SUPerlattice REfinement from X-rays

(SUPREX)

The SUPREX program incorporates the kinematical and optical formalisms described in the paper *Structural Refinement of Superlattices from X-ray Diffraction* by Eric E. Fullerton, Ivan K. Schuller, H. Vanderstraeten and Y. Bruynseraede, Phys. Rev. B. 45, 9292 (1992). However, in order to use this program as a *research tool*, it has to allow for (i) an easy exploration of the influence of parameters on the x-ray diffraction profiles, and (ii) a systematic comparison between the model and the experimental results. Furthermore, it must be possible to implement different models, which implies that the code has to be easy to read and modify.

A convenient user interface and the application of *object-oriented programming* techniques can fulfill these requirements.

In the first section, we will introduce the basic concepts of object-oriented programming. An example taken from the SUPREX program will show how easy readability and expandability is achieved. The second section gives an overview of the capabilities of the SUPREX program. The last section contains the actual manual.

1. Object-Oriented Programming.

Object-oriented programming is a programming technique mainly developed in the 80's and became the standard technique during the last few years. We do not want to explore this methodology in full detail here. Rather, we will introduce the basic concepts and the advantages of the approach. Introductory treatments can be found in various articles in *Byte* and *PC Magazine* (Tazelaar, 1989; Pountain, 1990; Duntemann & Marinacci, 1990, Rubenking, 1990) and in the object-oriented programming guide of the language we used, Turbo Pascal 5.5. Extensive treatments can be found in books written by, for instance, Goldberg & Robson (1983) and Meyer (1988).

To clarify the main concepts on an object-oriented programming language, we give here parts of three objects that are defined in the SUPREX program:

```
singlerec =     record
        fixed:   boolean;
        y1, y2: single;
end;
arrsinglerec = array[1..nr_fitpara] of singlerec;
models = object (characfit)
        fromitt ;        boolean;
        procedure calcydy; virtual;
        procedure calculate;
        procedure curfit;
end; {models}

optmat = record
        scatf, dens, d, n, dsigma: singlerec;
end;
optical = object (models
        mat: array[1..2] of optmat;
        lambda, dint, csigma: singlerec;
        scatfsub, denssub, scatftop, denstop, dtop, trwidth: singlerec;
        trsteps, m, rowint, rowlat, backgr, normal: singlerec;
        procedure calcydy; virtual;
end; {optical}

lorentzfunction = record
he, av, wi, po : singlerec;
end;
profile = object (models)
        lorentz : array {1..maxprofiles] of lorentzfunction;
        backgr, slope : singlerec;
        procedure calcydy; virtual;
        end; {profile}
```

The formal parameters in the procedures and functions were omitted for clarity.

1. OBJECT-ORIENTED PROGRAMMING

The keyword object denotes a collection of data fields *and* the *methods* that act on these fields. This is the concept of *encapsulation*. Object-oriented programming regards a program as an interaction between these *objects*.

The object **models** inherit all the data fields and methods from the object **characfit** (not shown). This means that this object has the same data fields and can use the methods of **characfit**, among others. This introduces the concept of *inheritance*.

The method **calculate** is an iterator, i.e. it generates a number of different profiles by linearly incrementing the parameters defining a model. **Curfit** fits the model with the experimental profile. Both routines use the *virtual* method (as indicated by the keyword virtual) **calcydy**. The actual models (optical, kinematical,…) are defined in objects that inherit from models the methods **calculate** and **curfit**. The example gives the optical object, which implements the optical formalism, and profile, a combination of Lorentz functions. These objects overwrite the virtual method **calcydy**, which now contains the appropriate routines to calculate the model, but do not repeat the iterator or the fitting algorithm (since they are inherited them). Through the technique of *late binding*, **calculate** and **curfit** know which of the different possible **calcydy** methods of the descendants it has to use. This is the concept of *polymorphism*.

This approach makes it trivial to add a new model to the program. Only a descendant of model has to be defined that declares the parameters defining the model, and implements the functional relationship between these parameters (in **calcydy**). The iterator and fitting algorithm are immediately available (in fact, these routines don't even have to be recompiled).

This explains why object-oriented programming becomes the standard software development methodology: it provides very good maintainability, reusability and extensibility.

