IPLT Documentation

Release 0.9.7

The IPLT authors

August 27, 2013

CONTENTS

1 Introduction **1**
- 1.1 Vision . . . 1
- 1.2 Overview . . . 1

2 Download and installation **3**
- 2.1 Binary packages . . . 3
- 2.2 Compiling from Source . . . 4

3 Modules **7**
- 3.1 iplt module: General classes for crystallographic processing . . . 7
- 3.2 iplt.alg module: Image and reflection list algorithms . . . 26
- 3.3 iplt.gui module: Graphical user interface . . . 126

4 Tutorials and Scripts **137**
- 4.1 Tutorials . . . 137
- 4.2 Short IPLT scripts perform common and useful tasks . . . 151
- 4.3 User contributed examples . . . 164

5 Diffraction processing **169**
- 5.1 Introduction . . . 169
- 5.2 Project creation . . . 169
- 5.3 Project management . . . 170
- 5.4 Diffraction data extraction . . . 172
- 5.5 Diffraction data merging . . . 173
- 5.6 Graphical user interface . . . 174
- 5.7 Command line interface . . . 176

6 Publications **185**

7 Contact **187**
- 7.1 Main developer . . . 187
- 7.2 Contributors . . . 187
- 7.3 Former Contributors . . . 187
- 7.4 Mailing Lists . . . 187
- 7.5 Financial support . . . 188
- 7.6 Contributions . . . 188
- 7.7 Website and Hosting . . . 188

Python Module Index **189**

CHAPTER

ONE

INTRODUCTION

1.1 Vision

The Image Processing Library & Toolbox aims to provide a comprehensive and powerful library for the electron microscopy community, in particular for the small fraction of 2d electron crystallographers among them. We use IPLT in our novel methodological development and re-implementation of existing 2DEX protocols, both for image processing as well as for diffraction processing of 2D crystals.

From the beginning on, the design emphasis of IPLT was placed on modularity, flexibility, and providing several levels for interested developers from the community to extend the framework.

1.2 Overview

At the heart of the modularity in IPLT is the concept of an algorithm object. Instead of providing a long list of image class member functions that would bloat the interface, algorithm objects ensure orthogonal design and code independence. In a nutshell, an algorithm object is applied to an image using either the `Apply` or `ApplyIP` methods of the `ImageHandle` interface. Each algorithm object may maintain a state, as thus makes it easier to store and retrieve potentially complicated results.

Another aspect of modularity is the availability of the main IPLT interface in two languages, namely C++ and Python. All IPLT modules are written in C++, and reflected to Python using the boost python library[1]. While both languages have completely different features and application purposes, the IPLT syntax in both is remarkably similar.

The following code snippets demonstrate these modularity features:

C++

```
ImageHandle image=ost::io::LoadImage("file.png");
ost::img::alg::Stat stat;
image.Apply(stat);
std::cout << "mean is " << stat.GetMean() << std::endl;
```

Python

```
image=ost.io.LoadImage("file.png")
stat=ost.img.alg.Stat()
image.Apply(stat)
print "mean is %f"%(stat.GetMean())
```

[1] http://www.boost.org

CHAPTER

TWO

DOWNLOAD AND INSTALLATION

2.1 Binary packages

Binary packages of IPLT are currently provided for Linux, Mac OS X and Windows. To download choose your platform:

2.1.1 Linux

The binary package for Linux was built on Centos 6.3 contains a stand-alone self-contained version of IPLT. It supports various Linux distribution.

Standalone package

To install IPLT (32-bit version) download the latest package:

- IPLT-0.9.7-Linux.tgz

Unpack the archive by using following command in a terminal:

tar -xzf <package name>.tgz

You can launch giplt or the giplt_diffraction_manager by changing to the package bin directory and then running the corresponding program:

```
cd <package name>/bin
./giplt
```

or

```
cd <package name>/bin
./giplt_diff_manager
```

2.1.2 Mac OSX

To install IPLT on Mac OSX (10.5,10.6) download the disk image. Once it is downloaded open it and drag the IPLT application and optionally the diffraction processing and correlation averaging applications to the Applications folder.

- IPLT-0.9.7-Darwin.dmg

2.1.3 Windows

To install IPLT on Windows download and run the installer application, which will guide you through the installation process.

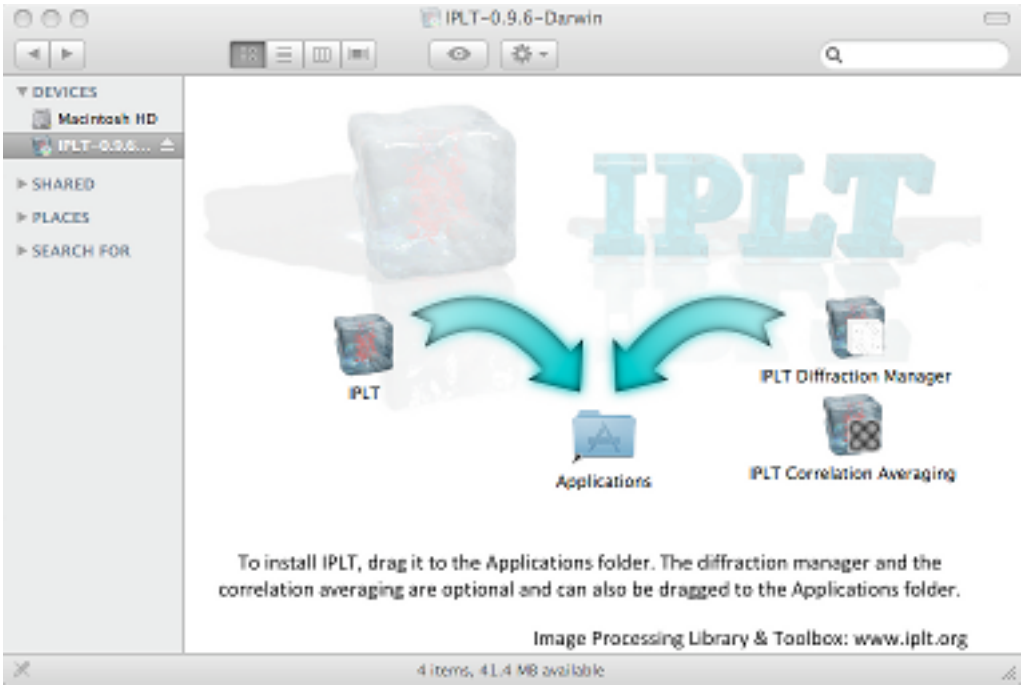

Figure 2.1: OSX Installer

- IPLT-0.9.6-Windows.exe

While a Windows version of IPLT is provided, it should be noted that the majority of the IPLT development and testing is performed on Linux an Mac OSX. For critical tasks it is therefore preferable to use either the Linux or OSX version of IPLT instead of the Windows version.

2.2 Compiling from Source

Alternatively IPLT can also be built from source. The building and installation instruction can be found here:

2.2.1 Building and Installing IPLT from Source

Installing the dependencies

IPLT is based on OpenStructure and uses the GNU Scientific library (gsl) in addition to the OpenStructure dependencies to perform some calculations. Therefore before IPLT can be compiled the following dependencies have to be installed:

- All dependencies for OpenStructure[1]
- gsl[2]
- matplotlib[3]

Downloading IPLT

IPLT uses git as the revision control system. The main repository can be browsed here[4]. To get the source code, use git clone:

[1] http://www.openstructure.org
[2] http://www.gnu.org/software/gsl/
[3] http://matplotlib.sourceforge.net/
[4] https://dng.biozentrum.unibas.ch/git/

```
git clone https://dng.biozentrum.unibas.ch/git/iplt.git <directory-name>
```

IPLT includes OpenStructure as a submodule which gets automatically built upon building of IPLT. To pull down the ost submodule use:

```
git submodule update --init
```

within the main iplt directory.

Configuring

IPLT uses CMake[5] for compiling and building the project. The next required step is to configure the build environment using cmake. You can do that by creating a build directory and invoking cmake within the built directory:

```
mkdir build
cd build
cmake .. <options>
```

There are two kinds of options: Options that let you control the building behaviour and options that let you tell CMake where to find the dependencies. All of them are passed to CMake with via -D<opt>=<value>.

Flags to Control the Dependencies

By default, CMake searches the standard directories for dependencies. However, on some systems, this might not be enough. Here is a short description of how CMake figures out what dependencies to take and how you can influence it.

- Boost is mainly controlled via the BOOST_ROOT option. If boost wasn't found, it should be set to the prefix of the boost installation.
- QT_QMAKE_EXECUTABLE defines the exact Qt installation to take. It should be set to the full path to qmake.
- PYTHON_ROOT is the Python equivalent of BOOST_ROOT. It should be set to the prefix path containing the python binary, headers and libraries.
- SYS_ROOT controls the general prefix for searching libraries and headers. By default, it is set to /.

Build Options

- If OPTIMIZE is set to 1, an optimized version of IPLT is built.

Building the Project (Linux/OSX)

Invoke make in the build directory. If you are using a multi-core machine, you can use the -j flag to run multiple jobs at once. For example to use 4 jobs simultaneously type:

```
make -j 4
```

Building the Project (Windows)

On Windows cmake will generate a solution file that you can open with Microsoft Visual Studio to build IPLT.

[5] http://cmake.org/

Creating packages (optional)

IPLT uses CPack to generate binary software packages for installation. To create a binary package use the following command in your build directory:

```
make package
```

CHAPTER

THREE

MODULES

IPLT is organized into three main modules:

3.1 iplt module: General classes for crystallographic processing

3.1.1 Lattice

Encapsulates a lattice defined by two lattice vectors $\vec{a}$ and $\vec{b}$, an origin $\vec{o}$, a barrel distortion K_b and a spiral distortion K_s. The vector $\vec{l}$ of a lattice point at index (h,k) is given by the following formula:

$$\vec{l} = \vec{o} + \begin{pmatrix} 1 + K_b \left|h\vec{a} + k\vec{b}\right|^2 & K_s \left|h\vec{a} + k\vec{b}\right|^2 \\ -K_s \left|h\vec{a} + k\vec{b}\right|^2 & 1 + K_b \left|h\vec{a} + k\vec{b}\right|^2 \end{pmatrix} \left(h\vec{a} + k\vec{b}\right)$$

For zero distortion constants, this simplifies to $\vec{l} = \vec{o} + h\vec{a} + k\vec{b}$. All values are dimensionless, but since a lattice is usually determined and/or used for an image, it's units should be considered to be pixels.

Examples

```
import iplt
lat=iplt.Lattice(Vec2(10,0), Vec2(0,15), Vec2(50,50))
print lat.CalcPosition(Point(10,10))
lat.SetBarrelDistortion(1e-6)
print lat.CalcPosition(Point(10,10))
```

Python interface

class `iplt.`**`Lattice`**

`CalcComponents`(*(Lattice)arg1*, *(object)postion*) → object :

Determines the fractional (h,k) index at the given fractional pixel position (x,y)

C++ signature : geom::Vec2 CalcComponents(iplt::Lattice {lvalue},geom::Vec2)

CalcComponents((Lattice)arg1, (Point)position) -> object : Determines the fractional (h,k) index at the given integer pixel position (x,y)

C++ signature : geom::Vec2 CalcComponents(iplt::Lattice {lvalue},ost::img::Point)

`CalcPosition`(*(Lattice)arg1*, *(object)index*) → object :

Calculates L based on the given fractional (h,k) index

C++ signature : geom::Vec2 CalcPosition(iplt::Lattice {lvalue},geom::Vec2)

CalcPosition((Lattice)arg1, (Point)index) -> object : Calculates L based on the given integer (h,k) index

C++ signature : geom::Vec2 CalcPosition(iplt::Lattice {lvalue},ost::img::Point)

DistanceTo(*(Lattice)arg1*, *(object)arg2*) → float :

C++ signature : float DistanceTo(iplt::Lattice {lvalue},geom::Vec2)

DistanceTo((Lattice)arg1, (Point)arg2) -> float :

C++ signature : float DistanceTo(iplt::Lattice {lvalue},ost::img::Point)

GetBarrelDistortion(*(Lattice)arg1*) → float :

C++ signature : float GetBarrelDistortion(iplt::Lattice {lvalue})

GetFirst(*(Lattice)arg1*) → object :

C++ signature : geom::Vec2 GetFirst(iplt::Lattice {lvalue})

GetOffset(*(Lattice)arg1*) → object :

C++ signature : geom::Vec2 GetOffset(iplt::Lattice {lvalue})

GetSecond(*(Lattice)arg1*) → object :

C++ signature : geom::Vec2 GetSecond(iplt::Lattice {lvalue})

GetSpiralDistortion(*(Lattice)arg1*) → float :

C++ signature : float GetSpiralDistortion(iplt::Lattice {lvalue})

SetBarrelDistortion(*(Lattice)arg1*, *(float)arg2*) → None :

C++ signature : void SetBarrelDistortion(iplt::Lattice {lvalue},float)

SetFirst(*(Lattice)arg1*, *(object)arg2*) → None :

C++ signature : void SetFirst(iplt::Lattice {lvalue},geom::Vec2)

SetOffset(*(Lattice)arg1*, *(object)arg2*) → None :

C++ signature : void SetOffset(iplt::Lattice {lvalue},geom::Vec2)

SetSecond(*(Lattice)arg1*, *(object)arg2*) → None :

C++ signature : void SetSecond(iplt::Lattice {lvalue},geom::Vec2)

SetSpiralDistortion(*(Lattice)arg1*, *(float)arg2*) → None :

C++ signature : void SetSpiralDistortion(iplt::Lattice {lvalue},float)

__init__(*(object)arg1*) → None :

Default ctor, will set first vector to {1,0}, second vector to {0,1}, and origin to {0,0}

C++ signature : void __init__(_object*)

__init__((object)arg1, (object)first, (object)second [, (object)origin]) -> None : Initialization with first and second vector, the origin defaults to {0,0} if not explicitly given

C++ signature : void __init__(_object*,geom::Vec2,geom::Vec2 [,geom::Vec2])

__init__((object)arg1, (object)arg2, (object)arg3, (object)arg4, (float)arg5, (float)arg6) -> None :

C++ signature : void __init__(_object*,geom::Vec2,geom::Vec2,geom::Vec2,float,float)

__init__((object)arg1, (object)arg2, (Point)arg3, (object)arg4, (Point)arg5, (object)arg6, (Point)arg7 [, (float)arg8 [, (float)arg9]]) -> None :

C++ signature : void __init__(_object*,geom::Vec2,ost::img::Point,geom::Vec2,ost::img::Point,geom::Vec2,ost::i
[,float [,float]])

iplt.**InvertLattice**(*(Lattice)arg1*) → Lattice :
Returns the inverted form of the given lattice

C++ signature : iplt::Lattice InvertLattice(iplt::Lattice)

The inversion is defined by:

$$P_{inv} = \frac{1}{D}\begin{pmatrix} -q_y \\ q_x \end{pmatrix}, Q_{inv} = \frac{1}{D}\begin{pmatrix} p_y \\ -p_x \end{pmatrix}, D = p_y q_x - p_x q_y$$

C++ Interface

class **iplt::Lattice**

encapsulates a 2D lattice

A lattice consists of two non-parallel vectors and a specific offset. Since it is usually associated with an image, its units are meant to be pixels!

Public Functions

Lattice()

Default is {1,0} {0,1} {0,0}.

Lattice(const geom::Vec2 & v1, const geom::Vec2 & v2, const geom::Vec2 & offset = geom::Vec2 (0.0, 0.0))

initialization with two non-parallel vectors

Lattice(const geom::Vec2 & v1, const geom::Vec2 & v2, const geom::Vec2 & offset, Real k3, Real s3)

initialization with two non-parallel vectors

Lattice(const geom::Vec2 & v1, const ost::img::Point & p1, const geom::Vec2 & v2, const ost::img::Point & p2, const geom::Vec2 & v3, const ost::img::Point & p3)

Lattice(const geom::Vec2 & v1, const ost::img::Point & p1, const geom::Vec2 & v2, const ost::img::Point & p2, const geom::Vec2 & v3, const ost::img::Point & p3, Real k3, Real s3 = 0)

Lattice(const geom::Vec2 & v1, const ost::img::Point & indx1, const geom::Vec2 & v2, const ost::img::Point & indx2, const geom::Vec2 & offset = geom::Vec2 (0.0, 0.0))

initialization

calculates a lattice based and vectors and indices (instead of the default (1,0) and (0,1))

void **SetFirst**(const geom::Vec2 & v)

set first vector

const geom::Vec2 & **GetFirst**()

retrieve first vector

void **SetSecond**(const geom::Vec2 & v)

set second vector

const geom::Vec2 & **GetSecond**()

retrieve second vector

const geom::Vec2 & **GetOffset**()

retrieve overall offset

void **SetOffset**(const geom::Vec2 & o)

set overall offset

void **SetBarrelDistortion**(Real k3)

Real **GetBarrelDistortion**()

void **SetSpiralDistortion**(Real k3)

Real **GetSpiralDistortion**()

Real **DistanceTo**(const ost::img::Point & p)

calc distance of a ost::img::Point to nearest lattice ost::img::Point

Real **DistanceTo**(const geom::Vec2 & v)

geom::Vec2 **CalcPosition**(const ost::img::Point & hk)

returns position based on index

geom::Vec2 **CalcPosition**(const geom::Vec2 & hk)

geom::Vec2 **CalcComponents**(const ost::img::Point & p)

return the components of a ost::img::Point

A ost::img::Point on the lattice should fullfill the following equation, where p is the ost::img::Point, a the first vector, b the second, and o the offset, and n,m are integers

p_x = n * a_x + m * b_x + o_x p_y = n * a_y + m * b_y + o_y

For ost::img::Points not on the lattice, m and n will have fractional components. This method returns m and n in the result geom::Vec2, with both the integer and fractional part

geom::Vec2 **CalcComponents**(const geom::Vec2 & v)

bool **operator==**(const *Lattice* & lat)

bool **operator!=**(const *Lattice* & lat)

DLLEXPORT_IPLT_BASE *Lattice* **InvertLattice**(const *Lattice* & lat)

invert domain of lattice spatial <-> reciprocal

3.1.2 Spatial and Reciprocal Unit Cells

The unit cell defined by the two classes `SpatialUnitCell` and `ReciprocalUnitCell` is a special case of a full 3D unit cell, tailored for use in 2D crystallography: it's parameters are given by the two axis lengths A and B, the angle γ between them, and a thickness C which represents the third dimension along the z-axis.

The distinction between spatial and reciprocal unit cell is for convenience purposes, since one form is implicitly convertible to the other (see Examples below).

Examples

```
import math, ex
su=ex.SpatialUnitCell(85,112,radians(120))
ru=ex.ReciprocalUnitCell(su) # conversion spatial->reciprocal
print "Reciprocal parameters: %g %g %g"%(ru.GetA(),ru.GetB(),ru.GetGamma())
su2=ex.SpatialUnitCell(ru) # conversion reciprocal->spatial
```

Interface

class `iplt.`**`SpatialUnitCell`**

`GetA` (*(_CellImpl)arg1*) → float :
Returns the length of the first unit cell axis

C++ signature : float GetA(iplt::cell_detail::CellImpl {lvalue})

`GetB` (*(_CellImpl)arg1*) → float :
Returns the length of the second unit cell axis

C++ signature : float GetB(iplt::cell_detail::CellImpl {lvalue})

`GetC` (*(_CellImpl)arg1*) → float :
Returns the thickness C

C++ signature : float GetC(iplt::cell_detail::CellImpl {lvalue})

`GetGamma` (*(_CellImpl)arg1*) → float :
Returns the gamma angle

C++ signature : float GetGamma(iplt::cell_detail::CellImpl {lvalue})

`GetPosition` (*(_CellImpl)arg1*, *(Point)arg2*) → object :

C++ signature : geom::Vec2 GetPosition(iplt::cell_detail::CellImpl {lvalue},ost::img::Point)

`GetSymmetry` (*(_CellImpl)arg1*) → Symmetry :

C++ signature : iplt::Symmetry GetSymmetry(iplt::cell_detail::CellImpl {lvalue})

`GetVecA` (*(_CellImpl)self*) → object :
Returns the first unit cell axis

C++ signature : geom::Vec2 GetVecA(iplt::cell_detail::CellImpl {lvalue})

`GetVecB` (*(_CellImpl)arg1*) → object :
Returns the second unit cell axis

C++ signature : geom::Vec2 GetVecB(iplt::cell_detail::CellImpl {lvalue})

`SetC` (*(_CellImpl)arg1*, *(float)arg2*) → None :

C++ signature : void SetC(iplt::cell_detail::CellImpl {lvalue},float)

`SetSymmetry` (*(_CellImpl)arg1*, *(str)arg2*) → None :

C++ signature : void SetSymmetry(iplt::cell_detail::CellImpl {lvalue},std::string)

SetSymmetry((_CellImpl)arg1, (Symmetry)arg2) -> None :

C++ signature : void SetSymmetry(iplt::cell_detail::CellImpl {lvalue},iplt::Symmetry)

__init__ (*(object)arg1*, *(float)arg2*, *(float)A*, *(float)B*[, *(float)gamma*[, *(str)symmetry*]]) → None :

Creates a spatial unit cell with the given values for A, B and gamma, plus an optional thickness C and symmetry

C++ signature : void __init__(_object*,float,float,float [,float [,std::string]])

__init__((object)arg1, (ReciprocalUnitCell)ruc) -> None : Creates a spatial unit cell from a reciprocal one

C++ signature : void __init__(_object*,iplt::ReciprocalUnitCell)

__init__((object)arg1, (Lattice)arg2, (object)arg3 [, (float)arg4 [, (str)lattice,sampling,thicknes,symmetry]]) -> N
Creates a spatial unit cell from a lattice, using the provided spatial pixel sampling and an optional thicknes C and symmetry

C++ signature : void __init__(_object*,iplt::Lattice,geom::Vec3 [,float [,std::string]])

class iplt.**ReciprocalUnitCell**

GetA (*(_CellImpl)arg1*) → float :
Returns the length of the first unit cell axis

C++ signature : float GetA(iplt::cell_detail::CellImpl {lvalue})

GetB (*(_CellImpl)arg1*) → float :
Returns the length of the second unit cell axis

C++ signature : float GetB(iplt::cell_detail::CellImpl {lvalue})

GetC (*(_CellImpl)arg1*) → float :
Returns the thickness C

C++ signature : float GetC(iplt::cell_detail::CellImpl {lvalue})

GetGamma (*(_CellImpl)arg1*) → float :
Returns the gamma angle

C++ signature : float GetGamma(iplt::cell_detail::CellImpl {lvalue})

GetPosition (*(_CellImpl)arg1*, *(Point)arg2*) → object :

C++ signature : geom::Vec2 GetPosition(iplt::cell_detail::CellImpl {lvalue},ost::img::Point)

GetSymmetry (*(_CellImpl)arg1*) → Symmetry :

C++ signature : iplt::Symmetry GetSymmetry(iplt::cell_detail::CellImpl {lvalue})

GetVecA (*(_CellImpl)self*) → object :
Returns the first unit cell axis

C++ signature : geom::Vec2 GetVecA(iplt::cell_detail::CellImpl {lvalue})

GetVecB (*(_CellImpl)arg1*) → object :
Returns the second unit cell axis

C++ signature : geom::Vec2 GetVecB(iplt::cell_detail::CellImpl {lvalue})

SetC (*(_CellImpl)arg1*, *(float)arg2*) → None :

C++ signature : void SetC(iplt::cell_detail::CellImpl {lvalue},float)

SetSymmetry (*(_CellImpl)arg1*, *(str)arg2*) → None :

C++ signature : void SetSymmetry(iplt::cell_detail::CellImpl {lvalue},std::string)

SetSymmetry((_CellImpl)arg1, (Symmetry)arg2) -> None :

C++ signature : void SetSymmetry(iplt::cell_detail::CellImpl {lvalue},iplt::Symmetry)

__init__ (*(object)arg1*, *(float)arg2*, *(float)A*, *(float)B*[, *(float)gamma*[, *(str)symmetry*]]) → None :

Creates a reciprocal unit cell with the given values for A, B and gamma, plus an optional thickness C and symmetry

C++ signature : void __init__(_object*,float,float,float [,float [,std::string]])

__init__((object)arg1, (SpatialUnitCell)ruc) -> None : Creates a reciprocal unit cell from a spatial one

C++ signature : void __init__(_object*,iplt::SpatialUnitCell)

__init__((object)arg1, (Lattice)arg2, (object)arg3 [, (float)arg4 [, (str)lattice,sampling,thicknes,symmetry]]) -> N
Creates a reciprocal unit cell from a lattice, using the provided reciprocal pixel sampling and an optional thicknes C and symmetry

C++ signature : void __init__(_object*,iplt::Lattice,geom::Vec3 [,float [,std::string]])

Construction

SpatialUnitCell() Creates a default spatial unit cell with A, B and C of length 1.0, and an angle γ of 90 degrees.

