Acta Crystallographica Section B
Structural Science
ISSN 0108-7681

New software for searching the Cambridge Structural Database and visualizing crystal structures

Ian J. Bruno, Jason C. Cole, Paul R. Edgington, Magnus Kessler, Clare F. Macrae, Patrick McCabe, Jonathan Pearson and Robin Taylor*

Cambridge Crystallographic Data Centre, 12 Union Road, Cambridge CB2 1EZ, UK

Correspondence e-mail: taylor@ccdc.cam.ac.uk

Received 1 February 2002
Accepted 19 February 2002

Two new programs have been developed for searching the Cambridge Structural Database (CSD) and visualizing database entries: *ConQuest* and *Mercury*. The former is a new search interface to the CSD, the latter is a high-performance crystal-structure visualizer with extensive facilities for exploring networks of intermolecular contacts. Particular emphasis has been placed on making the programs as intuitive as possible. Both *ConQuest* and *Mercury* run under Windows and various types of Unix, including Linux.

1. Introduction

The Cambridge Structural Database (CSD; Allen, 2002) contains the results of over a quarter of a million organic and metallo-organic crystal structure analyses. It has long been a valuable research tool in crystallography, structural chemistry and drug design, as borne out by the large number of published studies in which it has played a major part (Allen & Motherwell, 2002; Orpen, 2002; Taylor, 2002) and the high citation rating of several of these studies (Redman *et al.*, 2001). However, it became clear in the mid-1990s that the software for searching and visualizing the database was in need of major revision. While the functionality of the programs was good, they had fallen behind contemporary standards of usability and were not operable on the increasingly dominant Windows platform. These two considerations prompted us to embark on a concerted effort to provide improved programs to CSD users.

From the outset, emphasis was placed on developing user-friendly graphical user interfaces (GUIs) rather than extending program functionality. Intuitive interfaces are particularly important for CSD programs because of the nature of the user-base. While the CSD has its coterie of aficionados, many others are occasional users, accessing the database only every few weeks or so. Effectively, these can be regarded as 'new' users every time they access the programs, because they do not build up sufficient expertise to remember how things work from one session to another. Designing interfaces that would satisfy the needs of both expert and casual users was therefore a key aim of our work.

This paper describes two new programs, *ConQuest*, for searching the CSD, and *Mercury*, for visualizing crystal structures. They are successors to the programs *QUEST* (Allen *et al.*, 1991) and *PLUTO* (Motherwell & Shields, 2000), respectively.

2. *ConQuest*

2.1. Program language and architecture

Fortran code for searching the CSD was developed from as early as the 1970s, through to the 1990s (Allen *et al.*, 1991). The code performs a large variety of tasks, including two-dimensional and three-dimensional substructure searching, text searching, screening, geometry calculations, crystallographic calculations, such as cell reduction, and so on. However, it is poorly structured by modern standards and has become difficult to extend and maintain. For these reasons, we intend to replace it with an object-oriented C++ class library (see below). However, it was impossible for us to do this on the timescales required for the implementation of *ConQuest*. We therefore decided on a compromise, *viz.* to use the existing Fortran code for searching the CSD but provide an entirely new front end. This would achieve our key aims – greatly increased user-friendliness and a Windows implementation – while making use of existing code, which, although not ideal, provided the necessary, well tested search functionality.

Even with this simplification, the project was challenging: it is difficult to design from scratch an intuitive interface to complex functionality. We recognized that a good design would only be developed iteratively, by producing a series of prototype dialogues and refining them *via* usability testing. For this reason, we required a programming language that would allow rapid prototyping. We also wished to use object-oriented principles. These considerations led us to choose the Python scripting language (van Rossum, 1991) and the Tk GUI toolkit (Ousterhout, 1994), which have a well defined interface, Tkinter (Lundh, 1999). Not only did this combination allow extremely quick object-oriented prototyping, it facilitated the development of highly platform-independent code. We made use of Python Megawidgets (Telstra Corporation Ltd, 1997) to achieve incorporation of sophisticated user-interface elements without the need for low-level programming. Of particular importance for us was the PyOpenGL module (Hugunin *et al.*, 1997), which provides bindings to the OpenGL libraries and supports the Togl Tk widget (Paul & Bederson, 1996) for OpenGL rendering. This is used in *ConQuest* to provide a high-quality display of three-dimensional structures. Other Python facilities that were useful included the ReportLab Library (ReportLab Inc., 1998), a set of modules that enabled us to write pdf files which, when coupled with the Adobe Acrobat Viewer (Adobe Systems Inc., 1987), provide an easy route for generating printed output.

Effecting communications between the new Python *ConQuest* interface and the old Fortran search code (which we re-named *Thomas the Search Engine*, or *Thomas* for short; Awdry, 1946) was greatly facilitated by the fact that there is a simple ASCII command-line CSD query language (Allen *et al.*, 1991). This is taken as input by *Thomas* and fully describes the user's query. Therefore, it was necessary to (i) generate the appropriate series of ASCII commands from the Python/Tk *ConQuest* GUI; (ii) send this to *Thomas* as input; (iii) pick up the resulting hit information from *Thomas* and send it back to the GUI *via* a file-based asynchronous FIFO (first-in, first-out) buffer. *Thomas* performs a sequential search of the CSD, producing hit structures in a strict order. In *ConQuest*, however, the user has the ability to browse hit structures interactively, in a forwards or backwards direction; this can be done while the search is in progress and after it has finished. Search results can also be saved to file. The required persistent data storage and manipulation is achieved in *ConQuest* by the use of the Metakit database library (Wippler, 1996).