The SUPREX program has been written in Turbo Pascal, version 5.5. It has full object-oriented capabilities, provides a good set of graphical routines, double and extended precision and supports a coprocessor. Fortran does produce faster code for numerical computations, but this advantage is outweighed by the lack of support for writing and modifying a large and complex program like the SUPREX program.

2. The SUPREX program

The SUPREX program allows the calculations and fitting of different models, and provides data processing capabilities, file operations and graphical output to the screen and the printer. All these functions are easily available through menus, logically organized in a tree. Figure 1 shows the menu-tree of the current version of the program. We will give here an overview of only those keywords that are of

immediate importance for the physics involved. Instructions on how to use the program are given in section 3.

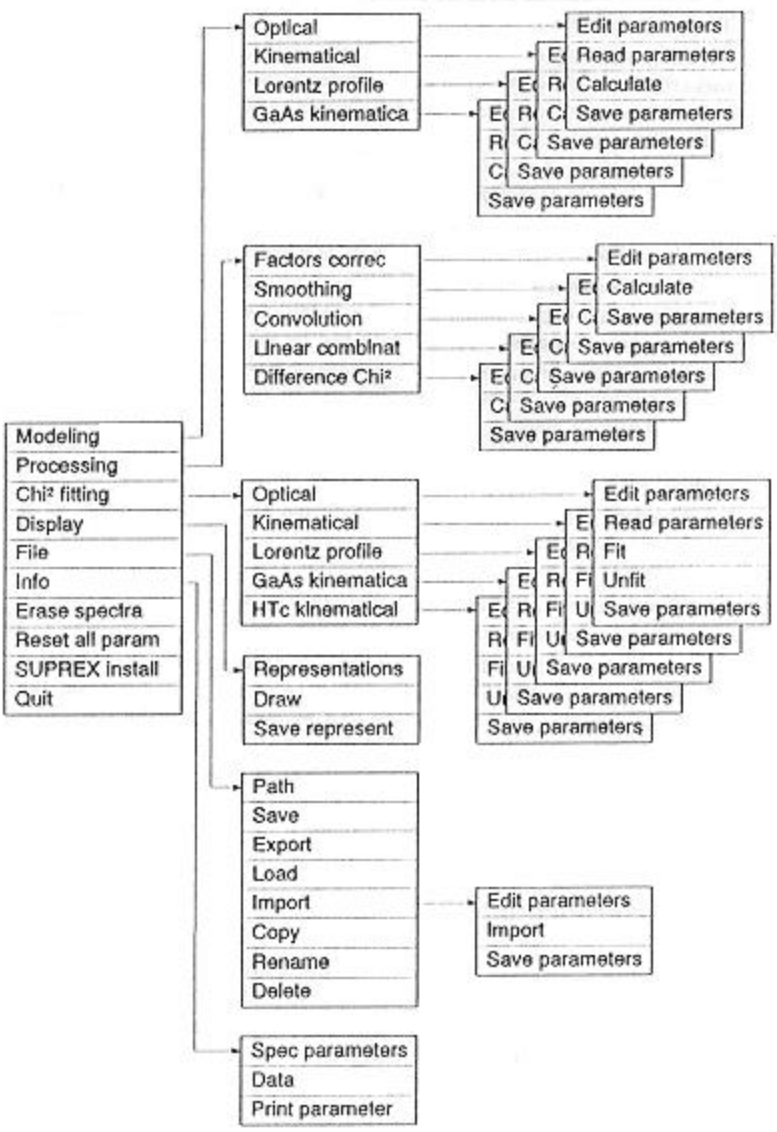

Figure 1: All functions provided by the SUPREX program are made available to the user through this menu-tree.

The kinematical formalism is calculated as $a(I+1_1+I_2) +b$, with a a normalization factor, I the diffracted intensity (Fullerton et. al. (1991), equation 7) I_1 and I_2 modified Lorentz functions (see further) and b a constant background. The optical formalism is calculated as $aI+b$, I the diffracted intensity as defined by equation 18 of Fullerton, et al. (1991). The kinematical formalism for a GaAs/AlAs superlattice (Schuller et. al., 1990) is implemented analogous to the kinematical formalism.

A set of modified Lorentz functions can be calculated or fitted to an experimental profile. A modified Lorentz function is defined as (Howard & Snyder, 1983):

$$l(x) = h(1 + \frac{x - \bar{x}}{w/2})^2)^{-p},$$

With h the height, $\bar{x}$ the mean, w the width (not at half maximum) and p is 1 for a true Lorentzian and 1.5 for an intermediate Lorentzian. The set of Lorentz functions is basically used to estimate peak positions and widths.

