Ultramicroscopy 6 (1981) 343–358
North-Holland Publishing Company

SPIDER—A MODULAR SOFTWARE SYSTEM FOR ELECTRON IMAGE PROCESSING

Joachim FRANK, Brian SHIMKIN * and Helen DOWSE **

Division of Laboratories and Research, New York State Department of Health, Albany, New York 12201, USA

Received 13 April 1981; revised 20 May 1981

The image-processing system SPIDER has been designed to operate on a minicomputer in a multiuser environment. SPIDER, which can be run either interactive or batch mode, makes a wide range of operations (including contrast enhancement, Fourier filtration, correlation averaging, and three-dimensional reconstruction) available for analysis of electron micrographs. The command language supports a hierarchical calling structure, branching commands, and DO-loops similar to those of FORTRAN.

1. Introduction

1.1. Electron image processing

Image processing has become increasingly important as a tool for interpretation and enhancement of electron micrographs. This is evident from recent reviews, which cover a wide range of applications [1–5]. The increased availability of minicomputers has now put the many image-processing schemes proposed over the years within the reach of electron microscopy groups.

Following Smith's scheme [6], the applications can be roughly grouped into three categories: restoration, image enhancement, and three-dimensional reconstruction.

Restoration is the attempt to eliminate distortions introduced by the objective lens of the electron microscope. Certain approximations in bright field electron microscopy of weakly scattering objects lead to a linear system description of image formation [7,8], which allows the object's potential distribution to be computed from the experimental data [9–11].

Image enhancement is a summary term denoting point-for-point operations, or operations involving local neighbors of image points, which aid in visual interpretation of images. Quantitative methods of image averaging aimed at enhancing the signal-to-noise ratio may also be included in this category. The importance of these methods in specimen-preserving high-resolution electron microscopy has been widely recognized since Unwin and Henderson's study of the purple membrane protein [12]. More recently, averaging methods for single particles have been developed [13–15], extending low-dose investigation to a larger class of biological specimens. Multivariate statistical techniques promise to be a powerful tool in structural analysis of single molecules and crystals [16,17].

Three-dimensional reconstruction is reconstruction, by various procedures, of an object's mass distribution in three dimensions from a series of micrographs showing the object in different views [4,18,19]. In the special case of an object having high symmetry, a single micrograph may be sufficient [18].

To represent an image recorded on a photographic film or plate in digital form, one must scan the film or plate on a computer-controlled digital microdensitometer, where optical density values are read on a raster, converted into digital form, and written onto a storage medium (tape, disk). In the reverse operation, often combined with the scanning function in the same instrument, digitally recorded images are displayed as a fine raster of points on a photographic film.

The increased flexibility of digital processing can best be realized when the system is designed in a modular way (fig. 1). Mathematical operations on images or sets of images are broken down into the simplest steps involving an entire image. These steps are standardized to permit their use in more than one context. In each step an input image (existing as a file on a mass storage medium) is processed, and an output file is created with the same format; the output file can then

* Present address: Gerber Systems Technology, Post Office Box 905, South Windsor, Connecticut 06074, USA.
** Present address: General Electric Company, 1 River Road, Schenectady, New York 12345, USA.

0304-3991/81/0000–0000/$02.50

be used as input for a subsequent operation. List and display operations make it possible to check and debug each processing step separately. More involved schemes can be built from these elementary operations by simply

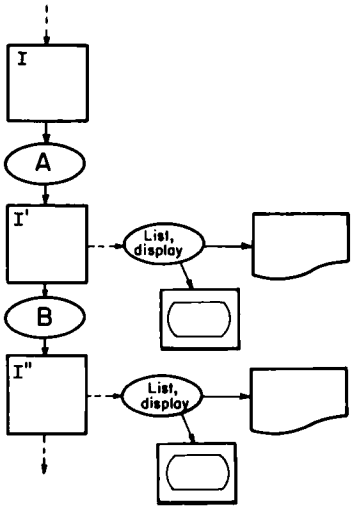


Fig. 1. Scheme of the data flow in a modular image processing system. Operation A creates image I' from input image I. At the end of operation A, I' resides along with I on a mass storage medium and can be listed and displayed before being subjected to step B.

stringing them together in a virtually infinite number of ways. The first comprehensive image-processing system with this design philosophy, VICAR [20], was developed at the Jet Propulsion Laboratory in Pasadena in the 1960s.

If a fast graytone display device is available, the decision on which processing step to choose next in a series of operations can be based on the result of the immediately preceding step, making the system interactive.

1.2. Existing modular software systems

The need for a modular design in electron image-processing systems has been recognized by a number of groups [6,21–30] and has been answered in various ways, depending on the particular computer installations of the originating laboratories. Of the software systems listed in table 1, IMPROC [27], MDPP [6], and EM [29] are written for large machines, while SEMPER [25], SPIDER [21], MIRAGE [22], PIC [28], and IMAGIC [30] are designed for minicomputers with 64 Kb storage or less.

Some correspondences exist between SPIDER and SEMPER [15,26] in the command language structure and in the system design. However, SEMPER is distinguished by freer rules of format and register assignments, while SPIDER has greater flexibility in the dynamic features of the language.

One difference between these two systems is that SPIDER supports a full dialogue with the user by printing out solicitation messages on the terminal, whereas SEMPER only returns results or error messages. Our experience is that solicitation messages in operations (as well as user-created solicitation messages in procedures; see section 3.6) are an invaluable aid in processing with a modular software system. With so many operations available, it is very difficult to memorize all of the various input sequences required. In contrast, the solicitation messages help even inexperienced users in interactive SPIDER sessions to comprehend and use the processing system within a short time. Interactive sessions also provide training in extended use of the language in procedure and batch command files.

The most comprehensive program package for three-dimensional reconstruction and various two-dimensional filtrations, which was developed at the Medical Research Council in Cambridge [31], is not a modular system in the sense described here. Rather it is a collection of programs that lacks the cohesiveness and versatility of other systems listed in table 1.

Development of a multipurpose software system is a very laborious task, often a byproduct of research activity. The successful design and implementation of such a system may therefore precede its publication by several years. For instance, EM existed in its basic form in 1970, 10 years before it was first described in the literature [29].

1.3. SPIDER hardware configuration and design considerations

SPIDER (System for Processing of Image Data in Electron microscopy and Related fields) was developed at the New York State Department of Health as a general-user facility for a wide range of applications associated with high-voltage and conventional-voltage microscopes. Among these applications are enhancement of low-contrast electron micrographs [32], computer filtration [33], single-molecule averaging [13–15], three-dimensional reconstruction [34], and evaluation of electron diffraction information.

