reconstruction, or by pooling data into groups with a narrow defocus range, which are then separately reconstructed and combined at the very end using CTF correction by Wiener filtering (Penczek et al., 1997). Although both strategies can be readily realized in SPIDER, the one most thoroughly tested and proven is the latter, sometimes referred to as defocus group reconstruction.

For a step-by-step description of the processing path, we refer to the flow diagram in Fig. 19.8.2.1. After defocus determination, particles are selected and aligned to a set of reference projections using rotational and translational cross-correlation. Eulerian angles yielding the highest cross-correlation are assigned. Next, the data are pooled into defocus groups, defined by the range of allowed defocus variation within each group. The range needs to be small enough to avoid resolution loss. Reconstruction of each defocus group yields a preliminary density map. These density maps are merged by CTF correction via Wiener filtering, yielding an initial CTF-corrected reconstruction. This reconstruction is then used as the reference in the first angular refinement of defocus group reconstructions, which follows the same principle as the initial alignment. Several cycles of angular refinement are used with progressively smaller angular steps, from the initial \( \sim 15^\circ \) to \( 0.5^\circ \) or even \( 0.2^\circ \). From a certain point on, comparisons are only done within a narrow cone of the previous Eulerian angle assignment for a given projection. The progress of the refinement is monitored by computing the Fourier shell correlation at each stage and by following the statistics of angle reassignment, watching for evidence of stability.

For supervised classification, not one but two three-dimensional references are used in the projection alignment (Fig. 19.8.2.1), and the assignment of the class and set of Eulerian angles is done according to the parameters of the reference projection producing the highest cross-correlation coefficient. For example, ribosome data may be split on the basis of inter-subunit ratchet rotation, by presenting ‘ratcheted’ and ‘non-ratcheted’ ribosomes as references.

19.8.3. Implementation of single-particle reconstruction in SPIDER

The implementation of the reconstruction strategy presented in the flow diagram (Fig. 19.8.2.1) makes use of the modular features of SPIDER and its hierarchical calling structure (Fig. 19.8.3.1). The computational task of the entire project (level 4, or uppermost in the hierarchy) can be visualized as being composed of building blocks (level 3), each representing a subtask, such as image alignment, defocus determination, angular refinement etc. Each of these building blocks is realized by either a set of elementary commands (level 1) or procedure files containing such commands (level 2). In the reconstruction task, examples of procedures are `deffsc.spi` and `ctf.spi`, which compute the resolution of each defocus group and apply CTF correction, respectively. On the uppermost project level, all data file names, directories and values of parameters must be specified, while the input for lower-level procedures can be variables whose value is not known until the time of execution.

19.8.3.1. Level 1: elementary commands

Commands in SPIDER consist of a short two-to-three letter operation mnemonic and a variable sub-option mnemonic. For example, `BP CG` invokes a program performing back-projection reconstruction using a conjugate gradient algorithm. In the interactive on-line mode, commands are entered at a prompt and the system responds with a series of subsequent prompts for more input specific to the operation the command invokes. In the off-line batch mode, relevant in the context of time-consuming reconstructions, both commands and subsequent inputs are placed in a batch or procedure file in the expected order for successive execution.

Figure 19.8.2.1
Flow diagram of reference-based single-particle reconstruction by defocus groups. For details, see text.
19. OTHER EXPERIMENTAL TECHNIQUES

There are over 400 different computational operations and sub-operations available in SPIDER, which can be grouped into the following categories:

(i) File handling: copying, format converting, montaging, stacking.

(ii) Real-space image manipulations: alignment, centre-of-gravity computation, contouring, contrast adjustment, edge detection, filtering, histogram computation, masking, mirroring, montaging, peak search, rotation, scaling, segmentation, thresholding, windowing.

(iii) Fourier-space image manipulations: autocorrelation, cross-correlation, CTF correction, Fourier filtering, Fourier interpolation, Fourier transformation, resolution determination.

(iv) Statistical analysis: averaging, classification of image sets, multivariate data analysis.

(v) Three-dimensional reconstruction: algebraic reconstruction technique (ART), constrained simultaneous iterative reconstruction technique (SIRT), projection, SIRT, weighted back-projection.

19.8.3.2. Level 2: procedures

The modular design of SPIDER allows incremental development and flexibility by allowing the user to write scripts in the form of procedure files. Procedures containing sequences of commands can be executed just like single commands. Procedures have their own name space, can pass arguments to other procedures and can request input during an interactive SPIDER session.