SpatialUnitCell(double A, double B, double gamma, double C=1.0) Creates a spatial unit cell with the given values for A, B and γ, plus an optional thickness C.

SpatialUnitCell(ReciprocalUnitCell c) Creates a spatial unit cell from a reciprocal one.

SpatialUnitCell(Lattice lat, Vec3 spatial_sampling, double C=1.0) Creates a spatial unit cell from a *lattice* object, using the provided spatial pixel sampling.

ReciprocalUnitCell() Creates a default reciprocal unit cell with A, B and C of length 1.0, and an angle γ of 90 degrees.

ReciprocalUnitCell(double A, double B, double gamma, double C=1.0) Creates a reciprocal unit cell with the given values for A, B and γ, plus an optional thickness C.

ReciprocalUnitCell(SpatialUnitCell c) Creates a reciprocal unit cell from a spatial one.

ReciprocalUnitCell(Lattice lat, Vec3 reciprocal_sampling, double C=1.0) Creates a spatial unit cell from a *lattice* object, using the provided reciprocal pixel sampling.

Parameter Getter/Setter

double GetA() Returns either the spatial or reciprocal length of unit cell axis A

double GetB() Returns either the spatial or reciprocal length of unit cell axis B

double GetGamma() Returns either the spatial or reciprocal angle γ

3.1.3 Overview: Reflections in IPLT

IPLT contains several dedicated classes and functions for comprehensive handling of reflection data from crystallographic processing. The *ReflectionList* serves as the container for all reflections, from which individual reflections retrieved by means of *ReflectionProxyters*, addressed by *ReflectionIndexes*. These *ReflectionLists* can be imported from and exported to CCP4 style `mtz` files by using the *ReflectionIO* functions.

Example Code

```
import iplt

# read mtz file and store it in a reflection list object
rlist = iplt.ImportMtz("file.mtz")

# iterate over all reflections in the list, returning each
# as a reflection proxyter
for rp in rlist:
  # use the reflection proxyter interface to work with each reflection
  print rp.GetIndex()
```

Components

Reflection List

The `ReflectionList` class encapsulates a list of reflections; each reflection is defined by a unique *reflection index* , and contains an arbitrary amount of (floating point) properties. These properties are defined on the level of the reflection list; as such, a reflection list can be conceptualized as a table, where the rows contain the individual reflections, and the columns the properties.

Access to the entries in the reflection list is only possible by means of a *reflection proxyter*, which is obtained upon iterating over an existing list. The reflections are returned in a strict order, based on their *index*.

A reflection list contains a *lattice* and a *unit cell*. The former is usually in (fractional) pixel dimensions, referring to the image the reflections were originally extracted from. The latter represents the unit cell of the dataset. Neither of these two is mandatory, however, the conversion from reciprocal z values (the third component in the *index*) to fractional `l` stored in the `mtz` file during Export uses the thickness of the *spatial unit cell*!

Reflection lists can be saved to disk as `mtz` files, see the *Reflection IO* entry for more information.

Interface

Construction

`ReflectionList()` Creates an empty reflection list with a default lattice and unit cell.

`ReflectionList(Lattice lat)` Creates an empty reflection list with the given lattice and a default unit cell

`ReflectionList(SpatialUnitCell uc)` Creates an empty reflection list with the given unit cell and a default lattice

`ReflectionList(Lattice lat, SpatialUnitCell uc)` Creates an empty reflection list with the given lattice and unit cell

`ReflectionList(SpatialUnitCell uc,Lattice lat)` see above

`ReflectionList(ReflectionList& rl, bool full_copy=True)` Creates a new reflection list based on the given one; if the (optional) `full_copy` flag is `False`, only the skeleton will be copied, meaning the lattice, the unit cell, and the properties. The complete set of reflections will only be copied if the `full_copy` flag is `True`.

Properties

`CopyProperties(ReflectionList rl)` Clears the list of reflections and all properties; it then creates a new set of properties from the given reflection list.

AddProperty(string prop, PropertyType type=PT_REAL) Adds a new property by the name `prop` to the reflection list. All existing reflections will have a value of zero for this property. The second optional parameters allows a type to be assigned to the property, which mirrors the types as defined for the `mtz` format.

- PT_REAL
- PT_INDEX
- PT_INTENSITY
- PT_AMPLITUDE
- PT_ANOMALOUS_DIFFERENCE
- PT_STDDEV
- PT_AMPLITUDE2
- PT_STDDEVA2
- PT_INTENSITY2
- PT_STDDEVI2
- PT_NORM_AMPLITUDE
- PT_PHASE
- PT_WEIGHT
- PT_PHASE_PROB
- PT_BATCH
- PT_MISYM
- PT_INTEGER

bool HasPropertyType(PropertyType ptype) Returns true if the reflection list has at least one property of the given type

string GetPropertyNameByType(PropertyType ptype) Returns the name of the first property of the given type

bool HasProperty(string prop) Returns true if the reflection list has a property of the given name

RenameProperty(string from, string& to) Renames a property

SetPropertyType(string prop, PropertyType type) Sets the property identified by its name to the given type

PropertyType GetPropertyType(string prop) Returns the type of a property identified by its name

int GetPropertyCount() Returns number of properties in reflection list

int GetPropertyIndex(string name) Returns the internal index of the property identified by its name

string GetPropertyName(int n) Returns the name of the property identified by its internal index

PropertyType GetPropertyType(int n) Returns the type of the property identified by its internal index

stringlist **GetPropertyList()** Returns all properties as a list of strings

Proxyters

Setters/Getters

Reflection Algorithms

Others
class `iplt.ReflectionList`

`AddProperty` (*(ReflectionList)arg1*, *(str)arg2*[, *(PropertyType)arg3*[, *(float)arg4*]]) → int :

C++ signature : unsigned int AddProperty(iplt::ReflectionList {lvalue},std::string [,iplt::PropertyType [,float]])

`AddProxyter` (*(ReflectionList)arg1*, *(ReflectionProxyter)arg2*) → ReflectionProxyter :

C++ signature : iplt::ReflectionProxyter AddProxyter(iplt::ReflectionList {lvalue},iplt::ReflectionProxyter)

`AddReflection` (*(ReflectionList)arg1*, *(ReflectionIndex)arg2*) → ReflectionProxyter :

C++ signature : iplt::ReflectionProxyter AddReflection(iplt::ReflectionList {lvalue},iplt::ReflectionIndex)

AddReflection((ReflectionList)arg1, (ReflectionIndex)arg2, (ReflectionProxyter)arg3) -> ReflectionProxyter :

C++ signature : iplt::ReflectionProxyter AddReflection(iplt::ReflectionList {lvalue},iplt::ReflectionIndex,iplt::ReflectionProxyter)

`Apply` (*(ReflectionList)arg1*, *(ReflectionNonModAlgorithm)arg2*) → None :

C++ signature : void Apply(iplt::ReflectionList {lvalue},iplt::ReflectionNonModAlgorithm {lvalue})

Apply((ReflectionList)arg1, (ReflectionModIPAlgorithm)arg2) -> ReflectionList :

C++ signature : iplt::ReflectionList Apply(iplt::ReflectionList {lvalue},iplt::ReflectionModIPAlgorithm {lvalue})

Apply((ReflectionList)arg1, (ReflectionConstModIPAlgorithm)arg2) -> ReflectionList :

C++ signature : iplt::ReflectionList Apply(iplt::ReflectionList {lvalue},iplt::ReflectionConstModIPAlgorithm)

Apply((ReflectionList)arg1, (ReflectionModOPAlgorithm)arg2) -> ReflectionList :

C++ signature : iplt::ReflectionList Apply(iplt::ReflectionList {lvalue},iplt::ReflectionModOPAlgorithm {lvalue})

Apply((ReflectionList)arg1, (ReflectionConstModOPAlgorithm)arg2) -> ReflectionList :

C++ signature : iplt::ReflectionList Apply(iplt::ReflectionList {lvalue},iplt::ReflectionConstModOPAlgorithm)

`ApplyIP` (*(ReflectionList)arg1*, *(ReflectionNonModAlgorithm)arg2*) → None :

C++ signature : void ApplyIP(iplt::ReflectionList {lvalue},iplt::ReflectionNonModAlgorithm {lvalue})

ApplyIP((ReflectionList)arg1, (ReflectionModIPAlgorithm)arg2) -> None :

C++ signature : void ApplyIP(iplt::ReflectionList {lvalue},iplt::ReflectionModIPAlgorithm {lvalue})

ApplyIP((ReflectionList)arg1, (ReflectionConstModIPAlgorithm)arg2) -> None :

C++ signature : void ApplyIP(iplt::ReflectionList {lvalue},iplt::ReflectionConstModIPAlgorithm)

ApplyIP((ReflectionList)arg1, (ReflectionModOPAlgorithm)arg2) -> None :

C++ signature : void ApplyIP(iplt::ReflectionList {lvalue},iplt::ReflectionModOPAlgorithm {lvalue})

ApplyIP((ReflectionList)arg1, (ReflectionConstModOPAlgorithm)arg2) -> None :

C++ signature : void ApplyIP(iplt::ReflectionList {lvalue},iplt::ReflectionConstModOPAlgorithm)

Begin (*(ReflectionList)arg1*) → ReflectionProxyter :

C++ signature : iplt::ReflectionProxyter Begin(iplt::ReflectionList {lvalue})

ClearReflections (*(ReflectionList)arg1*) → None :

C++ signature : void ClearReflections(iplt::ReflectionList {lvalue})

CopyProperties (*(ReflectionList)arg1*, *(ReflectionList)arg2*) → None :

C++ signature : void CopyProperties(iplt::ReflectionList {lvalue},iplt::ReflectionList)

CopyProxyter (*(ReflectionList)arg1*, *(ReflectionProxyter)arg2*) → ReflectionProxyter :

C++ signature : iplt::ReflectionProxyter CopyProxyter(iplt::ReflectionList {lvalue},iplt::ReflectionProxyter)

DeleteProperty (*(ReflectionList)arg1*, *(str)arg2*) → None :

C++ signature : void DeleteProperty(iplt::ReflectionList {lvalue},std::string)

Dump (*(ReflectionList)arg1*) → str :

C++ signature : std::string Dump(iplt::ReflectionList {lvalue})

DumpFormatted (*(ReflectionList)arg1*) → str :

C++ signature : std::string DumpFormatted(iplt::ReflectionList {lvalue})

DumpHeader (*(ReflectionList)arg1*) → str :

C++ signature : std::string DumpHeader(iplt::ReflectionList {lvalue})

Find (*(ReflectionList)arg1*, *(ReflectionIndex)arg2*) → ReflectionProxyter :

C++ signature : iplt::ReflectionProxyter Find(iplt::ReflectionList {lvalue},iplt::ReflectionIndex)

FindFirst (*(ReflectionList)arg1*, *(Point)arg2*) → ReflectionProxyter :

C++ signature : iplt::ReflectionProxyter FindFirst(iplt::ReflectionList {lvalue},ost::img::Point)

GetLattice (*(ReflectionList)arg1*) → Lattice :

C++ signature : iplt::Lattice GetLattice(iplt::ReflectionList {lvalue})

GetPropertyCount (*(ReflectionList)arg1*) → int :

C++ signature : unsigned int GetPropertyCount(iplt::ReflectionList {lvalue})

GetPropertyIndex (*(ReflectionList)arg1*, *(str)arg2*) → int :

C++ signature : unsigned int GetPropertyIndex(iplt::ReflectionList {lvalue},std::string)

GetPropertyName (*(ReflectionList)arg1*, *(int)arg2*) → str :

C++ signature : std::string GetPropertyName(iplt::ReflectionList {lvalue},int)

GetPropertyNameByType (*(ReflectionList)arg1*, *(PropertyType)arg2*) → str :

C++ signature : std::string GetPropertyNameByType(iplt::ReflectionList {lvalue},iplt::PropertyType)

GetPropertyType (*(ReflectionList)arg1*, *(str)arg2*) → PropertyType :

C++ signature : iplt::PropertyType GetPropertyType(iplt::ReflectionList {lvalue},std::string)

GetPropertyType((ReflectionList)arg1, (int)arg2) -> PropertyType :

C++ signature : iplt::PropertyType GetPropertyType(iplt::ReflectionList {lvalue},int)

GetResolution (*(ReflectionList)arg1*, *(ReflectionIndex)arg2*) → float :

C++ signature : float GetResolution(iplt::ReflectionList {lvalue},iplt::ReflectionIndex)

GetSymmetry (*(ReflectionList)arg1*) → Symmetry :

C++ signature : iplt::Symmetry GetSymmetry(iplt::ReflectionList {lvalue})

GetUnitCell (*(ReflectionList)arg1*) → SpatialUnitCell :

C++ signature : iplt::SpatialUnitCell GetUnitCell(iplt::ReflectionList {lvalue})

HasProperty (*(ReflectionList)arg1*, *(str)arg2*) → bool :

C++ signature : bool HasProperty(iplt::ReflectionList {lvalue},std::string)

HasPropertyType (*(ReflectionList)arg1*, *(PropertyType)arg2*) → bool :

C++ signature : bool HasPropertyType(iplt::ReflectionList {lvalue},iplt::PropertyType)

Merge (*(ReflectionList)arg1*, *(ReflectionList)arg2*) → None :

C++ signature : void Merge(iplt::ReflectionList {lvalue},iplt::ReflectionList)

NumReflections (*(ReflectionList)arg1*) → int :

C++ signature : unsigned int NumReflections(iplt::ReflectionList {lvalue})

RenameProperty (*(ReflectionList)arg1*, *(str)arg2*, *(str)arg3*) → None :

C++ signature : void RenameProperty(iplt::ReflectionList {lvalue},std::string,std::string)

SetLattice (*(ReflectionList)arg1*, *(Lattice)arg2*) → None :

C++ signature : void SetLattice(iplt::ReflectionList {lvalue},iplt::Lattice)

SetPropertyType (*(ReflectionList)arg1*, *(str)arg2*, *(PropertyType)arg3*) → None :

C++ signature : void SetPropertyType(iplt::ReflectionList {lvalue},std::string,iplt::PropertyType)

SetSymmetry (*(ReflectionList)arg1*, *(str)arg2*) → None :

C++ signature : void SetSymmetry(iplt::ReflectionList {lvalue},std::string)

SetSymmetry((ReflectionList)arg1, (Symmetry)arg2) -> None :

C++ signature : void SetSymmetry(iplt::ReflectionList {lvalue},iplt::Symmetry)

SetUnitCell (*(ReflectionList)arg1*, *(SpatialUnitCell)arg2*) → None :

C++ signature : void SetUnitCell(iplt::ReflectionList {lvalue},iplt::SpatialUnitCell)

__init__ (*(object)arg1*) → None :

C++ signature : void __init__(_object*)

__init__((object)arg1, (ReflectionList)arg2 [, (bool)arg3]) -> None :

C++ signature : void __init__(_object*,iplt::ReflectionList [,bool])

__init__((object)arg1, (Lattice)arg2) -> None :

C++ signature : void __init__(_object*,iplt::Lattice)

__init__((object)arg1, (SpatialUnitCell)arg2) -> None :

C++ signature : void __init__(_object*,iplt::SpatialUnitCell)

__init__((object)arg1, (Lattice)arg2, (SpatialUnitCell)arg3) -> None :

C++ signature : void __init__(_object*,iplt::Lattice,iplt::SpatialUnitCell)

__init__((object)arg1, (SpatialUnitCell)arg2, (Lattice)arg3) -> None :

C++ signature : void __init__(_object*,iplt::SpatialUnitCell,iplt::Lattice)

Examples This code snippet creates the following reflection list:

index	amp	sigamp
(1,3,0.132)	23.45	4.54
(4,7,0.237)	17.33	6.13

```
import ex
# create new list and add two properties
rlist=ex.ReflectionList()
rlist.AddProperty("amp")
rlist.AddProperty("sigamp")

# add reflections by means of reflection proxyters
rp=rlist.AddReflection(ex.ReflectionList(2,3,0.132))
rp.Set("amp",23.45)
rp.Set("sigamp",4.54)
rp=rlist.AddReflection(ex.ReflectionList(4,7,0.237))
rp.Set("amp",17.33)
rp.Set("sigamp",6.13)

# iterate over list, again accessing the reflections via the proxyters
for rp in rlist:
  print rp.GetIndex(),rp.Get("amp"),rp.Get("sigamp")
```

Reflection List Import and Export

The structure of the *reflection list* was inspired from the CCP4[1] style `mtz` format. For this reason, a reflection list can be exported to an `mtz` file, fully preserving its contents, and re-imported at a later stage. However, the import functionality is not meant to handle the complex and diverse capabilities of the `mtz` format, and cannot replace the CCP4 utilities to handle generic `mtz` files.

In addition, in order to support data interchange with MRC, an `aph` importer and exporter are available.

Interface

`ReflectionList ImportMtz(string file)` Import a *reflection list* from an `mtz` file

`ExportMtz(ReflectionList rl, string file)` Export a *reflection list* to an `mtz` file

`ReflectionList ImportAph(const string& file)` Import a *reflection list* from an `aph` file

`ExportAph(ReflectionList rl, string file, string title="", string order="")`
Export a *reflection list* to an `aph` file

`iplt.`**`ImportMtz`**`(`*(str)arg1*`)` → ReflectionList :

C++ signature : iplt::ReflectionList ImportMtz(std::string)

`iplt.`**`ExportMtz`**`(`*(ReflectionList)arg1*, *(str)arg2*`)` → None :

C++ signature : void ExportMtz(iplt::ReflectionList,std::string)

`iplt.`**`ImportAph`**`(`*(str)arg1*`)` → ReflectionList :

C++ signature : iplt::ReflectionList ImportAph(std::string)

`iplt.`**`ExportAph`**`(`*(ReflectionList)arg1*, *(str)arg2*[, *(str)arg3*[, *(str)arg4*]]`)` → None :

C++ signature : void ExportAph(iplt::ReflectionList,std::string [,std::string [,std::string]])

[1] http://www.ccp4.ac.uk/

Reflection Index

The `iplt::ReflectionIndex` class represents the special nature of the 2D crystallography reflection indices, defined by integer *h* and *k* and floating point reciprocal *z*.

The usage of the reflection index is explained in the entry for the *reflection list*.

class `iplt.`**`ReflectionIndex`**

`AsDuplet`(*(ReflectionIndex)arg1*) → Point :

C++ signature : ost::img::Point AsDuplet(iplt::ReflectionIndex {lvalue})

`AsTriplet`(*(ReflectionIndex)arg1*, *(float)arg2*) → Point :

C++ signature : ost::img::Point AsTriplet(iplt::ReflectionIndex {lvalue},float)

`GetH`(*(ReflectionIndex)arg1*) → int :

C++ signature : int GetH(iplt::ReflectionIndex {lvalue})

`GetK`(*(ReflectionIndex)arg1*) → int :

C++ signature : int GetK(iplt::ReflectionIndex {lvalue})

`GetL`(*(ReflectionIndex)arg1*, *(float)arg2*) → float :

C++ signature : float GetL(iplt::ReflectionIndex {lvalue},float)

`GetZStar`(*(ReflectionIndex)arg1*) → float :

C++ signature : float GetZStar(iplt::ReflectionIndex {lvalue})

`IsEqual`(*(ReflectionIndex)arg1*, *(ReflectionIndex)arg2*, *(float)arg3*) → bool :

C++ signature : bool IsEqual(iplt::ReflectionIndex {lvalue},iplt::ReflectionIndex,float)

`ToVec3`(*(ReflectionIndex)arg1*) → object :

C++ signature : geom::Vec3 ToVec3(iplt::ReflectionIndex {lvalue})

`__init__`(*(object)arg1*, *(int)arg2*, *(int)arg3*[, *(float)arg4*]) → None :

C++ signature : void __init__(_object*,int,int [,float])

__init__((object)arg1, (ReflectionIndex)arg2, (float)arg3) -> None :

C++ signature : void __init__(_object*,iplt::ReflectionIndex,float)

__init__((object)arg1, (Point)arg2) -> None :

C++ signature : void __init__(_object*,ost::img::Point)

__init__((object)arg1, (float)arg2, (Point)arg3) -> None :

C++ signature : void __init__(_object*,float,ost::img::Point)

__init__((object)arg1, (object)arg2) -> None :

C++ signature : void __init__(_object*,geom::Vec3)

Reflection Proxyter

The `ReflectionProxyter` class is used to access entries in a *reflection list*, and is provided automatically upon iteration over such a list. See the *reflection list* entry for more detail.

class `iplt.`**`ReflectionProxyter`**

`AtEnd`(*(ReflectionProxyter)arg1*) → bool :

C++ signature : bool AtEnd(iplt::ReflectionProxyter {lvalue})

Delete (*(ReflectionProxyter)arg1*) → None :

C++ signature : void Delete(iplt::ReflectionProxyter {lvalue})

Get (*(ReflectionProxyter)arg1*, *(str)arg2*) → float :

C++ signature : float Get(iplt::ReflectionProxyter {lvalue},std::string)

Get((ReflectionProxyter)arg1, (int)arg2) -> float :

C++ signature : float Get(iplt::ReflectionProxyter {lvalue},unsigned int)

GetIndex (*(ReflectionProxyter)arg1*) → ReflectionIndex :

C++ signature : iplt::ReflectionIndex GetIndex(iplt::ReflectionProxyter {lvalue})

GetResolution (*(ReflectionProxyter)arg1*) → float :

C++ signature : float GetResolution(iplt::ReflectionProxyter {lvalue})

Inc (*(ReflectionProxyter)arg1*) → None :

C++ signature : void Inc(iplt::ReflectionProxyter {lvalue})

Index (*(ReflectionProxyter)arg1*) → ReflectionIndex :

C++ signature : iplt::ReflectionIndex Index(iplt::ReflectionProxyter {lvalue})

IsValid (*(ReflectionProxyter)arg1*) → bool :

C++ signature : bool IsValid(iplt::ReflectionProxyter {lvalue})

Set (*(ReflectionProxyter)arg1*, *(str)arg2*, *(float)arg3*) → None :

C++ signature : void Set(iplt::ReflectionProxyter {lvalue},std::string,float)

Set((ReflectionProxyter)arg1, (int)arg2, (float)arg3) -> None :

C++ signature : void Set(iplt::ReflectionProxyter {lvalue},unsigned int,float)

__init__ (*(object)arg1*, *(ReflectionProxyter)arg2*) → None :

C++ signature : void __init__(_object*,iplt::ReflectionProxyter)

3.1.4 Tilt geometry

class iplt.**TiltGeometry**

ApplyReciprocal (*(TiltGeometry)arg1*, *(Lattice)arg2*) → Lattice :

C++ signature : iplt::Lattice ApplyReciprocal(iplt::TiltGeometry {lvalue},iplt::Lattice)

ApplyReciprocal((TiltGeometry)arg1, (object)arg2) -> object :

C++ signature : geom::Vec2 ApplyReciprocal(iplt::TiltGeometry {lvalue},geom::Vec2)

ApplySpatial (*(TiltGeometry)arg1*, *(Lattice)arg2*) → Lattice :

C++ signature : iplt::Lattice ApplySpatial(iplt::TiltGeometry {lvalue},iplt::Lattice)

ApplySpatial((TiltGeometry)arg1, (object)arg2) -> object :

C++ signature : geom::Vec2 ApplySpatial(iplt::TiltGeometry {lvalue},geom::Vec2)

GetReciprocalTransformation (*(TiltGeometry)arg1*) → object :

C++ signature : geom::Mat3 GetReciprocalTransformation(iplt::TiltGeometry {lvalue})

GetSpatialTransformation (*(TiltGeometry)arg1*) → object :

C++ signature : geom::Mat3 GetSpatialTransformation(iplt::TiltGeometry {lvalue})

GetTiltAngle (*(TiltGeometry)arg1*) → float :

C++ signature : float GetTiltAngle(iplt::TiltGeometry {lvalue})

GetXAxisAngle (*(TiltGeometry)arg1*) → float :

C++ signature : float GetXAxisAngle(iplt::TiltGeometry {lvalue})

SetTiltAngle (*(TiltGeometry)arg1*, *(float)arg2*) → None :

C++ signature : void SetTiltAngle(iplt::TiltGeometry {lvalue},float)

SetXAxisAngle (*(TiltGeometry)arg1*, *(float)arg2*) → None :

C++ signature : void SetXAxisAngle(iplt::TiltGeometry {lvalue},float)

__init__ (*(object)arg1*) → None :

C++ signature : void __init__(_object*)

__init__((object)arg1, (float)arg2, (float)arg3) -> None :

C++ signature : void __init__(_object*,float,float)

__init__((object)arg1, (TiltGeometry)arg2) -> None :

C++ signature : void __init__(_object*,iplt::TiltGeometry)

3.1.5 Symmetry

class iplt.**Symmetry**

GenerateSymmetryRelatedPoints (*(Symmetry)arg1*, *(ReflectionIndex)arg2*[, *(bool)arg3*]) → ReflectionIndexVec :

