The 4th International Workshop on FEL Science

DIXE: Software for Coherent X-ray Diffractive Imaging

N. M. Davidson, B. Abbey, H. M. Quiney and K. A. Nugent

The University of Melbourne

Thursday, September 1st 2011
The last decade has seen an advancement in diffractive imaging techniques due to a combination of superior experimental set-ups, growing computer power and the development of algorithms for image reconstruction.

Traditionally each researcher or group has written their own code for implementing the reconstruction algorithms - a task which is time consuming and error prone.

As experiments (and experimental collaborations) become larger and more complex, it is necessary that software is engineered and that standard software packages are created for the CDI community.

The software package HAWKE (F. Maia et. al.) exists for plane wave diffractive imaging, however, to the best of our knowledge there is no package publically available which also incorporates Fresnel diffractive imaging.
Introducing DIXE

DIXE - Diffractive Imaging for X-rays and Electrons

- Is a software library for performing:
  - Plane wave coherent diffractive imaging (Planar CDI)
  - Fresnel coherent diffractive imaging (FCDI)
    - Reconstruction of the white-field
    - Reconstruction of the exit-surface wave and transmission function of the sample
  - Phase-Diverse/Ptychographic image reconstruction

- In addition to a C++ library which users can import into their own C/C++ code, it also has wrapper routines for IDL users, and some basic command line tools.

- Various examples are provided to demonstrate how to use the C++ functions, IDL routines and command-line tools for both reconstruction and simulation.

- It is multi-platform - source code and instructions for installing are provided for Ubuntu (link), Mac (OS X 10.6), Windows with Cygwin. 32 bit binary files for IDL only are provided for Windows.

- The first developmental version of the software library is now ready for testing/use. It is available under the GNU General Public License.
How to use DIXE

1. Load input data
2. Initialise the reconstruction library
3. Iterate
4. Scale intensity
5. Apply support
6. Propagate to detector plane
7. Propagate to sample plane
8. Save Result
How to use DIXE

Functions are provided for reading hdf, tiff and ppm image files.

**In C++**

```cpp
// read in the data
Double_2D data;
read_image('data_file_name.tiff', data);

// read in the support shape
Double_2D support;
read_image('support_file_name.tiff', support);
```

**In IDL**

```idl```
data = CXS_READ_PPM(1024, 1024, 'data_file.ppm')
```idl```

etc. or use one of IDLs built in libraries to get the input in matrix form.
Using the software

The reconstruction code must be initialised for either plane-wave, Fresnel white-field, Fresnel or phase diverse reconstruction.

**In C++**

```cpp
// Create a complex 2D field which will hold the result of the reconstruction.
Complex_2D object_estimate(nx,ny);

// Create the planar CDI object which will be used to perform the reconstruction.
PlanarCDI planar(object_estimate);

// set the support and intensity
planar.set_support(support,false);
planar.set_intensity(data);
```

**In IDL**

```idl
CXS_INIT_PLANAR, data, support
```
How to use DIXE

1. Load input data
2. Initialise the reconstruction library
3. Iterate
4. Save Result

Run the reconstruction!

**In C++**

```cpp
planar.iterate();
```

**In IDL**

```idl
; Perform one iteration. The exit-
; surface-wave is return to 'result'
result = CXS_ITERATE()

; Perform 50 iterations
result = CXS_ITERATE(50)
```
**Save the result to file.**

**In C++**

```cpp
// write out the result:
// - as a complex binary file
write_cplx(temp_str.str(), object_estimate);
// - just the magnitude as an image
Double_2D object_mag;
object_estimate.get_2d(MAG, object_mag);
write_image("mag.tiff", object_mag);
```

**In IDL**

```idl
; Record the complex field to a binary file
CXS_WRITE_CPLX, result, 'result_file.cplx'
```

The magnitude and phase of the complex matrix can be viewed and saved using the standard IDL routines.
Software Options

Various aspects of the reconstruction can be configured:

- Changing between different algorithms (e.g. HIO to ER) or creating a custom algorithm.

C++

// to error reduction:
planar.set_algorithm(ER);
// or to a custom algorithm:
planar.set_custom_algorithm(0.5, 0, -1, 0, -1, 0, 0, 0, 0, 0);
// where the input parameters are coefficients to a set of projection operators.

IDL

CXS_SET_ALGORITHM, 'ER'
CXS_SET_CUSTOM_ALGORITHM, 0.5, 0, -1, 0, -1, 0, 0, 0, 0, 0

Available Algorithms

ER - error reduction, BIO - basic input-output, BOO - basic output-output, HIO - hybrid input-output, DM - difference map, SF - solvent-flipping, ASR - averaged successive reflections, HPR - hybrid projection reflection, RAAR - relaxed averaged alternating reflectors
Updating the support or use the shrink-wrap algorithm at any point in the reconstruction.