Table I
Modular image-processing systems in electron microscopy [a]

Name	Ref.	Origin	Machine [b]	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)
EM	[29]	Max-Planck-Inst. Biochemie, Martinsried, FRG	IBM 360 Siemens 4004 VAX-11/780	x		x	x	x	x				x	x	x	
IMAGIC	[30]	Biochem. Lab. Univ. Groningen, Netherlands	NORD 10	x	x	x	x	x	x	x				x	x	x
IMPROC	[27]	Cavendish Lab., Cambridge, UK	IBM 370/165	x	x	x	x	x						x	x	
MDPP	[6]	Biozentrum Univ., Basel, Switzerland	IBM 360/370	x	x	x	x	x	x	x		x		x		
MIRAGE	[22]	CENG, Grenoble, France	Intertechnique Pherimat S	x	x	x	x	x				P		x	x	
OIDIPS	[23]	Dept. Appl. Phys., Osaka Univ., Japan	CEC-555 Interdata 7/32				x	x						x		
PIC	[28]	Natl. Inst. Health, Bethesda, MD, USA	PDP-11/70		x	x	x	x						x	x	
SEMPER	[25]	Cavendish Lab., Cambridge, UK	PDP-8/E IBM 370/165 ... [c]	x	x	x	x	x			x		x	x	x	
SPIDER	[21]	New York State Dept. of Health, Albany, NY, USA	PDP-11/45 VAX-11/780	x	x		x	x	x		x	x	P		x	x

(1) Extended language = capability to handle DO-loops and procedures (macros) on the command language level.
(2) Plane lattice averaging.
(3) Rotational filtration.
(4) Image statistics.
(5) Correlation alignment.
(6) Handling of 3D info = three-dimensional file storage, general modular procedures for extraction sections from real space or Fourier space volumes.
(7) Helical reconstruction.
(8) ART reconstruction.
(9) Tilted lattice reconstruction.
(10) Tilted particle reconstruction.
(11) Image restoration.
(12) Interactive display.
(13) Multivariate statistics.

P = in preparation.

[a] The table presents the results of a survey sent to the authors of the systems listed.
[b] Where more than one installation is listed, the first is the one for which the system was originally developed.
[c] Ten more machines listed by the author.

The hardware consists of a Digital Equipment Corporation (DEC) PDP-11/45 computer, two 1600-bpi tape units, two 176-Mb disk units, a VERSATEC printer/plotter, a line printer, a Princeton Electronics Product (PEP) 801 graytone storage display system, and a Perkin Elmer PDS 1010A flatbed microdensitometer. The microdensitometer is run by a DEC PDP-11/05 computer operating independently of the PDP-11/45 and equipped with a tape unit and a film-writing option.

Some of the design requirements underlying the SPIDER system were very similar to those spelled out by Smith [6]: that the system be easily accessible to users having limited experience with computers; that all operations be available in a single job; that options for display, listing, and permanent storage of images be available; that installation-dependent features be avoided, where possible; and that the system be capable of processing any number of images with arbitrary sizes (within limits) and formats. Additional restrictions were imposed by our reliance on a small computer in a multiuser environment.

Averaging of single molecules by correlation methods [13–15] requires the processing of a large number of images that are realizations of the same molecule projection. Other applications, such as three-dimensional reconstruction from projections [34,35], also involve a series of files that must be subjected to essentially identical operations. These applications demand an efficient mechanism for defining repeated operations or sets of operations over a series of images. DO-loops that can be nested up to three deep were created in SPIDER for this purpose.

Finally, there were requirements for a hierarchical command structure and for a general mechanism to transfer values from one operation to another and from one run of the program system to another.

All of these requirements have been realized in a way that may be of general interest in the design of modular software systems.

2. System design

2.1. General

The SPIDER system consists of a master task DRIVER and a set of slave tasks, which are run under the DEC RSX11M operating system in a multiuser environment (fig. 2). The slave tasks are started by DRIVER and return control to DRIVER when finished. Each slave task performs a subset of the 120 operations

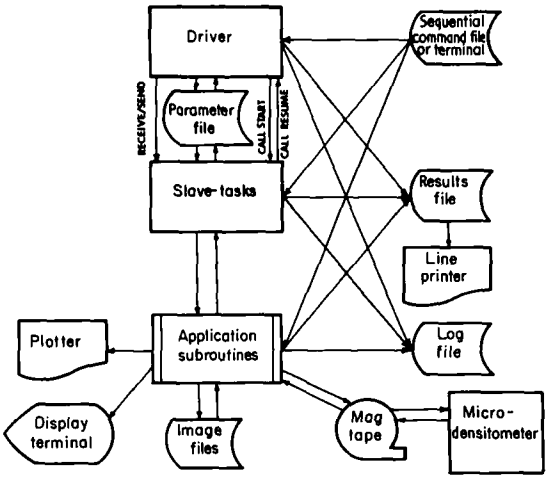


Fig. 2. Schematic representation of SPIDER procesing system [21]. DRIVER and the slave tasks communicate with each other by three means: (1) START/RESUME directives, (2) SEND/RECEIVE directives to pass the name of the current working area and project extension to the slave task, and (3) a parameter containing all information pertinent to the session and the current processing stage. The control sequence for the processing session is either contained in the sequential command (batch mode) or entered directly from the terminal (interactive mode). In the latter case, a log file stores all user input for later reference. (Reproduced with kind permission of the Microscopical Society of Canada.)

now available. A *session* is defined as a set of operations between the start of DRIVER and an end ("EN") command. Each session is distinguished from other parallel sessions by a unique *project code*. The user starts a session by executing DRIVER and specifying the project code and the data library he wishes to access. The image-processing commands are entered either directly on a terminal (interactive operation) or via a sequential command file previously prepared by using the text editor (batch operation). *Procedure files*, a special class of command files which allow run-time replacements, can be invoked interchangeably with the basic commands in either interactive or batch operation.

Upon encountering an operation command (e.g. "RT" for rotate), DRIVER activates the appropriate slave task and suspends itself. Once activated, the slave task performs the operation, soliciting any information required (names of input files, values of processing parameters, etc.) from the user. The slave task then asks for the next operation command and continues processing. When it encounters a command not contained among its operations, the slave task reactivates

DRIVER, passes on to DRIVER the current input line, and exits. DRIVER then continues as above.

The main advantages of this approach are minimization of compile and link time for program changes, minimization of core use, and an open-ended system design. DRIVER communicates with the slave tasks by means of a SEND/RECEIVE message and a sequential parameter file. The SEND/RECEIVE message contains the name of the user's working area and the name of the parameter file. This file, as well as other temporary system-generated files to be mentioned later, has a name unique to the session, so that two or more SPIDER sessions can be run simultaneously. Any information needed by the slave task (the file extension specifying the data library, the current command-file name, file pointers, etc.) is written to the parameter file by DRIVER and read by the slave task upon activation. Before exiting, the slave task updates the parameter file. Communication between DRIVER and the slave tasks, as well as the activation, suspension, and exiting of the various tasks, is transparent to the user.

DRIVER is the only task that retains all information pertinent to the session. It handles all global switches and system-related operations (DO-loops, procedures, etc.), while the slave tasks handle all operations involving data processing (e.g. Fourier transformation, masking, and rotation of images).

2.2. Coding

With a few expectations, all main programs and subroutines are written in FORTRAN IV. Subroutines for system-related functions (such as read/write accesses to the disk, communication with DRIVER, reading of parameter values, opening of files, and updating of statistical information) are standardized, facilitating the implementation of new operations into existing slave tasks or the creation of a new slave task.

