International Tables for Crystallography (2006). Vol. F, Section 17.1.2, pp. 353–354.

17. MODEL BUILDING AND COMPUTER GRAPHICS

17.1. Around *O*

BY G. J. KLEYWEGT, J.-Y. ZOU, M. KJELDGAARD AND T. A. JONES

17.1.1. Introduction

The first protein structures to be solved were built as wire models. It was only at the end of the 1970s and the beginning of the 1980s that the necessary hardware and software became available to allow crystallographers to construct their models using computers. The first commercially available computer graphics systems were very expensive, and by today's standards rather primitive in their line-drawing capabilities. They were usually controlled by mini-computers loaded with 64k words of memory and removable disks capable of holding 1 Mbyte of data. The limited amount of addressable memory was a severe limitation in software production. Furthermore, each computer graphics system had its own graphics programming library that was totally incompatible with those of other systems. Despite these limitations, the benefits of using computer-graphics-based systems became apparent and they were fairly rapidly adopted by the laboratories that could afford them. The main benefits were not in the construction of the initial model, but rather as a tool in crystallographic refinement (Jones, 1978). Making small manual changes to a model being refined was difficult and time-consuming work, but rather easy to accomplish even with low-powered graphics systems.

The most widely used program of the early days, *Frodo*, was available on most of the contemporary computer graphics systems. A major step forward occurred with the development of laboratory-scale 32-bit computers with virtual memory operating systems. In particular, the first Digital Equipment VAX models rapidly became the machines of choice in crystallographic laboratories. This allowed the contemplation of real-time improvements in models under construction (Jones, 1982; Jones & Liljas, 1984). Soon afterwards, colour became available in commercial graphics systems. This was much more than a cosmetic enhancement, since colour could be used to convey information vital to the crystallographer, such as main-chain/side-chain status codes for skeletonized electron density (Jones & Thirup, 1986). Unfortunately, there was still no common graphics programming standard, and moving to a new graphics system was a major effort (Pflugrath *et al.*, 1984).

The next major advance in hardware occurred with the development of the workstation, combining the computer and graphics in one package. Although pioneered by Sun, the major player in the crystallographic community was a small Californian company, Silicon Graphics, which rapidly became large. Running the Unix operating system, workstations flourished, but still lacked a graphics environment that was portable between different hardware platforms. This changed when OpenGL was adopted as an industry standard. At the same time, prices stabilized and began to drop in terms of price/performance. Only in the late 1990s have price/performance indicators plummeted with the arrival of PC/graphics-board combinations capable of meeting the expectations of the current generation of crystallographers. The crystallographic workstation on every desk has finally arrived.

17.1.2. *O*

O was designed by Alwyn Jones to overcome some of the drawbacks associated with using *Frodo*. These problems had arisen because of the history of the program. In particular, *O* was designed to use a general-purpose memory allocation system to store any kind of model-related data. This would allow the display of any number of molecules and the use of databases for modelling. Although the latter had been introduced in a *Frodo* variant (Jones & Thirup, 1986), *O* was designed to take the concept of database use further, some would say to the extreme. Furthermore, *O* was designed to make it easier for different developers to work on the code without interfering with each other. In the event, only Morten Kjeldgaard and Jin-Yu Zou made any developments with the program.

Much of the data used by *O* are kept in a memory allocation system, the *O* database. This database is used to save parameters used by the program (including such things as keywords), macromolecular coordinates and information derived from them (such as graphical objects). As such, the program has no built-in limitations concerning what can be saved and used. A set of coordinates can be downloaded from the Protein Data Bank (PDB) (Bernstein *et al.*, 1977) and stored as a series of vectors that describe the sequence, the residue names, the coordinates, the atom names, the unit cell *etc*. Some of these vectors contain residue-related data (*e.g.* the sequence), others contain atomic data (*e.g.* the atomic temperature factors), while yet others concern the molecule as a whole. The program therefore uses a strict naming convention in handling these data. Each molecule has a name, and the program then forces its own nomenclature for the standard atomic, residue and molecular properties. The user remains free to create new data outside *O*, bring them into the program by adopting the naming convention, and then make use of them to generate or manipulate graphical images. For example, a series of amino-acid sequences can be aligned with a computer program outside *O* and information on the degree of sequence conservation can be generated as a series of *O* data blocks. These can then be read into *O* and used to colour a Cα trace of a model, for example.