C++ signature : std::vector<iplt::ReflectionIndex, std::allocator<iplt::ReflectionIndex> > GenerateSymmetryRelatedPoints(iplt::Symmetry {lvalue},iplt::ReflectionIndex [,bool])

GenerateSymmetryRelatedPositions (*(Symmetry)arg1*, *(object)arg2*) → object :

C++ signature : std::vector<geom::Vec3, std::allocator<geom::Vec3> > GenerateSymmetryRelatedPositions(iplt::Symmetry {lvalue},geom::Vec3)

GetPointgroupName (*(Symmetry)arg1*) → str :

C++ signature : std::string GetPointgroupName(iplt::Symmetry {lvalue})

GetSpacegroupName (*(Symmetry)arg1*) → str :

C++ signature : std::string GetSpacegroupName(iplt::Symmetry {lvalue})

GetSpacegroupNumber (*(Symmetry)arg1*) → int :

C++ signature : unsigned int GetSpacegroupNumber(iplt::Symmetry {lvalue})

IsCentric (*(Symmetry)arg1*, *(ReflectionIndex)arg2*) → bool :

C++ signature : bool IsCentric(iplt::Symmetry {lvalue},iplt::ReflectionIndex)

IsCentric((Symmetry)arg1, (object)arg2) -> bool :

C++ signature : bool IsCentric(iplt::Symmetry {lvalue},iplt::SymmetryReflection)

IsInAsymmetricUnit (*(Symmetry)arg1*, *(ReflectionIndex)arg2*) → bool :

C++ signature : bool IsInAsymmetricUnit(iplt::Symmetry {lvalue},iplt::ReflectionIndex)

IsInAsymmetricUnit((Symmetry)arg1, (object)arg2) -> bool :

C++ signature : bool IsInAsymmetricUnit(iplt::Symmetry {lvalue},iplt::SymmetryReflection)

Symmetrize (*(Symmetry)arg1*, *(ReflectionIndex)arg2*) → ReflectionIndex :

C++ signature : iplt::ReflectionIndex Symmetrize(iplt::Symmetry {lvalue},iplt::ReflectionIndex)

Symmetrize((Symmetry)arg1, (ReflectionIndex)arg2, (float)arg3) -> float :

C++ signature : float Symmetrize(iplt::Symmetry {lvalue},iplt::ReflectionIndex,float)

__init__ (*(object)arg1*, *(int)arg2*) → None :

C++ signature : void __init__(_object*,unsigned int)

__init__((object)arg1, (str)arg2) -> None :

C++ signature : void __init__(_object*,std::string)

3.1.6 Defocus

class iplt.**Defocus**

GetAstigmatismAngle (*(Defocus)arg1*) → float :

C++ signature : float GetAstigmatismAngle(iplt::Defocus {lvalue})

GetDefocusX (*(Defocus)arg1*) → float :

C++ signature : float GetDefocusX(iplt::Defocus {lvalue})

GetDefocusY (*(Defocus)arg1*) → float :

C++ signature : float GetDefocusY(iplt::Defocus {lvalue})

GetMeanDefocus (*(Defocus)arg1*) → float :

C++ signature : float GetMeanDefocus(iplt::Defocus {lvalue})

SetAstigmatismAngle (*(Defocus)arg1*, *(float)arg2*) → None :

C++ signature : void SetAstigmatismAngle(iplt::Defocus {lvalue},float)

SetDefocusX (*(Defocus)arg1*, *(float)arg2*) → None :

C++ signature : void SetDefocusX(iplt::Defocus {lvalue},float)

SetDefocusY (*(Defocus)arg1*, *(float)arg2*) → None :

C++ signature : void SetDefocusY(iplt::Defocus {lvalue},float)

SetMeanDefocus (*(Defocus)arg1*, *(float)arg2*) → None :

C++ signature : void SetMeanDefocus(iplt::Defocus {lvalue},float)

__init__ (*(object)arg1*[, *(float)arg2*[, *(float)arg3*[, *(float)arg4*]]]) → None :

C++ signature : void __init__(_object* [,float [,float [,float]]])

3.1.7 Microscope data

class iplt.**MicroscopeData**

GetAccelerationVoltage (*(MicroscopeData)arg1*) → float :

C++ signature : float GetAccelerationVoltage(iplt::MicroscopeData {lvalue})

GetAmplitudeContrast (*(MicroscopeData)arg1*) → float :

C++ signature : float GetAmplitudeContrast(iplt::MicroscopeData {lvalue})

GetChromaticAberration (*(MicroscopeData)arg1*) → float :

C++ signature : float GetChromaticAberration(iplt::MicroscopeData {lvalue})

GetConvergenceAngle (*(MicroscopeData)arg1*) → float :

C++ signature : float GetConvergenceAngle(iplt::MicroscopeData {lvalue})

GetEnergySpread (*(MicroscopeData)arg1*) → float :

C++ signature : float GetEnergySpread(iplt::MicroscopeData {lvalue})

GetSphericalAberration (*(MicroscopeData)arg1*) → float :

C++ signature : float GetSphericalAberration(iplt::MicroscopeData {lvalue})

GetWavelength (*(MicroscopeData)arg1*) → float :

C++ signature : float GetWavelength(iplt::MicroscopeData {lvalue})

SetAccelerationVoltage (*(MicroscopeData)arg1*, *(float)arg2*) → None :

C++ signature : void SetAccelerationVoltage(iplt::MicroscopeData {lvalue},float)

SetAmplitudeContrast (*(MicroscopeData)arg1*, *(float)arg2*) → None :

C++ signature : void SetAmplitudeContrast(iplt::MicroscopeData {lvalue},float)

SetChromaticAberration (*(MicroscopeData)arg1*, *(float)arg2*) → None :

C++ signature : void SetChromaticAberration(iplt::MicroscopeData {lvalue},float)

SetConvergenceAngle (*(MicroscopeData)arg1*, *(float)arg2*) → None :

C++ signature : void SetConvergenceAngle(iplt::MicroscopeData {lvalue},float)

SetEnergySpread (*(MicroscopeData)arg1*, *(float)arg2*) → None :

C++ signature : void SetEnergySpread(iplt::MicroscopeData {lvalue},float)

SetSphericalAberration (*(MicroscopeData)arg1*, *(float)arg2*) → None :

C++ signature : void SetSphericalAberration(iplt::MicroscopeData {lvalue},float)

__init__ (*(object)arg1*[, *(float)arg2*[, *(float)arg3*[, *(float)arg4*[, *(float)arg5*[, *(float)arg6*[, *(float)arg7*]]]]]]) → None :

C++ signature : void __init__(_object* [,float [,float [,float [,float [,float [,float]]]]]])

3.1.8 TCIF data

class iplt.**TCIFData**

SetDefocus (*(TCIFData)arg1*, *(Defocus)arg2*) → None :

C++ signature : void SetDefocus(iplt::TCIFData {lvalue},iplt::Defocus)

SetMicroscopeData (*(TCIFData)arg1*, *(MicroscopeData)arg2*) → None :

C++ signature : void SetMicroscopeData(iplt::TCIFData {lvalue},iplt::MicroscopeData)

SetTiltGeometry (*(TCIFData)arg1*, *(TiltGeometry)arg2*) → None :

C++ signature : void SetTiltGeometry(iplt::TCIFData {lvalue},iplt::TiltGeometry)

__init__ (*(object)arg1*[, *(MicroscopeData)arg2*[, *(Defocus)arg3*[, *(TiltGeometry)arg4*]]]) → None :

C++ signature : void __init__(_object* [,iplt::MicroscopeData [,iplt::Defocus [,iplt::TiltGeometry]]])

__init__((object)arg1, (TCIFData)arg2) -> None :

C++ signature : void __init__(_object*,iplt::TCIFData)

3.1.9 Lattice Line Data

class iplt.**LatticeLineData**

Get (*(LatticeLineData)arg1*, *(ReflectionIndex)arg2*) → complex :

C++ signature : std::complex<float> Get(iplt::LatticeLineData {lvalue},iplt::ReflectionIndex)

GetAmplitude (*(LatticeLineData)arg1*, *(ReflectionIndex)arg2*) → float :

C++ signature : float GetAmplitude(iplt::LatticeLineData {lvalue},iplt::ReflectionIndex)

GetIndexList (*(LatticeLineData)arg1*) → PointList :

C++ signature : ost::img::PointList GetIndexList(iplt::LatticeLineData {lvalue})

GetPhase (*(LatticeLineData)arg1*, *(ReflectionIndex)arg2*) → float :

C++ signature : float GetPhase(iplt::LatticeLineData {lvalue},iplt::ReflectionIndex)

__init__ (*(object)arg1*, *(ReflectionList)arg2*[, *(bool)arg3*[, *(str)arg4*[, *(str)arg5*]]]) → None :

C++ signature : void __init__(_object*,iplt::ReflectionList [,bool [,std::string [,std::string]]])

3.2 iplt.alg module: Image and reflection list algorithms

3.2.1 Image algorithms

Canny

Description

Performs a Canny edge detection. Two parameters for upper and lower threshold have to be given to the constructor (normalized between 0 and 1).

Examples

C++

```
#include <ost/io/load_map.hh>
#include <iplt/alg/canny.hh>


int main()
{
  ImageHandle img = ost::io::LoadImage("lena.tif");
  img.ApplyIP( iplt::alg::Canny(0.4,0.1) );

  return 0;
}
```

Python

```
from ost.io import *
from iplt import *
img = LoadImage("lena.tif")
img.ApplyIP( alg.Canny(0.4,0.1) )
```

Figure 3.1: left: original image, right image after edge detection

Example image

Python interface

class `iplt.alg.`**`Canny`**

`__init__`(*(object)arg1*, *(float)arg2*, *(float)arg3*) → None :

C++ signature : void __init__(_object*,float,float)

C++ interface

class **iplt::alg::Canny**

Public Functions

Canny(Real upperthreshold, Real lowerthreshold)

ost::img::ImageHandle **Visit**(const ost::img::ConstImageHandle & i)

Close

Description

Calculates the binary morphological operation close on the image. The constructor expects a structuring element in form of an image. The structuring element can either be created manually or one of the *default elements* can be taken.

Examples

C++

```
#include <iplt/image.hh>
#include <iplt/alg/close.hh>

using namespace iplt;

int main()
{
  ImageHandle img = LoadImage("...");
  img.ApplyIP( alg::Close(alg::morph::Create8n()) );

  return 0;
}
```

Python

```
from iplt import *
img = LoadImage("...")
img.ApplyIP( alg.Close(alg.morph.Create8n()) )
```

Example image

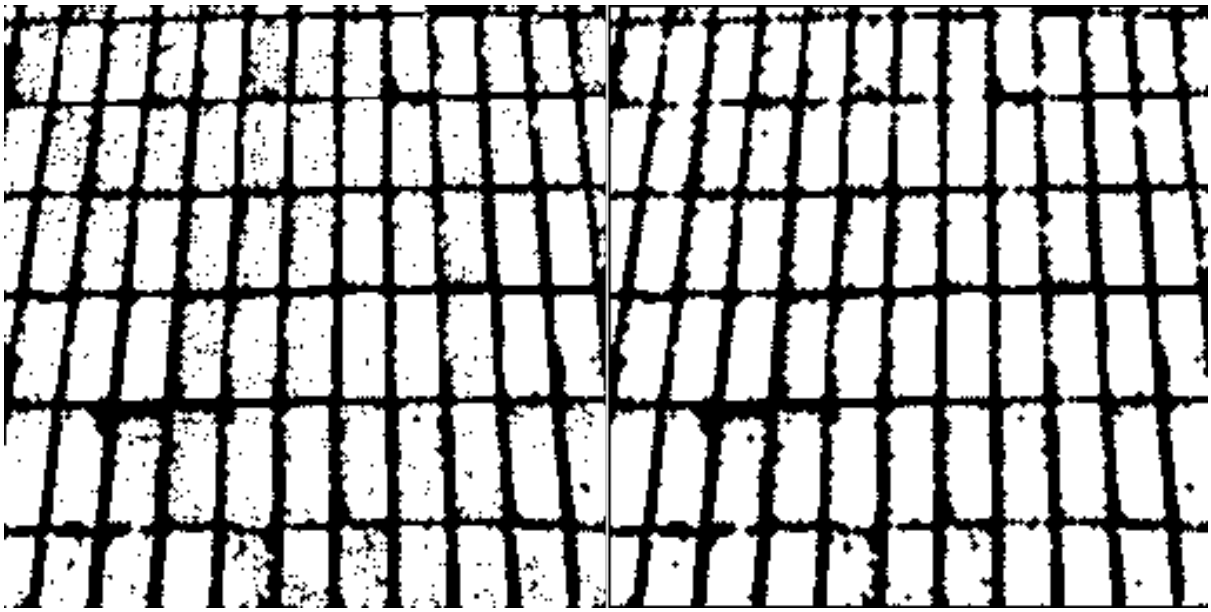

Figure 3.2: left: original image, right image after close operation

Python interface

class iplt.alg.**Close**

__init__(*(object)arg1*, *(ConstImageHandle)arg2*) → None :

C++ signature : void __init__(_object*,ost::img::ConstImageHandle)

C++ interface

class **iplt::alg::Close**

Public Functions

Close(const ost::img::ConstImageHandle & structure_element)

ost::img::ImageHandle **Visit**(const ost::img::ConstImageHandle & i)

Connectivity Counter

Python interface

class `iplt.alg.`**`ConnectivityCounter`**

`__init__`(*(object)arg1*) → None :

C++ signature : void __init__(_object*)

C++ interface

class **iplt::alg::ConnectivityCounter**

Public Functions

ConnectivityCounter(void)

ost::img::ImageHandle **Visit**(const ost::img::ConstImageHandle & i)

Contrast transfer function (CTF)

Python interface

class `iplt.alg.`**`CTF`**

`__init__`(*(object)arg1*[, *(TCIFData)arg2*]) → None :

C++ signature : void __init__(_object* [,iplt::TCIFData])

C++ interface

class **iplt::alg::CTF**

Public Functions

CTF(const TCIFData & data = TCIFData ())

ost::img::ImageHandle **Visit**(const ost::img::ConstImageHandle & i)

Contrast transfer function correction

Python interface

class `iplt.alg.`**`CTFCorrection`**

`GetSignalToNoiseRatio`(*(CTFCorrection)arg1*) → float :

C++ signature : float GetSignalToNoiseRatio(iplt::alg::CTFCorrection {lvalue})

`SetSignalToNoiseRatio`(*(CTFCorrection)arg1*, *(float)arg2*) → None :

C++ signature : void SetSignalToNoiseRatio(iplt::alg::CTFCorrection {lvalue},float)

`__init__`(*(object)arg1*[, *(TCIFData)arg2*[, *(float)arg3*]]) → None :

C++ signature : void __init__(_object* [,iplt::TCIFData [,float]])

C++ interface

class **iplt::alg::CTFCorrection**

Public Functions

CTFCorrection(const TCIFData & data = TCIFData (), Real snr = 10)

ost::img::ImageHandle **Visit**(const ost::img::ConstImageHandle & i)

void **SetSignalToNoiseRatio**(Real snr)

Real **GetSignalToNoiseRatio**()

Dilate

Description

Does a binary dilation on the image. The constructor expects a structuring element in form of an image handle. The structuring element can either be created manually or one of the *default elements* can be taken.

Examples

C++

```
#include <iplt/image.hh>
#include <iplt/alg/dilate.hh>

using namespace iplt;

int main()
{
  ImageHandle img = LoadImage("...");
  img.ApplyIP( alg::Dilate(alg::Create4n()) );

  return 0;
}
```

Python

```
from iplt import *
img = LoadImage("...")
img.ApplyIP( alg.Dilate(alg.Create4n()) )
```

Example image

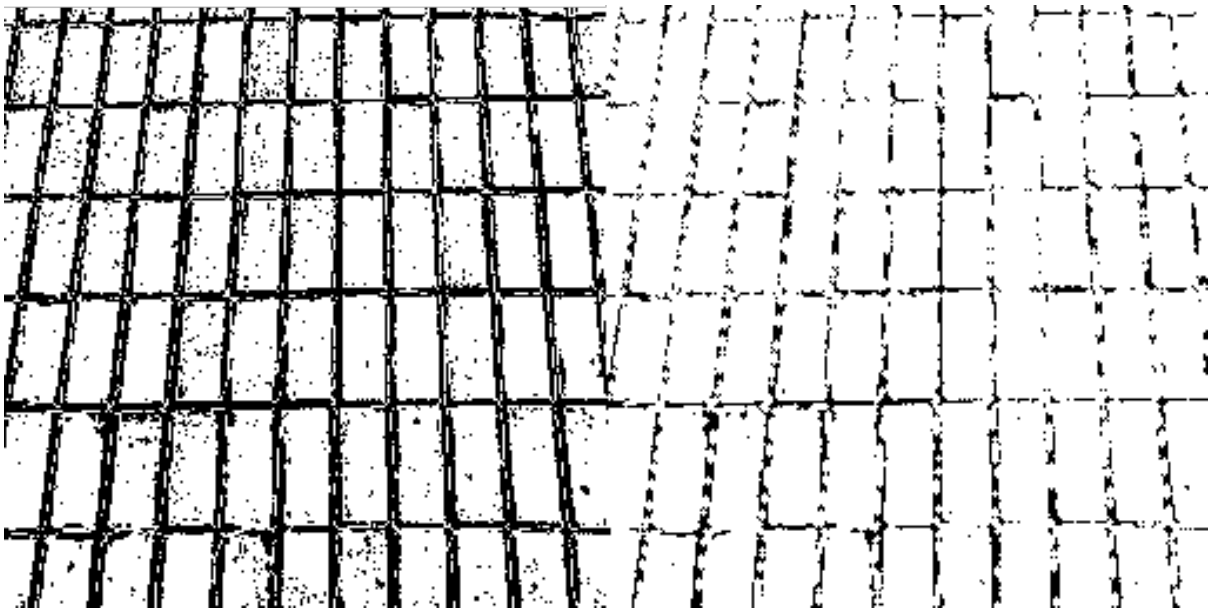

Figure 3.3: left: original image, right image after dilation

Python interface

class iplt.alg.**Dilate**

__init__ (*(object)arg1*, *(ConstImageHandle)arg2*) → None :

C++ signature : void __init__(_object*,ost::img::ConstImageHandle)

C++ interface

class **iplt::alg::Dilate**

Public Functions

Dilate(const ost::img::ConstImageHandle & structure_element)

ost::img::ImageHandle **Visit**(const ost::img::ConstImageHandle & i)

Erode

Description

Does a binary erosion on the image. The constructor expects a structuring element in form of an image handle. The structuring element can either be created manually or one of the *default elements* can be taken.

Examples

C++

```
#include <iplt/image.hh>
#include <iplt/alg/erode.hh>

using namespace iplt;

int main()
{
  ImageHandle img = LoadImage("...");
  img.ApplyIP( alg::Erode(alg::Create4n()) );

  return 0;
}
```

Python

```
from iplt import *
img = LoadImage("...")
img.ApplyIP( alg.Erode(alg.Create4n()) )
```

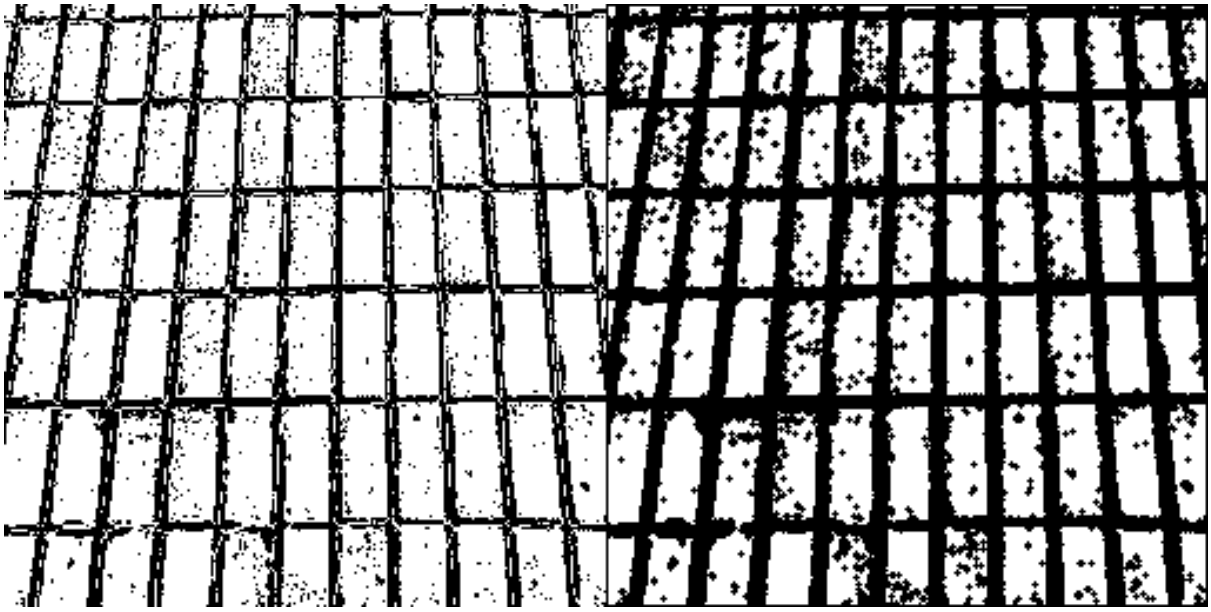

Figure 3.4: left: original image, right image after erosion

Example image

Python interface

class `iplt.alg.`**`Erode`**

`__init__`(*(object)arg1*, *(ConstImageHandle)arg2*) → None :

C++ signature : void __init__(_object*,ost::img::ConstImageHandle)

C++ interface

class **iplt::alg::Erode**

Public Functions

Erode(const ost::img::ConstImageHandle & structure_element)

ost::img::ImageHandle **Visit**(const ost::img::ConstImageHandle & i)

Fit background

Python interface

class `iplt.alg.`**`FitBackground`**

`Add`(*(FitBackground)arg1*, *(float)arg2*, *(float)arg3*, *(float)arg4*[, *(float)arg5*]) → None :

C++ signature : void Add(iplt::alg::FitBackground {lvalue},float,float,float [,float])

Apply(*(FitBackground)arg1*) → None :

C++ signature : void Apply(iplt::alg::FitBackground {lvalue})

AsImage(*(FitBackground)arg1*, *(Extent)arg2*) → ImageHandle :

C++ signature : ost::img::ImageHandle AsImage(iplt::alg::FitBackground {lvalue},ost::img::Extent)

GetB1(*(FitBackground)arg1*) → float :

C++ signature : float GetB1(iplt::alg::FitBackground {lvalue})

GetB2(*(FitBackground)arg1*) → float :

C++ signature : float GetB2(iplt::alg::FitBackground {lvalue})

GetC(*(FitBackground)arg1*) → float :

C++ signature : float GetC(iplt::alg::FitBackground {lvalue})

GetChi(*(FitBackground)arg1*) → float :

C++ signature : float GetChi(iplt::alg::FitBackground {lvalue})

GetOrigin(*(FitBackground)arg1*) → object :

C++ signature : geom::Vec2 GetOrigin(iplt::alg::FitBackground {lvalue})

GetS1(*(FitBackground)arg1*) → float :

C++ signature : float GetS1(iplt::alg::FitBackground {lvalue})

GetS2(*(FitBackground)arg1*) → float :

C++ signature : float GetS2(iplt::alg::FitBackground {lvalue})

__init__(*(object)arg1*[, *(bool)arg2*]) → None :

C++ signature : void __init__(_object* [,bool])

C++ interface

class **iplt::alg::FitBackground**

Public Functions

FitBackground(bool zo = false)

~FitBackground()

void **Add**(Real x, Real y, Real val, Real w = 1.0)

Real **GetS1**()

Real **GetB1**()

Real **GetS2**()

Real **GetB2**()

geom::Vec2 **GetOrigin**()

Real **GetC**()

Real **GetChi**()

void **Apply**()

ost::img::ImageHandle **AsImage**(const ost::img::Extent & e)

2D gaussian fit

Python interface

class iplt.alg.**FitGauss2D**

GetAbsQuality(*(FitGauss2D)arg1*) → float :

C++ signature : double GetAbsQuality(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::FitGauss2DBa
{lvalue})

GetChi(*(FitGauss2D)arg1*) → float :

C++ signature : double GetChi(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::FitGauss2DBase> {lvalue})

GetGOF (*(FitGauss2D)arg1*) → float :

C++ signature : double GetGOF(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::FitGauss2DBase> {lvalue})

GetIterationCount (*(FitGauss2D)arg1*) → int :

C++ signature : int GetIterationCount(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::FitGauss2DBas {lvalue})

GetMaxIter (*(FitGauss2D)arg1*) → int :

C++ signature : unsigned int GetMaxIter(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::FitGauss2DI {lvalue})

