Available online at www.sciencedirect.com
SCIENCE DIRECT
Journal of Structural Biology xxx (2006) xxx–xxx
Journal of Structural Biology
www.elsevier.com/locate/yjsbi

EMAN2: An extensible image processing suite for electron microscopy

Guang Tang [a], Liwei Peng [a], Philip R. Baldwin [b], Deepinder S. Mann [a], Wen Jiang [c], Ian Rees [a], Steven J. Ludtke [a,*]

[a] National Center for Macromolecular Imaging, Verna and Marrs McLean Department of Biochemistry and Molecular Biology, Baylor College of Medicine, Houston, TX 77030, USA
[b] The University of Texas – Houston Medical Center, Department of Biochemistry and Molecular Biology, Houston, TX 77030, USA
[c] Department of Biological Sciences, Purdue University, West Lafayette, IN 47907, USA

Received 2 May 2006; received in revised form 29 May 2006; accepted 31 May 2006

Abstract

EMAN is a scientific image processing package with a particular focus on single particle reconstruction from transmission electron microscopy (TEM) images. It was first released in 1999, and new versions have been released typically 2–3 times each year since that time. EMAN2 has been under development for the last two years, with a completely refactored image processing library, and a wide range of features to make it much more flexible and extensible than EMAN1. The user-level programs are better documented, more straightforward to use, and written in the Python scripting language, so advanced users can modify the programs' behavior without any recompilation. A completely rewritten 3D transformation class simplifies translation between Euler angle standards and symmetry conventions. The core C++ library has over 500 functions for image processing and associated tasks, and it is modular with introspection capabilities, so programmers can add new algorithms with minimal effort and programs can incorporate new capabilities automatically. Finally, a flexible new parallelism system has been designed to address the shortcomings in the rigid system in EMAN1.

Keywords: EMAN; Single particle analysis; cryoEM; TEM; Software; Image processing; Electron microscopy

1. Introduction

Electron cryomicroscopy (cryoEM) and single particle reconstruction have undergone dramatic growth over the last decade. This is due to a combination of improvements to equipment, computer processing power, and software. None of these improvements would have been sufficient on their own to produce the recent dramatic improvements in this field. It is now possible to produce reconstructions of molecules and macromolecular assemblies in the range of hundreds of kilodaltons to hundreds of megadaltons at subnanometer resolutions, for example see (Booth et al., 2004; Bottcher et al., 1997; Cheng et al., 2004; Fotin et al., 2004; Jiang et al., 2006; Ludtke et al., 2004; Matadeen et al., 1999; Zhou and Chiu, 2003). Unlike crystallography, single particle reconstruction becomes easier rather than more difficult as the size of the system increases. While tremendous strides have been made in determining the structure of individual proteins, as observed by the rapid growth of the PDB, it is clear that unlocking the secrets of the cell will require study of large systems of interacting proteins and/or RNA/DNA. The capability of producing structures from small quantities of large, fragile assemblies will continue to drive expansion of this field. Indeed, hybridizing intermediate resolution structures of assemblies from cryoEM with crystallographic reconstructions of individual components has become a standard practice in recent years, for example, see (Baker et al., 2003; Fotin et al., 2004; Gao et al., 2003; Ludtke et al., 2004; Milne et al., 2002; Zhang et al., 2003), as well as several software packages designed specifically for this task (Jiang et al., 2001; Volkmann and Hanein, 1999; Wriggers et al., 1999).

* Corresponding author. Fax: +1 713 798 8682.
E-mail address: sludtke@bcm.tmc.edu (S.J. Ludtke).

1047-8477/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jsb.2006.05.009

Over the last three decades, a wide range of scientific image processing packages have been developed for use with electron microscopy, including SPIDER (Frank et al., 1996), IMAGIC (van Heel et al., 1996), BSOFT (Heymann, 2001), FREALIGN (Grigorieff, 1998), EM (Hegerl, 1996), IMIRS (Liang et al., 2002), SUPRIM (Schroeter and Bretaudiere, 1996), IMOD (Kremer et al., 1996), PHOELIX (Carragher et al., 1996), PFT (Baker and Cheng, 1996), the MRC reconstruction tools (Crowther et al., 1996) and Xmipp (Sorzano et al., 2004). These packages range from full featured image processing environments to sets of tools focused on specific tasks related to a specific type of reconstruction. The very breadth of software still being developed in this field demonstrates that there are still opportunities to further optimize the techniques now in use, both in quality and computational efficiency. EMAN (Ludtke et al., 1999) was first introduced in 1999, and its popularity can be largely attributed to its suite of GUI tools, general ease of use, and its capability of performing fully CTF-corrected single particle reconstructions at high resolution with a high level of automation.

1.1. Evolution of EMAN

The original EMAN suite has a tiered architecture, including a scientific image processing library in C++ with partial Python bindings, a set of user-level command line applications for specific tasks written in both C++ and Python, and a set of GUI tools for various specific tasks, written in C++ using the QT toolkit. It has support for parallel processing on clusters, SMP supercomputers or sets of individual workstations. Historically, the library was originally written in Objective-C, then ported to C++, and eventually linked to the Python scripting language. This rather convoluted development process left the library with no clear organizational model for incorporating new features, either for the library or end-user programs. When we began to work with the PHENIX (Adams et al., 2004, 2002) project to produce SPARX (see the companion piece in this issue), it became clear that many features necessary to take the next steps towards higher resolution reconstructions and development of novel techniques could not be reasonably incorporated into the original EMAN1 design. For example, to design an extensible GUI, introspection, the ability to query software libraries for information about the functions they contain, is a critical element. In typical software libraries, including EMAN1, adding a single new function requires a time consuming recompilation of the entire software package. However, modern object-oriented design schemes permit adding new functionality while only recompiling the added code, easing the debugging process and saving hours of developer time. The process known as 'refactoring' involves restructuring existing code to provide the same capabilities, but with a more logical or flexible interface. Applying this process to the EMAN1 library provided the opportunity to redesign the library structure while retaining a majority of the well-tested image processing code from the original library.