2.3. Registers

Up to 100 storage spaces are available for storage and transfer of important values during the SPIDER session. The corresponding array of floating point numbers is passed from DRIVER to the slave tasks and vice versa as part of the parameter file. The storage spaces are called *registers* and are invoked in the command language by the symbols X0...X99, which may take the place of any integer or floating point number expected in the command stream. Registers are thus an important tool for exchange of information between operations within a session.

2.4. Global switches

Global switches are implemented in the system to invoke different modes of operation for the entire session:

(a) Print output can be spooled to the line printer on completion of each operation or on termination of the session.

(b) A trace switch allows the user to follow the progress of a batch run from messages printed on the terminal.

(c) Two modes of error response in the batch operation are available. In one mode an error immediately terminates the session. In the other mode the processing is allowed to go on, skipping to the next intelligible command.

(d) Read-only access to data in other working areas is available on request.

(e) Two versions of each slave task may be activated, depending on the state of a switch. This feature enables testing of modified versions of a task without disrupting the use of the existing system. It also allows efficient use of the available memory in test runs, if one set of the tasks is built with a minimum of buffer space sufficient for small images.

2.5. Image format

Images are stored one record per line as direct-access files on disk. Each density reading is stored as a 4-byte word. Both Perkin Elmer 1010A tapes and Optronics tapes can be read by the system. SPIDER accesses the PDP-11 file directory through special subroutines.

The image file consists of the image data and a certain number of records needed to accommodate additional space for the Fourier transform, a 128-point histogram, and the SPIDER image-processing label. The historgram and part of the image label contain statistical information on the image that, once acquired, is available to any subsequent operation. The status of the statistical information is recorded by flags, which are interrogated in each operation to decide whether the information is already stored or whether it needs to be computed.

Thirteen file formats are used and distinguished in the SPIDER system (table 2). The different types of random-access files are distinguished by label flags, which are checked in each access for consistency with the operation. The format for storage of the Fourier data is consistent with the optimized fast Fourier transformation program by Fraser [36].

In the SPIDER system all file names have the standard form ⟨ABC⟩⟨LMN⟩.⟨EXT⟩, where ⟨ABC⟩ are

Table 2
File formats used in the SPIDER system

Random access, unformatted	Sequential, formatted
Real 2-D	Document
Real 2-D polar	Procedure
Real 3-D	Batch
Fourier 2-D	Log [a]
Fourier 2-D polar	Results [a]
Fourier 3-D	
Nonimage parameter [a]	

[a] Created in each sessions.

three letters specifying a file series and ⟨LMN⟩ is a three-digit number between 001 and 999, specifying the number of the file in the series. ⟨EXT⟩ is the same file extension for the entire session and is used to restrict the access of the image procesing to a data library. Within the session only the first six letters of the file need to be specified, e.g. PIC008 or RES855.

To facilitate the processing of selected files from a file series, a variable file name specification has been created. With PIC00I, for example, the sequence "00I" is replaced by a three-digit number according to the value assigned to the index I in a command-level DO-loop. With RECX10 the three last characters denote a SPIDER register, causing a file name to be constructed from the pre-fix "REC" and a three-digit number stored in the register X10.

2.6. System–user communication

The system–user communication sequence can be summarized as: solicitation–user response–verification –log entry.

The solicitation line specifies the input that is next expected in the command sequence. In the interactive mode this line has an important mnemonic function. In the batch mode the solicitation line is printed out along with the input line to make the processing sequence intelligible.

The user response comes from the terminal or, in the batch mode, from the batch command file. The user input is then echoed in a verifying message to ensure the information accepted by the system corresponds to the input intended by the user. In addition, the opening of any file is accompanied by an explicit verification statement, which includes the complete file name, title, data and time of creation, data type and dimensions, and file disposition.

In the interactive mode the user input is copied into a sequential log file. It can be called up later to recreate the same results in a batch run. Changes of dimensions and parameters can easily be made by editing the log file after the session that created it.

All user input in interactive or batch operation is handled by special subroutines that combine the functions of printing the solicitation message, reading, echoing, and log-keeping. The input of parameter values is either explicit in free format or by reference to the contents of registers filled by previous computations.

3. Command language

At the beginning of the session and upon completing each operation, SPIDER solicits the next command by printing the message "OPERATION:". All possible user responses form a language. The syntactic rules of the language are set by rules of format (e.g., operation commands must be entered left-justified) and by the order of input parameters expected for a given operation. In the interactive mode the solicitation messages serve to enforce the rules of the language. In the batch mode the user must set up an error-free command text with the help of the user's manual and protocols of previous interactive sessions kept in a log file. (For handling of errors, see section 2.4.)

The various types of commands are listed in table 3.

3.1. Basic commands

There are over 100 basic commands relating to a variety of operations (see appendix 1). These commands can be roughly grouped into seven categories (examples are given in parentheses):

Table 3
Command types in SPIDER

Type of command	Example
Basic command	RT
Arithmetic interrogation	SIN(2−1./X10)+X20
Arithmetic assignment	X15=(2**X3)*150.6
Register interrogation	X10
Batch command	B43
Procedure command	PR1
Branching commands	
DO-loops	DO LB1 I=1, 10
labels	LB3
Conditional jumps	IF(X50.GT.4)GOTO LB3
Termination and return	EN, RE

file management (copy, file-information, delete, rename);
image movement and editing (mask, shift, rotate, interpolate, insert);
display (graytone display, contour, profile, list);
contrast enhancement (density stretching, histogram equalization, local averging);
Fourier operations (fast Fourier transform, filtration, cross- and autocorrelation);
three-dimensional reconstruction (project, back-project, stack, unstack);
interface with multivariate statistical analysis programs.

```
RUN DRIVER
(Spider V3 (06/10/80) on 09-Dec-80 at 19:07:20)
.enter project/data code:LMN/DAT
lmn/dat
.operation:  MO
   mo
.output file:  MOD001/TEST PICTURE
mod001.dat
.enter dims (nsam,nrow):  45,50
   45  50
db:[200,050]mod001.dat/test picture
(r)  45  50 created on 09-Dec-80 at 19:12:20 m
.(t)est/(s)ine/(c)irc/(w)ed/(r)an/(g)auss:  W
   w
.operation:  PD
   pd
.input1 file:  MOD001
   mod001.dat
db:[200,050]mod001.dat/test picture
(r)  45  50 created on 09-Dec-80 at 19:12:20 o
.output file:  PAD001/TEST PICTURE PADDED
   pad001.dat
.enter dims(nsam,nrow): 64,64
   64    64
db:[200,050]pad001.dat/test picture padded
(r)  64  64 created on 09-Dec-80 at 19:15:02 n
.average?(y/n),  (c) circular option:  Y
   y
.top left coos:  10,7
    10    7
.operation: FT
      ft
.input file:  PAD001
  pad001.dat
db:[200,050]pad001.dat/test picture padded
(r)   64  64 created on 09-Dec-80 at 19:15:02 o
.operation:  PW
   pw
.input file:  PAD001
    pad001.dat
db:[200,050]pow001.dat/test picture padded
(r)  64  64 created on 09-Dec-80 at 19:24:51 n
.operation:
```

Fig. 3. Example of computer/user dialogue in interactive session. The SPIDER system is run with a project code LMN and data code DAT. The computer messages are in lowercase letters; the user input is in capital letters. In this example a model image MOD001 with dimensions 45, 50 is created ("MO") which contains a density wedge. It is padded ("PD") into a 64×64 image PAD001, with the image average as background. The padded image is subsequently Fourier-transformed ("FT"); this operation overwrites the image file with the Fourier transform. Finally the operation "PW" computes the modulus of the Fourier transform and stores it into POW001. Each open access to a file results in a two-line statement showing the file title, disposition, format and creation data. The first access to the file PAD001 after Fourier transformation shows it to have Fourier format, denoted "(F)".