The program also has a strong macro capability that can be used to configure quite complex interactive tasks. It can also be used by a programmer outside *O* to generate data and a series of instructions for later interactive use.

Similarly, data generated within *O* can be exported to *O*-aware programs, significantly reducing the complexity of some crystallographic calculations. For example, real-space averaging of electron-density maps requires, as a minimum, both a series of operators describing the noncrystallographic symmetry (NCS) and a mask. These can be generated from scratch in *O* and improved and used by *O*-aware programs without the crystallographer needing to be concerned about the myriad details of axis definitions, rotations and translations.

Plotting is carried out within the *O* system *via* an intermediate metafile. When a user creates objects for display within *O*, calls are made to a set of low-level routines that create the OpenGL instructions on the workstation. Some objects are described in their entirety within the *O* database, but others are not. Molecular objects fall into the former category, whereas electron-density maps fall into the latter. There are two sets of plot commands, therefore, that are appropriate for each class. To plot an object made from a molecule, the user merely issues a *plot_object* command, and the appropriate metafile is written out, complete with viewing data. To plot other things, the user activates the *plot_on* command and then starts creating objects. Every time a low-level graphics routine is

called, something gets written into the metafile. This is terminated with a *plot_off* command. The metafile contains much extraneous data, for example, instructions to the *O* pulldown menu system. However, it is built up from objects that are arranged in a hierarchy, where the highest-level object is called *disp_all*. Some objects, therefore, call instances of others, while other objects contain graphics instructions that define line start and end points, for example.

This metafile can be processed, and so far three different programs are available. *OPLOT* (written by Morten Kjeldgaard) generates PostScript output, carrying out a full traversal of the object hierarchy. The other two programs do not carry out such a traversal, but merely process the objects specified by the user. One (written by Mark Harris and Alwyn Jones) generates output suitable for input to the ray-tracing program *PovRay* (see http://www.povray.org). The third program (written by Martin Berg) generates VRML output suitable for web-based viewing.

O is in continuous development, and interested readers are encouraged to visit the various internet sites that we maintain. There they will find detailed descriptions of the *O* command set, as well as various introductory exercises for learning how to use the program. The following publications describe various aspects of *O*-related features and methods:

(1) Jones *et al.* (1991) provide an introduction to the *O* database, a description of a residue-based electron-density goodness-of-fit indicator, the use of databases to construct a poly-alanine from a Cα trace, various real-space refinement algorithms *etc.* They also describe two useful indictors for detecting peptide-plane and side-chain errors that make use of comparisons with databases.

(2) Zou & Mowbray (1994) describe an evaluation of the use of databases in refinement.

(3) Zou & Jones (1996) describe their attempts towards finally automating the interpretation of electron-density maps. They also describe both qualitative and quantitative matching of the protein sequence to the map.

(4) Jones & Kjeldgaard (1997) review the different kinds of errors that can be introduced into a model and why these errors are made. They also describe some of the features of *O* and the steps needed in tracing and building a model (including the vital step of locating the sequence in the electron density).

(5) Mowbray *et al.* (1999) describe experiments aimed at evaluating the reproducibility of model building and discuss some of the more useful indicators of model error.

17.1.3. *RAVE*

RAVE is a suite of programs for electron-density map improvement and analysis, with a strong focus on averaging techniques (Kleywegt & Read, 1997). It is the successor of an older package (‘*A*’) (Jones, 1992), and at present it contains tools for single and multiple crystal form, single- and multiple-domain NCS averaging of electron-density maps, and the detection of structural units in such maps. The package works in conjunction with the *CCP*4 suite of programs (Collaborative Computational Project, Number 4, 1994).