The goals of the EMAN2 refactoring were to provide: easy extensibility, a complete logical Python interface, introspection capabilities for GUI integration, a properly designed documentation interface, a scheme for metadata management and built in unit-testing for reliability. For the end-user programs the aim was to remove many of the limitations present in the original EMAN1 library and to adapt knowledge developed over six years in using EMAN1 into the standard refinement processes. The EMAN2 library also provides much of the core functionality for the SPARX package (see the companion piece in this issue), though SPARX is now also integrating capabilities from other software suites to produce a flexible environment for cryoEM software development.

The philosophy behind EMAN is to provide a continuously updated set of tools representing the current state of the art in single particle reconstruction, packaged in an easy to use environment, permitting structures to be solved with a high level of confidence at the highest possible resolution. EMAN2 will eventually replace EMAN1 entirely, though EMAN2 was designed to coexist with EMAN1 to permit a gradual transition and not disrupt active research projects.

1.2. Python in Scientific Programming

Python is a full-featured object oriented scripting language (http://www.python.org). Over recent years, it has become the de-facto standard for a wide range of scientific software packages. For structural biology, this largely began with the visualization community, first with packages like Chimera (Pettersen et al., 2004), Vision (Sanner et al., 2002) and Pymol (http://pymol.sourceforge.net) which are written largely in Python with supporting libraries in C/C++. The trend then continued such that a vast majority of scientific visualization tools now offer Python bindings in some form. Unlike strongly typed languages such as Java, which force the end user to write very rigorously designed highly structured programs, Python has a relaxed, yet powerful, structure focused on getting results quickly and flexibly. For typical scientific end-users, not interested in writing large applications, but simply writing a small script for their own use to achieve a particular result efficiently, Python is ideal. Its language structure is very easy to learn, and the fact that it has become so widely used means that a small investment in learning the basic language syntax immediately provides the user with new capabilities in a wide range of software.

While Python is flexible and full featured, it is still an interpreted scripting language, meaning its performance is substantially worse than compiled languages such as C++/Fortran. For this reason, Python is not directly used for low-level, compute-intensive tasks. Rather a set of basic functions is coded in C/C++/Fortran, then provided to Python as a callable library. For example, in EMAN2, if one wished to calculate the FFT of an image, a Python

image object would be sent to a C++ FFT routine, then the result would be returned as a new Python image object. This process is, of course, transparent to the user, who simply enters 'fftimg = img.do_fft()'. Using this design methodology, Python provides a host of advantages in writing user-level programs with a negligible impact on performance.

2. EMAN2 design

The overall design of EMAN2 is shown in Fig. 1. The suite consists of

- C++ Core Library—The library includes over 500 high-performance image processing routines and associated classes for related operations. The library makes use of only the most widely supported C++ features to ease portability.
- Python Bindings—The full C++ library is made available through the Python language, with a calling syntax almost identical to the syntax used from C++. From Python the library appears like any other callable Python library.
- Command Line Programs—All command line programs are written in Python, permitting customization by advanced users without recompilation or knowledge of C++. All programs use a standard interface and documentation mechanism.
- GUI Widgets—EMAN2 includes a set of widgets for image display and manipulation, all usable from Python. Most development is focused on PYQT4 widgets, though basic image viewing widgets are also provided for WxPython.
- GUI Programs—All GUI programs are also written in Python with C++ support from the underlying libraries. This adds an unprecedented level of flexibility to the GUI tools as well as making them much easier to customize for specific applications.

The core library inherits much of the well-tested image processing code from the EMAN1 library, with the algorithms organized into a more logical structure (see Section 2.7). The core library contains an expansive image-processing library including a range of Fourier and real-space filters, 2D and 3D registration/alignment, image similarity metrics, 3D to 2D projections, 3D reconstruction, CTF correction, and so on. All together there are over 500 image processing functions, all optimized for work on cryoEM data.

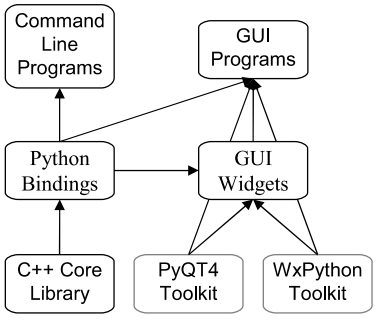

Fig. 1. Diagram of the overall design of the EMAN2 package.

2.1. Command-line programs

Much of the work to date has focused on producing a robust, easy to use core library, and attention is just shifting to producing command-line programs and GUI tools. Already, however, a number of command-line programs exist with functionality beyond that of EMAN1: *e2boxer.py* is EMAN2's automatic particle selection program. It will offer a variety of automatic particle picking algorithms, as well as eventually integrating GUI capabilities. The current version already offers a much more sophisticated reference-based particle picker than EMAN1. *e2proc2d.py* and *e2proc3d.py* are the EMAN2 versions of EMAN1's generic image processing and file format conversion utilities for 2-D and 3D images, respectively. Similarly, programs like *e2project3d.py*, *e2make3d.py* and *e2classifykmeans.py* are the equivalent EMAN2 programs to similarly named EMAN1 programs. *e2pdb2mrc.py* provides conversion from a PDB model to an electron density map, and is 10–100× faster than *pdb2mrc* in EMAN1, with no loss of accuracy. *e2scannereval.py* is a useful tool for evaluating problems with film scanners. It calculates a grid of local 2D power spectra across the image, then overlays them on the original image, so problems with, for example, bent film causing focus variations can be directly observed. *e2tomogram.py* is a tool for correlation-based tomographic tilt-series alignment with a variety of different algorithms for specific situations.