A letter after the creation date indicates whether the file existed before (o) or is being created (n) or modified (m).

Each operation, when invoked by one of the basic commands, will solicit all necessary information. Normally this comprises the names of input and output files, option specification, and input parameter values pertinent to the operation. In the following examples lower-case letters denote messages printed by the computer, and capitals denote user responses:

·operation: SH
·input file: PIC001
·output file: OUT003
·shift components (x,y): 5,6
·operation:

(Here we have left out the verifying messages.) An example of a complete record of user/system interaction, such as would be recorded on a hardcopy terminal, is shown in fig. 3.

Basic commands may have options or outputs arguments. For instance, 'TRO' invokes the Optronics format option of the tape-read command. PK X10,X11,X12 is equivalent to the instruction "peak search, and put the x, y coordinates and the value of the largest peak into the registers X10, X11, and X12 respectively".

3.2. Arithmetic interrogation (pocket calculator)

Any arithmetic expression involving the five basic operations (+, −, /, *, **) and numbers or SPIDER registers can be evaluated. In addition, the functions SIN, COS, EXP, LOG, SQRT, and PAD can be used. The PAD function returns, for a given argument, the next larger number that can be represented by a power of 2. For instance, PAD(5) = 8, PAD(45) = 64. With this function images can be automatically padded for operations involving radix-2 Fourier transformations (e.g. ref. [37]).

3.3. Arithmetic assignment

Arithmetic expressions appearing on the right-hand side of an assignment statement are evaluated as above, and the result is put into the register appearing on the left-hand side. The assignment statement is thus exactly equivalent to a FORTRAN arithmetic assignment.

Register interrogation

If a register is specified in the command position, its contents are printed on the terminal or line printer.

3.5. Batch command

A batch command has the form B⟨NM⟩, where ⟨NM⟩ is a two-digit number. Such a command switches

the SPIDER input command stream from the terminal to a sequential file named B⟨NM⟩.⟨PRJ⟩, where ⟨PRJ⟩ is the project code, thus terminating the interactive mode of operation. The batch file can contain any sequence of commands and parameter values as they would be entered by the user in the interactive session. The log file, which records each input line in an interactive session, can be used as a batch command file to reproduce the processing sequence.

3.6. Procedure command

The full flexibility of the processing system has been achieved by creating a procedure calling structure. A procedure may be regarded as a batch command sequence in which certain input lines have been left unspecified until execution time. Any file name or input line where one or more numerical constants are expected can be replaced by a substitution line with the general format ?⟨character string⟩?, where ⟨character string⟩ is the desired execution-time solicitation message. Such a line causes DRIVER to fetch the next-scheduled input from the next-higher level of the calling hierarchy, i.e., from the terminal or from a calling batch or procedure command stream.

In the interactive mode the character string has the function of a solicitation message. The sequence of user-system interaction in the execution of a procedure is therefore identical to the sequence followed in the execution of a basic command. In a listing of the procedure all substitution strings stand out from the rest of the code, making the purpose of the procedure immediately evident. For example:

```
RT
PIC001
PIC002
50.
SH
PIC002
SHI002
-3,8
```

This is a command string to rotate the image stored in PIC001 by 50° and store the result in a file named PIC002. Subsequently PIC002 is to be shifted by a vector with the components −3,8, and the result is to be stored in SHI002.

If such a command string is to be executed many times with different source and destination file names and different values of the parameters (rotation angle, shift vector components), it may be worthwhile to make the command string into a procedure. If the intermediate image file resulting from the rotation is of no interest, the procedure would be written in the following form:

RT1.GLS

```
RT
?INPUT FILE?
SCR001
?ROTATION ANGLE?
SH
SCR001
?OUTPUT FILE?
?SHIFT VECTOR (X,Y)?
RE
```

where GLS is the project code assigned to this session and stored in a file named RT1.GLS. "RT1" is a freely chosen name of the procedure with the general naming convention ⟨AB⟩⟨N⟩, where ⟨AB⟩ is any two letters of the alphabet and ⟨N⟩ is any digit, including 0. Any image name or parameter value in the command string that was chosen to be a variable of the procedure is replaced by a solicitation line in the procedure sequence. A return command "RE" terminates the procedure, passing control to the next-higher level of the command input stream.

When called interactively, the command RT1 will now generate the following dialogue:

```
.operation: RT1
?input file? PIC001
?rotation angle? 50.
?output file? SHI002
?shift vector (x,y)? -3,8
```

which, for the arguments chosen, leads to the same processing sequence as the initial batch command sequence.

In the batch mode, similarly, the command RT1 can always be used to effect a rotation and subsequent shifting of an image. The rules for the sequence of the input lines and the types of value input follow directly from the sequence of the solicitation lines and types of parameters solicited in the procedure. (Note that all text after a semicolon is interpreted as a command.) For example:

B01. GLS

```
;B01     A BATCH COMMAND SEQUENCE TO
;        DEMONSTRATE
;        A CALL TO RT1
MO       ; CREATE MODEL PICTURE: DENSITY
;        WEDGE
PIC001   ; NAME OF OUTPUT FILE
32, 32   ; DIMENSION OF OUTPUT FILE
W        ; MAKE IT A WEDGE
RT1      ; CALL PROCEDURE RT1
PIC001   ; INPUT FILE TO PROCEDURE
50.      ; ROTATION ANGLE
SHI001   ; OUTPUT FILE FROM PROCEDURE
 − 3,8   ; SHIFT VECTOR COMPONENTS
PR       ; USE OVERPRINTING FOR DISPLAY
;        OF RESULT
SHI002   ; INPUT TO OVERPRINTING
N        ; NO CONTOURING
EN P     ; END SESSION AND SPOOL RESULT
;        TO LINE PRINTER
```

3.7. Branching commands

The branching commands in SPIDER essentially duplicate the functions of DO-loop statements, labels, and logical IF statements in FORTRAN. In contrast to the pseudo DO-loop in other systems, where the repetition is achieved by creating multiple copies of the original command stream [22], the DO-loop is here realized by execution-time substitutions in the reading routines. Up to threefold nesting of DO-loops is allowed; but this restriction, similar to the restriction of the number of procedure nesting levels, is arbitrary. Its sole purpose is to limit the storage area used by internal table of DRIVER to a practical size.

Branching commands are meaningful only if they appear in a batch or procedure command sequence.

3.8. Communication between operations within a session

Within a session important parameter values can be transferred from one operation to another by use of the 100 registers, denoted X0...X99. Any register can take the place of any floating-point number or integer in arithmetic expressions or input argument positions. In addition, it can appear in an output argument position, such as OR X10, where the register accepts a value computed in the operation. File names can also be generalized by the use of registers; for example, the input PICX10 is replaced PIC004 if X10 has been previously assigned the value 4.