The parameters that determine these models can be edited, read from other profiles, and saved to file. It is possible to calculate a set of profiles by linearly incrementing a set of parameters up to a second set. The functions **Read parameters** and **Unfit** allow the use of the result of a calculation or a previous fit as starting parameters for the next fit.

We used the *Marquardt* non-linear fitting algorithm (Bevington, 1969, pp. 237-239), which minimizes the χ^2 difference between a model I_c and the measured intensities I_m:

$$x^2 = \sum_{i=1}^{Npnts} \left(\frac{I_c(i) - I_m(i)}{I_m(i)^2} \right)^2$$

where N_{pnts} is the number of points in the profile and ε an exponent defining the weighting factor of each point. The weighting factor is generally given by the uncertainty of the measured intensities, which is in Gaussian statistics given by the square root of the number of counts. This corresponds to $\varepsilon = 0.5$.

The processing capabilities include the application of correction factors (the Lorentz polarization factor, the absorption correction and geometrical factors (Fullerton et. al., 1991, equation 21), a smoothing algorithm (Press et. al., 1988, pp. 495-497) and a convolution/deconvolution algorithm (Press, et. al., 199, pp. 407-413). The lst two algorithms use the Fast Fourier transform, which can cause problems for very sharp diffraction peaks. A combination of different profiles and the χ^2 difference (equation 2) of two profiles can be calculated.

The diffraction profiles can be displayed on screen in two dimensions, with linear and logarithmic axes and eventually with a difference curve. A three-dimensional view of the data can be selected. The perspective view (Houthuys, 1986, pp. 14-20) is always such that no part of the graph is ever clipped (Vanderstraeten, 1989, pp. 26-29). Hidden lines can be removed using a floating horizon algorithm (Rogers, 1985, pp. 191-205). A contour plot can also be generated (Diamond, 1982). Graphics output is done with a screen dump.

The data can be written to and from file in binary format or in ASCII format. Elementary file operations are provided from within the program.

The parameters defining a profile and the function values can be printed on screen or on printer.

Combining all these features into one program ensures that the user can develop a good feeling for the influence of the parameters on the diffraction profiles, and that model and experiment can be compared efficiently. The latter has been done for a variety of multilayered systems, and the results are discussed by Fullerton et. al. (1991) and Schuller et. al. (1991).

3. Reference Manual

The SUPREX program, version 1.0 is a program used for the REFinement of SUperlattices from X-ray diffraction profiles. The physics incorporated in it will not be treated here. This manual only offers technical information on how to use the program.

3.1 Notation and Conventions

All text that is taken from the screen is written in non-proportional characters. Keystrokes are written as ,<'key'>. 'key' stands for the word, character or symbol on that key of the keyboard. <'key1'> <'key2'> means: hold 'key1' while 'key2' is pressed.

Exceptional circumstances are signaled by error messages. Most of these are self-explanatory: the reason why they appear and the remedy against it are clear from the message itself, from the physics behind it, or from the manual. We will therefore not list all possible messages here.

3.2 Installation and Startup

The program runs on an IBM compatible system with 640 kB memory, a VGA graphics adapter and a math coprocessor. The operating system has to be DOS or higher. The files **SUPREX.BAT,X_MAIN.EXE, SUPREX.PAR, SUPREX2.DAT, SUPREX3.SAT, EGAVGA.BGI, LITT.CHR, SANS.CHR, RMGRAF.COM, X_PSCR24.COM** and **X_PSCRHP.COM** are needed to run the program. The program will use between 10 kB and 535 kB temporary space on the J drive of the system. If this drive is not available, the line **LASTDRIVE=J** has to be included in the **CONFIG.SYS** file and *path1***SUBST J:** ***path2*** as the first line in the file **SUPREX.BAT**, with *path1* the path where the **SUBST DOS** routine is located, and *path2* the path where the temporary files may be located (preferably a virtual disk).

The supported printers for a graphical screen dump and/or a printout of parameters are the Epson LQ500 and HPLaserJet II (and compatibles). The proper printer must already be selected before startup by including the line **X_PSCR24**

=G14 (Epson LQ500) or **X-PSCRHP = G13** (HP LaserJetII) before the call to the main program (**X_MAIN**) in the batch file **SUPREX.BAT.**

The program is loaded by executing SUPREX from the command line, with the screen shown in Figure 2 as a result.

