Viewing diffraction images in CCTBX

Nathaniel Echols, Johan Hattne, Richard J. Gildea, Paul D. Adams, and Nicholas K. Sauter
Lawrence Berkeley National Laboratory, Berkeley, CA 94720

Correspondence email: NKSauter@LBL.Gov

Introduction

To facilitate ongoing efforts to analyze and process problematic diffraction images, we have developed a simple, embeddable application in CCTBX (Grosse-Kunstleve *et al.,* 2002) to view a wide variety of detector formats using the wxPython library (http://www.wxpython.org). In the past, *LABELIT* (Sauter 2011) was capable of generating image files containing annotated sections of the diffraction pattern, but this was limited to non-interactive use. Debugging of indexing and integration code necessitated a more robust and configurable interface capable of displaying images along with detailed visual feedback on the performance of the processing algorithms.

Program description

The standalone program can be run as *phenix.image_viewer* (also available in the PHENIX GUI under "Utilities"), and supports most common file formats (including ADSC, MAR, RAXIS, and CBF). The interface (Figure 1) is similar to the versatile X11-based image viewer *adxv* (http://www.scripps.edu/~arvai/adxv.html) in many respects, although we have diverged from this design where appropriate. (In particular, the functions of the left and middle mouse buttons have been swapped to simplify use on Macintosh laptops.) The image display itself occupies most of the main window, with view controls in a separate floating window. To aid navigation of large images at high magnification, the control window incorporates a thumbnail view of the overall image; clicking in this view recenters the main viewport. The thumbnail image incorporates a feature previously implemented in *LABELIT*: when sampling the raw image to produce the thumbnail, multiple image pixels are polled to produce each thumbnail pixel, with the highest intensity value being taken instead of the average value. This results in a much higher contrast that allows the lattice diffraction to be clearly visible in the thumbnail even for relatively weak images (Figure 1).

Internally, each detector image is essentially an integer array indexable by X,Y coordinates. This is converted to an ASCII string representing the image in a generic format, with the desired brightness and color scheme applied in place of the raw intensity values. The conversion is applied only when these settings are modified; scaling and panning is performed entirely by the built-in image manipulation functions in wxPython. Actual display is accomplished by rendering the appropriate section of the image as a bitmap. Although this method is less ideal at lower magnification due to the compression of spots and resulting loss of contrast, it allows interactive use to remain extremely fast. The zoom panel also allows display of individual intensity values superposed on the pixels (Figure 1), similar to features in *adxv* and *XDisplayF* (Otwinowski & Minor, 1997).

The viewer can easily be embedded in other applications, either as part of a larger interface or at the end of a command-line process. The "paint" method of the display panel is designed to allow extension with custom draw commands using the wxPython API. For instance, the following simple code overlays yellow circles around integrated spots:

```
def draw_spot_predictions (window, dc, image) :
  import wx
  scale = window.get_scale()
  predictions = image.get_drawable_predictions()
  dc.SetPen(wx.Pen((255,255,0), 1))
  dc.SetBrush(wx.TRANSPARENT_BRUSH)
  for (x, y) in predictions :
    dc.DrawCircle(x, y, 8*scale)
```

A slightly more complex real-world example is shown in Figure 2. The main limitation to these overlays is the number of function calls that can be performed before the interface slows noticeably. In practice, the programmer must incorporate information about the current scale and drawable area of the image, and simplify or clip the image annotations where appropriate. The Python class used to manage and convert image data provides a convenience method, get_drawable_points, that accepts as input a list of

