Commission on Crystallographic Computing

International Union of Crystallography
http://www.iucr.org/iucr-top/comm/ccom/

Newsletter No. 1, January 2003
http://www.iucr.org/iucr-top/comm/ccom/newsletters/

Table of Contents

(This Issue's Editor: Lachlan Cranswick)

COMPCOMM Chairman's Message

In the sixties of the last century, when I started to write my own crystallographic software (on papertape), to be executed on a 1MHz air-conditioned room filling 'personal computer' in a language called ALGOL, we were largely unaware of similar but largely incompatible developments in other crystallographic departments. One of the important tasks of the IUCr computing commission in that period was to bring people working on crystallographic software together in order to exchange experience and ideas that turned out to be successful. The practical format for this was the crystallographic computing school that was organised in conjunction with the triannual IUCr congress. The concrete output of such a school was in the form of a book in which the various useful algorithms were described that were implemented on a particular platform. Others could subsequently use those successful algorithms in there own software. Much of the (early) SHELX source code can be understood with reference to those publications. Much has changed since that time and for that matter the role of the IUCr Computing Commission. Few young crystallographers are nowadays capable to read the (FORTRAN) source code of widely used programs such as the SHELXL refinement program and thus unable to modify it to suite their new application. A lot of currently available software is in binary format only and thus not extendable and with a build-in finite lifetime. Many of the pioneers in the field have now retired or will so soon. A new generation of knowledgeable scientific research software developers has to be trained in order to balance the current trend of commercial black-box software.

The new computing commission elected during the IUCr meeting in Geneva has choosen to provide a platform of easy information exchange using the rich facilities of the Internet. Thanks to the efforts of our commission member Lachlan Cranswick (who also manages the excellent CCP14 WEB-site full of information on free software) we can now offer the first issue of a newsletter full of largely unpublished papers on various computing topics. The planning is that this newsletter will appear twice yearly. I would like to thank all contributors to the current issue and invite you to send your contributions for the next issue of the newsletter to Lachlan.

Ton Spek (a.l.spek@chem.uu.nl)

From the Editor of Newsletter No. 1

In this first issue of the on-line IUCr Computing Commission newsletter is a variety of topics where there should hopefully be at least one article of interest to anyone involved in crystallographic computing. Not only to those who are curious to know what lies behind the available black-button on the black-box software tools; but also those who wish to become involved in the creation of crystallography algorithms and software. Specialist crystallographic programmers are becoming an isolated minority within a scientific world increasingly made up people who use crystallography as a tool. However, those who use crystallography as a tool are dependent on crystallographic programmers for the creation of powerful software tools that make their work possible.

I believe it is no coincidence that those most respected in crystallography are heavily involved in the creation of crystallographic algorithms and software. Not only can their software give them a leading edge over those who are not developing their own analytical tools, but freely distributed software can also impact greatly amongst the scientific community out of all proportion to the private, individual usage of these packages by only their respective authors. To those wishing to develop a career in research crystallography, (as opposed to a series of disjointed short term contracts relating to crystallography), it is worth considering that an interest in the development of new crystallographic algorithms and software could be very handy in maintaining employability, and enabling opportunities for developing new areas of scientific research.

Lachlan Cranswick (l.m.d.cranswick@dl.ac.uk)

THE IUCR COMMISSION ON CRYSTALLOGRAPHIC COMPUTING - TRIENNIUM 2003-2005

Chairman: Prof. Dr. Anthony L. Spek
Director of National Single Crystal Service Facility,
Utrecht University,
H.R. Kruytgebouw, N-801,
Padualaan 8, 3584 CH Utrecht,
the Netherlands.
Tel: +31-30-2532538
Fax: +31-30-2533940
E-mail: a.l.spek@chem.uu.nl
WWW: http://www.cryst.chem.uu.nl/spea.html

Professor I. David Brown
Brockhouse Institute for Materials Research,
McMaster University,
Hamilton, Ontario, Canada
Tel: 1-(905)-525-9140 ext 24710
Fax: 1-(905)-521-2773
E-mail: idbrown@mcmaster.ca

Lachlan M. D. Cranswick
CCP14
School of Crystallography, Birkbeck College,
Malet Street, Bloomsbury,
WC1E 7HX, London, UK
Tel: (+44) (0)20 7631 6850
Fax: (+44) (0)20 7631 6803
E-mail: l.cranswick@dl.ac.uk

Dr Vincent Favre-Nicolin
CEA Grenoble
DRFMC/SP2M/Nano-structures et Rayonnement Synchrotron
17, rue des Martyrs 38054 Grenoble Cedex 9
38054 Grenoble Cedex 9 – France
Tel: (+33) 4 38 78 95 40
Fax: (+33) 4 38 78 51 97
E-mail: vincefn@users.sourceforge.net

Dr Ralf Grosse-Kunstleve
Lawrence Berkeley Lab
1 Cyclotron Road,
BLDG 4R0230,
Berkeley, CA 94720-8235, USA.
Tel: 510-486-5713
Fax: 510-486-5909
E-mail: rwgk@yahoo.com

Prof Alessandro Gualtieri
Università di Modena e Reggio Emilia,
Dipartimento di Scienze della Terra,
Via S.Eufemia, 19,
41100 Modena, Italy
Tel: +39-059-2055810
Fax: +39-059-2055887
E-mail: alex@unimore.it

Prof Ethan A Merritt
Department of Biological Structure
University of Washington
Box 357420, HSB G-514
Seattle, Washington, USA
Tel: 206 543 1861
Fax: 206 543 1524
E-mail: merritt@u.washington.edu

Dr. Simon Parsons
School of Chemistry
Joseph Black Building,
West Mains Road,
Edinburgh, Scotland EH9 3JJ, UK
Tel: +44 131 650 5804
Fax: +44 131 650 4743
E-mail: s.parsons@ed.ac.uk

Dr. Bev Vincent
RigakuMSC
9009 New Trails Dr,
The Woodlands, Texas 77381-5209, USA
Tel: 281-363-1033
Fax: 281-364-3628
E-mail: brv@RigakuMSC.com

Consultants

Dr David Watkin
Chemical Crystallography,
Oxford University,
9 Parks Road,
Oxford, OX1 3PD, UK.
Tel: +44 (0) 1865 272600
Fax: +44 (0) 1865 272699
E-mail: david.watkin@chemistry.oxford.ac.uk

Dr Harry Powell
MRC Laboratory of Molecular Biology,
Hills Road, Cambridge, CB2 2QH, UK.
Tel: +44 (0) 1223 248011
Fax: +44 (0) 1223 213556
E-mail: harry@mrc-lmb.cam.ac.uk

Modern crystallographic algorithms and data structures (a very personal approach)

Joerg Bergmann,
Ludwig-Renn-Allee 14, D-01217 Dresden, Germany.
E-mail: email@jbergmann.de - *WWW:* http://www.bgmn.de

1. Introduction

We will discuss the structure of crystallographic data indepth. As explained, advanced crystallographic algorithms need advanced crystallographic data structures for success.

I started learning programming in 1978 on some main-frame like computers. Well, we had some interactive computer facilities in our X-ray lab, but the computer programs as written in those days demanded a non-intuitive amount of input; showing nearly no inner intelligence of chosing default values. I begun to change that. Thus, I learnt that an intelligent program which hides the user from the need of input is a step ahead for an interactive program.

By developing these programs for some years, my programs were enriched with a lot of non-numeric stuff. While dealing with lots of non-numeric code, I looked around for solutions. Cite: "The choice of data representation is much complicated, often; it is determined by more than only the possibilities. It is related to the operations which should be carried out on the data." [1] From this point on, arrays (the usual data representation in Fortran) became one minor data representation amongst dozens of other possible data representations. Arrays are most likely good for numerics. They can be ineffective for handling non-numeric data. A reflection table or table of atomic positions is partially of non-numeric content.

After two dozens of years of writing computer programs (one dozen years working with C) and learning the common standardizations for crystallographic data I feel that Niklaus Wirth's idea is valid for external data representations too. Most of the common crystallographic standardizations date back to the time of old Fortran coded mainframes. Storage area was restricted, and so was the possible code. The external data representations of those days correspond to this type of code. The crystallographic community has fixed the basic ideas of such external data representations as their standards. By doing so, the algorithms of those days were also fixed. The development of up-to-date algorithms using such antiquated external data representations is very expensive. I will explain this by some examples.

2. Representation of crystal structure

From a practical point of view, the common representation of atomic positions is somewhat strange. It defines an asymmetric part of the full unit cell, and this part is filled with (possibly partial filled) atomic sites. This assymetric unit is expanded to generate the full contents of the unit cell. This is a purely theoretical point of view to crystallography, dealing with some periodic distribution of arbitrary values. But, in practice, we deal with atoms in a variety of situations. By handling atoms in the assymetric unit, one needs extra code and data to:

- decide whether the atom is on a special position and on which type
- therefore decide what is the maximum occupation factor of the atom
- therefore decide about the symmetry conditions of the atom

To overcome these problems, I do not use such data representation in my programs. Instead, I have an external file containing ASCII information about all possible space groups. For example:

```
Wyckoff=i N=6
x 2*x 0
-2*x -x 1/3
x -x 2/3
-x -2*x 0
2*x x 1/3
-x x 2/3
```

is part of the entry for spacegroup no. 181. The expressions will be handled by the DATA module as mentioned below.

- In the structure description, one needs to give the space group (Hermann Mauguin) and the Wyckoff symbol for every atomic position.

- Thus, a difference to the common representation is that input data must define only the free parameters of the Wyckoff position. In the above case, only x=... must be defined.

- Another difference to the common representation is that for every Wyckoff position, a setting of 1 (one) atom is valid. No partial atom values are needed to define full occupancy. Setting of 0.5 always means 50 per cent probability of having an atom at this position.

As mentioned above, this code is able to generate the symmetry of the special positions automatically. So an anisotropic Debye-Waller factor can be set simply by typing

```
TDS=ANISO
```

The algorithms which do that will be described in detail in the next section.

3. Automatic generation of special positions symmetry

By using this new external format, automatic correction of the Debye-Waller factor symmetry becomes simple. As explained, we have one set of atomic coordinates for each atomic position, instead of multiple coordinates according to the multiplicity ratio of general and special positions. So, we have only one positive definite matrice Bij in the anisotropic Debye-Waller factor:

exp(-B11*h*h-B22*k*k-B33*l*l-2*B12*h*k-2*B13*h*l-2*B23*k*l)

The remaining task is to automatically adjust the symmetries of the Bij of the atomic positions according to the crystal symmetry. This will be done by the following algorithm:

- calculate the numeric coordinates of the first atomic position of the special position.

- use these for x,y,z of the general coordinate triplets.

- This will result in a set of general positions. A multiple of them (the ratio of general to special position) will coincide with each atomic position

- Compute the derivatives of each general position to x,y,z. You will get a 3x3 transformation matrix for each general position.

- So, you have a set of transformation matrices for each atomic position (the ratio of general to special positions again).

- From each atomic position's set of transformation matrices, construct the Γ^1 operator. For details, see [2].

- Set up one (parameterised) positive definite matrix.

- Transform it using the Gamma1 matrices for each atomic position.

You will get a set of positive definite matrices (number of atomic positions), which will automatically hold the symmetry of the crystal. Use the positive definite matrix of each atomic position while computing the Debye-Waller factor. You will get a set of Debye-Waller factors for each hkl, one for each atomic position. By doing so, the correct structure factor of the whole crystal will be computed.

4. Advanced external formats

Most standard data formats as fixed by the crystallographic community use Fortran-like input until now: a number at a fixed position on a punch-card like data sheet for each input value. I will explain a more advanced modern format:

- At first, we no longer use fixed positions and/or data sheets of fixed length (no more punch cards!).

- We use assignments as input. This is well known as DATA input in PL/1. By doing so, one may decide which set of information one will input. The program then tries to use default values for the remaining input. As a side effect, the sequence of the input is arbitrary:

```
A=3 B=4 C=5
```

is the same input as

```
B=4 A=3 C=5
```

- We use expressions instead of numbers for each value. And the input may contain arbitrary assignments, which will be held in a special data structure while the program runs. So:

```
A=3 i=1 B=A+i C=A+2*i
```

is fully equivalent to the previous input.

The assignments will be evaluated in order. In the case of the BGMN Rietveld software (http://www.bgmn.de/), this special data structure is the DATA module. By evaluating the data instructions in order, the input data itself is a kind of code. The input data/code must follow the rules of a specialized language, the input language of BGMN in our case.

The user of the program may define dependencies of input values, each from each other or from generalized variables. In Rietveld programs, you must give a key (also called codewords) to define which input values are to be refined. By using generalized variables, you may refine those variables instead of the native structure variables.

I will explain this in the following two sections.

5. Example: Chain modelling of organic molecules

This example gives the structure description of an organic compound from [3].

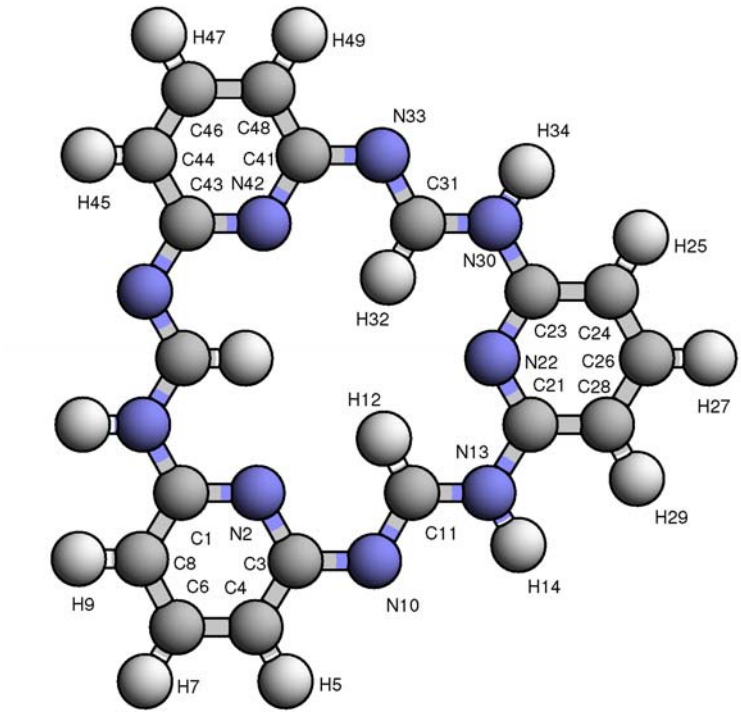


Fig 1: *Image of organic compound that can be defined by the following style of structure description.*

Comments start with //. The input data/code (structure description file) is as follows:

```
// part 1
// General crystal definition as to be set in every structure definition.
SpacegroupNo=11 TITEL=i24tn_tr1_14_b3_e
PHASE=i24tn_tr1_14_b3_e GEWICHT=SPHAR4
// Debye Waller Factors specific to the atoms H,C,N
PARAM=TDSH=0_0 PARAM=TDSC=0_0 PARAM=TDSN=0_0
PARAM=A=2.808_2.75^2.85 PARAM=B=1.462_1.4^1.5 PARAM=C=0.441_0.4^0.5
PARAM=BETA=90.05_89^91
RP=4 B1=ANISO PARAM=k1=0_0^1 PARAM=k2=0_0^0.00001
// General behaviour of the force field extension.
Theory=100.0 BondLevel=2
// Definition of bonded atoms. The force field will be computed
// for non-bonded atoms, only. Up to the 2nd bonding level each
// (see BondLevel=2).
Bondings=C1,N2,C3,C4,C6,C8,C1 Bondings=C3,N10,C11,N13,C21
...
Bondings=C51,H52 Bondings=N53,H54
// Definition of force field.
WW(aCC,-12,bCC,-6,(C1,C3,C4,C6,C8,C11,C21,C23,C24,C26,C28,C31,
C41,C43,C44,C46,C48,C51))
...
WW(cHNN,-12,dHNN,-10,(N50),(H54))
// Definition of some pseudo-positions in cartesian
// coordinates, which are useful for Eulerian rotation.
set(E0,0.000,0.000,0.000)
set(EX,1.000,0.000,0.000)
```

```
set(EY,0.000,1.000,0.000)
set(EZ,0.000,0.000,1.000)
// Definition of cartesian atomic coordinates of the
// (unfolded, unshifted) molecule.
set(O0,0.4800,0.0000,0.0600)
set(C1,0.0000,0.2425,0.2800)
set(N2,0.0000,0.2425,0.1400)
...
set(H54,-0.0000,0.1212,0.4600)
// Rotation of the molecule for 3 Eulerian angles rho1...rho3
PARAM=rho1=38.0_25.0^55.0 PARAM=rho2=71.0_55.0^90.0
PARAM=rho3=-12.0_-45.0^45.0
D(E0,EY,rho2,C1,N2,C3,C4,H5,C6,H7,C8,H9,N10)
...
D(E0,EY,rho2,N53,H54)
D(E0,EZ,rho3,C1,N2,C3,C4,H5,C6,H7,C8,H9,N10)
...
D(E0,EZ,rho3,N53,H54)
D(E0,EX,rho1,C1,N2,C3,C4,H5,C6,H7,C8,H9,N10)
...
D(E0,EX,rho1,N53,H54)
// Shift of the molecule for xx...zz
PARAM=xx=0.48_-0.48^2.8  PARAM=yy=0.0_-0.4^0.4 PARAM=zz=0.06_-0.40^0.4
T(xx,yy,zz,0,0,0,C1,N2,C3,C4,H5,C6,H7,C8,H9,N10)
...
T(xx,yy,zz,0,0,0,N53,H54)
// Shift of the single oxygen atom for xx0...zz0
PARAM=xx0=0.48_0.0^2.8 PARAM=yy0=0.0_0.0^0.4 PARAM=zz0=0.06_-0.1^0.4
T(xx0,yy0,zz0,0,0,0,O0)
// Rotation/torsion of parts of the molecule
// (folding of the molecular chain).
PARAM=rho0310=0_-90^90
D(C3,N10,rho0310,C11,H12,N13,H14,C21,N22,C23,C24,H25)
...
D(C3,N10,rho0310,H52,N53,H54)
PARAM=rho1011=0_-90^90
D(N10,C11,rho1011,N13,H14,C21,N22,C23,C24,H25)
...
D(N10,C11,rho1011,H52,N53,H54)
...
...
// Part 2: Definition of the Wyckoff positions. In this case,
// only general postions (Wyckoff=f). Note the following two points:
// -The atomic coordinates are computed by the 3 functions X()...Z(),
//  which are predefined by the Rietveld program BGMN
// -The (isotropic) Debye-Waller factors of the multiple H,C,N atoms
//  are neither independent nor generalized into a total Debye-Waller
//  factor of the structure. Instead of specific Debye-Waller factors
//  for each of H,C,N atoms were parametrized in part 1.
//  In following, this values are assigned to each TDS in part 2.
E=O Wyckoff=f x=X(O0) y=Y(O0) z=Z(O0) TDS=ANISO
E=C Wyckoff=f x=X(C1) y=Y(C1) z=Z(C1) TDS=TDSC
E=N Wyckoff=f x=X(N2) y=Y(N2) z=Z(N2) TDS=TDSN
...
...
```

6. Example: Real structure of the clay mineral kaolinite

The asymmetric bands of disordered clay minerals are described by hkl dependent peak broadening and shifting [4]. The disordering model of kaolinite [6] was tried to approximate as well as possible.

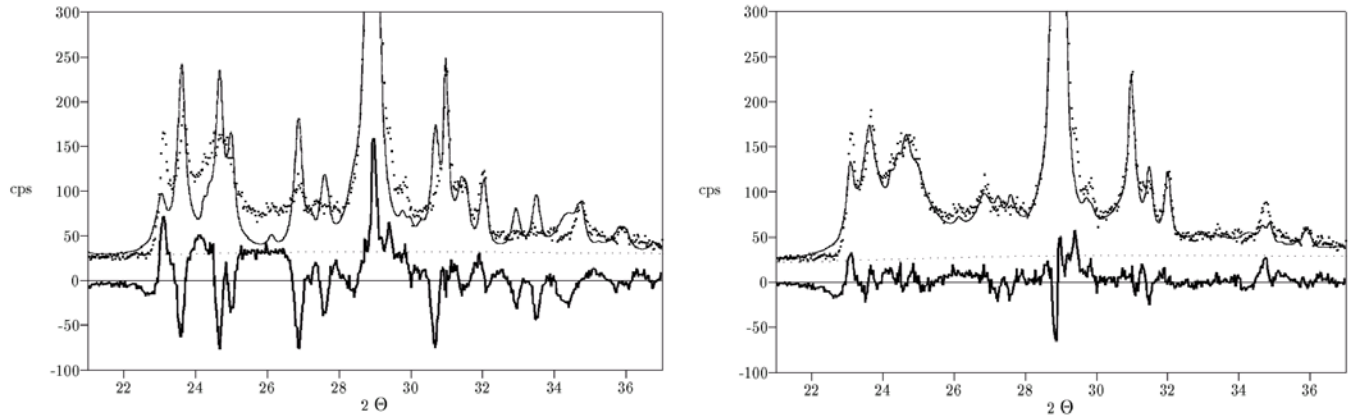


Fig 2 (a, b): *Rietveld plots of Koalinite quantitative analysis as modelled (a) without hkl dependent peak broadening (Rwp=19.8%, resulting kaolinite content = 73.0(8)%), and (b) with hkl dependent peak broadening (Rwp=9.0%, resulting kaolinite content = 89.5(4)%).*

The input data/code (structure description file) is as follows:

```
// part 1
TITEL=KAOLC1F SpacegroupNo=1 HermannMauguin=C1
PARAM=A=0.5155_0.5^0.52 PARAM=B=0.8945_0.88^0.91 PARAM=C=0.7405_0.72^0.75
PARAM=ALPHA=91.700_90^94 PARAM=BETA=104.900_100^107
PARAM=GAMMA=89.800_88.6^91
// RefMult=2: Each hkl causes two lines in the pattern,
// different shifts, different widths, different intensities,
// which generate strongly asymmetric peak shapes as common in clays
RefMult=2 GEWICHT=SPHAR6 RP=4
PARAM=g1=0.5_0^1
// different intensities
GEWICHT[1]=g1*GEWICHT GEWICHT[2]=(1-g1)*GEWICHT
GOAL:kaolfehlneu=GEWICHT
b10=ANISOLIN^0.1
// The length of the rots for the sub phase i (i=1,2) is b11*c1[iref]
PARAM=cb1=0.1_0^1
PARAM=b11=0.6^0.7_0
c1[2]==1 c1[1]=cb1
// The square of the additional with the squared lorentzian functions is
// c12sqr*sqr(c1[iref]*b11)
c12sqr==1
// Shift of the line is c2*sqr(c1[iref]*b11)
c2==1
// Change of lattice constants by tau
PARAM=c3=0_-3^3
PARAM=k2=0_0 // micro strain
// taua, taub ,tauc is the difference vector between tau as given in
//  "X-Ray Diffraction by Disordered Lamellar Structures"
//  Victor A. Drits - Cyril Tchoubar, Springer Berlin Heidelberg 1990
//  ISBN 3-540-51222-5, ISBN 0-387-51222-5
//  Kap. 8.1.5 Models for the Stacking Faults in Kaolinite
//  Pkt. 4: Model containing only enantiomorphic B layers, S. 245
// and the ideal translation b/3, multiplied with an arbitrary
// factor. This new tau should be a small value compared to
```

```
// lattice constants.
// tau in spherical coordinates
// as a difference to the above citation, tau has a significant
// value in c direction
PARAM=theta=0 PARAM=phi=0 PARAM=tau=0.2_-0.5^0.5
// tau in common coordinates
taua==tau*cos(theta)*cos(phi)
taub==tau*cos(theta)*sin(phi)
tauc==tau*sin(theta)
pi==2*acos(0)
// some helper variables
cosALPHA==cos(ALPHA*pi/180)
cosBETA==cos(BETA*pi/180)
sinGAMMA==sin(GAMMA*pi/180)
cosGAMMA==cos(GAMMA*pi/180)
K13==(cosALPHA*cosGAMMA-cosBETA)/A
K23==(cosBETA*cosGAMMA-cosALPHA)/B
K33==sqr(sinGAMMA)/C
// differnt widths B1,B2
// different shifts DELTAsk
// each depending from hkl as well as from iref
B1=cat(hklK==h*K13+k*K23+l*K33,hkltau==h*taua+k*taub+l*tauc,
b1k==K33*sqr(hkltau),
B2==k2*sqr(sk)+sqr(sqrt(sqr(sk)+
sqrt(c12sqr)*sqr(c1[iref]*ifthenelse(mod(k,3),b11+b1k,b1k)))-sk),
DELTAsk==sqrt(sqr(sk)+c2*sqr(c1[iref]*ifthenelse(mod(k,3),b11+b1k,b1k)))-sk+
c3*c1[iref]*K33*hklK*hkltau/sk,
b10+c1[iref]*ifthenelse(mod(k,3),b11+b1k,b1k)*abs(hklK)/sk)
GOAL=taua GOAL=taub GOAL=tauc
// part 2, not refined in this case
E=SI+4 Wyckoff=a x=0.9942 y=0.3393 z=0.0909 TDS=0.0044
E=SI+4 Wyckoff=a x=0.5064 y=0.1665 z=0.0913 TDS=0.0044
E=AL+3 Wyckoff=a x=0.2971 y=0.4957 z=0.4721 TDS=0.0083
E=AL+3 Wyckoff=a x=0.7926 y=0.3300 z=0.4699 TDS=0.0083
E=O Wyckoff=a x=0.0501 y=0.3539 z=0.3170 TDS=0.0071
E=O Wyckoff=a x=0.1214 y=0.6604 z=0.3175 TDS=0.0071
E=O Wyckoff=a x=0.0000 y=0.5000 z=0.0000 TDS=0.0071
E=O Wyckoff=a x=0.2085 y=0.2305 z=0.0247 TDS=0.0071
E=O Wyckoff=a x=0.2012 y=0.7657 z=0.0032 TDS=0.0071
E=O Wyckoff=a x=0.0510 y=0.9698 z=0.3220 TDS=0.0090
E=O Wyckoff=a x=0.9649 y=0.1665 z=0.6051 TDS=0.0090
E=O Wyckoff=a x=0.0348 y=0.4769 z=0.6080 TDS=0.0090
E=O Wyckoff=a x=0.0334 y=0.8570 z=0.6094 TDS=0.0090
```

In this case, only the lattice constants and the real structure will be refined. The ideal structure remains fixed.

7. Conclusion

The advantage of modern crystallographic data representations over traditional forms has been demonstrated. Representations of two totally different data sets (molecular chains, real structure of clay minerals) were given using the same frame set: the input language designed for the Rietveld program BGMN. Traditional crystallographic data represent a set of numbers. Advanced crystallographic data must represent a set of dependencies.

[1] Niklaus Wirth, "Algorithmen und Datenstrukturen (pascal version)", 5. edition, B.G. Teubner, Stuttgart (2000).
[2] J. Bergmann, T. Monecke, R. Kleeberg, "Alternative algorithm for the correction of preferred orientation in Rietveld analysis", J. Appl. Cryst. 34 (2001) pp. 16-19

[3] P. Friedel, J. Tobisch, D. Jehnichen, J. Bergmann, T. Taut, M. Rillich, C. Kunert, F. "Structure investigations of molecular crystals containing the ring cyclo-tris(2,6-pyridyl formamidine) by means of XPD and force field constrained Rietveld refinement", J. Appl. Cryst. 31 (1998), p. 874
[4] J. Bergmann and R. Kleeberg, Rietveld analysis of disordered layer silicates, proceedings of the 5th European Conference on Powder Diffraction (EPDIC 5) held in Parma, Italy, May 24-28, 1997, Mat. Sci. Forum, 278-281 (1998) part 1 pp. 300-305
[5] R. Kleeberg, J. Bergmann, Quantifizierung von fehlgeordneten Schichtsilikaten mit der Rietveld-Methode, Berichte der Deutschen Ton- und Tonmineralgruppe e.V. 5 (1997), pp. 35-44
[6] Victor A. Drits - Cyril Tchoubar, "X-Ray Diffraction by Disordered Lamellar Structures", Springer Berlin Heidelberg 1990

The Fortran 77 Cambridge Crystallographic Subroutine Library (CCSL).

P. Jane Brown,
Institut Laue Langevin, BP 156, Grenoble, Cedex France,
E-mail: brown@ill.fr – WWW: http://www.ill.fr/dif/ccsl/html/ccsldoc.html

Introduction

CCSL is a library of subroutines for doing crystallographic and related calculations. It was designed not so much as a system for determining crystal structures but as a toolbox to allow the crystallography of non-trivial structures to be easily introduced into user programs. The CCSL originated in the now defunct Crystallographic Laboratory of the Cavendish Laboratory, Cambridge in the early 1970's when the University first installed a computer which could interpret FORTRAN. It has subsequently been developed at the ILL by the present author, and at the Rutherford Laboratory by Judy Matthewman (now deceased), Bill David, and Bruce Forsyth amongst others. The main library now contains more than 500 subroutines and the master file some 50,000 lines of code. Full documentation is available on the web at http://www.ill.fr/dif/ccsl/html/ccsldoc.html.

There are three main areas in which a set of crystallographic subroutines can prove particularly useful to the solid state physicist; they are crystal symmetry, lattice geometry and scattering. The use of crystal symmetry is fundamental in CCSL; procedures are available to determine the constraints imposed by symmetry by the requirement that physical quantities, such as a bond length or an electron density, be invariant under the operations of the symmetry group. These determine the constraints, which must be, imposed in least squares refinements. For crystal geometry CCSL contains subroutines to manipulate vectors in real and reciprocal space which can be used in calculations of scattering angles, bond lengths etc. Subroutines, which calculate both atomic and magnetic structure factors and their derivatives with respect to the crystal parameters, are included. The system uses a common data structure to hold details of the crystal structure and the experiment being analysed. The data are given in the ***crystal data file*** which is a set of records up to 80 characters long; each identified by an initial letter indicating what kind of data it carries.

Use of CCSL

As noted above CCSL was designed to be used as a toolbox. The user is expected to provide a ***main*** program to do a particular job. This program should call CCSL procedures as necessary to set up the crystallography of the problem, do the required computation and report the results. The CCSL master file does however contain a number of main programs which have been written to accomplish common tasks and are maintained along with the library. They include programs for data analysis, straight forward crystallographic calculations such as structure factors and bond lengths, and several different types of least squares refinement. The modular nature of CCSL makes it easy to modify an existing main program or to write a completely new one since most of the work is done by calls to subroutines.

A simple example is given below of a program which could be used to design a diffraction experiment. It first finds reflections in a given range of $\sin\theta/\lambda$ and calculates a quality factor which measures their sensitivity to the information sought in the experiment. This list in then sorted in order of decreasing quality factor. For each non-zero entry, the set of equivalent reflections which are accessible with the chosen diffractometer geometry, is generated and printed.

```
      PROGRAM EXAMPL
      C
      CH Example program to demonstrate use of CCSL
      C
            PARAMETER (NREFS=100)
            DIMENSION H(3,NREFS),K(3),QUAL(NREFS),IP(NREFS),HSYM(3,24),PH(24)
           &,ANG(4)
            LOGICAL NOMORE,LERCHK,VISIBL,ABSENT
            COMMON /IOUNIT/LPT,ITI,ITO,IPLO,LUNI,IOUT
      C
      C Setting up phase
            CALL PREFIN('EXAMPL')     !Initialise and read crystal data file
            CALL SETFC                !Prepare for structure factor calculation
            CALL SETDC                !Read diffraction geometry
            CALL SYMUNI               !Makes an asymmetric unit in recip space
            CALL FUDGE                !Fudges the LSQ pointers (see later)
      C
            LUNO=NOPFIL(2)
            CALL ASK('Give limit in sin theta/lambda')
            CALL RDREAL(SLIM,1,IPT,20,IER)
            IF (IER.NE.0) STOP 'Error giving limit'
            CALL ASK('Number of equivalents wanted')
            CALL RDINTG(NUMEQ,1,IPT,20,IER)
            IF (IER.NE.0) STOP 'Error giving number'
            CALL SETGEN(SLIM)         !Prepare for reflection generation
      C Stop if there have been errors in the set-up phase
            CALL ERRCHK(0,0,'in EXAMPL')
      C End of set-up
      C
            NUM=1
          1 CALL GETGEN(H(1,NUM),NOMORE) !Generate hkl within an asymmetric unit
            IF (NOMORE) GO TO 2
      C Calculate quality factor
            QUAL(NUM)=QFACTR(H(1,NUM))
            IF (QUAL(NUM) .LT. .0001) GO TO 1
            IF (LERCHK(2,NUM,NREFS,-1,'reflections generated increase NREFS'))
           & GO TO 1
      C
          2 WRITE (LPT,11)
         11  FORMAT (/'    h   k   l   theta     nu     rho     QFactr')
            NUM=NUM-1
            CALL SORTX(QUAL,IP,NUM)
            DO 3 N=NUM,1,-1
              NP=IP(N)
              NUMES=1
              CALL SYMREF(H(1,NP),HSYM,NSYM,PH) !Generate equivalent indices
              DO 4 J=1,NSYM
      C ANGD3 is not in the main library it is a subroutine of the
      C main program GENREF
                CALL ANGLD3(HSYM(1,J),ANG,ABSENT)
                IF (ABSENT) GO TO 4
      C visible  should return .TRUE. if angles ANG are accessible.
      C          IF (.NOT.VISIBL(ANG)) GO TO 4
                CALL INDFIX(HSYM(1,J),K)
      C Report results
                WRITE (LUNO,12) K
         12     FORMAT (3I5)
                WRITE (LPT,10) K,(ANG(I),I=2,4),QUAL(NP)
         10     FORMAT (1X,3I4,3F8.2,2X,F10.4)
```

```
            NUMES=NUMES+1
            IF (NUMES.GT.NUMEQ) GO TO 3
        4   CONTINUE
        3 CONTINUE
          STOP
          END
C
C
      SUBROUTINE FUDGE
CH Fudges the pointers so that the only derivative calculated is
CH The x parameter of the first atom
C
      COMMON /PRBLEM/NFAM,NGENPS(6,1),NSPCPS(6,1),
     & LF1SP(5),LF3SP(10,1,1),LVFST1(6,1,1),
     & LBFST1(6,1,1),NVARF(6,1,1),
     & NBARF(6,1,1),LF6SP(3,1)
      COMMON /POSNS/NATOM,X(3,250),KX(3,250),AMULT(250),
     & TF(250),KTF(250),SITE(250),KSITE(250),
     & ISGEN(3,250),SDX(3,250),SDTF(250),SDSITE(250),
     & ATESDS,KOM17
       LOGICAL ATESDS
C
C Zero offsets
      LVFST1(2,1,1)=0
      NVARF(2,1,1)=1
C Derivative 1 to be x of atom 1
      KX(1,1)=1
      RETURN
      END
C
C
      FUNCTION QFACTR(H)
      DIMENSION H(3)
      COMMON /FCAL/FC,DERIVT(200),FCMOD,COSAL,SINAL,FCDERS(200)
      COMPLEX FC,DERIVT
C This could be anything, but for simplicity assume we want reflections
C sensitive to the x parameter of the first atom in the atom list
      CALL LFCALC(H)    !Calculate structure factor and derivatives
      QFACTR=FCMOD
      IF (FCMOD .GT.0.0001) QFACTR=ABS(FCDERS(1)/FCMOD)
      RETURN
      END
```

Modifying the tools

The main programs provided with CCSL can rather easily be used to try out ideas for a new formulation of some physical process. It could be a new extinction model, a different form-factor representation, a new peak-function in profile refinement etc. The first step is to identify the subroutine or subroutines which are involved, usually there is a setting up routine, a calculation routine and, for least squares applications, one which deals with shifts in the parameters. They may often be combined in a single multiple entry subroutine. These routines must then be modified, without change of name, to reflect the new formulation. After compilation the new subroutines are included in the link step before the CCSL library search so that they are used rather than the originals.

The CCSL Master file

CCSL is written in FORTRAN 77 and conforms very closely to the ANSI standard (X3.9-1978). This has many advantages as it makes the system relatively easy to install on diverse operating systems. It does however have several disadvantages associated with the FORTRAN language. In particular the sizes of all arrays have to be fixed when the library is compiled. To allow maximum flexibility while minimising problems of maintenance, the complete set of CCSL routines and main programs is held in a *Master File*.

The code in the master file is *not quite* FORTRAN 77, as in addition to the FORTRAN code it also contains control *codes* which allow:

1. The code to be split into sections. Current sections include:

   LIB the main subroutine library
   MAI main programs which use only routines from LIB
   PR the profile refinement library
   PF library of peak fitting functions
   PMA main programs for profile refinement
   PIG graphics routines for different types of hardware and software

2. The dimensions of some arrays to be varied.

   The arrays used for storing information about atomic positions, numbers of form-factors, maximum sizes of Fourier maps etc, are fixed by default to relatively modest sizes so as not to produce excessively large storage requirements. These quantities are held as symbolic parameters in the master file so that they can be customised when building a library.

3 The labelled COMMON blocks to be kept consistent

   Within the master file the labelled common blocks used by each subroutine are indicated by including just their names e.g.:

   /IOUNIT/

   /SYMDA/ and so on.

   When the library is built these are replaced by the full COMMON declarations which are held in a single block near the top of the master file, and the appropriate substitution for symbolic variables is made

4 System dependent features to be customised:

   Lines which are operating system or site dependent are tagged with labels of the form CWXYZ where WXYZ may be any of

LAX	Use an extension to FORTRAN 77 recognised by most compilers
PICKY	To enforce strict compliance with the FORTRAN 77 standard
[-]VMS	[don't] Include only if the operating system is VMS
[-]UNIX	[don't] Include only if the operating system is UNIX
ILL	Use ILL preferences: (extension for crystal data files is .cry listing file named *program_name*.lis)
RAL	Use RAL preferences: (extension for crystal data files is .ccl name for listing file is demanded interactively)

A set of ***perl*** scripts is used to manipulate the master file. They can be downloaded along with the master file by anonymous ftp from ftp://ftp.ill.fr/pub/dif/ccsl/.

A cross-platform crystallographic graphical interface with wxWindows.

Richard Cooper,
Chemical Crystallography Lab, 9 Parks Road, Oxford, OX1 3PD, UK.
E-mail: richard.cooper@chem.ox.ac.uk and WWW: http://www.xtl.ox.ac.uk/

CRYSTALS (a single crystal diffraction refinement and analysis software package), and Cameron (software for production of crystallographic diagrams) are both maintained by staff in the University of Oxford's Chemical Crystallography Laboratory. Both programs have had a graphical user interface added in order to take advantage of the facilities available in a windowed environment. This work was carried out as a student project, and a Microsoft Windows version of CRYSTALS and Cameron was released into the world in August 1999. The GUI was developed in C++ using classes from the Microsoft Foundation Classes (MFC) and it is currently being ported to wxWindows (http://www.wxwindows.org/) with the aim of releasing Linux and Macintosh versions.