GetQuality (*(FitGauss2D)arg1*) → float :

C++ signature : double GetQuality(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::FitGauss2DBase> {lvalue})

GetRelQuality (*(FitGauss2D)arg1*) → float :

C++ signature : double GetRelQuality(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::FitGauss2DBa {lvalue})

GetSigAmp (*(FitGauss2D)arg1*) → float :

C++ signature : double GetSigAmp(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::FitGauss2DBase> {lvalue})

SetLim (*(FitGauss2D)arg1*, *(float)arg2*, *(float)arg3*) → None :

C++ signature : void SetLim(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::FitGauss2DBase> {lvalue},double,double)

SetMaxIter (*(FitGauss2D)arg1*, *(int)arg2*) → None :

C++ signature : void SetMaxIter(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::FitGauss2DBase> {lvalue},int)

SetSigma (*(FitGauss2D)arg1*, *(float)arg2*) → None :

C++ signature : void SetSigma(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::FitGauss2DBase> {lvalue},double)

__init__ (*(object)arg1*[, *(float)arg2*[, *(float)arg3*[, *(float)arg4*[, *(float)arg5*[, *(float)arg6*[, *(float)arg7*[, *(float)arg8*[, *(float)arg9*[, *(float)arg10*]]]]]]]]]) → None :

C++ signature : void __init__(_object* [,float [,float [,float [,float [,float [,float [,float [,float [,float]]]]]]]]])

C++ interface

typedef ost::img::ImageStateNonModAlgorithm< *FitGauss2DBase* > **FitGauss2D**

class **iplt::alg::FitGauss2DBase**

2D Gaussian fitting algorithm

Public Type

Anonymous enum

Values:

- ID_A = =0 -
- ID_BX -
- ID_BY -
- ID_UX -
- ID_UY -
- ID_C -
- ID_W -
- ID_PX -
- ID_PY -

Public Functions

FitGauss2DBase(double A = 1.0, double Bx = 1.0, double By = 1.0, double C = 0.0, double ux = 0.0, double uy = 0.0, double w = 0.0, double Px = 0.0, double Py = 0.0)

void **SetSigma**(double s)

void **SetMaxIter**(int n)

unsigned int **GetMaxIter**()

void **SetLim**(double l1, double l2)

double **GetChi**()

void **SetChi**(double c)

double **GetGOF**()

void **SetGOF**(double g)

double **GetQuality**()

void **SetQuality**(double q)

double **GetAbsQuality**()

void **SetAbsQuality**(double q)

double **GetRelQuality**()

void **SetRelQuality**(double q)

int **GetIterationCount**()

double **GetSigAmp**()

void **SetSigAmp**(double sa)

template < typename T, class D >
void **VisitState**(const ost::img::ImageStateImpl< T, D > & isi)

Public Static Functons

String **GetAlgorithmName**()

Radial background fit

Python interface

class `iplt.alg.`**`FitRadialBackground`**

`GetMaxIter`(*(FitRadialBackground)arg1*) → int :

C++ signature : unsigned int GetMaxIter(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::FitRadialBa {lvalue})

`GetOrigin`(*(FitRadialBackground)arg1*) → object :

C++ signature : geom::Vec2 GetOrigin(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::FitRadialBack {lvalue})

`GetTolerance`(*(FitRadialBackground)arg1*) → float :

C++ signature : float GetTolerance(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::FitRadialBackgrou {lvalue})

`SetMaxIter`(*(FitRadialBackground)arg1*, *(int)arg2*) → None :

C++ signature : void SetMaxIter(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::FitRadialBackgroun {lvalue},unsigned int)

`SetTolerance`(*(FitRadialBackground)arg1*, *(float)arg2*) → None :

C++ signature : void SetTolerance(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::FitRadialBackgrou {lvalue},float)

`__init__`(*(object)arg1*[, *(MaskPtr)arg2*]) → None :

C++ signature : void __init__(_object* [,boost::shared_ptr<ost::img::MaskBase>])

C++ interface

typedef ost::img::ImageStateNonModAlgorithm< *FitRadialBackgroundBase* > **FitRadialBackground**

class **iplt::alg::FitRadialBackgroundBase**

Public Functions

FitRadialBackgroundBase()

FitRadialBackgroundBase(const ost::img::MaskPtr & m)

~FitRadialBackgroundBase()

unsigned int **GetMaxIter**()

void **SetMaxIter**(unsigned int maxiter)

Real **GetTolerance**()

void **SetTolerance**(Real tolerance)

geom::Vec2 **GetOrigin**()

template < typename T, class D >
void **VisitState**(const ost::img::ImageStateImpl< T, D > & is)

template < class D >
void **VisitState**(const ost::img::ImageStateImpl< Complex , D > & is)

Public Static Functons

String **GetAlgorithmName**()

Fit split gaussian 2D

Python interface

class iplt.alg.**FitSplitGauss2D**

GetAbsQuality(*(FitSplitGauss2D)arg1*) → float :

C++ signature : float GetAbsQuality(iplt::alg::FitSplitGauss2D {lvalue})

GetChi (*(FitSplitGauss2D)arg1*) → float :

C++ signature : float GetChi(iplt::alg::FitSplitGauss2D {lvalue})

GetGOF (*(FitSplitGauss2D)arg1*) → float :

C++ signature : float GetGOF(iplt::alg::FitSplitGauss2D {lvalue})

GetIterationCount (*(FitSplitGauss2D)arg1*) → int :

C++ signature : int GetIterationCount(iplt::alg::FitSplitGauss2D {lvalue})

GetQuality (*(FitSplitGauss2D)arg1*) → float :

C++ signature : float GetQuality(iplt::alg::FitSplitGauss2D {lvalue})

GetRelQuality (*(FitSplitGauss2D)arg1*) → float :

C++ signature : float GetRelQuality(iplt::alg::FitSplitGauss2D {lvalue})

GetSigAmp (*(FitSplitGauss2D)arg1*) → float :

C++ signature : float GetSigAmp(iplt::alg::FitSplitGauss2D {lvalue})

SetLim (*(FitSplitGauss2D)arg1*, *(float)arg2*, *(float)arg3*) → None :

C++ signature : void SetLim(iplt::alg::FitSplitGauss2D {lvalue},float,float)

SetMaxIter (*(FitSplitGauss2D)arg1*, *(int)arg2*) → None :

C++ signature : void SetMaxIter(iplt::alg::FitSplitGauss2D {lvalue},int)

SetSigma (*(FitSplitGauss2D)arg1*, *(float)arg2*) → None :

C++ signature : void SetSigma(iplt::alg::FitSplitGauss2D {lvalue},float)

__init__ (*(object)arg1*[, *(float)arg2*[, *(float)arg3*[, *(float)arg4*[, *(float)arg5*[, *(float)arg6*[, *(float)arg7*[, *(float)arg8*[, *(float)arg9*[, *(float)arg10*[, *(float)arg11*]]]]]]]]]]) → None :

C++ signature : void __init__(_object* [,float [,float [,float [,float [,float [,float [,float [,float [,float [,float]]]]]]]]]])

C++ interface

class **iplt::alg::FitSplitGauss2D**

2D Gaussian fitting algorithm

Public Type

Anonymous enum

Values:

- ID_A = =0 -
- ID_BX -
- ID_BY -
- ID_UX -
- ID_UY -
- ID_C -
- ID_W -
- ID_PX -

- ID_PY -
- ID_D -

Public Functions

FitSplitGauss2D(Real A = 1.0, Real Bx = 1.0, Real By = 1.0, Real C = 0.0, Real ux = 0.0, Real uy = 0.0, Real w = 0.0, Real Px = 0.0, Real Py = 0.0, Real delta = 0)

void **SetSigma**(Real s)

void **SetMaxIter**(int n)

void **SetLim**(Real l1, Real l2)

Real **GetChi**()

void **SetChi**(Real c)

Real **GetGOF**()

void **SetGOF**(Real g)

Real **GetQuality**()

void **SetQuality**(Real q)

Real **GetAbsQuality**()

void **SetAbsQuality**(Real q)

Real **GetRelQuality**()

void **SetRelQuality**(Real q)

int **GetIterationCount**()

Real **GetSigAmp**()

void **SetSigAmp**(Real sa)

void **Visit**(const ost::img::ConstImageHandle & i)

visitor implementation for images

Fourier mask

deprecated!

Python interface

class iplt.alg.**FourierMask**

__init__ (*(object)arg1*, *(Lattice)arg2*, *(Extent)arg3*, *(float)arg4*) → None :

C++ signature : void __init__(_object*,iplt::Lattice,ost::img::Extent,float)

C++ interface

class **iplt::alg::FourierMask**

Mask out lattice in fourier space.

Public Functions

FourierMask(const *Lattice* & lat, const ost::img::Extent & subext, Real hw)

ost::img::ImageHandle **Visit**(const ost::img::ConstImageHandle & ih)

2D Gaussian

Python interface

class `iplt.alg.`**`ParamsGauss2D`**

`GetA` (*(ParamsGauss2D)arg1*) → float :

C++ signature : float GetA(iplt::alg::ParamsGauss2D {lvalue})

`GetB` (*(ParamsGauss2D)arg1*) → object :

C++ signature : geom::Vec2 GetB(iplt::alg::ParamsGauss2D {lvalue})

`GetBx` (*(ParamsGauss2D)arg1*) → float :

C++ signature : float GetBx(iplt::alg::ParamsGauss2D {lvalue})

`GetBy` (*(ParamsGauss2D)arg1*) → float :

C++ signature : float GetBy(iplt::alg::ParamsGauss2D {lvalue})

`GetC` (*(ParamsGauss2D)arg1*) → float :

C++ signature : float GetC(iplt::alg::ParamsGauss2D {lvalue})

`GetP` (*(ParamsGauss2D)arg1*) → object :

C++ signature : geom::Vec2 GetP(iplt::alg::ParamsGauss2D {lvalue})

`GetPx` (*(ParamsGauss2D)arg1*) → float :

C++ signature : float GetPx(iplt::alg::ParamsGauss2D {lvalue})

`GetPy` (*(ParamsGauss2D)arg1*) → float :

C++ signature : float GetPy(iplt::alg::ParamsGauss2D {lvalue})

`GetU` (*(ParamsGauss2D)arg1*) → object :

C++ signature : geom::Vec2 GetU(iplt::alg::ParamsGauss2D {lvalue})

GetU((ParamsGauss2D)arg1) -> object :

C++ signature : geom::Vec2 GetU(iplt::alg::ParamsGauss2D {lvalue})

`GetUx` (*(ParamsGauss2D)arg1*) → float :

C++ signature : float GetUx(iplt::alg::ParamsGauss2D {lvalue})

`GetUy` (*(ParamsGauss2D)arg1*) → float :

C++ signature : float GetUy(iplt::alg::ParamsGauss2D {lvalue})

GetValue (*(ParamsGauss2D)arg1*, *(object)arg2*) → float :

C++ signature : float GetValue(iplt::alg::ParamsGauss2D {lvalue},geom::Vec2)

GetVolume (*(ParamsGauss2D)arg1*) → float :

C++ signature : float GetVolume(iplt::alg::ParamsGauss2D {lvalue})

GetW (*(ParamsGauss2D)arg1*) → float :

C++ signature : float GetW(iplt::alg::ParamsGauss2D {lvalue})

SetA (*(ParamsGauss2D)arg1*, *(float)arg2*) → None :

C++ signature : void SetA(iplt::alg::ParamsGauss2D {lvalue},float)

SetB (*(ParamsGauss2D)arg1*, *(object)arg2*) → None :

C++ signature : void SetB(iplt::alg::ParamsGauss2D {lvalue},geom::Vec2)

SetBx (*(ParamsGauss2D)arg1*, *(float)arg2*) → None :

C++ signature : void SetBx(iplt::alg::ParamsGauss2D {lvalue},float)

SetBy (*(ParamsGauss2D)arg1*, *(float)arg2*) → None :

C++ signature : void SetBy(iplt::alg::ParamsGauss2D {lvalue},float)

SetC (*(ParamsGauss2D)arg1*, *(float)arg2*) → None :

C++ signature : void SetC(iplt::alg::ParamsGauss2D {lvalue},float)

SetPx (*(ParamsGauss2D)arg1*, *(float)arg2*) → None :

C++ signature : void SetPx(iplt::alg::ParamsGauss2D {lvalue},float)

SetPy (*(ParamsGauss2D)arg1*, *(float)arg2*) → None :

C++ signature : void SetPy(iplt::alg::ParamsGauss2D {lvalue},float)

SetU (*(ParamsGauss2D)arg1*, *(object)arg2*) → None :

C++ signature : void SetU(iplt::alg::ParamsGauss2D {lvalue},geom::Vec2)

SetU((ParamsGauss2D)arg1, (object)arg2) -> None :

C++ signature : void SetU(iplt::alg::ParamsGauss2D {lvalue},geom::Vec2)

SetUx (*(ParamsGauss2D)arg1*, *(float)arg2*) → None :

C++ signature : void SetUx(iplt::alg::ParamsGauss2D {lvalue},float)

SetUy (*(ParamsGauss2D)arg1*, *(float)arg2*) → None :

C++ signature : void SetUy(iplt::alg::ParamsGauss2D {lvalue},float)

SetW (*(ParamsGauss2D)arg1*, *(float)arg2*) → None :

C++ signature : void SetW(iplt::alg::ParamsGauss2D {lvalue},float)

__init__ (*(object)arg1*, *(float)arg2*, *(float)arg3*, *(float)arg4*, *(float)arg5*, *(float)arg6*, *(float)arg7*, *(float)arg8*, *(float)arg9*, *(float)arg10*) → None :

C++ signature : void __init__(_object*,float,float,float,float,float,float,float,float,float)

C++ interface

class **iplt::alg::ParamsGauss2D**

store parameters that describe a 2D pseudo-gaussian

The parameters are meant for the following 2D pseudo-gaussian function:

x' = x-ux;

y' = y-uy;

x'' = x' cos[w] - y' sin[w]

y'' = x' sin[w] + y' cos[w]

f(x,y) = A exp[(x''/Bx)^2 + (y''/By)^2] + Px * x' + Py * y' + C

Public Functions

ParamsGauss2D(Real A, Real Bx, Real By, Real C, Real ux, Real uy, Real w, Real Px, Real Py)

ParamsGauss2D()

Real **GetA**()

void **SetA**(Real A)

Real **GetBx**()

void **SetBx**(Real Bx)

Real **GetBy**()

void **SetBy**(Real By)

Real **GetC**()

void **SetC**(Real C)

Real **GetUx**()

void **SetUx**(Real ux)

Real **GetUy**()

void **SetUy**(Real uy)

Real **GetW**()

void **SetW**(Real w)

Real **GetPx**()

void **SetPx**(Real Px)

Real **GetPy**()

void **SetPy**(Real Py)

geom::Vec2 **GetB**()

void **SetB**(const geom::Vec2 & b)

geom::Vec2 **GetU**()

void **SetU**(const geom::Vec2 & u)

geom::Vec2 **GetP**()

void **SetP**(const geom::Vec2 & p)

Real **GetVolume**()

Return volume of fitted gaussian.

Real **GetBackground**(const geom::Vec2 & p)

Real **GetValue**(const geom::Vec2 & p)

Hit and miss

Python interface

class iplt.alg.**HitAndMiss**

__init__(*(object)arg1*, *(ConstImageHandle)arg2*) → None :

C++ signature : void __init__(_object*,ost::img::ConstImageHandle)

C++ interface

class **iplt::alg::HitAndMiss**

Public Functions

HitAndMiss(const ost::img::ConstImageHandle & structure_element)

ost::img::ImageHandle **Visit**(const ost::img::ConstImageHandle & i)

Hysteresis

Python interface

class iplt.alg.**Hysteresis**

__init__(*(object)arg1*, *(float)arg2*, *(float)arg3*) → None :

C++ signature : void __init__(_object*,float,float)

C++ interface

class **iplt::alg::Hysteresis**

Public Functions

Hysteresis(Real upperthreshold, Real lowerthreshold)

void **Visit**(ost::img::ImageHandle & i)

Integrate area

Python interface

class iplt.alg.**IntegrateArea**

GetAverage(*(IntegrateAreaBase)arg1*[, *(int)arg2*[, *(int)arg3*]]) → float :

C++ signature : float GetAverage(iplt::alg::IntegrateAreaBase {lvalue} [,int [,int]])

GetNumSamples(*(IntegrateAreaBase)arg1*[, *(int)arg2*[, *(int)arg3*]]) → int :

C++ signature : unsigned int GetNumSamples(iplt::alg::IntegrateAreaBase {lvalue} [,int [,int]])

GetStandardDeviation(*(IntegrateAreaBase)arg1*[, *(int)arg2*[, *(int)arg3*]]) → float :

C++ signature : float GetStandardDeviation(iplt::alg::IntegrateAreaBase {lvalue} [,int [,int]])

GetSum(*(IntegrateAreaBase)arg1*[, *(int)arg2*[, *(int)arg3*]]) → float :

C++ signature : float GetSum(iplt::alg::IntegrateAreaBase {lvalue} [,int [,int]])

GetVariance(*(IntegrateAreaBase)arg1*[, *(int)arg2*[, *(int)arg3*]]) → float :

C++ signature : float GetVariance(iplt::alg::IntegrateAreaBase {lvalue} [,int [,int]])

__init__(*(object)arg1*[, *(Extent)arg2*]) → None :

C++ signature : void __init__(_object* [,ost::img::Extent])

C++ interface

typedef ost::img::ImageStateNonModAlgorithm< *IntegrateAreaBase* > **IntegrateArea**

class **iplt::alg::IntegrateAreaBase**

Public Functions

IntegrateAreaBase()

IntegrateAreaBase(const ost::img::Extent & e)

Real **GetSum**(int start = 0, int end = -1)

unsigned int **GetNumSamples**(int start = 0, int end = -1)

Real **GetAverage**(int start = 0, int end = -1)

Real **GetStandardDeviation**(int start = 0, int end = -1)

Real **GetVariance**(int start = 0, int end = -1)

template < typename T, class D >
void **VisitState**(const ost::img::ImageStateImpl< T, D > & isi)

Public Static Functons

String **GetAlgorithmName**()

Lattice extract

Python interface

class iplt.alg.**LatticeExtract**

GetCorrectedImage(*(LatticeExtract)arg1*) → ImageHandle :

C++ signature : ost::img::ImageHandle GetCorrectedImage(iplt::alg::LatticeExtract {lvalue})

GetEndSize(*(LatticeExtract)arg1*) → int :

C++ signature : int GetEndSize(iplt::alg::LatticeExtract {lvalue})

GetLattice(*(LatticeExtract)arg1*) → Lattice :

C++ signature : iplt::Lattice GetLattice(iplt::alg::LatticeExtract {lvalue})

GetList(*(LatticeExtract)arg1*) → LatticePointList :

C++ signature : iplt::alg::LatticePointList GetList(iplt::alg::LatticeExtract {lvalue})

GetStartSize(*(LatticeExtract)arg1*) → int :

C++ signature : int GetStartSize(iplt::alg::LatticeExtract {lvalue})

SetFitSize(*(LatticeExtract)arg1*, *(int)arg2*, *(int)arg3*) → None :

C++ signature : void SetFitSize(iplt::alg::LatticeExtract {lvalue},int,int)

SetLattice(*(LatticeExtract)arg1*, *(Lattice)arg2*) → None :

C++ signature : void SetLattice(iplt::alg::LatticeExtract {lvalue},iplt::Lattice)

__init__(*(object)arg1*, *(Lattice)arg2*, *(int)arg3*, *(int)arg4*, *(bool)arg5*, *(bool)arg6*) → None :

C++ signature : void __init__(_object*,iplt::Lattice,int,int,bool,bool)

C++ interface

class **iplt::alg::LatticeExtract**

Lattice Extract.

Use a given lattice to extract values from an image

Public Functions

LatticeExtract(const *Lattice* & lat, int size_start, int size_end, bool dynamic_fitting, bool pre_check)

initialize with lattice and fit radius

void **SetLattice**(const *Lattice* & l)

set new lattice to use

const *Lattice* & **GetLattice**()

retrieve current lattice

void **SetFitSize**(int start, int end)

int **GetStartSize**()

int **GetEndSize**()

void **SetDynamicMode**(bool d)

LatticePointList **GetList**()

Get resulting lattice point list.

void **Visit**(const ost::img::ConstImageHandle & img)

algorithm interface

ost::img::ImageHandle **GetCorrectedImage**()

Lattice filters

Python interface

class iplt.alg.**LatticeFilter**

__init__(*(object)arg1*, *(Lattice)arg2*, *(int)arg3*) → None :

C++ signature : void __init__(_object*,iplt::Lattice,unsigned int)

class iplt.alg.**GaussianLatticeFilter**

__init__(*(object)arg1*, *(Lattice)arg2*, *(int)arg3*) → None :

C++ signature : void __init__(_object*,iplt::Lattice,unsigned int)

C++ interface

class **iplt::alg::LatticeFilter**

Lattice Filter.

Public Functions

LatticeFilter(*Lattice* lattice, unsigned int size)

void **Visit**(ost::img::ImageHandle & ih)

class **iplt::alg::GaussianLatticeFilter**

Gaussian *Lattice* Filter.

Public Functions

GaussianLatticeFilter(*Lattice* lattice, unsigned int size)

void **Visit**(ost::img::ImageHandle & ih)

Lattice Gaussian Extract

ython interface

class iplt.alg.**LatticeGaussianExtract**

GetBoxSize(*(LatticeGaussianExtract)arg1*) → int :

C++ signature : unsigned int GetBoxSize(iplt::alg::LatticeGaussianExtract {lvalue})

GetLattice(*(LatticeGaussianExtract)arg1*) → Lattice :

C++ signature : iplt::Lattice GetLattice(iplt::alg::LatticeGaussianExtract {lvalue})

GetList(*(LatticeGaussianExtract)arg1*) → LatticePointList :

C++ signature : iplt::alg::LatticePointList GetList(iplt::alg::LatticeGaussianExtract {lvalue})

GetMaxIter(*(LatticeGaussianExtract)arg1*) → int :

C++ signature : unsigned int GetMaxIter(iplt::alg::LatticeGaussianExtract {lvalue})

SetBoxSize(*(LatticeGaussianExtract)arg1*, *(int)arg2*) → None :

C++ signature : void SetBoxSize(iplt::alg::LatticeGaussianExtract {lvalue},unsigned int)

SetLattice(*(LatticeGaussianExtract)arg1*, *(Lattice)arg2*) → None :

C++ signature : void SetLattice(iplt::alg::LatticeGaussianExtract {lvalue},iplt::Lattice)

SetLim(*(LatticeGaussianExtract)arg1*, *(float)arg2*, *(float)arg3*) → None :

C++ signature : void SetLim(iplt::alg::LatticeGaussianExtract {lvalue},float,float)

SetMaxIter(*(LatticeGaussianExtract)arg1*, *(int)arg2*) → None :

C++ signature : void SetMaxIter(iplt::alg::LatticeGaussianExtract {lvalue},unsigned int)

__init__(*(object)arg1*, *(Lattice)arg2*, *(int)arg3*[, *(int)arg4*[, *(float)arg5*[, *(float)arg6*]]]) → None :

C++ signature : void __init__(_object*,iplt::Lattice,int [,int [,float [,float]]])

__init__((object)arg1, (LatticePointList)arg2, (int)arg3 [, (int)arg4 [, (float)arg5 [, (float)arg6]]]) -> None :

C++ signature : void __init__(_object*,iplt::alg::LatticePointList,int [,int [,float [,float]]])

C++ interface

class **iplt::alg::LatticeGaussianExtract**

Lattice Gaussian Fit Extract.

Use a given lattice to perform 2D gaussian fit on each predicted lattice position

Public Functions

LatticeGaussianExtract(const *Lattice* & lat, unsigned int box_size, unsigned int maxiter = 100, Real abslim = 1.0e-12, Real rellim = 1.0e-12)

initialize with lattice and fit radius

LatticeGaussianExtract(const LatticePointList & lpl, unsigned int box_size, unsigned int maxiter = 100, Real abslim = 1.0e-12, Real rellim = 1.0e-12)

initialize with lattice point list and fit radius

void **SetLattice**(const *Lattice* & l)

set new lattice to use

const *Lattice* & **GetLattice**()

retrieve current lattice

void **SetBoxSize**(unsigned int sz)

set box size to use

unsigned int **GetBoxSize**()

retrieve current box size

void **SetMaxIter**(unsigned int val)

set maximum iteration count

unsigned int **GetMaxIter**()

retrieve maximum iteration count

void **SetLim**(Real lim1, Real lim2)

LatticePointList **GetList**()

Get resulting lattice point list.

void **Visit**(const ost::img::ConstImageHandle & img)

algorithm interface

Lattice Search

The lattice search algorithm is one of the more complex algorithms provided by IPLT. To quote the 2007 publication on IPLT:

In a first step a peak search is performed and the parameters (sensitivity and desired peak size) of the peak search algorithms are refined according to the number of peaks found. This is done to avoid too many peaks (likely to introduce more noise peaks) or too few peaks. From this peak list, the distance vector between each pair of peaks is calculated. All these distance vectors contribute to a new image, the vector image. The vector image is generated starting from an image with all the pixel values set to zero. The pixel value is increased by one at each position of the image (index) defined by the components of the distance vectors. On this image a second peak search is performed and a second vector image is generated from these peaks. In this second vector image the two closest points to the centre that are not collinear are taken as a first guess for the lattice vectors.