2.2. User-interface design principles

GUI design was guided by the following principles.

(i) Although we were influenced by conventional wisdom, as catalogued, for instance, in the interface 'hall of shame' (Isys Information Architects, 1996), we determined that the ultimate test of an interface is usability testing. Virtually every interface component was therefore tested by observing users who had never seen the program navigate their way without help through a series of tasks. Most of our initial dialogue designs were refined, sometimes very significantly, in the light of test outcomes.

(ii) Several possible routes to functionality were provided when usability tests indicated this was necessary. For example, tests suggested that users of the substructure drawing tool are approximately equally divided into two camps: those who look for an action button (*e.g.* to set the element type of an atom) and then seek to pick the objects to which that action is to be applied; and those who try to select the objects first and then look for the action button. Consequently, both routes were usually implemented.

(iii) Pull-down menus accessible *via* the right-hand mouse button were provided in those parts of the interface where users might be expected to do highly interactive work with the mouse (the two key areas are the substructure drawing tool and the three-dimensional visualizer). This is very efficient, since it reduces mouse movement (*e.g.* users have no need to go to a top-level menu). On the other hand, tests showed that many users do not think of clicking the right-hand mouse button, so it is desirable to provide alternative routes to the functionality. This was achieved in the substructure drawing tool; in the three-dimensional visualizer, we contented ourselves with displaying a permanent message alerting users to the availability of a menu *via* the right-hand mouse button.

(iv) We followed many common interface conventions, not because this is important *per se* but because what is conventional is also familiar to users, and therefore intuitive. Obvious examples which will be familiar to all PC users are a top-level 'File' button offering functions for saving and reading files, exiting the program *etc.*, and an 'Edit' button in the substructure drawing tool with options for copying, cutting, pasting, undoing *etc*. We also looked at conventions established in well known packages such as *ChemDraw* (CambridgeSoft Corporation, 2001) and *ISIS/Draw* (MDL Information Systems, Inc., 2001) and followed these where appropriate.

(v) User errors are trapped, but only when it can be established beyond doubt that they are indeed errors. For

Table 1
Summary of *ConQuest* functionality.

Main panes/windows are listed in bold.

Option	Functionality
Build Queries	Query building; input/output of queries; query management
Draw	(i) Construction of two-dimensional (chemical connectivity) substructure queries (including variable atom/bond types, ring constraints, variable attachment points *etc.*) (ii) Construction of three-dimensional substructure queries (constraints on distances, angles, torsions, inter-plane angles, pharmacophore searches *etc.*) (iii) Construction of nonbonded contact queries (intermolecular and intramolecular hydrogen bonds and other interactions)
Peptide	Construction of peptide-sequence queries
Author/Journal	Construction of bibliographic queries
Compound Name	Construction of chemical name queries
Elements	Construction of queries for chemical elements or groups of elements
Formula	Construction of formula queries, *e.g.* $C_{3-6}H_{12-24}O_{\leq 4}$
Space Group	Construction of queries for exact space groups; aspect symbols (*e.g.* *C**/*c*), crystal systems
Unit Cell	Construction of queries for reduced cell parameters or authors' cell parameters
Z/Density	Construction of queries for *Z*, *Z*′, density *etc.*
Experimental	Construction of queries for experimental conditions and results, *e.g.* *R* factor, X-ray/neutron diffraction, disorder, temperature *etc.*
All Text	Construction of text and keyword queries, *e.g.* polymorph, drug
Refcode	Construction of queries to find specific entries by their CSD identifiers
Combine Queries	Boolean operations on queries
Search Setup	(i) Choice of database or database subset to be searched (ii) Selection of common filters (*e.g.* *R* factor, exclude disordered structures) (iii) Selection of advanced options (standardize *X*—H covalent bond lengths before testing for hydrogen bonds)
View Results	(i) Browsing of chemical diagrams, bibliographic, chemical, crystallographic and experimental information for search hits (ii) Three-dimensional visualization (iii) Suppression of unwanted hits (iv) Output of search results in various formats, *e.g.* CIF, MOL2 (v) Transfer of search results to other programs, *e.g.* *Mercury*, *Vista*, *Excel* (vi) Viewing of previous searches; viewing of entire CSD

example, typing in a negative cell parameter will cause the relevant entry box to be highlighted and the 'Search' button of the dialogue box to be disabled so that the user cannot proceed until the mistake is corrected. On the other hand, we declined to check for apparent 'scientific' errors, *e.g.* a substructure with 5-valent carbon, since users know more science than we could program into *ConQuest* (in fact, chemical connectivities involving formally 5-valent C atoms are found in the CSD, *e.g.* in some boron cages).

(vi) We tried to avoid pop-up messages of the type 'Do you really want to do that?', which are notoriously irritating to users. Where we felt that such a precaution was necessary, we provided it in a dialogue that also offered useful options, hoping thereby to reduce annoyance. An obvious example is that before *ConQuest* is closed, the user is asked which search results they want to save. A less obvious example is that on requesting a search to be started, users are given a dialogue which allows them to set common search filters, subset selection *etc.* In both cases, a hidden purpose of the dialogues is to allow users to change their minds, *i.e.* not close the program, not start a search.