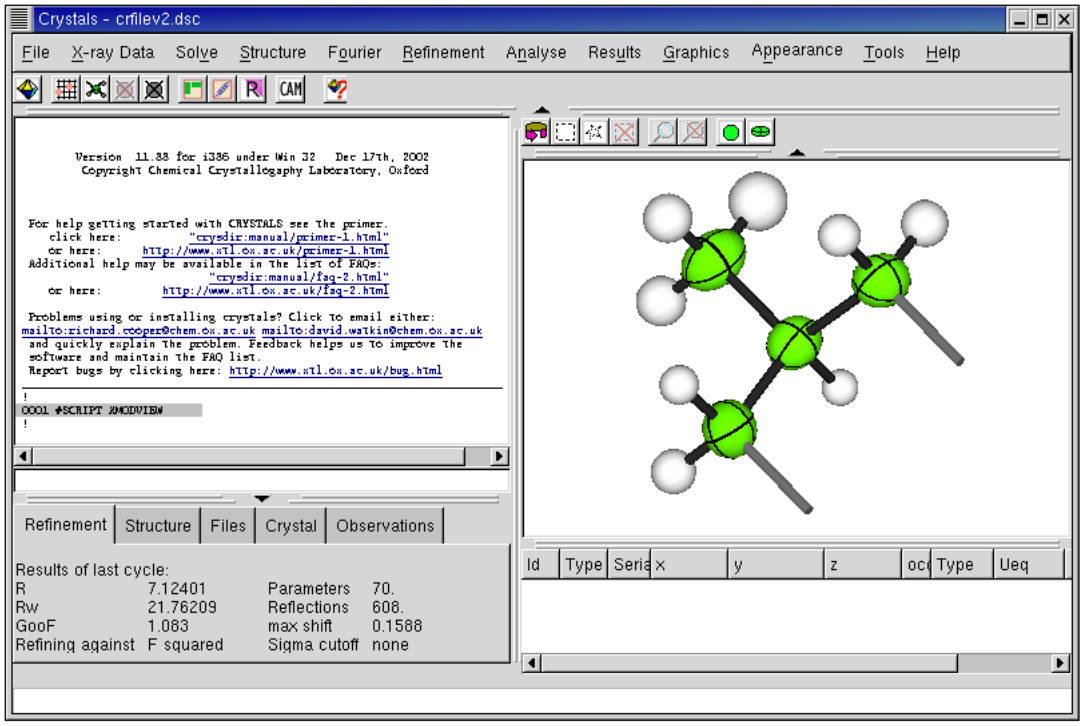


Figure 1: *CRYSTALS running under X on Linux using the wxWindows library.*

Toolkits and libraries

There are three mature, free or free-ish, cross-platform C++ GUI class libraries available that are being actively developed and supported: wxWindows, Qt and V.

wxWindows is a C++ class library that allows developers to create graphical C++ programs using a common API across a range of different platforms (most importantly for us: Linux, Windows and Macintosh). It is distributed with source code under a modified L-GPL licence, meaning that developers can distribute software which *uses* the library without having to release source code resulting in an active user base of academics, commercial companies and open-sourcers. Rather than providing the lowest-common-denominator across all platforms, wxWindows often re-implements controls that are missing in one platform and provides access to platform specific functionality where it might reasonably be required (e.g. Taskbar icons under Windows). As a result it has a rich set of GUI classes comparable to Borland's OWL or Microsoft's MFC.

V (http://vgui.sourceforge.net/) is very similar in ambition to wxWindows but seems to be a less active project with fewer developers and users.

Qt is also a cross-platform C++ toolkit (http://www.trolltech.com/) and like wxWindows it uses the 'native look-and-feel' for the user interface on each platform. Event tables are replaced with a signal-slot mechanism with the result that programs require pre-compilation with a meta-object compiler to produce valid C++. The Unix library version is GPL'd and is used by the KDE desktop environment for UNIX. However the free Windows version lags well behind the current commercial release and the licensing is restrictive (it is only free if developing open-source software, *i.e.* **not** LGPL) putting it beyond the use of many academic developers who have neither permission to GPL their code, nor funding to purchase commercial licenses for graphical interface libraries (but that's another argument).

Where wxWindows is most impressive is in its support for a wide variety of compilers and environments – on Windows alone the library and applications can be built with no less than 8 different compilers, including the free cygwin and mingw32 compilers and even some old 16-bit compilers running on Windows 3.1! On UNIX it will happily link against the GTK or Motif/Lesstif libraries. The developers resist moves to use new C++ language features, such as templates and exceptions, as this would make the library *less* portable. This level of commitment to compatibility is extremely reassuring for the future maintenance of your software.

Why not use Java or Python? There is a lot of non-platform specific CRYSTALS code already in C++ and a huge amount of crystallographic Fortran code, a third language would require us to either port the C++ or give us a headache trying to compile and link three languages!

Crystals original GUI design

The usual way to use wxWindows (or MFC for that matter) is to treat it as a *framework* that provides the application with all the services it requires (graphical interface, string and file handling etc) and also runs the program for you (start-up, event handling). We have attempted to eschew this pervasive reliance on a single product, and instead just use the wxWindows library where it is required (graphics and events).

The graphical user interface for CRYSTALS was originally designed with portability in mind: a Macintosh version was developed in parallel and though never released it ensured that the code was well structured for cross platform development.

CRYSTALS and its GUI, as implemented using the MFC class library is a many layered thing. It may be divided into four layers with the science going on in the bottom layers (1 and 2) and the user interacting with the top layer (4):

1. At the lowest level are the core crystallographic functions – accessed by instructions for manipulating and storing data (*e.g.* `#RESTRAIN / DIST 1.54,0.01=C(1) TO C(2) / END`). Such instructions may be issued by the user if they know what they are doing and/or have read the manual.
2. A macro programming language, SCRIPTS, for: interacting with the user, analysing data, making decisions and constructing low-level instructions on behalf of the user. SCRIPTS make it easy to use the program without knowing the syntax, as all required information can be prompted for. SCRIPT macros are stored in text files and have a PASCAL-like syntax so they can be altered by the clued-up user.
3. A GUI manager. Controls all user input and output of the program. As well as simple text I/O the GUI manager parses commands from the lower levels to create, layout and display dialog boxes and windows (Figure 2). The language for defining GUI objects has no decision making capabilities, but it can be issued from the SCRIPT macros, thereby allowing detailed control over what is displayed. Events occurring in the GUI (*e.g.* button presses, menu selection, text entered)

are passed down to the lower levels where they will be processed as SCRIPT input or straight low-level instructions depending on the current state of the program.

4. The GUI itself consists of controls (buttons, checkboxes, drawing areas, edit fields, menus *etc*.) each of which is derived from a platform specific class (e.g. CButton in MFC). A portable API is maintained by insisting that only added functions of the derived class may be called. For example, we may not directly call the *SetWindowText(char *)* member function of a derived CButton, but instead provide a generic public function *SetText(char *)* which does the same job, but will be the same on all platforms. Following this design isolates all platform specific code in the derived object classes, and porting the application should simply be a matter deriving a full set of GUI classes.

The creation, layout and management of windows and dialog boxes from the SCRIPT language gives us a tool which allows rapid development and testing of ideas by anyone using the program – SCRIPTS are interpreted at run-time so no recompilation is required.

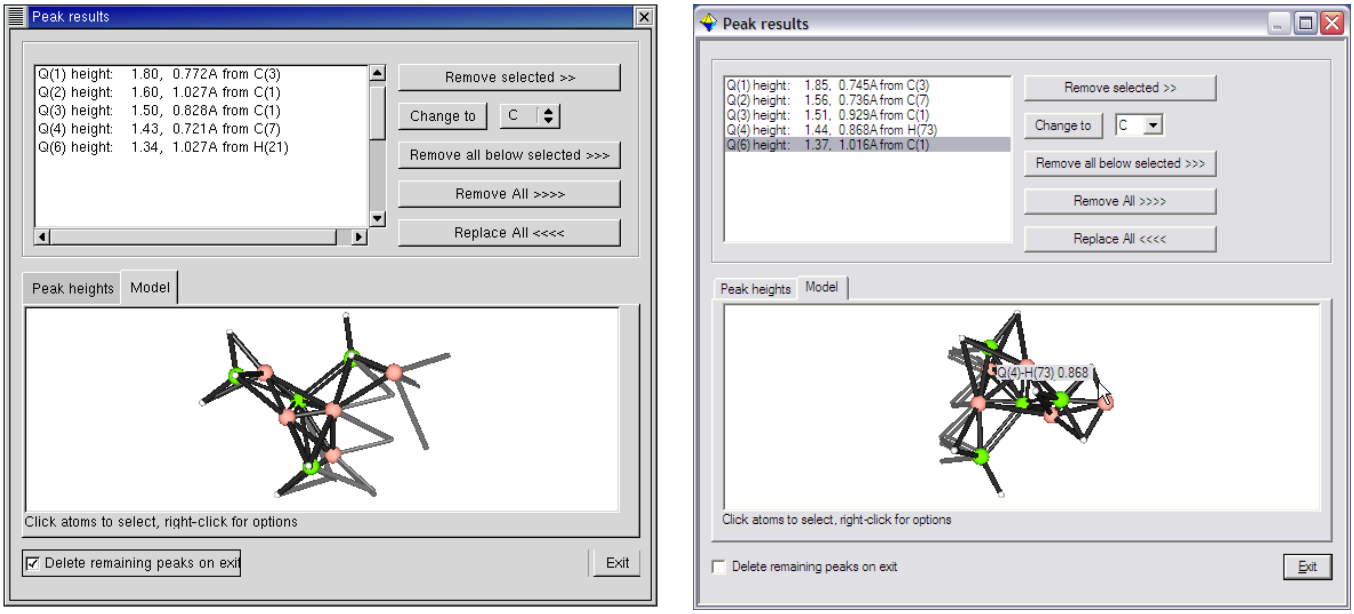


Figure 2: *Layout code provides portability – Redhat Linux (left), Windows XP (right)*

Porting from MFC

It must be emphasized that while we could immerse ourselves in the wealth of wxWindows facilities – free tools for designing dialog boxes, automatic layout of controls, and platform independent thread, file, drag-and-drop and network functions – this is not the approach we have taken. Instead we are currently regarding wxWindows as just another platform providing derivable C++ classes alongside the working Windows MFC and never-finished Mac platforms. This approach is important to keep levels 1-3 of the code *platform and library* independent. Clean separation of different levels is a common philosophy in CRYSTALS and one which should ultimately payoff in terms of maintainability: presently the GUI (levels 3 and 4) can be stripped off completely and a working command line version compiled with a few keystrokes. Similarly the SCRIPT parser (level 2) can be discarded with no effect on the crystallographic core of the program.

If you are porting an entire MFC framework application to wxWindows the recommended path is to recreate the applications user interface using available tools, and then transfer the guts of the code across from MFC to wxWindows. It is not recommended to try to run the two frameworks at once.

Upon perusing the wxWindows documentation (which, for a free software project, is some of the most complete and helpful ever found) the resemblance of the class hierarchy to MFC is immediately

noticeable. Instead of CWinApp, we have wxApp, wxFrame instead of CFrameWnd, wxButton instead of CButton, and even wxPaintDC and wxMemoryDC in place of their MFC device-context cousins. Names of member functions and the order of parameters differ slightly, but the overall design is very similar. For instance to create a new instance of an edit control in each the two libraries:

```
MFC:
                CEdit* myEd = new CEdit();
                myEd->Create(style, sizeRect, parent, id)
wxWindows:
                wxTextCtrl* myEd = new wxTextCtrl();
                myEd->Create(parent, id, label, position, size);
```

and the event handling of the two frameworks using macros is analogous:

```
MFC:
                BEGIN_MESSAGE_MAP( DerivedEditBox, CEdit )
                    ON_WM_CHAR()
                    ON_WM_KEYDOWN()
                END_MESSAGE_MAP()
wxWindows:
                BEGIN_EVENT_TABLE( DerivedEditBox, wxTextCtrl )
                    EVT_CHAR( DerivedEditBox::OnChar )
                    EVT_KEY_DOWN( DerivedEditBox::OnKeyDown )
                END_EVENT_TABLE()
```

MFC wraps the Win32 API very thinly in places, with the result that you must often understand the underlying Win32 to write code using MFC calls. In these cases wxWindows actually makes functions easier to use. The following fragments of code return the height, in pixels, of some text in a control. In the wxWindows version there is no need to know about device context handles or make Win32 API calls:

```
MFC:
                CString text;
                SIZE size;
                HDC hdc= (HDC) (GetDC()->m_hAttribDC);
                GetWindowText(text);
                GetTextExtentPoint32(hdc, text, text.GetLength(), &size);
                return (size.cy+5);
wxWindows:
                int cx,cy;
                GetTextExtent( GetLabel(), &cx, &cy );
                return (cy+5);
```

In a similar vein, one of the most useful wrappers is wxGLCanvas. OpenGL initialisation, requiring PixelFormatDescriptors and GLContexts, is thankfully hidden away – you simply create the wxGLCanvas and start issuing OpenGL commands.

Gotchas

Though the documentation is excellent for the mature core of the library, some newer classes, like wxGLCanvas are not so well documented. In these cases, the next best source of information is probably the source codes in the demo and samples subdirectories – there are plenty – find an example that uses the class of interest and see how it works. Failing that, the advantage of having the source is that you can step through the library code itself to see what is going on.

It is always worth compiling and testing things in the samples directory when you first install wxWindows – it can save you hours of hacking at your own code in vain only to later find that *nothing* compiles anyway because something is setup incorrectly.

From release to release some member functions may change slightly. This shifting API is the downside of not using a commercially supported product, such as Qt, where stability is considered much more important. Nevertheless, the changes are never large, and as it is now over ten years old, the underlying design of wxWindows is fairly well established and will not change that much.

A problem that may crop up from time-to-time is actually due to the similar style of MFC and wxWindows: it is easy to assume that you know what a member function does because it has a similar name. For example, MFC's CString::Format function is a member function which will set the text of the CString object: str.Format("%d",123); while wxWindows' Format function is static and returns a wxString, so you would say instead: str = wxString::Format("%d",123); Tread carefully.

Summary

In summary, wxWindows is similar to MFC offering a similar class hierarchy, event tables, and even things like device contexts for drawing on. In the case of CRYSTALS this was enough to make porting our MFC application to Linux fairly painless. Where the two libraries diverge, it is usually to the advantage of wxWindows – smaller executables when statically linked, simpler calls, and most importantly of all cross-platform support!

Web Links

WxWindows: http://www.wxwindows.org/
CRYSTALS: http://www.xtl.ox.ac.uk/crystals.html
List of free GUI toolkits (not just C++): http://www.atai.org/guitool/
Porting MFC Applications to Linux: http://www-106.ibm.com/developerworks/library/l-mfc/?n-l-4182

Acknowledgements

The initial design and implementation of the GUI and layout manager was carried out under the supervision of David Watkin in the Chemical Crystallography Laboratory, University of Oxford and in collaboration with Ludwig Macko and Markus Neuburger at the University of Basel, Switzerland.
The CCDC funded the DPhil project during which the initial GUI was developed.
EPSRC grant GR/N06830/01 funded the project Crystallography for the Next Generation.

Portable graphic interface for the Fortran program Jana2000.

Michal Dusek and Vaclav Petricek,
Institute of Physics, Academy of sciences of the Czech Republic, Na Slovance 2, 182 21 Praha 8, Czech Republic.
E-mail: dusek@fzu.cz and petricek@fzu.cz; WWW: http://www-xray.fzu.cz/jana/jana.html;
FTP: ftp://ftp.fzu.cz/pub/cryst/jana2000/

Introduction

Jana2000 [1] is a crystallographic computing system with graphical user interface freely available in http://www-xray.fzu.cz/jana/jana.html. It runs on UNIX and Windows platforms without any change of the look and feel (see Fig 1) and with exactly the same way of control. The program is distributed as compiled binary file for Windows and as a source code for UNIX. The Windows version is developed with Lahey Fortran 95 [2] and Winteracter [3].

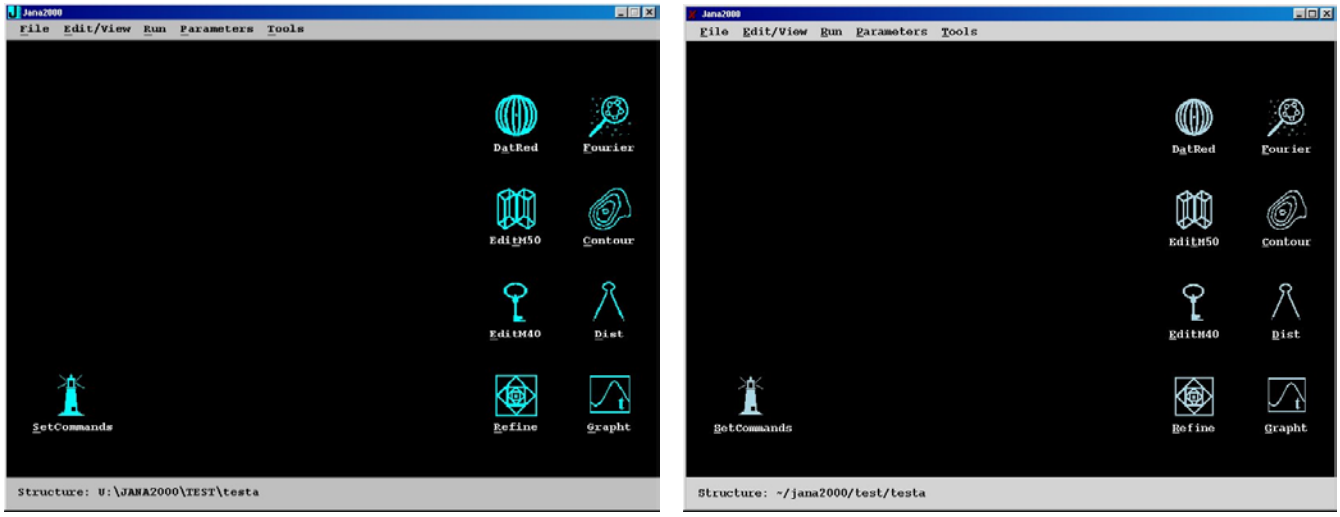


Fig 1: *The basic window of Jana2000 in Windows (a) and UNIX (b) environment.*

Most of the system is written in FORTRAN77 and is common for both UNIX and Windows platforms (see Fig. 2). The development of the graphic interface started in the early 90's when the portability of higher level graphics libraries between various UNIX platforms was rather limited. Moreover, the version for Windows ran in a DOS mode with completely different graphics possibilities. Therefore the most convenient way was to use only primitive graphics elements like plotting of line, filling of area, testing of keyboard and mouse events etc. and create own higher level graphics objects. With this access the same look and behaviour of the graphic interface was guaranteed under UNIX and DOS.

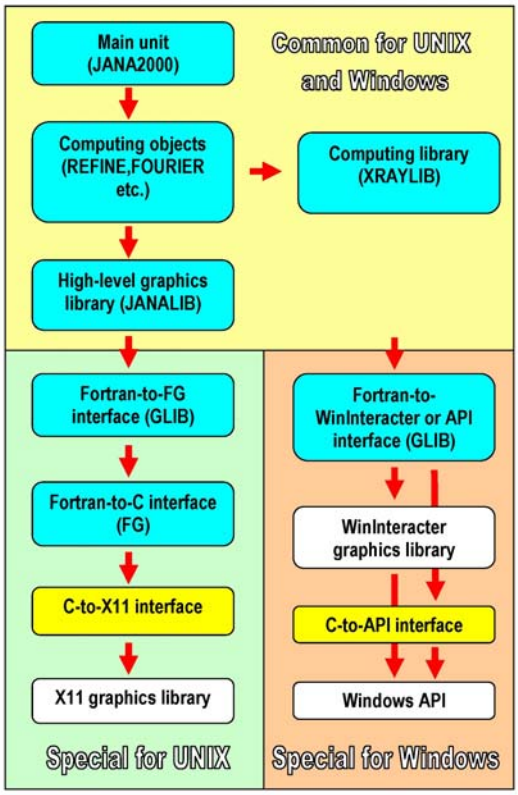


Fig. 2: *Flow diagram of Jana2000. The blue and yellow objects are part of Jana2000 and they are written in FORTRAN77 and C language, respectively. The white objects must be available in the computer (X11 and Windows API in the user side, Winteracter only for compilation of the Windows version).*

At present DOS emulation is no more used for Jana2000 and the problem of a platform independent graphics could be solved using for instance OpenGL libraries. However, we decided to keep our original graphic objects for the following reasons:

- They are independent on any other software.
- Their control is very similar to that we could get with professional package.
- They are fast and reliable.
- They do not require any preliminary installation at the user side.

Description

Jana2000 runs in a typical event loop, meaning that it waits - after initialization of the graphics and creating of the basic window - for the keyboard, mouse or expose events. Those events trigger an action, typically opening of a user dialogue or starting of some calculation. In order to keep the graphics fast and simple Jana2000 uses only one window for all purposes; user menus, forms and other objects are plotted in that window instead of using child windows as typically do professional graphic systems. Another maybe more limiting difference of standard systems is that cut-and-paste functions utilizing clipboard are not possible with graphics of Jana2000. We hope to activate these functions in future.

The high-level objects, graphics primitives and events testing can be used elsewhere in the common part of Jana2000 (Fig. 2) as the corresponding procedures are written in FORTRAN. Every such graphical request is processed in the following sequence:

Calling procedure -> JANALIB -> GLIB, where JANALIB containing the high level objects (Fig. 3) is common for UNIX and Window environment while GLIB that calls graphic primitives and event testing is different.

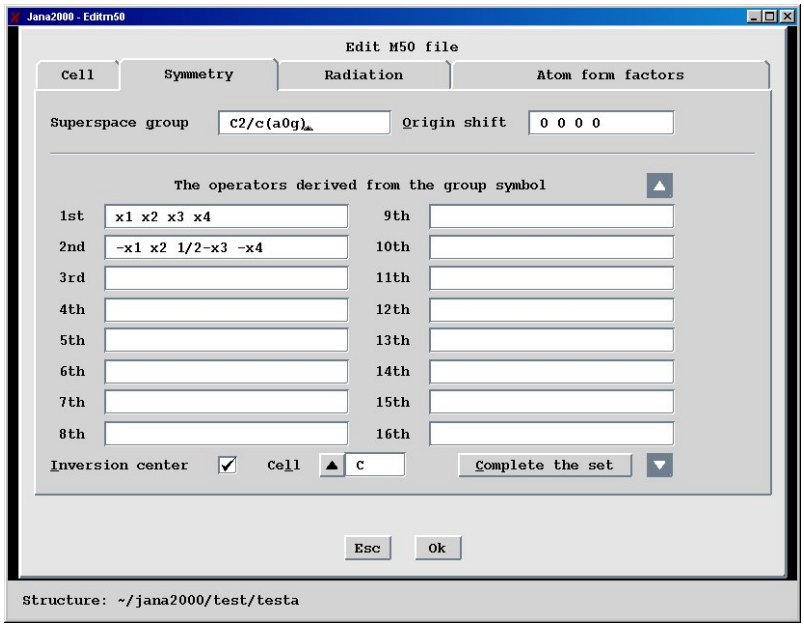


Fig. 3: *Example of a high-level graphic object in Jana2000*

Under Windows GLIB calls functions of Winteracter that represent Fortran-to-Windows API interface. However, Winteracter does not cover all necessary API functions so some of them are called through a special interface written in C. As the second way is more stable it will be preferred in the future. The Windows API functions are automatically available in any Windows operating system and on the level used in Jana2000 their behaviour is independent on the type of Windows starting from Windows95.

Under UNIX the calling sequence is:

GLIB -> FORTRAN part of FG -> C part of FG -> X11,

where FG is still part of Jana2000. The C part of FG calls directly functions of X11 library that is free software automatically available in any system with X-windows. Its portability across UNIX platforms is guaranteed. The FORTRAN part of FG has been developed in order to unify FORTRAN->C communication. Normally each compiler offers special directives for this purpose that could be very inconvenient because UNIX version of Jana2000 is compiled by users. However, if the arguments of C functions called from FORTRAN are 4 byte integers no special directives are necessary and the code is completely platform independent. Therefore, the main reason for FORTRAN part of FG is to convert any argument to integers. Another source of incompatibility inheres in names of C functions as some FORTRAN compiler expect underscores at the end of function names and the others not. Also case of letters is not guaranteed. This problem is solved in the Makefile that defines this behaviour for various systems and compilers and manages that a proper function names are activated in the FG header files.

Availability

Jana2000 is free software that can be used without any limitation on the assumption that its authors are properly cited. The source code can be downloaded with the UNIX installation files from the above mentioned WWW page. However, the only supported usage of the system is for crystallographic computing. Usage of the graphics interface and other internal libraries is not documented.

References

[1] Petricek,V. and Dusek,M.(2000). The crystallographic computing system JANA2000. Institute of Physics, Praha, Czech Republic
[2] Lahey Fortran 95, version 5.6 (http://www.lahey.com/)
[3] Winteracter, version 2.0 (http://www.winteracter.com/)

Using the Salford Software Clearwin™ library for creating GUI's with FORTRAN

Louis J. Farrugia
Department of Chemistry, Joseph Black Building, University Of Glasgow, Glasgow G12 8QQ, Telephone +44 (0)141 330 5137; FAX +44 (0)141 330 4888;
E-mail: louis@chem.gla.ac.uk – WWW: http://www.chem.gla.ac.uk/~louis/software/

Although the computer language FORTRAN originally arose in the commercial environment of IBM, the advantages of a freely available and *portable* high-level language were quickly realised by academic scientists. The introduction of standardised FORTRAN dialects, especially FORTRAN 77, greatly enhanced the acceptability of the language. Indeed, although FORTRAN is one of the earliest high-level languages (with all the disadvantages that entails), it is still in wide use today in the scientific community. This is despite major developments in computing languages, such as *object oriented programming*. Undoubtedly, one reason for the survival of the "dinosaur" language FORTRAN is that scientific programmers are skilled amateurs in that field, with a disinclination to learn new languages. However, it is also true that FORTRAN is efficient and well suited for many scientific purposes.

The advent of Graphical User Interfaces (GUI) created a problem for FORTRAN (and indeed for most computer languages) in that there is no in-built support for this feature. The implementation of GUI's is often platform specific, though a number of excellent cross-platform libraries have been created, examples of which include Tcl/Tk (Ousterhout), Python (http://www.python.org/), GTK+ (http://www.gtk.org/) and Java (http://java.sun.com/). None of the cross-platform GUI building libraries have gained universal acceptance, possibly because most are *interpreted* rather than *compiled* languages,

with the speed disadvantages that this entails (though this is becoming much less of an issue as processor speeds increase).

For the FORTRAN programmer, without much experience in C or Windows™ programming (like this author), the Clearwin™ library created by Salford Software (http://www.salfordsoftware.co.uk/) offers a number of advantages. It consists of a set of "format" functions, which are themselves a binding to the native Windows™ API (application program interface). It provides a rapid way to adapt existing FORTRAN code to run under Windows™, often with little or no change to the code. Of course it also has the serious disadvantage that the code is completely compiler specific. In this short article, I will discuss two examples of using this library to enable easy porting to Windows™ and to create GUI's for crystallographic FORTRAN programs.

The program Platon (Spek - http://www.cryst.chem.uu.nl/platon/) is designed for Unix systems and is a "multi-purpose crystallographic tool". It includes a wide variety of functionality, and is used extensively in the crystallographic community, for structure visualisation, geometrical calculations, structure validation to name but a few uses. It is distributed as source code by the author and can be easily compiled for a variety of Unix systems running *X-Windows*.

The program is written almost entirely in standard FORTRAN-77, but there are two problems regarding porting it to Windows™. Firstly, it uses a graphical interface, albeit a crude one, which utilises a number of *X-Windows* calls. These implement functions such as creating/destroying windows with graphical capability, responding to mouse events and simple drawing primitives. Fortunately, the link to these *X-Windows* calls is solely through the routine *xwin* in the source file *xdrvr.c*. This routine is written in C to facilitate linking to the *X-Windows* libraries and passes or receives information from the main FORTRAN code through its arguments only. In the Windows™ port, this routine has been replaced by a new FORTRAN routine, which is a fairly literal translation of the C code. For instance the code section ....

```
if ( *ind == -1)              /* check for X11        */
  {
    theDisplay = XOpenDisplay(display_name);
    if(theDisplay != NULL) *z = 1; else *z =  0;
    if (*z == 1) XCloseDisplay(theDisplay);
  }
  else if ( *ind == 0 )          /* close the window   */
  {
    if (wopen == 1) XFlush(theDisplay);
    XFreeGC(theDisplay, theGC);
    XCloseDisplay(theDisplay);
    wopen = 0;
    return(0);
  }
```

is replaced by ....

```
      if (ind .eq. -1) then
         theDisplay=XopenDisplay()
         if(theDisplay .ne. 0) then
           z=1
         else
           z=0
         endif
      else if (ind .eq. 0) then
         if(wopen .eq. 1) call XFlush()
         call XCloseDisplay()
         wopen = 0
         return
```

The names of the routines and variables in the FORTRAN replacement are kept as closely similar as possible. While this is not strictly necessary, it makes updating this routine much easier, if and when the source code in *xdrvr.c* changes (which it does only rarely). The calls to *X-Windows* functions such as

XCloseDisplay are replaced with new functions with the same name, which in turn make analogous calls to the Clearwin™ library. For example ....

```
subroutine XCloseDisplay()
include <windows.ins>
common/platon_control/GraphicsControl
common/platon_l/OK_to_close
logical  OK_to_close
integer GraphicsControl

OK_to_close = .true.
GraphicsControl=0
call window_update@(GraphicsControl)
end
```

The variable *GraphicsControl* is a handle to a previously created graphics-capable window, and setting its value to zero and then updating it using the Clearwin™ call to *window_update@* causes this window to be destroyed.

The second problem with porting Platon to Windows™ is much more difficult to surmount. This concerns the "System S" facility, which relies on a number of Unix specific features, such as soft links. While the concept of a link also exists in Windows™, these links do not behave in the same way as Unix links when using a command line interface. As a result, the "System S" facility has not yet been included in the Windows™ port of Platon. A preliminary Windows™ port of Platon using the CYGWIN emulation of GNU Unix (http://www.cygwin.com/) has only had partial success in implementing the "System S" facility. The use of this method is unlikely to be favoured by the general crystallographic community, since it also involves installing a CYGWIN environment and running *X-Windows* servers. Since Platon is updated very frequently, a script has been written to automate the updates. This script downloads the new source code archive, extracts the FORTRAN file, removes all the "System S" code and makes a few minor editorial changes. The resultant FORTRAN source is then compiled and linked with the routines described above, and a new Windows™ port copied to the web-site. Unless major changes have been made to the Platon source, this is undertaken completely automatically.

The program WinGX (Farrugia - http://www.chem.gla.ac.uk/~louis/software/wingx/) is a collection of (mainly) public domain crystallographic programs, with a common interface to facilitate transfer of data. Almost all are written in FORTRAN. There are numerous GUI's in this program, all of which have been created with the Clearwin™ libraries, which provide a binding to the Windows™ API libraries. The entire Clearwin™ functionality is included in one run-time library SALFLIBC.DLL. In WinGX, all the Clearwin™ calls creating GUI's have been encapsulated in another library (WGXLIB04.DLL). This library has calls which in general either create widgets or describe the positioning of widgets in the resultant window. As an example, the full code used to create the following GUI for the LEPAGE option in WinGX will be explained.

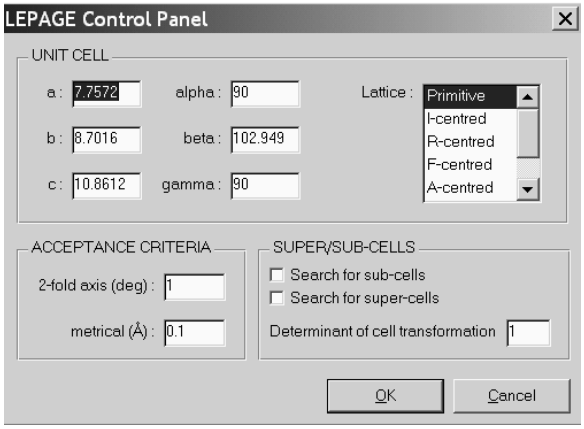


Fig. 1: *Image of graphical interface for LEPAGE as implemented within WinGX*

This GUI is created and destroyed inside the integer function *DisplayLepageBox* which passes back to the calling routine the values of the acceptance criteria and the required determinant for cell transformation, whether to search for sub-cells or supercells and the chosen lattice type. Default values for the cell constants and lattice type are passed when the function is called. The full code for this routine is as follows...

```
integer function DisplayLepageBox(latt,par,icnt)
integer      icnt(2)
character    latt
character*20 lattices(7),clabel(8)
real         par(9)
real*8       dpar(8)

lattices(1) = 'Primitive'
lattices(2) = 'I-centred'
lattices(3) = 'R-centred'
lattices(4) = 'F-centred'
lattices(5) = 'A-centred'
lattices(6) = 'B-centred'
lattices(7) = 'C-centred'
clabel(1)  = 'a :'
clabel(2)  = 'b :'
clabel(3)  = 'c :'
clabel(4)  = 'alpha :'
clabel(5)  = 'beta :'
clabel(6)  = 'gamma :'
clabel(7)  = '2-fold axis (deg) :'
clabel(8)  = 'metrical (Å) :'
par(7) = 1.0
par(8) = 1.0
par(9) = 0.1
do i=1,6
  dpar(i) = par(i)
enddo
dpar(7) = par(8)
dpar(8) = par(9)
idet = nint(par(7))
ilatt = 1

do i=1,7
   if(latt .eq. lattices(i)(1:1)) ilatt = i
enddo

call wgxOpenDialogBoxW('LEPAGE Control Panel')
call wgxOpenBoxW('named_l','UNIT CELL')
call wgxOpenBoxW('invisible',' ')
call wgxWriteStringW(clabel(1),3,1)
call wgxEditRealW(dpar(1),7)
call wgxWriteStringW(clabel(4),6,1)
call wgxEditRealW(dpar(4),7)
call wgxFormFeedW()
call wgxWriteStringW(clabel(2),3,1)
call wgxEditRealW(dpar(2),7)
call wgxWriteStringW(clabel(5),6,1)
call wgxEditRealW(dpar(5),7)
call wgxFormFeedW()
call wgxWriteStringW(clabel(3),3,1)
call wgxEditRealW(dpar(3),7)
call wgxWriteStringW(clabel(6),6,1)
call wgxEditRealW(dpar(6),7)
call wgxCloseBoxW()
call wgxSpaceW()
call wgxSpaceW()
call wgxOpenBoxW('invisible',' ')
call wgxWriteStringW('Lattice :',7,1)
call wgxListBoxW(8,5,lattices,7,ilatt)
```

```
      call wgxCloseBoxW()
      call wgxCloseBoxW()
      call wgxFormFeedW()
      call wgxNewLineW()
      call wgxOpenBoxW('named_l','ACCEPTANCE CRITERIA')
      call wgxWriteStringW(clabel(7),11,1)
      call wgxEditRealExW(dpar(7),6,0.0d0,0.0d0,1.0d0)
      call wgxFormFeedW()
      call wgxWriteStringW(clabel(8),11,1)
      call wgxEditRealExW(dpar(8),6,0.0d0,0.0d0,1.0d0)
      call wgxCloseBoxW()
      call wgxOpenBoxW('named_l','SUPER/SUB-CELLS')
      call wgxCheckBoxW('Search for sub-cells',icnt(1))
      call wgxFormFeedW()
      call wgxCheckBoxW('Search for super-cells',icnt(2))
      call wgxFormFeedW()
      call wgxTextW('Determinant of cell transformation')
      call wgxEditIntegerExW(idet,4,0,1,4)
      call wgxGangControlW(2,icnt)
      call wgxCloseBoxW()
      call wgxCloseDialogBoxW(1,i,0)
      if(i .eq. 0) i = 2
      DisplayLepageBox = i
      if(i .ne. 1) goto 1
      do i=1,6
        par(i) = dpar(i)
      enddo
      par(7) = float(idet)
      par(8) = dpar(7)
      par(9) = dpar(8)
      latt = lattices(ilatt)(1:1)
1     end
```

The first part of this routine initialises some character variables for the display and the remaining unitialised parameters. The first call which creates the GUI is to *wgxOpenDialogBoxW(),* which sets the style of the dialog box and gives the caption shown in the menu bar. The code for this routine is the one which contains the calls Clearwin™ function *winio@* and is compiled in WGXLIB04.DLL.

```
      subroutine wgxOpenDialogBoxW(caption)
      implicit none
      character*(*) caption
      integer       winio@,i,l1,leng
      logical       wgxFontPresent
      l1 = min(80,leng(caption))
      i=winio@('%sy[3d_thin,thin_border]&')
      if(wgxFontPresent('MS Sans Serif')) then
         i=winio@('%fn[MS Sans Serif]%ts&',0.85d0)
      else
         i = winio@('%fd&')
      endif
      i=winio@('%bg[btnface]%tc[btntext]&')
      if(l1 .gt. 0) i=winio@('%ca@&',caption(1:l1))
      end
```

The call *wgxOpenDialogBoxW(),* is always paired with a call to *wgxCloseDialogBoxW()*, which is the last call describing the GUI. With the arguments given in this example, this provides the standard two OK/Cancel button widgets which are used in most of the WinGX dialog boxes.

In most cases, the names of the functions called in between are self-explanatory. The paired calls to *wgxOpenBoxW()* and *wgxCloseBoxW()* create boxes, which in the above example are either invisible (and used for grouping together widget controls) or are visible as labelled, scored boxes. The contents of the box are defined by the intermediate calls - the first two calls in the first box draw a text string *clabel(1)* and then provide an edit box where the value of the real parameter *dpar(1)* (cell constant a) is displayed, and may be modified. The call to *wgxFormFeedW()* moves the insertion point for the next

widget below all previous widgets in that box. A list box widget showing all the acceptable values for the lattice type is produced by *wgxListBoxW()* with arguments defining the text strings displayed and the finally chosen lattice type - an integer variable *ilatt*.

The two check box widgets created by separate calls to *wgxCheckBoxW()* indicate whether the user wishes to search for subcells or supercells. In this example, these two widgets are *ganged,* using a call to *wgxGangControW()*, so that only one may be selected at any time, though neither need to be selected. In this way the programmer ensures that only logically sensible commands are returned by the calling function, and in the end passed to the program code of LEPAGE (Spek). The return value of *DisplayLepageBox* indicates whether the user hit the OK or Cancel button. If the latter were the case, then none of the arguments are updated, and the execution of the remaining LEPAGE code is avoided.

The use of the *wgxXXXXW()* library calls adds an extra layer between the FORTRAN code and the Windows™ API libraries, and may seem at first superfluous. However it means that all the code which calls these routine conforms to FORTRAN standards (either 77 or 95). In addition, in the future an entirely different set of GUI creating library functions could be linked to them, allowing the program to be ported to other platforms.

References

L.J. Farrugia, *J. Appl. Cryst*., 1999, ***32*****,** 837-838.
J. Ousterhout, **Tcl and the Tk Toolkit**, Addison-Wesley, ISBN 0-201-63337-X
A. L. Spek, *Acta Crystallogr., Sect A*, 1990, ***46***, C34.

State of the Toolbox: an overview of the Computational Crystallography Toolbox (CCTBX)

Ralf W. Grosse-Kunstleve and Paul D. Adams
Lawrence Berkeley National Laboratory, One Cyclotron Road, BLDG 4R0230, Berkeley, CA 94720-8235, USA
E-mail: RWGrosse-Kunstleve@lbl.gov - WWW: http://cci.lbl.gov/

1. Introduction

Back in November 2001 we released version 1.0 of the Computational Crystallography Toolbox (cctbx). In this article we give an overview of the new developments that will be available in the 2.0 release. While the code base is deemed stable and ready for release, the documentation still needs updating. We hope to be able to do this in the near future.

The cctbx is being developed as the open source component of the PHENIX system. The goal of the PHENIX project is to advance automation of macromolecular structure determination. PHENIX depends on the cctbx, but not vice versa. This hierarchical approach enforces a clean design of the cctbx as a reusable library.

To maximize reusability and, maybe even more importantly, to give individual developers a notion of privacy, the new cctbx is organized as a set of smaller modules. This is very much like a village (the cctbx project) with individual houses (modules) for each family (groups of developers, of any size including one).

The cctbx code base is available without restrictions and free of charge to all interested developers, both academic and commercial. The entire community is invited to actively participate in the development of the code base. A sophisticated technical infrastructure that enables community based software development is provided by SourceForge. This service is also free of charge and open to the entire world.