3.3 The Users Interface

The users interface consists of all components that are used for the communication between the user and the program, and they are listed in Figure 2.

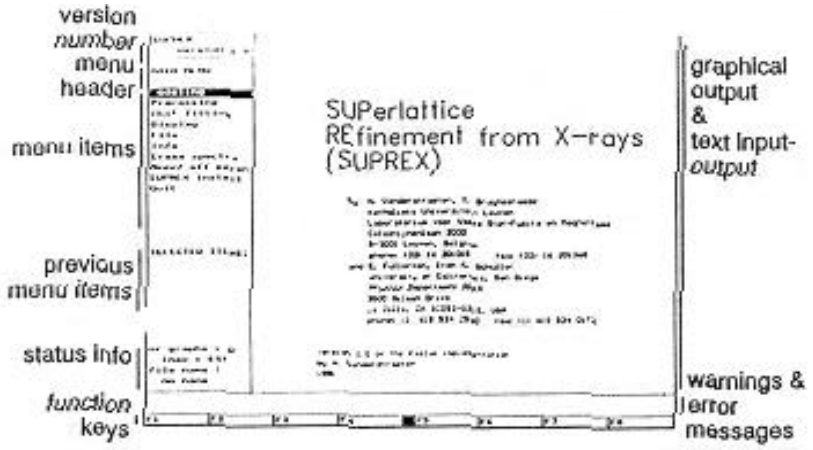

Figure 2: The initial screen of the SUPREX program. The designation of the different fields is shown.

A menu-item is selected by pressing the key corresponding to the first highlighted letter, or by highlighting the item using the cursor keys and pressing <Enter>. The result is a menu or an action. The complete menu-tree is shown in Figure 1. Each menu, except for the main menu, has also an implicit command, the <Esc> key, that allows to go one menu up.

An action often requires the input of text (see for instance the Edit parameters items in Figure 1). The conventions given here apply to *all* cases where input of text is required. Text is always input on the cursor in overwrite (white cursor) or insert (blue cursor) mode. <Ins> toggles between these modes. If any of the editing keys < ←>, <→>, <Ins>, <Del>, <←>, <Home> or <End>is used as the very first keystroke for a string, the old string is preserved, otherwise it is overwritten. With <←> and <→> the cursor is put on the first or after the last character. <Del> deletes the character on the cursor, <←> deletes the character before the cursor.

The keys <↑> and <↓> restore the contents of the field, and move the cursor one line up or down (and eventually wrap around).

<Enter> interprets the input. If the rules valid for that field are violated, the input is rejected, the field restored, an error message displayed and input can start again on the same line. Otherwise, the value is accepted, the field updated and the cursor advances one line.

<Esc> restores the field if the text on the current line is modified, and quits the menu-command that gave rise to the editing session otherwise.

Sometimes the program displays two values on one line, separated by a slash (like x/y). Valid inputs are for example 20, 20/, /25 and 20/25. The first two are equivalent and change only the first value, the third changes the second value and the fourth changes both values.

Several File commands require one or two file names. There are two possibilities to input a file name. After a particular File command is selected a list is shown of the files corresponding to the current path (as set up by File – Path). With the keys < ←>, <→>, <↑> and <↓> <PgUp>, <PgDn>, <Home> and <End> a highlighted bar can be moved over this list, or if the bar cannot be moved anymore in that particular direction, the file list is moved. <Enter> selects the file name on which the highlighted bar is positioned.

If one doesn't want to select a file name from the list, <Esc> quits from the list, and the file name(s) can be typed in. The general rules for input of text are bold from now on.

While interpreting the file name, all missing components (path, name and extension) are taken from the path name (set by **File – Path** and displayed above the list of file names). If the resulting full file name is not valid (the normal DOS conventions hold), an error message is displayed. The input restarts from the file list.

3.4 The Menu-commands

Basically, the program is a manager of a *group of graphs* (by graph we mean a single set of function values with one common x-axis). All necessary operations on this group of graphs (inserting, modifying, discarding, displaying,…) are provided through the menu-commands (figure 1).

In general, the symbols used in this thesis for the parameters are retained in the program, or, if this was not possible, a meaningful name was used. We will therefore only explain those parameters that are introduced for the first time.