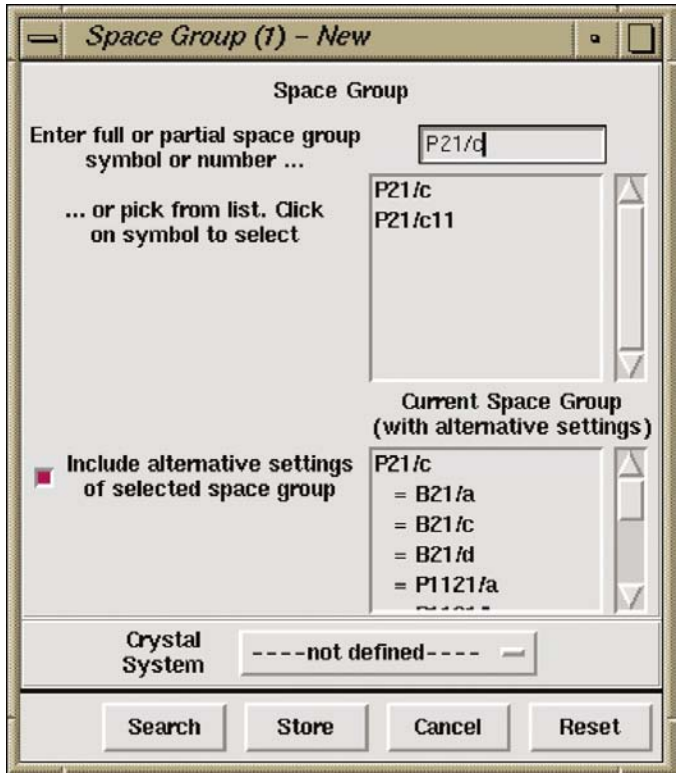

Figure 1
ConQuest dialogue box for building space-group queries.

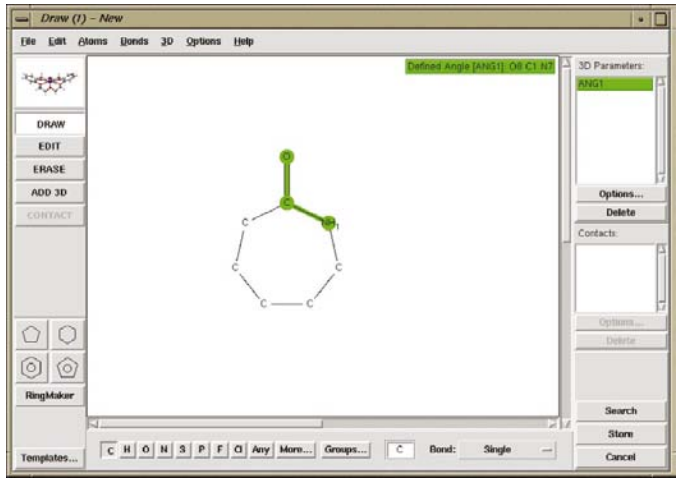

Figure 2
Example substructure query in the *ConQuest* drawing window.

(vii) We avoided the use of icons, preferring instead to label buttons with words. This was based on the belief that, while pictures nominally break down language barriers, icons are often international only to the extent that they are equally incomprehensible to all nationalities. We had the advantage, of course, that virtually all scientists have at least a basic understanding of English (although internationalization of *ConQuest* by offering a choice of languages in the interface is an important long-term goal).

(viii) Although *ConQuest* is fully documented, we regarded it as suggestive of a GUI design failure if users wished to have recourse to 'Help' facilities. In passing, we noted that balloon help and advisory messages in the program interface are frequently ignored by users, so, although worthwhile, are of limited value.

2.3. Program functionality

ConQuest program functionality (Table 1) closely mirrors that of its predecessor *QUEST* (Allen *et al.*, 1991), but is presented in a very different way. The layout of functionality in Table 1 reflects the organization of the *ConQuest* interface, which has three main panes, 'Build Queries', 'Combine Queries' and 'View Results', together with a 'Search Setup' dialogue box that appears just prior to commencing a search.

2.3.1. 'Build Queries'. Build Queries offers a choice of dialogues for constructing queries. An example is illustrated in Fig. 1. Undoubtedly the most important and complex is the substructure drawing tool, which enables construction of two-dimensional and three-dimensional substructure queries and nonbonded-contact queries (*e.g.* Fig. 2). Substructure drawing is achieved by dragging the cursor to draw bonds, and using menu options to set atom and bond properties, cyclicity constraints *etc.* Templates of common rings, chemical groups and other substructures are provided as shortcuts and a variety of edit options permit the size, shape and position of the drawn substructure to be altered. Dialogue boxes allow objects, such as vectors and planes, and parameters, such as torsion angles and distances, to be defined and, in the case of parameters, to be constrained to user-specified numerical limits.

Other dialogue boxes group together similar data items, as indicated in Table 1. As queries are constructed, they are listed in the Build Queries pane and can be edited, deleted, saved and turned on or off (Fig. 3). Hitting the Search button in the Build Queries pane is interpreted by the program as a request to find CSD entries which satisfy all of the queries that are turned on, *i.e.* a logical *and* is applied.

2.3.2. 'Combine Queries'. Combine Queries allows more sophisticated logical operations on queries. Since Boolean logic can be difficult, it is presented in *ConQuest* in 'natural language' form. Thus, three boxes in the interface are labelled 'must have' (*i.e.* Boolean *and*), 'must not have' (*i.e.* Boolean *not*) and 'must have at least one of' (*i.e.* Boolean *or*). Query icons are dragged into the appropriate box. For example, the arrangement in Fig. 4 corresponds to the logical combination of queries: (query 4) *and* (*not* query 3) *and* (query 1 *or* query 2).