2. High level organization

The SourceForge cctbx project currently contains these modules:

libtbx
: The build system common to all other modules. This is a very thin wrapper around the SCons software construction tool. The libtbx also contains platform-independent instructions for building the Boost.Python library.

scitbx
: Contains C++ libraries for general scientific computing (i.e. libraries that are not specific to crystallographic applications): a family of high-level C++ array types, a fast Fourier transform library, and a C++ port of the popular LBFGS conjugate gradient minimizer, all including Python bindings. These libraries are separated from the crystallographic code base to make them easily accessible for non-crystallographic application developers.

cctbx
: Python and C++ libraries for general crystallographic applications, useful for both small-molecule and macro-molecular crystallography. The libraries in the cctbx module cover everything from algorithms for the handling of unit cells to high-level building blocks for refinement algorithms. Note the distinction between the cctbx *project* and the cctbx *module*. In retrospect we should have chosen a different name for the project, but the current naming reflects how the modules have evolved and it would be too disruptive to start a grand renaming.

iotbx
: The youngest member in the family: evolving libraries for reading and writing established file formats. At the moment the iotbx contains C++ and Python interfaces for reading CCP4 MTZ files and ADSC X-ray detector images.

3. The beach in the box

If you go to the beach you will find massive amounts of a material known to crystallographers as quartz, which in this case is just a fancy word for sand. As an example here is the cctbx way of playing in the sandbox:

```
from cctbx import xray
from cctbx import crystal
from cctbx.array_family import flex

quartz_structure = xray.structure(
  special_position_settings=crystal.special_position_settings(
    crystal_symmetry=crystal.symmetry(
      unit_cell=(5.01,5.01,5.47,90,90,120),
      space_group_symbol="P6222")),
  scatterers=flex.xray_scatterer([
    xray.scatterer(
      label="Si",
      site=(1/2.,1/2.,1/3.),
      u=0.2),
    xray.scatterer(
      label="O",
      site=(0.197,-0.197,0.83333),
      u=0)]))

quartz_structure.show_summary().show_scatterers()
```

Running this script with Python produces the output:

```
Number of scatterers: 2
At special positions: 2
Unit cell: (5.01, 5.01, 5.47, 90, 90, 120)
Space group: P 62 2 2 (No. 180)
Label  M  Coordinates            Occ  Uiso or Ustar
Si     3  0.5000  0.5000  0.3333 1.00 0.2000
O      6  0.1970 -0.1970  0.8333 1.00 0.0000
```

Note that the script is pure Python, even though at first sight the format might appear to be specifically designed for crystallographic data. Now let's give the `quartz_structure` a rest break at the beach:

```
from scitbx.python_utils import easy_pickle
easy_pickle.dump("beach", quartz_structure)
```

This creates a file with the name `beach` containing all the information required for restoring the `quartz_structure` **object**, which is the technical term for the entire hierarchy of data referenced by the `quartz_structure` identifier in the Python script. A very important point to notice is that the `easy_pickle` module used for storing the `quartz_structure` is not specific to our object. `easy_pickle` will store and restore (almost) any user-defined Python object.

Being automatically generated, the `beach` file is not pretty to look at, but this is not important because we can easily resurrect the original object to extract any information that we might be interested in. In a potentially *different script* on a potentially *different computer* with a potentially *different operating system*:

```
from scitbx.python_utils import easy_pickle
quartz_structure = easy_pickle.load("beach")
```

Note that it is not necessary to explicitly import the relevant cctbx modules in this script. Python's pickle module does it for us automatically after inspecting the `beach` file.

In practice a "live object" in memory is much more valuable than information stored in a good-old file format because there are often many different questions one can ask about particular real-world objects. For example, we could ask: What are the site symmetries of the atoms in the `quartz_structure`? Here is how we ask that question in Python's language:

```
for scatterer in quartz_structure.scatterers():
  print "%s:" % scatterer.label, "%8.4f %8.4f %8.4f" % scatterer.site
  site_symmetry = quartz_structure.site_symmetry(scatterer.site)
  print "  point group type:", site_symmetry.point_group_type()
  print "  special position operator:",  site_symmetry.special_op()
```

Answer:

```
Si:   0.5000   0.5000   0.3333
  point group type: 222
  special position operator: 1/2,1/2,1/3
O:   0.1970  -0.1970   0.8333
  point group type: 2
  special position operator: 1/2*x-1/2*y,-1/2*x+1/2*y,5/6
```

Another question we could ask: What are the structure factors up to a resolution of d_min=2 Angstrom?

```
f_calc = quartz_structure.structure_factors(d_min=2).f_calc_array()
f_calc.show_summary().show_array()
```

Answer:

```
Miller array info: None
Type of data: complex_double, size=7
Type of sigmas: None
Number of Miller indices: 7
Anomalous flag: None
Unit cell: (5.01, 5.01, 5.47, 90, 90, 120)
Space group: P 62 2 2 (No. 180)
(1, 0, 0) (-11.3483432953-3.90019504038e-16j)
(1, 0, 1) (-14.9620947104-25.915108226j)
(1, 0, 2) (1.46915343413-2.54464839202j)
(1, 1, 0) (-12.8387095938+0j)
(1, 1, 1) (5.39203951708-9.3392864j)
(2, 0, 0) (-1.80942693741-2.84059649279e-16j)
(2, 0, 1) (4.95031293935+8.57419352432j)
```

Now we could turn our attention to the new `f_calc` object and start asking different questions. For example: What are the d-spacings?

```
f_calc.d_spacings().show_array()
```

Answer:

```
(1, 0, 0) 4.33878727296
(1, 0, 1) 3.39927502294
(1, 0, 2) 2.31368408207
(1, 1, 0) 2.505
(1, 1, 1) 2.27753582331
(2, 0, 0) 2.16939363648
(2, 0, 1) 2.01658808355
```

Sometimes questions alone are not enough. We actually want to do something. For example select only low-resolution intensities:

```
low_resolution_only                                                    =
f_calc.apply_selection(f_calc.d_spacings().data() > 2.5)
low_resolution_only.show_array()
```

Answer:

```
(1, 0, 0) (-11.3483432953-3.90019504038e-16j)
(1, 0, 1) (-14.9620947104-25.915108226j)
(1, 1, 0) (-12.8387095938+0j)
```

Of course, the cctbx does not have a canned answer to every question, even if it is a reasonable question. Fortunately, by virtue of being a Python based system, the cctbx does lend itself to being extended and embedded in order to form answers to questions that one might come across. The cctbx has now reached a degree of completeness where this can quite often be done without venturing into the deeper and darker layers, the C++ core that we haven't so far presented.

4. At the very bottom

Python is a great language for just about everything. It is just unfortunate that we do not currently have machines smart enough to turn any Python script into efficient machine language (but visit the PSYCO web site to witness mankind stretching out its feelers in that direction). Certainly, future generations will pity us for having to resort to counting bits and bytes in order to get our work done (imagine yourself with a set of Beevers-Lipson strips getting ready for a structure factor calculation).

Some core components of the cctbx started out as 'C' libraries (SgInfo, AtomInfo). Moving from 'C' to C++ including the Standard Template Library (STL) was a major step away from the bits-and-bytes counting era. For example, switching to C++ exception handling for dealing with errors reduced the source code size significantly and resulted in much improved readability. Equally important, using `std::vector` for managing dynamically allocated memory was a huge improvement over using 'C' style raw memory allocation functions (`malloc()` and `free()`). However, the idea of using `std::vector` throughout the cctbx wasn't very satisfactory: for small arrays such as 3x3 rotation matrices the dynamic memory allocation overhead can become a rate-limiting factor, and for large arrays the copy-semantics enforce a coding style that is difficult to follow. For example, consider a member function of a space group class that computes an array of multiplicities given an array of Miller indices. The scalar version of this function would certainly look like this:

```
int multiplicity(miller::index<> const& h);
```

Here is the direct translation to a vector version:

```
std::vector<int> multiplicity(std::vector<miller::index<> > const& h);
```

However, std::vector has deep-copy semantics (the same is true for std::valarray). This results in the following:

```
std::vector<int> multiplicity(std::vector<miller::index<> > const& h)
{
  std::vector<int> result; // Constructs the array.
  result.reserve(h.size()); // Optional, but improves performance.
  for(std::size_t i=0; i<h.size();i++) { // Loops over all Miller indices.
    result.push_back(multiplicity(h[i])); // Uses the scalar overload
  }                                       // to do the actual work.
  return result; // Ouch!
}
```

"Ouch" indicates that the *entire array* is copied when the function returns! While this might still be acceptable for arrays of Miller indices which are mostly of moderate size, it becomes a real burden when dealing with large maps. But returning to the example, in order to avoid the copying overhead the function above could be coded as:

```
void multiplicity(std::vector<miller::index<> > const& h,
                  std::vector<int>& result);
```

Unfortunately this is not only harder to read, but also more difficult to use because the result has to be instantiated before the function is called. This prevents convenient chaining of the type used in the `quartz_structure` examples above.

Other major problems are the absence of a multi-dimensional array type in the STL and limited support for algebraic array operations. We considered using Blitz++, and `boost::multi_array`, but these do only partially meet our specific requirements. For small arrays we actively used `boost::array` for some time, but this was also not entirely satisfactory due to the lack of convenient constructors which again prevents chaining. So eventually we started the major effort of implementing a family of small and large array types that address all our requirements and are as uniform as possible: the scitbx array family.

5scitbx.array_family.flex

The scitbx array family forms the backbone of the cctbx project. Viewed from the C++ side the family is quite big and diverse, but viewed from the Python side things are a lot simpler, as usual. All small C++ array types are transparently mapped to standard Python tuples. This gives immediate access to the rich and familiar set of standard tools for manipulating tuples. All large array types are transparently and uniformly mapped to a group of Python types in the `scitbx.array_family.flex` module. For example:

```
from scitbx.array_family import flex
flex.double(30) # a 1-dimensional array of 30 double-precision values
flex.int(flex.grid(2,3)) # a 2-dimensional array of 2x3 integer values
```

For numeric element types the `flex` type supports algebraic operations:

```
>>> a = flex.double([1,2,3])
>>> b = flex.double([3,2,1])
```

```
>>> tuple(a+b)
(4.0, 4.0, 4.0)
>>> tuple(flex.sqrt(a+b))
(2.0, 2.0, 2.0)
```

The `flex` type also supports a large number of other functions (abs, sin, pow, etc.), slicing, and as seen in the `quartz_structure` example above, pickling.

If all this looks similar to the popular Numeric module: it is at the surface. However, there are two very important differences:

- Under the hood the `flex` types are instantiations of a C++ array type that resembles familiar STL container types as much as possible. In contrast Numeric presents itself with a raw 'C' API.
- It is straightforward to implement other `flex` types with custom user-defined element types, even outside the scitbx module. This would be extremely difficult to do with Numeric, and is virtually impossible if the user-defined types are implemented in C++.

6cctbx.array_family.flex

The `cctbx.array_family.flex` inherits all `flex` types from the `scitbx.array_family.flex` module and adds a few types specific to the cctbx module, for example:

```
from cctbx.array_family import flex
flex.miller_index(((1,2,3), (2,3,4)))
flex.hendrickson_lattman(((1,2,3,4), (2,3,4,5)))
```

Another example is `flex.xray_scatterer` used in the `quartz_structure` above. The cctbx specific C++ code for establishing these Python types is fairly minimal (about 470 lines for exposing 6 types, including full pickle support and all copyright statements). This approach can therefore be easily adopted for user-defined types in other modules.

7. A balancing act

Python's **convenience of use** is directly related to the way the Python type system works: all type information is evaluated at runtime. For example consider this trivial function:

```
def plus(a, b):
  return a + b
```

It works instantly for many different argument types:

```
>>> plus(1, 2) # integer values
3
>>> plus(1+2j, 2+3j) # complex values
(3+5j)
>>> plus(['a', 'b'], ['c', 'd']) # lists
['a', 'b', 'c', 'd']
```

It works because the meaning of `a + b` is determined at runtime based on the actual types of `a` and `b`.

The **runtime efficiency** of C++ code is directly related to the way the C++ type system works: type information is usually evaluated at compile-time (virtual functions are an exception which we will not

consider here). Fortunately C++ has a very powerful mechanism that helps us avoid explicitly coding polymorphic functions over and over again:

```
template <typename T>
T plus(T const& a, T const& b)
{
  return a + b;
}
```

This template function is automatically *instantiated* for a given type `T` as used:

```
int a = 1;
int b = 2;
int c = plus(a, b); // implicitly instantiates plus with T==int
```

Given a system that is based on both Python and C++ we have the freedom of choosing the quick-and-easy runtime polymorphism offered by Python, or the more efficient compile-time polymorphism offered by C++.

An important consideration in deciding which solution is the most appropriate for a given problem is that a polymorphic Python function requires very little memory at runtime. In contrast, each new instantiation of a template function eventually results in a complete copy of the corresponding machine language instructions tailored for the specific types involved. This point may seem subtle at first, but being overly generous with the use of C++ compile-time polymorphism can lead to very large executable sizes and excessive compile times.

A comparison of the `plus` Python function and its C++ counterpart shows that the notational overhead of the C++ syntax can easily double the size of the source code. Therefore a programmer, given the choice, will naturally lean towards the Python solution until the runtime penalty due to the dynamic typing is prohibitive for a given application. However, when putting a dynamically typed system and a statically typed system together there are situations where it is important to carefully consider the best balance.

8. Hybrid systems

Considerations of the type discussed in the previous section directly lead to the following situation:

```
>>> a = flex.int((1,2,3))
>>> b = flex.double((2,3,4))
>>> a * b
TypeError: unsupported operand type(s) for *:
'scitbx_boost.array_family.flex_scitbx_ext.int' and
'scitbx_boost.array_family.flex_scitbx_ext.double'
```

In passing we note that there is a simple solution which will produce the desired result:

```
a.as_double() * b
```

However, for the purpose of this discussion let's pretend that this solution does not exist. Of course the first question is: what is the reason for the apparently stupid limitation?

As mentioned before, the Python `flex` types are implemented as instantiations of C++ class templates. This ensures that all array operations are very fast. However, from the discussion in the previous section it follows that exposing the full class with its many member functions to Python **for each element type** (`int`, `double`, `miller::index<>`, etc.) creates very sizable object files. If only *homogeneous*

operators `int * int`, `double * double`, etc.) are used the combined size of the object files scales linearly with the number of element types involved. However, if the library is expanded to support *heterogeneous* operators (`int * double`, `double * int`, etc.) the combined object files grow proportional to the square of the number of array element types involved! With current technology this is simply prohibitive.

Limitations of the kind discussed here will apply to any hybrid dynamically/statically typed system. In the broader picture the limitation shown above is just one typical example. If we want to enjoy the many benefits of using Python *and* have a system that produces results with a reasonable runtime efficiency we have to adopt the approach of sparsely sampling the space of possible C++ template instantiations. For this idea to work in practice we need a powerful and easy to use language-integration tool as discussed in the next section.

9. Building bridges

The cctbx project has evolved together with the Boost.Python library. All Python/C++ bindings in the cctbx project are implemented using this library. Here is a simplified example of how it works in practice:

This is the C++ class that we want to use from Python:

```
//! Parallelepiped that contains an asymmetric unit.
class brick
{
  public:
    //! Constructor.
    /*! Determines the parallelepiped given a space group type.
     */
    explicit
    brick(space_group_type const& sg_type);

    //! Formats the information about the brick as a string.
    /*! Example: 0<=x<=1/8; -1/8<=y<=0; 1/8<z<7/8
     */
    std::string as_string() const;

    //! Tests if a given point is inside the brick.
    bool is_inside(tr_vec const& p) const;
};
```

These are the corresponding Boost.Python bindings:

```
class_<brick>("brick", no_init)
  .def(init<space_group_type const&>())
  .def("__str__", &brick::as_string)
  .def("is_inside", &brick::is_inside)
;
```

And here is how the class is used in Python:

```
>>> from cctbx import sgtbx
>>> brick = sgtbx.brick(sgtbx.space_group_type("I a -3 d"))
>>> print brick
0<=x<=1/8; -1/8<=y<=0; 1/8<z<7/8
>>> brick.is_inside(sgtbx.tr_vec((0,0,0)))
0
```

Typically it only takes a few minutes to implement the Python bindings for a new class or function. Since it usually takes orders of magnitudes longer to implement C++ classes and functions the extra time spent on the Python bindings is in general negligible.

10. Thinking hybrid

Boost.Python's ease of use enables us to *think hybrid* when developing new algorithms. We can conveniently start with a Python implementation. The rich set of precompiled tools included in the scitbx and the cctbx gives us a head start because many operations are already carried out at C++ speed even though we are only using Python to assemble the new functionality. If necessary, the working procedure can be used to discover the rate-limiting sub-algorithms. To maximize performance these can be reimplemented in C++, together with the Python bindings needed to tie them back into the existing higher-level procedure.

To give an example, this approach was used in the development of the *Euclidean model matching algorithm* found in the cctbx (`cctbx/euclidean_model_matching.py`). This algorithm is used to compare heavy-atom substructures or anomalously scattering substructures from isomorphous replacement or anomalous diffraction experiments. The algorithm was first implemented in about 300 lines of pure Python. We wrote another 200 lines of comprehensive regression tests for thoroughly debugging the implementation. For a while the pure Python code actually worked fast enough for us, until we started to work with a very large substructure with 66 anomalous scatterers. Some simple optimizations of the Python implementation resulted only in a modest speedup, but after replacing about 30 lines of Python with a C++ implementation the algorithm runs about 50 times faster.

Of course there is still more to gain by reimplementing the entire algorithm in C++. However, one has to keep in mind that developing C++ code is typically much more time-consuming than developing in Python. For example, the 30 lines of Python mentioned in the previous paragraph turned into more than 100 lines of C++, not counting the additional 13 lines for the `Boost.Python` bindings. It is also important to keep in mind that developing maintainable C++ code requires much more hard-earned working experience than developing useful Python code. C++ has many pitfalls that one must learn to avoid. In contrast the Python language is structured in a way that steers even the novice programmer onto safe routes. In fact, Python was originally conceived as a language for teaching programming. Amazingly this heritage is still preserved even though Python has grown to be a very richly featured language.

11. A piece of cake

Conceptually it is a trivial task to compile and link portable source code. However, in real life this is one of the most time-consuming nuisances, in particular if multiple, diverse platforms have to be supported. In the version 1.0 release of the cctbx we made an attempt to address this with the home-made *fast track* build system. Of course home-made is often good enough, but a professional solution is almost always better, especially if it comes with no strings attached.
Fortunately in the meantime such has system has become available: SCons, short for Software Construction tool. This is a perfect fit for the cctbx because the SCons developers have apparently adopted a similar "professional is better than home-made" philosophy: SCons is implemented in pure Python, and SCons configuration files (the equivalent of Makefiles) are pure Python scripts. SCons has

many advantages compared to a traditional make-based build system. To quote some points from the SCons documentation:

- Global view of all dependencies -- no more multiple build passes or reordering targets to build everything.
- Reliable detection of build changes using MD5 signatures -- no more "clock skew detected, build may be incomplete".
- Built-in support for C, C++, Fortran, Yacc and Lex.
- Improved support for parallel builds - like make -j but keeps N jobs running simultaneously regardless of directory hierarchy.
- Building from central repositories of source code and/or pre-built targets.
- Designed from the ground up for cross-platform builds, and known to work on Linux, POSIX, Windows NT, Mac OS X, Tru64 Unix, and OS/2.

When we moved from our home-grown build system to SCons we found all these points to be perfectly true. It only took very little effort to write a small configure script for setting up a master SConstruct file based on the user's choice of which cctbx modules to use and which to ignore. New modules can easily be tied into this system simply by providing a SConstruct file in the module's top-level directory. The author of the new module has complete control over the build process. The existing settings can simply be reused, customized, or totally replaced, all within one uniform and 100% platform-independent framework, the Python language.

12. Bundling up

As mentioned in the introduction we are currently in the process of completing and extending the documentation of the cctbx modules. When we are finished we will create a self-contained bundle including the four cctbx modules and all its dependencies (Python, Boost, SCons). For Windows and potentially for some Unix platforms we will provide binary distributions that allow users to directly start working with the Python interfaces without worrying about the compilation. However, since SCons will be included, anybody with a working C++ compiler and a burning desire to work in that language has immediate access to all the tools that we are using in our development.

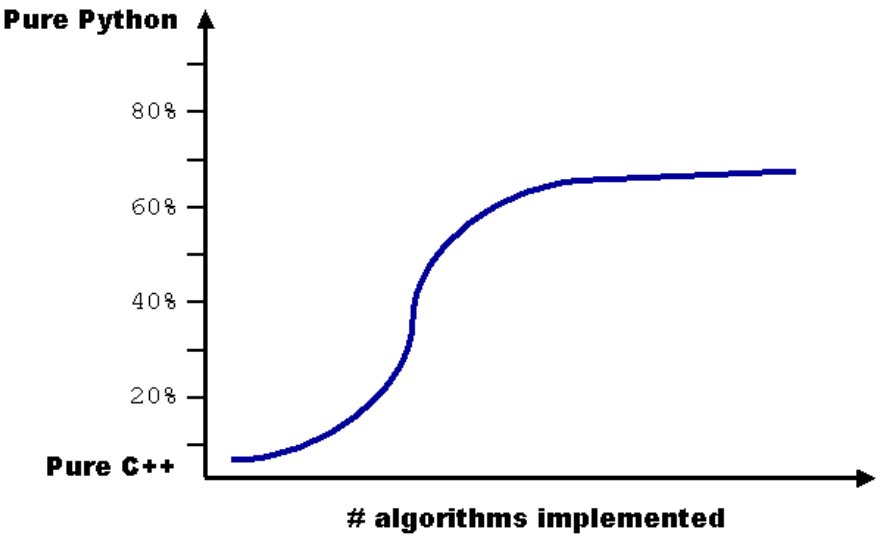
We'd like to emphasize that all C++ libraries in the cctbx project can still be used without Python, as was the case in cctbx version 1.0. The only restriction is that our build system obviously depends on Python. But, Python being more portable than C++, this should never really be a restriction in practice. And of course there is nothing which prevents our C++ libraries from being compiled with other tools.

13. Conclusion

The cctbx 2.0 covers a wide spectrum of fundamental algorithms and data structures needed for crystallographic software development. However, there is still an important blank spots on the map, namely fast Fourier transform based computations of derivatives of the structure factor equation. This is the last essential block of functionality required for building an efficient refinement program for macro-molecular structures. After the release we will focus our attention on removing this blank spot from the map.

Of course there is a plethora of other more specific algorithms that are needed to further the automation of macro-molecular structure determination, and some of these will still be general enough to warrant inclusion in a library to enable their use in different contexts. Therefore we expect the cctbx project to

grow continually for some time to come. The split of the cctbx project into several submodules ensures that this growth will be manageable, and that developers that only need a subset of the functionality do not have carry around large unused packages.
Looking back, the cctbx started out mainly as a library of C++ classes with 'C' heritage, and for a while the growth was mainly concentrated on the C++ parts. However, a very important difference between the 1.0 release and the upcoming 2.0 release is that the Python parts now constitute a much more significant fraction of the total sources. We expect this trend to continue, as illustrated qualitatively in this figure:



This figure shows the ratio of newly added C++ and Python code over time as new applications are implemented. We expect this ratio to level out near 70% Python. From an inside viewpoint the increasing ability to solve new problems mostly with the easy-to-use Python language rather than a necessarily more arcane statically typed language is the return on a major investment, namely our involvement in the development of Boost.Python. From an outside viewpoint we hope that the ability to solve some problems entirely using only Python will enable a larger group of scientist to participate in the rapid development of new algorithms. It is also important to notice that Python is an ideal language for integrating diverse existing tools, no matter which language they are written in. If portability is not an issue this can be a great solution to some problems. We are convinced that the cctbx can be very useful as an intelligent mediator between otherwise incompatible crystallographic applications.

14. Acknowledgments

We would like to thank David Abrahams for creating the amazing Boost.Python library and for patiently supporting the entire open source community. The iotbx module is developed and maintained by Nick Sauter who also made many suggestions regarding the design of the other modules. We would like to thank Airlie McCoy for allowing us to adapt some parts of the Phaser package (FFT structure factor calculation). Kevin Cowtan has contributed algorithms for the handling of reciprocal space asymmetric units. We are also grateful for his development of the Clipper library from which we have adapted some source code fragments. Our work was funded in part by the US Department of Energy under Contract No. DE-AC03-76SF00098. We gratefully acknowledge the financial support of NIH/NIGMS.

Link to references: http://cci.lbl.gov/publications/

Using advanced methods of computer graphics for crystallographic needs.

Michal Hušák,
Department of Solid State Chemistry, Prague Institute of Chemical Technology, Technique 5, 166 28 Prague 6, Czech Republic;
E-mail: husakm@vscht.cz - http://www.ccp14.ac.uk/ccp/web-mirrors/marchingcube-fourierviewer/~husakm/Public/MarchingCubeELD/MarchingCubeELD.htm

OpenGL – the way to get speed.

Crystallographers often need to visualize really complex objects. Typical examples of such objects are molecular structures, electron density maps or even the complete arrangement of all atoms inside a part of the crystal. Without seen the picture of such models, it is often really difficult to interpret the data.

It is possible to catch much more from the image when a real time manipulation is possible. It means that we need the very fast creation of graphics. Due to development hardware for computer games, very fast graphic operations are now possible even on a standard PC.

Graphics can be generated not only by the computer CPU, but by a special processors on the graphic card. Such processors can do this job usually much more faster than CPU. To be able to use the power of these specialized processors we must use an API compatible with the graphics card drivers during code development. Two of the most important API's for communication with graphic hardware exist now: OpenGL and Microsoft DirectX. In OpenGL it is possible to do the same as under DirectX; and in addition the code written in OpenGL is often compatible with non-Microsoft operating systems (e.g., Linux, IRIX). From this point of view the only API suitable for crystallographic applications graphic acceleration is OpenGL .

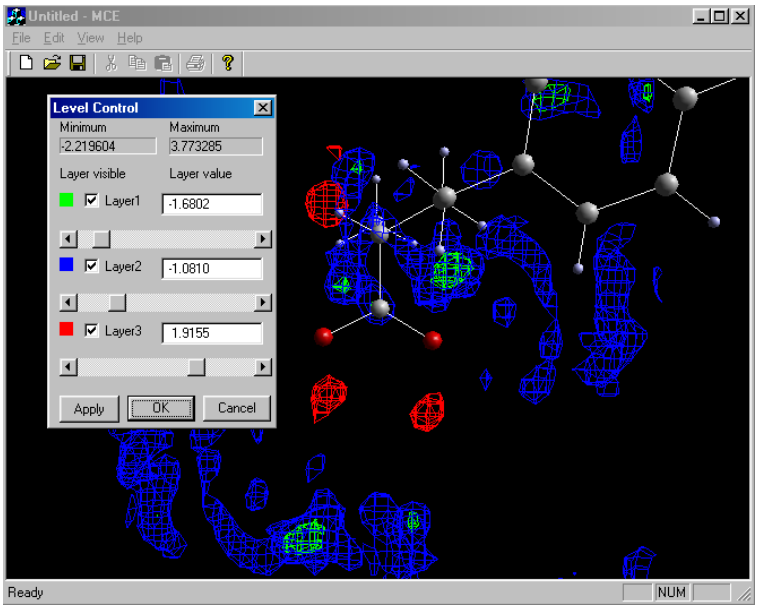
What can OpenGL do for you? It can take care of super fast drawing of all the geometric primitives – lines, spheres, planes, triangles, etc. In addition it can take care of a lot of other effect like simulating the lighting of your objects, selecting objects and object manipulation. The best support for OpenGL exists in C and C++ programming languages. There exist interfaces for communication with OpenGL API for other coding languages as well, (e.g. Delphi and for Visual Basic).

Using OpenGL programs in your code require several steps. At first you must initialize the OpenGL hardware engine and attach it to your output window. This step is OS specific and the correct command sequence differs for each operating system. The next step is to determine the view on the scene (your virtual camera position and the size of the scene). The last step is the sequence for the geometric primitives drawing. Code sequence for drawing a line from point X1,Y1,Z1 to point X2,Y2,Z2 can look e.g. like this:

```
glBegin(GL_LINES);
     glVertex3f(X1,Y1,Z1);
     glVertex3f(X2,Y2,Z2);
glEnd();
```

The best method to start with OpenGL coding is to study freely available source code tutorials. One of the best collection of such tutorials is on following webpage: http://nehe.gamedev.net/. You can find on this webpage how to initialize OpenGL under Windows, MacOS and typical UNIX systems as well as complete sample codes showing simple and very advanced OpenGL coding techniques. The official source of OpenGL information could be found on http://www.opengl.org/ and in written form in the excellent "Red Book" (Ref. 1). A very good book about using OpenGL under Windows in combination with MFC is *"OpenGL Programming for Windows 95 and Windows NT"*, by Ron Fosner" (Ref. 2).

Almost all modern molecular visualization and crystallographic programs already use OpenGL for graphic acceleration. Examples of freely available programs are VMD, MolMol or the whole crystallographic determination package CRYSTALS. From commercial programs the OpenGL is used in almost all ACCELRYS software products (e.g. DS Viewer PRO) or in the CSD database GUI - ConQuest . The author of this article had developed a MCE code (Pic 1.) using OpenGL for very fast and interactive electron density map visualization.



Pic 1: *Example of an output from a MCE program using fast OpenGL hardware-accelerated output for interactive difference electron density map visualization. This code is a part of latest CRYSTALS release .*

Stereoscopic visualization

Even when you have very fast graphics and can perform real time manipulation, your output will look 2D and flat. Thus to catch some important futures from such output (e.g. molecular interactions, chirality, docking conditions) is almost impossible. The human brain needs 2 different views on the object from each eye to be able to synthesize a true 3D image in the mind.

The methods for delivering to each eye different image are titled stereoscopic visualization. Following possibilities exist:

- Shuterglasses. This method uses special glasses which are synchronized with the work of your PC monitor. Synchronization can be done with the help of IR emitter or by cable. The monitor usually works at image refresh rate 120 Hz or so. The glasses make visualization through left and right eye possible at a 60 Hz frequency. As the result, one eye only sees odd frames and the second only even frames. Modern graphics card with OpenGL stereoscopic support (Quadro, Oxygen, Wildcat) can do this job.
- Polarized projection. This method is useful for large audience visualization. The left and right eye images are projected by 2 projectors with polarization filters. Both filters are 90 degree rotated. It is necessary to use specialized polarization glasses to view the images. Each eye sees the correct image only because the opposite one with different polarization is filtered. Graphic cards with dual output (Quadro) can create suitable output.
- Autostereoscopic 3D monitors. The principle of this device is the projection of different images in different directions. No glasses are necessary to watch the effect. The monitors usually must track the user position and optimize the left-right image direction projection from this information (Pic. 2).

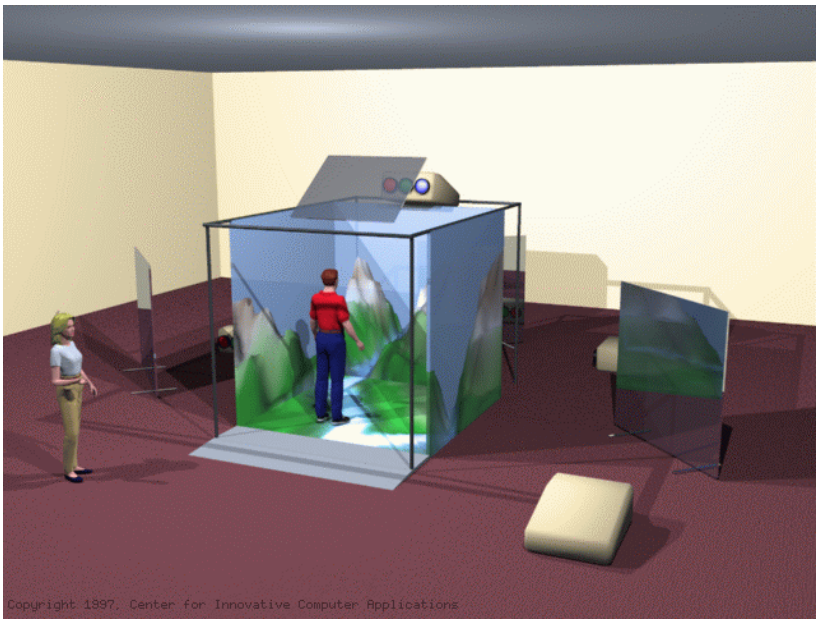
From existing software, the stereoscopic visualization is supported: VMD, MolMol, DS Viewer PRO or the protein structure solution system O.

Pic 2: *Autostereoscopic monitor. (Image downloaded from SeeReal Technologies GmbH (former Dresden3D) - http://www.dresden3d.com/)*

Advanced VR HW devices

To see your data in stereoscopic 3D is nice, but how would you like to be inside your model ? To be able to walk between atoms, touch your force field maps, listen to the force field intensities ? Even such things are possible …

One of the devices which can put you inside the data is the CAVE device. The CAVE is a cube and 5 to 8 walls of this cube could be used as stereoscopic projection screens (Pic. 3). The user is inside the cube, his position is tracked and superimposed in the virtual environment. The device is equipped with environmental sound system as well which can serve even to data "sonification". The user can be by the sound e.g. notified about the force field intensity in the current position.



Pic 3: *The scheme of the CAVE device. (Image downloaded from Dave Pape CAVE - http://www.evl.uic.edu/pape/CAVE/)*

The support for CAVE is built-in to VMD software and in the "Marching Cubes VR" software for electron map visualization developed by the author of this article (http://www.ccp14.ac.uk/tutorial/marchingcube/). Complete information about the CAVE device can be found at this link: http://evlweb.eecs.uic.edu/pape/CAVE/

Pic 4: *Using the HMD as an interface for MCE_VR electron maps visualization software. The gun serves for placing atoms in correct positions in the model.*

The main disadvantage of the CAVE device is that it is a big and expensive. The price acceptable VR alternative to CAVE is an HMD (Head Mounted Display). HMD can track the user motion just as well as CAVE. And it can produce stereoscopic image by the help of using 2 different screens for each eye. The main disadvantage of existing HMD is low field of view and low per eye graphic resolution. This could change in the near future due to new technologies used for microdisplays manufacturing. The "MCE_VR" code is an example of experimental program supporting output to VFX3D HMD (Pic. 4).
Another method for advanced object visualization is volumetric display. The principle of this display is a projection of the object on some 3Dimensional fast moving (rotating plane) or partially transparent (artificial fog) material usually by laser technology (Pic. 5).

Pic 5: *DNA molecule visualized by a volumetric display. (Image downloaded from Actuality System Inc. - http://www.actuality-systems.com/)*

References:

1. The OpenGL Programming Guide 3rd Edition The Official Guide to Learning OpenGL Version 1.2, Mason Woo, OpenGL Architecture Review Board, Jackie Neider, Tom Davis, Dave Shreiner. Addison-Wesley, 1999
2. OpenGL Programming for Windows 95 and Windows NT, by Ron Fosner. Addison-Wesley , 1996

The author would like to thank the Grant Agency of the Czech Republic (grant No. GACR: 203/01/0700)

Object Oriented programming and fast computation techniques in Maud, a program for powder diffraction analysis written in Java™

Luca Lutterotti[1] and Mauro Bortolotti[2]
(1) (Present address) Earth and Planetary Science Department, University of California at Berkeley, 94720 - Berkeley, CA, USA, Dipartimento Ingegneria dei Materiali, Università di Trento, via Mesiano, 77, 38050 - Trento, Italy, Luca.Lutterotti@ing.unitn.it – WWW: http://www.ing.unitn.it/~luttero/ ; and Dipartimento Ingegneria dei Materiali, Università di Trento, via Mesiano, 77, 38050 - Trento, Italy, mbortolotti@inwind.it

Introduction

The Maud project [1] started in 1996 as a mere exercise using the one year old Java language [2]. The idea was of writing a full Rietveld program easy to use and develop. But one idea was that the easy to use and develop concept should not lead to an underpowered program. Java at that time was just in its infancy, not ready for a full scientific program, but very promising for the possibility to run it on different platforms; powerful and clean, it was particularly focused on writing interfaces and web applet. Was it ready for a scientific application in need of speed and highly demanding mathematical performances? At the beginning not, still in 1996, with version 1.0 of the jdk (Java Development Kit), there were few bugs preventing a full computation of a powder pattern when the memory requirement was more than a 1 or 2 Mb. But one thing was already clear, writing a moderately complex interface was more easy than with other languages and coming from C/C++ finally you have a language were you don't need to deal with pointers. This means mainly a little bit less speed for the user but much less painful debugging for the developer. The first public version of Maud was released on the web in the early fall of 1998 with the jdk 1.1 that was the first really usable Java version for applications.

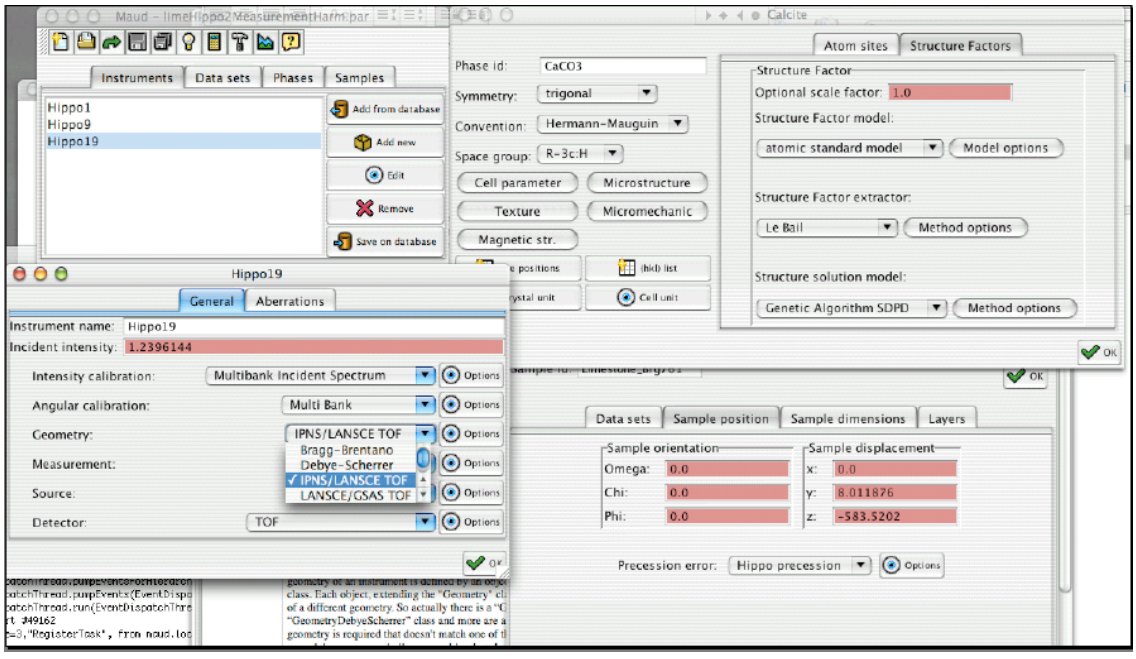


Fig 1: *General view of the program interface where several plug-in models are visible. For example in the Instrument view (Hippo19 window), the combo box is open to see some of the different geometries already available; all the other combo boxes in the same frame correspond to other characteristics that support the plug-in structure and can be extended by providing additional models. For the phase frame (Calcite) under the Structure Factors tab panel there are others tree model-extendable features to work on structure solution.*

In the early years more time was spent on designing the program and especially the internal structure to focus on the following aspects:

- Objects and their arrangement inside the program should reflect the way a researcher is approaching diffraction, from the experiment to the analysis. So this means for example, that there should be an instrument, specified by several other objects like geometry, radiation etc. The root object was identified in an analysis containing basically 4 kinds of objects: samples, instruments, sets of data and phases. Each set of data has associated one instrument; each sample contains some of the phases and has associated some sets of data collected on it and so on.
- No compromise on future development and possible analyses. So the structure should be open as much as possible and easily extendable. This should be not too much in contrast with speed an easy development.
- Modifiable and extendable by advanced users. In order to accomplish that, the user must be able to write their own objects with algorithms (the controller) and also the interface (the view) to manage them. So instead of a pure separated controller and view model, a mixed one has been adopted in which each class can specify its own controller and view or inherits them from the superclass.

