolkit. One problem in developing a package that we hope will have a long life is that when new and better toolkits become available converting a package to use the alternative toolkit can be extremely difficult. We have attempted to 'future-proof' *CCP4mg* by clearly modularizing the interface to GUI and graphics toolkits. In the case of the GUI, each GUI window is defined in the Python code as a Python nested list in which each element consists of a pair of items: a label defining the type of data and the data themselves.

The GUI renderer is the Tcl/Tk toolkit (Tcl Developers; http://www.tcltk.org) with extensions originally developed for the *CCP4i* package (Potterton *et al.*, 2003). The compatibility with *CCP4i* means that the two packages have a similar look and feel and that appropriate developments in either package can benefit the other. The Tcl interpreter is run embedded within the main Python interpreter process with socket communication between the two. An outline of the basic GUI mechanism is as follows: the GUI window is defined in the Python code as a nested list and the GUI handling code in the main process converts the definition to a text definition of a Tcl nested list and sends this definition to the GUI renderer, which interprets the definition and draws a window.

3. Graphical capabilities

3.1. Atomic model representation

Molecules may be represented in many ways. Typically, line models may be drawn on paper or balls and sticks used to build physical models. *CCP4mg* provides a very simple menu system to select from different graphical representations. The

options available are lines, ball and stick, solid cylinders, space-filling spheres and the ribbon cartoons used to represent different secondary-structure types.

CCP4mg offers the ability to colour any representation type in a variety of different ways. One may colour by atom type, residue type, chain, position in chain, secondary-structure type, physical property (for example, temperature factor, solvent-accessible surface area) and many others. Custom colouring schemes may also be defined.

Multiple different sets of selected atoms may be defined. This may be performed using a varied choice of predefined selections such as all peptide atoms, C^{α} atoms, hydrophobic residues, a particular chain *etc*. Specific selections of atoms or residues may also be defined using the selection language.

Using the combination of different selection sets represented in different styles and colours, varied and detailed illustrations of molecular structure may very quickly and easily be built up.

3.2. Maps

Electron-density maps are displayed as the usual chicken-wire isosurfaces. The contouring level can be changed in real time. The map is 'continuous crystal'; that is, the user can move as far as they like in any direction through the crystal and the appropriate section of map will be displayed.

4. Plug-in applications

It will be possible for developers to incorporate a scientific application into *CCP4mg* without needing extensive knowledge of the program or the existing code. An application programmers' interface (API) will be provided with documentation and working examples. The API will enable an application programmer to define a GUI, access and edit the loaded model and experimental data, control the appearance of the displayed objects and generate novel graphical objects.

Application programmers will need to write Python code to define the GUI and handle command input from the GUI. The core of their application could be written in Python, C or C++ and will need to use the MMDB and Clipper libraries to access the model and experimental data, respectively. An extremely useful feature of Python is that Python modules, including the interfaces to C or C++ libraries, can be imported into a Python program at run-time. This means that the core *CCP4mg* program and applications can be distributed separately in executable form and a user could install a plug-in application without needing to rebuild the package.

5. Analysis tools

5.1. Secondary-structure matching (SSM)

CCP4mg has an application to superpose multiple structurally homologous protein structures (Krissinel & Henrick, 2004), requiring no user assistance in specifying equivalent features in the structures. The geometrical equivalencies between the structures are established to a first approximation by matching graphs built on a vectorial representation of secondary-structure elements (Mitchell *et al.*, 1989). A specially designed graph-matching procedure looks for pairs of secondary-structure elements (SSEs) that have a close relative orientation and separation distance in both graphs. Identification of geometrically equivalent SSEs allows a rough structure superposition (a rotation search is employed if only one SSE pair is matched), which is used as an initial guess for matching individual residues, represented by backbone C^{α} atoms. The matching procedure is based on analysing the distances between C^{α} atoms of superposed structures and is aimed at optimization of a quality function balancing the r.m.s.d. and the number of matched C^{α} pairs. The structure orientation is then refined by least-squares superposition of the matched C^{α} atoms and the cycle 'matching-superposition' is iterated until convergence is reached. This method will superpose both highly homologous structures and robustly superpose dissimilar structures with only a few equivalent SSEs. The *CCP4mg* interface enables the user to select limited regions of the protein for superposition and will display both the best match and alternative sub-optimal matches.