The command **Edit parameters** allows to edit the parameters. **Save parameters** saves the current values of the particular set of parameters to a file. When the program is started, all parameters are initialized to the values stored in this file. The same can be accomplished from within the program with the command **Reset**

all parameters. **Read parameters** allows to set the parameters of a particular model to the values contained in another graph, which has to be calculated (with Modeling of Chi2 fitting), with the same model.

Calculate, Fit or **Import** perform the actual operation with all parameters in their current state. These commands also append one or more graphs to the current group of graphs.

For references to algorithms, we refer to section 2.

Modeling. This group of commands allows to calculate a set of graphs, where the parameters linearly increment from their starting values to their end values, in a number of steps that can be specified. The parameters having the letter **F** or **V** before them can be made variable.

The input of **V** or / makes the parameter a variable, and start and stop values are displayed separated by a slash. **F** makes the parameter fixed, and only one value is displayed. Entering x (some value) changes only the first value, as does **x/**, which also makes the parameter a variable. **/y** changes the last value and **x/y** changes both. The later two inputs also force the parameter to be a variable.

Some caution is necessary in converting thicknesses from one model to another. For the kinematical model, we have, as before, the modulation length
$\Lambda = t_A + t_B + 2a$ with t the thickness of a layer and a the interface distance. Note, however, that $t = (n - 1)d$ for *both* a crystalline layer (with n and d relevant parameters) and an amorphous layer (t as a relevant parameter). The continuous thickness fluctuations on an amorphous layer are given by $\sqrt{n}\sigma$. For the optical model and the GaAs/AlAs superlattice kinematical model, we have $t = nd$.

Processing. The different correction factors that can be applied are defined in section 2, and χ^2 is defined in equation 2.

The full width window parameter of the command **Smoothing** is in units of number of points, i.e. independent of the x-axis. The same parameter of the command **Convolution** is in the same units of the current x-axis, and gives the full width at half maximum of a Gaussian profile with which the diffraction profile is convoluted or deconvoluted.

Chi2 fitting. With these commands, a specific model is fitted to the *last* graph of the group of graphs currently in memory. Although all parameters again have a letter **F** or **V** before them, not all can be made variable (an input that attempts to make these parameters variable is refused). The allowed inputs to make a parameter fixed or variable are the same as for **Modeling**. The second value is in this case the increment Δa for the numerical derivatives used by the fitting algorithm:

$$f'(a) = \frac{f(a + \Delta a) - f(a)}{\Delta a}$$

A value of 0.001 or 0.0001 is normally used. The program crashes if a derivative becomes zero (the program does not check for it since it seldom occurs and slows down the calculation too much). In this case, a value of 0.1 is more suitable.

The command **Fit** starts the fitting, and appends first a graph calculated with the starting parameters to the group of graphs, and subsequently all iterations. **Unfit** discards all these graphs again, but allows to use the parameters of the last iteration as the starting parameters for the next fit.

Display. This command allows to draw the graphs currently in memory on screen (and then on printer). Three possible representations can be selected: a contour plot (the number of contour lines can be selected), a two-dimensional plot of at more two graphs at a time and a three-dimensional plot. In the latter representation, the point of view and direction of view can be chosen relative to a 1 by 1 box that encloses all graphs. Hidden lines can be removed. Figure 3 shows a screen dump of a two-dimensional representation, with a difference curve of the two graphs.

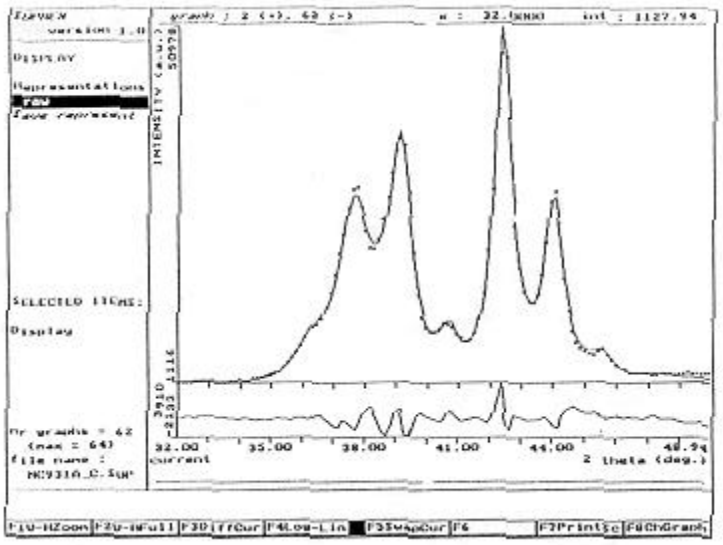

Figure 3: Screen dump of a two-dimensional representation.