Presentation of logic in this way limits the complexity of Boolean expressions that can be set up; for example, it would not be possible to extend the above expression to: (query 4) *and* (*not* query 3) *and* (query 1 *or* query 2) *and* (query 5 *or* query 6).

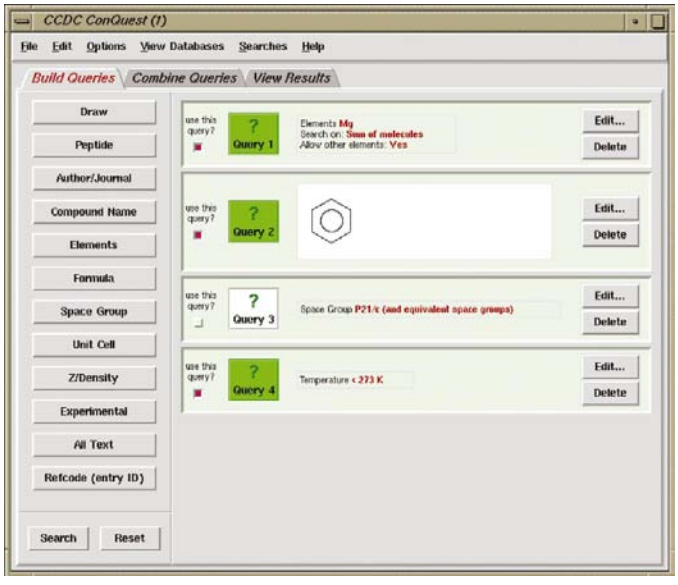

Figure 3
ConQuest 'Build Queries' pane showing four queries, three turned on and one turned off. The result of performing a search will be to find structures that contain magnesium and a benzene ring and were determined at a temperature below 273 K.

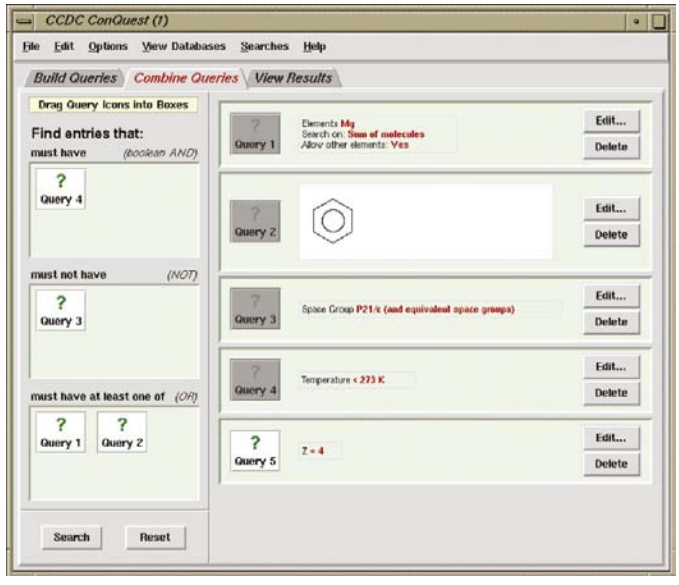

Figure 4
ConQuest 'Combine Queries' pane. The result of performing a search will be to find structures that satisfy query 4, do not satisfy query 3, and satisfy queries 1 or 2, or both. Query 5 has not been dragged into a box, and so is ignored.

However, this limitation is worthwhile in order to achieve the simplicity of presentation afforded by the Combine Queries layout. More complex Boolean combinations than are possible with Combine Queries can be achieved by running two or more searches, confining the second search to the hit list obtained from the first, and so on. While slightly laborious, this 'Boolean combination in stages' has the advantage of being much easier for users to understand.

2.3.3. 'Search Setup'. When a Search button is hit, the Search Setup dialogue box is shown. This allows selection of the database or database subset to be searched (users may opt to search the CSD and/or in-house databases in CSD format, or may wish to confine the search to a previously created hit list). It also offers common filters, such as *R*-factor limits.

2.3.4. 'View Results'. Once a search is initiated, results are transferred from the search engine to the View Results pane as the search proceeds. Thus, early hits can be inspected while the search is still running (important for some three-dimensional searches which can take a long time). Hits are selected for viewing from a scrolling list of 'Refcodes' (CSD entry identifiers). Hits that are stereoisomers of one another are hyperlinked. Information for each hit is displayed in a series of panes, *e.g.* 'Diagram' (chemical structure), 'Chemical' (compound name, formula *etc*.). This is something of a design weakness, because it is not easily possible to see different types of information at once, *e.g.* the three-dimensional structure and the compound name. However, there is a new option (scheduled for release in April 2002) to display the chemical diagram in a separate window of its own. Hit information is highlighted. Thus, those atoms and bonds in a chemical diagram that match a search substructure are coloured differently from the remainder of the diagram; for a text search, matching items of text are displayed with a distinctive background colour. An example View Results display is shown in Fig. 5.