5.2. Surface properties

Macromolecular-surface representation plays a key role in both the analysis and interpretation of molecular structures, particularly when enhanced by the mapping of properties onto that representation. These properties may simply be characteristics of the underlying residues (*e.g.* the extent to which a part of the surface is conserved among homologous proteins), but may also be aspects of the molecular field that the molecule presents to the environment (*e.g.* the electrostatic field). Within *CCP4mg*, we will provide a comprehensive set of tools for the representation of protein surfaces that will allow flexible mapping of molecular properties onto that surface, including sequence and structural conservation as well as 'molecular-field' properties such as electrostatic and hydrophobic potentials. The program will also present tools for the comparison of molecular surfaces from homologous proteins or single proteins in multiple conformations, a challenge that has so far received little attention.

5.2.1. Electrostatic potential. Analysis of the electrostatic potential on protein surfaces has been highly illuminating in the understanding of ligand binding (especially nucleic acids), catalysis and electrostatic steering. Electrostatic potential calculations are achieved in practice by solving the Poisson–Boltzmann equation,

$$\nabla \varepsilon(r) \nabla \varphi(r) - \rho_{\text{macro}}(r) + \sum_i q_i n_i^0 \exp[-q_i \varphi(r)/kT].$$

This equation describes the propagation of electrostatic potential through a medium in which the dielectric constant varies and in which counterions are free to diffuse. Solution is generally achieved through an iterative finite-difference approach, with the local assignment of dielectric constant based upon the solvent-excluded surface of the protein. We will implement finite-difference Poisson–Boltzmann electrostatic potential calculations in *CCP4mg*, taking advantage of

the atomic typing of *CCP4mg*, and applying published numerical methods for solving the equation.

5.2.2. Hydrophobic potential. The characterization of a protein surface as hydrophobic or hydrophilic has generally been performed on the basis of the identity of the underlying atom: either by residue type ('hydrophobic residues' giving a hydrophobic surface type, *etc.*) or element (carbon or sulfur giving a hydrophobic surface type and oxygen or nitrogen giving a hydrophilic surface type). Either of these characterizations neglects the basic characteristics of hydrophobicity: large charged residues such as arginine contain both a hydrophobic aliphatic portion as well as a charged hydrophilic guanidinium group. That guanidinium group, moreover, forms polar interactions in a highly direction-dependent fashion. Water molecules are preferentially distributed in the plane of the guanidinium group and are excluded from interaction with the π-electron system above and below this plane. Arguments such as this suggest that new tools are needed to characterize surface hydrophobicity.

A recent development of the program *GRID* (Goodford, 1985) has offered such a tool. The potential functions in *GRID* are based on summed pairwise interactions of a 'probe' atom with all surrounding atoms. Conventionally, the potential is evaluated on a three-dimensional lattice, but the program also allows the evaluation of the potential at arbitrary positions. The new hydrophobic probe in *GRID* can be characterized as forming favourable dispersion interactions with surrounding atoms, but being strongly disfavoured at sites where a water molecule would form good polar interactions. The sophisticated hydrogen-bonding function in *GRID* therefore provides a directional dependence to the hydrophobic potential. This potential provides intuitively correct results and has been found to be a metric of protein hydrophobicity that is highly predictive in the identification of sites of protein–protein interaction. We will introduce this functionality to *CCP4mg* in two stages, at first using *GRID* as an accessory program to calculate hydrophobic potential, but subsequently with potential function calculations performed by *CCP4mg* itself, again taking advantage of the atomic typing functionality.

6. Publication and presentation

6.1. Tools for generating presentation graphics

For a molecular-graphics user, the task of compiling the appropriate picture for a presentation can be very time-consuming. We have attempted to simplify that task by providing some easy-to-use tools.

The Display Table GUI window (Fig. 1) lists everything currently displayed showing each data object (*e.g.* model read from a PDB file or map generated from an experimental data file) and listing one or more display objects that represent the data object. For each type of display object there are appropriate interfaces to customize the appearance of the object. In particular, for model display objects, the selection of displayed atoms, the colouring scheme and the representation style (*e.g.* ribbons or spheres) can be controlled from simple menus.