All representations have various function keys assigned (figure 3) with which the plot can be zoomed in or out, the graphs to be displayed can be selected, a logarithmic or linear intensity axis can be selected, and a screen dump can be made. The screen dump prints the screen, and the parameters of the graphs on the display. It is important that the same printer is selected (with **Suprex install**) as the one selected during startup (see section 3.2).

File. These commands provide various file operations. **Path** sets the path, and is important when selecting file names (see section 3.3). **Save** saves the group of graphs to file in a compact binary format, while **Export** saves them in ASCII format (one graph per column, the first line contains information about the x-axis). **Load** loads (or appends) a group of graphs from a file previously saved with the SUPREX program, version 1.0. **Import** imports ASCII codes, where the original x-axis and the current x-axis have to be specified. A separate specification of these x-axes allows for skipping the first or last points, and selection one point out of every *n*.

Info. These commands allow to print the parameters of the graphs on screen (**Spec parameters**) or on printer (**Print parameters**). The type of printer is selected with **SUPREX install**. **Data** allows to view and/or modify the function values. To modify a value, move the highlighted bar on that value (analogous to selecting a file name, see section A.3) and press <Enter>. The familiar cursor appears. The same conventions as for editing a value (see section 3.3) hold.

Erase spectra. This command discards a range of graphs from the group.

Reset all parameters. With this command, all parameters are initialized to the values stored on file. (see **Save parameters**).

Suprex install Installs a HPLaserJet II or Epson LQ500 for the printout of parameters (see **Draw** and **Info**).

Quit. This command quits from the program, but allows to save the group of graphs first if this is not yet done.

REFERENCES

Bevington, P.R. (1969) *Data Reduction and Error Analysis for the Physical Sciences*, New York: McGraw-Hill.

Diamond, R. (1982). In *Computational Crystallography*, edited by D. Sayre, pp. 266-273. Oxford: Clarendon Press.

Duntemann, J. & Marinacci, C. (1990). *Byte* **15**(4), 261-284.

Fullerton, E.E., Schuller, I.K., Vanderstraeten, H., and Bruynseraede, Y. Phys. Rev. B. **45**, 9292 (1992).

Goldberg, A. & Robson, D. (1983). *Smalltalk-80 The Language and its Implementation.* Reading: Addison-Wesley.

Houthuys, P. (1986). *Een nieuw algorithme vor climinatic van erborgen lijnen en vlakken opererend in de voorwerpsruimle*. Ph.D. Thesis, Katholieke Universiteit Leuven (unpublished).

Howard, S.D. & Snyder, R.L. (1983). *Adv. In X-ray Analysis* **26**, 73-80.*Object-oriened programming guide, Turbo Pascal 5.5* (1988). Borland International, Inc.

Pountain, D. (1990). *Byte* **15**(2) 257-264.

Press, W. H., Flannery, B.P., Teukolsky, S.A. & Vetterling, W.T. (1988). *Numerical Recipes, The Art of Scientific Computing*. Cambridge: Cambridge Univ. Press.

Rogers, D.F. (1985). *Procedural elements for computer graphics*. New York: McGraw-Hill.

Rubenking, N.J. (1990). *Pc Magazine* **9**(3), 263-281.

Schuller, I.K., Grimsditch, M., Chambers, F., Devane, G., Vanderstraeten, H., Neerinck, D., Locquet, J.P., Bruynseraede, Y., (1990). *Phys. Rev. Lett.* **65**, 1235-1238.

Schuller, I.K., Fullerton, E.E., Vanderstraeten, H., and Bruynseraede, Y., (1991). Mat. Res. Soc. Symp. Proc. Accepted

Tazelaar, K.M., (editor) (1989). *Byte* **14**(3), 228-271.

Vanderstraeten, H., (1989) *Ondersteunende programmatuur voor de scanning tunneling microscoop.* Masters thesis, Katholieke Universiteit Leuven (unpublished).