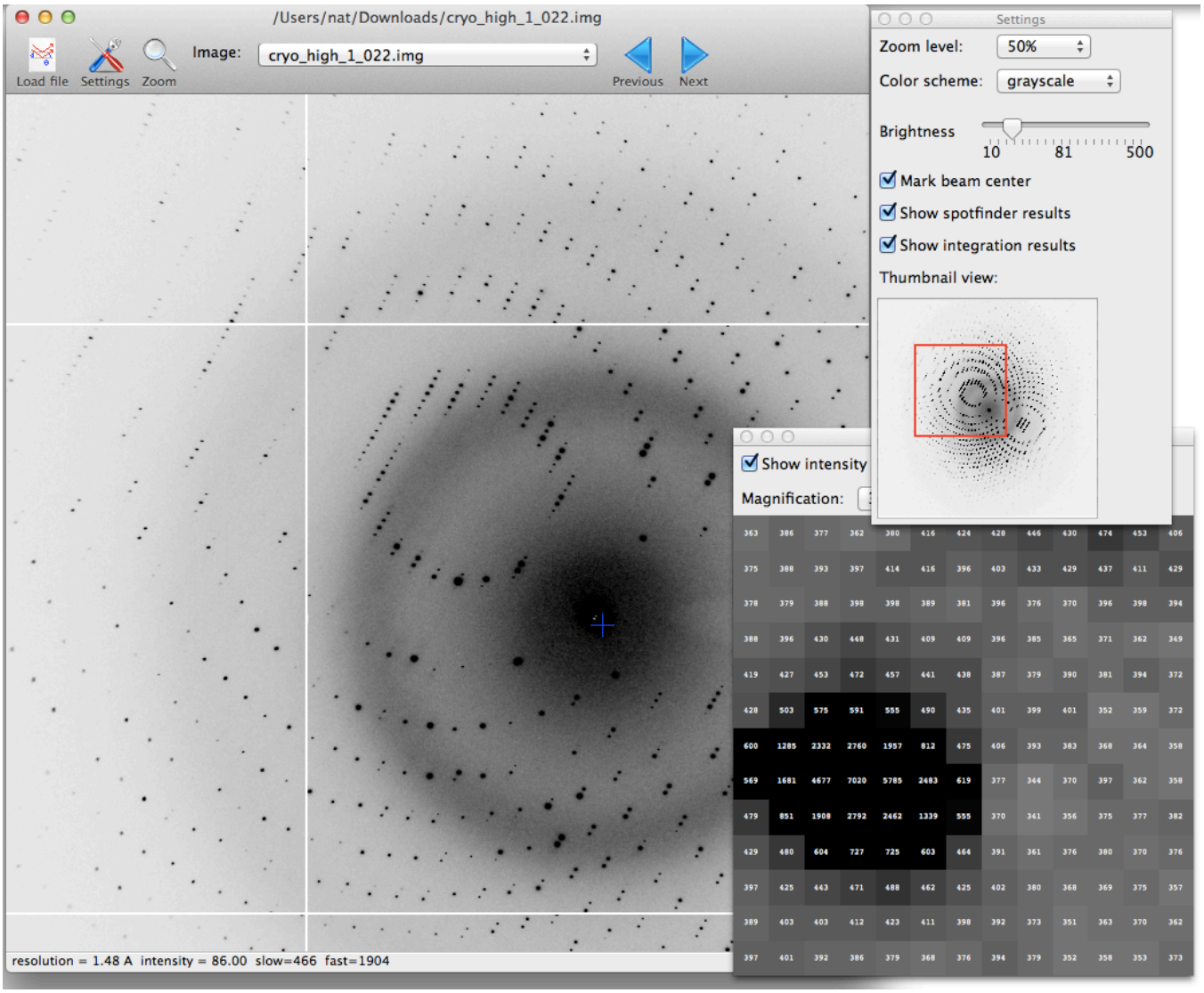

Figure 1: The main image viewer interface, showing a diffraction pattern from *E. coli* dihydrofolate reductase (provided by James Fraser, UCSF). The zoom window is visible at lower right.

x,y coordinates (*i.e.* detector pixels) and filters out those which occur outside the current view area.