Presentation graphics particularly need text annotation. OpenGL, the underlying graphics library for *CCP4mg* and many other molecular-graphics program, does not provide a particularly simple programming interface for the display of text. OpenGL displays text as simple bitmaps, but has no inbuilt access to system fonts, so one must convert a string of text into an appropriate bitmap by some other means. We have developed a library that can render an item of text into a bitmap using any system font and size. The library works on X11 and Microsoft Windows and the programmer need not
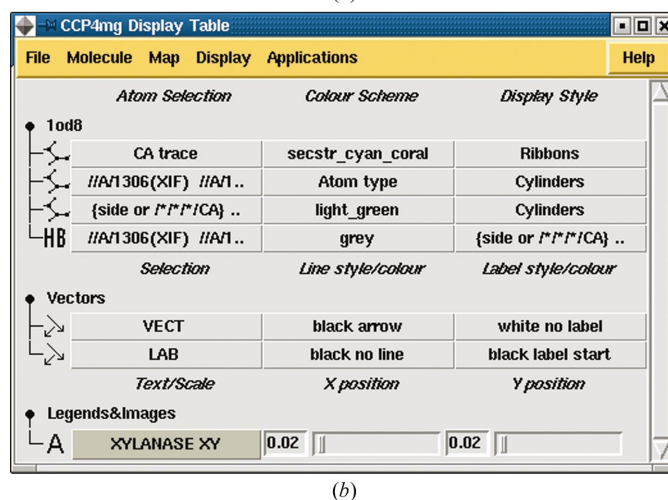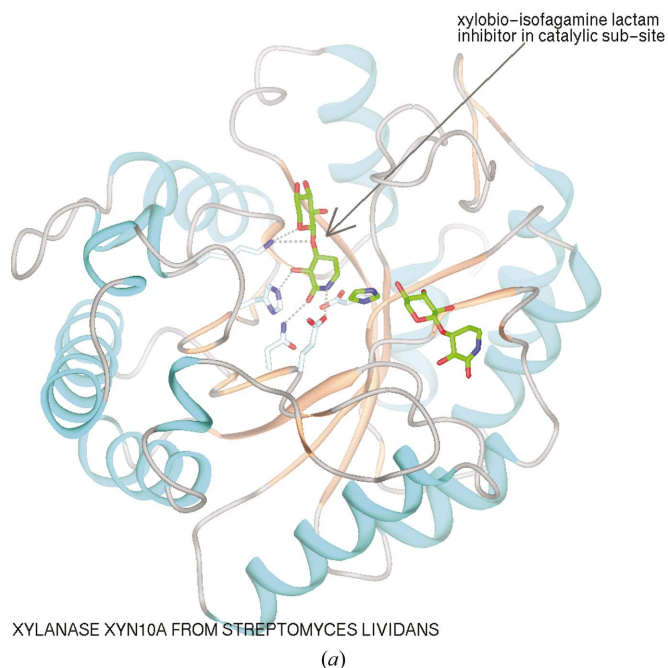

(a)

(b)

Figure 1
(*a*) A screenshot from *CCP4mg* showing a molecule of xylanase and bound inhibitors (Gloster *et al.*, 2003). Creating a picture such as this within *CCP4mg* is straightforward and the display table GUI which controls the display is shown in (*b*). The table lists the data objects (the model, 1od8, a vector object which provides the three-dimensional labelling on the model and a two-dimensional legend). Each data object has one or more display objects that is some selected portion of the data object displayed in a selected style. The buttons and sliders *etc.* associated with each display object allow the user to change the appearance of the display object.

know about the underlying graphics implementation. Text may be positioned in the three-dimensional coordinates of the model (*i.e.* annotating the model) or in the two-dimensional coordinates of the window (*i.e.* a legend).

CCP4mg can also import and display bitmap images such as an organization logo or a graph or a chemical diagram that might complement the molecular graphics. It is often helpful to add arbitrary features such as arrows or links between atoms to a presentation, so *CCP4mg* supports arbitrary vectors that can optionally have a text label. The vectors can be displayed in a variety of styles: dashed or solid lines, cylinders, arrows or cones. The position of the text label relative to the vector can be controlled and the text label can have any chosen font style. The vectors can be imported (using a simple mmCIF file format) or can be created within the program.