The first 10 registers are reserved for system- and image-related quantities and are updated during each operation. Image dimensions and values of statistical parameters can thus be accessed and used in a dynamic design of the batch or procedure command sequence.

Each register, due to a special feature of DRIVER, is local to the batch or procedure in which it appears. The use of a particular register, say X20, in a given procedure does not interfere with the value stored in the register X20 used in a calling batch or procedure. For example:

B01.GLS

```
.
.
.
X10 = 5
PR1
X10
.
.
.
EN
```

PR1.GLS

```
X10 = 1
.
.
.
RE
```

Here the register X10 is local to procedure PR1, and will therefore have no effect on the value of X10 in the calling batch sequence B01: the value of X10 printed out in response to the interrogation is 5.

Transfer of values from one level of the calling hierarchy to another is achieved through a special command "RR" (read register). Execution of the sequence

B01.GLS

```
.
.
X10 = 5
PR1
X10
.
.
EN
```

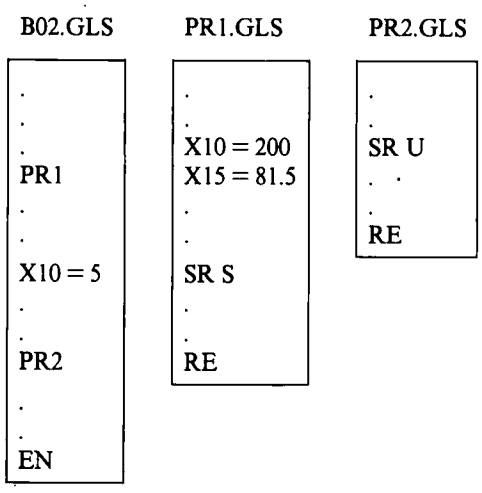
PR1.GLS

```
RR X20
?ENTER HIGHEST REFLECTION ORDER?
.
.
.
RE
```

will transfer the value of register X10 into local register X20 in PR1 in response to the user-created solicitation message "ENTER HIGHEST REFLECTION ORDER".

Transfer of register values between one procedure and another can be achieved by using the command "SR", which has options for saving (S) and unsaving

(U) the values active in all 100 registers. For example:



Here the commands SR S and SR U cause the values of X10 and X15 to be transferred from PR1 to PR2 along with the contents of all other registers. Because of the local character of registers the transferred value of X10 is not changed by the assignment X10 = 5 in the calling sequence. Since the "SR" unsave call in PR2 affects only the local register contents, the value assigned to X10 in the calling sequence remains the same after the PR2 call.

Another means of communication, both within a session and between sessions, is the document file, which will be described in the following section.

3.9. Communication between sessions

All register contents vanish when a session is terminated. A general vehicle for communication of register contents between sessions has been created in the form of a document file. This is a sequential, formatted file organized into keyed document records. These records may or may not coincide with the physical 80-byte records used, depending on the number of registers transferred. The general saving or updating command is

SD ⟨key⟩, X⟨N1⟩, X⟨N2⟩, X⟨N3⟩,...,X⟨NJ⟩
⟨document file name⟩

where ⟨key⟩ is an integer number or a register containing an integer (with the meaning of a particle number, a DO-loop index, etc.) and X⟨N1⟩ ... X⟨NJ⟩ is an arbitrary sequence of J registers.

An unsave command of the form

UD ⟨key⟩, X⟨M1⟩, X⟨M2⟩, X⟨M3⟩,...,X⟨MI⟩
⟨document file name⟩

will pick out the last entry under the number ⟨key⟩ and transfer the contents of the document record into the arbitrary sequence of I registers X⟨M1⟩, X⟨M2⟩, X⟨M3⟩,...,X⟨MI⟩ with I ≤ J. The transfer is determined, not by the register numbers themselves, but the sequence of the register numbers in the SD and UD commands. For example:

X10 = 1
X11 = 5.2
SD 5,X10,X11
DCM001

UD 5,X50,X20
DCM001

This sequence will transfer the values 1 and 5.2 from X10 and X11 into X50 and X20 respectively.

A typical application of the document file is the documentation of rotations, shift vector components and correlation coefficients calculated in the alignment of a series of particle. By the use of the document file any of these quantities may be applied in a subsequent batch run to a different set of files, or they may be used to recreate the set of aligned files from the raw data.

Finally a "list document" option allows the user to tabulate the entire document file using any desired headings. For example, the command

LD 'particle number', 'Angle', 'X shift', 'Y shift', 'corr'
DCM001

will cause the first four entries in each document record to be listed with the key numbers in ascending order and with the column headings 'Angle', 'X shift', etc., applied.

4. Example

An example of extended use of the language is the procedure RA0 (fig. 4), which performs *N*-fold rotational averaging of an uncentered particle contained in an image. The particle is assumed to be centrosymmetric. In this example a batch file B01 is prepared to subject 10 particles, contained in the image series PAR001-PAR010, to six fold rotational averaging. The batch commands instruct SPIDER to apply RA0 with $N = 6$ to 10 particles. RA0 brings the particle into the

image center, generates $N-1$ symmetry-related positions by rotating the particle and adds these to the centered input image.

The method of centering a centrosymmetric motif makes use of cross-correlation [13,37]. First the shift vector between the motif as it appears in the 180°-rotated image is determined. This shift vector has the same direction, and is twice as long, as the vector needed to bring the particle into the image center. The procedure CT1 which is called at the beginning of RA0 is based on this principle. The image is rotated by 180° and cross-correlated with its 180°-rotated version. Both images are padded into a larger array as preparation for cross-correlation to prevent artifacts in the cross-correlation function due to circular overlap [39]. The shift vector found in the search of the correlation peak ("PK") is then halved and applied to the original image. The output file of CT1, TMP001 thus contains an image with the particle in centered position. Through the use of system registers containing image dimensions, procedures can be written in such a way that they are applicable to images of any size.

Another feature of the command language, the use of symbolic references to previously entered input lines, is evident from RA0. In this procedure P1 and P3 are used to invoke previously entered file names.