The current approach is to consider the EMAN1 programs individually, and convert them to EMAN2, simultaneously making improvements based on knowledge gained in development and use of EMAN1. This process will continue until all useful functionality has been converted from EMAN1. Nonetheless, EMAN2 is already a useful package, as the tools it does provide are often substantially improved over the corresponding tools in EMAN1.

2.2. Using EMAN2 from Python

In addition to the command-line programs, which require no programming knowledge, the EMAN2 library can be used interactively from the Python prompt. This interface provides even those with minimal programming skills access to a wide range of complex capabilities. Fig. 2 shows a snapshot of the interactive WxPython shell with several images displayed. One useful feature of this interface is that when images are displayed, the display updates in real-time. That is, the following, entered interactively at the *e2.py* prompt:

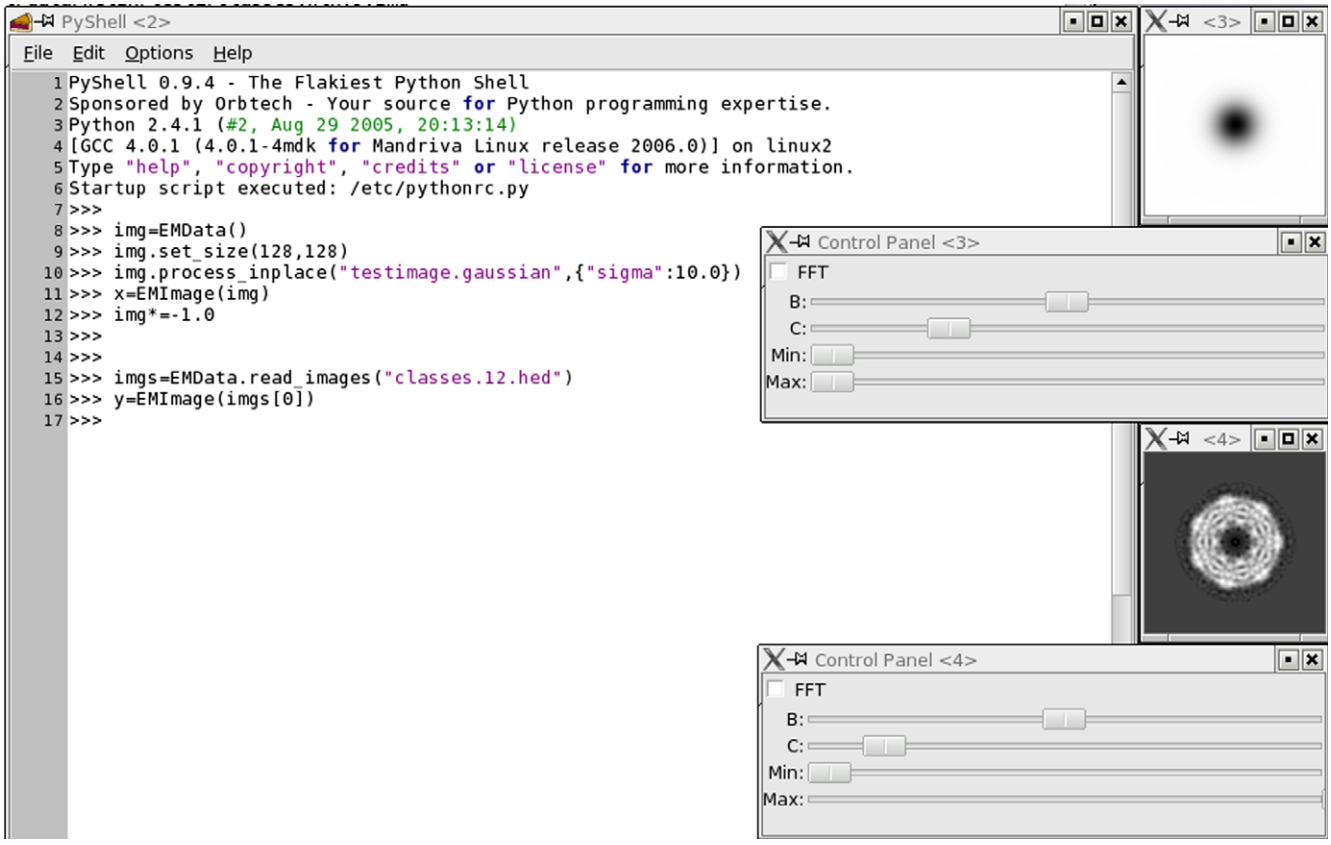

Fig. 2. An example of an interactive Python session using the *e2.py* shell. Image displays update immediately whenever an image is modified. As in EMAN1, using the middle mouse button on any image widget will produce a control panel for adjusting image display options.

```
img=EMData()
img.set_size(128,128)
img.process_inplace("testimage.gaussian",{"sigma":10.0})
x=EMImage(img)
img*=-1.0
```

will create a new image object, 128×128 pixels in size, fill it with a Gaussian decay in the center of the image, display the image on the screen, then invert the image contrast. The image is first displayed immediately after the 'x = EMImage(img)' command. When the image is inverted, and any other time it is changed in the future, the screen display immediately updates. This permits the user to perform tasks like interactively tuning filter parameters with continuous feedback.