Since compiling a picture or movie can take some time to generate, it is useful that *CCP4mg* can save the composition of a picture (*i.e.* the Display Table) and the view at any time and restore them later.

6.2. Output options (PostScript, bitmap, ray tracing)

The final image on the screen may be output as a bitmap image in a variety of common image formats. The bitmap image can be created at an arbitrary resolution by rendering smaller subsections or tiles of the current image individually and then 'stitching' the tiles back together. For example, one might have the graphics window currently at 800×800 pixels. By opting to create an image from four tiles, the top-left, top-right, bottom-left and bottom-right quadrants of the image will each be displayed at 800×800 pixels and then combined to create a final 1600×1600 image. The image may be optionally smoothed by anti-aliasing.

Adobe PostScript output of line representations (bonds, electron-density isosurfaces) can also be produced. Since PostScript provides a vector description of lines and text, the produced images are resolution independent.

The scene description, current viewpoint, displayed objects *etc.* can also be written out to a file in a format for input into a ray-tracing program. At the moment this output file can only be in *POV-Ray* format, but we expect to add others. Ray-tracing may also be implemented in *CCP4mg* itself in the future, depending on demand.

6.3. Movies

Movies are a useful tool for both presentations and web-based dissemination of information. The basic mechanism to generating movies is to save one frame at a time as the screen display is slowly transformed. The frames are subsequently compiled into a movie file such as an animated GIF. To create a movie, the user can set up preferred camera positions using the conventional mouse control of position, orientation and zoom and save these positions. The program will then move through the scene, interpolating between preferred positions and saving snapshots at suitable intervals. It will be possible to update the model from a dynamics trajectory or morph or change the composition of the scene while moving through the scene.

7. Rebuilding and refinement tools

The first generations of crystallographic modelling packages were designed to provide tools for the user to define every detail of the final model. Atoms, residues or fragments could be moved with dial boxes and general torsion angles could be rotated. Regularization was applied to bring the model back into conformance with known preferred parameters for bond lengths and angles, but a few other options existed to draw on the expanding database of known structural preferences. A consequence of granting freedom to users was, in some cases, the taking of liberties (Jones & Kleywegt, 1995) and this observation lead to a philosophical development towards algorithms in which known stereochemical preferences were built into the atomic model from the beginning. This approach was implemented in the form of database integration, which provided the user with a 'Lego' toolkit of main-chain and side-chain fragments that were derived from reliably determined structures.

As computing speed increased, algorithms that permitted the automated fitting of main-chain and side-chain conformations to density maps made their appearance. Adoption of these new features in existing programs certainly facilitates the construction of a more accurate model by hand, but can only be of use where the existing community of users takes the time to discover new features. At the same time, since the authors of modelling packages are not the same people that write refinement software, a recurring struggle has been to convert between specific incarnations of PDB files, where there is tendency to lose in each refinement cycle some of the information painfully inserted in the previous one. The central philosophy of *CCP4mg* for rebuilding and refinement is twofold: to provide a small, obvious and highly automated set of reliable tools for most common processes (while keeping manual options available) and to give transparent interaction with the refinement software *REFMAC*5 (Murshudov *et al.*, 1997).

The most general rebuilding tool will be a real-space refinement and regularization option. This will allow local refinement of any portion of the model for which a *REFMAC* dictionary is available. Further tools are available for specific tasks as described in the following sections.

7.1. Proteins

The following protein-specific tools are planned. Further details are given in Emsley & Cowtan (2004).

7.1.1. Main chain. The principle main-chain rebuilding tool is for the building of short loops. Especially when coming from an automated building program or from manual building of an initial model, it is common to have to join the C-terminus of one fragment to the N-terminus of another. Often the density for loops is weak and fitting on density alone is not enough. Current computing power allows the systematic exploration of

the Ramachandran-allowed space for stretches of 5–6 residues and proposal of the best fit to density that satisfies both geometry and density criteria. For stretches of up to 9–10 residues Monte Carlo approaches can explore a significant region of the conformational space.

7.1.2. Side chain. Side chains may be automatically rebuilt using a search over common rotamers. For minor errors or unusual conformations, real-space refinement is adequate to fit the density.