**C++**

```cpp
planar.set_support(support);
planar.apply_shrinkwrap()
```

**IDL**

```idl
CXS_SET_SUPPORT, my_support_matrix
CXS_APPLY_SHRINKWRAP
```
Updating the support or use the shrink-wrap algorithm at any point in the reconstruction.

C++

```c++
planar.set_support(support);
planar.apply_shrinkwrap()
```

IDL

```idl
CXS_SET_SUPPORT, my_support_matrix
CXS_APPLY_SHRINKWRAP
```

Starting with any exit-surface wave initialisation (e.g. reloading a result from a previous reconstruction), or use the default initialisation.

C++

```c++
// if a complex matrix called 'esw' has already been loaded or created
PlanarCDI planar(esw); // or
planar.set_exit_surface_wave(esw);
// or to initialise with random numbers do
planar.initialise_estimate();
// for Fresnel CDI you can also use
my_fresnel.set_transmission_function(transmission)
```

IDL

```idl
CXS_INIT_PLANAR, my_data, my_support, my_starting_point
CXS_INITIALISE_ESW
```
And there are many others options.. for example:

- Choice of image file to read in and out from.
- Saving a subset of results with the lowest error metric (e.g. for the HIO algorithm).
- Adding extra constraints such as charge flipping, enforce homogeneity of the material etc.
- Phase-diverse/ptychographic reconstruction may be performed in parallel or series.
- Automatic alignment of image positions (transverse to beam) for phase-diverse/ptychographic reconstruction.

If a user requires some special functionality which is not provided, they can easily create a customised reconstruction algorithm using the building blocks provided.
Fresnel Reconstruction in IDL (using C. Putkunz’s data)

- Reconstruction of the white-field at the detector:
  - Load images of the white-field intensity (in the detector plane) and support (zone plate plane)
  - Initialise the white-field reconstruction library
  - Perform 20 iterations of 3-plane propagation (detector $\rightarrow$ focal $\rightarrow$ zone-plate)

- Reconstruction of the sample:
  - Load the sample’s diffraction pattern and the support shape.
  - Initialise the Fresnel reconstruction library
  - Set some extra constraints (e.g. charge-flipping)
  - Perform 20 iterations to reconstruct the sample’s exit-surface-wave

- Save the result
Video Demonstration

```
bash-3.2$ idl
IDL Version 8.0 (linux x86_64 m64). (c) 2010, ITT Visual Information Solutions
Installation number: 97562-1.
Licensed for use by: University of Melbourne

IDL> .Compile CXS_interface.pro
```
List of Routines

- CXS_ADD_COMPLEX_CONSTRAINT_REGION
- CXS_APPLY_SHRINKWRAP
- CXS_APPLY_SUPPORT
- CXS_CLEAR_MEMORY
- CXS_GET_BEST_RESULT
- CXS_GET_ERROR
- CXS_GET_INTENSITY_AUTOCORRELATION
- CXS_GET_ROUND_SUPPORT
- CXS_GET_SUPPORT
- CXS_GET_TRANSMISSION_FUNCTION
- CXS_INITIALISE_ESW
- CXS_INT_FRESNEL
- CXS_INT_FRESNEL_WF
- CXS_INT_PHASE_DIVERSE
- CXS_INT_PLANAR
- CXS_ITERATE
- CXS_PRINT_ALGORITHM
- CXS_PROPAGATE_FROM_DETECTOR
- CXS_PROPAGATE_TO_DETECTOR
- CXS_READ_CPLX
- CXS_READ_DBIN
- CXS_READ_PPM
- CXS_READ_TIFF
- CXS_SCALE_INTENSITY
- CXS_SET_ALGORITHM
- CXS_SET_BEAM_STOP
- CXS_SET_CHARGE_FLIPPING
- CXS_SET_CUSTOM_ALGORITHM
- CXS_SET_INTENSITY
- CXS_SET_RELAXATION_PARAMETER
- CXS_SET_SUPPORT
- CXS_SET_TRANSUNITY_CONSTRAINT
- CXS_WRITE_CPLX
- CXS_WRITE_DBIN
Each “class” (box) is one module of the library with its associated functions.
Some simple tools are provided for the Windows and Linux/Unix terminal as a demonstrative tool or to obtain results quickly. However, they are less flexibility than using C++ or IDL.

- **CDI_reconstruction.exe** - For performing Planar or Fresnel reconstruction.
  
  ```
  CDI_reconstruction.exe <config filename> <reco_type> <seed>
  ```

- **PhaseDiverseFresnelRec.exe** - For performing phase-diverse and ptychographic reconstructions of Fresnel data.
  