After the initial effort spent on designing the infrastructure the work has been concentrated on implementing the full program to cover as much aspects as possible in diffraction analyses with at least one or a couple of models for each methodology. The program covers now from Rietveld, to ab initio structure solution, to quantitative analysis, line broadening (crystallite size and microstrains), texture and residual stresses. Finally also reflectivity has been added for a specific project to couple it with diffraction. Only powder diffraction was considered.

In the last period of time, the general structure of the program has been fixed and the optimization step has been started. In the same time we started to write the documentation in order to provide help for users and for possible future developers who want to put their hands to customize or extend the program for their needs. This article is a starting point in this effort and we will discuss briefly the general OO (Object Oriented) structure of the program by some examples with code on how some tasks have been accomplished or can be extended, and some special techniques for fast computation in Java.

Plug-in structure and examples

To simplify and generalize the program a basic object has been developed and nearly all the other objects extend this class. This basic object provides the general structure and support for loading/saving, managing parameters, tree of objects and plug-in objects. Using that class as superclass greatly simplify the addition of different models in the program to extend and customize its features. For example, the geometry of an instrument is defined by an object extending the basic "Geometry" class. Each object, extending the "Geometry" class, can be used by Maud as a choice of a different geometry. So actually there is a “GeometryBraggBrentano” class, a “GeometryDebyeScherrer” class and more are available. Every time a new instrument geometry is required that doesn't match one of the already available in Maud, you can extend the one more similar overwriting just the method that need a different computation. A specific example has been setup to show how to include in Maud a new geometry in which the diffraction pattern is obtained by integrating the Laue rings obtained by an Imaging Plate or CCD flat camera in transmission. The example shows how to provide a different Lorentz-Polarization correction for that specific geometry. The coding example with the instructions to compile and include the class in Maud is available via http://www.ing.unitn.it/~luttero/maud/.

Another extendable feature of Maud is concerning the diffraction datafile format. To let Maud recognize and load a new data format it is possible to write a standalone class, compile it and add it to the libraries. A specific example with all the instructions and steps is reported at the same address as for the previous geometry case.

Further examples will be available in the future at that address to cover more extendable features of the program.

Fast computing for crystallography and Java

The following technique were used for the program Maud and we relate them to the Java language, but most of them are very general and have their non Java counterpart so you can apply them also to every other language or program.

Most of the people/developers around think of Java as a poor language in term of computation speed. This is not true and the wrong feeling is related principally to two reasons. The first is that Java at the beginning was just an interpreted language not running natively on the platform. This has changed a lot in the last few years with the advent of JIT (Just In Time) compilers whose job is to compile natively on the fly the java binary code for the specific processor as the various classes are loaded and used by the JVM (Java Virtual Machine). Running natively, there are no reasons why a division, a multiplication or a trigonometric function should be slower in Java than in another language. For the Windows platform for example the JIT are so optimized that in a pure mathematical computation most of the time you reach 90-95% of the speed of the C counterpart. The second fact is related instead to the general slowness of the interfaces written in Java. This is mainly due to the fact that most of these interfaces are coded using the Swing library that is very complete and fancy (changeable look and feel included) for the number of widgets it provide, but the internal design was not focused on speed and it is well known it provides a redundant number of event monitoring tasks and layers even for simple windows. But this has nothing to do with the speed of execution in Java. Interfaces written in pure AWT (the original library for interfaces in Java) are instead pretty fast, but more simple. Some developers, to optimize for speed, just use AWT for most of the interface, adding few Swing components just for the more fancy controls.

The case of the swing library is a typical case that shows how in an object oriented environment you can write very robust and complicated programs/libraries, but it is pretty easy to loose speed. Focusing on speed generally weakens your program and it is not very easy to write complicated OO programs that are both robust and fast. Non-OO languages normally permit to easily write fast programs, but then it is very difficult to create one that is very large and robust.

General strategies for speed optimization (profiling + rewriting, caching and native computation)

Before proceeding, I would like to recall the sentence of a developer (I forget the name and even the exact words) but the important point is that "… The developing of a program is generally divided in two parts: the general implementation to provide all the features the program needs, and the optimization step. In my experience the optimization step always starts too early…"

In the case of the Maud program a similar approach has been used and we realized that every time we try an optimization to gain speed we are likely to loose generality, ease of maintenance but we are especially introducing bugs. Let's see one example.

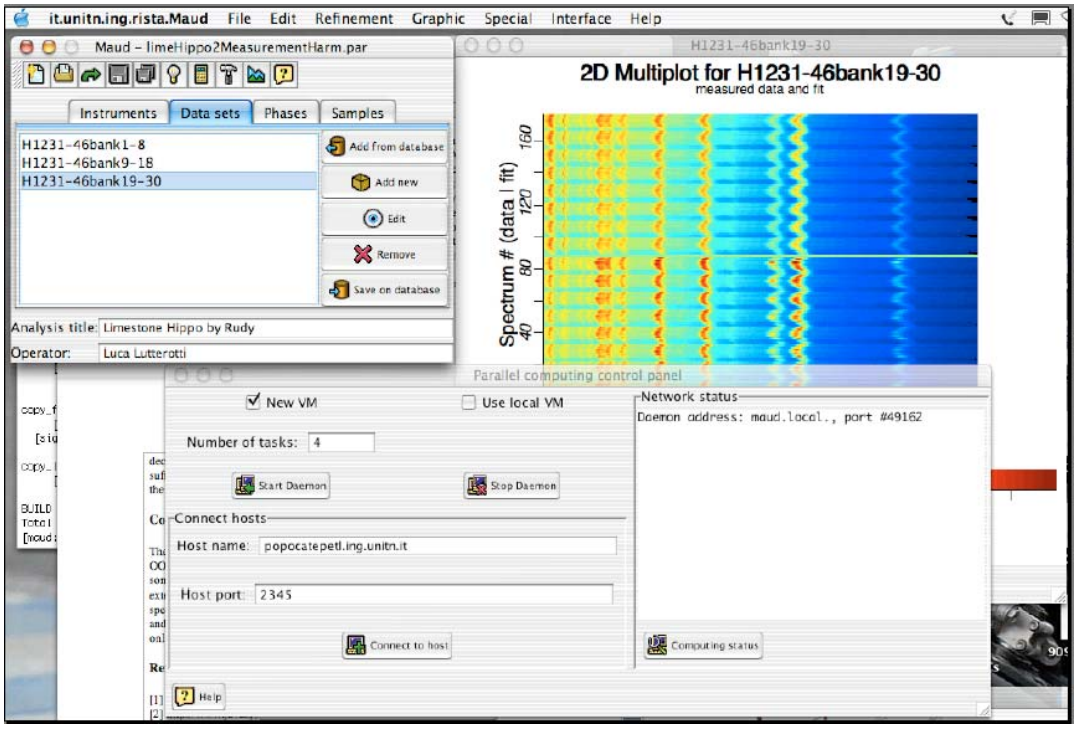


Fig 2: *Maud in action with the JPVM console (for distribute computing) opened. The Daemon has been started on the local machine (principal machine) and it is ready to accept incoming connections. The same console can be used by another machine running Maud to connect to this principal machine by entering the TCP/IP address and port number as shown in the same window. Pressing the "Connect to Host" button, start the connection and then the connecting machine is ready to accept tasks distributed by the principal machine.*

Suppose we have a phase object that contains other objects some of which are atoms or lists of atoms. Then we have another object that is in charge of optimizing the atoms positions from a set of intensities. This object needs at a certain time to compute the structure factors based on the current positions of the atoms. So it asks to the phase to compute the structure factor for a certain (hkl); the method (or object, if you want it to be changeable or customizable) that computes the structure factor asks for the current position and scattering factor to each atom. The phase computes then all the equivalent positions based on the current space group and gives them back to the method responsible of structure factor computation. The resulting OO code is simple, easy to read, maintain, debug and modify, but what about speed? If the structure factor is evaluated once or just a few times, than it is perfect; but suppose you have a genetic algorithm to solve the structure that is asking this computation several times, then your program will become extremely slow.

The first technique to speed up the computation is to cache the more requested data instead of re-computing it all the time. In this case we have different possibilities. First the phase may store the list of all actual equivalent positions for all atoms and use them without the need to recalculate them each time. We have to provide a mechanism to re-compute these positions every time something changes like the space-group, one atom position or an atom is removed or added. Further optimization can be provided in the structure factor computation method to optimize the computation for different space groups etc.

The problem with the last approach is that the refreshing mechanism should be very complete and robust; otherwise something will not be computed properly. But then, if the structure of the computation need to be changed, it becomes very difficult because the refreshing mechanism must be taken into account and the object encapsulation has been partially lost. Indeed if the refreshing mechanism is very good, you can even improve the speed over a normal procedural program, as we will avoid redundant computation for something that has not been changed.

In conclusion, the "computing only when refresh needed" technique is very powerful but prone to bugs and costly from the programming point of view. Another drawback of the caching mechanism is about the large memory consumption for storing the data. In general it is not needed that everything in the program is structured following this model. Most of the programs just spend a consistent fraction of cpu time on few functions and just optimizing these few functions is sufficient to get nearly full speed.

In Maud the last approach has been adopted. In any case a general mechanism for refreshing computation only when needed has been provided by a basic object that all the diffraction objects (phases, atoms, instruments, samples etc.) must extend. Then, additional gain in performance has been achieved (in a version still to be released because of testing) using a profiling technique to identify where the program was spending most of the time. These algorithms have then been optimised in order to maximum the speed.

In Java, profiling is included in the basic jdk of every Java implementation. It can be used in every Java program, even without having access to the source code. In the case of Maud, the program can be started by typing the following on a command line (DOS window or UNIX shell; the following is for the UNIX shell):

>java –mx1024M –cp Maud.jar:lib/miscLib.jar……:lib/ij.jar it.unitn.ing.rista.Maud

We are omitting here all the *.jar libraries contained in the lib folder that should go on the place of the dots sequence. To use profiling you just add the following string (for full reference type java –help and follow directions):

-Xrunhprof:heap=sites,cpu=samples,depth=15,monitor=y,thread=y,doe=y

just before the –mx1024M argument that specify the maximum memory amount that can be used by the program (1Gb). If we perform the computation we need to optimize, at the end a log file is produced reporting statistically where the program is spending most of its time (in percentage and cpu times) and also the entire tree of the calling process of every thread. If debug is turned on during the compilation of the java libraries it returns even the corresponding line numbers in the source code. So it is quite easy to identify where you need to optimize your computation efficiently at the minimum developer effort.

In Maud, for a simple structure refining of one phase with one spectrum, most of the cpu time was dedicated to the computing of the profile function for all peaks to generate the spectrum. The first thing done was to optimize this computation in pure Java at maximum by avoiding any unnecessary redundant computation and optimizing every loop. For example a division is well known to be slower than a multiplication so every division by a constant was substituted by the multiplication with the reciprocal value of that constant. Further speed gain on this particular routine can be done by more radical approaches as reported in the following paragraph and they have been tested or are in use in the last Maud version (available soon).

Special techniques for fast computation

Many techniques have been developed for fast computation on today computers. We will report here some of them that we tested using the Maud program. Some of them require special computers or processors; others are more general.

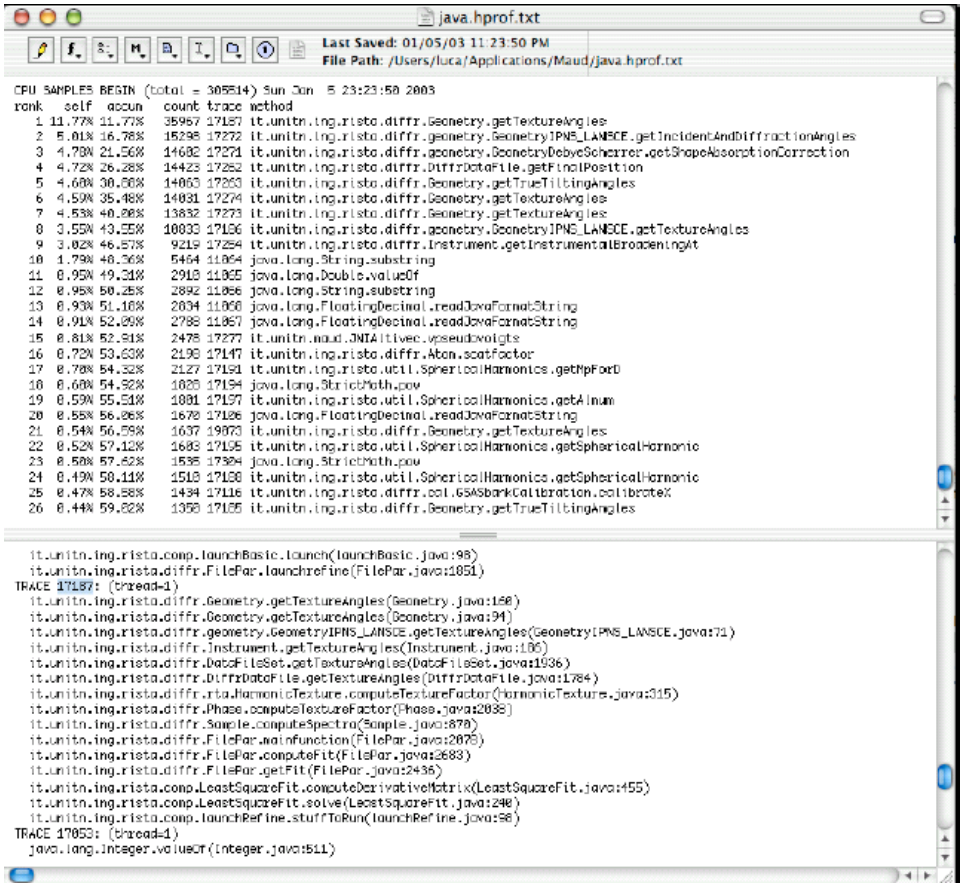


Fig 3: *Results of the profiling after using the vector library JNIAltivec. As you can see the method vpseudovoigts is now ranked 15th and acoount only for 0.81% of the cpu time. After this profiling also the getTextureAngles (method to convert angles from instrument definitions to pole figure angles for texture analysis) routine has been implemented in the vector library. The window is split in two; the getTextureAngles method tracked in the first panel has a trace number 17187 that can be used to find the entire trace stack shown in the lower panel. Source code lines in parenthesis.*

Vector computation.

Some recent processors, as for example the Motorola G4 or the upcoming IBM Power4 include a vector unit (for the G4 is called Altivec [3]) that performs operations in parallel on vector elements instead of scalar values. This is useful only if the computation can be arranged to work on vector elements instead of single values and special coding techniques must be used to take advantage of it. For example in the case of the peak function computation, a special Java native vector library (in C/C++ language) including the Pseudo-Voigt has been developed for the Altivec [4]. The native C/C++ routine specification (usable from Java directly) is:

public static native int vpseudovoigts(float[] x, float[] f, int[] minindex, int[] maxindex, float[] intensity, float[] eta, float[] hwhm, float[] position);

It requires an array of “x” coordinates and return the intensities computed in the array “f”. The arrays “intensity”, “eta”, “hwhm”, “position”, “minindex” and “maxindex” contain the values, for all Pseduo-Voigt peaks, respectively of the integrated intensity, Gaussian content (from 0, full Gaussian, to 1, full Cauchy), half width at half maximum, center position in x coordinate, starting and final index of the x coordinate for the range over which the function should be computed. The native C/C++ routine divides the x and f arrays in multiple lengths of the vector unit (128 bit unit, 4 floats of 32 bit each for the Altivec) and for each PV peak performs the computation in vector basic operations. A different version of the routine was created at the beginning doing the same computation on only one Pseudo-Voigt at time (vpseudovoigt); but then most of the cpu time was spent to move the “x” and “f” arrays between the

native library and the Java Virtual machine. So a further gain in speed has been realized passing all the PV peaks at once (vpseudovoigts). Both routines are available in the library. Special care has been devoted in the native library to avoid moving too much data from the "scalar" to the "vector" unit, as this could be a bottleneck in the vector computation. A gain by more than a factor of two has been gained over a pure native scalar approach and even more respect to the pure Java approach. In the overall economy of the program (only simple Rietveld refinements) a gain in speed between 20 and 30% has been realized. More optimizations can be done by converting other critical routines to vector computation.

Parallel computing

Another powerful approach is to use parallel computing to benefit of multiprocessor machines. Java makes it very easy to write programs for parallel computing. Java has built-in a thread system, so the critical part is just to divide the computation in parallel tasks. After this has been accomplished it is possible to put each task in a different thread. Threads are executed in Java concurrently and if more than one processor is available the Java Virtual machine spreads automatically the different threads over the available processors. Creating and running a thread is very easy, so we just write:

```
Thread task[] = new Thread[allTasksWeNeed];
for (int i = 0; i < allTasksWeNeed; i++) {
  (task[i] = new Thread() {
    public void run() {
      // here goes the computation code for each task
      .....
    }
  }).start();
}
boolean allTaskFinished = false; // we have to wait now until all tasks has been completed
while (!allTaskFinished) {
  allTaskFinished = true;
  for (int i = 0; i < allTasksWeNeed; i++)
    if (task[i].isAlive()) allTaskFinished = false;
  // we sleep for a while before to check again
  try {Thread.currentThread().sleep(100);} catch (InterruptedException e) {}
}
```

So in these few lines we create a subclass of Thread, we overwrite the method run() to perform our task and we start the thread with the method start(); this for all the tasks we need to run in parallel. At the end we have to wait and check until all threads have finished their computation.

Distributed computing

Multiprocessor machine (with more than 2 processors) are not widely available, so a less efficient but still powerful approach is to distribute the parallel tasks over networked computers. A well known example of this is the SETI at Home project [5]. The JPVM [6] library has been modified and integrated in Maud to provide the support for distribute computing. The JPVM is a library emulating the PVM distributed system and uses basically messaging to spread the computation over others JVMs. The advantages of the system are that it can be used in a heterogeneous network, both over TCP/IP or a cluster. The drawback is that being a messaging system it performs efficiently only for long independent tasks. Otherwise too much time will be spent on messaging over computation. After some unsuccessful trials where the computation was not distributed efficiently, the final strategy was to use the JPVM directly in the least squares algorithm at the beginning. Maud, differently form other Rietveld programs, uses numerical derivatives. This permits the incorporation of some methodologies for which no analytical expression of the derivative is possible. The disadvantage is that for each refinable parameter at least one time the entire

function (or all spectra) needs to be computed, so the speed cannot match in any case the one of a program using analytical derivatives. Using JPVM for derivative computation, each distributed task will be in charge of computing just one derivative and also the least squares matrix can be divided over the network decreasing the memory requirement per computer during such process. In a sufficiently fast network environment this greatly improves the computation speed in the least squares step. The system is still under testing and optimization.

Conclusions

The use of the Java environment has permitted us to write the Maud program with an OO structure that is easily extendable and quite powerful. In the present article we showed some examples of how easy it is to extend some features by writing the desired extension. Also, we have shown how it is possible to optimize the program for speed using different approaches ranging from code optimization, to vector, parallel and distribute computing. Our libraries, which are usable by other programs, are available via the Internet.

References

[1] Maud program, http://www.ing.unitn.it/~luttero/maud/
[2] http://www.javasoft.com/ or http://java.sun.com/
[3] http://www.simdtech.org/home
[4] JAltivec library, downloadable at http://www.ing.unitn.it/~luttero/javaonMac, source code included.
[5] Seti At Home, http://www.seti.org
[6] JPVM, http://www.cs.virginia.EDU/jpvm

Crystallographic Fortran 90 Modules Library (CrysFML): a simple toolbox for crystallographic computing programs.

Juan Rodríguez-Carvajal[1] and Javier González-Platas[2]
(1) Laboratoire Léon Brillouin (CEA-CNRS), CEA/Saclay, 91191 Gif sur Yvette Cedex, France, E-mail: juan@llb.saclay.cea.fr – WWW: http://www-llb.cea.fr/fullweb/powder.htm and (2) Departamento de Física Fundamental II, Universidad de la Laguna, Tenerife, Spain, E-mail: jplatas@ull.es

Abstract

We describe in this paper a short introduction to the Crystallographic Fortran Modules Library (*CrysFML*). This set of modules has been, and is still being developed, by us to facilitate the design an the elaboration of crystallographic computing programs. The whole library is written in a subset of Fortran 95 (F-language) for which free compilers are available. The source code is available to those academic groups interested in cooperative scientific software development.

Introduction

There is presently a huge amount of academic crystallographic programs that are available to the scientific community through the CCP4 and CCP14 web sites [1]. The programs are distributed mainly in the form of executable codes. Source codes are not distributed in many cases. We are aware of only two general sets of computing procedures that are freely distributed and are similar to the system described in this paper. The first is the Cambridge Crystallographic Subroutines Library (CCSL) that has been developed mostly by P.J. Brown and J.C. Matthewman with contributions, mainly for powder diffraction, of W.I.F. David, J.B. Forsyth, J.H. Matthewman and J.P. Wright [2]. The second, Computational Crystallography Toolbox (*cctbx*), by R.W. Grosse-Kunstleve *et al.* [3] is quite recent and several publications have appeared in Journal of Applied Crystallography (see [3] and references therein).

To our knowledge CCSL is presently the most complete set of crystallographic procedures that are freely available. CCSL is written mostly in Fortran 77, but the style of programming is still quite close to old Fortran 66. Many manipulations of array procedures written in CCSL are now obsolete if one uses the modern Fortran 90/95 language. The same is true for some machine dependent procedures that are implemented in CCSL. There are several main programs based in CCSL that are mostly used by the neutron crystallographic community. As we will see below the aim of CCSL is similar to ours. CCSL was developed in order to facilitate the development of crystallographic computing programs by the users of special diffraction techniques (polarized neutrons, for instance). The main drawback of CCSL, if one looks from one the modern programming paradigms (structured or object oriented programming), is the lack of modularity and the excessive intricacy of the procedures. The use of COMMON blocks and EQUIVALENCE statements is the way of communicating global variables between independent subroutines and functions.

The case of *cctbx* is in some sense the opposite extreme. There are still many useful procedures for crystallographic computing that need to be developed, but the whole library is written following the object oriented programming (OOP) paradigm. The programming language is C++. We have no experience using this toolbox, so we cannot make positive or negative statements about it.

We will not discussed in this paper the advantages and drawbacks of the different programming languages, but we have to say that seven years ago we were faced to the alternative between Fortran 90 and C++ in order to continue the development of several computing programs. After a period of hesitation and trials we decided clearly to work in Fortran 90 for the following reasons:

1. Simplicity and clarity of the syntax and the new facilities for global array manipulation. This is important for the common scientist that may write programs occasionally. This makes programming in Fortran more natural and problem solving oriented.
2. Availability of many OOP techniques in modern Fortran: user-defined types, encapsulation, overload of procedures and functions. We consider the lacking features (e.g. inheritance and class methods) of less importance for scientific computing than those already available. In any case these lacking features will be easily implemented (simplifying the code) as soon as they become available in the forthcoming new standard Fortran 200x (see reference [4]).
3. The powerful implicit interface provided by encapsulating all functions and subroutines in MODULES, allowing to catch many errors at compile time, if one uses the *intent* attribute for procedure arguments. We may consider that Module Oriented Programming (MOP) as an alternative/complement to OOP.
4. Efficiency of the generated executable codes compared to C++ programs of similar complexity.
5. Compatibility with legacy code and availability of a huge amount of free mathematical subroutines and functions. The criticism of many people fanatic of the OOP paradigm about the lack of re-usability of procedures written by following the structured programming paradigm is clearly not adequate. The major parts of mathematical software systems (LAPACK for instance) that are still in use, and in the kernel of many well-known packages, are written in Fortran 77.

One of us (JRC) started many years ago to work with CCSL and realized that it was more easy to re-design progressively a new library using Fortran 90 to encapsulate similar procedures in modules than to modify the whole CCSL. We have then developed a set of Fortran 95 modules, Crystallographic Fortran Modules Library (*CrysFML*), which may be *used* (in the Fortran 95 sense) in crystallographic and diffraction computing programs. The scope of the new library is identical to that of CCSL, except that, at present, it is not so complete as it is CCSL. Modern array syntax and new features of Fortran 95 are used through the modules. We take advantage of all OOP techniques already available in Fortran and we have used, in our practice, a subset of Fortran 95 called *F* for which free compilers are available for most of the current operating systems [5]. We aim to preserve the efficiency, the simplicity and the adequacy of modern Fortran for numerical calculations. *CrysFML* has never been communicated to the scientific community but a preliminary account was presented in a poster [6] and a short talk in the computer fayre

at the last International Union of Crystallography meeting in Geneva. The present status of *CrysFML* will be summarized in this paper.

Structure of CrysFML

The present *CrysFML* contains procedures for reading files of many different formats, string utilities for handling the reading in interpreted free format, generation and reading of CIF files, mathematical modules, modules for generating space groups from their Hermann-Mauguin or Hall symbols for whatever setting. More generic space groups with non-conventional lattice centring vectors can also be built using user-defined generators. Reflection handling modules may be used for generating reflections in selected regions of reciprocal space and for calculating structure factors, etc.

The documentation is written within the source code using special comment symbols and rules. A document, in HTML format, containing the description of all modules and procedures can be generated using the program *get_doc* based itself on *CrysFML*. An example of the generated document is seen in Figure 1.

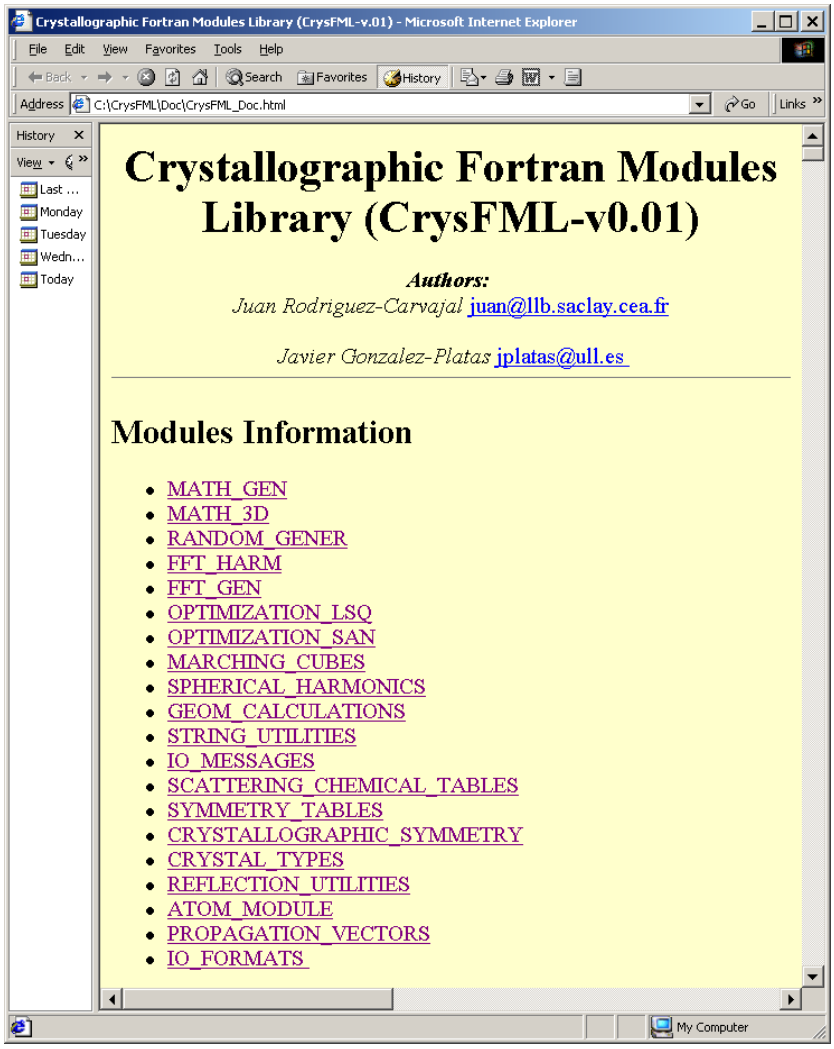


Figure 1: *Starting page of the documentation written automatically in HTML by a Fortran program that gets the information from the source code comments.*

As an example of the style used to write the modules we give in Figure 2 a part of the source code constituting the starting lines of the module *Crystallographic_Symmetry*. One can see easily how the dependencies and accessibility of the different procedures and variables within a module are defined.

```
Module Crystallographic_Symmetry

   !---- Used External Modules ----!
   !Use Mod_fun    !To be commented for non-F compilers
   Use Symmetry_Tables
   Use Math_gen,         only: sp, Sort
   Use Math_3d,          only: Traza, Determ_A, matrix_inverse, Zbelong, Modulo_Lat, &
                               equal_matrix, Resolv_Sist_3x3
   Use String_Utilities, only: Equal_Sets_Text, Pack_String, Get_Fraction_2Dig,      &
                               Get_Fraction_1Dig, Frac_Trans_1Dig, L_Case,           &
                               U_case, Ucase, Getnum
   implicit none

   private

   !---- List of public variables and types ----!
   public :: err_mess_symm, err_symm
   public :: hexa, inlat, lat_type, ltr, Nlat
   public :: SpaceG, Space_Group_Type, Sym_Oper_Type, Wyckoff_Type, Wyck_Pos_Type

   !---- List of public overloaded operators ----!
   public ::  operator (*), operator (==)

   !---- List of public functions ----!
   public  :: ApplySO, Axes_Rotation, Get_Laue_Num, Get_Multip_Pos, Get_Occ_Site,     &
              Get_Pointgroup_Num, Is_New_Op, Lattice_Trans, Spgr_Equal, Sym_Prod

   !---- List of public subroutines ----!
   public  :: Decodmatmag, Get_Centring_Vectors, Get_Crystal_System, Get_Lattice_Type,           &
              Get_Laue_Pg, Get_Laue_Str, Get_orbit, Get_Pointgroup_Str, Get_So_From_Fix,         &
              Get_So_From_Gener,Get_So_From_Hall, Get_So_From_Hms, Get_Spg_From_Gener,           &
              Get_Stabilizer,Get_String_Resolv,Get_SubOrbits,Get_Symel, Get_Symkov, Get_SymSymb, &
              Init_Err_Symm, Inverse_Symm, Latsym, Read_Msymm, Read_Xsym, Searchop,              &
              Set_Spacegroup, Setting_Change, Sym_B_Relations, Sym_Prod_St, Symmetry_Symbol,     &
              Write_Spaceg, Write_Sym, Write_Wyckoff, Wyckoff_Orbit
```

Figure 2: *Header of the module Crystallographic_Symmetry, showing the modules it uses and all the public types and procedures available to other modules using it.*

The structure of *CrysFML* is quite simple:

a) On top of the hierarchy there are *generic mathematical modules* (e.g. *Math_gen*, *Math_3D*, *Random_Gener*, *FFT_Harm*, *FFT_Gen*, *Optimization_LSQ*, *Optimization_SAN*…) that may be used by other more specialized modules.
b) Other highly generic modules are concerned with the manipulation of string characters. In particular the module *String_Utilities* and the module *Free_Format_Reading* are quite useful for the use indicated by their own names.
c) A small module defining the way the messages are passed to the screen or windows, *IO_messages*, with generic subroutines that adopt different aspects depending on the use or not of third party graphical libraries.
d) Chemical and scattering data modules.
e) Modules related with symmetry: *Symmetry_tables* and *Crystallographic_Symmetry*
f) Crystallographic modules: *Crystal_types*, *Reflection_utilities*, *Atoms_Modules*, *Structure_Factors*, *Propagation_vectors*, etc.
g) Specialized modules: *Peak_Shapes*, *IO_Formats*, etc.
h) Candidates: a set of modules that are being considered to form part of the library in future developments after debugging.

In spite of the internal dependencies of modules in *CrysFML*, the user is not obliged to use all the modules, or even to create a unique library. There are many standalone modules. One can easily take modules for a particular purpose and disregard the remaining modules.

Making programs using CrysFML

Let us discuss two simple examples of programs of general crystallographic interest using the module *Crystallographic_Symmetry* and the module *Reflection_utilities*. First of all the user should be aware of the existence in *CrysFML* of the Fortran type called *Space_Group_Type*, which is the most important defined type in symmetry modules. The definition is given in Figure 3.

```
Type :: Space_Group_Type
     Integer                                 :: NumSpg        ! Number of the Space Group
     Character(len=20)                       :: SPG_Symb      ! Hermann-Mauguin Symbol
     Character(len=16)                       :: Hall          ! Hall symbol
     Character(len=12)                       :: CrystalSys    ! Crystal system
     Character(len= 5)                       :: Laue          ! Laue Class
     Character(len= 5)                       :: PG            ! Point group
     Character(len= 5)                       :: Info          ! Extra information
     Character(len=80)                       :: SG_setting    ! Information about the SG setting
                                                              ! (IT,KO,ML,ZA,Table,Standard,UnConventional)
     Logical                                 :: Hexa          !
     Character(len= 1)                       :: SPG_lat       ! Lattice type
     Character(len= 2)                       :: SPG_latsy     ! Lattice type Symbol
     Integer                                 :: NumLat        ! Number of lattice points in a cell
     real(kind=sp), dimension(3,12)          :: Latt_trans    ! Lattice translations
     Character(len=51)                       :: Bravais       ! String with Bravais symbol + translations
     Character(len=26)                       :: Centre        ! Centric or Acentric
     Integer                                 :: Centred       ! =0 Centric(-1 no at origin)
                                                              ! =1 Acentric
                                                              ! =2 Centric(-1 at origin)
     real(kind=sp), dimension(3)             :: Centre_coord  ! Fractional coordinates of the inversion centre
     Integer                                 :: NumOps        ! Number of reduced set of S.O.
     Integer                                 :: Multip        ! Multiplicity of the general position
     Integer                                 :: Num_gen       ! Minimum numb. of oper. to generate the group
     type(Sym_Oper_Type), dimension(192)     :: SymOp         ! Symmetry operators
     Character(len=40), dimension(192)       :: SymopSymb     ! Strings form of symmetry operators
     type(wyckoff_type)                      :: Wyckoff       ! Wyckoff Information
     real(kind=sp), dimension(3,2)           :: R_Asym_Unit   ! Asymmetric unit in real(kind=sp) space
  End Type Space_Group_Type
```

Figure 3 : *Generic space group type. One can define arrays of space groups and work globally with the user defined symbols derived from Space_Group_Type.*

From the large number of procedures existing in *Crystallographic_Symmetry* one of the most useful is the subroutine *Set_Spacegroup*. Apart from the simplest cases described in the caption of Figure 4, one may call the subroutine providing user-defined generators in the array of strings *gen*. One can make a call to the subroutine as follows:

```
! Declarations omitted
Ngen=3
Gen(1)="y,-x, z"
Gen(2)="-x,-y,-z"
Gen(3)="x+1/2, y+1/2, -z"
Call Set_Spacegroup(Spacegen,Spacegroup,Gen,Ngen,"GEN")
```

On output the object *Spacegroup* of type *Space_Group_type* is filled with all possible information obtained from the list of given generators.

```
Subroutine Set_Spacegroup(Spacegen,Spacegroup,Gen,Ngen,Mode,Force_Hall)
   !----Arguments ----!
   character (len=*),                intent(in )           :: SpaceGen
   Type (Space_Group_Type),          intent(out)           :: SpaceGroup
   character (len=*), dimension(:),  intent(in ), optional :: gen
   Integer,                          intent(in ), optional :: ngen
   character (len=*),                intent(in ), optional :: Mode
   character (len=*),                intent(in ), optional :: force_hall

   .........
```

Figure 4: *Header of the subroutine Set_Spacegroup. Only two arguments are needed in the most simple cases. The string SpaceGen may contain the Hermann-Mauguin (H-M) symbol, the Hall symbol or simply the number of the space group. The object Spacegroup is provided by a call to the subroutine.*

An example of a simple program that gives as output the information contained in the object *Spacegroup* is given in Figure 5. The program uses the module *Crystallographic_Symmetry*, but only three public procedures. In the declaration part it is defined the string *spg_symb* that may contain the number of the space group, the H-M or the Hall symbol. The object SPG will hold all the information concerning the space group. The program has an infinite loop in which it is asked to enter a space group, if nothing is given (pressing the "enter" key) the program exits from the loop and stops.

```
!----------------------------------------
!  Example of simple program using CrysFML
!----------------------------------------
  Program Get_SPG_info
    use crystallographic_symmetry, only:  &
        space_group_type,set_spacegroup, write_spaceg
    character(len=20)       :: spg_symb
    type(space_group_type)  :: SPG

    do
      write(unit=*,fmt="(a)",advance="no") &
      " => Please enter a space group (H-M/Hall/number): "
      read(unit=*,fmt="(a)") spg_symb
      if(len_trim(spg_symb) == 0) exit
      call set_spacegroup(spg_symb,SPG)
      call write_spaceg(SPG,full=.true.)
    end do
    stop
  End Program Get_SPG_info
```

Figure 5: *Code of a program calling symmetry procedures of CrysFML.*

The argument *full* in procedure *write_spaceg* means that all detailed information in asked to be output in the screen. One may change the instruction to write directly to an already opened file. For instance writing:

```
Call write_spaceg(SPG,iunit=3,full=.true.)
```

directs the output to the file connected with logical unit 3. An example of the output of the above simple program is given is figure 6. The answer to the question asked by the program was "15" (without quotes).

It may be noticed that all the important information is output. Not only the alphanumeric form of the symmetry operators is given but also the symmetry symbol (kind of symmetry element: axis, plane, … and its localisation in the space) associated with the operator. To get this symbol, an internal call to the subroutine *Symmetry_Symbol* (deriving the symmetry symbol from the symmetry operator) is performed in *write_spaceg* when the argument *full* is present. The output of the Wyckoff positions is only possible, at present, when the space group is in one of the standard settings. It is not possible to derive algorithmically the order of Wyckoff positions as it is given in the International Tables of Crystallography. For that reason the corresponding information for the standard setting is described within the module *Symmetry_Tables*.

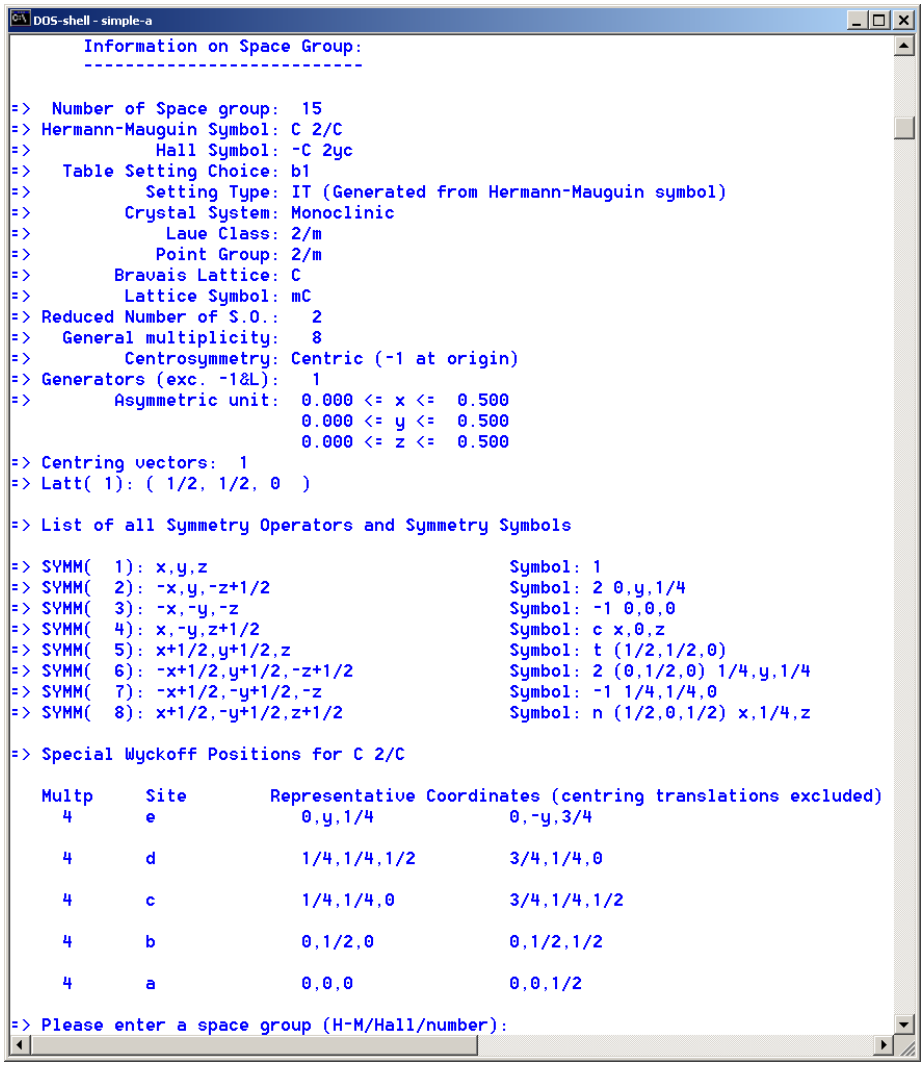


Figure 6: *Example of output of the program in Figure 5.*

This example shows how simple is to write a useful non-trivial program by using *CrysFML*.