As another simple example, we consider the problem of reading in an image stack, filtering each image, and writing the results to a new file. This example reads the entire stack into memory, but one could just as easily process one image at a time.

```
imgs=EMData.read_images("inputstack.spi")
for i in imgs:
  i.process_inplace("filter.low-
  pass.gauss",{"sigma":0.25})
  i.write_image("outputstack.hdf,"-1)
```

Naturally, most standard tasks can be performed with the command-line programs, without any need to learn Python. However, users with basic knowledge of Python can make EMAN2 a much more powerful tool.

2.3. Supported image file formats

EMAN2 supports a wide range of image file formats (Table 1), and due to the library's modular structure, adding additional formats is a relatively trivial task in most cases. Whenever possible, full read/write support is provided. In some instances where formal format specifications were not available, such as the Gatan dm2/dm3 formats, insufficient information was available to offer a robust file

Table 1
Overview of the currently supported file formats in EMAN2

MRC	R/W	IMAGIC	R/W
SPIDER	R/W	HDF5	R/W
PIF	R/W	ICOS	R/W
VTK	R/W	PGM	R/W
Amira	R/W	Xplor	W
Gatan DM2	R	Gatan DM3	R
TIFF	R	Scans-a-lot	R
LST	R/W	PNG	R/W
Video-4-Linux	R	JPEG	W

New formats will be added upon request given the availability of a suitable format specification.

output routine. EMAN1 uses specific formats by default for various purposes, for example, the MRC format is the default for single large images and 3D reconstructions, and the IMAGIC format is the default for stacks of boxed-out particle images.

In EMAN2, the HDF5 file format is being adopted as a standard for internal use. Single particle reconstruction involves large amounts of metadata. Individual particles, for example, may have CTF parameters, information about which micrograph they came from and where, results of previous classifications, etc. Historically each file format defined a header containing any necessary metadata. Unfortunately in virtually all formats this header is fixed, with no mechanism for extension or modification. Meaning, if the information that must be stored is not defined in the header, either it cannot be stored, or one of the existing fields must be misappropriated for the task. The HDF5 (http://hdf.ncsa.uiuc.edu/HDF5) format was designed explicitly for scientific computation, and supports an extensible set of arbitrarily typed metadata entries to be associated with each image in any stack it contains. It is very similar to using an XML file with associated binary data, and, in fact, the metadata can be extracted from the HDF5 file in XML form. The library fully supports this mechanism, and when reading/writing images to HDF5 files, all metadata associated with an image, including any user-defined parameters, are automatically archived with the image. When loaded as Python objects, all metadata is immediately accessible by name through a dictionary/hash associated with the image.

EMAN2 will continue to provide full support for reading and writing other formats, however, HDF5 storage capabilities will be used within EMAN2 to store much of the metadata generated during processing. Routines will exist to extract and insert this metadata to/from the binary HDF5 files in a textual form, and the GUI tools will provide mechanisms for examining and mining such associated metadata. When converting to/from other file formats, any metadata supported by the header of the external format will, of course, be read/written, but none of the other available cryoEM formats offer sufficient flexibility to store all of the necessary metadata.

2.4. 3D transforms

One of the more difficult aspects of moving between software packages in cryoEM is not file format conversion, but Euler angle conversions. Each of the available packages uses a different convention for parameterizing 3D orientations. EMAN2 offers a Transform3D class (see Baldwin and Penzek in this issue), which incorporates the conventions for the most common orientation conventions, including: EMAN, IMAGIC, SPIDER, FREALIGN, MRC and several quaternion style conventions often used in visualization software. This transform class also serves as the core class governing unitary transformations and symmetry opertations. Internally, the transformation is represented as a 4×4 matrix, permitting arbitrary linear transformations plus translations. Typically such transformations are limited to unitary transformations, i.e. rigid body motion, and scaling, though the library also supports skewing by directly manipulating the matrix. The Transform3D object supports modifying and retrieving rotational, translational and scaling parameters, as well as converting to/from the various Euler/quaternion conventions. As an example of all this, consider the following simple sequence of python calls:

```
RA=Transform3D(EULER_SPIDER,0,30.0,0)
RA.set_posttrans(Vec3f(1,0,0))
RA_EMAN = RA.get_rotation(EULER_EMAN)
print(RA_EMAN["az"],RA_EMAN["alt"],
RA_EMAN["phi"])
```

This sequence defines RA, a 3D rotation in SPIDER notation, of 30 degrees around the *y*-axis, then modifies the transformation to include a translation along the *x*-axis after the application of the rotation. The rotational part of the transformation is next converted to EMAN, and the three Euler angles in EMAN's ZXZ' convention is printed out. Since the rotation is defined in the ZYZ' SPIDER convention, the ZXZ' rotation corresponds to (−90.0, 30.0, 90.0), and the value of 90.0 (that is, the first Euler) is printed to the screen. In EMAN2, all angles both in Python library calls and in end-user applications are specified in degrees.

2.5. GUI tools

There are several Python-compatible cross-platform GUI toolkits currently available, including Tkinter, PyGTK (www.pygtk.org), WxPython (www.wxpython.org) and PyQT (Rempt, 2001). Each of these toolkits has its own set of advantages and disadvantages. PHENIX is based on the WxPython toolkit, while EMAN1 makes use of the QT toolkit in C++. In EMAN2 we take a hybrid approach to make the package as flexible as possible. Current EMAN2 image display widgets use WxPython for PHENIX compatibility. Both of the example programs shown in Figs. 2 and 3 are written using the WxPython widgets. QT is a semi-commercial package (www.trolltech.com), free for use with free projects, and offers the best overall design and largest feature-set of the available toolkits. New EMAN2 development is now focused primarily on PyQT4.