As a consequence, there are lots of parameters to set, as detailed in the interface description below.

Info Entry

```
<LatticeSearch>
  <GaussFilterStrength value="1.0" type="float"/>
  <PeakSearchSensitivity value="0.1" type="float"/>
  <MinimalLatticeLength value="10.0" type="float"/>
  <MaximalLatticeLength value="1.0e4" type="float"/>
  <PeakSearchParams>
    <OuterWindowSize value="10" type="int"/>
    <OuterWindowSensitivity value="1.0" type="float"/>
    <InnerWindowSize value="0" type="int"/>
    <InnerWindowSensitivity value="0.0" type="float"/>
  </PeakSearchParams>
</LatticeSearch>
```

Python interface
class `iplt.alg.`**`LatticeSearch`**

`AddPeakSearchExclusion`(*(LatticeSearch)arg1*, *(Extent)arg2*) → None :

C++ signature : void AddPeakSearchExclusion(iplt::alg::LatticeSearch {lvalue},ost::img::Extent)

`GetBestPoints`(*(LatticeSearch)arg1*) → object :

C++ signature : std::vector<geom::Vec2, std::allocator<geom::Vec2> > GetBestPoints(iplt::alg::LatticeSearch {lvalue})

`GetClosePoints`(*(LatticeSearch)arg1*) → PointList :

C++ signature : ost::img::PointList GetClosePoints(iplt::alg::LatticeSearch {lvalue})

`GetImagePeaks`(*(LatticeSearch)arg1*) → PeakList :

C++ signature : std::vector<ost::img::Peak, std::allocator<ost::img::Peak> > GetImagePeaks(iplt::alg::LatticeSearch {lvalue})

`GetImagePeaks2`(*(LatticeSearch)arg1*) → PeakList :

C++ signature : std::vector<ost::img::Peak, std::allocator<ost::img::Peak> > GetImagePeaks2(iplt::alg::LatticeSearch {lvalue})

`GetLattice`(*(LatticeSearch)arg1*) → Lattice :

C++ signature : iplt::Lattice GetLattice(iplt::alg::LatticeSearch {lvalue})

`GetTempLattice`(*(LatticeSearch)arg1*) → Lattice :

C++ signature : iplt::Lattice GetTempLattice(iplt::alg::LatticeSearch {lvalue})

`GetVectorImage`(*(LatticeSearch)arg1*) → ImageHandle :

C++ signature : ost::img::ImageHandle GetVectorImage(iplt::alg::LatticeSearch {lvalue})

SetDVIPeakSearchParams((*LatticeSearch)arg1*, *(int)arg2*, *(float)arg3*, *(int)arg4*, *(float)arg5*) → None :

C++ signature : void SetDVIPeakSearchParams(iplt::alg::LatticeSearch {lvalue},int,float,int,float)

SetGaussFilterStrength((*LatticeSearch)arg1*, *(float)arg2*) → None :

C++ signature : void SetGaussFilterStrength(iplt::alg::LatticeSearch {lvalue},float)

SetMaximalLatticeLength((*LatticeSearch)arg1*, *(float)arg2*) → None :

C++ signature : void SetMaximalLatticeLength(iplt::alg::LatticeSearch {lvalue},float)

SetMinimalLatticeLength((*LatticeSearch)arg1*, *(float)arg2*) → None :

C++ signature : void SetMinimalLatticeLength(iplt::alg::LatticeSearch {lvalue},float)

SetMinimalQuality((*LatticeSearch)arg1*, *(float)arg2*) → None :

C++ signature : void SetMinimalQuality(iplt::alg::LatticeSearch {lvalue},float)

SetPeakSearchLimits((*LatticeSearch)arg1*, *(int)arg2*, *(int)arg3*, *(int)arg4*) → None :

C++ signature : void SetPeakSearchLimits(iplt::alg::LatticeSearch {lvalue},int,int,int)

SetPeakSearchParams((*LatticeSearch)arg1*, *(int)arg2*, *(float)arg3*, *(int)arg4*, *(float)arg5*[, *(int)arg6*[, *(float)arg7*]]) → None :

C++ signature : void SetPeakSearchParams(iplt::alg::LatticeSearch {lvalue},int,float,int,float [,int [,float]])

__init__((*object)arg1*) → None :

C++ signature : void __init__(_object*)

iplt.alg.**UpdateFromInfo**((*LatticeSearch)arg1*, *(object)arg2*) → None :

C++ signature : void UpdateFromInfo(iplt::alg::LatticeSearch {lvalue},ost::info::InfoGroup)

UpdateFromInfo((LatticeExtract)arg1, (object)arg2) -> None :

C++ signature : void UpdateFromInfo(iplt::alg::LatticeExtract {lvalue},ost::info::InfoGroup)

UpdateFromInfo((LatticeGaussianExtract)arg1, (object)arg2) -> None :

C++ signature : void UpdateFromInfo(iplt::alg::LatticeGaussianExtract {lvalue},ost::info::InfoGroup)

Updates a given lattice search object from the given info group. The following entries are recognized:

Name	Type
GaussFilterStrength	float
PeakSearchSensitivity	float
MinimalLatticeLength	float
MaximalLatticeLength	float
PeakSearchParams/OuterWindowSize	int
PeakSearchParams/OuterWindowSensitivity	float
PeakSearchParams/InnerWindowSize	int
PeakSearchParams/InnerWindowSensitivity	float

C++ interface

class **iplt::alg::LatticeExtract**

Lattice Extract.

Use a given lattice to extract values from an image

Public Functions

LatticeExtract(const *Lattice* & lat, int size_start, int size_end, bool dynamic_fitting, bool pre_check)

initialize with lattice and fit radius

void **SetLattice**(const *Lattice* & l)

set new lattice to use

const *Lattice* & **GetLattice**()

retrieve current lattice

void **SetFitSize**(int start, int end)

int **GetStartSize**()

int **GetEndSize**()

void **SetDynamicMode**(bool d)

LatticePointList **GetList**()

Get resulting lattice point list.

void **Visit**(const ost::img::ConstImageHandle & img)

algorithm interface

ost::img::ImageHandle **GetCorrectedImage**()

DLLEXPORT_IPLT_ALG void **UpdateFromInfo**(*LatticeExtract* & le, const ost::info::InfoGroup & g)

Set lattice extract parameters based on info group.

The parameters from the given *LatticeExtract* object are modified according to the given InfoGroup instance. The resulting *LatticeExtract* object is returned.

Note that only the parameters that are actually present in the info group are used, allowing cumulative use of this function.

Median filter

Python interface

class iplt.alg.**MedianFilter**

__init__(*(object)arg1* [, *(int)arg2*]) → None :

C++ signature : void __init__(_object* [,unsigned int])

C++ interface

typedef ost::img::ImageStateConstModOPAlgorithm< *MedianFilterBase* > **MedianFilter**

class **iplt::alg::MedianFilterBase**

Public Functions

MedianFilterBase(unsigned int size = 3)

template < typename V, class D >
ost::img::ImageStateBasePtr **VisitState**(const ost::img::ImageStateImpl< V, D > & isi)

Public Static Functons

String **GetAlgorithmName**()

Non-max supressor

Python interface

class iplt.alg.**NonMaxSupressor**

__init__(*(object)arg1*) → None :

C++ signature : void __init__(_object*)

C++ interface

class **iplt::alg::NonMaxSupressor**

Public Functions

NonMaxSupressor()

ost::img::ImageHandle **Visit**(const ost::img::ConstImageHandle & i)

Open

Description

Calculates the binary morphological operation open on the image. The constructor expects a structuring element in form of an image handle. The structuring element can either be created manually or one of the *default elements* can be taken.

Examples

C++

```
#include <iplt/image.hh>
#include <iplt/alg/open.hh>

using namespace iplt;

int main()
{
  ImageHandle img = LoadImage("...");
  img.ApplyIP( alg::Open(alg::Create4n()) );

  return 0;
}
```

Python

```
from iplt import *
img = LoadImage("...")
img.ApplyIP( alg.Open(alg.Create4n()) )
```

Example image

Python interface

class `iplt.alg.`**`Open`**

`__init__`(*(object)arg1*, *(ConstImageHandle)arg2*) → None :

C++ signature : void __init__(_object*,ost::img::ConstImageHandle)

C++ interface

class **iplt::alg::Open**

Public Functions

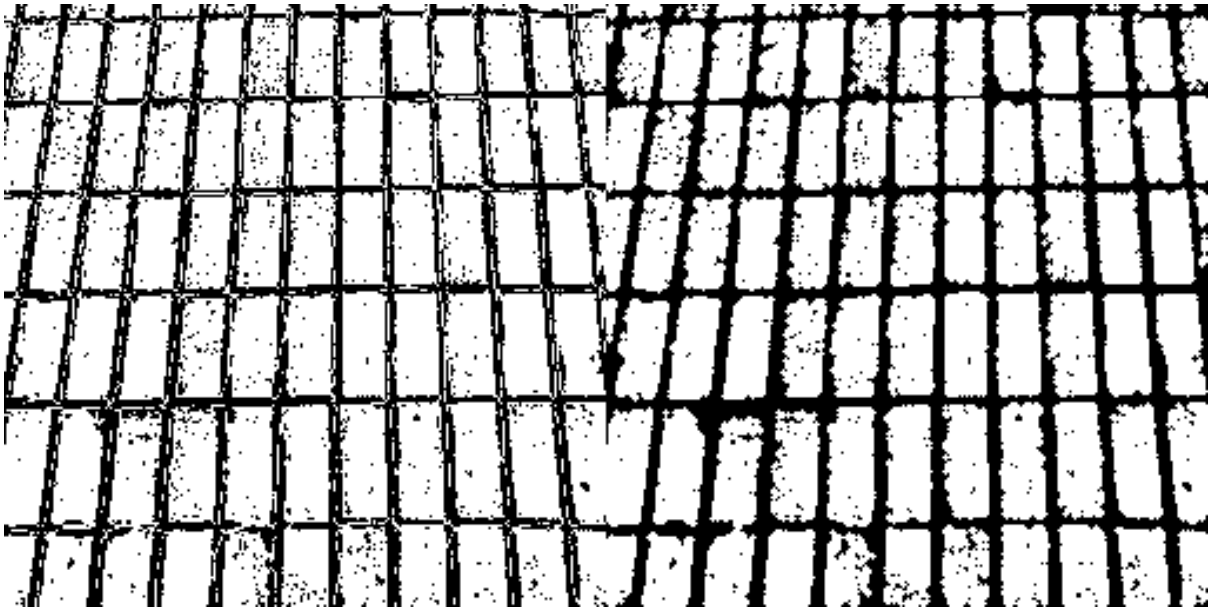

Figure 3.5: left: original image, right image after open algorithm

Open(const ost::img::ConstImageHandle & structure_element)

ost::img::ImageHandle **Visit**(const ost::img::ConstImageHandle & i)

Peak Integration

Python interface

class `iplt.alg.`**`PeakIntegration`**

`GetBackground`(*(PeakIntegration)arg1*) → float :

C++ signature : float GetBackground(iplt::alg::PeakIntegration {lvalue})

`GetEndSize`(*(PeakIntegration)arg1*) → int :

C++ signature : int GetEndSize(iplt::alg::PeakIntegration {lvalue})

`GetFinalSize`(*(PeakIntegration)arg1*) → int :

C++ signature : int GetFinalSize(iplt::alg::PeakIntegration {lvalue})

`GetMaximalSize`(*(PeakIntegration)arg1*) → Size :

C++ signature : ost::img::Size GetMaximalSize(iplt::alg::PeakIntegration {lvalue})

`GetQuality`(*(PeakIntegration)arg1*) → float :

C++ signature : float GetQuality(iplt::alg::PeakIntegration {lvalue})

`GetSigma`(*(PeakIntegration)arg1*) → float :

C++ signature : float GetSigma(iplt::alg::PeakIntegration {lvalue})

`GetStartSize`(*(PeakIntegration)arg1*) → int :

C++ signature : int GetStartSize(iplt::alg::PeakIntegration {lvalue})

GetVolume(*(PeakIntegration)arg1*) → float :

C++ signature : float GetVolume(iplt::alg::PeakIntegration {lvalue})

SetBackground(*(PeakIntegration)arg1*, *(float)arg2*) → None :

C++ signature : void SetBackground(iplt::alg::PeakIntegration {lvalue},float)

SetBackgroundRimSize(*(PeakIntegration)arg1*, *(int)arg2*) → None :

C++ signature : void SetBackgroundRimSize(iplt::alg::PeakIntegration {lvalue},int)

SetMethod(*(PeakIntegration)arg1*, *(int)arg2*) → None :

C++ signature : void SetMethod(iplt::alg::PeakIntegration {lvalue},int)

SetSigma(*(PeakIntegration)arg1*, *(float)arg2*) → None :

C++ signature : void SetSigma(iplt::alg::PeakIntegration {lvalue},float)

SetVolume(*(PeakIntegration)arg1*, *(float)arg2*) → None :

C++ signature : void SetVolume(iplt::alg::PeakIntegration {lvalue},float)

__init__(*(object)arg1*[, *(int)arg2*[, *(int)arg3*[, *(int)arg4*]]]) → None :

C++ signature : void __init__(_object* [,int [,int [,int]]])

C++ interface

class **iplt::alg::PeakIntegration**

Public Functions

PeakIntegration(int startsize = 5, int endsize = 6, int laxsize = 2)

Real **GetVolume**()

void **SetVolume**(Real v)

Real **GetSigma**()

void **SetSigma**(Real s)

Real **GetBackground**()

void **SetBackground**(Real b)

Real **GetBackgroundSigma**()

void **SetBackgroundSigma**(Real b)

Real **GetAverageRatio**()

void **SetAverageRatio**(Real r)

Real **GetQuality**()

Real **GetWeight**()

returns GaussianP(i/sigi)

int **GetFinalSize**()

int **GetStartSize**()

int **GetEndSize**()

void **SetBackgroundRimSize**(int bgs)

ost::img::Size **GetMaximalSize**()

void **SetMethod**(int m)

void **Visit**(const ost::img::ConstImageHandle & i)

visitor implementation for images

Peak search

Python interface

class iplt.alg.**PeakSearch**

AddExclusion (*(PeakSearch)arg1*, *(Extent)arg2*) → None :

C++ signature : void AddExclusion(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::PeakSearchBase> {lvalue},ost::img::Extent)

ClearPeakList (*(PeakSearch)arg1*) → None :

C++ signature : void ClearPeakList(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::PeakSearchBase> {lvalue})

GetInnerCountLimit (*(PeakSearch)arg1*) → int :

C++ signature : unsigned int GetInnerCountLimit(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::Pea {lvalue})

GetInnerCountSensitivity (*(PeakSearch)arg1*) → float :

C++ signature : float GetInnerCountSensitivity(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::PeakS {lvalue})

GetInnerSensitivity (*(PeakSearch)arg1*) → float :

C++ signature : float GetInnerSensitivity(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::PeakSearchI {lvalue})

GetInnerWindowSize (*(PeakSearch)arg1*) → int :

C++ signature : unsigned int GetInnerWindowSize(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::Pe {lvalue})

GetMode (*(PeakSearch)arg1*) → int :

C++ signature : unsigned int GetMode(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::PeakSearchBa {lvalue})

GetOuterSensitivity (*(PeakSearch)arg1*) → float :

C++ signature : float GetOuterSensitivity(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::PeakSearch {lvalue})

GetOuterWindowSize (*(PeakSearch)arg1*) → int :

C++ signature : unsigned int GetOuterWindowSize(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::Pe {lvalue})

GetPeakList (*(PeakSearch)arg1*) → PeakList :

C++ signature : std::vector<ost::img::Peak, std::allocator<ost::img::Peak> > GetPeakList(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::PeakSearchBase> {lvalue})

SetInnerCountLimit (*(PeakSearch)arg1*, *(int)arg2*) → None :

C++ signature : void SetInnerCountLimit(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::PeakSearch {lvalue},unsigned int)

SetInnerCountSensitivity (*(PeakSearch)arg1*, *(float)arg2*) → None :

C++ signature : void SetInnerCountSensitivity(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::PeakS {lvalue},float)

SetInnerSensitivity (*(PeakSearch)arg1*, *(float)arg2*) → None :

C++ signature : void SetInnerSensitivity(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::PeakSearchB {lvalue},float)

SetInnerWindowSize (*(PeakSearch)arg1*, *(int)arg2*) → None :

C++ signature : void SetInnerWindowSize(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::PeakSearc {lvalue},unsigned int)

SetMode (*(PeakSearch)arg1*, *(int)arg2*) → None :

C++ signature : void SetMode(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::PeakSearchBase> {lvalue},unsigned int)

SetOuterSensitivity (*(PeakSearch)arg1*, *(float)arg2*) → None :

C++ signature : void SetOuterSensitivity(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::PeakSearchI {lvalue},float)

SetOuterWindowSize (*(PeakSearch)arg1*, *(int)arg2*) → None :

C++ signature : void SetOuterWindowSize(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::PeakSearc {lvalue},unsigned int)

SetThreshold (*(PeakSearch)arg1*, *(float)arg2*) → None :

C++ signature : void SetThreshold(ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::PeakSearchBase> {lvalue},float)

__init__ (*(object)arg1*[, *(int)arg2*[, *(float)arg3*[, *(int)arg4*[, *(float)arg5*[, *(int)arg6*[, *(float)arg7*[, *(int)arg8*]]]]]]]) → None :

C++ signature : void __init__(_object* [,int [,float [,int [,float [,int [,float [,unsigned int]]]]]]])

C++ interface

typedef ost::img::ImageStateNonModAlgorithm< *PeakSearchBase* > **PeakSearch**

class **iplt::alg::PeakSearchBase**

Peak Search.

Applies a peak search on the supplied image

Public Type

modes enum

Values:

- MODES_START -
- ABSOLUTE -
- RELATIVE_TO_PEAK -
- RELATIVE_TO_STDEV -
- MODES_END -

Public Functions

PeakSearchBase(unsigned int outer_win = 10, Real outer_sens = 1.0, unsigned int inner_win = 0, Real inner_sens = 0.0, int inner_count_lim = 0, Real inner_count_sens = 0.0, unsigned int mode_ = RELATIVE_TO_STDEV)

Initialization.

template < typename T, class D >
void **VisitState**(const ost::img::ImageStateImpl< T, D > & isi)

image state algorithm interface

ost::img::PeakList **GetPeakList**()

Main “return” function: returns the PeakList.

void **ClearPeakList**()

Clears the peak list removing all peaks in the list.

void **SetOuterWindowSize**(unsigned int outer_win)

Get/Set the size of the outer window and the sensitivity of the search.

unsigned int **GetOuterWindowSize**()

void **SetOuterSensitivity**(Real outer_sens)

Real **GetOuterSensitivity**()

void **SetInnerWindowSize**(unsigned int outer_win)

Get/Set the size of the inner window and the sensitivity of the search.

unsigned int **GetInnerWindowSize**()

void **SetInnerSensitivity**(Real outer_sens)

Real **GetInnerSensitivity**()

void **SetInnerCountLimit**(unsigned int limit)

Get/Set the inner count lim and sens.

unsigned int **GetInnerCountLimit**()

void **SetInnerCountSensitivity**(Real inner_count_sens)

Real **GetInnerCountSensitivity**()

void **SetThreshold**(Real minimum)

Set the minimal value a ost::img::Point has to have to be a peak.

void **AddExclusion**(const ost::img::Extent & e)

void **SetMode**(unsigned int mode)

Set/get the peak search mode.

unsigned int **GetMode**()

Public Static Functons

String **GetAlgorithmName**()

Prune

Python interface

class iplt.alg.**Prune**

__init__(*(object)arg1*, *(int)arg2*) → None :

C++ signature : void __init__(_object*,unsigned int)

C++ interface

class **iplt::alg::Prune**

Public Functions

Prune(unsigned int rounds)

void **Visit**(ost::img::ImageHandle & im)

Restrict

Python interface

class iplt.alg.**Restrict**

__init__(*(object)arg1*, *(str)arg2*) → None :

C++ signature : void __init__(_object*,std::string)

__init__((object)arg1, (Restrict)arg2) -> None :

C++ signature : void __init__(_object*,iplt::alg::Restrict)

C++ interface

class **iplt::alg::Restrict**

Public Functions

Restrict(const Symmetry & s)

Restrict(String s)

ReflectionList **Visit**(const ReflectionList &)

Skeleton

Python interface

class `iplt.alg.`**`Skeleton`**

`__init__`(*(object)arg1*) → None :

C++ signature : void __init__(_object*)

C++ interface

class **iplt::alg::Skeleton**

Public Functions

Skeleton(void)

void **Visit**(ost::img::ImageHandle & i)

Sobel edge detection

Description

Performs a sobel edge detection. The gradient vectors are represented by complex values.

Examples

C++

```
#include <ost/io/load_map.hh>
#include <iplt/alg/sobel.hh>


int main()
{
  ImageHandle img = ost::io::LoadImage("lena.tif");
  img.ApplyIP( iplt::alg::Sobel() );

  return 0;
}
```

Python

```
from ost.io import *
from iplt import *
img = LoadImage("lena.tif")
img.ApplyIP( alg.Sobel() )
```

Example image

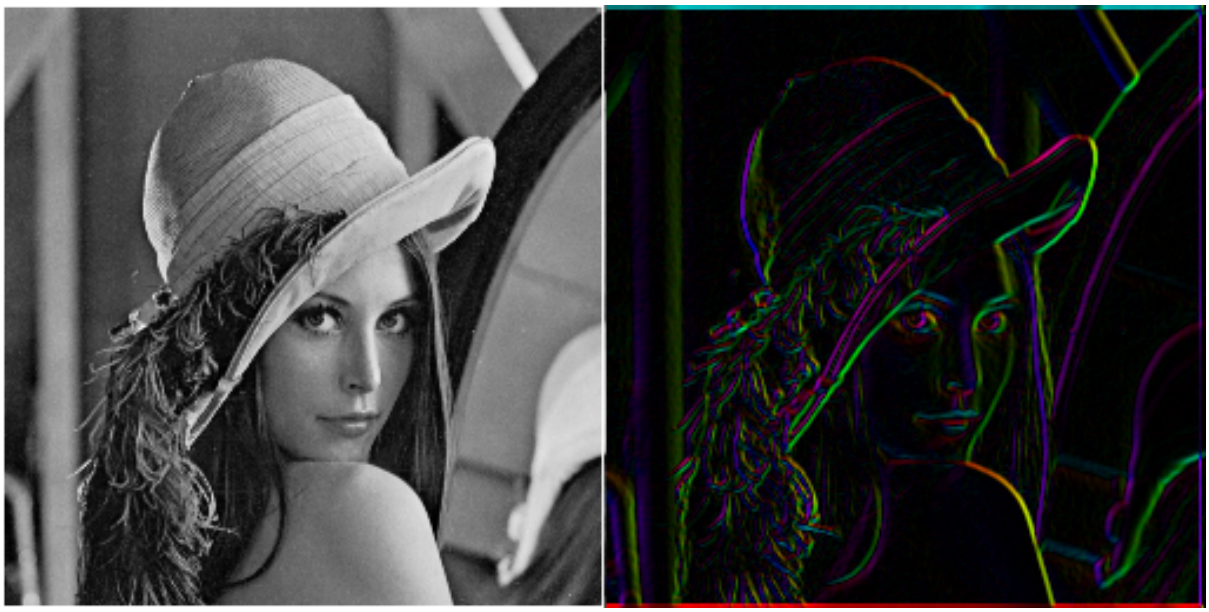

Figure 3.6: left: original image, right image after edge detection

Python interface

class `iplt.alg.`**`Sobel`**

`__init__`(*(object)arg1*) → None :

C++ signature : void __init__(_object*)

C++ interface

class **iplt::alg::Sobel**

Public Functions

Sobel(void)

ost::img::ImageHandle **Visit**(const ost::img::ConstImageHandle & i)

Split Gauss 2d

Python interface

class `iplt.alg.`**`ParamsSplitGauss2D`**

`GetDelta`(*(ParamsSplitGauss2D)arg1*) → float :

C++ signature : float GetDelta(iplt::alg::ParamsSplitGauss2D {lvalue})

`SetDelta`(*(ParamsSplitGauss2D)arg1*, *(float)arg2*) → None :

C++ signature : void SetDelta(iplt::alg::ParamsSplitGauss2D {lvalue},float)

`__init__`()
Raises an exception This class cannot be instantiated from Python

C++ interface

class **iplt::alg::ParamsSplitGauss2D**

Split Gauss2D Parameters

Stores the Parameters for a Pseudo Split Gauss2D function, given by:

x0' = x-ux

x1' = x0'-delta

x2' = x0'+delta

y' = y-uy

x1'' = x1' cos[w] - y' sin[w]

x2'' = x2' cos[w] - y' sin[w]

y'' = x0' sin[w] + y' cos[w]

f(x,y) = A (exp[(x1''/Bx)^2]+exp[(x2''/Bx)^2]) exp[(y''/By)^2] + Px * x0' + Py * y' + C

Note: This is not a true is-an relationship, but refactoring *ParamsGauss2D* to add another layer in order to follow strict OOD codex seems overkill.