Another example, the procedures used to align particles that have random orientations and positions, was reported earlier [21].

a

```
; B01   DEMONSTRATE USAGE OF PROCEDURE RAO
DO LB1 I=1,10     ; DO-LOOP OVER PARTICLE SERIES
RAO       ; CALL GENERAL PROCEDURE FOR ROTATIONAL AVERAGING
PAR00I    ; INPUT FILE FOR ROTATIONAL AVERAGING
(6.)      ; MAKE IT SIX-FOLD
RAV00I    ; OUTPUT OF ROTATIONAL AVERAGING
LB1       ; END OF DO-LOOP
EN        ; END BATCH RUN
```

b

```
; RA0  PROCEDURE TO PERFORM AUTOMATIC CENTERING AND N-FOLD ROTATIONAL
; AVERAGING OF AN IMAGE CONTAINING A CENTRO-SYMMETRIC MOTIF.
FI        ; FILE INFORMATION TO INTRODUCE FILE TO BE AVERAGED
?IMAGE TO BE ROTATIONALLY AVERAGED?      ; INPUT TO FILE INFO = INPUT TO RA0
X11=X1    ; STORE NUMBER OF SAMPLES OF INPUT IMAGE IN REGISTER X11
X12=X2    ; STORE NUMBER OF ROWS OF INPUT IMAGE IN REGISTER X12
CT1       ; PROCEDURE CALL: CENTER MOTIF USING CROSS-CORRELATION
P1        ; INPUT TO CT1 = FILENAME ENTERED IN RESPONSE TO 1ST QUERY
TMP001    ; TEMPORARY FILE TO STORE CENTERED OUTPUT IMAGE
RR X20    ; READ SYMMETRY FACTOR INTO REGISTER X20
?SYMMETRY FACTOR?         ; VALUE OF SYMMETRY FACTOR = INPUT TO RA0
FI        ; FILE INFORMATION USED TO READ IN NAME OF OUTPUT FILE
?ROTATIONALLY AVERAGED IMAGE?   ; OUTPUT FILE OF RA0
IF(X20.EQ.1.)GOTO LB2     ; ONE-FOLD AVERAGING MEANS TAKE NO ACTION
IF(X20.LE.0.)GOTO LB3     ; GENERATE ERROR MESSAGE IF FACTOR .LE.0
; FROM HERE ON NORMAL SYMMETRY FACTORS GREATER THAN 1
BL        ; CREATE ZERO BACKGROUND FILE WITH THE DESIRED OUTPUT FILENAME
P3        ; OUTPUT OF BL = FILENAME ENTERED IN RESPONSE TO 3RD QUERY
X11,X12   ; DIMENSIONS OF FILE TO BE CREATED = DIMENSIONS OF INPUT IMAGE
;           THAT WERE PREVIOUSLY STORED
DO LB1 I=2,X20   ; DO-LOOP TO GENERATE AND ADD X20-1 ROTATED, SYMMETRY-
;                  RELATED IMAGES
X30=(I-1)*360./X20        ; CALCULATE ROTATION ANGLE
RT        ; ROTATE CENTERED FILE
TMP001    ; INPUT TO ROTATE = OUTPUT OF PROCEDURE CALL CT1 ABOVE
ROT001    ; OUTPUT OF ROTATE GOES INTO TEMPORARY FILE
X30       ; ROTATION ANGLE AS CALCULATED BEFORE
AD        ; ADD OUTPUT OF ROTATE TO PREVIOUS AVERAGE
P3        ; IMAGE TO BE ADDED TO = RESPONSE TO THIRD QUERY
ROT001    ; IMAGE TO BE ADDED = OUTPUT OF ROTATE
*         ; NO FURTHER IMAGES TO BE ADDED THIS STEP
LB1       ; END DO-LOOP
DE        ; DELETE TEMPORARY FILES
ROT001    ; NAME OF FILE TO BE DELETED
DE
TMP001
RE        ; NORMAL RETURN
LB2       ; JUMP HERE FOR TRIVIAL CASE OF ONE-FOLD SYMMETRY
CP        ; IN CASE OF ONE-FOLD AVERAGING, SIMPLY COPY INTO OUTPUT OF RA0
TMP001    ; OUTPUT OF CT1 = INPUT TO COPY
P3        ; OUTPUT OF COPY = FILENAME ENTERED IN RESPONSE TO 3RD QUERY
RE        ; RETURN FROM ONE-FOLD
LB3       ; JUMP HERE FOR ILLEGAL SYMMETRIZATION FACTORS
CK        ; *** ERROR *** SYMMETRIZATION FACTOR ZERO OR NEGATIVE
RE        ; RETURN FROM ERROR SECTION
```

c

```
; CT1.DOC  PROCEDURE TO CENTER A CENTRO-SYMMETRIC MOTIF
RT        ; ROTATE INPUT IMAGE BY 180 DEGREES TO CREATE SYMMETRY-RELATED IMAGE
?IMAGE TO BE CENTERED? ; INPUT TO ROTATION
INV001    ; OUTPUT OF ROTATION
(180.)    ; ROTATION ANGLE
X21=PAD(2*X1)    ; COMPUTE DIMENSIONS OF PADDED ARRAY. MAKE THEM TWICE THE
X22=PAD(2*X2)    ; DIMENSIONS OF IMAGE, AND ROUND UP TO NEXT INTEGER THAT IS A
;                  POWER OF TWO
PD        ; PAD INPUT IMAGE AS PREPARATION OF CROSS-CORRELATION
P1        ; INPUT TO PADDING = FILENAME ENTERED IN RESPONSE TO 1ST QUERY
PAD001    ; OUTPUT OF PADDING GOES INTO TEMPORARY FILE
X21,X22   ; DIMENSION OF PADDED ARRAY AS CALCULATED BEFORE
Y         ; YES, USE AVERAGE OF IMAGE FOR PADDING BACKGROUND
(1,1)     ; PAD INTO UPPER LEFT CORNER (NOTE THAT PADDING COORDINATES ARE
;           IRRELEVANT FOR CROSS-CORRELATION RESULT, PROVIDED THE SAME ONES
;           ARE USED FOR BOTH IMAGES)
PD        ; PAD 180-DEGREES ROTATED IMAGE
INV001    ; INPUT TO PADING = OUTPUT OF ROTATE
PAD002    ; OUTPUT OF PADDING GOES INTO TEMPORARY FILE
X21,X22   ; DIMENSIONS OF PADDED ARRAY AS ABOVE
Y         ; YES, USE AVERAGE OF IMAGE FOR PADDING BACKGROUND
(1,1)     ; USE SAME PADDING COORDINATES AS ABOVE
CC        ; CROSS-CORRELATE PADDED INPUT WITH PADDED, 180 DEGR. ROTATED IMAGE
PAD001    ; USE PADDED ORIGINAL AS FIRST INPUT TO CC; TO BE OVERWRITTEN BY
;           CROSS-CORRELATION FUNCTION
PAD002    ; USE PADDED, 180-DEGR ROTATED IMAGE AS SECOND INPUT TO CC; TO BE
;           OVERWRITTEN BY ITS FOURIER TRANSFORM
N         ; NO FILTRATION OF CONJUGATE FOURIER CROSS-PRODUCT
PK X11,X12       ; SEARCH THE CROSS-CORRELATION FUNCTION FOR PEAK, STORE
;         ; POSITIONS IN REGISTERS X11,X12 FOR LATER USE
PAD001    ; INPUT TO PEAK-SEARCH = CROSS-CORRELATION FUNCTION FROM PREVIOUS STEP
(3)       ; LIST OF 3 HIGHEST PEAKS ON PRINTOUT
X11=-X11/2       ; CALCULATE NEGATIVE HALF OF PEAK SHIFT VECTOR. THIS IS THE
X12=-X12/2       ; VECTOR BY WHICH THE IMAGE HAS TO BE SHIFTED TO BE CENTERED
SH        ; SHIFT ORIGINAL IMAGE
P1        ; INPUT TO SHIFT = FILENAME ENTERED IN RESPONSE TO 1ST QUERY
?CENTERED FILE? ; OUTPUT OF SHIFT = OUTPUT OF THIS PROCEDURE
X11,X12   ; USE VECTOR COMPONENTS COMPUTED ABOVE TO CENTER IMAGE.
DE        ; DELETE ALL TEMPORARY FILES
PAD001
DE
PAD002
DE
INV001
RE        ; RETURN
```

Fig. 4. Example of a three level calling structure in command language. Ten images are stored in PAR001,..., PAR010, each containing a particle projection in uncentered position. To be computed is the sixfold, rotationally symmetrized average of the centered particle. The user need only set up B01 (a), a batch file utilizing a DO-loop over the particle series, and a call to RA0 (b; general rotational symmetrization), specifying 6 as the symmetry count. The job of centering a centrosymmetric motif is delegated to the procedure CT1 (c) which is called by RA0. Both RA0 and CT1 are part of a standard procedure library and make use of existing basic operations such as cross-correlation, rotation, and shifting. Since no compilation and linking are involved, the development of the procedures takes no more than a few minutes.