At present, the GUI tools available in EMAN2 are quite primitive, and we rely heavily on the excellent set of tools provided by EMAN1. However, a suite of redesigned GUI tools for EMAN2 is under active development, and will be provided incrementally as they are completed. As the GUIs are now written directly in Python, a range of new capabilities can be provided in EMAN2 which would have been impractical to implement in EMAN1. For example, in

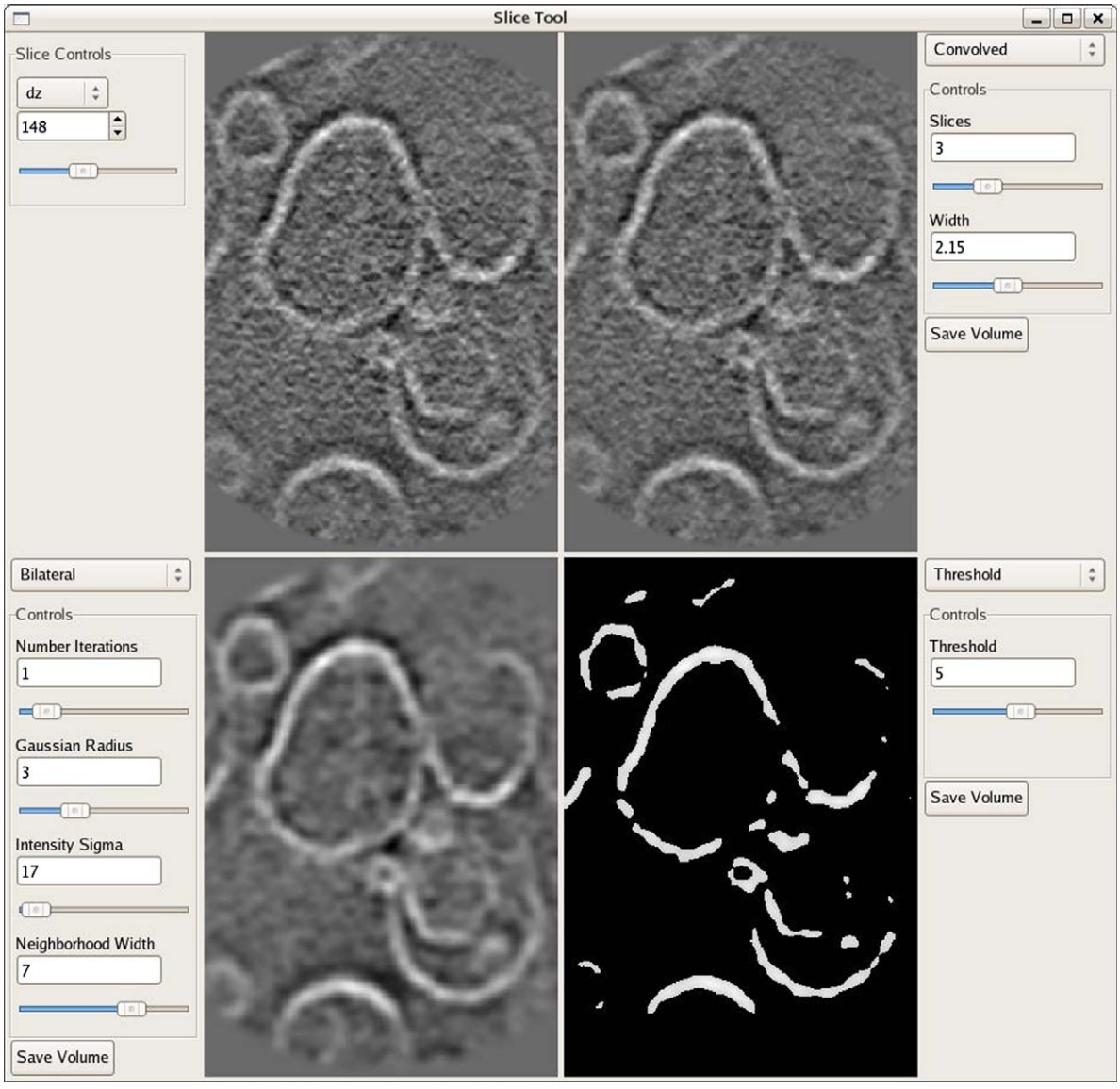

Fig. 3. The first full GUI application written in EMAN2. This is an interactive filtration tool for optimizing filters in tomographic reconstructions.

EMAN1, the file browser allowed the user to compare two images to each other in a variety of different ways. The list of comparisons was naturally fixed, and there was no capability to compare images in different files. In EMAN2, in addition to the fixed list of comparisons, an option will be provided for the user to enter their own 1-line python command to operate on the two images, which will then be dynamically applied as the user browses through different images. Again, similar to the design of EMAN1, EMAN2 will not have a single GUI interface for all aspects of the reconstruction, but rather a set of interrelated tools aimed at making specific tasks easier. An example of one of the first GUI programs written in EMAN2 is shown in Fig. 3. This program allows the user to apply a wide range of filters to slices from a tomographic reconstruction and interactively modify their parameters. This permits the user to rapidly determine the optimal filtration parameters for a particular reconstruction.