Public Functions

ParamsSplitGauss2D(Real A, Real Bx, Real By, Real C, Real ux, Real uy, Real w, Real Px, Real Py, Real delta)

ParamsSplitGauss2D()

Real **GetDelta**()

void **SetDelta**(Real d)

Real **GetValue**(const geom::Vec2 & p)

Spread gauss 2D

C++ Interface

class **iplt::alg::SpreadGauss2D**

Public Functions

SpreadGauss2D(const geom::Vec2 & xy, Real amp, Real sigma)

void **Visit**(ost::img::ImageHandle & ih)

Structuring Elements

IPLT provides following functions to generate several commonly used structuring elements.

- Create8n
- Create4n
- CreateCornerTL
- CreateCornerBL
- CreateCornerTR
- CreateCornerBL

Error: Macro Image(se8n.png,nolink) failed

```
Attachment AlgStructuringElements: se8n.png does not exist.
```

Error: Macro Image(se4n.png,nolink) failed

```
Attachment AlgStructuringElements: se4n.png does not exist.
```

Error: Macro Image(seCornerTL.png,nolink) failed

```
Attachment AlgStructuringElements: seCornerTL.png does not exist.
```

Error: Macro Image(seCornerBL.png,nolink) failed

```
Attachment AlgStructuringElements: seCornerBL.png does not exist.
```

Error: Macro Image(seCornerTR.png,nolink) failed

```
Attachment AlgStructuringElements: seCornerTR.png does not exist.
```

Error: Macro Image(seCornerBL.png,nolink) failed

```
Attachment AlgStructuringElements: seCornerBL.png does not exist.
```

8n

4n

CornerTL

CornerBL

CornerTR

CornerBL

Subtract radial background

Python interface

class iplt.alg.**SubtractRadialBackground**

__init__(*(object)arg1*, *(object)arg2*[, *(MaskPtr)arg3*]) → None :

C++ signature : void __init__(_object*,geom::Vec2 [,boost::shared_ptr<ost::img::MaskBase>])

C++ interface

typedef ost::img::ImageStateConstModIPAlgorithm< *SubtractRadialBackgroundBase* > **SubtractRadialBackground**

class **iplt::alg::SubtractRadialBackgroundBase**

Public Functions

SubtractRadialBackgroundBase()

SubtractRadialBackgroundBase(geom::Vec2 origin, const ost::img::MaskPtr & m = Mask(ost::img::Extent ()))

template < typename T, class D >
void **VisitState**(ost::img::ImageStateImpl< T, D > & is)

Public Static Functons

String **GetAlgorithmName**()

3.2.2 Reflection list algorithms

Ewald tilter

Python interface

class iplt.alg.**EwaldTilter**

__init__(*(object)arg1*, *(float)arg2*) → None :

C++ signature : void __init__(_object*,float)

C++ interface

class **iplt::alg::EwaldTilter**

Public Functions

EwaldTilter(Real lambda)

ReflectionList **Visit**(const ReflectionList &)

Expand

Python interface

class iplt.alg.**Expand**

__init__(*(object)arg1*[, *(bool)arg2*]) → None :

C++ signature : void __init__(_object* [,bool])

__init__((object)arg1, (str)arg2 [, (bool)arg3]) -> None :

C++ signature : void __init__(_object*,std::string [,bool])

__init__((object)arg1, (Symmetry)arg2 [, (bool)arg3]) -> None :

C++ signature : void __init__(_object*,iplt::Symmetry [,bool])

C++ interface

class **iplt::alg::Expand**

Public Functions

Expand(bool hermitian = true)

Expand(const Symmetry & s, bool hermitian = true)

Expand(String s, bool hermitian = true)

ReflectionList **Visit**(const ReflectionList &)

Composite lattice line fit

Python interface

class `iplt.alg.`**`LLFitAmpComposite`**

`Add`(*(LLFitAmpComposite)arg1*, *(LLFitBase)arg2*) → None :

C++ signature : void Add(iplt::alg::LLFitAmpComposite {lvalue},boost::shared_ptr<iplt::alg::LLFitBase>)

`__init__`(*(object)arg1*) → None :

C++ signature : void __init__(_object*)

C++ interface

class **iplt::alg::LLFitAmpComposite**

Public Functions

LLFitAmpComposite()

LLFitBasePtr **Clone**()

void **Apply**()

do actual fitting

std::pair< Real , Real > **CalcIntensAt**(Real zstar)

unsigned int **GetIterCount**()

std::multimap< Real , Complex > **GetComponents**()

void **Cleanup**()

void **Add**(const LLFitBasePtr & f)

Gaussian sum lattice line fit

Python interface

class iplt.alg.**LLFitAmpGaussianSum**

__init__(*(object)arg1*, *(int)arg2*, *(float)arg3*[, *(guessmode)arg4*]) → None :

C++ signature : void __init__(_object*,int,float [,iplt::alg::LLFitBase::guessmode])

C++ interface

class **iplt::alg::LLFitAmpGaussianSum**

Public Functions

LLFitAmpGaussianSum(unsigned int count, Real sampling, guessmode gm = GUESSMODE_RANDOM)

LLFitBasePtr **Clone**()

void **Apply**()

do actual fitting

std::pair< Real , Real > **CalcIntensAt**(Real zstar)

std::multimap< Real , Complex > **GetComponents**()

Lattice line fit with sinc interpolation

Python interface

class iplt.alg.**LLFitAmpSincIntpol**

__init__(*(object)arg1*, *(int)arg2*, *(float)arg3*[, *(guessmode)arg4*]) → None :

C++ signature : void __init__(_object*,int,float [,iplt::alg::LLFitBase::guessmode])

class iplt.alg.**LLFitAmpSincIntpolHerm**

__init__(*(object)arg1*, *(int)arg2*, *(float)arg3*[, *(guessmode)arg4*]) → None :

C++ signature : void __init__(_object*,int,float [,iplt::alg::LLFitBase::guessmode])

C++ interface

class **iplt::alg::LLFitAmpSincIntpol**

Public Functions

LLFitAmpSincIntpol(unsigned int count, Real sampling, guessmode gm = GUESSMODE_BINAVERAGE)

LLFitBasePtr **Clone**()

void **Apply**()

do actual fitting

std::pair< Real , Real > **CalcIntensAt**(Real zstar)

std::multimap< Real , Complex > **GetComponents**()

class **iplt::alg::LLFitAmpSincIntpolHerm**

Public Functions

LLFitAmpSincIntpolHerm(unsigned int count, Real sampling, guessmode gm = GUESSMODE_BINAVERAGE)

LLFitBasePtr **Clone**()

void **Apply**()

do actual fitting

std::pair< Real , Real > **CalcIntensAt**(Real zstar)

std::multimap< Real , Complex > **GetComponents**()

SQ Lattice ine fit with sinc interpolation

Python interface

class `iplt.alg.`**`LLFitAmpSincIntpolSQ`**

`__init__`(*(object)arg1*, *(int)arg2*, *(float)arg3*) → None :

C++ signature : void __init__(_object*,int,float)

C++ interface

class **iplt::alg::LLFitAmpSincIntpolSQ**

Public Functions

LLFitAmpSincIntpolSQ(unsigned int count, Real sampling, guessmode gm = GUESSMODE_RANDOM)

~LLFitAmpSincIntpolSQ()

LLFitBasePtr **Clone**()

void **Apply**()

do actual fitting

std::pair< Real , Real > **CalcIntensAt**(Real zstar)

std::multimap< Real , Complex > **GetComponents**()

Local lattice line binning

Python interface

class `iplt.alg.`**`LocalLatticeLineBin`**

`GetAverage`(*(LocalLatticeLineBin)arg1*, *(ReflectionIndex)arg2*) → float :

C++ signature : float GetAverage(iplt::alg::LocalLatticeLineBin {lvalue},iplt::ReflectionIndex)

`GetStandardDeviation`(*(LocalLatticeLineBin)arg1*, *(ReflectionIndex)arg2*) → float :

C++ signature : float GetStandardDeviation(iplt::alg::LocalLatticeLineBin {lvalue},iplt::ReflectionIndex)

`GetWeightedAverage`(*(LocalLatticeLineBin)arg1*, *(ReflectionIndex)arg2*) → float :

C++ signature : float GetWeightedAverage(iplt::alg::LocalLatticeLineBin {lvalue},iplt::ReflectionIndex)

`__init__`(*(object)arg1*, *(ReflectionList)arg2*, *(str)arg3*, *(str)arg4*, *(float)arg5*) → None :

C++ signature : void __init__(_object*,iplt::ReflectionList,std::string,std::string,float)

C++ interface

class **iplt::alg::LocalLatticeLineBin**

Public Functions

LocalLatticeLineBin(const ReflectionList & merge, const String & miobs, const String & msigiobs, Real zwin)

Real **GetAverage**(const ReflectionIndex & idx)

Real **GetWeightedAverage**(const ReflectionIndex & idx)

Real **GetStandardDeviation**(const ReflectionIndex & idx)

R-factors

Description

Python interface

class `iplt.alg.`**`RFriedelCalculator`**

`__init__`(*(object)arg1*, *(float)arg2*, *(float)arg3*, *(int)arg4*) → None :

C++ signature : void __init__(_object*,float,float,int)

class `iplt.alg.`**`RSymCalculator`**

`__init__`(*(object)arg1*, *(Symmetry)arg2*, *(float)arg3*, *(float)arg4*, *(float)arg5*, *(int)arg6*) → None :

C++ signature : void __init__(_object*,iplt::Symmetry,float,float,float,int)

class `iplt.alg.`**`RMergeCalculator`**

`SetSigmaRejectionLevel`(*(RMergeCalculator)arg1*, *(float)arg2*) → None :

C++ signature : void SetSigmaRejectionLevel(iplt::alg::RMergeCalculator {lvalue},float)

`__init__`(*(object)arg1*, *(ReflectionList)arg2*, *(str)arg3*, *(str)arg4*, *(float)arg5*, *(float)arg6*, *(float)arg7*, *(int)arg8*) → None :

C++ signature : void __init__(_object*,iplt::ReflectionList,std::string,std::string,float,float,float,int)

class `iplt.alg.`**`RMeasCalculator`**

`SetSigmaRejectionLevel`(*(RMergeCalculator)arg1*, *(float)arg2*) → None :

C++ signature : void SetSigmaRejectionLevel(iplt::alg::RMergeCalculator {lvalue},float)

__init__(*(object)arg1*, *(ReflectionList)arg2*, *(str)arg3*, *(str)arg4*, *(float)arg5*, *(float)arg6*, *(float)arg7*, *(int)arg8*) → None :

C++ signature : void __init__(_object*,iplt::ReflectionList,std::string,std::string,float,float,float,int)

class iplt.alg.**RFitCalculator**

__init__(*(object)arg1*, *(ReflectionList)arg2*, *(str)arg3*, *(str)arg4*, *(float)arg5*, *(float)arg6*, *(int)arg7*) → None :

C++ signature : void __init__(_object*,iplt::ReflectionList,std::string,std::string,float,float,int)

C++ interface

class **iplt::alg::RFriedelCalculator**

Public Functions

RFriedelCalculator(Real rlow, Real rhigh, int bcount)

ReflectionList **Apply**(const ReflectionList & data, const String & diobs, const String & dsigiobs)

class **iplt::alg::RSymCalculator**

Public Functions

RSymCalculator(const Symmetry & sym, Real zlim, Real rlow, Real rhigh, int bcount)

ReflectionList **Apply**(const ReflectionList & data, const String & diobs, const String & dsigiobs)

class **iplt::alg::RMergeCalculator**

Public Functions

RMergeCalculator(const ReflectionList & merge, const String & miobs, const String & msigiobs, Real zlim, Real rlow, Real rhigh, int bcount)

ReflectionList **Apply**(const ReflectionList & data, const String & diobs, const String & dsigiobs)

void **SetSigmaRejectionLevel**(Real r)

class **iplt::alg::RMeasCalculator**

Public Functions

RMeasCalculator(const ReflectionList & merge, const String & miobs, const String & msigiobs, Real zlim, Real rlow, Real rhigh, int bcount)

ReflectionList **Apply**(const ReflectionList & data, const String & diobs, const String & dsigiobs)

void **SetSigmaRejectionLevel**(Real r)

class **iplt::alg::RFitCalculator**

Public Functions

RFitCalculator(const ReflectionList & curve, const String & miobs, const String & msigiobs, Real rlow, Real rhigh, int bcount)

ReflectionList **Apply**(const ReflectionList & data, const String & diobs, const String & dsigiobs)

Index statistics

Python interface

class `iplt.alg.`**`RListIndexStat`**

`GetIndexList`(*(RListIndexStat)arg1*) → PointList :

C++ signature : ost::img::PointList GetIndexList(iplt::alg::RListIndexStat {lvalue})

`GetMaximumH`(*(RListIndexStat)arg1*) → int :

C++ signature : int GetMaximumH(iplt::alg::RListIndexStat {lvalue})

GetMaximumK(*(RListIndexStat)arg1*) → int :

C++ signature : int GetMaximumK(iplt::alg::RListIndexStat {lvalue})

GetMaximumZStar(*(RListIndexStat)arg1*) → float :

C++ signature : float GetMaximumZStar(iplt::alg::RListIndexStat {lvalue})

GetMinimumH(*(RListIndexStat)arg1*) → int :

C++ signature : int GetMinimumH(iplt::alg::RListIndexStat {lvalue})

GetMinimumK(*(RListIndexStat)arg1*) → int :

C++ signature : int GetMinimumK(iplt::alg::RListIndexStat {lvalue})

GetMinimumZStar(*(RListIndexStat)arg1*) → float :

C++ signature : float GetMinimumZStar(iplt::alg::RListIndexStat {lvalue})

__init__(*(object)arg1*) → None :

C++ signature : void __init__(_object*)

C++ interface

class **iplt::alg::RListIndexStat**

Public Functions

RListIndexStat()

~RListIndexStat()

void **Visit**(const ReflectionList &)

int **GetMinimumH**()

int **GetMinimumK**()

int **GetMaximumH**()

int **GetMaximumK**()

Real **GetMinimumZStar**()

Real **GetMaximumZStar**()

PointList **GetIndexList**()

Lattice line fitting

Python interface

class iplt.alg.**RlistLLFit**

GetCurve(*(RlistLLFit)arg1*) → ReflectionList :

C++ signature : iplt::ReflectionList GetCurve(iplt::alg::RlistLLFit {lvalue})

GetDiscretizedData(*(RlistLLFit)arg1*) → ReflectionList :

C++ signature : iplt::ReflectionList GetDiscretizedData(iplt::alg::RlistLLFit {lvalue})

__init__(*(object)arg1*, *(LLFitBase)arg2*, *(float)arg3*, *(int)arg4*[, *(int)arg5*[, *(str)arg6*[, *(str)arg7*]]]) → None :

C++ signature : void __init__(_object*,boost::shared_ptr<iplt::alg::LLFitBase>,float,unsigned int [,unsigned int [,std::string [,std::string]]])

C++ interface

class **iplt::alg::RlistLLFit**

Public Type

guessmode enum

Values:

- GUESSMODE_BINAVERAGE -
- GUESSMODE_CONST -
- GUESSMODE_RANDOM -

Public Functions

RlistLLFit(const LLFitBasePtr & llfit, Real max_resolution, unsigned int min_bincount, unsigned int bootstrapping_iterations = 0, const String & iobs_name = “”, const String & sigiobs_name = “”)

ReflectionList **Visit**(const ReflectionList & rlist)

ReflectionList **GetDiscretizedData**()

ReflectionList **GetCurve**()

Property filter

Python interface

class iplt.alg.**RListPropertyFilter**

__init__(*(object)arg1*, *(str)arg2*, *(float)arg3*, *(float)arg4*) → None :

C++ signature : void __init__(_object*,std::string,float,float)

__init__((object)arg1, (PropertyType)arg2, (float)arg3, (float)arg4) -> None :

C++ signature : void __init__(_object*,iplt::PropertyType,float,float)

C++ interface

class **iplt::alg::RListPropertyFilter**

Public Functions

RListPropertyFilter(const String & property, Real start, Real end)

RListPropertyFilter(const PropertyType & property_type, Real start, Real end)

ReflectionList **Visit**(const ReflectionList &)

Range filter

Python interface

class iplt.alg.**RListRangeFilter**

AddRange (*(RListRangeFilter)arg1*, *(ReflectionIndex)arg2*, *(ReflectionIndex)arg3*) → None :

C++ signature : void AddRange(iplt::alg::RListRangeFilter {lvalue},iplt::ReflectionIndex,iplt::ReflectionIndex)

ClearRanges (*(RListRangeFilter)arg1*) → None :

C++ signature : void ClearRanges(iplt::alg::RListRangeFilter {lvalue})

__init__ (*(object)arg1*) → None :

C++ signature : void __init__(_object*)

__init__((object)arg1, (ReflectionIndex)arg2, (ReflectionIndex)arg3) -> None :

C++ signature : void __init__(_object*,iplt::ReflectionIndex,iplt::ReflectionIndex)

C++ interface

class **iplt::alg::RListRangeFilter**

Public Functions

RListRangeFilter()

~RListRangeFilter()

RListRangeFilter(const ReflectionIndex & start, const ReflectionIndex & end)

void **AddRange**(const ReflectionIndex & start, const ReflectionIndex & end)

void **ClearRanges**()

ReflectionList **Visit**(const ReflectionList &)

Resolution filter

Python interface

class iplt.alg.**RListResolutionFilter**

__init__ (*(object)arg1*, *(float)arg2*, *(float)arg3*) → None :

C++ signature : void __init__(_object*,float,float)

C++ interface

class **iplt::alg::RListResolutionFilter**

Public Functions

RListResolutionFilter(Real start, Real end)

ReflectionList **Visit**(const ReflectionList &)

Split

Python interface

class iplt.alg.**RListSplitByProperty**

GetRListMap (*(RListSplitByProperty)arg1*) → PropertyReflectionListMap :

C++ signature : std::map<float, iplt::ReflectionList, std::less<float>, std::allocator<std::pair<float const, iplt::ReflectionList> > > GetRListMap(iplt::alg::RListSplitByProperty {lvalue})

__init__ (*(object)arg1*, *(str)arg2*) → None :

C++ signature : void __init__(_object*,std::string)

class iplt.alg.**RListSplitByIndex**

GetRListMap (*(RListSplitByIndex)arg1*) → IndexReflectionListMap :

C++ signature : std::map<ost::img::Point, iplt::ReflectionList, std::less<ost::img::Point>, std::allocator<std::pair<ost::img::Point const, iplt::ReflectionList> > > GetRListMap(iplt::alg::RListSplitByIndex {lvalue})

__init__ (*(object)arg1*) → None :

C++ signature : void __init__(_object*)

C++ interface

class **iplt::alg::RListSplitByProperty**

Public Type

typedef std::map< Real , ReflectionList > **PropertyReflectionListMap**

Public Functions

RListSplitByProperty(const String & property)

~RListSplitByProperty()

void **Visit**(const ReflectionList &)

PropertyReflectionListMap **GetRListMap**()

class **iplt::alg::RListSplitByIndex**

Public Type

typedef std::map< Point , ReflectionList > **IndexReflectionListMap**

Public Functions

RListSplitByIndex()

~RListSplitByIndex()

void **Visit**(const ReflectionList &)

IndexReflectionListMap **GetRListMap**()

Statistics

Python interface

class `iplt.alg.`**`RListStat`**

`GetIndexList`(*(RListStat)arg1*) → PointList :

C++ signature : ost::img::PointList GetIndexList(iplt::alg::RListStat {lvalue})

GetMaximum(*(RListStat)arg1*, *(str)arg2*) → float :

C++ signature : float GetMaximum(iplt::alg::RListStat {lvalue},std::string)

GetMaximumH(*(RListStat)arg1*) → int :

C++ signature : int GetMaximumH(iplt::alg::RListStat {lvalue})

GetMaximumK(*(RListStat)arg1*) → int :

C++ signature : int GetMaximumK(iplt::alg::RListStat {lvalue})

GetMaximumResolution(*(RListStat)arg1*) → float :

C++ signature : float GetMaximumResolution(iplt::alg::RListStat {lvalue})

GetMaximumZStar(*(RListStat)arg1*) → float :

C++ signature : float GetMaximumZStar(iplt::alg::RListStat {lvalue})

GetMean(*(RListStat)arg1*, *(str)arg2*) → float :

C++ signature : float GetMean(iplt::alg::RListStat {lvalue},std::string)

GetMeanH(*(RListStat)arg1*) → float :

C++ signature : float GetMeanH(iplt::alg::RListStat {lvalue})

GetMeanK(*(RListStat)arg1*) → float :

C++ signature : float GetMeanK(iplt::alg::RListStat {lvalue})

GetMeanResolution(*(RListStat)arg1*) → float :

C++ signature : float GetMeanResolution(iplt::alg::RListStat {lvalue})

GetMeanZStar(*(RListStat)arg1*) → float :

C++ signature : float GetMeanZStar(iplt::alg::RListStat {lvalue})

GetMinimum(*(RListStat)arg1*, *(str)arg2*) → float :

C++ signature : float GetMinimum(iplt::alg::RListStat {lvalue},std::string)

GetMinimumH(*(RListStat)arg1*) → int :

C++ signature : int GetMinimumH(iplt::alg::RListStat {lvalue})

GetMinimumK(*(RListStat)arg1*) → int :

C++ signature : int GetMinimumK(iplt::alg::RListStat {lvalue})

GetMinimumResolution(*(RListStat)arg1*) → float :

C++ signature : float GetMinimumResolution(iplt::alg::RListStat {lvalue})

GetMinimumZStar(*(RListStat)arg1*) → float :

C++ signature : float GetMinimumZStar(iplt::alg::RListStat {lvalue})

__init__(*(object)arg1*) → None :

C++ signature : void __init__(_object*)

C++ interface

class **iplt::alg::RListStat**

Public Functions

RListStat()

~RListStat()

void **Visit**(const ReflectionList &)

int **GetMinimumH**()

int **GetMinimumK**()

int **GetMaximumH**()

int **GetMaximumK**()

Real **GetMinimumZStar**()

Real **GetMaximumZStar**()

Real **GetMinimumResolution**()

Real **GetMaximumResolution**()

Real **GetMeanH**()

Real **GetMeanK**()

Real **GetMeanZStar**()

Real **GetMeanResolution**()

Real **GetMinimum**(String name)

Real **GetMaximum**(String name)

Real **GetMean**(String name)

PointList **GetIndexList**()

Tilt geometry refinement

Python interface

class `iplt.alg.`**`RListTiltGeometryRefine`**

`GetMaxIter`(*(RListTiltGeometryRefine)arg1*) → int :

C++ signature : unsigned int GetMaxIter(iplt::alg::RListTiltGeometryRefine {lvalue})

`GetRMerge`(*(RListTiltGeometryRefine)arg1*) → float :

C++ signature : float GetRMerge(iplt::alg::RListTiltGeometryRefine {lvalue})

`GetTiltGeometry`(*(RListTiltGeometryRefine)arg1*) → LatticeTiltGeometry :

C++ signature : iplt::LatticeTiltGeometry GetTiltGeometry(iplt::alg::RListTiltGeometryRefine {lvalue})

`GetTolerance`(*(RListTiltGeometryRefine)arg1*) → float :

C++ signature : float GetTolerance(iplt::alg::RListTiltGeometryRefine {lvalue})

GetWeightedRMerge(*(RListTiltGeometryRefine)arg1*) → float :

C++ signature : float GetWeightedRMerge(iplt::alg::RListTiltGeometryRefine {lvalue})

SetMaxIter(*(RListTiltGeometryRefine)arg1*, *(int)arg2*) → None :

C++ signature : void SetMaxIter(iplt::alg::RListTiltGeometryRefine {lvalue},unsigned int)

SetTiltGeometry(*(RListTiltGeometryRefine)arg1*, *(LatticeTiltGeometry)arg2*) → None :

C++ signature : void SetTiltGeometry(iplt::alg::RListTiltGeometryRefine {lvalue},iplt::LatticeTiltGeometry)

SetTolerance(*(RListTiltGeometryRefine)arg1*, *(float)arg2*) → None :