A single-particle reconstruction project requires the application of many complex operations on perhaps several hundred thousand images, in a 30- to 40-step process that involves branching and decision making. To handle these tasks, the scripting facilities of SPIDER include commands for conditional expressions, iteration over numbered file names and document manipulation.

SPIDER stores and retrieves numerical results in document files. These are keyed sequential text files containing a table of values, and are used to permanently store items such as image shift coordinates, rotation angles and particle numbers. These values can be retrieved

Figure 19.8.3.1
Schematic representation of the SPIDER hierarchy of functional units and calling levels.

Figure 19.8.4.1
A SPIRE form, displaying the parameters, input files and output files of a SPIDER procedure file. The user can edit the values in the white entry boxes or browse for files (e.g. by clicking on the ‘noise file’ button). The changes can either be saved to update the procedure file or executed in SPIDER immediately (‘Save’ and ‘Execute’ buttons).

Figure 19.8.5.1
Reconstruction of the E. coli ribosome bound with the Phe-tRNA-EF-Tu-GDP ternary complex at 6.7 Å resolution, obtained by the protocol outlined here (LeBarron et al., 2008).
by different procedures or even subsequent SPIDER sessions.

There are over 30 document and scripting operations in SPIDER:

(i) **Document operations**: retrieve values, select items, sort document, store values.

(ii) **Scripting operations**: assign numerical variables, assign string variables, capture values from SPIDER operations, check numerical values, carry out conditional execution and looping, evaluate mathematical expressions, make file enquiries, pass values to/from procedures, set global switches.

19.8.4. Implementation of single-particle reconstruction in SPIDER

SPIRE (the SPIDER reconstruction engine) is an interactive GUI for managing a SPIDER reconstruction (Baxter et al., 2007). A project of single-particle reconstruction may involve dozens of procedure files, each of which may contain hundreds of individual SPIDER operations. Procedures usually read SPIDER input files (images, volumes or text document files), carry out some processing and write out new SPIDER files. The outputs of one procedure are the inputs of the next, and perhaps of other procedures further down the processing stream. Hundreds of thousands of files may be generated during the course of a project.

SPIRE was developed to manage the numerous procedures and data files involved in a SPIDER project. As each procedure is executed, SPIRE keeps track of the output files generated and adds them to a project database. Users can run the same procedure several times, changing input parameters, and each ‘run’ is given a unique identification number by SPIRE. SPIDER is executed in a controlled environment by SPIRE, in which error conditions can be trapped and appropriate messages displayed to the user. Successful runs are added to the project database. SPIRE provides a graphical interface to these tasks, allowing the user to make changes to the inputs and outputs of a procedure in a displayed form (Fig. 19.8.4.1). Project-wide parameters used by multiple procedures are also graphically viewable and editable. The project database has a spreadsheet-like interface, which can display SPIDER images, volumes and stack files in Jweb, as well as text documents in a user-specified editor. Procedure files can be edited, new procedures may be added to a project, and the way a procedure is displayed can be changed.

Projects are represented in SPIRE by configuration files. These are text files that use XML tags to specify content, such as the order of procedure files and the directory structure of a project. SPIRE provides a configuration editor to manipulate configurations, and thus the overall structure of a SPIDER project. If there is a laboratory database system, SPIRE provides some capability to upload project results using SQL commands. Finally, SPIRE allows the user to write the project file to disk in HTML format, so that a web browser can be used to find out when batch files were run and which data files were generated during the project. SPIRE is written in the Python programming language; for graphics it uses the Tkinter module (the interface to the Tk library).

19.8.5. Conclusion

SPIDER was designed to allow high versatility in the design of program flow, so the protocol presented here and in the article by Shaikh et al. (2008), which has been used successfully in many applications (for a recent example, see Fig. 19.8.5.1), can be readily changed to meet the challenges of a new project by users who have received a modest amount of training. The same goes for the implementation of alternative strategies, such as CTF correction of raw data, which is beyond the scope of this chapter.

This work was funded by HHMI and grants from NCRR/NIH P41 RR01219 (to JF), P01 GM064692 (to Robert M. Glaeser) and NIH R01 GM150601 (to AL). The authors thank Lila fino-Rubenstien for assistance with the illustrations.

References