Let us consider another example. Suppose that we want to write a procedure to list all possible space groups compatible this a list of "observed" reflections obtained from a Le Bail fit of a powder diffraction pattern. It is quite easy to see how to do that by reading the code in Figure 7.

```
Program Check_Group
  use crystallographic_symmetry, only: Space_Group_Type, set_spacegroup
  use reflections_utilities, only: Hkl_Absent
  use Symmetry_Tables, only: spgr_info, Set_Spgr_Info

  ....... ! Read reflections, apply criterion of "goodness" for checking,
          ! set indices i1,i2 for search in space group tables ...
  ....... ! omitted for simplicity
  call Set_Spgr_Info()
  m=0
  do_group: do i=i1,i2
    hms=adjustl(spgr_info(i)%HM)
    hall=spgr_info(i)%hall
    if( hms(1:1) /= "P" .and. .not. check_cent ) cycle do_group ! Skip centred groups
    call set_spacegroup(hall,Spacegroup,Force_Hall="y")
    do j=1,nhkl
       if(good(j) == 0) cycle  !Skip reflections that are not good (overlap) for checking
       absent=Hkl_Absent(hkl(:,j), Spacegroup)
       if(absent .and. intensity(j) > threshold) cycle do_group  !Group not allowed
    end do
    ! Passing here means that all reflections are allowed in the group -> Possible group!
    m=m+1
    num_group(m)=i
  end do  do_group
  write(unit=*,fmt=*) " => LIST OF POSSIBLE SPACE GROUPS, a total of ",m," groups are possible"
  write(unit=*,fmt=*) "     ------------------------------------------------------"
  write(unit=*,fmt=*) "     Number(IT)      Hermann-Mauguin Symbol     Hall Symbol"
  write(unit=*,fmt=*) "     ------------------------------------------------------"
  do i=1,m
    j=num_group(i)
    hms=adjustl(spgr_info(j)%HM)
    hall=spgr_info(j)%hall
    numg=spgr_info(j)%N
    write(unit=*,fmt="(i10,4a)")  numg,"                  ",hms,"          ",hall
  end do
  ...............
```

Figure 7: *Program providing a list of all the possible space groups that are compatible with a set of "observed" reflections.*

Part of the program has been omitted for simplicity and to save space. The code is sufficiently clear to understand the procedure. The important points are:

1. Read *hkl* in the array *hkl*(:,:), *intensity*, *sigma*, Bragg angle and *fwhm*
2. Establish a criterion for consider reflections good for checking taking into account the possible overlap and a threshold in intensity. The integer array *good*(:) has a component equal to zero if the corresponding reflection is not good for checking.
3. After making use of the module *symmetry_tables* set up the array *spgr_info*, which has components of type *spgr_info_type*: structure containing the number of the space group, the H-M and Hall symbols and additional information to obtain the asymmetric unit in real space.
4. Tell to the program if centred space groups have to be checked (logical variable *check_cent*) and set the indices to check in the loop over space groups. These indices depend on the crystal system and are available from the *symmetry_tables* module.
5. Use the logical function *HKL_absent* from *Reflection_Utilities*, to test the possible groups.

The code above is a minimal test. If needed, one can refine the procedure and generate a hierarchy of space groups and figures of merit.

Computing programs currently using CrysFML

All multipattern versions of *FullProf* are based on *CrysFML*. The source code of *FullProf* was re-written completely during 1997, and progressively its dependency on CrysFML has increased.

Other public programs using *CrysFML* are FOURIER, GFOURIER and *EdPCR*. These programs work on Windows and Linux and are already distributed via the LLB Web site. The first two programs are dedicated to i) the Fourier analysis of diffraction data and ii) *EdPCR*, a new (still under development) Graphic User Interface (GUI) to the *FullProf* input control file (.PCR extension).

A number of other programs are also based on CrysFML but are not yet distributed publicly in the scientific community but strongly used by people working in collaboration within our group at LLB, ILL,

PSI and several Universities and National Laboratories in European Countries. A non-exhaustive list is the following:

BASIREPS: Program for calculating basis functions of irreducible representations of space groups. This program is useful for determining magnetic structures and phonon symmetry analysis. BASIREPS is presently distributed together with *EdPCR*.

SIMBO: Program for the analysis of the magnetic topology of an arbitrary crystal structure. Generates a formal description of the Fourier transform of the exchange interactions to be used by other programs and a file to be used by the MonteCarlo code MCMAG.

ENERMAG: Program to analyse the classical magnetic energy as a function of the exchange interactions and the point in the Brillouin Zone. It uses one of the output files of SIMBO. This program can be used to generate theoretical magnetic phase diagrams in the J-space in order to get insight into the experimentally determined magnetic structures.

SIMILAR: Program to convert settings for describing a crystallographic structure. It determines automatically the splitting of Wyckoff positions on going from a space group to one of their subgroups. Calculates subgroups of a space group, co-set decompositions, etc.

DATARED: Program for data reduction of single crystal data. It handles twinning and incommensurate magnetic and crystal structures. At present it can read data provided by several integration programs, in particular several versions of COLL5 (LLB and ILL). Prepares files to be read by *FullProf* when using single crystals.

Current tasks and conclusions

We hope to have demonstrated in this short and biased document the usefulness of *CrysFML* for rapidly writing crystallographic programs. We still have more work to do to improve the library and to increates its capabilities. Some of the remaining tasks that concern the development of *CrysFML* are the following:

Complete the documentation of the existing modules and make a PDF version of the manual.
Continue development of the library, implementing more modules.
Prepare a Web site for *CrysFML* with a protocol for downloading the library.

We expect, after putting the *CrysFML* in a public Web site to increase the number of informal participants on the project. For reasons related to our schedule we cannot guarantee a date for the public availability of *CrysFML*. In the meantime, if somebody wishes to obtain the library, send an E-mail to juan@llb.saclay.cea.fr.

References:

[1] The sites of CCP4, dedicated mainly to macromolecular crystallography, and CCP14 are respectively: http://www.ccp4.ac.uk/ and http://www.ccp14.ac.uk/.
[2] P.J. Brown and J.C. Matthewman, CCSL: http://www.ill.fr/dif/ccsl/html/ccsldoc.html
[3] R.W. Grosse-Kunstleve *et al*., *J. Appl. Cryst*. **35**, 126 (2002)
[4] John Reid, WG5 Convener, *The new features of Fortran 2000*, PDF document available directly from the Internet: ftp://ftp.nag.co.uk/sc22wg5/N1451-N1500/N1495.pdf.
[5] All free F-compilers can be downloaded from the site: ftp://ftp.swcp.com/pub/walt/F
See also http://www.fortran.com/F/compilers.html
[6] J. Rodríguez-Carvajal and J. González-Platas, *Acta Cryst.* A**58** (Supplement), C87.

Use of JAVA applets for teaching

Jean-Jacques Rousseau,
Université du Maine, Le Mans, France.
E-mail: jjrouss@univ-lemans.fr – WWW: http://www.univ-lemans.fr/enseignements/physique/02/

Why use Java for teaching?

- It is **a portable language** which generates code for a virtual machine: if the browser installed on the computer (PC, Mac...) is equipped with a Java virtual machine, it is possible to download and exploit the applets. These applets use, in a transparent way for the programmer, the graphic objects (buttons, elevators/sliders, etc...) of the operating system.
- It is a **reliable and protected** language **:** an applet cannot write on the user hard disc.
- It is a language equipped with simple but sufficient **graphic capacities**.
- It is a language which allows the realization of **graphic animations**
- The mathematical library respects the IEEE 754 standards and allows scientific computation.
- It is very simple to incorporate applets in a HTML formatted page.
- The Java syntax is very close to that of C. Java is very strict with the types of data and it is a strongly object oriented language. For simple applications, one can use it like a traditional language.
- Finally, I made the choice, for reasons of maximum compatibility, to use only the Java versions 1.0 and 1.1. To use the later versions which are admittedly equipped with higher graphic capacities, most browsers will ask for a "plug-in"which is usually easy to find and install.

The applet “space groups”

This applet (approximately 3500 lines of code) which gives the chart of the 230 space groups is available at the address:
http://www.univ-lemans.fr/enseignements/physique/02/cristallo/espace.html

Applet design

For each selected space group, the program carries out 4 operations:

- Layout of the symmetry elements.
- Determination of the equivalent positions in literal form.
- Calculation and layout of the equivalent positions.
- Determination of the diffraction conditions.

Layout of the symmetry elements

This layout is carried out starting from a table where the elementary generators of each space group are coded. The application of some simple rules on lattice translation and on the nature of the elements of symmetry of the space group makes it possible to reduce this table.

For all the centrosymetric groups, the origin is taken on a centre of inversion allowing to simplify later calculations of the equivalent positions.

For some cubic groups, elements of symmetry were voluntarily omitted for clarity.
As an example, here is how the three representations of the Amm2 space group are coded:

```
G[38]="MV000090 MV000000 CV002500 200000 210025";
H[38]="MV000090 MH0000 BH1025 02001090 1200209025";
K[38]="MH0000 MV000000 AV002500 02000000 12002500";
```

CV000090 => C = mirror c, V = vertical, 00 and 25 = coordinates (in 1/100) of the origin from the mirror, 00 = angle (in degree) of the mirror with Ox.

1200209025 => 12 = horizontal axis 2_1, 00 = passes by the origin, 20 = to draw in 2nd position, 90 = angle in degrees of the axis with Ox, 25 = dimension of the axis into 1/100.
210025 => vertical axis 2_1 which passes by the point of coordinates 00/100 and 25/100.

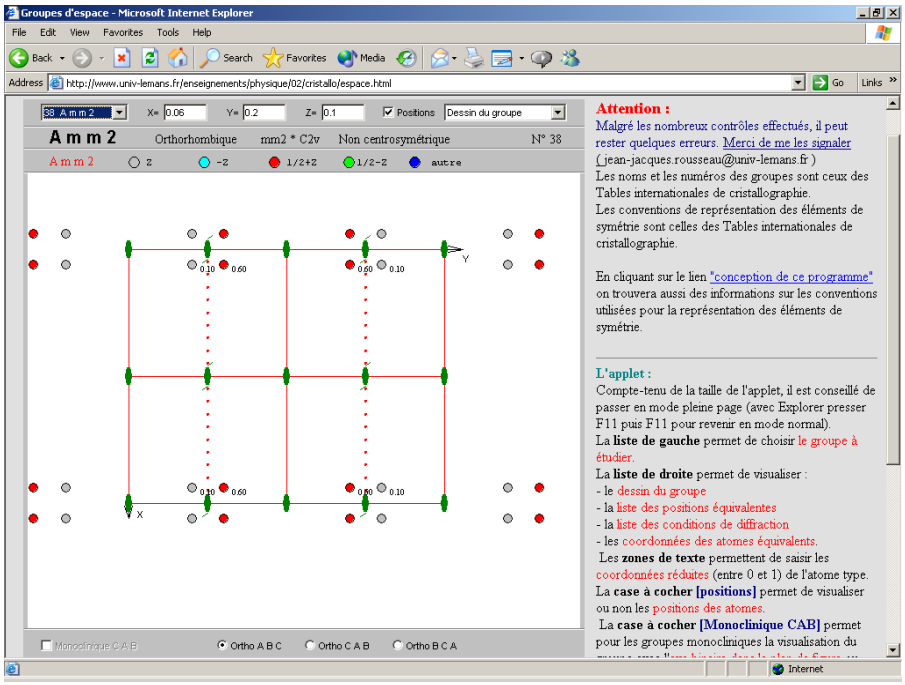


Fig. 1: *Screen image of the space groups applet showing the drawing of symmetry elements of Amm2.*

Determination of the equivalent positions in literal form.

In the calculation of the equivalent positions, one separates the effect of the operations of symmetry of the specific point group from the influences of the translations. After decoding of the Herman-Mauguin symbol of the space group, one determines by application of the point symmetry operators how the coordinates of the node [1, 1, 1] change. The 3x3 matrices thus obtained for each equivalent position are preserved in a table (matrix) **S.** In the same way a table **T** (vector) contains the sum of the intrinsic translations related to the symmetry elements (screw axes and glide planes) and of the translations related to the position of the symmetry elements in the reference system used.

The initial triplet **X** (vector of components x, y, z) is transformed into a triplet **X'** according to the matrix relation **X' = S.X + T**

The analysis of the components of tables **S** and **T** allows the drawing up of the table of the equivalent general positions in literal form.

The translations related to the lattice mode are then treated globally for displaying the equivalent positions. Finally, for each position thus determined, one adds for the realization of the diagram the integer lattice translations (1, 0, 0) (-1, 0, 0) (0, 1, 0) (0, -1, 0) (1, 1, 0) (-1, 1, 0) (1, -1, 0) (-1, -1, 0).

During the displaying of the list of the literal values, one specifies at the beginning of this list the lattice translations which must be possibly added. The generators used are not always the same ones as those of the International Tables for Crystallography, or are used in a different order. This is why the order of the displayed list can differ from that of the Tables.

Calculation and layout of the equivalent positions

In the calculation of the equivalent positions in numerical form, the user must enter the reduced coordinates of the initial atom. The program calculates the list of the numerical coordinates of the equivalent atoms starting from the list of the literal coordinates and carries out the layout on the screen.

If the atom is in a special position, the program generates two (4, 8, ...) atoms having the same coordinates. A procedure makes it possible to eliminate the redundant atoms and to display only one atom: this allows the study of the special positions. All calculations are carried out in reduced coordinates. For the layouts, the conversion between these crystallographic coordinates and the screen coordinates takes account of the system under study.

For the trigonal, hexagonal and cubic space groups, it is often interesting to modify the default values of the coordinates in order to obtain a more readable projection.

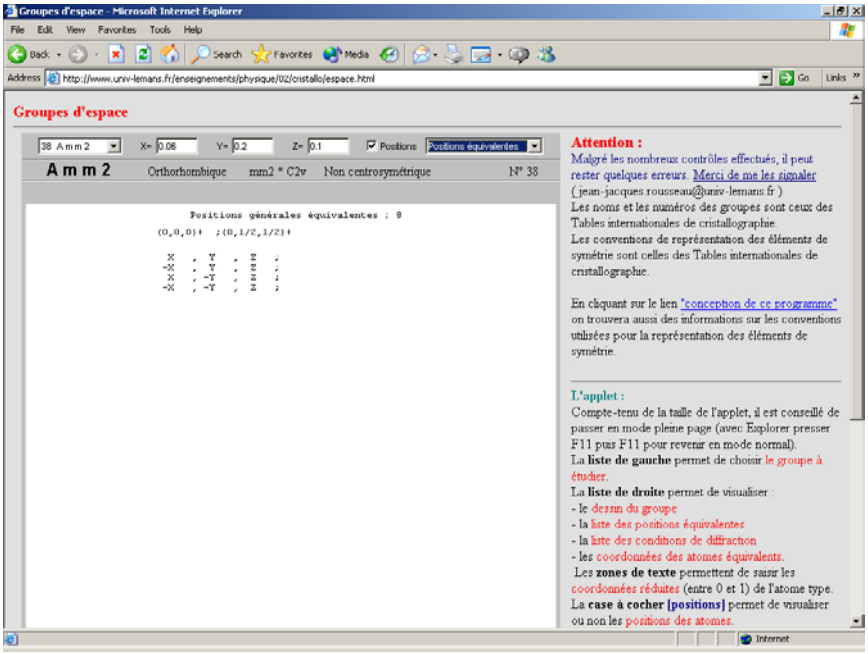


Fig. 2: *Screen image of the space groups applet showing the output of equivalent positions for Amm2.*

Note: An atom is in special position if it is placed on a non-screwed rotation axis or in a non-glide mirror. In the International Tables, the special positions are listed and marked by their Wyckoff symbol (number of equivalent positions followed by an allotted letter in a conventional way).

Determination of the diffraction conditions

The determination of the systematic extinctions for general positions is made starting from the analysis of the contents of table **T**. The program displays the list of the conditions on the indices of the reticular planes which lead to diffraction. The first line of display corresponds to the conditions suitable for the lattice mode. The next lines correspond to the conditions related to the elements of symmetry of translation of the group. The extinctions which are induced by atoms in special positions (on one or more elements of symmetry without translation) are not calculated.

An animation: the Ewald construction

For making my students appreciate all the subtleties of the Ewald construction, I wrote the small program available at the address:

http://www.univ-lemans.fr/enseignements/physique/02/cristallo/ewald.html

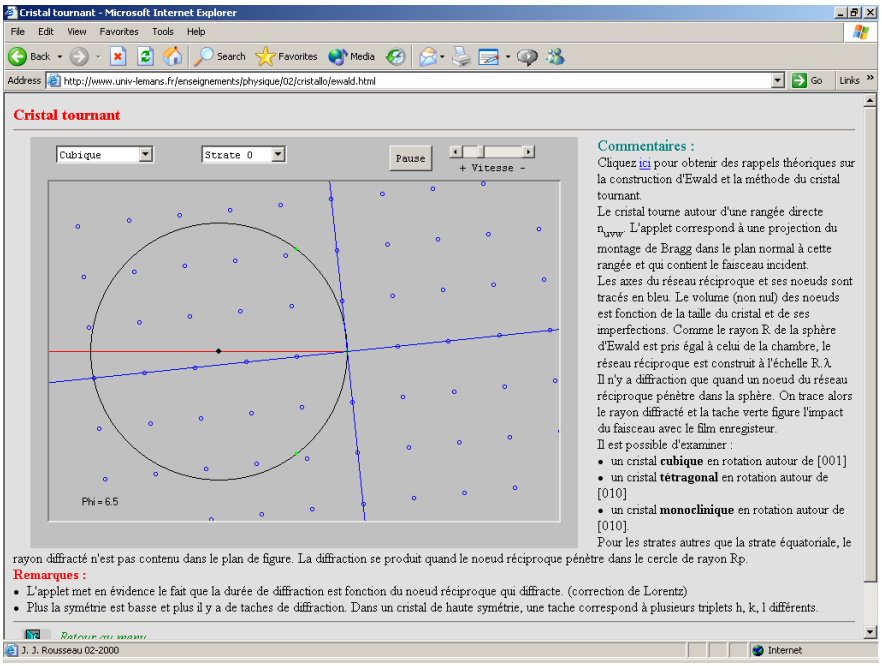


Fig. 3: *Screen image of the Ewald construction applet.*

It is an animation in which I rotate a plane of the reciprocal lattice in the plane of the figure. When a node penetrates the Ewald sphere, the diffracted beam is traced and a spot is drawn on the film.

To obtain fluid animations, without flickering, I highly recommend the method of the double-buffer. See for example (but it is in French):

http://www.univ-lemans.fr/enseignements/physique/02/java/mnujava.html

How does an animation function?

It is necessary to start by defining a "thread" with the instruction:

```
private Thread runner = null;
```

Then it is necessary to implement the methods start(), stop() and run().

```
public void start ()
{       if (runner == null) {
                runner = new Thread(this);
                runner.start( ); }}

public void stop()
{       if (runner ! =  null) {
                runner.stop( );
                runner = null;}}

public void run( )
{       while (true)
         {      try {
                        repaint( );
                        Thread.sleep(time);}
                catch (InterruptedException e)
                        {stop( );}}}
```

A call to the subroutine start( ) begins the subroutine run( ); a call to stop( ) stops it.
The subroutine run( ) calls via the subroutine repaint( ) the subroutine paint( ) which redraws the applet then stops (Thread.sleep( )) for one length of time equal to “time” milliseconds.

In the subroutine paint( ), I calculate the positions of all the reciprocal lattice nodes and I seek for those being on the sphere.

One modifies the animation speed by changing the value of the variable "time".

Cross-platform C++ GUI development using Qt

Barry R Smith,
School of Crystallography, Birkbeck, University of London, Malet St, London, WC1E 7HX, UK.
E-mail: b.smith@mail.cryst.bbk.ac.uk – WWW: http://people.cryst.bbk.ac.uk/~ubcg05s/

The GUI (graphical user interface) has taken up a central role in modern software development. GUIs can range from highly complex layouts, down to just a few buttons and menus controlling a simple application. I'm sure we've all seen badly designed GUIs, but done right they can vastly improve the user's experience. Whether you're planning to write the next killer application in your field, or perhaps just want to put a friendly face on top of a command-line driven program, which tool should you use for the job? Your decision might naturally be swayed by which programming languages you already know, or are prepared to learn. Part of the decision will also depend on which operating systems you want your software to run on. Platforms such as Microsoft Windows, Unix/Linux and Mac OS X do not share common system libraries for drawing to the screen, traditionally forcing programmers to maintain more than one set of source code or support just one platform. Learning any new language or graphics toolkit can require a serious investment of time and effort. In this article I will try to put the case for *Qt*, a widely used cross-platform C++ GUI toolkit. Qt is the flagship product of a Norwegian software company called Trolltech (http://www.trolltech.com/). Trolltech was founded in 1994, whilst Qt development dates back to 1992.

Nowadays there is a wide choice of software out there for the budding GUI designer — which is a *good* thing. For Windows-only development, Visual Basic or MFC have long been popular. However, like many programmers, I am keen on multi-platform development, including Linux and the various flavors of Unix (IRIX, Solaris, Tru64, AIX, and others). Under Unix, the traditional way of creating a GUI is via the Motif toolkit. However, rather than signing-up to any of these solutions, maybe your time is better spent working with a cross-platform toolkit?

Qt, wxWindows, Fltk and FOX — to name but four — are all cross-platform C++ GUI toolkits you might use (see http://www.atai.org/guitool/ for a better list). Recently I've done work using Fltk, and hence I do not use Qt exclusively. Since I am not a Java programmer myself, I will conveniently avoid the C++ versus Java debate. I'm not here to criticize other tools: I've seen great things done with Java, Python, Fltk *et al*, but for this article I will concentrate on the *pros* and *cons* of Qt itself, and why I enjoy using it. I will discuss pricing and licensing later, but as a side note I should stress the need to always read (and understand!) all software licenses relevant to your project, be they Qt or anything else.

Using Qt, you can support Windows, Unix/Linux and Mac OS X whilst maintaining a single set of C++ source code, and avoid platform-specific libraries such as MFC or Motif. When you compile your application, Qt uses the target platform's high-performance drawing functions; its classes are not inefficient 'wrappers' around other libraries' functions. Qt is a highly object-oriented (OO) library, encouraging good OO design in the programmer. It generates efficient code, with conservative use of memory. The end result looks and feels just like the user expects from their chosen operating system and desktop environment. This 'look and feel' is important — you don't want your application's fonts, colours, widget style or overall behavior to seem out of place on the user's desktop. Equally, in the commercial world you don't want to look less professional when compared to your competitor's product.

Even if you are targeting just one platform right now, its sensible to keep your options open rather than become locked into that platform, and locking out potential users or customers.

I think you will find the Qt API (application programming interface) to be amongst the best around. Actually, I'll stick my head on the block and say you won't find better. The Qt widget-set (i.e. set of classes) is huge, but I'll try to convince you that it is very easy to use.

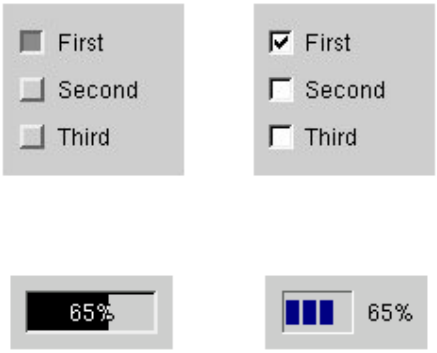


Figure 1: *Sample Qt widgets shown in Unix (left) and Windows (right). Top: a group of checkboxes (`QCheckBox`). Bottom: a progress bar (`QProgressBar`).*

Figure 1 shows two simple widgets and how they appear on typical Unix and Windows systems. You can probably guess the purpose of most Qt classes just from reading their names: common widgets include `QPushButton`, `QPopupMenu`, `QMainWindow`, `QCheckBox`, `QDialog`, `QProgressBar`, and so on. Most user-interface classes derive from a `QWidget` base class. Layout is important in any GUI, e.g. arranging widgets into rows, or defining which areas will expand when the user stretches a dialog box or window. Classes such as `QHBoxLayout` and `QGrid` are for this purpose. `QTable` goes beyond basic layout, providing a powerful spreadsheet-style grid of editable cells. Other specialized classes include `QCanvas`, which is ideal for highly optimized 2D display of sprites, lines and polygons, also providing collision detection and double-buffering. For 3D graphics, the industry-standard OpenGL can be used via the `QGLWidget` class, but you will need to understand the OpenGL API first since Qt does not provide much help here (perhaps disappointingly for those wanting a quick route to 3D). The display of editable text, with rich text formatting or HTML is commonly required in modern software. These are easily implemented using `QTextEdit` and `QTextBrowser` respectively. The latest version of Qt (3.1) brings some useful improvements, such as a new class for syntax highlighting.

Qt is not just about buttons and other graphical building-blocks. `QString` is Qt's own string class, with a rich feature set including internationalization, regular expression matching and all the functions you would expect from using strings from other libraries. In order to overcome cross-platform STL issues, Trolltech also decided to reimplement a number of other STL-style classes, such as the object containers `QMap`, `QPair` and `QValueList`. Network classes exist for programming client/server applications, i.e. using FTP or HTTP. Database classes let you integrate SQL searches, but this is an area I've yet to explore myself. XML is becoming widely accepted as a standard data format, and is suitably well-supported in Qt.

As you've probably noticed, 'wizards' seem to be turning up all over the place in software these days. Like most things, when done well they are very effective. Qt's `QWizard` class is excellent for creating these. Nowadays, users also expect a 'Drag and Drop' (DnD) interface, especially on Windows platforms, e.g. to load files or perform other operations. Qt has classes such as `QDragObject` to add this functionality. Learning to use these techniques can improve the usability of your software immensely.

Qt also takes care of other cross-platform issues that you might encounter. With `QDate` and `QFile` you won't need to worry about how the target platforms deal with dates, times, and file systems.

More advanced programmers will be interested in Qt's thread support, MDI (multi-document interface), plug-ins, and scripting possibilities. I've also not mentioned that Qt supports some embedded operating systems, such as those used in handheld Linux devices.

As a scientific programmer, perhaps the most obvious feature that seems lacking from Qt is purpose-built graph and chart classes. Sure, you can write your own or download third-party widget-sets such as *Qwt* (which is admittedly rather good), but it would be nice to see properly supported 2D and 3D graph classes, even if not part of the core Qt library (its important to keep GUI libraries from becoming too bloated with 'specialist' classes).

Many companies, large and small, are using Qt. You will find well-known names such as NASA listed as Qt users — see the TrollTech website for more case studies and articles. Probably the most well-known application written using Qt is KDE (http://www.kde.org/), the desktop environment that comes with many Linux distributions. Whether you use KDE or not, its an excellent example of what Qt can do, and testimony to the quality of it's classes. Since this is a crystallography newsletter, what better way to see Qt in action than to download CCDC's free crystal structure viewer, *Mercury* (http://www.ccdc.cam.ac.uk/prods/mercury/). It is available for Windows, Linux, Solaris and IRIX operating systems. Mercury makes use of the OpenGL module in Qt for it's high-quality molecular graphics display. I feel I should also point out, having worked at CCDC and seen the huge amount of effort that goes into such a project, that there is of course a lot of *non*-GUI related work required when creating a powerful scientific application such as Mercury.

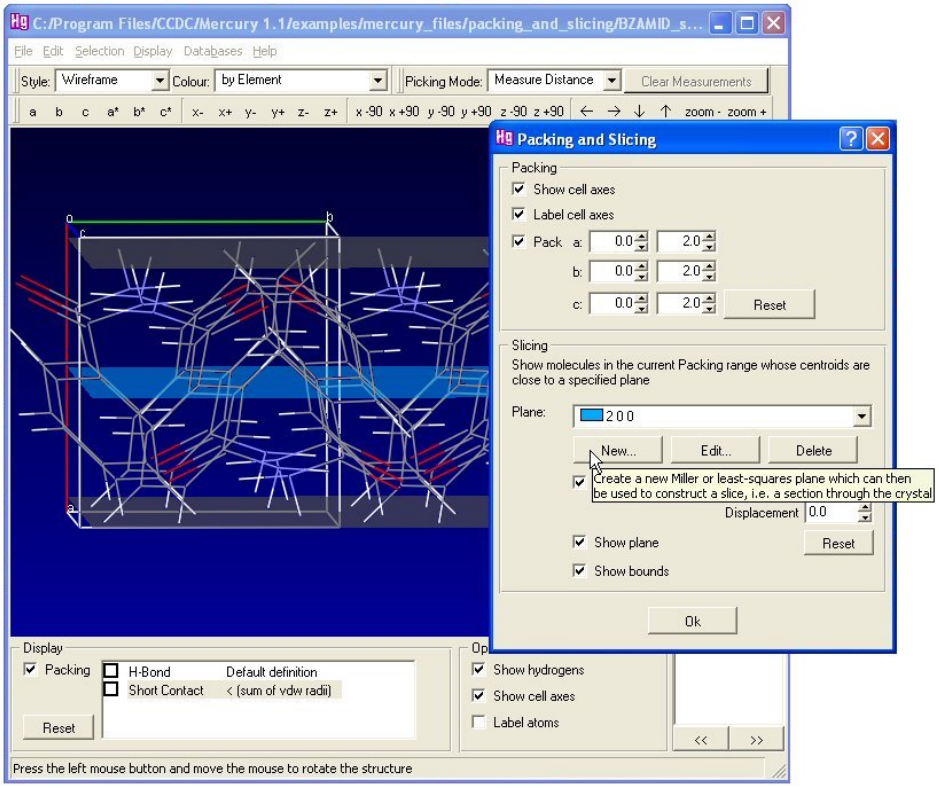


Figure 2: *Mercury screenshot (Windows version).*

If you think about it, a user-interface is really about letting the user control an application and be productive in what they want to do. Buttons get pressed, menu items get selected, text gets entered, *things happen*. Many widgets will be involved during this, and must react to events. Hence, the GUI is not a collection of static graphics, but an active *communicating* set of items. So, how does the programmer control this? Qt implements its own unique way, called 'Signals and Slots'. At this point I will introduce a test application to demonstrate some real C++/Qt code. As an exercise, I recently decided to write a simple protein structure viewer. The idea was to spend no more that a day on the

project, from start to finish. Figure 3 shows the result. The same code compiles on Windows (Developer Studio) and Linux (GCC) without modification, and as you can see, the application inherits each platform's native look and feel. I restricted the program to drawing the protein backbone only, hence the name *Bbone*.

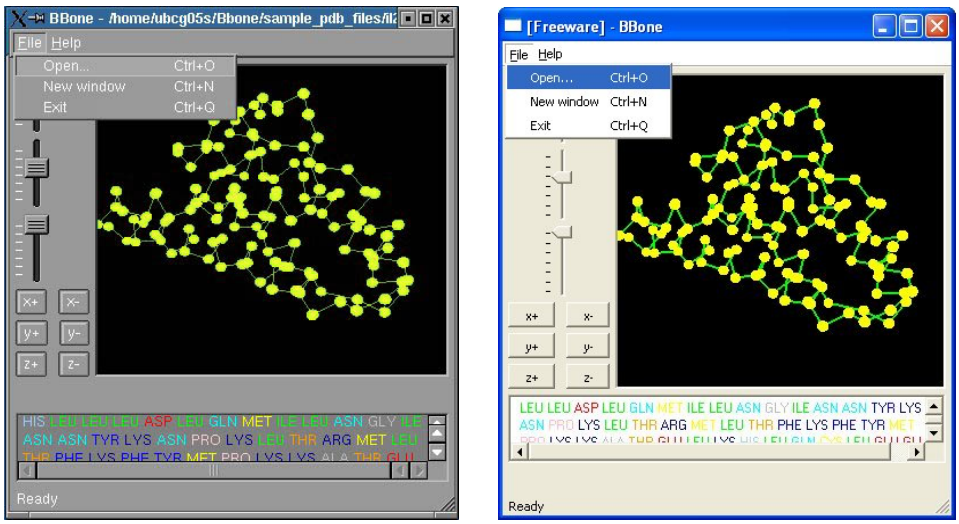


Figure 3: *Screenshots of the Linux (left) and Windows (right) versions of a simple protein visualiser application written using Qt.*

The simple GUI utilises `QButton` and `QSlider` widgets to let the user manipulate the molecule by rotation and translation. Returning to the idea of signals and slots, let's look at the code for implementing one of the 'sliders' in Bbone:

```
// Create our OpenGL molecule in a frame (implementation not shown here).
molgl = new GLBox( frame, "glbox");
// Create slider for x rotation.
QSlider* xsl = new QSlider ( 0, 360, 60, 0, QSlider::Vertical, this, "xsl" );
xsl->setTickmarks( QSlider::Left );
connect( xsl, SIGNAL( valueChanged(int) ), molgl, SLOT( setXRotation(int) ) );
```

First, lets concentrate on the instantiation of the `QSlider`. Widget constructor arguments vary, but usually require at least a pointer to a parent widget. This makes for simplified memory handling, since deletion of a parent widget automatically calls the destructor of any child widgets it knows about. The extra arguments in the `QSlider` constructor let us create a slider from 0 to 360 degrees, with 60 steps, and start it at zero. The `connect` statement then defines what we want the slider to do when it is activated by the user's mouse, i.e. the molecule must rotate around the x axis. The code takes the form of two object pointers, plus a signal from the first and a slot from the second, thus establishing the connection. Built-in classes already have many signals and slots available, such as the `clicked()` signal emitted by a `QPushButton`, or you can write your own. Slots are really just normal member functions, but with the added ability to 'listen' for signals.

A nice consequence of Trolltech's signal and slot design is that it keeps widgets as independent as possible. Signals need not be connected to any slots, or may be connected to many. Slots may listen for multiple signals. This makes for rapid GUI development, and easy maintenance of code. There is a small CPU overhead for this flexibility, but except in very specific situations it is not noticeable. One valid criticism leveled at the signals and slots approach is that it requires a preprocessing step before compiling. In the code snippets above, the keywords `connect`, `SIGNAL` and `SLOT` are not real C++, but instead a preprocessor spots these definitions and creates the appropriate code for them, somewhat behind the programmer's back. However, for the C++ purist this can feel like a rather non-standard way to solve the problem. For everyone else, it's a clear, clean, robust way to get your widgets talking to eachother.

Now we have our three sliders, this is a good time to look at how we can arrange them in a nice layout. If we want them to appear stacked one above the other, we can use `QVBoxLayout`, as shown in the code below.

```
// Arrange the sliders on top of each other
QVBoxLayout* vlayout = new QVBoxLayout( 5, "sliderbox");
vlayout->addWidget( xsl );
vlayout->addWidget( ysl );
vlayout->addWidget( zsl );
```

I'll briefly summarize the other parts of the program. The list of amino acids is done via a `QTextBrowser` widget, which will automatically show scrollbars if required. For the balls-and-sticks display I used OpenGL via Qt's `QGLWidget` class — admittedly not a trivial task. Adding 'File' and 'Help' menus and a `QStatusBar` finishes off the application. Of course, the program needs more work to be genuinely useful — the Mercury developers need not worry quite yet! Also, as I eluded to earlier, there is always plenty of non-GUI work to do. Reliably parsing the PDB file turned out to be rather tricky to write, but I could of course employ some third-party code to improve this.

Qt is not a completely free product. You probably guessed there had to be a catch somewhere. The licensing is a little complicated, but boils down to the following: Qt is free for developing open-source (GPL) projects under Unix and Linux. Closed-source, or internal company projects, require a license. Currently, Qt3 for Windows or Mac OS X is only available by purchasing a license. There are a few caveats to this. Firstly, academic users can obtain a 60% discount. For purely teaching purposes, free site licenses are also available. As described on Trolltech's website, staff at colleges and universities are finding that Qt is ideal for teaching the fundamentals of object-oriented design and C++ programming. At one stage Trolltech did release a free *non-commercial* version of Qt for Windows. This was great for the shareware community but unfortunately led to mis-use by commercial organisations, and hence Trolltech has not (at the time of writing) released Qt3 this way, although one can still use Qt2.3 under that license. It is difficult to criticise Trolltech too much for this pricing structure (although many shareware developers do) since academics get a hefty discount, and the commercial funding has in-part led to Qt being as good as it is, and assures Qt will be around in years to come. The latter point is rather relevant in the recent dot-com doom and gloom.

Personally, one of the biggest compliments I can give to Trolltech is for their documentation and support. The API documentation is amongst the best I've seen for any product, and is now like an old friend when I look it up. This is an important consideration when choosing a GUI toolkit. For relatively small projects you might not encounter major problems, but if committing significant time and effort to a piece of software (and potentially much longer in subsequent maintenance and support) there is a real chance you will hit against complex problems that need solving fast. Qt's documentation is extensive, from numerous examples and tutorials, to clear yet detailed descriptions of all classes and functions. Other topics, such as Signals and Slots or Drag and Drop are covered in individual articles. Another valuable source of reference is the Qt users' mailing list, which has searchable archives via the Trolltech website. The list is very active, and there's a good chance your problem is solved by going there. For paying customers, direct E-mail support is available, and on the few times I've had to use it I've received prompt and useful replies, often from the actual programmer responsible for maintaining the relevant piece of code (try that with Microsoft!).

Many GUI toolkits and software development environments include some kind of WYSIWYG application to visually create a user-interface. All versions of Qt come with *Designer* (naturally, also written in C++/Qt). Over the years this has developed into a powerful tool, allowing the programmer to drag and drop widgets onto a central form, quickly building and previewing your application. It also provides a simple way to connect signals and slots, by visually drawing lines between them. Even if you don't use Designer for your entire GUI, it is particularly useful for something like a 'Preferences' dialog box which can contain lots of laborious layout which probably won't be much fun to do manually. Designer uses XML ('.ui' files) to store it's layouts, and these are only converted (via another

preprocessor) to real C++ classes at compile-time. Since the classes get regenerated from the '.ui' files every time you compile, any custom functionality must be added to derived classes, but this works surprisingly well.