C++ signature : void SetTolerance(iplt::alg::RListTiltGeometryRefine {lvalue},float)

__init__(*(object)arg1*, *(ReflectionList)arg2*, *(str)arg3*, *(str)arg4*, *(float)arg5*, *(float)arg6*, *(float)arg7*, *(int)arg8*[, *(str)arg9*[, *(str)arg10*]]) → None :

C++ signature : void __init__(_object*,iplt::ReflectionList,std::string,std::string,float,float,float,int [,std::string [,std::string]])

C++ interface

class **iplt::alg::RListTiltGeometryRefine**

Public Functions

RListTiltGeometryRefine(const ReflectionList & reference, const String & miobs, const String & msigiobs, Real zlim, Real rlow, Real rhigh, int bcount, const String & alterate_h = “”, const String & alterate_k = “”)

~RListTiltGeometryRefine()

ReflectionList **Visit**(const ReflectionList &)

void **SetTiltGeometry**(const LatticeTiltGeometry & tg)

LatticeTiltGeometry **GetTiltGeometry**()

unsigned int **GetMaxIter**()

void **SetMaxIter**(unsigned int maxiter)

Real **GetTolerance**()

void **SetTolerance**(Real tolerance)

Real **GetRMerge**()

Real **GetWeightedRMerge**()

Weighted binning

Python interface

iplt.alg.**RListWBin**(*(ReflectionList)arg1*, *(int)arg2*, *(float)arg3*, *(float)arg4*, *(str)arg5*, *(str)arg6*)
→ WeightedBin :

C++ signature : iplt::alg::WeightedBin RListWBin(iplt::ReflectionList,int,float,float,std::string,std::string)

C++ interface

WeightedBin DLLEXPORT_IPLT_ALG **RListWBin**(const ReflectionList & rlist, int bcount, Real rlow, Real rhigh, const String & m_i, const String & m_sigi)

Symmetrize

Python interface

class iplt.alg.**Symmetrize**

__init__(*(object)arg1*, *(str)arg2*) → None :

C++ signature : void __init__(_object*,std::string)

__init__((object)arg1, (Symmetry)arg2) -> None :

C++ signature : void __init__(_object*,iplt::Symmetry)

C++ interface

class **iplt::alg::Symmetrize**

Public Functions

Symmetrize(const Symmetry & s)

Symmetrize(String s)

ReflectionList **Visit**(const ReflectionList &)

Tilter

Python interface

class `iplt.alg.`**`Tilter`**

`__init__`(*(object)arg1* [, *(LatticeTiltGeometry)arg2* [, *(str)arg3* [, *(str)arg4*]]]) → None :

C++ signature : void __init__(_object* [,iplt::LatticeTiltGeometry [,std::string [,std::string]]])

C++ interface

class **iplt::alg::Tilter**

Public Functions

Tilter()

Tilter(const LatticeTiltGeometry & tg, const String & alterate_h = “”, const String & alternate_k = “”)

ReflectionList **Visit**(const ReflectionList &)

Unique

Python interface

class iplt.alg.**Unique**

__init__ (*(object)arg1*) → None :

C++ signature : void __init__(_object*)

__init__((object)arg1, (Unique)arg2) -> None :

C++ signature : void __init__(_object*,iplt::alg::Unique)

C++ interface

class **iplt::alg::Unique**

Public Functions

Unique()

ReflectionList **Visit**(const ReflectionList &)

Reduce

Python interface

class iplt.alg.**Reduce**

__init__ (*(object)arg1*, *(str)arg2*) → None :

C++ signature : void __init__(_object*,std::string)

__init__((object)arg1, (Reduce)arg2) -> None :

C++ signature : void __init__(_object*,iplt::alg::Reduce)

C++ interface

class **iplt::alg::Reduce**

Public Functions

Reduce(const Symmetry & s)

Reduce(String s)

ReflectionList **Visit**(const ReflectionList &)

3.2.3 Anisotropic scaling

Description

Anisotropic scaling with three parameters against a reference dataset.

$$log(\frac{I(h,k)}{I_0}) = A + B1h^2 + B2k^2$$

Python interface

class `iplt.alg.`**`AnisoScaling`**

`Add`(*(AnisoScaling)arg1*, *(float)arg2*, *(float)arg3*, *(float)arg4*, *(float)arg5*) → None :

C++ signature : void Add(iplt::alg::AnisoScaling {lvalue},float,float,float,float)

`Apply`(*(AnisoScaling)arg1*) → None :

C++ signature : void Apply(iplt::alg::AnisoScaling {lvalue})

`Estimate`(*(AnisoScaling)arg1*, *(float)arg2*, *(float)arg3*) → float :

C++ signature : float Estimate(iplt::alg::AnisoScaling {lvalue},float,float)

`GetA`(*(AnisoScaling)arg1*) → float :

C++ signature : float GetA(iplt::alg::AnisoScaling {lvalue})

`GetB1`(*(AnisoScaling)arg1*) → float :

C++ signature : float GetB1(iplt::alg::AnisoScaling {lvalue})

`GetB2`(*(AnisoScaling)arg1*) → float :

C++ signature : float GetB2(iplt::alg::AnisoScaling {lvalue})

`__init__`(*(object)arg1*) → None :

C++ signature : void __init__(_object*)

C++ interface

class **iplt::alg::AnisoScaling**

Public Type

typedef std::vector< *FitEntry* > **FitEntryList**

Public Functions

AnisoScaling()

void **Add**(Real x, Real y1, Real y2, Real w)

void **Apply**()

Real **Estimate**(Real y1, Real y2)

Real **GetA**()

Real **GetB1**()

Real **GetB2**()

class **FitEntry**

Public Members

Real **x**

Real **y1**

Real **y2**

Real **w**

3.2.4 Create Image from reflection list

CreateImageFromReflectionList creates an image from a ReflectionList.

Python interface

`iplt.alg.`**`CreateImageFromReflectionList`**(*(ReflectionList)arg1*[, *(int)arg2*[, *(int)arg3*]])
→ ImageHandle :

C++ signature : ost::img::ImageHandle CreateImageFromReflectionList(iplt::ReflectionList [,unsigned int [,unsigned int]])

C++ Interface

ImageHandle DLLEXPORT_IPLT_ALG **CreateImageFromReflectionList**(const :: iplt::ReflectionList & in, unsigned int nr_of_unit_cells = 1, unsigned int dimensions = 2)

3.2.5 Exponential fit

Python interface

class iplt.alg.**ExpFit**

Add(*(ExpFit)arg1*, *(float)arg2*, *(float)arg3*) → None :

C++ signature : void Add(iplt::alg::ExpFit {lvalue},float,float)

Add((ExpFit)arg1, (float)arg2, (float)arg3, (float)arg4) -> None :

C++ signature : void Add(iplt::alg::ExpFit {lvalue},float,float,float)

Apply(*(ExpFit)arg1*) → None :

C++ signature : void Apply(iplt::alg::ExpFit {lvalue})

Estimate(*(ExpFit)arg1*, *(float)arg2*) → tuple :

C++ signature : boost::python::tuple Estimate(iplt::alg::ExpFit {lvalue},float)

GetB(*(ExpFit)arg1*) → float :

C++ signature : float GetB(iplt::alg::ExpFit {lvalue})

GetBsigB(*(ExpFit)arg1*) → tuple :

C++ signature : boost::python::tuple GetBsigB(iplt::alg::ExpFit)

GetC(*(ExpFit)arg1*) → float :

C++ signature : float GetC(iplt::alg::ExpFit {lvalue})

GetChi(*(ExpFit)arg1*) → float :

C++ signature : float GetChi(iplt::alg::ExpFit {lvalue})

GetCsigC(*(ExpFit)arg1*) → tuple :

C++ signature : boost::python::tuple GetCsigC(iplt::alg::ExpFit)

GetS(*(ExpFit)arg1*) → float :

C++ signature : float GetS(iplt::alg::ExpFit {lvalue})

GetSsigS(*(ExpFit)arg1*) → tuple :

C++ signature : boost::python::tuple GetSsigS(iplt::alg::ExpFit)

SetLimits(*(ExpFit)arg1*, *(float)arg2*, *(float)arg3*) → None :

C++ signature : void SetLimits(iplt::alg::ExpFit {lvalue},float,float)

SetMaxIter(*(ExpFit)arg1*, *(int)arg2*) → None :

C++ signature : void SetMaxIter(iplt::alg::ExpFit {lvalue},unsigned int)

__init__(*(object)arg1*[, *(bool)arg2*]) → None :

C++ signature : void __init__(_object* [,bool])

C++ interface

class **iplt::alg::ExpFit**

Public Functions

ExpFit(bool zero_offset = false)

void **Add**(Real x, Real y)

void **Add**(Real x, Real y, Real w)

Real **GetS**()

std::pair< Real , Real > **GetSsigS**()

Real **GetB**()

std::pair< Real , Real > **GetBsigB**()

Real **GetC**()

std::pair< Real , Real > **GetCsigC**()

Real **GetChi**()

void **SetMaxIter**(unsigned int m)

void **SetLimits**(Real l1, Real l2)

void **Apply**()

std::pair< Real , Real > **Estimate**(Real x)

3.2.6 Explicit 2D gaussian fit

Python interface

class iplt.alg.**FitExplGauss2D**

Fit (*(FitExplGauss2D)arg1*, *(ExplGauss2DEntryList)arg2*) → None :

C++ signature : void Fit(iplt::alg::FitExplGauss2D {lvalue},std::vector<iplt::alg::ExplGauss2DEntry, std::allocator<iplt::alg::ExplGauss2DEntry> >)

GetAbsQuality (*(FitExplGauss2D)arg1*) → float :

C++ signature : double GetAbsQuality(iplt::alg::FitExplGauss2D {lvalue})

GetChi (*(FitExplGauss2D)arg1*) → float :

C++ signature : double GetChi(iplt::alg::FitExplGauss2D {lvalue})

GetGOF (*(FitExplGauss2D)arg1*) → float :

C++ signature : double GetGOF(iplt::alg::FitExplGauss2D {lvalue})

GetIterationCount (*(FitExplGauss2D)arg1*) → int :

C++ signature : int GetIterationCount(iplt::alg::FitExplGauss2D {lvalue})

GetQuality (*(FitExplGauss2D)arg1*) → float :

C++ signature : double GetQuality(iplt::alg::FitExplGauss2D {lvalue})

GetRelQuality (*(FitExplGauss2D)arg1*) → float :

C++ signature : double GetRelQuality(iplt::alg::FitExplGauss2D {lvalue})

GetSigAmp (*(FitExplGauss2D)arg1*) → float :

C++ signature : double GetSigAmp(iplt::alg::FitExplGauss2D {lvalue})

SetLim (*(FitExplGauss2D)arg1*, *(float)arg2*, *(float)arg3*) → None :

C++ signature : void SetLim(iplt::alg::FitExplGauss2D {lvalue},double,double)

SetMaxIter (*(FitExplGauss2D)arg1*, *(int)arg2*) → None :

C++ signature : void SetMaxIter(iplt::alg::FitExplGauss2D {lvalue},int)

SetSigma(*(FitExplGauss2D)arg1*, *(float)arg2*) → None :

C++ signature : void SetSigma(iplt::alg::FitExplGauss2D {lvalue},double)

__init__(*(object)arg1*[, *(float)arg2*[, *(float)arg3*[, *(float)arg4*[, *(float)arg5*[, *(float)arg6*[, *(float)arg7*[, *(float)arg8*[, *(float)arg9*[, *(float)arg10*]]]]]]]]]) → None :

C++ signature : void __init__(_object* [,float [,float [,float [,float [,float [,float [,float [,float [,float]]]]]]]]])

C++ interface

class **iplt::alg::FitExplGauss2D**

2D Gaussian explicit fitting algorithm

Public Type

Anonymous enum

Values:

- ID_A = =0 -
- ID_BX -
- ID_BY -
- ID_UX -
- ID_UY -
- ID_C -
- ID_W -
- ID_PX -
- ID_PY -

Public Functions

FitExplGauss2D(double A = 1.0, double Bx = 1.0, double By = 1.0, double C = 0.0, double ux = 0.0, double uy = 0.0, double w = 0.0, double Px = 0.0, double Py = 0.0)

void **SetSigma**(double s)

void **SetMaxIter**(int n)

void **SetLim**(double l1, double l2)

double **GetChi**()

double **GetGOF**()

double **GetQuality**()

double **GetAbsQuality**()

double **GetRelQuality**()

int **GetIterationCount**()

double **GetSigAmp**()

void **Fit**(const ExplGauss2DEntryList & entry_list)

3.2.7 GSL wrapper

Interface

class **iplt::alg::ConstGSLVector**

Public Functions

ConstGSLVector(const gsl_vector * v)

~ConstGSLVector()

double **Get**(size_t i)

operator const gsl_vector *()

class **iplt::alg::GSLVector**

Public Functions

GSLVector(size_t n)

GSLVector(gsl_vector * v)

~GSLVector()

void **Set**(size_t i, double val)

double **Get**(size_t i)

std::vector< double > **GetVector**()

operator gsl_vector *()

class **iplt::alg::ConstGSLMatrix**

Public Functions

ConstGSLMatrix(const gsl_matrix * v)

~ConstGSLMatrix()

double **Get**(size_t i, size_t j)

operator const gsl_matrix *()

class **iplt::alg::GSLMatrix**

Public Functions

GSLMatrix(size_t m, size_t n)

GSLMatrix(gsl_matrix * v)

~GSLMatrix()

void **Set**(size_t i, size_t j, double val)

double **Get**(size_t i, size_t j)

operator gsl_matrix *()

class **iplt::alg::GSLMultiminFunction**

Public Functions

GSLMultiminFunction(size_t n)

~GSLMultiminFunction()

double **Func**(const *ConstGSLVector* & x)

class **iplt::alg::GSLMultiminFunctionWithDerivatives**

Public Functions

GSLMultiminFunctionWithDerivatives(size_t n, size_t p)

~GSLMultiminFunctionWithDerivatives()

int **Function**(const *ConstGSLVector* & x, *GSLVector* & f)

int **Derivatives**(const *ConstGSLVector* & x, *GSLMatrix* & J)

int **FunctionAndDerivatives**(const *ConstGSLVector* & x, *GSLVector* & f, *GSLMatrix* & J)

void **Bracket**(*GSLVector* & x, *GSLVector* & dx, *GSLVector* & f, *GSLMatrix* & J)

void **SetGuess**(*GSLVector* & x)

class **iplt::alg::GSLNMSimplex2**

Public Functions

GSLNMSimplex2(gsl_multimin_function * func, size_t maxiter = 100, Real tolerance = 0.001)

int **Iterate**()

void **SetMaxIter**(size_t maxiter)

void **SetTolerance**(Real tolerance)

void **SetX**(size_t i, double x)

void **SetStepSize**(size_t i, double x)

double **GetX**(size_t i)

class **iplt::alg::GSLFDFSolverLMSDER**

Public Functions

GSLFDFSolverLMSDER(*GSLMultiminFunctionWithDerivatives* * func, size_t maxiter = 100, Real limit1 = 0.001, Real limit2 = 0.0001)

int **Iterate**()

void **SetMaxIter**(size_t maxiter)

void **SetIterationLimits**(Real, Real)

void **SetX**(size_t i, double x)

double **GetX**(size_t i)

std::vector< double > **GetXVector**()

double **GetCovar**(size_t i, size_t j)

double **GetChi**()

double **GetGOF**()

unsigned int **GetNumIterations**()

3.2.8 Lattice Point

Python interface

class iplt.alg.**LatticePoint**

CalcAveBG (*(LatticePoint)arg1*, *(bool)arg2*) → None :

C++ signature : void CalcAveBG(iplt::alg::LatticePoint {lvalue},bool)

GetAveBG (*(LatticePoint)arg1*) → float :

C++ signature : float GetAveBG(iplt::alg::LatticePoint {lvalue})

GetAveSigBG (*(LatticePoint)arg1*) → float :

C++ signature : float GetAveSigBG(iplt::alg::LatticePoint {lvalue})

GetFit (*(LatticePoint)arg1*) → FitGauss2D :

C++ signature : ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::FitGauss2DBase> GetFit(iplt::alg::LatticePoint {lvalue})

GetICBG (*(LatticePoint)arg1*) → float :

C++ signature : float GetICBG(iplt::alg::LatticePoint {lvalue})

GetPI (*(LatticePoint)arg1*) → PeakIntegration :

C++ signature : iplt::alg::PeakIntegration GetPI(iplt::alg::LatticePoint {lvalue})

GetPosition (*(LatticePoint)arg1*) → object :

C++ signature : geom::Vec2 GetPosition(iplt::alg::LatticePoint {lvalue})

GetRegion (*(LatticePoint)arg1*) → Extent :

C++ signature : ost::img::Extent GetRegion(iplt::alg::LatticePoint {lvalue})

GetStat (*(LatticePoint)arg1*) → Stat :

C++ signature : ost::img::image_state::ImageStateNonModAlgorithm<ost::img::alg::StatBase> GetStat(iplt::alg::LatticePoint {lvalue})

MemSize (*(LatticePoint)arg1*) → int :

C++ signature : long MemSize(iplt::alg::LatticePoint {lvalue})

SetAveBG (*(LatticePoint)arg1*, *(float)arg2*) → None :

C++ signature : void SetAveBG(iplt::alg::LatticePoint {lvalue},float)

SetAveSigBG (*(LatticePoint)arg1*, *(float)arg2*) → None :

C++ signature : void SetAveSigBG(iplt::alg::LatticePoint {lvalue},float)

SetFit (*(LatticePoint)arg1*, *(FitGauss2D)arg2*) → None :

C++ signature : void SetFit(iplt::alg::LatticePoint {lvalue},ost::img::image_state::ImageStateNonModAlgorithm<iplt:

SetICBG (*(LatticePoint)arg1*, *(float)arg2*) → None :

C++ signature : void SetICBG(iplt::alg::LatticePoint {lvalue},float)

SetPI (*(LatticePoint)arg1*, *(PeakIntegration)arg2*) → None :

C++ signature : void SetPI(iplt::alg::LatticePoint {lvalue},iplt::alg::PeakIntegration)

SetPosition (*(LatticePoint)arg1*, *(object)arg2*) → None :

C++ signature : void SetPosition(iplt::alg::LatticePoint {lvalue},geom::Vec2)

SetRegion (*(LatticePoint)arg1*, *(Extent)arg2*) → None :

C++ signature : void SetRegion(iplt::alg::LatticePoint {lvalue},ost::img::Extent)

SetStat (*(LatticePoint)arg1*, *(Stat)arg2*) → None :

C++ signature : void SetStat(iplt::alg::LatticePoint {lvalue},ost::img::image_state::ImageStateNonModAlgorithm<ost

__init__ (*(object)arg1*[, *(object)arg2*[, *(Extent)arg3*[, *(Stat)arg4*[, *(FitGauss2D)arg5*[, *(PeakIntegration)arg6*[, *(float)arg7*[, *(float)arg8*[, *(float)arg9*]]]]]]]]) → None :

C++ signature : void __init__(_object* [,geom::Vec2 [,ost::img::Extent [,ost::img::image_state::ImageStateNonModAlgorithm<ost::img::alg::StatBase> [,ost::img::image_state::ImageStateNonModAlgorithm<iplt::alg::FitGauss2DBase> [,iplt::alg::PeakIntegration [,float [,float [,float]]]]]]]])

C++ Interface

class **iplt::alg::LatticePoint**

Encapsulate various aspects of a lattice point.

This class allows storage of the various parameters that are connected to a particular lattice point:

Note that the index is not stored here, as this is only applicable together with a *Lattice*, such as the case for LatticePointProxyter in connection with a LatticePointList;

Public Functions

LatticePoint(const geom::Vec2 & pos = Vec2(), const ost::img::Extent & region = Extent (), const Stat & stat = Stat(), const *FitGauss2D* & fit = *FitGauss2D* (), const *PeakIntegration* & pi = *PeakIntegration* (), Real ic_bg = 0.0, Real ave_bg = 0.0, Real ave_sigbg = 0.0)

geom::Vec2 **GetPosition**()

void **SetPosition**(const geom::Vec2 & p)

FitGauss2D **GetFit**()

void **SetFit**(const *FitGauss2D* & fit)

ost::img::Extent **GetRegion**()

void **SetRegion**(const ost::img::Extent & h)

Stat **GetStat**()

void **SetStat**(const Stat & stat)

PeakIntegration **GetPI**()

void **SetPI**(const *PeakIntegration* & pi)

Real **GetICBG**()

void **SetICBG**(Real icbg)

Real **GetAveBG**()

void **SetAveBG**(Real abg)

Real **GetAveSigBG**()

void **SetAveSigBG**(Real asbg)

void **CalcAveBG**(bool smart)

operator ost::img::Point()

long **MemSize**()

3.2.9 Linear fit

Given a number of datapoints (x,y), calculates the parameters 'm' and 'b' that fit best (in the least squares sense) the following model:

$$y = mx + b$$

Datapoints may have weights assigned to them for both x and y axes. The weight is the reciprocal of the variance. The returned goodness of fit is the square root of chi^2.

Python interface

class `iplt.alg.`**`LinearFit`**

`AddDatapoint`(*(LinearFit)arg1*, *(float)arg2*, *(float)arg3*) → None :

C++ signature : void AddDatapoint(iplt::alg::LinearFit {lvalue},double,double)

AddDatapoint((LinearFit)arg1, (float)arg2, (float)arg3, (float)arg4) -> None :

C++ signature : void AddDatapoint(iplt::alg::LinearFit {lvalue},double,double,double)

AddDatapoint((LinearFit)arg1, (float)arg2, (float)arg3, (float)arg4, (float)arg5) -> None :

C++ signature : void AddDatapoint(iplt::alg::LinearFit {lvalue},double,double,double,double)

`Apply`(*(LinearFit)arg1*) → tuple :

C++ signature : boost::python::tuple Apply(iplt::alg::LinearFit {lvalue})

`Clear`(*(LinearFit)arg1*) → None :

C++ signature : void Clear(iplt::alg::LinearFit {lvalue})

`Estimate`(*(LinearFit)arg1*, *(float)arg2*) → tuple :

C++ signature : boost::python::tuple Estimate(iplt::alg::LinearFit {lvalue},double)

`ForceOrigin`(*(LinearFit)arg1*, *(bool)arg2*) → None :

C++ signature : void ForceOrigin(iplt::alg::LinearFit {lvalue},bool)

`GetGOF`(*(LinearFit)arg1*) → float :

C++ signature : double GetGOF(iplt::alg::LinearFit {lvalue})

`GetOffset`(*(LinearFit)arg1*) → float :

C++ signature : double GetOffset(iplt::alg::LinearFit {lvalue})

`GetOffsetVariance`(*(LinearFit)arg1*) → float :

C++ signature : double GetOffsetVariance(iplt::alg::LinearFit {lvalue})

`GetQ`(*(LinearFit)arg1*) → float :

C++ signature : double GetQ(iplt::alg::LinearFit {lvalue})

GetRSQ(*(LinearFit)arg1*) → float :

C++ signature : double GetRSQ(iplt::alg::LinearFit {lvalue})

GetScale(*(LinearFit)arg1*) → float :

C++ signature : double GetScale(iplt::alg::LinearFit {lvalue})

GetScaleVariance(*(LinearFit)arg1*) → float :

C++ signature : double GetScaleVariance(iplt::alg::LinearFit {lvalue})

GetWGOF(*(LinearFit)arg1*) → float :

C++ signature : double GetWGOF(iplt::alg::LinearFit {lvalue})

PointCount(*(LinearFit)arg1*) → int :

C++ signature : int PointCount(iplt::alg::LinearFit {lvalue})

__init__(*(object)arg1*) → None :

C++ signature : void __init__(_object*)

C++ interface

class **iplt::alg::LinearFit**

Helper class for linear least squares fit.

Public Functions

LinearFit()

ctor

void **ForceOrigin**(bool f)

force linear fit to have an offset of zero

void **AddDatapoint**(double x, double y)

add datapoint, using default weights of 1.0

void **AddDatapoint**(double x, double y, double wy)

add datapoint and weight for y, using default x weight of 1.0

void **AddDatapoint**(double x, double wx, double y, double wy)

add datapoint and weight for both coordinates

std::pair< double, double > **Apply**()

Apply fit with given datapoints, return result as pair, with 'm' as first and 'b' as second

std::pair< double, double > **Estimate**(double x)

int **PointCount**()

return number points

void **Clear**()

clear points

double **GetScale**()

double **GetOffset**()

double **GetGOF**()

double **GetWGOF**()

double **GetQ**()

double **GetRSQ**()

double **GetScaleVariance**()

double **GetOffsetVariance**()

3.2.10 Lattice line prediciction

Python interface

class iplt.alg.**LLPredict**

Get(*(LLPredict)arg1*, *(ReflectionIndex)arg2*) → complex :

C++ signature : std::complex<float> Get(iplt::alg::LLPredict {lvalue},iplt::ReflectionIndex)

__init__(*(object)arg1*, *(ConstImageHandle)arg2*) → None :

C++ signature : void __init__(_object*,ost::img::ConstImageHandle)

C++ interface

class **iplt::alg::LLPredict**

Predict lattice line values.