2.6. Parallelism

EMAN2 has adopted a two level system for parallelism. In this system, EMAN2 first breaks large processing tasks into modular pieces which need to be distributed to individual processors for computation. There will then be a variety of back-end programs for executing jobs in parallel using MPI or other parallelism systems. This separation opens the possibility of supporting other parallel computing environments like Condor (Litzkow et al., 1988) or systems similar to FoldingAtHome (Shirts and Pande, 2006) in the future, without any substantial internal changes. The first release of EMAN2 will support two options, MPI and execution on a single workstation. In addition, certain algorithms requiring fine-grained parallelism, not suited to the above mechanism, may also make explicit use of MPI. However, all such algorithms will also function in environments, such as a single desktop computer, where MPI is not available.

2.7. Extensibility

The deficiencies in EMAN1's internal design, like most packages, are historical in nature. For each category of algorithm, EMAN1 originally contained only a single algorithm, i.e. a single image comparison metric, a single 3D reconstruction algorithm, a single projection generation algorithm. Over time, this naturally expanded, such that there are now five different image similarity metrics, four different projection generation algorithms, and so on. Development of faster and/or more accurate algorithms continues at a rapid pace in the cryoEM community, driven by the struggle to achieve ever higher resolutions and devise new techniques for extracting more information from the raw image data. EMAN2 embraces this concept of ever-expanding algorithms by adopting a modular approach for each algorithm category. Each specific algorithm is named, takes a set of named parameters, and is individually documented. This full modular functionality is available from both Python and C++.

In the refactored library, when calling a method such as a filter, for example, there is a single 'process()' method which handles all available filters. Similarly, there is a single method for calculating any of the available image similarity metrics, 'cmp(),' performing image registration/alignment, 'align(),' and generating projections, 'project().' In addition, extensible classes exist for performing 3D reconstructions and combining sets of images using various mathematical techniques. For example, to apply a low-pass Gaussian filter to an image, one would call:

```
img2=img.process("filter.lowpass.gauss",
{"sigma":0.15})
```

This mechanism supports full introspection, that is, a function exists to determine each method's name, list of available parameters, and documentation at runtime. For example, if a 3D visualization program wished to permit arbitrary filtration of the model prior to visualization (Fig. 3), the library could be interrogated for a list of all currently available filters and the parameters for each, updating the GUI interactively. This design also simplifies the process of adding new filters. Templates are provided for each class of operations, and the programmer need only fill in the name, desired parameters, documentation and the code implementing the actual filter algorithm. The automatic build system handles everything else. After inserting the information in the appropriate template, running 'make' will cause the new method to be immediately available in C++, Python and any other applications making use of the introspection mechanism. The calling syntax for all of these modular functions is mirrored in both C++ and Python, so, in cases where performance is a concern, Python scripts can be painlessly converted into C++, with only minor syntactical changes.

In addition this mechanism permits certain routines to take the names of other routines as parameters. For example, when aligning one image to another, various similarity metrics could be used to judge the optimal alignment. The align() method takes as an argument the name of the 'cmp()' similarity metric to be used internally during alignment. This permits detailed control over internal functioning of complex algorithms without requiring recompilation or, in fact, any changes at all to the core library.

Internally, this infrastructure is implemented using several now-standard C++ programming design patterns (Gamma et al., 1995): Abstract Factories, Factory Methods and Singleton Classes. This permits easy organization of a large number of related classes implementing the modular functionality in the library, and providing C++ with many capabilities for dynamic behavior often limited to scripting languages. However, while this design is critical to the design of the EMAN2 core library, it is largely concealed, even from programmers wishing to extend the capabilities of the library. By providing templates for the various modular classes, it is possible to add new filters, comparators, and so on, without knowing anything about the internal design of the library, or complicated object oriented programming paradigms.

2.8. Documentation

EMAN2 also incorporates inline documentation as an important aspect of its overall design. All algorithms implemented in the C++ library are documented inline using JavaDoc (http://java.sun.com/j2se/javadoc) style comments. When building the library, this documentation is parsed using the doxygen (http://www.stack.nl/~dimitri/doxygen) documentation system to produce a complete set of fully linked reference pages for the entire EMAN2 library at the C++/Python level. Higher level documentation, such as user-level documentation for the individual filters and other modular routines, and for the end-user command-line programs is written by hand using an object-oriented database allowing the user to search and sort the manual using a variety of tags and keywords and even annotate manual pages with additional commentary. In addition, the manual can be accessed programmatically to produce a formatted printed manual representing a current snapshot of the full documentation. This design encourages the developers to keep the documentation synchronized with the actual code and for users to incorporate suggestions based on actual usage into the documentation.

2.9. EMAN2 vs. EMAN1

While much of the conceptual design of EMAN1 persists in EMAN2, there are also a number of fundamental changes. EMAN2 was designed at the outset to coexist with EMAN1, so users are not forced to immediately transition from one package to the other. While there is much overlap between the packages, each currently offers capabilities not available in the other. Eventually all useful EMAN1 capabilities will be integrated with EMAN2, but during the transition period, most users will want to have both packages installed.

To avoid naming conflicts with both EMAN1 and other image processing packages, all EMAN2 programs start with an 'e2' prefix. For example *proc2d* in EMAN1 has become *e2proc2d.py* in EMAN2. The '.py' extension denotes that the program is actually a Python script. This is the second major change. All user-programs in EMAN2 are written in Python rather than C++, though all of the compute-intensive operations are still occurring via the C++ library called from Python. This has no real impact on the typical end-user, aside from the fact that the more sophisticated users will be able to change the behavior of any of the EMAN2 programs without having to download the full source code, or, in fact, without having to recompile anything at all. As mentioned above, GUI programs are also being written in Python through the PyQt4 toolkit rather than in C++. This permits much easier customization of the GUI tools for specific purposes without requiring recompilation.