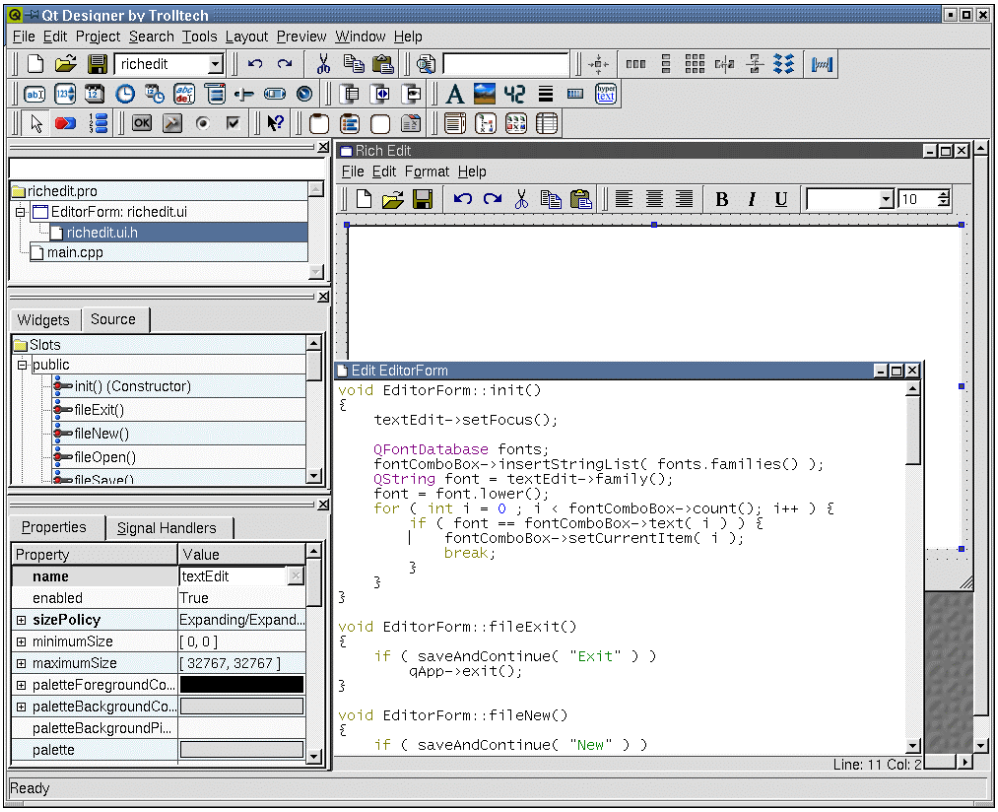


Figure 4: *Screenshot of Qt Designer, a visual GUI builder. It allows you to lay out widgets, connect signals and slots, and preview the results immediately. (image taken from the Qt website)*

I'll finish by mentioning a couple of other related tools. One particular job that can create headaches for even the best programmer is the production and maintenance of Makefiles (or the equivalent files required for other compilers). Luckily, Trolltech have created a separate program called *Qmake* (previously named *Tmake*) to solve this problem. Instead of maintaining Makefiles you keep a more generic project file, which Qmake then analyses to create the Makefile, taking care of any platform-specific libraries, paths or other details. Although Qmake was primarily designed for use with Qt it can in fact be used with any C++ project. In a professional software development environment, the use of source-control software such as CVS can be combined with Qmake to set up a fully automated build process.

The second tool I'd like to recommend is *Doxygen*, a free (GPL) program which creates documentation directly from your source code. It is not produced by Trolltech, but they use Doxygen to generate their Qt class documentation web pages. It works by extracting text comments from the source, and its a good idea to spend a few minutes learning the special formatting it looks for, so that you can keep your code well-documented from the outset. Current and future users of your code will thank you! As well as nicely-formatted HTML output, it can also generate Latex and Windows Help files. Another good feature is its ability to draw class hierarchies and simple UML relationships, all automatically. See http://www.doxygen.org/ for more information.

So, should you be using Qt? If you want to learn C++ and OO methods, Qt is a great place to start. For small projects, requiring relatively simple user interfaces, one can argue that almost any of the previously listed toolkits or programming languages will produce satisfactory results. Comparing the pricing structure of Qt to some of the free options may understandably lead you to take another route if you're using Windows rather than Linux. Recently I made such a decision for a small project I was working on (the Fltk-based project mentioned in the introduction). For commercial software companies, it's likely

that the added support that comes with a license easily outweighs the cost issue. I think the strength of Qt lies in larger application development, where you require a fully-fledged GUI, powerful underlying features, and robust, well-documented classes. In that case, you can't go wrong with Qt.

Old Wine in New Bottles: Creating Graphical User Interfaces for FORTRAN programs *via* Tcl/Tk

Brian H. Toby,
Crystallography Team, NIST Center for Neutron Research, Stop 8562 , National Institute of Standards and Technology, Gaithersburg, MD 20899-8562, USA. Tel: 301-975-4297; Fax: 301-921-9847; E-mail: Brian.Toby@NIST.gov; WWW: http://www.ncnr.nist.gov/xtal

About a decade ago, I started looking for a *good* way to create graphical user interfaces (GUI) for the FORTRAN programs that I was using on a regular basis and for those I was writing. To me "good" means: (1) Easy to learn. Learning new skills is important, but time is always scarce; (2) cross platform. I want the same software to work on my desktop SGI, my Linux home computer and my coworker's Windows box; (3) cheap. I want to encourage other people to collaborate on software -- if they must purchase something to do this, they probably won't.

I spent many years looking.

The solution that eventually adopted is a scripting language, called Tcl, and a GUI tool, called Tk. Some people feel that other scripting languages, notably Perl and Python might be better choices than Tcl. Those folks may be right. What I can say is that Tcl was very easy for me to learn and it turns out to be very powerful, so I have never needed anything else. Also, while Tk is available for Perl and Python, it was originally created by the author of Tcl, John Ousterhout, so it comes as no surprise that Tk works best with Tcl. For my purposes, I'll consider them a single language, Tcl/Tk. Java gets lots of press, too, and can do many of the same things. At the time when I was looking, platform-independent GUI tools for Java were not available and Sun and Microsoft were fighting for control of the turf. The former is no longer true, but from what I can see, there are still many platform-dependency issues Java. True, I do need to tweak my Tcl/Tk code a bit to get around some of the limitations of Windows. However, to take one project (EXPGUI) as an example, in ≈37,000 lines of code only a few hundred lines are either Windows or Unix-specific. Those sections of the code check what operating system is being used and react accordingly. Thus, *the exact same source code is used on all platforms.*

In this article, I will tell you a little bit about Tcl/Tk. I'll also give an overview of some of the projects that I have tackled using Tcl/Tk. Lest I forget to mention: Tcl/Tk and, for that matter, all my software is available on the Internet, free, and with source code.

Tcl/Tk

Perhaps the easiest way to give you a feel for Tcl/Tk is to give you an example of a very simple program. Suppose you have Tcl/Tk loaded on your computer (and there is a fair chance you do), you can start the command interpreter by typing "wish" in Unix. In Windows, you might have to find the WISH82.EXE file in a folder like c:\GSAS\TCL832\BIN and then click on it. A window for commands and a window for graphics will then be displayed. Into the command window, type the following commands, one line at a time:

```
set w .b
button $w -text "Hello World" -command {puts "Hello World"}
pack $w
```

These three lines of code create a GUI with a working button and invoke three commands, "set", "button" and "pack". The first line defines a variable, w, which contains a character string ".b". Note, I did not need to declare the variable. The second line creates a button and the third line places the button on the screen. In fact, I could have written this program in a single line. Note, also that no compilation was needed. Tcl interpreted the commands as they were typed. We can change things as we go, too. For example, now type:

```
$w config -command {tk_dialog .msg msg Hello "" "" OK}
```

The button action now changes to display a new window when pressed.

One of the major criticisms of interpreted languages is speed. A program that is compiled will run faster than one that is interpreted. I have two comments on this, though. Speed is not always very important in a GUI. When you push a button something should happen soon. Nevertheless, if soon is 0.00005 seconds or 0.05 seconds, you will be hard pressed to tell. Modern computers can do an awful lot in 1/20 of a second. Second, Tcl interprets commands once and saves the byte-compiled code, so slow is not always *that* slow.

Tcl is designed to have commands added to the language. So, you can define your own Tcl command, say, `Solve_Structure_and_Publish` either as a procedure composed of other Tcl/Tk commands or, for better speed and power, by adding compiled C (or FORTRAN) code into the language. This means that rather than adding a macro language to your program, you can add your program to Tcl/Tk and in the process get a full-featured GUI and macro language. Many commercial vendors do this, but to be honest, it can take some work and takes effort to support on multiple platforms, so I now avoid it. I'll discuss that more below. However, many people have written collections of commands, called packages, which add capabilities to Tcl/Tk that are not present in the native language. One example that I use extensively for scientific graphics is called BLT. Another is called LA, for linear algebra.

Scientific Graphics

One might think that scientific graphics (xy plots, etc.) should be easy to do in a GUI environment, but that has not been my experience. The scientific plotting package I use, BLT, allows creates and manipulates graphs (for example, zooming in, change colors,...). BLT is a blessing and a curse. It is keeps getting more features, which sometimes break old code. It creates really mediocre hard-copy output. It makes Tcl/Tk installation that much more complex. Until the advent of OSX, BLT was not supported on the Mac. However, it is easy to use and offers lots of power.

Some Tcl/Tk Web Links

FAQ links: http://www.purl.org/NET/Tcl-FAQ/
yet more links: http://www.cetus-links.org/oo_tcl_tk.html
misc Tcl/Tk programming ideas: http://aspn.activestate.com/ASPN/Cookbook/Tcl
Wiki (programming tips, etc): http://wiki.tcl.tk/0 and http://wiki.tcl.tk/969
Newsgroup comp.lang.tcl info: http://www.purl.org/net/tcl-welcome

Tcl/Tk Successes: A Personal Gallery

EXPGUI. I imagine that my front-end to the GSAS crystallographic package, EXPGUI, is the most widely used software package that I have written. It uses Tcl/Tk to edit the "experiment" (.EXP) file that GSAS uses for input. EXPGUI can also be used to launch all the other GSAS programs, for example the least-squares minimizer, GENLES. With time, I have added more scientific graphics to the program.

Most of EXPGUI is written in Tcl/Tk. An exception is the sections of the program that adds new phases or histograms to a refinement. This involves many complex steps; it was much easier to borrow FORTRAN code from the GSAS EXPEDT program than try to rewrite it in Tcl/Tk. So, when the user

presses the "add phase" or "add histogram" program, (or, for that matter, plots the results), an external FORTRAN program is called. Its fast, so the user probably does not know that. The major stumbling block in getting EXPGUI to work cross-platform is that Tcl/Tk has trouble starting Windows "console" applications (where a program runs in a "DOS window") under Windows-95/98/ME. For that platform, I use a package called WINEXEC to work around this. (At some point, I'll switch to a newer package by D. Gravereaux called winutils.) All the tricks used in EXPGUI can be figured out by reading the source code. [See http://www.ncnr.nist.gov/programs/crystallography/software/expgui/expgui_intro.html]

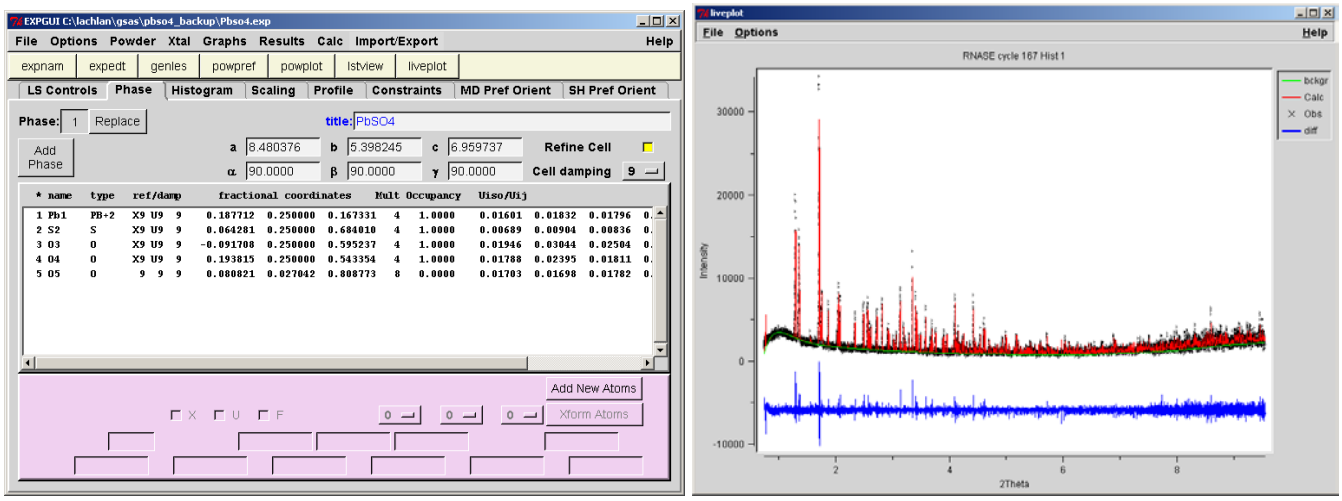


Fig 1 (a, b): *Screen image examples of EPGUI*

CMPR. The name for CMPR was taken from a prehistoric VAX program that I wrote to plot powder diffraction data and do peak fitting. With time, I have added many more features. I consider CMPR to be sort of a "Swiss Army Knife" where I add a new blade (capability) as I have time and need. One of the unique features of CMPR is peak position visualizer that superimposes reflection positions on top of diffraction data. The neat part is that the GUI has "sliders" for each appropriate unit cell parameter. Adjusting the slider via the mouse causes the reflection positions to move seemingly instantaneously. It makes a nice teaching tool for showing how changing unit cell symmetry splits classes of reflections. It can also be used for pattern indexing and for looking at space group extinctions. From a programming perspective, the way it works is a bit surprising. When I wrote the code, I got to the point where I had a set of unit cell parameters, a wavelength, and a list of *hkl* values. My plan was to imbed a new command into Tcl that would take that input and spit out a list of 2θ, Q values, or whatever. However, for testing, my Tcl code wrote this input to a file, invoked a short FORTRAN program which would do the number-crunching and write the peak positions back to a file. The Tcl/Tk script then read the peak positions back by Tcl/Tk. This test code, though clumsy, gave response time that was already sufficiently fast, even with the 90 MHz PC I was then using, that I never bothered to add the new command. I would not have believed it, but this means that *Tcl/Tk can fork external FORTRAN programs quickly enough to do primitive computer animation* -- even in Windows! Amazing.
[See http://www.ncnr.nist.gov/programs/crystallography/software/cmpr/cmpr.html]

LOGIC. The LOGIC program is a clone of a VAX application for Boolean searching of the ICDD-JCPDS database. It was the project that I used to learn Tcl/Tk. For this project, I took all of the search capabilities from the original FORTRAN code, rewrote the VAX-specific low-level subroutines in C. Then wrote C wrappers around the upper-level FORTRAN code. Getting all the naming and argument passing right so that C can call FORTRAN and vice versa with different compilers for the SGI, Linux and Windows was a pain, but it works (now one can use gcc for Windows). I then wrote a Tk GUI. Voila, a platform-independent code for accessing the JCPDS-ICDD database. Later, having all the database access routines already written in Tcl/Tk, it was pretty easy to add ICDD entry plotting into EXPGUI and CMPR. LOGIC was written in the mid-90's and I am not maintaining it actively at present, but I do hear from people who are using it. [See http://www.ncnr.nist.gov/programs/crystallography/software/logic.html] Also in the mid-90's, to show the power of intranet-based searching of the database, I created a web

interface by creating a Tcl cgi script that used the guts from LOGIC. I had hoped to slap it together with the help of my Tcl/Tk guru, Przemek Klosowski, over a weekend, but it took a few weeks before we resolved a weird bug. Still, not bad for a first attempt at a web application.
[See ftp://www.ncnr.nist.gov/pub/cryst/powdersuite/icdd_search.tar.gz].

NCNR Crystal Data Searching. Use of FORTRAN code embedded into Tcl was chosen for this project, so that I could open the database once, and string searches together in a sequential fashion, but it made LOGIC hard to maintain and distribute. For my next web-based search program, I decided to try a different approach, where each search would: i) start a FORTRAN program; ii) load the previous search results; iii) perform a new search; and iv) write the result to file. Yes, this approach is wasteful compared to the one used in LOGIC. It probably squanders hundreds, if not thousands of microseconds each use, but the code is so much easier to write and maintain. Besides, for a web application, the search programs really does need to be started each time the user presses "Search" on her/his browser screen.

For reasons unclear to me, NIST is no longer selling the Crystal Data database in the format I used for this project, but the web interface is in use within my center and the code can be found on-line. I think this is a nice example for how to set up a web-based Boolean search engine. [See ftp://www.ncnr.nist.gov/pub/cryst/powdersuite/crystaldata.html].

CIF Applications. Most recently, I have been at work on a number of programs for creating, viewing and editing crystallographic information files (CIFs). To give an idea of how powerful Tcl/Tk is for lexical processing, a complete Tcl/Tk parser and browser took little more than 2,000 lines of code. Another program built on this parser reads and plots powder diffraction patterns read from CIF. As of this writing, these projects have not made it to the web, but look for a link on http://www.ncnr.nist.gov/xtal soon.

In conclusion, Tcl/Tk is a pretty nifty way to provide modern interfaces with valuable, but sometimes hard-to-use older codes. It is also easy enough to learn, such that several people have contributed code to CMPR and EXPGUI.

Fitting Equations of State

Ross J. Angel[1] and Ian G. Wood[2]
(1) Virgina Tech Crystallography Laboratory, Virginia Tech, Blacksburg, VA 24060, USA. E-mail: rangel@vt.edu ; http://www.crystal.vt.edu/ and (2) Dept. Earth Sciences, University College London, Gower St., London, WC1E 6BT, England.

Introduction

The rapid developments in high-pressure diffraction techniques over the past decade have made them available as a routine tool to many researchers. To the newcomer in high-pressure research, the topic of Equations of State (EoS), or the relationship between pressure and volume of a solid, appears arcane and complicated by many differing formulations, many errors of derivation in the literature and, to a crystallographer's view, the frequent use of statistically-dubious fitting procedures. In this brief introduction we outline the issues involved in determining the parameters of isothermal EoS from P-V data and describe the essential elements of a freely available software package, EosFit v5.2 that implements the necessary procedures. Anderson (1995) provides an in-depth analysis of EoS theory, and Fei (1995) outlines the ways in which isothermal EoS can be expanded to describe P-V-T data. The issues involved in fitting EoS are covered in much more detail in Angel (2001), and in the manual accompanying the EosFit v5.2 code, available from http://www.crystal.vt.edu/crystal/software/.

EoS formulations

There is no fundamental thermodynamically correct formulation for the EoS of a solid, so many have been developed based upon various assumptions about the behaviour of dense solids. There is no reason to suppose any are correct for a given solid, and this must always be remembered when one is attempting to determine EoS parameters through the fitting of P-V or P-V-T data. The most commonly used formulations for EoS are the Birch-Murnaghan, the Vinet and the natural strain (see reviews by Anderson, 1995 and Angel, 2001). The Murnaghan EoS is popular because of its simple functional form, but it does not accurately describe the compression of solids beyond about 10% and should not therefore be used beyond this regime. The issues that arise in fitting EoS can be illustrated by just one of these, the Birch-Murnaghan EoS (Birch, 1947):

$$P = 3K_0 f_E \left(1+2f_E\right)^{5/2}\left(1+\frac{3}{2}\left(K_0'-4\right)f_E+\frac{3}{2}\left(K_0K_0''+\left(K_0'-4\right)\left(K_0'-3\right)+\frac{35}{9}\right)f_E^2\right)$$

in which $f_E = \left[\left(V_0/V\right)^{2/3}-1\right]/2$. The parameters of the EoS are therefore the bulk modulus $K_0 = -V_0\left(\partial P/\partial V\right)_{P=0}$ and its pressure derivatives, $K_0' = \left(\partial K/\partial P\right)_{P=0}$ and $K_0'' = \left(\partial^2 K/\partial P^2\right)_{P=0}$, all evaluated at zero pressure, and the zero-pressure volume, V_0. This equation is based upon a Taylor expansion of the free energy in terms of f_E which can be truncated at any power of f_E. Such truncation implies non-zero values for some of the higher-order parameters. Thus, if this EoS is truncated at second-order in the energy then the coefficient of f_E must be identical to zero, which requires that K_0' has the fixed value of 4 (higher-order terms are ignored). The third-order truncation, in which the coefficient of f_E^2 is set to zero yields a three-parameter EoS (with V_0, K_0 and K_0') with an implied value of $K_0'' = \frac{-1}{K_0}\left(\left(3-K_0'\right)\left(4-K_0'\right)+\frac{35}{9}\right)$. Other EoS yield different implied values for the parameters upon truncation (Anderson, 1995).

Thus a least-squares fit of P-V data to determine the EoS parameters of a solid must proceed in a step-wise fashion:

1. Refine V_0 and K_0 (which scale the *entire* equation of state) with K_0' fixed to the implied value of 4.
2. Refine V_0, K_0 and K_0', and test whether the quality of fit as measured by χ_w^2 has improved and whether K_0' deviates significantly from the implied value. If it does not, then refinement should be terminated after step 1.
3. If the value of K_0' was significantly different from the implied value, expand to the next parameter, K_0'', and determine whether it deviates from its implied value. If it does not, then refinement should be terminated after step 2.

These assessments are complicated by the high correlation (often >90%) between the EoS parameters in the least-squares process. This leads some authors into the temptation to fix some parameters (such as V_0) while refining higher-order terms thereby biasing the results to the authors' pre-conceived misconceptions. They forget that V_0 is the zero-pressure volume and is not measurable. The room pressure volume is only a close, but biased, estimate of V_0. In addition it is not absolutely known but has an associated experimental uncertainty that must be included in a least-squares refinement procedure. This fixing of V_0 is often hidden in an f_E-F_E fit in which the EoS is re-written in terms of $F_E = P/3f_E\left(1+2f_E\right)^{5/2}$ which then reduces it to a simple polynomial in f_E. But the calculation of both f_E and F requires knowledge of V_0, so this procedure should never be used to determine EoS parameters. The only statistically valid approach is to refine the EoS parameters directly to a properly weighted dataset.

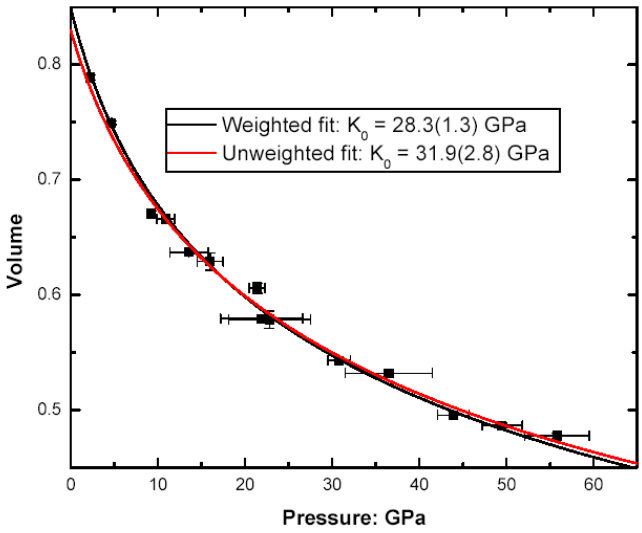


Fig 1: *An example of the effect of weighting data in a fit of the EoS. Pressure-volume data for the B2 phase of KCl from Campbell and Heinz (1991). The black line is a weighted fit to the data that yields a larger V_0 and a smaller K_0 than the unweighted fit shown by the red line.*

Implementation

Least-squares fitting of EoS is implemented as part of the EosFit v5.2 program package (Angel, 2001). The least-squares part of the program is derived from an original program written in Basic by Mike Glazer in the late 1970's and later converted to Fortran by Jenny Piper and Ian Wood. The attraction of this code was that it was written to use external subroutines to define both the value of the function and the necessary derivatives. Although extensively modified for use in the EosFit program, the structure of the original code is retained.

The form of most EoS means that it is simpler to consider volume (and temperature if necessary) as the independent variable in EosFit, with pressure as the dependent variable. The function minimised by least-squares is therefore $\sum_i^n w_i \left(P_{obs,i} - EoS\left(V_{obs,i}, T_{obs,i}\right)\right)^2$ in which *EoS(V,T)* is the pressure calculated for a given V and T from the current set of EoS parameters. Because experimental determinations of EoS data invariably include uncertainties in the volume and temperature measurements the weight assigned to each data point in the least-squares is calculated by the effective variance method (Orear 1982) from all of the available experimental uncertainty estimates:

$$w^{-1} = \sigma^2 = \sigma_p^2 + \sigma_V^2 \cdot \left[\left(\frac{\partial P}{\partial V}\right)_T\right]^2 + \sigma_T^2 \left[\left(\frac{\partial P}{\partial T}\right)_V\right]^2$$

There are a number of technical points involved in the derivation of this relationship that may affect the estimates of the values of both the values of the EoS parameters and their uncertainties obtained by least-squares. First, it is assumed that the partial derivatives are constant over the pressure interval between P and $P+\delta P$. This is reasonable provided δP is small compared to the bulk modulus of the material. Second, although the values of σ are correct, the derivatives are incorrectly calculated at the experimentally observed values of V and T rather than at the values estimated by the least-squares fit of the EoS. This leads to a slight over-estimate of the parameter uncertainties (Lybanon 1984, Reed 1992), although the effect is usually insignificant for a slowly varying function such as an EoS with small experimental uncertainties in the data. It is certainly preferable to using unit weights or ignoring the experimental uncertainties in the independent variable, volume. Through the use of thermodynamic

identities (e.g. Anderson 1995) the relationship reduces to: $w^{-1} = \sigma^2 = \sigma_p^2 + \sigma_V^2 \cdot \left(\frac{K}{V}\right)^2 + \sigma_T^2 (\alpha K)^2$. The weights are dependent upon the values of the EoS parameters, and are therefore updated every least-squares cycle.

In EoSFit derivatives are calculated analytically from the current values of the EoS parameters because these appear to lead to more stable least-squares behaviour than derivatives calculated numerically. The normal equations matrix is then solved by the Choleski method. All variables are declared real*8 to avoid round-off errors. Each cycle of least-squares is concluded with the recalculation of the implied values in the EoS. Least-squares refinement is terminated when the sum $\sum |shift/esd|$ for the refined parameters is less than 0.00001. If the value of χ_w^2 from the refinement is greater than unity the variance-covariance matrix is multiplied by χ_w^2 prior to the extraction and reporting of the parameter esd's. As noted above, the correlation between the refined parameters of EoS is quite severe and, in comparing parameter values with other determinations, the covariance must always be considered. EosFit provides the entire variance-covariance matrix, and it is intended to add the calculation of confidence ellipses when time permits.

There are a number of subsidiary programs built into EosFit v5.2 to perform calculations related to EoS. These include calculating volume given the EoS parameters and a pressure and temperature. With the exception of the Murnaghan EoS, the equations relating P, T and V cannot be easily inverted to give V as a function of T and P. Therefore such calculations are performed by a numerical search in which the volume is adjusted until the predicted pressure calculated analytically from the EoS parameters matches the requested pressure to within 0.001%. This level of precision is a somewhat arbitrary choice but is sufficient to prevent any significant rounding errors.

The variation of individual lattice parameters with pressure and temperature is also handled by EosFit. The program substitutes the cube of the lattice parameter for the volume in the equations for the EoS and calculations then proceed as before. The value of "*linear*- K_0" obtained from fitting the isothermal equation in this way is related to the zero-pressure linear compressibility β_0 of the axis by $-1/3K_0 = \beta_0 = l_0^{-1}(\partial l/\partial P)_{P=0}$ in which l_0 is the length of the unit-cell axis at zero pressure. Such an approach, while ad-hoc in non-cubic materials, provides values of β_0 in good agreement with those derived from experimental measurements of the elastic stiffness tensor.

References

Anderson OL (1995) Equations of state of solids for geophysics and ceramic science. Oxford University Press, Oxford.

Angel RJ, (2001) Equations of State. In Hazen, R.M., Downs, R.T. (Eds.), High-pressure, high-temperature crystal chemistry. Reviews in Mineralogy and Geochemistry, 41, 35-60.

Birch F (1947) Finite elastic strain of cubic crystals. Phys Rev 71:809-824

Campbell AJ, Heinz DL (1991) Compression of KCl in the B2 structure to 56 GPa. J Phys Chem Solids 52:495-499

Fei Y (1995) Thermal expansion. In: Ahrens TJ (ed) Mineral physics and crystallography, A handbook of physical constants. AGU, Washington DC.

Lybanon M (1984) A better least-squares method when both variables have uncertainties. Am J Phys 52:22-26

Orear J (1982) Least squares when both variables have uncertainties. Am J Phys 50:912-916

Reed BC (1992) Linear least-squares fits with errors in both coordinates. II: Comments on parameter variances. Am J Phys 60:59-62

The Threat of Patents on Crystallographic Algorithms and Software

Vincent Favre-Nicolin
CEA Grenoble, DRFMC/SP2M/Nano-structures et Rayonnement Synchrotron, 17 rue des Martyrs 38054 Grenoble Cedex 9, 38054 Grenoble Cedex 9, France.
E-mail: vincefn@users.sourceforge.net – WWW: http://objcryst.sourceforge.net/

As of today crystal structures can be determined with relatively limited costs once the diffraction data has been collected: most software is available for free, and even better many programs are open-source, so that you can modify the algorithm to suit your needs.

Much more important, there is an almost absolute freedom in algorithm development. Any new or improved algorithm can be developed, published and distributed freely for using or testing, with no other demand than acknowledging the works of your peers.

This almost ideal situation (in terms of research freedom) could disappear soon due to the extension of patents to algorithms and software. The aim of this article is to present a short introduction to patents, and show how software patents have the power to shut down a large number of open-source software, and could lead to a dramatic decrease in the availability of new algorithms.

I Patents

I.a The good: encouraging publication of knowledge

A common misunderstanding is that patents exist to protect inventors from anyone stealing their discoveries without a legitimate retribution. In the days before patents, inventors (individuals or companies) had a simple way to protect their inventive craft: keeping it secret, so that no-one could copy it. And often the new technique would be lost after the death of the inventors. Patents were introduced to stop this loss of knowledge, by establishing a contract: the inventor would publish his discovery in detail, and in return all people in society would have to pay him for the right to use his inventions for a given time after his invention was made public. So *the original and most important aim of patents is to encourage scientific and technological discoveries and their open distribution* (protecting inventors is just a means, not a goal).

Patents are vital in today's world: the most obvious sector in which they are useful is the pharmaceutical industry: after spending 10-15 years developing a new drug, involving many different scientists, it is clearly legitimate that the corporation get the exclusive right to use or license (allowing other corporations to produce the drug) this discovery for the following 20 years, if only to pay back for the years of research. Compared to the workload required for the discovery (dozens of "man-years" of work), 20 years is relatively small. And of course, after 20 years the knowledge is public domain, so all can benefit from it.

Now what is an invention, worthy of being patented ? In the USA, it must be "new, non obvious and useful" and in Europe "new, inventive and susceptible of industrial application", both very broad definitions[1]. In other words, there is -in practice- little or no criterion on the complexity, cleverness, and development time corresponding to the invention.

1My apologies to citizens of other countries, I have browsed through patent information for some of them, but it is beyond this article to detail subtle differences between all patent systems. There is, however, no fundamental difference between all these patent legislations.

And this has led to strange situations with patents: indeed, a drug discovery is protected just as much as (to give a famous example) US Patent #6,368,227, which is a new "Method of swinging on a swing" [2], sideways swinging to be more technical...

Of course the above (real!) example is fun and *probably* harmless, even if you'd better tell your kids to use their swing only in the normal way for the next 20 years, lest a lawyer comes by and sues for patent infringement.

I.b The bad: upstream vs. downstream patents

The vagueness of the patent system, leads to a increasing number of granted patents on relatively "small" or partial inventions. So instead of just granting patents on complete discoveries (directly susceptible of industrial application), patents can be deposited on "upstream" inventions, which are new but cannot be used *as such*.

Examples can be found (again) in the pharmaceutical industry[3]. Instead of patenting a finalized drug, individual molecules/proteins are regularly patented (upstream patents). The trouble with the abundance of these upstream patents is that a disease with complex treatments (e.g., cancer) require a combination of a number of these molecules, and very often it is *not even be possible to do research on a combination of these molecules*, because some holder of an "upstream" patent will either refuse or ask unreasonable royalties. So the excess in the patenting is not simply affecting the marketing of new drugs (as should be), but also their development, leading to a result exactly opposite to the original spirit of the patent system.

Such effects of current patent systems can be seen in various industries, where corporations are applying for many upstream patents. These can be used to gain a monopoly on a broad family of methods, but very often they are just "defensive patents", i.e. only to be used if sued by another corporation, resulting in a "cold war" of patents, where each corporation holds patents useful to others, so that none dare sue. And as it is extremely costly to apply for a patent, and much more to defend it, small companies can be easy preys for larger ones.

I.c The ugly: trivial patents and parasites

Worse than upstream patents are those that are altogether obvious and yet not screened by patent offices. This lack of screening is due to the evaluation process, which cannot be done by specialists in the field (peer review), due to the secrecy necessary with patents. Not being an expert in the field, the examiner has little chance to be aware of any "prior art" not mentioned in the application, or of its obvious nature.

Blooming on to of these trivial and often very broad patents are so-called "patent parasites", small companies who live by buying and holding trivial (and therefore upstream) patents and suing large corporations for the right to use them. As these companies only live by holding patents (and not using them), the sued corporations cannot use any of their "defensive" patents. Most companies will simply give in and pay, as winning a trial would cost them significantly more than agreeing to the licensing terms.

A good example can be found on http://youmaybenext.com/ : a company (Pangea Intellectual Properties) holds patents on trivial methods such as "using graphical and textual information on a video screen for purposes of making a sale", which applies to about every e-commerce web site.

Another example is a patent held by Forgent, which claims that a 16-year-old patent applies to jpeg compression[4], despite the fact that it was developed as an open standard. Companies like Sony have

2See (for example) http://slashdot.org/articles/02/04/15/2324253.shtml , and the links within
3http://www.thenewrepublic.com/docprint.mhtml?i=20021007&s=thompson100702
4http://www.extremetech.com/article2/0,3973,389261,00.asp

preferred to pay rather than engage in long and costly trials, yielding more than $20 million in revenues for Forgent.

II Software patents

II.a Background

Software has been relatively less affected by patents, mostly because the European patent convention (the Munich Convention[5]) explicitly forbids "discoveries, scientific theories and mathematical methods" and "programs for computers". However this is rapidly changing with a strong lobbying from large corporations to legalize the use of patents in Europe as well: the European Patent Office is already granting patents on software in spite of the Munich Convention, and there is a European Directive being worked on by the European Commission. See http://swpat.ffii.org/ and links therein for more information.

II.b Patentability of software- the weight of trivial inventions

The patentability of software is largely debated for several reasons:

- the 20-year term of patents is grossly unadapted to the computing world, patents would only add inertia.

- most of them are seen as invalid, either because there exists "prior art" (someone else published or used the same technique before the patent application), or because it represents a "trivial" method[6]

- the "research" cost of software methods is vastly smaller than for industrial-grade techniques. (anyone can be inventive with a 1000 €($) computer). Most of the costs involved in software are in the writing, debugging, testing the interaction with other parts of the application/operating system, but rarely in a research of new software methods or algorithms.

- even without worldwide patentability of software methods, the computing world has been thriving for the last 20 years, contributing substantially to the world's economical growth. What could patents improve ?

- copyright laws (see II.c) already protect software

More fundamentally, there is no indication that software patents have encouraged technological progress in any form, which is the *raison d'être* of patents. The opposite effect could be dramatically true, e.g. British Telecom holds a patent which -BT claimed- covered the hyperlink (the cornerstone of the World Wide Web)[7]. Should this claim be legitimate (it has finally been outruled last summer), and have been enforced -say- ten years ago, the World Wide Web would never have developed as remarkably as it did, slowing down the technological revolution that is the widespread use of the Internet.

Another famous software patent is held by Amazon who patented[8] the idea of "buying using a single click" on a web site, and is now suing -successfully- companies who also use this obvious idea, hindering the development of e-commerce.

So software patents are mostly effective to create a monopoly for the patent holder, locking technological development rather than encouraging it.

---

5 http://www.european-patent-office.org/legal/epc/e/ma1.html

6 Numerous examples can be found the internet. See for example http://lpf.ai.mit.edu/Patents/, the method of using "exclusive or" to display graphical cursors on a computer screeen..

7 e.g. see http://www.wired.com/news/business/0,1367,50356,00.html

8 http://www.noamazon.com/

Of course there *are* a very few algorithms which require long research and careful tuning before being validated (e.g. video encoding), but these methods are still outdated far before the 20-year term of patents, and are too few to legitimate all other patents.

II.c Copyright protection

Another reason to exclude software from being patentable is that copyright laws already provide a fair protection for authors, by outlawing any unlicensed copying of the end product (the complete program).

II.d Opposing patents

If most software patents are illegitimate, why not fight them before a court ? That seems like the legitimate way of solving all issues, but patent litigation costs far too much for that to be practical, especially if you intend to go against a large corporation with an experienced legal department. Even "challenging" a patent (which is done with the patent office and not before a court), requires a significant amount of money, if only to hire a specialized attorney (unless *you* can talk legalese). This is hardly possible for small companies or individuals, which are the ones who would need the most to be defended against abusive patents.

II.e Open-Source / Free Software[9]

Open-Source and Free Software have played an increasing role in the computing world since the late 1980's, and now allowing complete operating systems with a full suite of applications for the end users. Unfortunately, this kind of software is the easiest prey to software patents because:

- it does not use software to make profit[10] since it is available for free, and can be modified and redistributed for free as well. Therefore not only do free software companies have little money (to use for legal defense), but they are not interested in getting "defensive" patents themselves, as it is inherently against their principles.

- many OSS/Free software developers are individuals programming on their spare time, and have neither the money nor the time to defend themselves against even the most trivial patents

In other words, if software patents are made legal worldwide, large corporations will have the power to shut down a large part of free software, by enforcing patents so trivial that they can affect a wide range of applications. *It would be a striking paradox that the patent system, which was set up to ensure the spread of knowledge, could be used to annihilate the efforts of hundreds of thousands of individuals who have contributed, by their selfless efforts, to open computing knowledge*.

So far there has been little or no patent litigation against OSS/Free software, companies like HP, IBM, Thomson having a policy of not prosecuting against open-source developers. But this is a purely *unofficial* policy, without any formal assurance that this will continue.

III Software patents and Crystallography

II.a General patents

Our field is already subject to a number of patents. Some are general (upstream) patents not directly related to crystallography:

9see http://perens.com/Articles/Patents.html
10Open-source / Free software companies can be profitable, but only by selling "services" associated to the software.

- the "**Marching Cubes**" algorithm is quite popular: if you have ever used a system which *displays electronic density in 3D using wireframes*, chances are that your software uses that algorithm and is infringing on US Patent 4,710,876 "*System and Method for the Display of Surface Structures Contained Within the Interior Region of a Solid Body*" and 4,885,688, "*Minimization of directed points generated in three dimensional dividing cubes method*".

- Of course structure determination is hungry with Fourier **transforms**, and although the initial FFT algorithm is not patented[11], there are hundreds of patents related to FFTs...

- Other "general" patents are about molecular representation, such as US Patent 5,249,137 [Xerox Corp.], whose abstract is: "*A **computer-aided chemical illustration system** is disclosed. Techniques provided include: 1) efficient drawing of bonds; 2) drawing different bond types during a single mode; 3) determining bisect angles for bonds; 4) labeling atoms on the fly; 5) automatic alignment of atom labels; 6) custom alignment of atom labels; 7) changing the type, style, or orientation of an object while it is being drawn; 8) detection of ring structures; and 9) shifting bonds around on a ring".*

- The following applies to every 3D representation where you can rotate the molecule with a mouse: US Patent #4,835,528 [Texas Instruments]: "*Cursor control system*". Excerpt from abstract: "*A cursor control system for computer displays moves a cursor unambiguously in three dimensions using a two dimensional input device*".

I'll let you guess how many structure determination packages should be thrown away due to the above.

II.b Crystallographic patents

There are also a number of patents directly applicable to our field. Just to list only a few[12]:

- US Patent #6411676 [Nonius]: "*Method for determining parameters of a unit cell of a crystal structure using diffraction*". Method: uses a 2D detector, and several orientations of the crystal.