  ```
  PhaseDiverseFresnelRec.exe <data list filename> <iterations> <sub-iterations>
  ```
  
  optional parameters:
  
  ```
  <beta> <gamma> <running-mode> <do alignment> <seed> <unity> <flipping>
  ```

- Conversion between image file formats.
Future Objectives

- In the short-term we plan to add functionality to the code for:
  - reconstructing with partial coherence and
  - data cleaning and raw data processing.

- As well as some technical improvements:
  - allow parallel processing,
  - add experimental parameters to the meta data of input/output images and
  - writing binding for the library in more languages (in particular Matlab and/or python).

- As more members of the community begin using the code, there is no doubt that bugs and extra requirements will be revealed. We plan to invest some effort in making the software as robust as possible.

- More documentation.
**Summary**

- CXS has been developing software for the coherent diffractive imaging community.
- The first developmental version of the software library is now ready for testing/use.
- We have already had success within CXS with new members using the software.
- The priority for future releases will be parallelisation of the code and added functionality for partial coherence.

http://www.ph.unimelb.edu.au/~ndavidson/cxs/

Acknowledgements go to various Latrobe and Melbourne University members of CXS who have tested the software or provided test data: M. Jones, C. Putkunz, A. Torrance, A. Carroll, M. B. Luu and I. Peterson.
EXTRA SLIDES
**SPEED**

<table>
<thead>
<tr>
<th>1.2 GHz Laptop with 1.5 GB RAM</th>
<th>Iteration per Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>PlanarCDI (ER)</td>
<td>3.3</td>
</tr>
<tr>
<td>PlanarCDI (HIO)</td>
<td>2.7</td>
</tr>
<tr>
<td>FresnelCDI - white field</td>
<td>2.0</td>
</tr>
<tr>
<td>FresnelCDI</td>
<td>2.7</td>
</tr>
<tr>
<td>FresnelCDI with Complex Constraints</td>
<td>1.8</td>
</tr>
</tbody>
</table>

**Table:** CPU time for a 1024×1024 pixel reconstruction.
## Memory Usage

<table>
<thead>
<tr>
<th>Method</th>
<th>Number of $N \times N$ pixel arrays used</th>
<th>With $p=4$, $N=1024$ (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PlanarCDI (ER)</td>
<td>4</td>
<td>17</td>
</tr>
<tr>
<td>PlanarCDI (HIO)</td>
<td>10</td>
<td>42</td>
</tr>
<tr>
<td>FresnelCDI - white field</td>
<td>4.5</td>
<td>19</td>
</tr>
<tr>
<td>FresnelCDI</td>
<td>8.5</td>
<td>36</td>
</tr>
<tr>
<td>FresnelCDI with complex constraints</td>
<td>$11.5 + c$</td>
<td>$57$ (c=2)</td>
</tr>
<tr>
<td>Phase-Diverse/Ptychography</td>
<td>$I(1 + X) + 2M/N$</td>
<td>$30,000$ ($I=400, M = 1100^2$, $X = \text{FresnelCDI}$)</td>
</tr>
</tbody>
</table>

**Table:** The memory used……. In total the memory used will be the factor given in the second column of Figure 2 multiplied by $pN^2$. ($N$ - image side length in pixels, $p$ - precision in bytes, $c$ - number of complex constraint regions, $I$ - number of images, $M$ - area, in pixels, of the phase-diverse image, $X$ - number of arrays needed for a single reconstruction )
A preliminary set of requirements can be found on the website. The functionality which the software should provide for reconstruction is:

**Essential**

- Planar reconstruction
- Reconstruct the phase of the white-field using 3-plane propagation
- Fresnel reconstruction
- Holographic 1 step reconstruction
- Refine the experimental parameters during reconstruction
- Update the support or use the shrink-wrap algorithm
- Be able to change between algorithms (e.g. HIO to ER)
- Stop a reconstruction, save the result and restart at the same place
- Perform Simulations

**Very Useful**

- Ptychography
- Additional Constraints

**Extension**

- Partial CDI reconstruction
- Form an initial support (e.g. based on autocorrelation)
- Automatically check for stagnation during reconstruction
- Automatically check that transmission function is consistent with object properties (e.g. check for phase wrapping)
- Automatically check for consistency between ESW and object properties (e.g. check for phase wrapping)
- Estimate the resolution

✓ - implemented
Requirements for cleaning raw data prior to input into the reconstruction algorithms:

**Essential**
- ✓ Read HDF files and convert between different file formats
- ✓ Darkfield Subtraction
- ✓ Get data on a log scale

**Very Useful**
- Plot the average intensity versus frame number, full-frame and ROI
- Check for saturation/dead pixels
- Run correlations on data and white-fields
- Extract ring current, motor positions etc. from HDF files
- Check the beam stability in data over time
- Check correlations after dark-field subtraction
- ✓ Merge (add or average) data files together
- Determine the normalisation of the white-field to data
- ✓ Get the autocorrelation function

**Extension**
- Scroll through O(100) low resolution version of the data
Quick Links

All questions and comments should be directed to nadiamd@unimelb.edu.au

Package source code tar-ball - Release 0.1 (this is a test release with basic functionality).