Three-dimensional structures can be viewed in a visualizer which offers basic functionality, such as a choice of display styles, labelling options, measurement of parameters, display of unit-cell contents, display of geometrical objects such as centroids and least-squares mean planes, display of hit substructures and output of displays in jpeg format. However, we did not wish to complicate the *ConQuest* interface by including in it complex visualization options that would only be required by a minority of users. For advanced three-dimensional visualization, therefore, lists of structures found by *ConQuest* searches can be transferred easily to the *Mercury* program (see below). Top-level menu options in View Results allow search results to be saved in various formats, or data to be transferred to *Vista* (CCDC, 1994) or Microsoft *Excel* for statistical analysis and spreadsheeting. Individual entries in the hit list can be suppressed if they are not wanted; this means that they will continue to be visible in View Results but will not be exported if files are saved or data transferred to other programs.

During a *ConQuest* session, users may move freely between the three main panes, start any number of searches, even if previous searches are still running, or review the results of any of these searches. Search results may be saved in the native binary format of the program (a `.cqs` file). When read back in, saved search results appear exactly as if they had just been obtained. On exiting the program, users are asked whether any searches are to be saved and whether searches that are still running are to be aborted or allowed to continue to completion (if the latter, the results can be viewed in a subsequent *ConQuest* session).

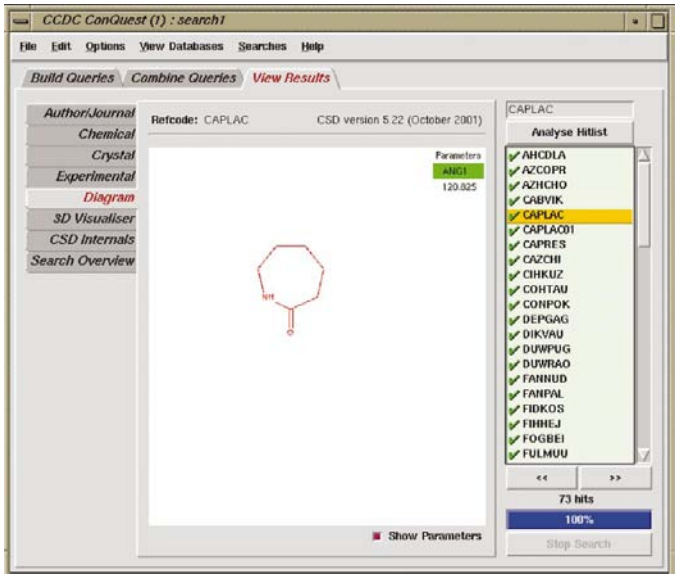

Figure 5
ConQuest 'View Results' pane showing the chemical diagram of hit entry CAPLAC (Winkler & Dunitz, 1975).

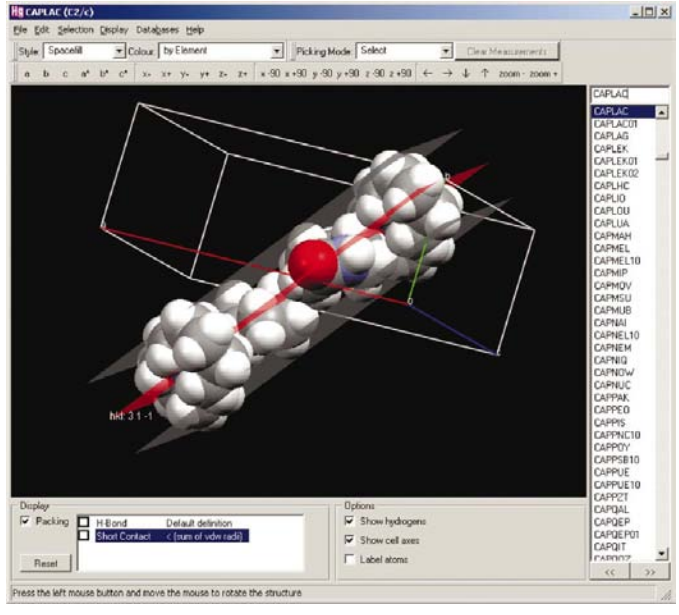

Figure 6
Mercury main window with display showing a slice through the crystal structure of caprolactam (Winkler & Dunitz, 1975).

2.4. Supported platforms

ConQuest is currently supported on Windows 98, ME, NT, 2000 and XP, RedHat Linux 6.2 and above (and equivalents), Solaris 2.6 and above, Irix 6.5 and above, and AIX 4.3 and above.

3. Mercury

3.1. Program language and architecture

The development of *Mercury* was started about 2 years after that of *ConQuest*, by which time we had begun to replace some of the Fortran code in *Thomas* by C++ classes (in particular, classes for holding and manipulating molecules and crystals). Consequently, we decided that *Mercury* would use no legacy code but would be entirely written in object-oriented C++. The decision to write *Mercury* in a different language from *ConQuest* reflects a change in the relative importance of two conflicting factors: the need to deliver software to users as quickly as possible, and the need to maximize the maintainability and extensibility of code. The former priority was dominant in the decision to use legacy Fortran code for *ConQuest*; the latter had become more important (and more realistically attainable) by the time *Mercury* development was begun.