The program *distl.image_viewer* (Figure 3) provides a simple example of how the module may be quickly re-used as a driver for additional computation. The settings panel is modified to control the applicaton *distl.signal_strength*, a spot-finding tool designed for rapid evaluation of crystal diffraction (Sauter, 2010). Spot detection is triggered by wxPython events, such as changing the "minimum spot area" field, and the interface is redrawn with the new predictions. The subclassed interface consists of less than 200 lines of Python code (rstbx/viewer/spotfinder_frame.py), most of which implements the additional controls and

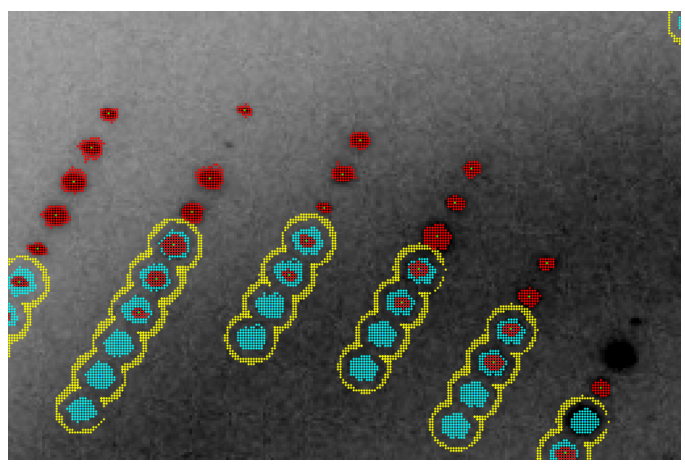

Figure 2: Closeup of the diffraction pattern shown in Figure 1 with the spotfinder results (red), integration masks (cyan), and background masks (yellow) overlaid.