Some preliminary specifications (written by Brian) - please give feedback on them!

Doxygen documentation of the source code (updated daily)

User questionnaire

Package Description

How to Get Started Easily

How to Install

How to Use
C++ Functions - Doxygen

The class which performs Fresnel CDI reconstruction. More...

#include <FresnelCDI.h>

Public Member Functions

FresnelCDI (Complex 2D &init_guess, Complex 2D &white_field, double beam_wavelength, double focal_detector_length, double focal_sample_length, double pixel_size, double normalisation=1.0, int n_best=1)
virtual ~FresnelCDI ()
virtual void initialise_estimate (int seed=0)
virtual void get_transmission_function (Complex 2D &result, bool enforce_unit_mag=true)
virtual void scale_intensity (Complex 2D &c)
virtual void propagate_from_detector (Complex 2D &c)
virtual void propagate_to_detector (Complex 2D &c)
void set_normalisation (double normalisation)
void set_experimental_parameters (double beam_wavelength, double focal_detector_length, double focal_sample_length, double pixel_size)
virtual void iterate ()
Complex 2D * get_best_result (double &error, int index=0)
void set_support (const Double 2D &object_support, bool soften=false)
void set_intensity (const Double 2D &detector_intensity)
void set_beam_stop (const Double 2D &beam_stop_region)
void set_relaxation_parameter (double relaxation_parameter)
void set_intensity_autocorrelation (Double 2D &autoc)
void set_algorithm (int alg)
void set_custom_algorithm (double m1, double m2, double m3, double m4, double m5, double m6, double m7, double m8, double m9, double m10)
void print_algorithm ()
double get_error ()
virtual void apply_shrinkwrap (double gauss_width=1.5, double threshold=0.1)
void get_support (Double 2D &support)
virtual void apply_support (Complex 2D &c)
virtual void project_intensity (Complex 2D &c)
void set_fft_type (int type)
void set_complex_contrain_function (void (*)(Complex 2D *)&transmission))

Static Public Member Functions

static int getAlgFromName (std::string algorithm_name)

Protected Member Functions

void apply_threshold (Double 2D &array, double threshold)
Introduction
Using the software
Plans and summary

Doxygen cont.

static std::map<std::string, 
    int * > alpNameMap = PlanarCDI::set_up_algorithm_name_map();

Detailed Description

Author:
Nadia Davidson <nadandrimmelb.edu.au>

The class used for performing Fresnel CDI reconstruction (for white-field reconstruction see FresnelCDI_WF). It inherits most methods from PlanarCDI, so please look at the documentation of this class also. Although there are some differences in the underlying code between this class and the planar case, the interface is generally unchanged. Therefore users should refer to the instructions for PlanarCDI to understand how to use a FresnelCDI object in their own code. Only the differences relevant to users will be documented here.

Definition at line 27 of file FresnelCDI.h.

Constructor & Destructor Documentation

FresnelCDI (Complex_2D & initial_guessed,
    Complex_2D & white_field,
    double beam_wavelength,
    double focal_detector_length,
    double focal_sample_length,
    double pixel_size,
    double normalisation = 1.0,
    int n_best = 1
    )

Create a FresnelCDI object. All of the dimensions which are given as input should be in the same units, e.g. metres.

Parameters:
initial_guess The complex 2-D field which is modified by this object. This represents the exit-surface-wave of the sample. It may be pre-initialised to a best first guess (e.g. manually); it may be loaded from a file (e.g. the output from a previous reconstruction job); or it may be uninitialised, and the default initialisation provided by FresnelCDI may be used.
white_field The reconstructed white-field (which has been previously determined using FresnelCDI_WF or otherwise)
beam_wavelength Wavelength of the beam
focal_detector_length The distance from the focal point to the detector
focal_sample_length The distance from the focal point to the sample
pixel_size The diameter of one of the detector pixels.
normalisation The amount to scale the data before subtracting the white field illumination. By default this is 1.0.

Definition at line 16 of file FresnelCDI++. 

References Complex_2D::copy(), illumination, pixel_length, PlanarCDI::set_algorithm(), set_experimental_parameters(), and wavelength.

virtual ~FresnelCDI () [inline, virtual]