We used the C++ Qt library (Trolltech AS, 1995) for building the GUI and OpenGL for three-dimensional graphics rendering. Because of the time required for compilation, development in Qt tends to be slightly slower than in the Python/Tk scripting environment, but the performance of the resulting interface is better (there being no use of interpreted languages). A major factor leading to our decision to use Qt was its signal/slot mechanism, which provides a way of achieving separation between the code needed to respond to user requests and the code needed to keep the GUI in an up-to-date state. Qt supports a wide variety of platforms and

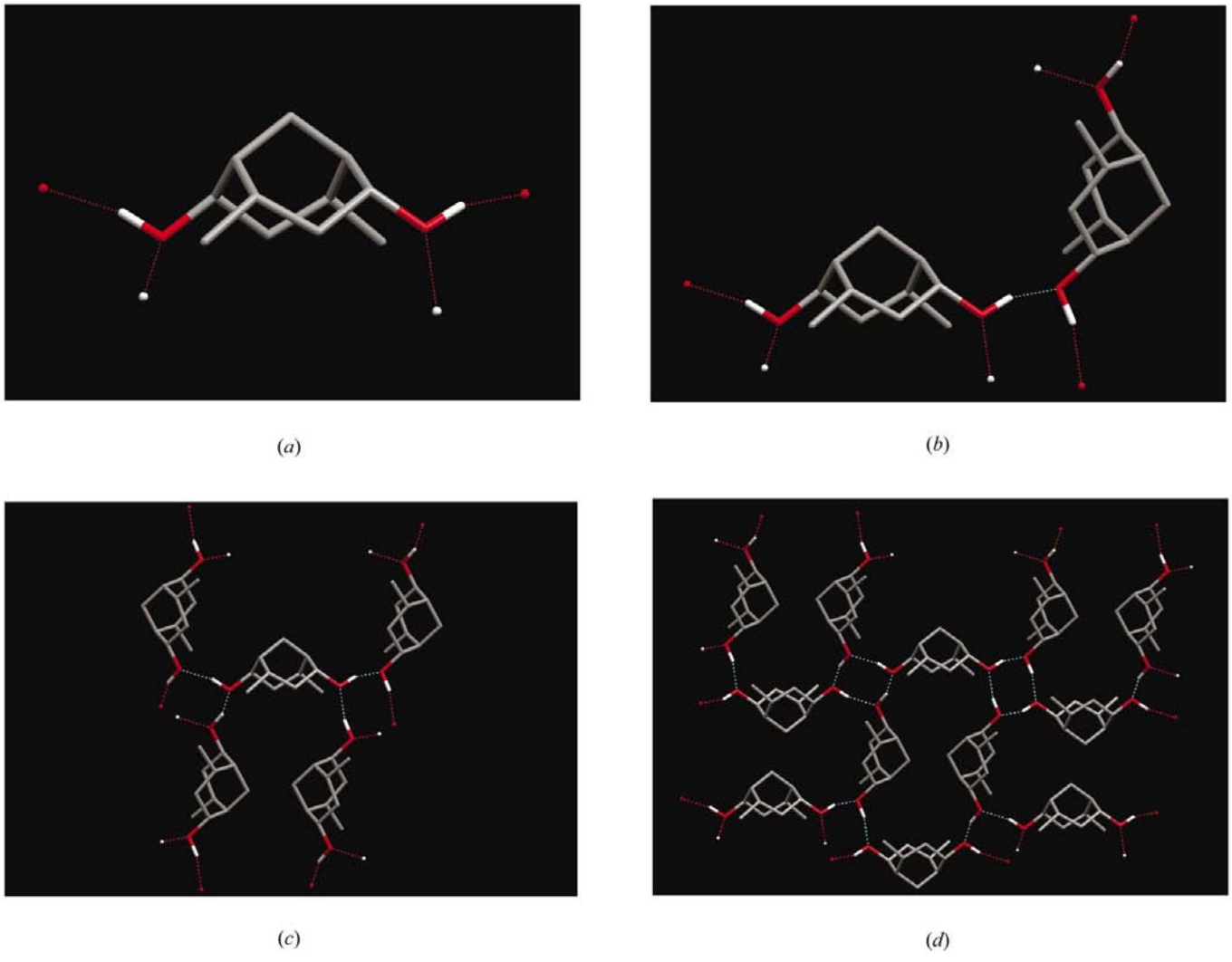

Figure 7
Mercury plots of (*a*) a single molecule of KAPNAQ (Nguyen *et al.*, 2000) with the hydrogen bonds it forms, (*b*) the same molecule with a single hydogen-bonded neighbour, (*c*) the original molecule with all hydrogen-bonded neighbours and (*d*) an extended hydrogen-bonded network.

provides easy solutions to a number of portability issues, such as platform-specific file-naming conventions.

While we remain comfortable with our choice of Qt, we nevertheless wish to keep as many options open as possible, should we need to move to different GUI libraries in future. *Mercury* was therefore designed to separate GUI-dependent from GUI-independent code. As an example, the program contains a Qt-specific class called QtRendererWidget and another class, Renderer, which knows nothing about Qt. A mouse event in the *Mercury* visualizer area (*e.g.* clicking at a certain position) is recorded by QtRendererWidget and passed to Renderer, which does the computation or data manipulation that has been called for (*e.g.* detecting that the click was on an atom and changing the state of that atom accordingly). Essentially, QtRendererWidget is a Qt-specific binding of the functionality provided by Renderer; it would be easy to replace it by a class specific to a different GUI library without changing the code in Renderer itself.

Similarly, the program is designed to separate graphics-related from non-graphics-related code. For example, Atom objects have attributes such as charge, radius and label but not display properties such as colour. These are contained within a GraphicsAtom object which contains an Atom together with additional visualization-relevant data. Atom objects can therefore be used for a wide variety of (non-visual) CSD-related applications and extended to GraphicsAtom objects when there is a need to render three-dimensional images.