5. Documentation

The system documentation consists essentially of four parts: a *user introduction* explaining the main concepts; a *command manual*, which outlines the system/user dialogue for each operation; a *cross-reference table*, which lists the commands and the relevant sections in the user introduction for a large number of keywords; and an alphabetical *list of subroutines* making up the package, along with brief explanations of their functions. All four parts of the documentation are updated each time an operation is changed or added to the existing package.

In addition there is documentation of the SPIDER procedures. User-built procedures are normally very specific to a project and have no practical value beyond the lifetime of the project. Only a few procedures that are generally useful are adopted into the system's procedure library. The system/user dialogue for such procedures is explained in the same way as the dialogue for basic commands in the command manual. Documentation of the remaining procedures is the user's responsibility.

6. Transportability

SPIDER was designed initially for the DEC RSX11D operating system and then converted to run under RSX11M version 3.2. The system layout into several tasks will be advantageous for any minicomputer system with multi-user environment. The main features affecting the transportability are that (1) initiation of slave tasks by a resident master task is supported; (2) SEND/RECEIVE communication allows the master task to initialize important parameters of the slave task; (3) file names are created and dynamically changed at various points in the program system according to the DEC file-11 naming convention; and (4) file information (existence of file, number of records and record length) can be interrogated in FORTRAN routines.

These functions are normally supported by most minicomputer configurations, and for these the implementation of the SPIDER system should offer no particular problem. The most drastic changes concern the system-specific file conventions.

7. Conclusions

Our software system resulted from an attempt to realize a large variety of image-processing operations on a small computer and to construct a control language that allows branching, iterating and procedure-nesting on several levels.

The price for the flexibility of the system is overhead time for searching of commands and swapping of slave tasks. However, this price may not be too high, considering the time and expenditures involved in the creation of rapidly ageing, project-oriented programs.

A separate paper will be devoted to applications of the SPIDER system in the main areas of electron image analysis.

Acknowledgements

Many colleagues and students have contributed to the gradual growth of SPIDER. We wish to acknowledge the help of Timothy Bilash, William Goldfarb, Richard Green, Robert Marshal, Richard Pelavin, and Vicky Riffle. Special thanks are due to Dr. William Moyer for many suggestions and assistance.

We thank Adriana Verschoor for stylistic corrections and comments. We are grateful to Dr. P.R. Smith for suggesting the survey in this form, and to all authors of other software systems for sending us detailed information.

Appendix 1. Table of SPIDER commands

(Commands created for interfacing multivariate statistical analysis programs are not included in this list.)

Operation	Short description	Function
AC	Autocorrelation	Auto-correlate an image using Fourier method
AD	Add	Add two or more images point-for-point
AF	Angular Fourier	Fourier transform in azimuthal direction
AI	Angular interpolation	Convert cartesian into polar representation
AR	Arithmetic operation	Perform point-for-point arithmetic operation on image
BC	Box convolution	Form local average and mix with original image

Appendix 1, Table of SPIDER commands (continued)

Operation	Short description	Function
BL	Blank	Create image with constant background
BP	Back projection	Back-project in two or three dimensions
CC	Cross-correlation	Cross-correlate two images using Fourier method
CE	Contrast enhancement	Stretch density scale/histogram-equalize
CF	Construct Fourier	Construct a Fourier transform from set of amplitudes and phases
CH	Correlation histogram	Plot histogram of correlation values or other values stored in document file
CN	Convolution	Convolute two images with each other using Fourier method
CO	Contour	Contour plot
CP	Copy	Copy
CR	Cross-reference	Extract Fourier from existing projection-Fourier stack
CS	Central slice	Obtain 2-D section of 3-D volume in arbitrary direction
CT	Concatenate	Concatenate two or more images
DC	Determine common line	Determine tilt axis and tilt angle from marker coordinates for back-projection
DE	Delete	Delete file
DF	Density foldover	Obtain display with bit-clipping
DO	DO-loop	Start of DO-loop
DU	Dust	Reset image points that are off the mean by 3 standard deviations or more
ED	Edge enhancement	Enhance edges in image by using recursive filtering
EF	Extract Fourier	Extract 2-D Fourier from 3-D Fourier
EN	End	End SPIDER session
EX	Exit	End SPIDER session and save LOG file
FC	File contour	Contour image by bit-clipping and superpose on image
FF	Fourier filter	Apply low- or high-pass filter function to Fourier transform
FI	File information	Show statistical attributes of image
FL	Fourier list	Print selected portions of Fourier transform
FP	Fourier interpolation	Interpolate by padding Fourier transform
FS	Find statistics	Compute statistical attributes of image
FT	Fourier transform	Compute Fourier transform or inverse
GF	General filter	Mask Fourier transform on reciprocal lattice
GP	Generate projections	Extract 1-D projection lines from image series for back projection
GS	Gray scale	Display image using Versatec halftone software
HI	Histogram	Compute and display histogram of image
IF	Logical IF	Conditional jump depending on arithmetic comparisons
IN	Insert	Insert an imago into a larger one
IP	Interpolate	Interpolate into arbitrary rectangular format using bilinear interpolation
LB#	Label	Destination label for conditional jumps and DO-loops
LD	List document	List contents of document file ordered by key
LI	List file	List any file by rows
MA	Mask	Apply circular mask to image
MD	Mode	Select global processing mode
ME	Menu	Display menu of commands
MO	Model	Create model image
MR	Mirror	Create mirror-related version of image

Appendix 1, Table of SPIDER commands (continued)