7.2. Nucleic acids and other polymers

These will initially be handled through the built-in real-space refinement methods and through the interface to *REFMAC*5. In the long term, specific tools may be developed for these tasks.

7.3. Interaction with *REFMAC*5 and *CCP4i*

CCP4i, the interface to many programs of the *CCP*4 suite, maintains a database that records information such as input parameters and files and output files for each job performed. *CCP4mg* will record all steps that change the structure under refinement and save these to the *CCP4i* database.

CCP4mg v.2 will have a smooth interface to run *REFMAC*5 *CCP4i*. The user will optionally have access to the *CCP4i* *REFMAC*5 task interface if they wish to make changes from the default behaviour. On completion of the *REFMAC*5 job, the latest coordinate data and phase data will be loaded into *CCP4mg*.

7.4. Interaction with validation tools

There are numerous programs that are used to validate a structure prior to deposition such as *WHATCHECK* (Vriend, 1990) and *PROCHECK* (Laskowski *et al.*, 1993) and these will be evaluated from the point of view of providing useful information to guide model rebuilding. A refinement program such as *REFMAC*5 also knows about the quality of the model. In future, *REFMAC*5 will output an analysis including residue-by-residue map correlation, geometric violations, large discrepancies in *B* values and departures from the prior probability distribution. This analysis will be presented to the user in a simple graphical form with an option to 'Go to the Next Badly Fitted Bit of Model'.

7.5. Initial model-building tools

Initial model building will focus on the building of protein structures with waters and ligands. Other polymers will be considered at a later date. The model-building tools will combine stages selected from several existing programs with the aim of providing the most efficient set of modern tools. Many tools are under development and are described in Emsley & Cowtan (2004). The task is divided into the following stages.

7.5.1. Skeletonization. This is carried out by a modified version of the Greer algorithm (Greer, 1985), which has been further developed by Cowtan, Emsley and Foadi (unpublished results) to increase the efficiency of the method and remove some of the arbitrary factors. Like the electron-density map, the skeleton is calculated in a continuous crystal.

7.5.2. Baton-building. To construct a C^α trace from the skeleton, a baton is used (Jones & Kjeldgaard, 1994) to represent the vector connecting a pair of adjacent C^α atoms in the sequence. The user is offered a sorted list of possible baton positions, which are chosen on the basis of likely C^α positions in the skeleton and pseudo-Ramachandran angles from the geometry of the preceding C^α atoms (Oldfield, 2001). The user can often simply accept the offered position or cycle through the alternatives. When building helices, it is sometimes necessary to reduce the length of the baton to counteract the effect of building too tight a helix. The baton may also be rotated to any orientation if none of the proposed orientations are suitable; however, this step is seldom necessary.

7.5.3. From C^α to main chain. Main-chain atoms are generated using the method of Esnouf (1997). The C^α model is broken into a set of overlapping six-residue fragments. These fragments are tested against a database of fragments generated from high-resolution structures (1.3 Å or better) from the PDB. Fragments that are similar in shape and have small differences in the positions of the C^α atoms after a least-squares fit are merged with a weight corresponding to the goodness of fit. This procedure works well for clear secondary structure, but for less well fitting fragments results in particular in main-chain O atoms with too short a bond to main-chain C atoms. This is easily fixed by refinement or regularization.

7.5.4. Sequence assignment. Sequence assignment involves two stages: an image-recognition step in which the likely side-chain types for each amino acid are determined and then a comparison stage against the sequence to see if each chain fragment can be identified from the arrangement of likely side chains. These will be accomplished using the *GUIside* software (A. Perrakis), which in a first step analyses patterns of neighbourhood relationships amongst pseudo-atoms fitted into the electron density and then calculates *Z* scores for each side-chain type at each position in the fragment. The sequence is matched against the fragment at each possible residue offset and a combined score for that combination of residue types is determined. If an unambiguous match is identified, then the corresponding residue types are assigned to the residues.

7.5.5. Side-chain building. The building of side chains is identical to the corresponding rebuilding problem, with a rotamer search followed by real-space refinement.

7.5.6. Fitting waters. Waters may be fitted by looping over the peaks in the difference map or over peaks in the best electron-density map masked to exclude fitted atoms. Each peak may be displayed in turn and the user may opt for a water to be placed at the peak maximum.