3.2. User-interface design

GUI design was based on the same principles as outlined above for *ConQuest*, although the differing functionalities of the programs obviously lead to differences in the appearances of their interfaces. Fig. 6 shows a screenshot of the main *Mercury* window and an example crystal structure display. As befits the program's primary purpose, the bulk of the main window is given over to the three-dimensional graphics display. The list of CSD refcodes on the right mimics the View Results pane of *ConQuest*. Because of the highly interactive nature of *Mercury*, many options are accessible by clicking the right-hand mouse button in the display-area background. Almost all of these options are also accessible by other routes. The main window contains buttons and check-boxes for switching on and off very common options, *e.g.* display of H atoms, cell axes or short nonbonded contacts.

3.3. Program functionality

The primary purpose of *Mercury* is to provide advanced functionality for viewing crystal structures in three dimensions. This includes the following.

(i) The ability to load hit lists from *ConQuest* searches (including calling up *Mercury* directly from *ConQuest*; see above), or to browse the entire CSD or other databases in CSD format, or to read in crystal structures in other common formats (MOL2, PDB, CIF, MOL). On platforms that support drag-and-drop, structure files can be opened in *Mercury* by dragging the file name onto the *Mercury* desk-top icon or onto the display area of an open *Mercury* window. When run in `-client` mode, a *Mercury* session will listen for and respond to requests from other programs to open files (this is how data are transferred from *ConQuest* to *Mercury*).

(ii) The ability to rotate, translate and scale the three-dimensional crystal structure display and to view down cell axes, reciprocal cell axes and normals to planes.

(iii) The usual range of three-dimensional visualization options (different display styles, colouring and labelling options, ability to hide and then redisplay atoms, molecules *etc.*).

(iv) The ability to measure distances, angles and torsion angles.

(v) The ability to create and display centroids, least-squares mean planes and Miller planes.

(vi) The ability to display unit-cell axes and the contents of any number of unit cells in any direction (including fractions of unit cells).

(vii) The ability to display a slice through the crystal in any direction. For example, Fig. 6 shows molecules in the structure of caprolactam (Winkler & Dunitz, 1975) whose centroids lie within 2.5 Å of the $(31\bar{1})$ Miller plane. Displays such as this can be useful in rationalizing crystal morphology and predicting how to control it (Clydesdale *et al.*, 1997).

(viii) The ability to undo and redo actions.

(ix) The ability to save the display as a file of molecules or as a graphics image, and, on a PC, to copy images onto the clipboard. *Mercury* has its own binary format (the `.mry` file) which enables displays to be saved and then read back in again, such that all program settings (except view direction and scale) are retained.

Without doubt, the most important functionality in *Mercury* is the ability to locate, display and build networks of intermolecular and/or intramolecular hydrogen bonds, short nonbonded contacts, and user-specified types of contacts. The user begins by defining the types of nonbonded contacts that are of interest. Typical examples are (i) intermolecular hydrogen bonds between ammonium donors and carboxylate acceptors, (ii) any intermolecular contact shorter than the sum of the van der Waals radii of the atoms involved, or (iii) intermolecular or intramolecular Cl$\cdots$O contacts in the range 2–4 Å. A program option is then used to find all the contacts of the types thus defined that are formed by the molecules in the crystallographic asymmetric unit. For example, Fig. 7(*a*) shows the OH$\cdots$O hydrogen-bond contacts found by *Mercury* for a molecule in the crystal structure KAPNAQ (4,8-dimethylbicyclo[3.3.1]nonane-2,6-diol; Nguyen *et al.*, 2000). This starting point can then be used to build a network of contacts in the extended crystal structure, following the methodology of Motherwell & Shields (2000). For example, clicking on an individual contact with the mouse causes the neighbouring molecule to be shown in full (Fig. 7*b*). Clicking on an 'Expand All' button causes all neighbouring molecules to be shown (Fig. 7*c*). This feature can be used repetitively (Fig. 7*d*).

Judicious use of the contact-searching functionality in tandem with network expansion allows the packing in a crystal structure to be explored with ease. Molecules can be coloured

by symmetry equivalence (*i.e.* symmetry-related molecules assigned the same colour) to aid interpretation of crystal packing; this is particularly useful for structures with $Z' > 1$. Default definitions of hydrogen bonds and short nonbonded contacts are provided. This means that all key contacts formed by a molecule in a crystal structure can be found with one mouse click.

3.4. Supported platforms

Mercury is currently supported on Windows 98, ME, NT, 2000 and XP, RedHat Linux 6.2 and above (and equivalents), Solaris 2.6 and above, and Irix 6.5 and above.

4. Future directions

Although *ConQuest* and *Mercury* represent significant steps forward, particularly in making the CSD accessible to non-expert users, much remains to be done. Obviously, we will make further improvements to both programs. In addition, there are three main areas in which we seek to advance. First, we aim to replace most of our legacy code with a new object-oriented C++ Toolkit, containing classes for holding molecular and crystallographic data, performing geometry calculations, substructure searching, screening, symmetry generation *etc*. We have made significant progress with this already, as indicated by the fact that the new Toolkit provides all the crystallographic functionality in *Mercury*, but several person-years of effort are required for its completion.

Secondly, we need to improve the integration of our various programs at the user-interface level. In particular, core programs for searching and analysing the CSD, such as *ConQuest*, *Mercury*, *Vista* (CCDC, 1994) and *IsoGen* (Bruno *et al.*, 1997), have separate interfaces with different 'look-and-feel' characteristics (although this is less so for the two most recent programs, *ConQuest* and *Mercury*). We need to build common user-interface components that will be shared between the programs. This will improve both ease of maintenance and user-friendliness.