- US Patent #6,438,205 [Accelrys] "*System and method for reducing phase ambiguity of crystal structure factors*". related to phase distributions,...

- US Patent #6,198,796 [Rigaku]: "*Method and apparatus of automatically selecting Bragg reflections, method and system of automatically determining crystallographic orientation*". Uses the two most intense Bragg reflections as a basis for orientation.

- US Patent #6,438,204 [Accelrys]: "*Linear prediction of structure factors in x-ray crystallography*"

- US Patent #6,345,235 [Edgecombe and Ableson]: "*Method and apparatus for determining multi-dimensional structure*". excerpt from the abstract: "*The method is applicable to a wide range of data relating to fields such as crystallography, fluid dynamics, edge detection, and financial markets, to determine the topology of structures contained therein*"

- US Patent #4,592,082 [USA]: "*Quantitative determination of mineral composition by powder X-ray diffraction*". Excerpt from abstract: "*Ratios of the highest intensity peak of each mineral to be quantified in the sample and the highest intensity peak of a reference mineral contained in the sample are used to calculate sample composition*".

11The authors avoided to patent FFT, see the story at: http://www.siam.org/siamnews/mtc/mtc593.htm
12More can be found at http://www.ccp14.ac.uk/maths/software-patents/

- US Patent #5,353,236 [Subbiah]: "*High-resolution crystallographic modeling of a macromolecule*". Comprises electron density envelopes, phase determination and recycling, MIR,...

- US Patent #6,192,103 [Bede Scientific, Inc.]: "*Fitting of X-ray scattering data using evolutionary algorithms*"

- World Patent #WO9906824 [Shankland and David]: "*Method and apparatus for determining molecular crystal structures*". Includes methods: powder diffraction, description using torsion angles, reduction of data to intensities & covariance matrix, genetic algorithms...

- US Patent #20020107643 [Hendrickson and Honig]: "*Process for pan-genomic determination of macromolecular atomic structures*". Includes methods: genomics database, cloning, purifying, crystallization, MAD measurements, phasing, structure determination, combined analysis of structures in the same family,...

- US Patent #5,418,944 [IBM]: "*Knowledge-based molecular retrieval system and method using a hierarchy of molecular structures in the knowledge base*"

- US Patent #5,386,507 [Teig and Kahn]: "*Computer graphics system for selectively modeling molecules and investigating the chemical and physical properties thereof*". Uses an editor presenting simultaneously 2D and 3D representations.

- US Patent #4,855,931 [Saunders / Yale]: "*Stochastic method for finding molecular conformations*". Uses random displacements of atoms, with constraints and energy evaluation...
...

An interesting game to play after reading these patents is to list all the software you are regularly using for crystallography, and decide how many are infringing on one or several existing patents. Then, try to work without this software (assuming they would not be granted a license to use the technology)...

Of course, by studying carefully all the patents and the date at which they where deposited, you will see that all rely largely on "prior art", and on ideas that are obvious to us specialists. So you *could* prove most of them are actually invalid... if you have the money and the time to go to court...

II.c Crystallography be with patents – locking innovative research

As can be seen from the above list of patents, a large range of inventions can be patented: probably more than half of the articles published in *Acta Cryst. A* and *J. Appl. Cryst* could lead to patents on new crystallographic developments. The first result of enforcing these patents would be of course to restrict the distribution of all the free computer programmes which are daily used in every crystallographic laboratory.

But the issue of patents on software and algorithms goes beyond the problem of those based on prior art or trivial ideas. A lot of highly complex algorithms are used and improved regularly: direct methods, Fourier recycling and density modification, maximum likelihood... Nobody could deny the innovative (sometimes brilliant) character of these research results. But these algorithms are so vital to all modern crystallographic research that patenting them, giving a single group or corporation the right to use or license them, would considerably hinder research progress. *Many complex algorithms are "fundamental theory" to the Crystallographic Community and therefore should never be patented*. It would be like asking aircraft engineers to build new airplanes without using gravity laws because these are patented.

IV Conclusion and perspective on the debate at the European level

To add (certified) neutral information, it is interesting to read the report by an intellectual property expert on "*The Patentability of Computer Programmes*"[13] presented at the European Parliament. Excerpts follow:

- *"It has not been demonstrated that software patents contribute to innovation. The opposite may be true as well."*

- *"Proponents of the patent system all too easily assume that patents are good for SMEs [Small and Medium-sized Enterprises]. In fact, patents can have serious negative consequences for SMEs. For many software-developing SMEs software copyright protection is probably sufficient, obviating any need for patent protection."*

- *"Developers of Open Source Software are relatively vulnerable to patent infringement claims, particularly in respect of trivial patents. Rooting out such patents may obviate the need for special protection of Open Source Software developers."*

- and from the conclusion: *"Patents may not really be needed to stimulate investments in software development, while, on the other hand, software might be more susceptible to the potential negative effects of patents. Software development really is a matter of ideas , and it may not be appropriate to grant exclusive rights on many of those ideas, even if they are novel and not obvious. Still, there do not seem to exist compelling reasons to deny patentability to all software-related inventions in principle."*

The last sentence unfortunately tells a lot about the decision process engaged in Europe. Although *the report asserts that no single evidence as been brought of positive effects of software patents*, the current trend would still be to allow these, because of the pressure to "harmonize" legislation with other countries (USA, Japan), and -probably more important- the intense lobbying of large US software corporations. Indeed, the latter, having been granted numerous patents other the years in the USA, would gain much to see software patents granted in Europe[14].

The aforementioned document nevertheless indicates that it would be necessary to rule out all inventions which are "trivial" or use "prior art" in order to eliminate the most damaging effects of software patents. The trouble is that the current practice of patent offices around the world clearly indicates that this is not possible, if only because patent reviewers cannot be all-knowing and evaluate correctly the validity of patent applications on vastly different subjects (moreover *all* countries would have to follow the same policy for that to be effective).

Many small points in this article can be argued upon, but it should remain clear that software patents are a threat: *the delicate balance (existing in other fields) between "encouraging publication of inventions" and "locking patents" (monopolies, patent parasites) is clearly broken in the field of software development*.

As researchers in a scientific field hungry with algorithms, *we could doubly lose from software patents, first on the availability of free software, and second on the liberty of doing academic research on new algorithms and distributing them*.

Of course, there is a fair chance that even with software patents legalized worldwide, companies involved in crystallographic software would not enforce them against open-source software and academic developers, since corporate and academic developers generally live close together. But with the increasing perception of patents as legal assets, and as decision-making about these matters falls out of the hands of scientists, this may not last. This "sitting duck" policy could prove deadly, and it would be

13 read the "study" document at http://www.europarl.eu.int/hearings/20021107/juri/default_en.htm

14 They are so deeply involved in the legislation process that the first draft of the European Commission on Software Patents was written by an employee of the Business Software Alliance, a company defending the interests of large american software corporations (Microsoft & co). See http://swpat.ffii.org/papers/eubsa-swpat0202/index.en.html

well for the crystallographic community to voice its dissent against software patents, notably with politicians.

Further references have been collected by L. Cranswick and can be found at
http://www.ccp14.ac.uk/maths/software-patents/

Searching Patents Databases

It is possible to search granted patents and applications using the web-based interfaces of Patent Offices. The most efficient is the US Patent Office http://www.uspto.gov/patft/ (see Fig 1 below), with 3910 granted patents and 1140 applications for a search with keyword "crystallography" through years 1976-2003. It is also possible to search European patents (although less practical) through a web interface: http://ec.espacenet.com/espacenet/

Be careful to try different terms for the search, as the language used for patents is often quite different than the one used in scientific publications (read a few patents to get an idea). And do not forget to include in your search all methods applicable, i.e. optimization algorithms, graphics displays, databases, mathematical transforms,...

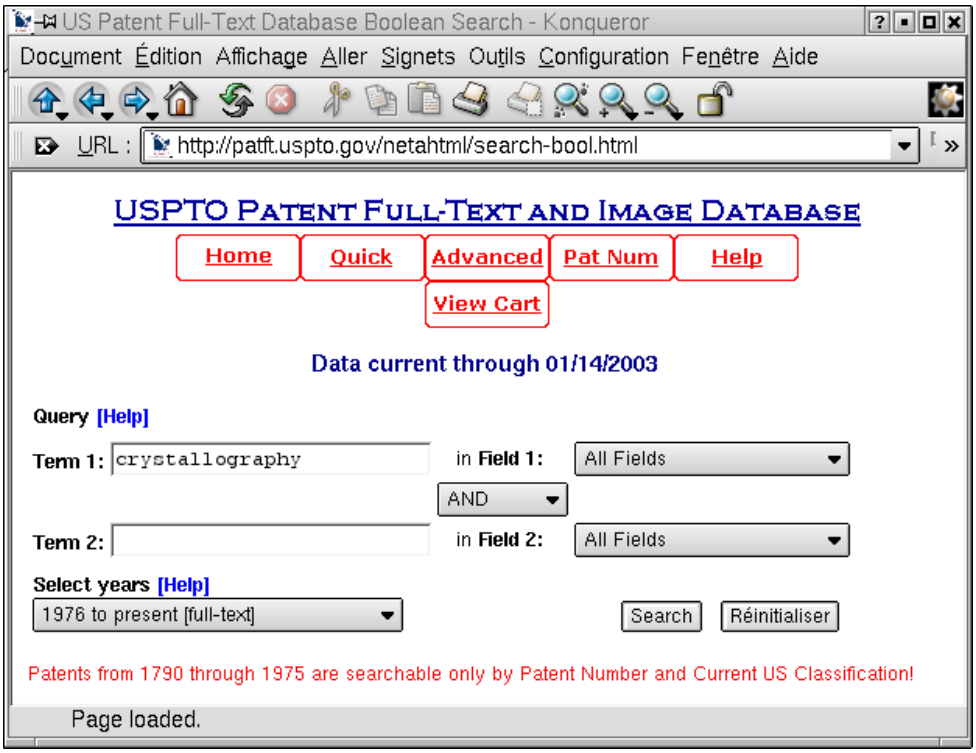


Fig 1: *Web interface to search the US Patent Office database of granted patents via* http://www.uspto.gov/patft/. *Be sure to try different wording for your search, and also try a wider search to include upstream mathematical/computer methods,...*

Order through random numbers : Indexing and solving crystal structures from powder diffraction data using Monte Carlo methods

Armel Le Bail
Université du Maine, Laboratoire des Fluorures, CNRS UMR 6010, Avenue O. Messiaen, 72085 Le Mans Cedex 9, France.
Email : alb@cristal.org - Web : http://www.cristal.org/

When exact equations do not exist, or if they are too complex to manipulate, a problem-solving approach in crystallography (anywhere in fact), inelegant, maybe not deserving the Gregori Aminof Price (GAP) though able to fill a gap sometimes, brute force, but efficient, is the use of random numbers. Randomness finally can end in the finding of a desired order (compatible with both diffraction data and crystal chemistry). This is nothing else than a large scale trial and error process, involving 10^6 to 10^9 trials, if not much more. Quite easy and fast to develop with the programming language of your choice. Good languages offer efficient-enough intrinsic functions which can produce pseudo-random number sequences (for instance the RAN subroutine in Fortran). These random numbers are used for building algorithms producing "random walks" (stochastic techniques, ordinary or Markov chain Monte Carlo, Metropolis algorithm, etc... but this text will avoid the specialized terminology) of which applications have grown exponentially during the past twenty years, in absolutely all domains (literature is quite huge, introductions can be found on the web, see references [1-3]). The reason which may preclude their use is a prohibitive CPU time if a trial needs too long calculations for checking if the change in the parameter randomly modified should be retained or not. But computers are going faster so that software are already written today for problems that will be solved in a matter of minutes or seconds in a near future (10-20 years), even if several days of calculations can be necessary currently. On a standard PC running at 2.5GHz, the 10^9 trials are made in one night if 20000 trials are performed in 1 second. This opens horizons, moreover trial and error process are easy to adapt for parallel computing.

Order is only a kind of disorder (and vice versa) so that if disorder is desired rather than order, a small change in the algorithm is all the business. An non-exhaustive list of computer programs applying random numbers for adjusting parameters (atom or molecule coordinates, cell dimensions, etc) that should match crystallographic data (diffraction or diffusion, intensity and/or peak position and peak shape, etc) follows :

- Small angle scattering, Monte Carlo calculation of the interparticle interference [4].
- RMCA [5] :for modelling disorder in amorphous compounds, matching neutron and X-ray interference functions plus other extra data like EXAFS, NMR (...). More than 100 applications of this program were published.
- DISCUS [6] : for analysis of diffuse scattering from single crystals.
- OCTOPUS, SA, DASH, PowderSolve, Endeavour, PSSP, FOX, TOPAS, EAGER, FOCUS, SAFE, MAUD, EXPO, ESPOIR (... no place for all references, typing the program names as keyword with Google should return some hyperlinks) : for structure solution from powder diffraction data, matching either the raw diffraction pattern [7] or the extracted structure factor amplitudes, or mathematical representations of them. See the recent SDPD (Structure Determination by Powder Diffractometry) Round Robin 2 which seems to establish the triumph of this approach [8].

In most of these programs, a parameter, an atom or a molecule is selected at random, moved randomly, the observed data is then compared with the calculated one, the fit quality is estimated, the move is accepted if the fit quality is improved, but also times to times if it is not improved (in order to not being trapped in a false minima), annealing may be simulated by reducing progressively the amplitude of the moves, and this is all the deal. Absurdities may well occur during the atoms (or molecules) random walk. They quite easily virtually pierce each other, without provoking any nuclear fusion or particle/radiation emission, fortunately. Introducing anti-bump distance restraint during the process is not always a good idea, some problems can be solved better without it. Even the order constraint of a given space group

may not be a good idea, and the initially disordered atoms seem to appreciate more to come to order in the less constrained space group P1, you just need then to check for higher symmetry at the end of the process. At the beginning of the process, some program developers find more attractive to mimic the Big Bang (even if controversial), getting all atoms at the origin first, letting them choose another place at random. Some start from atoms randomly distributed. On the figure 1 is shown the ESPOIR [9] program in action, solving the $La_2W_2O_9$ structure [10] from neutron diffraction data (extracted "|Fobs|"), searching randomly 9 oxygen atoms while the La and W atoms known from X-ray data are immobile. In five minutes, 20 tests of 200000 oxygen atom moves are realized of which few are retained (~500), leading 2 times to the solution with a final Rp~6% on the pseudo-powder pattern regenerated from the extracted '|Fobs|' in order to overcome overlapping problems.

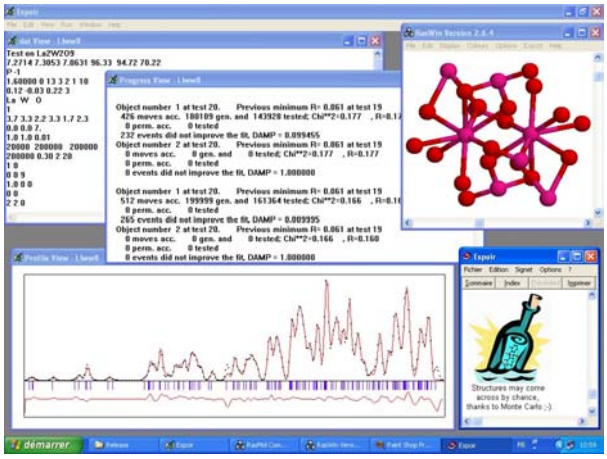


Fig. 1: *ESPOIR finding oxygen atoms in the $La_2W_2O_9$ structure by Monte Carlo from neutron data.*

In some circumstances, the Monte Carlo method is deliberately used as a kind of refinement process (in fact parameter adjustment, the parameter being either an atom coordinate or any crystallographic characteristic) replacing the least-square method. And the applications are not limited to atom moves, other problems in crystallography are candidates :

- Optimization of the design of neutron scattering instruments [11]. No less than five programs are described in Neutron News volume 13, issue 4, 2002.
- Magnetism : magnetic structures simulation by MCMAG [12], spin orientation estimations in FULLPROF [13].
- Crystallite size and microstrain distribution functions estimation from line profile broadening analysis [14].
- Structure factor amplitudes extraction from powder data by the Le Bail method revisited by Altomare in EXPO [15]: no equipartition of overlapping reflections but random values agreeing with the total amplitude sum seems to lead to more efficiency in further structure solution attempts by the direct methods. Etc.

Applications to indexing powder patterns was made recently by using a genetic algorithm [16] or Monte Carlo in the McMaille program [17]. In the latter case, random numbers are used for deciding which cell parameter to vary next in the process, for attributing a new value (between imposed Pmin and Pmax) to the cell parameter selected for a change. This is illustrated below in Fortran language by two calls of the RAN function (returning a random number between 0. and 1.) for deciding which parameter to vary in hexagonal or tetragonal symmetry (CELP(IP=1) = *a*, CELP(IP=3) = *c*) and what new value the selected parameter will have, large changes being allowed at this level :

```
C     Which parameter to vary ? a or c ?
      IP=3                          ! c parameter selected
      IF(RAN(ISEED).GT.0.5)IP=1     ! or a in 50% cases (GT = greater than)
      CELP(IP)=PMIN(IP)+DELTA(IP)*RAN(ISEED)    ! give a new value to a or c
      NTRIAL=NTRIAL+1               ! count the trials
```

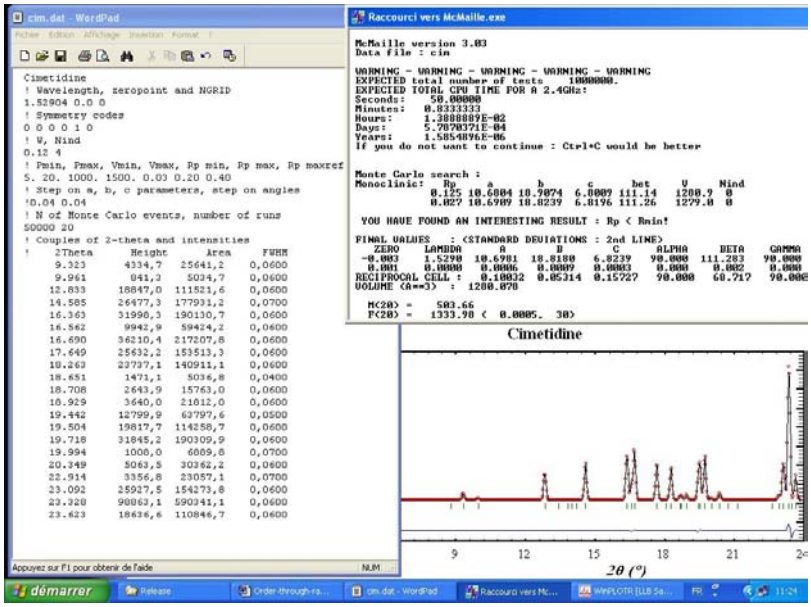


Fig. 2: *The cimetidine (pseudo-) powder pattern Monte Carlo-indexed by McMaille.*

When a cell proposal is found to explain all observed reflection positions (or a part of them if unindexed reflections are allowed), or if the profile fit reliability Rp is lower than a selected value (say 50%), then random numbers are used again for applying that time very small cell parameter changes accepted only if Rp decreases (the large changes being accepted sometimes even if Rp increases). This is more similar to simulated quenching than to annealing. A least-square refinement would not converge starting from a 50% reliability factor, but trial and error does it well (in 100-3000 trials from cubic to triclinic). Unfortunately, this apparently simple problem with up to 6 cell parameters maximum to determine from a set of 20 reflection diffracting angles is transformed into a difficult one (not one clear cell proposal is obtained, but several dubious cells are ranked with similar figures if merit) essentially because of data inaccuracies and sample impurity, but also in special cases when the data which would allow the finding of some parameter are obscured by too much data corresponding to the other parameters (anisotropic cells, for instance characterized by 2 long cell parameters and a very short one). Due to large computing time, McMaille will really be competitive with more mathematically elegant approaches (ITO, TREOR, DICVOL) in ten years, but can already propose a monoclinic cell (figure 2) in one minute, after "only" 10^6 cell trials, when a systematic grid search with 1000 steps for each of the four monoclinic parameters would need 10^{12} trials (2 years of calculation). No need to explain the difficulty with triclinic cells, or with adding a supplementary adjustable parameter which would be the zeropoint... Is that interesting? Well, indexing from powder diffraction data is not a pure mathematical problem and requires the use of complementary approaches, because the various computer programs are at their best in different cases. Any new efficient approach is welcome.

Now, think to your own problems in terms of adjustment by random parameter changes, evaluate the feasibility and interest of your project if compared to other calculation techniques, and, if conclusive, just build the program (preferably open source). Developing McMaille needed 2 months, full time, to one crystallographer/developer.

References

[1] http://www.taygeta.com/mcarlo/references.html
[2] http://www.statslab.cam.ac.uk/~mcmc/pages/list.html
[3] http://www.cooper.edu/engineering/chemechem/monte.html
[4] S.R. Hubbard, S. R. & S. Doniach, S. (1988). *J. Appl. Cryst.* **21**, 953-959.
[5] D. A. Keen & R.L. McGreevy (1990). *Nature* **344**, 423-424.
RMCA program : http://www.studsvik.uu.se/software/rmc/rmc.htm

[6] T.R. Welberry & Th. Proffen (1998). *J. Appl. Cryst*. **31**, 309-317. DISCUS program : http://www.uni-wuerzburg.de/mineralogie/crystal/discus/discus.html
[7] Tremayne, M., Kariuki, B. M. & Harris, K. D. M. (1996). *J. Appl. Cryst.* 29, 211-214.
[8] SDPD Round Robin - 2 (2002). http://www.cristal.org/sdpdrr2/
[9] A. Le Bail (2001). *Mat. Sci. Forum*, **378-381**, 65-70. ESPOIR program : http://www.cristal.org/sdpd/espoir/
[10] Y. Laligant, A. Le Bail & F. Goutenoire (2001). *J. Solid State Chem*. **159**, 223-237.
[11] P.A. Seeger, L.L. Daemen, E. Farhi, W.-T. Lee , X.-L. Wang, L. Passell, J. Šaroun & G. Zsigmond (2002). *Neutron News* **13(4)**, 24-29.
[12] P. Lacorre & J. Pannetier (1987). *J. Magn. Magn. Mat*. **71**, 63-82.
[13] J. Rodriguez-Carvajal (2001). *Mat. Sci. Forum* **378-381**, 268-273. FULLPROF program : http://www-llb.cea.fr/fullweb/powder.htm
[14] P.E. Di Nunzio, S. Martelli & R. Ricci Bitti (1995). *J. Appl. Cryst*. **28**, 146-159.
[15] A. Altomare, C. Giacovazzo, A.G.G. Moliterni & R. Rizzi, R. (2001). *J. Appl. Cryst*. **34**, 704-709. EXPO program : http://www.ic.cnr.it
[16] B.M. Kariuki, S.A. Belmonte, M.I. McMahon,, R.L. Johnston, K.D.M. Harris and R.J. Nelmes (1999). *J. Synchrotron Rad*. **6**, 87-92.
[17] McMaille powder indexing program (2002). http://www.cristal.org/McMaille/

A Protein Crystallographer's Toolkit: Essential programs for successful structure determination

Claire Naylor
School of Crystallography, Birkbeck, University of London, Malet St, London, WC1E 7HX, UK.
E-mail: c.naylor@mail.cryst.bbk.ac.uk – WWW: http://www.cryst.bbk.ac.uk/people/claire_naylor.html

Background

In Sales speak, (protein) crystallographer's have traditionally been 'early adopters' of new computing technology. As a community, we were in the vanguard of those utilising the new computers for practical work before World War II, and this has continued through to the modern day with a steady stream of ever more complex programs taking advantage of the explosion in computing power. The requirement for such high level computing has meant that protein crystallography tended to be carried out in dedicated departments, with well set up networks, and access to a large pool of computational expertise. In these circumstances, we have perhaps been perceived by more laboratory-based molecular biologists as happiest when situated as in figure 1.

Fig. 1: *A happy molecular biologist, content with a nice suite of installed crystallographic programs*

More recent years have seen a massive explosion in the size of the community and in the accessibility of the technique to researchers without extensive training in the area. Crystallographic groups made up of perhaps only 1 or 2 senior researchers within more general molecular biology departments are becoming the norm, and in this environment, setting up an appropriate computing environment can prove

challenging, especially for those with little computer management experience. It can be difficult to identify all the software required, and in such situation the crystallographic computing environment can unfortunately deteriorate rapidly to that depicted in figure 2a and b.

Fig. 2 (a, b): *A not so happy molecular biologist, whose frustration is possibly being caused by a narrow and/or poorly installed suite of crystallographic programs.*

The department of crystallography at Birkbeck College has a long history and a successful crystallographic computing environment, this article contains a brief description, with links to appropriate Websites, of programs that we find essential in the structure determination process. Two things will strike the reader immediately: the programs are free to academic institutions, and thus there is no barrier to the new researcher becoming involved in crystallography for the first time. There is some redundancy: in many cases, we maintain two program suites that apparently do the same job in different ways. It is frequently found that problems will yield to one form of analysis, but appear intractable under another, such redundancy is therefore always desirable.

Operating Systems

Protein crystallography has in the past relied on Silicon Graphics machines in many circumstances, though this is now yielding to cheaper Linux boxes that are at least as efficient. Compilations for both these machine types are thus generally available. Dec alpha machines are also used by the community, and compilations for this system are usually available. The Mac OS-X system is much less frequently used, but fortunately there are enthusiasts willing to share their experience of porting crystallography programs to this operating system (http://chemistry.ucsc.edu/~wgscott/crystallography_on_OS_X.html).

CCP4

Finally the programs! Firstly, in a category of its own, comes the Combined Computational Project 4 (CCP4; http://www.ccp4.ac.uk/), supported by the United Kingdom's BBSRC. This provides a program suite not only covering all aspects of protein crystallography from data collection to data validation and graphics, but also provides utilities to port your data from/into a variety of other well known crystallography programs. CCP4 is currently at version 4.2.2 with 5.0 expected later this year. Following the introduction of an easily used graphical interface (ccp4I; figure 3), this is probably the single most heavily used set of programs in the department. New graduate students have little difficult understanding it and can make rapid progress. In addition, dedicated support staff and one of the community's most popular bulletin boards mean that help is always available to the new user. Individual programs available through CCP4 will be described in the sections below.

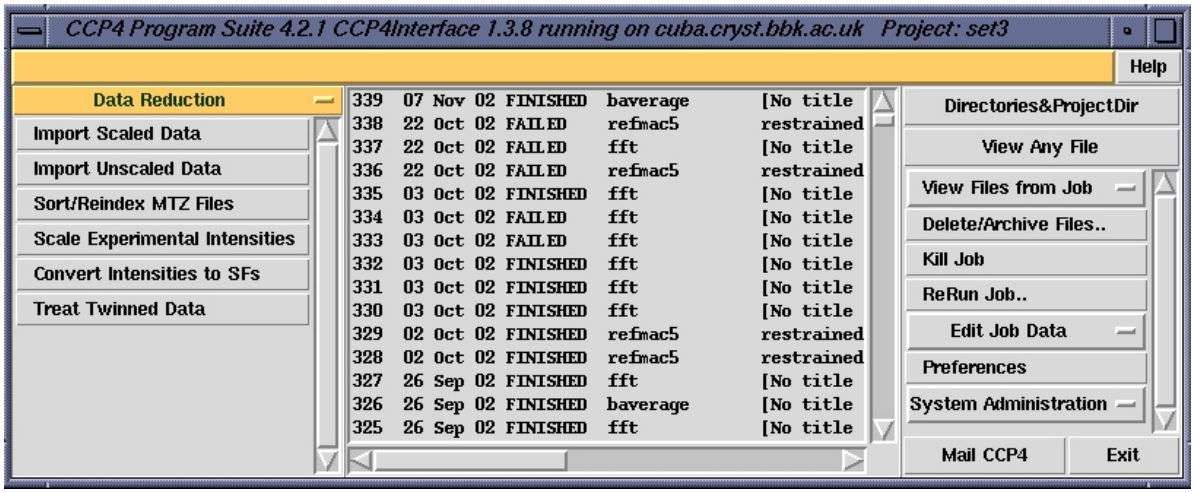


Fig. 3: *Example of the graphical user interface in the CCP4 suite*

Data Processing and Scaling

Birkbeck maintains two packages for this process. Mosflm (http://www.mrc-lmb.cam.ac.uk/harry/frames/) and Scala (both part of the ccp4 suite), and the free-to-academics version of the HKL package (http://www.hkl-xray.com/). Both have graphical interfaces (see figures 4 and 5), but Mosflm is perhaps easier for the novice user to handle, consisting primarily of clickable menu options, while Denzo utilises a command line. Both, however, are very effective and processing even the most difficult datasets.

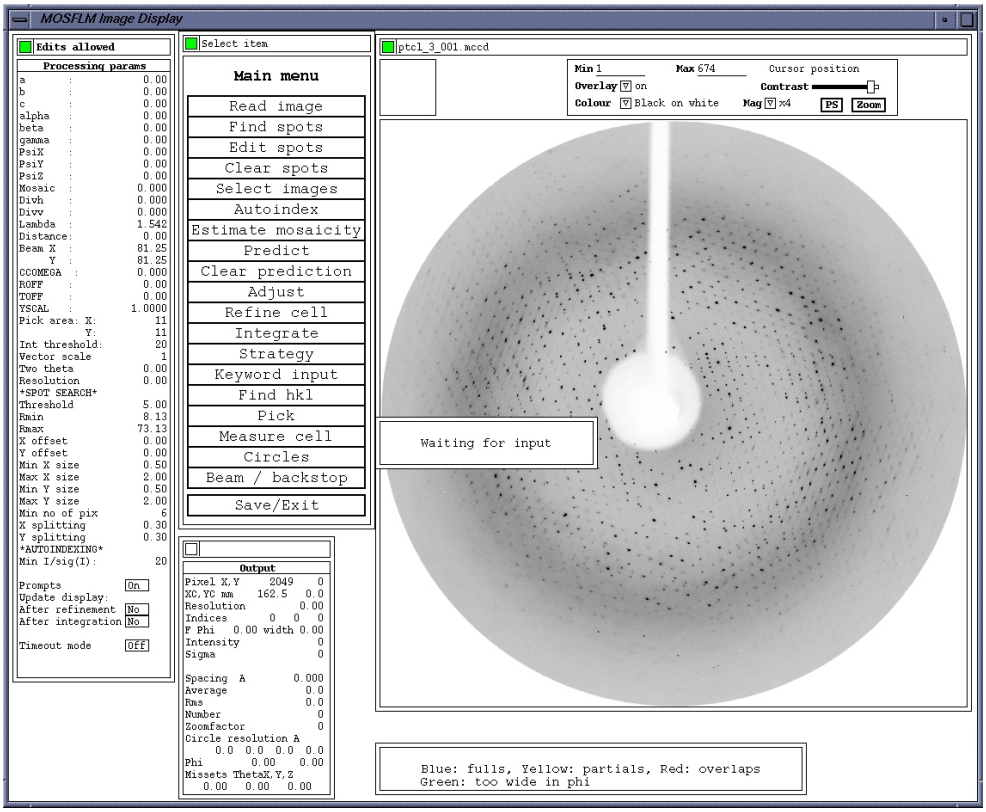


Fig. 4: *Screen image of MOSFLM*

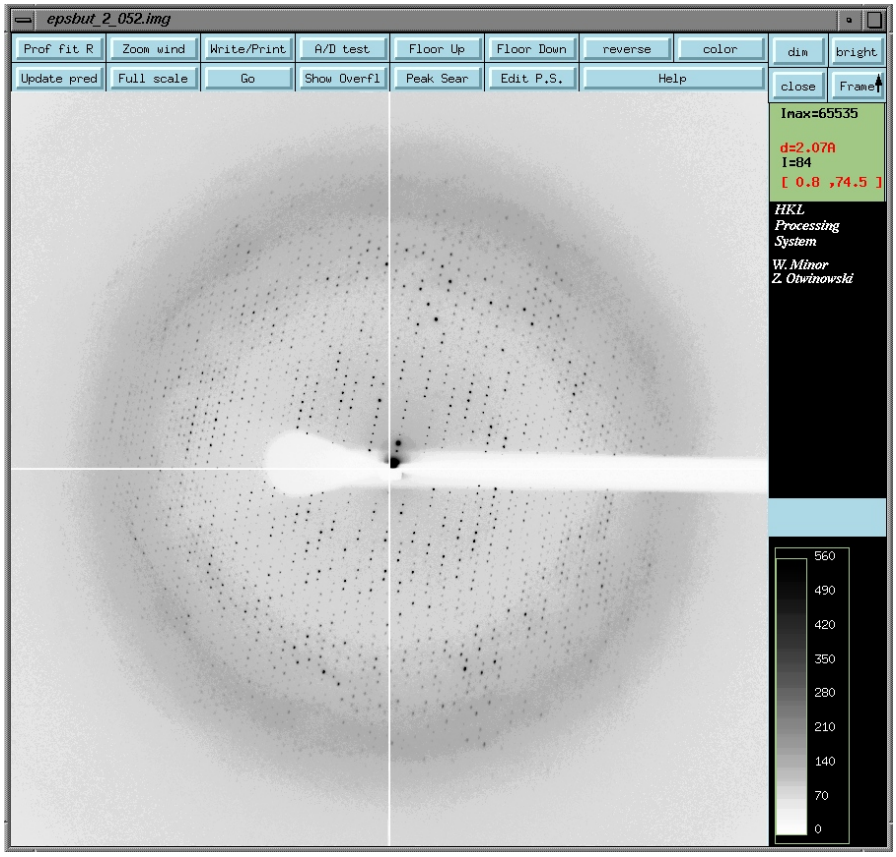


Fig. 5: *Screen image of DENZO in action*

Heavy atom site identification

There are now many programs available for the automatic determination of heavy atom positions, by a number of different methods, all are successful in different circumstances and we find it is invaluable to support as large a variety as possible. Strategies available within the department to solve the heavy atom substructure include: Patterson search, as employed in rsps under ccp4, and the far more sophisticated and comprehensible Solve from T. Terwiliger (http://resolve.lanl.gov/); direct methods in shelxd a part of G. Sheldrick's Shelx suite (http://shelx.uni-ac.gwdg.de/SHELX/) and Acorn, the latest direct methods program available under ccp4; and finally the combined direct methods and Molecular dynamics methods employed by SnB from (http://www.hwi.buffalo.edu/SnB/), now also available as part of the site identification and refinement package, BnP (http://www.hwi.buffalo.edu/BnP/).

Heavy atom Refinement

There is no doubt that the best program for carrying out this task is Sharp, written by G. Bricogne et al and available from http://www.globalphasing.com/. The installation of this program is none trivial for the novice computer manager, but is worth the large amount of effort this take for the high quality maps it produces. Both BnP and Solve/Resolve also carry out heavy atom position refinement.

Molecular Replacement

CCP4 provides three very useful programs for structure solution by molecular replacement. J. Navaza's AmoRe is very rapid, allowing the user to carry out multiple trials of different models and resolution ranges to maximise the possibility of finding a solution. MolRep, a more sophisticated program that utilises Maximum Likelyhood targets, is slower, but more likely to find the right answer with a poorer model. Finally R. Read's Beast is the latest molecular replacement program, it is extremely slow and thorough but very successful, and is thus the best choice for the most challanging problems. Also supported in the department are two programs that use a different methodology: EPMR

(http://www.msg.ucsf.edu/local/programs/epmr/epmr.html) uses an evolutionary search algorithm: repeated rounds optimizing rotational and translational parameters for a set of random models and discarding the poorest scorers. CoMo carries out a six-dimensional search (both rotational and translational parameters), which can be successful where more traditional methods have failed.

Density Modification

The interpretability of an initial map can be improved by both solvent flattening and non-crystallographic symmetry averaging, as well as a range of less powerful techniques. CCP4 provides DM and Solomon (which has the novel 'solvent-flipping' algorithm) for this. T. Terwilliger's Resolve (http://resolve.lanl.gov/) also carries out these functions with slightly different methodology. Finally the Uppsala Software Factory (http://xray.bmc.uu.se/~gerard/manuals/) provides a number of programs to carry out these operations.

Automated Building

Arp/Warp (http://www.embl-hamburg.de/ARP/) is perhaps the most famous automated building package, and the latest version (6.0), although downloaded separately, can be started from within ccp4i. Arp/warp requires resolutions in excess of 2.0Å to succeed. Those of us less fortunate can use Resolve to generate and initial chain trace, or FFFear, which uses a real space search to place stretches of idealised helix and strand in an electron density map. Those blessed with atomic resolution data will want to try the direct method structure solution available in Acorn in CCP4.

Model Building

No matter how good the data, we all have to compare the model and the map sooner or later. Two programs are used for this within this department. Alwyn Jones' O (http://www.imsb.au.dk/~mok/o/) was for many years the only widely used protein crystallography building program. It is still extremely good for model building, especially from scratch. However, Duncan McRae's XtalView (http://www.scripps.edu/pub/dem-web/toc.html) is being increasingly used and is particularly good when making alterations to an existing model.

Refinement

Once a model has been built, Birkbeck supports a number of programs for refining it. CNS (http://cns.csb.yale.edu/v1.1/), with A. Brunger's simulated annealing protocols is perhaps most useful at medium to low resolution. Refmac 5, from the CCP4 suite is very efficient at slightly higher resolution, and finally where the data approach atomic resolution, then G. Sheldrick's SHELXL offers the greatest flexibility.

Model Validation

It is easy to forget tools for model validation, but trying to ensure our model is as accurate and unbiased as possible is an important part of the structure solution process. Under the CCP4 suite, ProCheck and SFCheck provide a number of useful indicators.

The above is a very selective list from the vast area of protein crystallography programs available, and is biased towards those that are free of charge. However, installation of at least one program from each of the above categories should allow the user to solve their structure, without too many scenes like those in figure 2!

Merging data from a multi-detector continuous scanning powder diffraction system

Jon P. Wright, Gavin B. M. Vaughan and Andy N. Fitch,
European Synchrotron Radiation Facility, BP-220, F-38043, Grenoble Cedex, France.
E-mail: fitch@esrf.fr – vaughan@esrf.fr – wright@esrf.fr ; WWW:
http://www.esrf.fr/exp_facilities/ID31/contacts.html - http://www.esrf.fr/exp_facilities/ID11/handbook/staff/gavin/Gavin.html - http://www.esrf.fr/exp_facilities/ID11/handbook/staff/jon/jon.html

For a wide range of problems in powder diffraction, high angular resolution is crucial for untangling overlapped reflections. Higher resolution can always be obtained at the cost of a decreased count rate, and the scientist must balance the requirements for resolution against those of flux. The extremely high intensity and collimation of a synchrotron X-ray source provides an excellent basis for a general purpose high resolution powder diffraction instrument. At the ESRF a parallel beam geometry was selected and originally implemented using X-rays from a bending magnet source on beam line BM16, with nine analyser crystals mounted on a comb support with constant angular offsets of ~2 degrees (figure 1). That instrument has recently been moved to beam line ID31, where an undulator source gives a large enhancement in the incident X-ray flux and lower divergence in the incident beam.