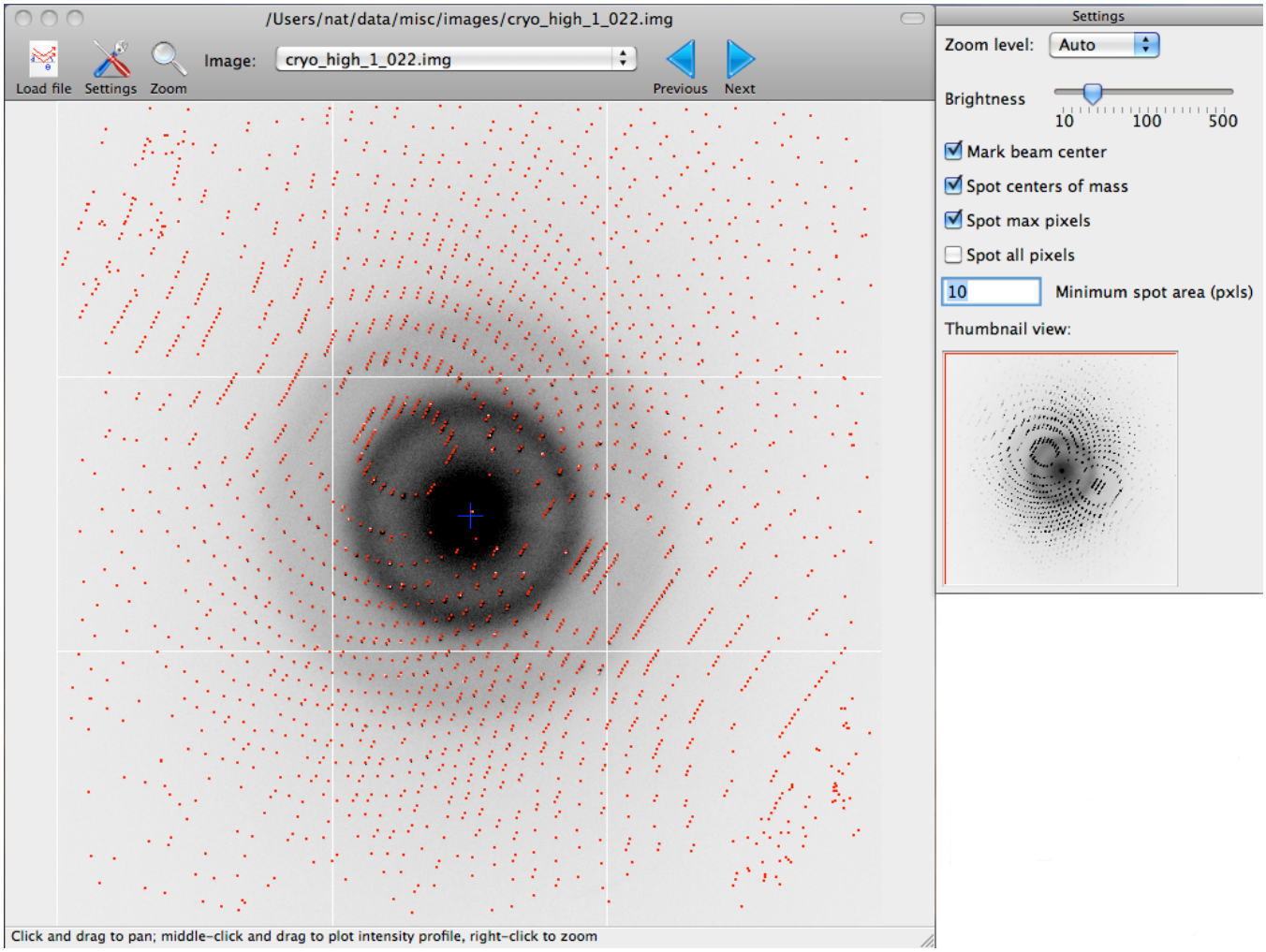

Figure 3: The spotfinder front end, *distl.image_viewer*.

event handling.

Future directions

We extensively use this module as a development aid. An NIH-funded project, "Realizing New Horizons in X-ray Crystallography Data Processing", is focused on problems in diffraction image processing that can not be handled automatically (or at all) by current software, such as split or multiple lattices (Sauter & Poon, 2010), severe anisotropy, and other common pathologies. Although the eventual goal is a suite of programs that can be run unattended (either graphically or from the command line), interactive, real-time visualization of the results will be continue to be essential, especially for the early stages of development. As the program framework is relatively simple and easily extended, we anticipate that it could also be adapted to monitoring live beamline data with minimal effort.

In the long term, we hope to integrate this tool with others developed by the CCI group, including alternative methods for displaying raw diffraction data (Sauter, 2011), visualization of reciprocal space (Echols & Adams, 2011), and analysis of processed data quality (Zwart *et al.*, 2005).

Availability

All code described here is open-source under the CCTBX license, and may be downloaded from http://cctbx.sf.net. The CCTBX builds do not include the required graphical libraries, so additional installation of wxPython (version 2.8.12.1 or newer) and Matplotlib (1.0.1 or newer) is necessary. (These are included in the *PHENIX* installers.)

Acknowledgments

We thank the NIH (grant numbers R01-GM095887 and P01-GM063210) and the *PHENIX* Industrial Consortium for financial support.

References

Echols, N. & Adams, P.D. (2011) Computational Crystallography Newsletter 2, 88-92.

Grosse-Kunstleve, R. W., Sauter, N.K., Moriarty N.W. & Adams, P.D. (2002). J Appl Cryst. 35, 126-136.

Otwinowski, Z. & Minor, W. (1997). Methods in Enzymology 276, 307-326.

Sauter, N.K. & Poon, B.K. (2010). J Appl Cryst. 43, 611-616.

Sauter, N.K. (2010). Computational Crystallography Newsletter 1, 18-23.

Sauter N.K. (2011). Computational Crystallography Newsletter 2, 15-24.

Zwart, P.H., Grosse-Kunsteleve, R.W., & Adams, P.D. (2005). CCP4 Newsletter Winter, Contribution 7.