7.5.7. Ligand fitting. Ligand fitting involves identification of larger unaccounted-for density features. This is achieved by selecting a cutoff level for density to be fitted. All density features above this level are marked and their shapes determined. The orientation of the density is matched to the orientation of the ligand model by comparing the principal axes of the densities. Finally, the density from the model and the map are compared in order to pick the best orientation. In

research papers

the case of flexible ligands, multiple searches are carried out with each of a range of ligand conformations sampling the available torsions within the ligand.

References

Carson, M. (1997). *Methods Enzymol.* **277**, 493–505.

Christopher, J. A., Swanson, R. & Baldwin, T. O. (1996). *Comput. Chem.* **20**, 339–345.

Cowtan, K. (2003). *IUCr Comput. Commission Newsl.* **2**, 4–9.

DeLano, W. L. (2002). *The PyMOL Molecular Graphics System.* San Carlos, CA, USA: DeLano Scientific.

Emsley, P. & Cowtan, K. (2004). *Acta Cryst.* D**60**, 2126–2132.

Esnouf, R. M. (1997). *Acta Cryst.* D**53**, 665–672.

Esnouf, R. M. (1999). *Acta Cryst.* D**55**, 938–940.

Gloster, T., Williams, S. J., Tarling, C. A., Roberts, S., Dupont, C., Jodoin, P., Shareck, F., Withers, S. G. & Davies, G. J. (2003). *Chem. Commun.* pp. 945–948.

Goodford, P. J. (1985). *J. Med. Chem.* **28**, 849–578.

Greer, J. (1985). *Methods Enzymol.* **115**, 206–224.

Guex, N. & Peitsch, M. C. (1997). *Electrophoresis*, **18**, 2714–2723.

Hartshorn, M. J. (2002). *J. Comput. Aided Mol. Des.* **16**, 871–881.

Jones, T. A. (1978). *J. Appl. Cryst.* **11**, 268–272.

Jones, T. A. & Kjeldgaard, M. (1994). *Proceedings of the CCP4 Study Weekend. From First Map To Final Model*, edited by S. Bailey, D. Hubbard & R. Waller, pp. 1–13. Warrington: Daresbury Laboratory.

Jones, T. A. & Kleywegt, G. J. (1995). *Structure*, **3**, 535–540.

Jones, T. A., Zou, Y. J., Cowan, S. W. & Kjeldgaard, M. (1991). *Acta Cryst.* A**47**, 110–119.

Kraulis, P. J. (1991). *J. Appl. Cryst.* **24**, 946–950.

Krissinel, E. B. & Henrick, K. (2004). *Acta Cryst.* D**60**, 2256–2268.

Krissinel, E. B., Winn, M. D., Ballard, C. C., Ashton, A. W., Patel, P., Potterton, E. A., McNicholas, S. J., Cowtan, K. D. & Emsley, P. (2004). *Acta Cryst.* D**60**, 2250–2255.

Laskowski, R. A., MacArthur, M. W., Moss, D. S. & Thornton, J. M. (1993). *J. Appl. Cryst.* **26**, 283–291.

McRee, D. E. (1999). *J. Struct. Biol.* **125**, 156–165.

Martz, E. (2002). *Trends Biochem. Sci.* **27**, 107–109.

Merritt, E. A. & Bacon, D. J. (1997). *Methods Enzymol.* **227**, 505–524.

Mitchell, E. M., Artymiuk, P. J., Rice, D. W. & Willett, P. (1989). *J. Mol. Biol.* **212**, 151–166.

Murshudov, G. N., Vagin, A. A. & Dodson, E. J. (1997). *Acta Cryst.* D**53**, 240–255.

Nicholls, A., Sharp, K. A. & Honig, B. (1991). *Proteins*, **11**, 281–296.

Oldfield, T. J. (2001). *Acta Cryst.* D**57**, 82–94.

Potterton, E., Briggs, P., Turkenburg, M. & Dodson, E. (2003). *Acta Cryst.* D**59**, 1131–1137.

Turk, D. (2001). *Methods in Macromolecular Crystallography*, edited by D. Turk & L. Johnson, pp. 148–155. Amsterdam: IOS Press.

Vriend, G. (1990). *J. Mol. Graph.* **8**, 52–56.