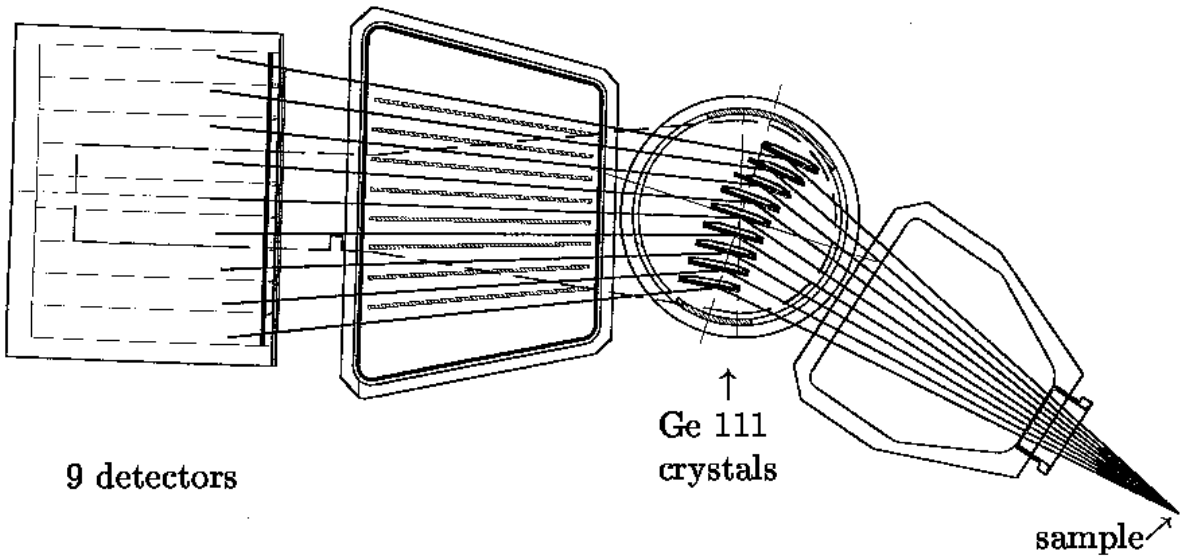


Figure 1: *Picture of the multianalyser stage [1].*

In optimising the data collection rate from a high resolution instrument one of the surprising but important factors turns out to be the time for the mechanical system to move. Collecting data with a step scan introduces a significant dead time while the detector arm is repositioned between measurements of adjacent data points. An alternative scheme is to drive the detector arm continuously and record accumulated counts and the detector arm's angular position repeatedly throughout the scan. This scheme is advantageous as the dead time is eliminated, however some complications are introduced in that the data no longer arrive in a familiar format. Also the data from the nine separate channels and usually a series of scans need to be normalised and merged together to provide a convenient format for Rietveld refinement and other data analysis programs. These procedures are somewhat analogous to those in use with time of flight neutron diffractometers, where many detectors are rebinned to produce a histogram of data. We outline the main aspects of the data processing procedures implemented here at ESRF for data collected on the high-resolution powder diffractometer. Other procedures are available for data collected using area detectors, which have been described elsewhere [2].

The hardware is set up to record the state of the instrument as a function of time. The detector arm's angular position is read from the encoder mounted on the diffractometer axis and the counts received in

each of the nine detectors since the last angle reading are also recorded. The raw data produced by the instrument are a list of angle positions, times and counts, with each line in the data file representing the state of the instrument at a particular time. Thus the raw counts represent those photons which were detected during the time that the instrument moved from the previous angular position to the current one. The raw data files (in ascii) contain a header giving the initial position of the detector arm, along with various other motors present on the beam line, in a standard ESRF SPEC format. Currently the system is triggered by a hardware clock, giving steps which are constant in time but not necessarily angle. The angle encoder has a precision (stepsize) of 0.00005º and is accurate to ±0.5 arcsec in 360º, allowing a series scans to be merged together without encountering problems due to changes in zero-shift. Offsets between the nine detector channels are constant (by mechanical construction) and are also independent of energy, since all nine analyser crystals are mounted on the same rigid support.

For very strongly diffracting samples the maximum count rate available from the scintillator detectors (~100kHz) or the maximum angular velocity (16 deg/min) available from the motors can be rate limiting factors, although such cases are relatively rare. In practice the diffractometer is usually scanned at an angular velocity of the order 1-10 degrees per minute with a sampling time such that 2000 points per degree are measured (or 1 point for every 10 encoder steps).

Rebinning the data is a straightforward computational procedure, provided it is assumed that the counts arrived uniformly in between the sampling times. In practice the experiment should always be carried out such that the sampling step size is much less than that required for the final binned histogram, which is normally the case with default software settings. Figure 2 illustrates the behaviour of the algorithm used, where the "spread over many bins" case should generally be avoided. The data which are produced are a histogram, which can potentially cause problems in some Rietveld software if a very coarse step size is used. (The peak function should be the integral over the bin width and not the value at the centre, which is only a sufficiently good approximation when the function is linear over the bin width; usually the case if the step size is less than the FWHM/5).

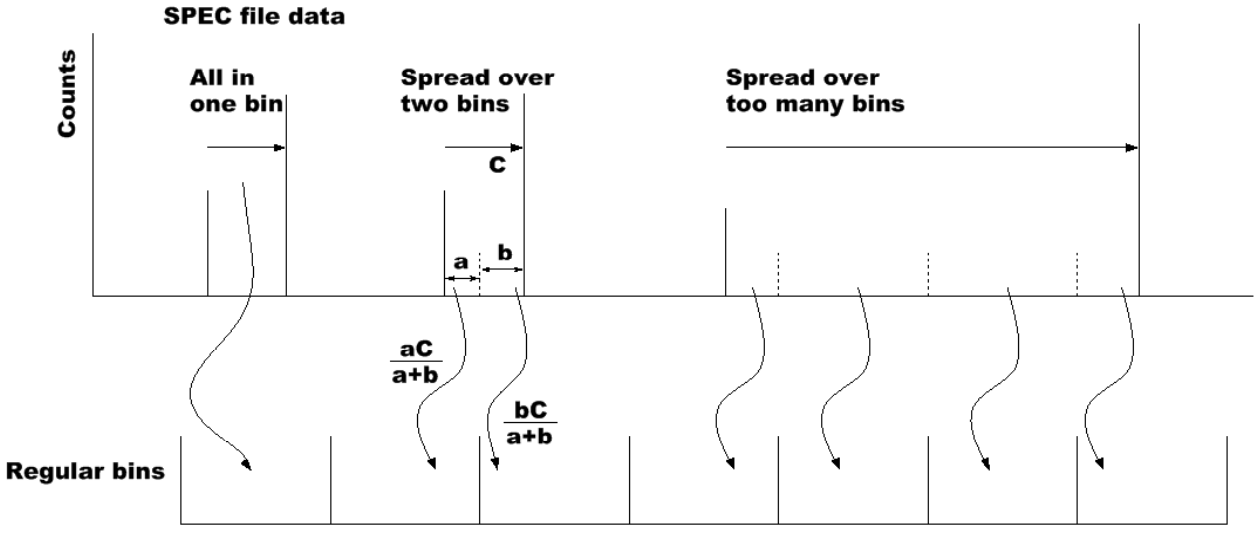


Figure 2: *Rebinning the raw data.*

Since the nine channels are offset from each other it is necessary to rebin each channel separately, and also to rebin the incident beam monitor counts along with each channel, since the incident X-ray intensity may change significantly during the time it takes for detectors to reach equivalent places in the 2θ scan. Detector and monitor counts collected during different scans but for the same channel can simply be summed after they have been assigned to equivalent angular bins. Data points that are collected when the incident flux is below a threshold, such as during the few minutes it takes to refill the storage ring, are excluded.

Combining the data together from the nine different channels requires some corrections to be made, apart from the simple addition of the angular offset. In the most common experimental arrangement the sample is mounted in a capillary and all the channels have equivalent geometric and absorbtion factors, so that only the detector efficiency needs to be taken into account. The relative efficiency of the detector channels can easily be determined from those data points where all of the nine channels have contributed and a capillary sample was used. For flat plate samples it is only possible to provide a θ-2θ geometry for one of the nine channels, meaning that appropriate corrections must be made before the data can be combined. These corrections are further complicated in the cases where the diffracted beam is clipped or obscured when a long sample length is illuminated by the X-ray beam, particularly at low angles. Capillary geometry is generally to be preferred for these reasons, as well as to avoid unwanted texture effects.

Propagating the errors from the original counts to the final data file is very important for later analysis and refinement and requires an "esd" data format for most popular refinement packages. When some data points may only have been measured once by one channel but others may have been visited several times by all nine detectors the statistical weighting will be very different. A subtlety which arises in the error propagation comes from the usual approximation of taking sqrt(counts) to represent the Poisson distribution for the counting statistics. At very low count rates this approximation becomes progressively worse, failing completely when zero counts are recorded. The easiest way to avoid these pitfalls is to sum the data together as far as possible before taking the square roots, and also to apply the efficiency correction to the monitor, rather than the detector signal, since this will generally be the larger of the two. Within the actual software it is preferable to use the quotient of sums when combining detector and monitor counts from different channels, rather than the weighted sum of quotients, simply because the weights can be difficult to determine at low count rates.

In the current software the signal, S, is given by S = C/M, where C is the sum over scans and channels of detected counts and M is the equivalent sum of efficiency corrected monitor counts. The error in the signal is then given[15] by: $\langle S\rangle^2 = (C+a)/M^2 + (C\langle M\rangle/M^2)^2$, where $\langle M\rangle$ is the error in the summed corrected monitor counts and a is a constant to avoid taking sqrt(0) as the error in zero counts, by default set to 0.5, which may be user defined. A single channel has a corrected monitor value given by m = n.e, where n is the number of monitor counts and e is the efficiency. Adding the errors for the nine individual monitor spectra in quadrature provides the error in the summed corrected monitor counts where. The error is given by $\langle m\rangle^2 = n(e^2 + n\langle e\rangle^2)$, where e and $\langle e\rangle$ are the efficiency and error in the efficiency respectively. The uncertainty in the detector efficiency is usually negligible in practice, provided a proper calibration is carried out.

The data analysis routines were originally developed as a collection of C programs which wrote intermediate results to files and were coordinated by Unix shell scripts. Coincident with the move of the instrument from the bending magnet source to the undulator, the software was completely rewritten in Fortran in order to cope with an increased data collection rate and exploit improved computer hardware. Sufficient computer memory is now available to retain the intermediate results without writing to file, allowing data to be processed using only a single pass through the raw data file. Figure 3 shows typical output from running the program.

High data collection rates with a fast turnover of experiments and frequent changes to the sample environment can all potentially lead to problems with the final dataset, which must be diagnosed as soon as possible. Comparing the signal from the nine channels and ensuring that all are measuring the same signal, at least within statistical expectations may identify many problems. Similarly the data from a series of scans may also be compared to ensure that the sample has remained stable over time. This is an advantage of collecting a series of faster scans, compared to a slow single scan which must eventually be abandoned if it is later found that the sample was not in the same state at the beginning and end of the experiment.

[15] Using $(\langle S\rangle/S)^2 = (\langle C\rangle/C)^2 + (\langle M\rangle/M)^2$ for uncorrelated observations, and taking $\langle C\rangle^2 = C+a$

```
diffract31:~% id31sum data/nac_050_18-12-2002_si111.dat 0.001 1 10 renorm

temp.res file found and read in
 -30.000     < tth <   160.00     step= 0.10000E-02 npts=      190000
Processing scan     1    2    3    4    5    6    7    8    9   10
Determining detector efficiencies
Channel efficiences found from     43887 points where all detectors overlap
Efficiencies from temp.res file, the values found now are compared
Det     Offset    Old Eff    Old <E>    New Eff    New <E>
  0  8.0606637  1.0000000  0.0000000  1.0407409  0.0006292
  1  5.8858071  1.0000000  0.0000000  0.9677989  0.0006151
  2  3.9589980  1.0000000  0.0000000  0.9637898  0.0006241
  3  2.0975745  1.0000000  0.0000000  1.0696193  0.0006726
  4  0.0000000  1.0000000  0.0000000  1.0615594  0.0006832
  5 -1.9474072  1.0000000  0.0000000  0.9937005  0.0006717
  6 -3.9962779  1.0000000  0.0000000  0.9673909  0.0006772
  7 -6.0458136  1.0000000  0.0000000  0.9413822  0.0006848
  8 -8.0536742  1.0000000  0.0000000  0.9940181  0.0007245
Created temp.res file
R_exp =   5.446 with      10 pts having I/<I> less than 3, from   76116 pts obs
Reduced chi**2 for channel merge =   0.9876 from     535831 pairs of pts
 0.54% differ by >3 sigma,  0.0174% by >6 sigma (ideally 0.04% and 0.0000%)
Wrote diag.mtv file, outliers at   6.0000
Wrote nac_050_18-12-2002_si111.epf
Time taken was  16.76/s
```

Figure 3: *Typical output from the binning program. The "temp.res" file mentioned in the output is used to store efficiency and offset information between runs of the program.*

A graphic example of the dangers of combining data without performing checks, or indeed only carrying out one slow scan with a single detector channel is shown on figure 4. Such spurious peaks might provide an insurmountable challenge for indexing the diffraction pattern, but in fact correspond to thermal instability in the experiment. A hiccupping cryogenic gas stream was occasionally sending jets of warm shroud gas onto the capillary, causing brief temperature excursions of many tens of Celsius. The thermal expansion of the sample was sufficient that when one of the detectors was measuring the Bragg peak it had moved to a completely different position.

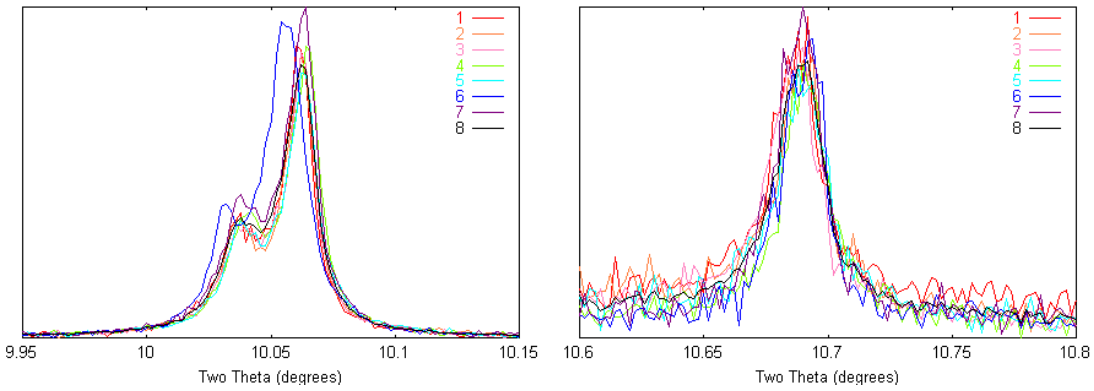


Figure 4: *Plots of data from individual detector channels, showing spurious data due to temperature instability. In the left diagram the peak in channel 6 is clearly displaced, but not due to an incorrect detector offset, as shown by the agreement of peak positions in the diagram on the right.*

In conclusion, the data from the multi-detector continuous scans collected at ESRF are routinely found to give excellent Rietveld refinements, indexing figures of merit (M20) of the order several hundred and represent a much more efficient way to collect data compared to step scanning. Data with variable counting time strategies, particularly counting for longer at high angles for structural refinement problems, may easily be collected as a series of scans for later merging. The extra complications

introduced by collecting data from multiple detectors and irregular step sizes can be treated in a simple way, provided appropriate statistics are applied.

Acknowledgements

Thanks to the ESRF instrument control software personnel, especially A. Beteva, J. Klora and A. Homs for the implementation of the continuous scanning control procedures.

References:

1. J.-L. Hodeau, *et al*., SPIE Proceedings, **3448** (1998) 353-361.
2. A P Hammersley, http://www.esrf.fr/computing/scientific/FIT2D/

---

ccp4i - the CCP4 graphical user interface

Martyn Winn,
Daresbury Laboratory, Keckwick Lane, Daresbury, Warrington, Cheshire, WA4 4AD, UK;
E-mail: M.D.Winn@dl.ac.uk ; WWW: http://www.ccp4.ac.uk/ - http://www.ccp4.ac.uk/martyn/martyn.html

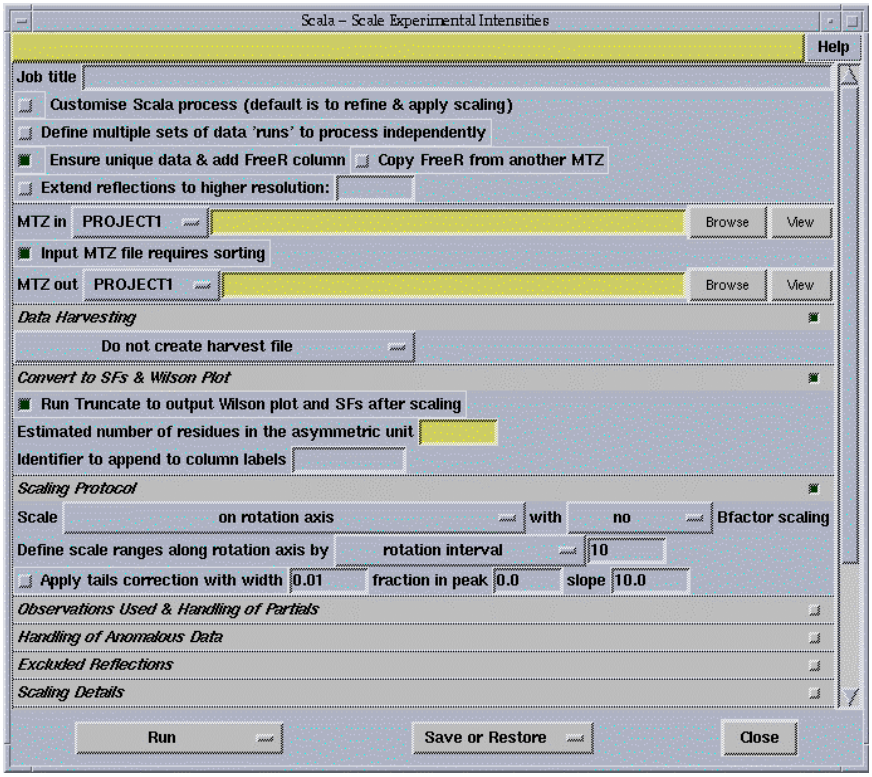
History

The Collaborative Computational Project No. 4 (CCP4) is a BBSRC-funded project to promote collaboration in the development and use of software in macromolecular crystallography. As part of its remit, CCP4 produces a suite of some 160 programs covering functionality for structure solution. The majority of the suite consists of Fortran 77 programs which can be built and run on most UNIX platforms, Windows and Macintosh OS X. Each Fortran program reads in control data on standard input together with a small number of standard format data files, and writes out log information to standard output together with modified data files. Traditionally, shell scripts are used to run a series of programs, linking the output of one program to the input of a following program.

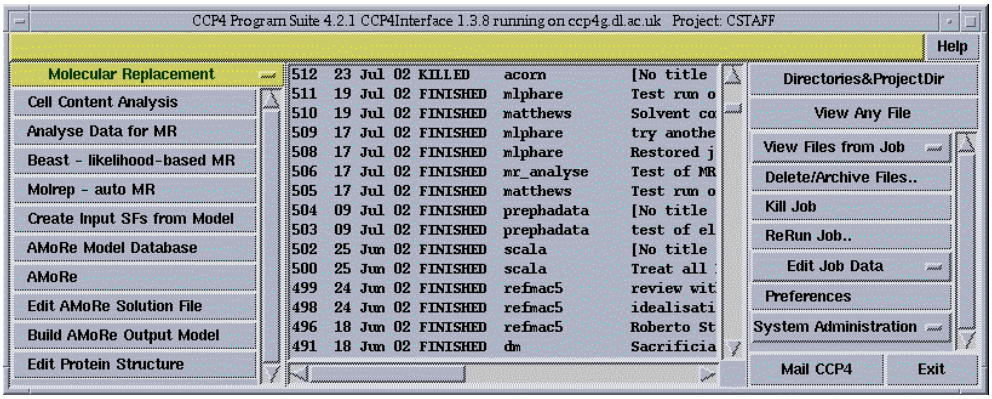
A few years ago, it was decided that a Graphical User Interface (GUI) was required for the CCP4 suite. The decision was partly driven by the increasing proportion of non-crystallographers using the suite for whom guidance in using the programs would be very valuable. In addition, students are now more familiar with a point-and-click environment rather than command-line driven programs. The CCP4 GUI, named "ccp4i", was designed and written by Liz Potterton at the University of York over the period 1997-2001, with the first public release in 1999. Since early 2001, ccp4i has been maintained and developed by the CCP4 team at Daresbury Laboratory, in particular Peter Briggs.

Design of ccp4i

ccp4i was written with a number of fundamental design principles in mind. Most importantly, the GUI was to be a separate layer from the CCP4 programs, and was not to impact on the latter in any way. Thus, users could continue to use CCP4 from the command line if they so wished. At one level, therefore, ccp4i simply prepares input for CCP4 programs and presents the output to the user. In fact, computations in ccp4i are centred on "tasks" rather than individual programs, where a task is a sequence of programs (specified by the developer) corresponding to a particular step in structure solution. Each task is associated with a single window where the user enters filenames for data files and other control input. A typical task window looks like:



Task windows are typically launched from the main window of ccp4i which looks like:



On the left-hand side is a list of available tasks. Clicking on a task launches the aforementioned task window. In the centre panel is a list of jobs that have been run or are running, together with some status information. Finally, on the right-hand side are tools for viewing the output of jobs and for administering jobs and customising ccp4i. Output files can be viewed by a variety of utilities which are made available with a plug-in approach. Help is available at several levels in ccp4i – a simple on-screen message line is bound to the cursor location, while clicking on the "Help" button in each window takes the user to the appropriate section of the html documentation pages.

Thus ccp4i facilitates the preparation of program input, provides tools for viewing output, and gives guidance to the user through an integrated help system and through the provision of well-defined crystallography tasks. In addition, ccp4i includes a set of basic project management tools. A user's work is categorised according to "project", corresponding roughly to an individual structure solution. The user works in one project at a time, and each project has information on all jobs run within that project. Previous jobs can be reviewed and if desired re-run with slightly modified parameters.

Technical Aspects

ccp4i is written in standard Tcl/Tk, the only extension used being Blt for drawing line plots. Tcl/Tk was chosen since it was at the time well-established, widely ported and perceived to be easy to use. Java was

considered but Tcl/Tk was thought to be a more natural choice for the intended purpose. Today, we would probably use Python, but while Tcl has some limitations as a scripting language, it continues to serve us well for ccp4i. The latest GUI developments in CCP4 will also use Tcl/Tk for the sake of compatibility with existing tools, but will in addition make use of IncrTcl, the object-orientated extension.

Jobs are run from a separate tclsh process. This process reads job parameters into the local scope, sources the appropriate tcl run script, and finally exec's the program executable. The job parameters are held in a file, which is written by the task interface, and which can be re-used or modified at a later date. More recently, ccp4i has been adapted to run Python scripts, although the interface remains coded in Tcl/Tk.

Use by 3rd party software

Because ccp4i was designed as a separate layer to the CCP4 suite, it is relatively easy to use it to interface to 3rd party programs, provided they can be run from scripts. To create a new task interface, at most four files need to be written: 1) a tcl file defining the layout of the task interface, 2) a file defining the parameters required and default values if appropriate, 3) a run script to run the program with the parameters defined by the interface, and optionally 4) a template file for the program input. Library procedures are available to facilitate writing these files. Once written, these files can simply be added to the CCP4 distribution, and the new task registered via a tool in the main interface window.

Interfaces have now been written for some non-CCP4 protein crystallography software, namely ARP/wARP and SHELXD. ARP/wARP (http://www.arp-warp.org/) is a software suite for the automatic building of macromolecular models. SHELXD is a program for structure solution of small molecules and heavy atom substructures of macromolecules (http://shelx.uni-ac.gwdg.de/SHELX/). Further 3rd party interfaces are under development.

Scripting in ccp4i

As well as providing a friendly user interface, ccp4i provides an environment for scripting. The run scripts referred to above take the place of traditional shell scripts and allow for the sequential running of programs with a limited amount of logic applied (i.e. with the actual sequence depending on the parameters supplied). The run scripts are written by the developer, and the user is not encouraged to change them, though this is possible.

A small amount of processing also takes place at the interface level. Utility programs are run at various points to check the contents of files, to perform format conversions, and to check the consistency of parameters. From the point of view of the user, ccp4i hides much of the administration of data files.

Nowadays, there is a drive towards increased automation, with the user expecting to make fewer decisions, except those of a scientific nature. Automation occurs at the level of individual programs, with more sophisticated algorithms, but also at the scripting layer. As mentioned above, the ccp4i run scripts embody some logic, but this is principally in terms of pre-defined parameters. Clearly what is needed is decision making based on the outcomes of previous steps in the pipeline. Steps are now being taken to develop CCP4 and the ccp4i environment in this direction.

Future directions

As described above, ccp4i performs several functions, including those of graphical interface, project management, scripting, data management and job control. While ccp4i performs these functions well within CCP4, it is desirable to adopt a more modular system. This would be more appropriate for distributed systems, with interface, programs and data located on geographically disperse systems. It would also facilitate linking to external resources, such as remote databases.

The first stage in developing a more modular system is to separate out the existing functionality of ccp4i. The intention is to develop the underlying architecture, while preserving the functionality that the user sees. The existing ccp4i will be split into three components: 1) a Database Handler for controlling access to the job database, 2) a Graphical User Interface, and 3) a Resource Handler for linking the other components to each other, to CCP4 applications and to external resources. Once this separation is achieved, the components can be developed further quasi-independently. The Graphical User Interface will be kept as purely a graphical layer, with the minimum of additional functionality. Likewise, the Resource Handler should be a thin layer who's principle job is to pass messages to the appropriate resource. An additional component, not yet implemented, is an Expert System which embodies knowledge and logic for the purpose of automating structure solution.

Many of these ideas have been tested in recent developments in Mosflm and the associated DNA project. See http://www.mrc-lmb.cam.ac.uk/harry/mosflm/ and http://www.dna.ac.uk/ for further details. Communication between modules in these projects is via XML-encoded messages passed through sockets, and CCP4 is likely to adopt a similar scheme.

Availability

ccp4i is distributed under Part (i) of the CCP4 licence. Visit http://www.ccp4.ac.uk for more details, or contact CCP4 staff at ccp4@ccp4.ac.uk

---

Meeting, workshop and school reports

Crystals and Twinning workshop during the IUCr Congress, Geneva , August 2002

A workshop on the first day of the IUCr Congress and General Assembly aimed to cover methods for dealing with twinned data sets including tools for analysis and refinement within the structure refinement package CRYSTALS.

The day was organised by Richard Cooper, David Watkin, and Simon Parsons, and included talks from Christer Svensson of Lund University and Lee Daniels of Rigaku/MSC.

Fig 1: *The workshop presenters. From left to right: David Watkin, Simon Parsons, Christer Svensson, Richard Cooper and Lee Daniels.*

The morning was an introduction to basic structure solution, refinement and analysis using the CRYSTALS package. The majority of the time was hands-on, working through examples on the PCs provided: Although some of these were of questionable quality we muddled through with the help of participant's laptops and the enthusiastic support of the on-site IT support and eventually had everyone paired up around a working machine. Recent developments to the tools for analysing data during the refinement were highlighted. Plots of I and σ(I), merging-R and systematic absence violations gave users the chance to spot problems with their data before they even solved the structure. Later, plots of residuals, weighting schemes and F_o against F_c, proved to be very useful for spotting trends such as extinction and outliers in the data.

The morning finished with a complementary talk from Lee Daniels about CrystalStructure, Rigaku/MSC's structure refinement and analysis package, followed by a five minute space group quiz for participants, written using CRYSTALS own scripting language for which the prize was a traditional triangular-prism chocolate bar of the region.

The first half of the afternoon focused on the twinning. Simon Parsons gave an excellent lecture on the causes and problems arising from twinning and covered many programs that can be used to identify and resolve the problem at either the data collection or the refinement stage of an analysis. Christer Svensson talked about his program TwinSolve for the routine solution of twinned data sets. The workshop broke into small groups to work through a set of questions, covering both meroderal and non-merohedral twinning.

Everyone solved three twinned data sets, including one requiring use of a Patterson map. The structures were refined, a twin law found and applied and then the structures refined to completion. The use of ROTAX and the addition of a graphical interface make these steps very straight-forward within CRYSTALS to the extent that these could perhaps be described as a 'routine twins'.

Fig 2: *Participants at the Crystals and twinning Workshop*

Finally, everyone took part in another interactive quiz, this time "Who wants to be a crystallographer?" including the famous *50/50*, *ask the audience* and *phone a friend* options. Not only did the winner obtain the prestige of being an *expert crystallographer*, but also another local triangular chocolate speciality.

CRYSTALS is free for academic and not-for-profit institutions from http://www.xtl.ox.ac.uk/crystals.html. As always, we gained a number of useful ideas from new users at the workshop about how the system could be improved, and we are grateful to everyone who participated. Special thanks go to Lachlan Cranswick who provided much help in working around the hardware related problems that we encountered.

Richard Cooper
Chemical Crystallography Lab, 9 Parks Rd, Oxford
richard.cooper@chem.ox.ac.uk
WWW: http://www.xtl.ox.ac.uk/

Software Fayre during the IUCr Congress, Geneva , August 2002

Thanks to the IUCr Geneva congress organisers, a Software Fayre was held during most of the IUCr Geneva Congress (first and last day excluded). Computers with Windows and Linux were supplied, as well as a projector system for presentations. A variety of talks and software demonstrations were given, including Alan Hewat on the ICSD for Web; Richard Cooper and David Watkin on the Crystals suite; Radovan Cerny on Fox; etc. The following photos should hopefully give a flavour of what went on.

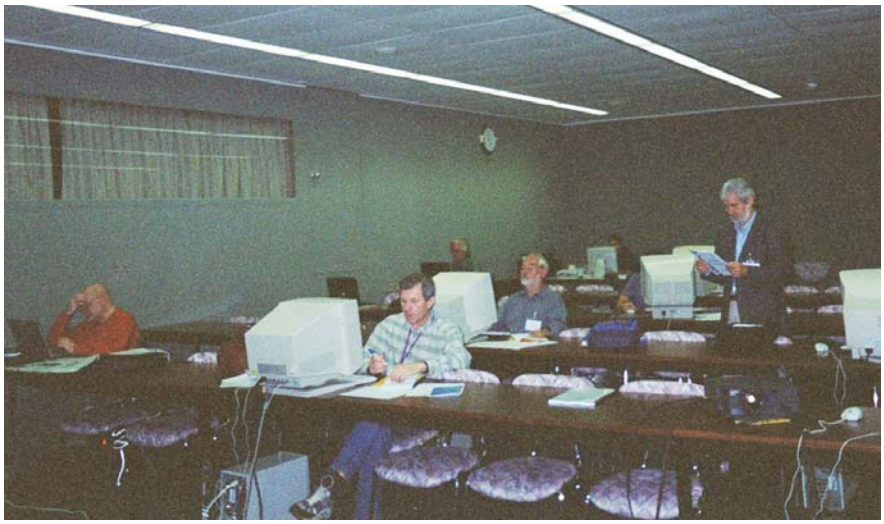


Fig 1: Software authors setting up software and preparing for their demonstrations (Robin Shirley, Paul Beurskens and David Watkin are visible) plus participants trying out the various software in their own time.

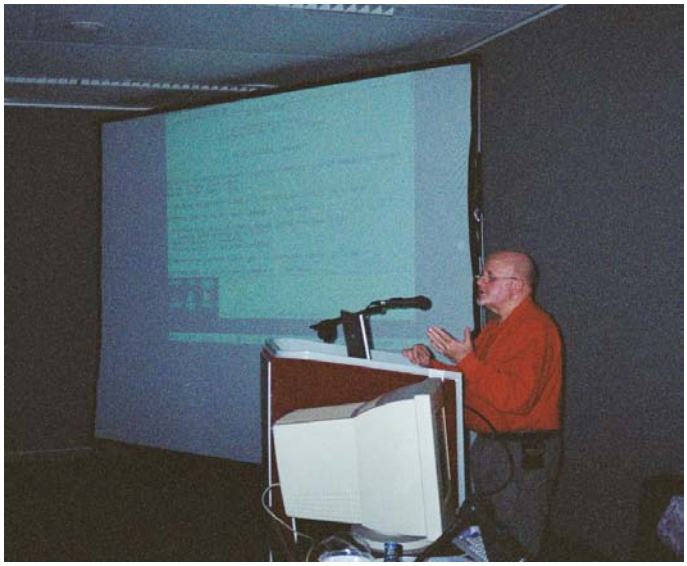


Fig 2: Robin Shirley explaning and demonstrating the Crysfire powder indexing suite (http://www.ccp14.ac.uk/tutorial/crys/)

Fig 3: Luca Lutterotti setting up for his MAUD for Java materials analysis Rietveld software presentation. (http://www.ing.unitn.it/~luttero/) Ralf W. Grosse-Kunstleve tapping away on a laptop in preparation for his demonstration.

Fig 4: Paul Beurskens speaking and Bob Gould controlling the PC (not pictured) during a Dirdif single crystal structure solution demonstration. (http://www-xtal.sci.kun.nl/xtal/documents/software/dirdif.html)

Fig 5: Ralf Grosse-Kunstleve demonstration and explanation of Phenix and the Computational Crystallography Toolbox (CCTBX). (http://cctbx.sourceforge.net/ - http://www.phenix-online.org/ - http://cci.lbl.gov/)

Fig 6: Charles Weeks (standing) and William Furey (seated) demonstrate BnP (SnB-PHASES Interface – SnB for direct methods / dual space structure solution - and PHASES for complete protein phasing). (http://www.hwi.buffalo.edu/BnP/)

Fig 7: Pierre Villars discusses and demonstrates the Pauling File database (http://www.paulingfile.com/).

Future Symposia relevant to Crystallographic Computing

During ACA 2003, 26th to 31st July, Cincinnati, USA: a symposium on *"Future strategies for successful crystallographic computing"*

One half-day session organized by Ross Angel (rangel@vt.edu ; Virginia Tech Crystallography Lab) and David Watkin (david.watkin@chemistry.oxford.ac.uk ; Oxford)

In the past the majority of crystallographic software has been developed within the academic environment and distributed at no cost to the community. As software packages grow and their use becomes more widespread, the costs of supporting and maintaining them also grow. As the financial and legal pressures on academic institutions continue to mount, it is not clear that either Universities or funding agencies will necessarily see the case for continuing their previous subsidy of such operations. Commercial development of software, on the other hand, potentially brings dangers of lack of diversity and responsiveness. With this background, the Commission on Crystallographic Computing of the IUCr held a session at the Geneva Congress to start discussions on what strategies the crystallographic community should pursue to ensure the continuing maintenance and development of high-quality software while at the same time optimizing the needs of the funding agencies, Universities, government research institutions and industry. This session is offered as an opportunity to continue those discussions, and to present views on the various strategies currently being pursued.

Confirmed Speakers:
David Watkin (Oxford)
Lachlan Cranswick (London)
Brian Toby (NIST, Gaithersburg, MD)
Mathias Meyer (Wroclaw, Poland)

More details of the ACA meeting and registration and abstract submission at
http://www.hwi.buffalo.edu/aca/

Ross Angel
Research Professor in Crystallography
Crystallography Laboratory Tel: 540-231-7974
Dept. Geological Sciences Fax: 540-231-3386
Virginia Tech
Blacksburg VA 24060-0420 USA
http://www.crystal.vt.edu/crystal/

ECA-21 2003, 24th to 29th August 2003, Durban, South Africa

A number of sessions relevant to Crystallographic Computing are occurring at the ECA-21 meeting in Durban South Africa.

For more details and registration information, refer to the conference homepage at http://www.sacrs.org.za/ecm21/

- Plenary Lecture 6: presented by Bill David (wifd@isise.rl.ac.uk): *Structure determination from powders - crossing the 100-atom threshold.*

- SIG 3 – 1: *New developments in the structure determination of quasicrystals. Ch*air: G Chapuis, (gervais.chapuis@ic.unil.ch); Co-chair: W Steurer (w.steurer@kristall.erdw.ethz.ch)

- SIG6 – 4: *Automatic structure determination: Challenges for the Future. Chair: A.L. Spek (spea@chem.uu.nl); Co-chair: D Billing (dave@hobbes.gh.wits.ac.za)*

- SIG8 - 5: *Modern programming languages and programming techniques. Can they lead to higher reliability and programmer productivity? Chair L. Cranswick (l.m.d.cranswick@dl.ac.uk); Co-chair: J. Dillen JLMD@land.sun.ac.za*

- SIG9 – 6: *Diffraction image processing and data quality. Chair: E Dodson (ccp4@yorvic.york.ac.uk); Co-chair: S Parsons (s.parsons@ed.ac.uk)*

- SIG9 – 7: *Algorithms of the future; Chair: D Watkin. (david.watkin@chem.ox.ac.uk); Co-chair: A Urzhumtsev (sacha@lcm3b.uhp-nancy.fr)*

- SIG9 – 8: *Indexing Powder Diffraction Patterns: An opportunity for new heuristic and global optimisation methods Chair: R Shirley (r.shirley@surrey.ac.uk); Co-chair: C Rademeyer (cor@spl.setpoint.co.za)*

Free one day Single Crystal and Powder Diffraction Crystallographic Software Workshop as part of ECA 2003 in Durban, South Africa

With a main theme of "Crystallographic Software Wizardry in Single Crystal and Powder Diffraction", and thanks to the conference organisers, there will be a free one day single crystal and powder diffraction software workshop on Sunday 24th August 2003 (at the start of the ECA conference). The emphasis will be on practical 20 minute "power user" examples of features within the software that casual users may not appreciate. The workshop will take place in the same facility as the conference, that being the International Conference Centre in Durban, South Africa.

The workshop webpage, with links to the presenters and the software being presented is at:

http://www.ccp14.ac.uk/projects/ecm21-durban2003/

The morning will consist of the powder diffraction session. This will include: use of the ICDD database; powder indexing of large volume cells (including proteins); powder indexing of impure samples using whole profile methods; structure solution using direct methods and EXPO; structure solution using real space methods and the FOX software; Rietveld refinement of complex inorganic materials using Fullprof; Rietveld Structure Refinement of protein powder diffraction data using GSAS

The afternoon will then be on single crystal methods. Topics include: Using CCDs for visually finding tricky cells, supercells and incommensurate cells; Advanced absorption correction options using Platon and Euhedral; Data processing of Bruker and Nonius CCD data using the WinGX Single Crystal suite, incorporating Sortav; SnB (Shake-n-Bake) direct methods software; Dirdif - fragment searching to solve structures that direct methods won't; Crystals to CCDC Mogul geometry validation; and non-trivial applications of Platon, Addsym and intra/inter-molecular validation

Speakers who are presently listed at time of writing include (for Powder Diffraction): Brian O'Connor, Robin Shirley, Armel Le Bail, Radovan Cerny, Mauro Bortolotti, Luca Lutterotti, Juan Rodriguez-Carvajal and Jon Wright. (Single Crystal): Martin Lutz, Louis Farrugia, Charles Weeks, Bob Gould, David Watkin and Ton Spek.

While the workshop is free, places are limited. People attending ECM-21 and who wish to also go to the software workshop are requested to E-mail their interest in attending to confcall@yebo.co.za **and** also Cc a copy to Lachlan Cranswick at l.m.d.cranswick@dl.ac.uk.

Lachlan M. D. Cranswick
CCP14 - School of Crystallography,
Birkbeck, University of London,
Malet Street, Bloomsbury,
WC1E 7HX, London, UK
Tel: (+44) 020 7631 6850
Fax: (+44) 020 7631 6803
E-mail: l.m.d.cranswick@dl.ac.uk
WWW: http://www.ccp14.ac.uk

Call for Contributions to the Next CompComm Newsletter

The next issue of the Compcomm Newsletter is expected to appear around July of 2003 with the primary theme still to be determined. If no-one is else is co-opted, it will be edited by Lachlan Cranswick.

Contributions would be greatly appreciated on matters of interest to the crystallographic computing community, e.g. meeting reports, future meetings, developments in software, algorithms, coding, programming languages, techniques and news of general interest.
Please send articles and suggestions directly to the editor.

Lachlan M.D. Cranswick
CCP14 - School of Crystallography,
Birkbeck, University of London,
Malet Street, Bloomsbury,
WC1E 7HX, London, UK.
E-mail: **L.Cranswick@dl.ac.uk**