Operation	Short description	Function
MU	Multiply	Multiply two images point-for-point
OR	Orientation	Find orientation between two images or auto-correlation functions
PA	Patch	Add small image onto large image at arbitrary position
PD	Pad	Pad image with average or background constant to make it larger
PF	Profile	Plot profile of a selected image row
PH	Phase Fourier stack	Apply phase shift to projection-Fourier stack
PJ	projection	Compute 1-D or 2-D projection of 2-D or 3-D volume
PK	Peak search	Search positions of N highest peaks
PO	Poem	(Operation to celebrate the 100th command)
PR	Print	Display image using overprinting
PS	Pick slice	Pick slice from 3-D volume
PW	Power spectrum	Compute modulus of Fourier transform
RA	Ramp	Determine least-squares density wedge of image and subtract
RC	Real space convolution	Convolute image with arbitrary rectangular array
RD	reduce transform	Create reduced Fourier transform from amplitudes and phases of reflections
RE	Return	Return from procedure to next-higher level of command language
RF	Rotational filter	Filter angular Fourier
RN	Rename	Rename file
RO	Rotational average	Compute rotationally averaged profile
RR	Read register	Read number into register
RT	Rotate	Rotate image
SC	Scale Fourier stack	Scale projection-Fourier stack
SD	Save document	Store register contents in document file
SF	Stack Fourier	Edit projection-Fourier stack (add, delete, insert)
SH	Shift	Shift image
SI	Stack interpolation	Interpolate projection-Fourier stack into 3-D Fourier
SK	Stack 2-D slices	Create 3-D volume by stacking images representing slices
SL	Slice	Slice a 3-D volume in arbitrary direction
SQ	Square	Square image point-for-point
SR	Save registers	Save/unsave registers temporarily
SS	Serial section	Align images of serial section according to marker positions
ST	Set label	Edit statistical and protection label of file
SU	Subtract	Subtract two images point-for-point
SZ	Squeeze	Shear image to conform with arbitrary unit vector angle
TA	Tilt angle	Refine tilt angle using positions of reflections
TF	Tilted transfer function	Generate a transform function vs. defocus display
TI	Tape information	List contents of tape from microdensimeter
TM	Time	Print wall clock time
TP	Three-D plot	Make perspective plot of an image
TR	Tape read	Read image from tape to disk
TT	Title	Change title of image
TV	TV display	Display image on halftone display system
TW	Tape write	Write image onto tape in microdensitometer format
UD	Unsave document	Read registers from document file
WI	Window	Window out portion of image

Appendix 1, Table of SPIDER commands (continued)

Operation	Short description	Function
WT	TV Window	Window image interactively/index reciprocal lattice interactively
WU	Wurzel (square root)	Compute point-for-point square root of image
WV	Window averaging	Window out and sum portions from image according to vectors stored in document file

References

[1] P.W. Hawkes, Computer Graph. Image Processing 8 (1978) 406.
[2] D.L. Misell, Image Analysis, Enhancement and Interpretation (North-Holland, Amsterdam, 1978).
[3] W.O. Saxton, Computer Techniques For Image Processing, in: Advances in Electronics and Electron Physics, Suppl. 10, Ed. L. Marton (Academic Press, New York, 1978).
[4] R.A. Crowther and A. Klug, Ann. Rev. Biochem. 44 (1975) 161.
[5] J. Frank, J. Microsc. 117 (1979) 25.
[6] P.R. Smith, Ultramicroscopy 3 (1978) 153.
[7] P.W. Hawkes, in: Computer Processing of Electron Microscope Images, Ed. P.W. Hawkes (Springer, Berlin, 1980).
[8] J. Frank, in: Advanced Techniques in Biological Electron Microscopy, Ed. J.K. Koehler (Springer, Berlin, 1973).
[9] J. Frank, Biophys. J. 12 (1972) 484.
[10] O. Kübler, M. Hahn and J. Seredynski, Optik 51 (1978) 171, 235.
[11] R.E. Burge, T.C. Dainty and R.F. Scott, Ultramicroscopy 2 (1977) 169.
[12] P.N.T. Unwin and R. Henderson, J. Mol. biol. 94 (1975) 425.
[13] J. Frank, W. Goldfarb, D. Eisenberg and T.S. Baker, Ultramicroscopy 3 (1978) 283.
[14] H.P. Zingsheim, D.-Ch. Neugebauer, F.J. Barrantes and J. Frank, Proc. Natl. Acad. Sci. USA 77 (1980) 952.
[15] J. Frank and W. Goldfarb, in: Electron Microscopy in Molecular Dimensions; State of the Art and Strategies for the Future, Ed. W. Baumeister and W. Vogell (Springer, Berlin, 1980) p. 261.
[16] J. Frank and M. van Heel, in: Pattern Recognition in Practice, Eds. E.S. Gelsema and L.N. Kanal (North-Holland, Amsterdam, 1980) p. 235.
[17] M. van Heel and J. Frank, Ultramicroscopy 6 (1981) 187.
[18] D.. DeRosier and A. Klug, Nature 217 (1968) 130.
[19] J.E. Mellema, in: Computer Processing of Electron Microscope Images, Ed. P.W. Hawkes (Springer, Berlin, 1980) p. 89.
[20] F.C. Billingsley, Advances in Optical and Electron Microscopy, Vol. 4, Eds. Barer and V.E. Coslett (Academic Press, London, 1971) p. 127.
[21] J. Frank and B. Shimkin, in: Proc. 9th Intern. Congr. on Electron Microscopy, Ed. J.M. Sturgess (Microscopical Soc. Canada, Toronto, Ontario, 1978) Vol. I, p. 210.
[22] R.H. Wade, A. Brisson and L. Tranqui, J. Microsc. Spectrosc. Electron. 5 (1980) 699.
[23] S. Kawata, Y. Ichioka and T. Suzuki, J. Phys. E (Sci. Instr.) 11 (1978) 1191.
[24] S. Kawata, Y. Ichioka and T. Suzuki, Optik 52 (1978) 235.
[25] W.O. Saxton, T.J. Pitt and M. Horner, Ultramicroscopy 4 (1979) 343.
[26] M. Horner, in: Developments in Electron Microscopy and Analysis, Ed. J.A. Venables (Academic Press, London, 1976) p. 209.
[27] W.O. Saxton, Computer Graph. Image Processing 3 (1974) 266.
[28] B.L. Trus and A.C. Steven, Ultramicroscopy 6 (1981) 383.
[29] R. Hegerl, in: Electron Microscopy 1980, Eds. P. Brederoo and W. de Priester (7th European Congr. on Electron Microscopy Foundation, Leiden, 1980) Vol. II, p. 700.
[30] M. van Heel and W. Keegstra, Ultramicroscopy, to be published.
[31] D.J. DeRosier and P.B. Moore, J. Mol. Biol. 52 (1970) 355.
[32] W. Goldfarb and J. Frank, in: Proc. 9th Intern. Congr. on Electron Microscopy, Ed. J.M. Sturgess (Microscopical Soc. Canada, Toronto, Ontario, 1978) Vol. I, p. 22.
[33] W. Goldfarb, J. Frank, J.C. Hsung, C.H. Kim and T.E. King, in: Cytochrome Oxidase, Eds. T. King, Y. Orri, B. Chance and K. Okunuki (Elsevier–North Holland–Biomedical Press, Amsterdam, 1978) p. 161.
[34] J. Frank, J.N. Turner, M. Marko, K. Asmus and D.F. Parsons, in: Proc. 38th Ann. Meeting EMSA, San Francisco, 1980, Ed. G.W. Bailey (Claitor, Baton Rouge, LA) p. 46.
[35] R. Gordon, R. Bender and G.T. Herman, J. Theoret. Biol. 29 (1970) 471.
[36] D. Fraser, ACM Trans. Math. Softw. 5 (1979) 500.
[37] J. Frank, in: Computer processing of Electron Microscope Images, Ed. P.W. Hawkes (Springer, Berlin, 1980) p. 187.