All arguments to command-line programs have now adopted the standard Unix command structure. For example, in EMAN1, a typical command to apply a Gaussian low-pass filter followed by a circular mask to a set of images is

```
proc2d in.img out.img apix=1.5 lp=9.5 mask=32
```

In EMAN2 this would remain quite similar:

```
e2proc2d.py in.img out.img --apix=1.5 --lp=9.5 --mask=32
```

However, order of operations is now respected in the *e2proc2d.py* and *e2proc3d.py* commands. In EMAN1, reversing the order of the lp= and mask= options would have no impact on the final result, the order of specific operations is fixed internally. In EMAN2, reversing the command order would also reverse the order of operations, changing whether the mask is filtered or not.

Much like EMAN1, full help is provided for all commands, so 'e2proc2d.py' alone will produce usage information, and 'e2proc2d.py –help' will provide detailed information on the meaning of each option. In addition, full documentation is available through a flexible user interface at http://blake.bcm.tmc.edu/EMAN2.

Historically, a significant problem for users trying to mix EMAN1 with other software packages was limited support for odd-sized images. For example, applying a low-pass filter to a 63×63 pixel image was impossible in EMAN1 without first resizing it to 62 or 64 pixels. EMAN2 fully supports both even and odd sized images for virtually all operations. Each case has its own self-consistent definition of the image center: $n/2$ for even sized images and $(n-1)/2$ for odd sized images (Fig. 4).

An example of EMAN2's flexibility, as discussed above, is the ability to use different FFT libraries. While FFTW remains as a choice, EMAN2 also contains an internal FFT routine for compatibility, and the capability to use vendor-optimized FFT libraries, such as those provided by Intel and AMD, will also be provided.

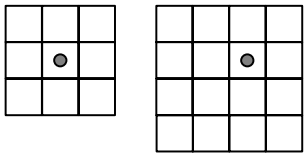

Fig. 4. Shows the center position used in EMAN2 for odd and even sized images. Note that the dot representing center is in the middle of a pixel in both cases. EMAN2 uses an origin of (0,0) when referring to pixel locations.

2.10. Cross-platform support

EMAN2 was designed with portability in mind, and only standard and widely implemented C++ language features are used in the library design. EMAN2 will fully support Linux, OSX and Windows. Other Unix platforms, such as SGI Irix and IBM AIX, will be supported upon request assuming availability of a suitable machine for compilation. Unlike EMAN1, it will be possible to run full refinements on the Windows platform in EMAN2.

3. Conclusions

EMAN2 represents a substantial advancement over EMAN1 in extensibility, ease of use and capabilities. As described earlier, the transition from EMAN1 to EMAN2 will be gradual. The entire EMAN2 core library and Python bindings are fully functional. Basic GUI widgets exist, though they are not yet feature-complete. A number of programs have already been ported from EMAN1, and several new programs have been created. Prerelease versions of the library are updated daily, and are available from http://blake.bcm.tmc.edu/EMAN2 with full source. Access to the EMAN CVS archive is available for those wishing to contribute code directly to the archive.

Acknowledgments

We thank Pawel Penczek and Chao Yang for their contributions to EMAN2 via the SPARX project. Current EMAN2 development has been supported by NIH P41RR02250, P01GM064692 and P01AI055672. Continuing development is supported by R01GM080139.

References

Adams, P.D., Gopal, K., Grosse-Kunstleve, R.W., Hung, L.W., Ioerger, T.R., McCoy, A.J., Moriarty, N.W., Pai, R.K., Read, R.J., Romo, T.D., et al., 2004. Recent developments in the PHENIX software for automated crystallographic structure determination. J. Synchrotron Radiat. 11, 53–55.

Adams, P.D., Grosse-Kunstleve, R.W., Hung, L.W., Ioerger, T.R., McCoy, A.J., Moriarty, N.W., Read, R.J., Sacchettini, J.C., Sauter, N.K., Terwilliger, T.C., 2002. PHENIX: building new software for automated crystallographic structure determination. Acta Crystallogr. D Biol. Crystallogr. 58, 1948–1954.

Baker, M.L., Jiang, W., Bowman, B.R., Zhou, Z.H., Quiocho, F.A., Rixon, F.J., Chiu, W., 2003. Architecture of the herpes simplex virus major

capsid protein derived from structural bioinformatics. J. Mol. Biol. 331, 447–456.

Baker, T.S., Cheng, R.H., 1996. A model-based approach for determining orientations of biological macromolecules imaged by cryoelectron microscopy. J. Struct. Biol. 116, 120–130.

Booth, C.R., Jiang, W., Baker, M.L., Zhou, Z.H., Ludtke, S.J., Chiu, W., 2004. A 9 angstrom single particle reconstruction from CCD captured images on a 200 kV electron cryomicroscope. J. Struct. Biol. 147, 116–127.

Bottcher, B., Wynne, S.A., Crowther, R.A., 1997. Determination of the fold of the core protein of hepatitis B virus by electron cryomicroscopy. Nature 386, 88–91.

Carragher, B., Whittaker, M., Milligan, R.A., 1996. Helical processing using PHOELIX. J. Struct. Biol. 116, 107–112.

Cheng, Y., Zak, O., Aisen, P., Harrison, S.C., Walz, T., 2004. Structure of the human transferrin receptor–transferrin complex. Cell 116, 565–576.

Crowther, R.A., Henderson, R., Smith, J.M., 1996. MRC image processing programs. J. Struct. Biol. 116, 9–16.