RAVE contains the following programs for density averaging involving one crystal form:

(1) *AVE* (Jones, 1992). This program carries out the averaging step and the expansion step (in which the averaged density is projected back into the entire unit cell or asymmetric unit).

(2) *COMA* (Kleywegt & Jones, 1999). This program uses the algorithm of Read (Vellieux *et al.*, 1995) to calculate local density correlation maps that can be used to delineate masks (molecular envelopes). It can also be used to validate structural differences between NCS-related molecules (Kleywegt, 1999*b*).

(3) *IMP* (Jones, 1992). This program can be used to optimize NCS operators relating two copies of a molecule (or domain) inside the same cell. The program adjusts the initial operator (*e.g.* obtained from heavy-atom positions) so as to maximize the correlation coefficient between the density inside the envelope and its NCS-related counterpart. The procedure can be controlled by the user or run in automatic mode, which usually gives satisfactory results.

(4) *NCS6D*. This program can be used to find NCS operators in cases where it is difficult to obtain them by other means. The program uses a set of *BONES* atoms (or a PDB file) and, for a large number of combinations of rotations and translations, calculates the correlation coefficient between the density around the atoms and that obtained after application of the rotation and translation. This approach was used, for instance, to find the operators in the case of maltoporin (Schirmer *et al.*, 1995).

(5) *COMDEM*. If a molecule contains multiple domains that have different NCS relationships, the individual domain densities can be averaged with *AVE* and subsequently combined with this program. *AVE* can then be used to expand the density back into the unit cell or asymmetric unit.

(6) *SPANCSI*. This program is useful when NCS-related molecules are known or suspected to have very different average temperature factors. One option is to analyse the similarities between NCS-related copies of the molecular density (variance, correlation coefficient, *R* factor). In addition, the program can carry out electron-density averaging and expansion, in which each copy of the density is scaled by its variance.

RAVE also contains tools for averaging between different crystal forms, namely:

(1) *MASKIT* (Kleywegt & Jones, 1999). This program calculates a local density correlation map from the density of two different crystals or crystal forms, using Read’s algorithm (Vellieux *et al.*, 1995). This program can also be used to validate structural differences between related molecules, for which experimental electron density is available (Kleywegt, 1999*b*).

(2) *MAVE*. This program does the (skew) density averaging and expansion steps, but now separately because the density of the various crystal forms has to be averaged as well. This program also contains an option to improve operators that relate the position and orientation of the molecular envelope (mask) in one crystal form with those in other crystal forms.

(3) *COMDEM*. This program combines the individual (possibly averaged) densities from various crystal forms. The densities are scaled according to the number of molecules whose (averaged) density they represent as well as according to their variance.

(4) *CRAVE*. Since the book-keeping for multiple-crystal-form averaging can become rather complicated, this program can be used to generate one large C-shell script that will execute a user-defined number of cycles of multiple-crystal-form averaging.

More recently, *RAVE* has been expanded to include tools that can be of use in map interpretation:

(1) *ESSENS* (Kleywegt & Jones, 1997*a*). This program takes a (rigid) structural template (*e.g.* a penta-alanine helix or strand, or a ligand) and calculates how well it fits the density by doing an exhaustive rotational search for every grid point in the map. The resulting score map will reveal places in the map where the centre of gravity of the template fits the density well. The method is very effective for detecting secondary-structure elements (prior to human map interpretation), as discussed by Kleywegt & Jones (1997*a*). The *ESSENS* algorithm has also been implemented within *O* (Jones & Kleywegt, 2001).

(2) *SOLEX*. This program can be used to extract the best-fitting positions and orientations of a structural template as found in an *ESSENS* calculation. If the search used a template in helix or strand conformation, the program can also be used to combine short stretches of helix and strand into longer units. The results (helices