Finally, we will continue our development of 'knowledge bases' and applications software driven by these knowledge bases. Our aim is to extract key types of information from the CSD and make them available in derivative databases, so that they can be easily viewed by users at a GUI and also accessed by an application through a programming interface. The first example of such a product was *IsoStar* (Bruno *et al.*, 1997), which is a knowledge base of intermolecular interactions. It contains information about a huge variety of nonbonded contacts, taken not only from small-molecule (CSD) structures but also from protein–ligand complexes. *IsoStar* is used by the applications program *SuperStar* (Verdonk *et al.*, 1999, 2001) to predict binding 'hot spots' in enzyme active sites. The predictions are entirely knowledge-based, *i.e.* derive solely from *IsoStar* data. *SuperStar* therefore exemplifies a program that answers a specific question of relevance to drug design and does so by manipulating data derived ultimately and solely from a crystallographic database. Use of the CSD 'behind the scenes' in this way seems to us a most powerful way of exploiting crystal structure data and one which we intend to develop much further; for example, we are now developing a knowledge base of intramolecular geometry data.

Most of the staff of the Cambridge Crystallographic Data Centre, and many external users, helped in program development by providing feedback on interfaces. Frank Allen, Karen Lipscomb, Steve Maginn and David Watson contributed to user documentation. Frank Allen, Sam Motherwell and Greg Shields provided invaluable scientific advice. Lucia Rodriguez-Monge contributed to the CIF reader used by *Mercury* and Lucy Purkis to the MOL reader.

References

Adobe Systems Inc. (1987). *Adobe Acrobat Reader*. Adobe Systems Inc., San Jose, California, USA.
Allen, F. H. (2002). *Acta Cryst.* B**58**, 380–388.
Allen, F. H., Davies, J. E., Galloy, J. J., Johnson, O., Kennard, O., Macrae, C. F., Mitchell, E. M., Mitchell, G. F., Smith, J. M. & Watson, D. G. (1991). *J. Chem. Inf. Comput. Sci.* **31**, 187–204.
Allen, F. H. & Motherwell, W. D. S. (2002). *Acta Cryst.* B**58**, 407–422.
Awdry, W. V. (1946). *Thomas the Tank Engine*. Leicester, England: Ward.
Bruno, I. J., Cole, J. C., Lommerse, J. P. M., Rowland, R. S., Taylor, R. & Verdonk, M. L. (1997). *J. Comput. Aid. Mol. Des.* **11**, 525–537.
CambridgeSoft Corporation (2001). *ChemDraw*. CambridgeSoft Corporation, Cambridge, Massachusetts, USA.
CCDC (1994). *Vista – A Program for the Analysis and Display of Data Retrieved from the CSD*. Cambridge Crystallographic Data Centre, 12 Union Road, Cambridge, England.
Clydesdale, G., Roberts, K. J. & Walker, E. M. (1997). *Theoretical Aspects and Computer Modeling of the Solid State*, edited by A. Gavezzotti, pp. 224–226. Chichester: Wiley.
Hugunin, J., Schwaller, T. & Ascher, D. (1997). *PyOpenGL*, http://pyopengl.sourceforge.net/.
Lundh, F. (1999). *Tkinter*, http://www.pythonware.com/.
Isys Information Architects (1996). *Interface Hall of Shame*, http://www.iarchitect.com/.
MDL Information Systems, Inc. (2001). *ISIS/Draw*. MDL Information Systems Inc., San Leandro, California, USA.
Motherwell, W. D. S. & Shields, G. P. (2000). *RPLUTO*, http://www.ccdc.cam.ac.uk/prods/rpluto/.
Nguyen, V. T., Bishop, R., Craig, D. C. & Scudder, M. L. (2000). *CrystEngComm*, 7; see http://www.rsc.org/is/journals/current/crystengcomm/cecpub.htm.
Orpen, A. G. (2002). *Acta Cryst.* B**58**,
Ousterhout, J. K. (1994). *Tcl and the Tk Toolkit.* Reading, Massachusetts: Addison-Wesley.
Paul, B. & Bederson, B. (1996). *Togl*, http://togl.sourceforge.net/.
Redman, J., Willett, P., Allen, F. H. & Taylor, R. (2001). *J. Appl. Cryst.* **34**, 375–380.
ReportLab Inc. (1998). *ReportLab*. ReportLab Inc., Highland Park, New Jersey, USA.
Rossum, G. van (1991). *Python*, http://www.python.org/.
Taylor, R. (2002). *Acta Cryst.* D**58**, 879–888.
Telstra Corporation Ltd (1997). *Python Megawidgets*, http://pmw.sourceforge.net/.
Trolltech AS (1995). *Qt*. Trolltech AS, Oslo, Norway.

Verdonk, M. L., Cole, J. C. & Taylor, R. (1999). *J. Mol. Biol.* **289**, 1093–1108.
Verdonk, M. L., Cole, J. C., Watson, P., Gillet, V. & Willett, P. (2001). *J. Mol. Biol.* **307**, 841–859.
Winkler, F. K. & Dunitz, J. D. (1975). *Acta Cryst.* B**31**, 268–269.
Wippler, J.-C. (1996). *MetaKit.* Equi4 Software, Houten, The Netherlands.