Fotin, A., Cheng, Y., Sliz, P., Grigorieff, N., Harrison, S.C., Kirchhausen, T., Walz, T., 2004. Molecular model for a complete clathrin lattice from electron cryomicroscopy. Nature 432, 573–579.

Frank, J., Radermacher, M., Penczek, P., Zhu, J., Li, Y., Ladjadj, M., Leith, A., 1996. SPIDER and WEB: processing and visualization of images in 3D electron microscopy and related fields. J. Struct. Biol. 116, 190–199.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995. Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Gao, H., Sengupta, J., Valle, M., Korostelev, A., Eswar, N., Stagg, S.M., Van Roey, P., Agrawal, R.K., Harvey, S.C., Sali, A., et al., 2003. Study of the structural dynamics of the *E. coli* 70S ribosome using real-space refinement. Cell 113, 789–801.

Grigorieff, N., 1998. Three-dimensional structure of bovine NADH:ubiquinone oxidoreductase (complex I) at 22 A in ice. J. Mol. Biol. 277, 1033–1046.

Hegerl, R., 1996. The EM program package: a platform for image processing in biological electron microscopy. J. Struct. Biol. 116, 30–34.

Heymann, J.B., 2001. Bsoft: image and molecular processing in electron microscopy. J. Struct. Biol. 133, 156–169.

Jiang, W., Baker, M.L., Ludtke, S.J., Chiu, W., 2001. Bridging the information gap: computational tools for intermediate resolution structure interpretation. J. Mol. Biol. 308, 1033–1044.

Jiang, W., Chang, J., Jakana, J., Weigele, P., King, J., Chiu, W., 2006. Structure of epsilon15 bacteriophage reveals genome organization and DNA packaging/injection apparatus. Nature 439, 612–616.

Kremer, J.R., Mastronarde, D.N., McIntosh, J.R., 1996. Computer visualization of three-dimensional image data using IMOD. J. Struct. Biol. 116, 71–76.

Liang, Y., Ke, E.Y., Zhou, Z.H., 2002. IMIRS: a high-resolution 3D reconstruction package integrated with a relational image database. J. Struct. Biol. 137, 292–304.

Litzkow, M.J., Livny, M., Mutka, M.W. 1988. Condor-a hunter of idle workstations. Distributed Computing Systems, 1988, 8th International Conference on, pp. 104–111.

Ludtke, S.J., Baldwin, P.R., Chiu, W., 1999. EMAN: semiautomated software for high-resolution single-particle reconstructions. J. Struct. Biol. 128, 82–97.

Ludtke, S.J., Chen, D.-H., Song, J.-L., Chuang, D.T., Chiu, W., 2004. Seeing GroEL at 6 Å resolution by single particle electron cryomicroscopy. Structure 12, 1129–1136.

Matadeen, R., Patwardhan, A., Gowen, B., Orlova, E.V., Pape, T., Cuff, M., Mueller, F., Brimacombe, R., van Heel, M., 1999. The *Escherichia coli* large ribosomal subunit at 7.5 Å resolution. Structure Fold. Des. 7, 1575–1583.

Milne, J.L., Shi, D., Rosenthal, P.B., Sunshine, J.S., Domingo, G.J., Wu, X., Brooks, B.R., Perham, R.N., Henderson, R., Subramaniam, S., 2002. Molecular architecture and mechanism of an icosahedral pyruvate dehydrogenase complex: a multifunctional catalytic machine. EMBO J. 21, 5587–5598.

Pettersen, E.F., Goddard, T.D., Huang, C.C., Couch, G.S., Greenblatt, D.M., Meng, E.C., Ferrin, T.E., 2004. UCSF Chimera—a visualization system for exploratory research and analysis. J. Comput. Chem. 25, 1605–1612.

Rempt, B., 2001. Python's PyQt toolkit. Dr. Dobb's Journal of Software Tools 26, 88.

Sanner, M.F., Stoffler, D., Olson, A.J. 2002. ViPEr, a Visual Programming Environment for Python, Paper presented at: 10th International Python conference.

Schroeter, J.P., Bretaudiere, J.P., 1996. SUPRIM: easily modified image processing software. J. Struct. Biol. 116, 131–137.

Shirts, M., Pande, V.S. 2006. Screen Savers of the World Unite! COMPUTING 10, p. 43.

Sorzano, C.O., Marabini, R., Velazquez-Muriel, J., Bilbao-Castro, J.R., Scheres, S.H., Carazo, J.M., Pascual-Montano, A., 2004. XMIPP: a new generation of an open-source image processing package for electron microscopy. J. Struct. Biol. 148, 194–204.

van Heel, M., Harauz, G., Orlova, E.V., Schmidt, R., Schatz, M., 1996. A new generation of the IMAGIC image processing system. J. Struct. Biol. 116, 17–24.

Volkmann, N., Hanein, D., 1999. Quantitative fitting of atomic models into observed densities derived by electron microscopy. J. Struct. Biol. 125, 176–184.

Wriggers, W., Milligan, R.A., McCammon, J.A., 1999. Situs: A package for docking crystal structures into low-resolution maps from electron microscopy. J. Struct. Biol. 125, 185–195.

Zhang, X., Walker, S.B., Chipman, P.R., Nibert, M.L., Baker, T.S., 2003. Reovirus polymerase lambda 3 localized by cryo-electron microscopy of virions at a resolution of 7.6 A. Nat. Struct. Biol. 10, 1011–1018.

Zhou, Z.H., Chiu, W., 2003. Structural determination of icosahedral viruses by electron cryomicroscopy at sub-nanometer resolution. Adv. Protein Chem. 64, 93–130.