Given a real-space volume (ie 3D density), predict values for for a particular lattice line {h,k}, at reciprocal height z*.

The internal calculation uses an explicit fourier summation, whose exponential contains the multiplication of 'zstar' with 'z' (the reciprocal distance with the spatial one), where zstar is given directly, and 'z' is generated from the index and the spatial sampling.

If zstar is given in reciprocal angstrom, for example, the spatial sampling should be given in angstrom.

Public Functions

LLPredict(const ost::img::ConstImageHandle & ih)

initialize with 3D volume

The pixel sampling must be properly set to ensure correct conversion from frequency to fractional index

Complex **Get**(const ReflectionIndex & ri)

get prediction for lattice line hk and zstar

ost::img::ImageHandle **GetRef**()

debug interface, do not use

3.2.11 Logarithmic fit

Python interface

class iplt.alg.**LogFit**

Add(*(LogFit)arg1*, *(float)arg2*, *(float)arg3*) → None :

C++ signature : void Add(iplt::alg::LogFit {lvalue},float,float)

Add((LogFit)arg1, (float)arg2, (float)arg3, (float)arg4) -> None :

C++ signature : void Add(iplt::alg::LogFit {lvalue},float,float,float)

Apply(*(LogFit)arg1*) → None :

C++ signature : void Apply(iplt::alg::LogFit {lvalue})

GetB(*(LogFit)arg1*) → float :

C++ signature : float GetB(iplt::alg::LogFit {lvalue})

GetC(*(LogFit)arg1*) → float :

C++ signature : float GetC(iplt::alg::LogFit {lvalue})

GetChi(*(LogFit)arg1*) → float :

C++ signature : float GetChi(iplt::alg::LogFit {lvalue})

GetS(*(LogFit)arg1*) → float :

C++ signature : float GetS(iplt::alg::LogFit {lvalue})

SetLimits(*(LogFit)arg1*, *(float)arg2*, *(float)arg3*) → None :

C++ signature : void SetLimits(iplt::alg::LogFit {lvalue},float,float)

SetMaxIter(*(LogFit)arg1*, *(int)arg2*) → None :

C++ signature : void SetMaxIter(iplt::alg::LogFit {lvalue},unsigned int)

__init__(*(object)arg1*[, *(bool)arg2*]) → None :

C++ signature : void __init__(_object* [,bool])

C++ interface

class **iplt::alg::LogFit**

Public Functions

LogFit(bool zero_offset = false)

void **Add**(Real x, Real y)

void **Add**(Real x, Real y, Real w)

Real **GetS**()

Real **GetB**()

Real **GetC**()

Real **GetChi**()

void **SetMaxIter**(unsigned int m)

void **SetLimits**(Real l1, Real l2)

void **Apply**()

3.2.12 Multi exponential fit

Python interface

class iplt.alg.**NExpFit**

Add(*(NExpFit)arg1*, *(float)arg2*, *(float)arg3*) → None :

C++ signature : void Add(iplt::alg::NExpFit {lvalue},float,float)

Add((NExpFit)arg1, (float)arg2, (float)arg3, (float)arg4) -> None :

C++ signature : void Add(iplt::alg::NExpFit {lvalue},float,float,float)

Apply(*(NExpFit)arg1*) → None :

C++ signature : void Apply(iplt::alg::NExpFit {lvalue})

GetBVec(*(NExpFit)arg1*) → list :

C++ signature : boost::python::list GetBVec(iplt::alg::NExpFit {lvalue})

GetC(*(NExpFit)arg1*) → float :

C++ signature : float GetC(iplt::alg::NExpFit {lvalue})

GetChi(*(NExpFit)arg1*) → float :

C++ signature : float GetChi(iplt::alg::NExpFit {lvalue})

GetSVec(*(NExpFit)arg1*) → list :

C++ signature : boost::python::list GetSVec(iplt::alg::NExpFit {lvalue})

__init__(*(object)arg1*[, *(int)arg2*[, *(bool)arg3*]]) → None :

C++ signature : void __init__(_object* [,unsigned int [,bool]])

C++ interface

class **iplt::alg::NExpFit**

Public Functions

NExpFit(unsigned int num = 1, bool zero_offset = false)

~NExpFit()

void **Add**(Real x, Real y)

void **Add**(Real x, Real y, Real w)

std::vector< Real > **GetSVec**()

std::vector< Real > **GetBVec**()

Real **GetC**()

Real **GetChi**()

void **Apply**()

3.2.13 Numerical utilities

C++ Interface

int **max**(int a, int b)

void **swap**(double * a, double * b)

double **sign**(double a, double b)

void **shft3**(double * a, double * b, double * c, double d)

DLLEXPORT_IPLT_ALG void **mnbrak**(double * ax, double * bx, double * cx, double * fa, double * fb, double * fc, double(*)(double, void *) func, void * func_params)

DLLEXPORT_IPLT_ALG double **brent**(double ax, double bx, double cx, double(*)(double, void *) func, double tol, double * xmin, void * func_params)

double **zbrent**(double(*)(double, void *) func, double x1, double x2, double tol, void * func_params)

DLLEXPORT_IPLT_ALG geom::Vec2 **polint**(const std::vector< geom::Vec2 > & points, Real x)

3.2.14 Reflection list resolution shell binning

Describtion

In essence builds a histogramm of amplitudes in resolution shells, using either the reciprocal of the resolution or the reciprocal of the squared resolution.

Python interface

class iplt.alg.**ResolutionBin**

Add(*(ResolutionBin)arg1*, *(float)arg2*, *(float)arg3*, *(float)arg4*) → None :

C++ signature : void Add(iplt::alg::ResolutionBin {lvalue},float,float,float)

GetBinAverage(*(ResolutionBin)arg1*, *(int)arg2*) → float :

C++ signature : float GetBinAverage(iplt::alg::ResolutionBin {lvalue},int)

GetBinCount(*(ResolutionBin)arg1*) → int :

C++ signature : int GetBinCount(iplt::alg::ResolutionBin {lvalue})

GetBinResolution(*(ResolutionBin)arg1*, *(int)arg2*) → float :

C++ signature : float GetBinResolution(iplt::alg::ResolutionBin {lvalue},int)

GetBinStdDev(*(ResolutionBin)arg1*, *(int)arg2*) → float :

C++ signature : float GetBinStdDev(iplt::alg::ResolutionBin {lvalue},int)

__init__(*(object)arg1*) → None :

C++ signature : void __init__(_object*)

__init__((object)arg1, (int)arg2, (float)arg3, (float)arg4 [, (bool)arg5]) -> None :

C++ signature : void __init__(_object*,int,float,float [,bool])

C++ interface

class **iplt::alg::ResolutionBin**

Public Functions

ResolutionBin()

default ctor with count=1, lower limit=1 and higher limit=2

ResolutionBin(int bincount, Real lim_low, Real lim_hi, bool use_rsq = false)

Initialize with number of bins and resolution bounds.

bincount: number of bins to use reslow: 'lower' resolution limit in A reshi: ' higher' resolution limit in A

optional boolean: indicate wether resolutions are squared before binning their reciprocal values, which is for instance used by the wilson fit facility

void **Add**(Real resol, Real val, Real w = 1.0)

Add value at resolution (in A)

Real **GetBinAverage**(int n)

Retrieve average value from specified bin.

ave=sum[value_i*weight_i]/sum[weight_i]

Real **GetBinStdDev**(int n)

Retrieve std dev from specified bin.

sdev^2 = sum[value_i*weight_i-<ave>]/sum[weight_i]

int **GetBinCount**()

Return number of bins.

Real **GetBinResolution**(int n)

Retrieve lower resolution bound of this bin.

class **BinEntry**

Public Members

Real **value**

Real **weight**

3.2.15 Lattice refinement

Python interface

`iplt.alg.`**`RefineLattice`**(*(LatticePointList)arg1*[, *(bool)arg2*[, *(bool)arg3*]]) → Lattice :

C++ signature : iplt::Lattice RefineLattice(iplt::alg::LatticePointList [,bool [,bool]])

C++ interface

DLLEXPORT_IPLT_ALG *Lattice* **RefineLattice**(const LatticePointList & lpl, bool fit_distortion = true, bool fit_origin = true)

Refine lattice based on existing lattice point list.

Uses the coordinate offsets of each lattice point to refine the lattice using a least square minimization:

For each lattice point, the integer indices u and v are known, as well as the fractional position on the image. The sought lattice, defined by the vectors P,Q,O, a barrel distortion constant kb, and a spiral distortion ks gives the solution to each point as follows:

L_ix = (u_i * Px + v_i * Qx) (1 + kb * | (u_i * P + v_i * Q) | ^2)

L_iy = (u_i * Py + v_i * Qy) (1 + kb * | (u_i * P + v_i * Q) | ^2)

If the distortion can be ignored, ie by forcing kb and ks to zero, then this equation simplifies to one with only linear terms, and a linear least squares procdure is utilized. Otherwise, a non-linear least squares procedure must be employed.

3.2.16 Weighted bin

Python interface

class iplt.alg.**WeightedBin**

Add (*(WeightedBin)arg1*, *(float)arg2*, *(float)arg3*, *(float)arg4*) → None :

C++ signature : void Add(iplt::alg::WeightedBin {lvalue},float,float,float)

CalcBin (*(WeightedBin)arg1*, *(float)arg2*) → int :

C++ signature : int CalcBin(iplt::alg::WeightedBin {lvalue},float)

GetAverage (*(WeightedBin)arg1*, *(int)arg2*) → float :

C++ signature : float GetAverage(iplt::alg::WeightedBin {lvalue},unsigned int)

GetBinCount (*(WeightedBin)arg1*) → int :

C++ signature : unsigned int GetBinCount(iplt::alg::WeightedBin {lvalue})

GetLimit (*(WeightedBin)arg1*, *(int)arg2*) → float :

C++ signature : float GetLimit(iplt::alg::WeightedBin {lvalue},unsigned int)

GetLowerLimit (*(WeightedBin)arg1*, *(int)arg2*) → float :

C++ signature : float GetLowerLimit(iplt::alg::WeightedBin {lvalue},unsigned int)

GetSize (*(WeightedBin)arg1*, *(int)arg2*) → int :

C++ signature : unsigned int GetSize(iplt::alg::WeightedBin {lvalue},unsigned int)

GetStdDev (*(WeightedBin)arg1*, *(int)arg2*) → float :

C++ signature : float GetStdDev(iplt::alg::WeightedBin {lvalue},unsigned int)

GetUpperLimit (*(WeightedBin)arg1*, *(int)arg2*) → float :

C++ signature : float GetUpperLimit(iplt::alg::WeightedBin {lvalue},unsigned int)

GetWeightAverage (*(WeightedBin)arg1*, *(int)arg2*) → float :

C++ signature : float GetWeightAverage(iplt::alg::WeightedBin {lvalue},unsigned int)

__init__ (*(object)arg1*) → None :

C++ signature : void __init__(_object*)

__init__((object)arg1, (int)arg2, (float)arg3, (float)arg4) -> None :

C++ signature : void __init__(_object*,int,float,float)

C++ interface

class **iplt::alg::WeightedBin**

Public Functions

WeightedBin()

default ctor with count=1, lower limit=1 and higher limit=2, not really useful

WeightedBin(unsigned int bincount, Real lim_low, Real lim_hi)

Initialize with number of bins and bounds.

bincount: number of bins to use limlow: lower limit limhi: higher limit

void **Add**(Real x, Real y, Real weight = 1.0)

Add value at a given point, possibly with a weight.

int **CalcBin**(Real x)

Real **GetAverage**(unsigned int bin)

Retrieve (weighted) average value from specified bin.

ave=sum[value_i*weight_i]/sum[weight_i]

Real **GetStdDev**(unsigned int bin)

Retrieve (weighted) std dev from specified bin.

sdev^2 = sum[value_i*weight_i-<ave>]/sum[weight_i]

Real **GetWeightAverage**(unsigned int bin)

Retrieve average of weights.

<we>=1.0/sum[1.0/weight_i]

this is simply the stdev additive rule, where a cumulative standard deviation of additions is given by sqrt[sum[stdev_i^2]]

unsigned int **GetSize**(unsigned int bin)

return number of entries in given bin

unsigned int **GetBinCount**()

Return number of bins.

Real **GetLimit**(unsigned int bin)

Retrieve middle limit of this bin.

Real **GetLowerLimit**(unsigned int bin)

Real **GetUpperLimit**(unsigned int bin)

class **BinEntry**

Public Functions

BinEntry()

Public Members

unsigned int **num**

Real **sum**

Real **sum2**

Real **weightsum**

Real **iweightsum**

3.2.17 Weighted flexible bin

Python interface

class iplt.alg.**WeightedFlexibleBin**

Add(*(WeightedFlexibleBin)arg1*, *(float)arg2*, *(float)arg3*, *(float)arg4*) → None :

C++ signature : void Add(iplt::alg::WeightedFlexibleBin {lvalue},float,float,float)

CalcBin(*(WeightedFlexibleBin)arg1*, *(float)arg2*) → int :

C++ signature : int CalcBin(iplt::alg::WeightedFlexibleBin {lvalue},float)

GetAverage(*(WeightedFlexibleBin)arg1*, *(int)arg2*) → float :

C++ signature : float GetAverage(iplt::alg::WeightedFlexibleBin {lvalue},unsigned int)

GetBinCount(*(WeightedFlexibleBin)arg1*) → int :

C++ signature : int GetBinCount(iplt::alg::WeightedFlexibleBin {lvalue})

GetLimit(*(WeightedFlexibleBin)arg1*, *(int)arg2*) → float :

C++ signature : float GetLimit(iplt::alg::WeightedFlexibleBin {lvalue},unsigned int)

GetLowerLimit(*(WeightedFlexibleBin)arg1*, *(int)arg2*) → float :

C++ signature : float GetLowerLimit(iplt::alg::WeightedFlexibleBin {lvalue},unsigned int)

GetSize(*(WeightedFlexibleBin)arg1*, *(int)arg2*) → int :

C++ signature : int GetSize(iplt::alg::WeightedFlexibleBin {lvalue},unsigned int)

GetStdDev (*(WeightedFlexibleBin)arg1*, *(int)arg2*) → float :

C++ signature : float GetStdDev(iplt::alg::WeightedFlexibleBin {lvalue},unsigned int)

GetUpperLimit (*(WeightedFlexibleBin)arg1*, *(int)arg2*) → float :

C++ signature : float GetUpperLimit(iplt::alg::WeightedFlexibleBin {lvalue},unsigned int)

GetWeightAverage (*(WeightedFlexibleBin)arg1*, *(int)arg2*) → float :

C++ signature : float GetWeightAverage(iplt::alg::WeightedFlexibleBin {lvalue},unsigned int)

SetBinCount (*(WeightedFlexibleBin)arg1*, *(int)arg2*) → None :

C++ signature : void SetBinCount(iplt::alg::WeightedFlexibleBin {lvalue},unsigned int)

SetGlobalLowerLimit (*(WeightedFlexibleBin)arg1*, *(float)arg2*) → None :

C++ signature : void SetGlobalLowerLimit(iplt::alg::WeightedFlexibleBin {lvalue},float)

SetGlobalUpperLimit (*(WeightedFlexibleBin)arg1*, *(float)arg2*) → None :

C++ signature : void SetGlobalUpperLimit(iplt::alg::WeightedFlexibleBin {lvalue},float)

__init__ (*(object)arg1*) → None :

C++ signature : void __init__(_object*)

__init__((object)arg1, (int)arg2, (float)arg3, (float)arg4) -> None :

C++ signature : void __init__(_object*,int,float,float)

C++ interface

class **iplt::alg::WeightedFlexibleBin**

Public Functions

WeightedFlexibleBin()

default ctor with count=1, lower limit=1 and higher limit=2, not really useful

WeightedFlexibleBin(unsigned int bincount, Real lim_low, Real lim_hi)

Initialize with number of bins and bounds.

bincount: number of bins to use limlow: lower limit limhi: higher limit

void **Add**(Real x, Real y, Real weight = 1.0)

Add value at a given point, possibly with a weight.

int **CalcBin**(Real x)

Real **GetAverage**(unsigned int bin)

Retrieve (weighted) average value from specified bin.

ave=sum[value_i*weight_i]/sum[weight_i]

Real **GetStdDev**(unsigned int bin)

Retrieve (weighted) std dev from specified bin.

sdev^2 = sum[value_i*weight_i-<ave>]/sum[weight_i]

Real **GetWeightAverage**(unsigned int bin)

Retrieve average of weights.

<we>=1.0/sum[1.0/weight_i]

this is simply the stdev additive rule, where a cumulative standard deviation of additions is given by sqrt[sum[stdev_i^2]]

int **GetSize**(unsigned int bin)

return number of entries in given bin

int **GetBinCount**()

Return number of bins.

void **SetBinCount**(unsigned int count)

Real **GetLimit**(unsigned int bin)

Retrieve middle limit of this bin.

Real **GetLowerLimit**(unsigned int bin)

Real **GetUpperLimit**(unsigned int bin)

void **SetGlobalLowerLimit**(Real limit)

void **SetGlobalUpperLimit**(Real limit)

class **BinEntry**

Public Functions

BinEntry()

BinEntry(Real xx, Real yy, Real ww)

bool **operator<**(const BinEntry & rhs)

Public Members

Real **x**

Real **y**

Real **w**

3.3 iplt.gui module: Graphical user interface

3.3.1 CTF overlay

Python interface

class `iplt.gui.`**`CTFOverlay`**

`GetTCIFData`(*(CTFOverlay)arg1*) → TCIFData :

C++ signature : iplt::TCIFData GetTCIFData(iplt::gui::CTFOverlay {lvalue})

`SetTCIFData`(*(CTFOverlay)arg1*, *(TCIFData)arg2*) → None :

C++ signature : void SetTCIFData(iplt::gui::CTFOverlay {lvalue},iplt::TCIFData)

`__init__`(*(object)arg1*[, *(TCIFData)arg2*]) → None :

C++ signature : void __init__(_object* [,iplt::TCIFData])

C++ interface

class **iplt::gui::CTFOverlay**

Public Functions

CTFOverlay(const TCIFData & data = TCIFData ())

TCIFData **GetTCIFData**()

void **SetTCIFData**(const TCIFData & data)

void **OnDraw**(QPainter & pnt, ost::img::gui::DataViewerPanel * dvp, bool is_active)

call when redrawing is necessary

params: the drawing context, the parent data viewer, as well as a flag to indicate wether this overlay is currently active

void **OnMenuEvent**(QAction * e)

bool **OnMouseEvent**(QMouseEvent * e, ost::img::gui::DataViewerPanel * dvp, const QPoint & lastmouse)

mouse event handler

if an overlay is active, it receives the mouse events. if it handles the event for itself, it should return true, otherwise false should be returned to give the data viewer a chance to handle the event for itself

bool **OnKeyEvent**(QKeyEvent * e, ost::img::gui::DataViewerPanel * dvp)

key event handler

see comments for the mouse events

QMenu * **GetMenu**()

return overlay specific pulldown menu

3.3.2 Lattice overlay

Keyboard and mouse bindings

The user can modify the lattice overlay by pressing `Control` (`Command` on OSX) and using the LMB (Left Mouse Button), CMB (Center Mouse Button), RMB (Right Mouse Button) or keys on the keyboard.

Control-LMB	Move Lattice Point
Control-CMB	Rotate Lattice
Control-RMB	Shift Origin
Control-1	Set first anchor
Control-2	Set second anchor
Control-F	fix new lattice
Control-R	reject new lattice

Python interface

class iplt.gui.**LatticeOverlay**

SetLattice(*(LatticeOverlay)arg1*, *(Lattice)arg2*) → None :

C++ signature : void SetLattice(iplt::gui::LatticeOverlay {lvalue},iplt::Lattice)

SetMinAngle(*(LatticeOverlay)arg1*, *(float)arg2*) → None :

C++ signature : void SetMinAngle(iplt::gui::LatticeOverlay {lvalue},double)

__init__(*(object)arg1*, *(Lattice)arg2*[, *(str)arg3*]) → None :

C++ signature : void __init__(_object*,iplt::Lattice [,std::string])

__init__((object)arg1 [, (str)arg2]) -> None :

C++ signature : void __init__(_object* [,std::string])

C++ interface

class **iplt::gui::LatticeOverlay**

Public Functions

LatticeOverlay(const String & name = “Lattice”)

LatticeOverlay(const *Lattice* & lat, const String & name = “Lattice”)

void **OnMenuEvent**(QAction * e)

void **OnDraw**(QPainter & pnt, ost::img::gui::DataViewerPanel * dvp, bool is_active)

call when redrawing is necessary

params: the drawing context, the parent data viewer, as well as a flag to indicate wether this overlay is currently active

bool **OnMouseEvent**(QMouseEvent * e, ost::img::gui::DataViewerPanel * dvp, const QPoint & lastmouse)

mouse event handler

if an overlay is active, it receives the mouse events. if it handles the event for itself, it should return true, otherwise false should be returned to give the data viewer a chance to handle the event for itself

bool **OnKeyEvent**(QKeyEvent * e, ost::img::gui::DataViewerPanel * dvp)

key event handler

see comments for the mouse events

void **SetMinAngle**(const double & a)

double **GetMinAngle**()

bool **OnEditLattice**(const QPoint & p, ost::img::gui::DataViewerPanel * dvp)

3.3.3 Low Pass overlay

Python interface

class `iplt.gui.`**`LPOverlay`**

`__init__`(*(object)arg1*) → None :

C++ signature : void __init__(_object*)

C++ interface

class **iplt::gui::LPOverlay**

Public Functions

LPOverlay()

void **SetCutOff**(ost::img::Point cutoff)

ost::img::Point **GetCutOff**()

void **OnDraw**(QPainter & pnt, ost::img::gui::DataViewerPanel * dvp, bool is_active)

call when redrawing is necessary

params: the drawing context, the parent data viewer, as well as a flag to indicate wether this overlay is currently active

void **OnMenuEvent**(QAction * e)

bool **OnMouseEvent**(QMouseEvent * e, ost::img::gui::DataViewerPanel * dvp, const QPoint & lastmouse)

mouse event handler

if an overlay is active, it receives the mouse events. if it handles the event for itself, it should return true, otherwise false should be returned to give the data viewer a chance to handle the event for itself

bool **OnKeyEvent**(QKeyEvent * e, ost::img::gui::DataViewerPanel * dvp)

key event handler

see comments for the mouse events

QMenu * **GetMenu**()

return overlay specific pulldown menu

3.3.4 Reflection list overlay

Python interface

class `iplt.gui.`**`RListOverlay`**

`AddResolutionRing`(*(RListOverlay)arg1*, *(float)arg2*) → None :

C++ signature : void AddResolutionRing(iplt::gui::RListOverlay {lvalue},double)

`DisplayResolutionRings`(*(RListOverlay)arg1*, *(bool)arg2*) → None :

C++ signature : void DisplayResolutionRings(iplt::gui::RListOverlay {lvalue},bool)

`GetReflectionList`(*(RListOverlay)arg1*) → ReflectionList :

C++ signature : iplt::ReflectionList GetReflectionList(iplt::gui::RListOverlay {lvalue})

`HighlightSymmetryRelatedPoints`(*(RListOverlay)arg1*, *(Point)arg2*) → None :

C++ signature : void HighlightSymmetryRelatedPoints(iplt::gui::RListOverlay {lvalue},ost::img::Point)

RemoveResolutionRing (*(RListOverlay)arg1*, *(float)arg2*) → None :

C++ signature : void RemoveResolutionRing(iplt::gui::RListOverlay {lvalue},double)

SetReflectionList (*(RListOverlay)arg1*, *(ReflectionList)arg2*) → None :

C++ signature : void SetReflectionList(iplt::gui::RListOverlay {lvalue},iplt::ReflectionList)

__init__ (*(object)arg1*, *(ReflectionList)arg2*[, *(str)arg3*]) → None :

C++ signature : void __init__(_object*,iplt::ReflectionList [,std::string])

C++ interface

class **iplt::gui::RListOverlay**

Public Functions

RListOverlay(const ReflectionList & rlist, const String & name = "ReflectionList")

void **OnDraw**(QPainter & pnt, ost::img::gui::DataViewerPanel * dvp, bool is_active)

call when redrawing is necessary

params: the drawing context, the parent data viewer, as well as a flag to indicate wether this overlay is currently active

void **OnMenuEvent**(QAction * e)

bool **OnMouseEvent**(QMouseEvent * e, ost::img::gui::DataViewerPanel * dvp, const QPoint & lastmouse)

mouse event handler

if an overlay is active, it receives the mouse events. if it handles the event for itself, it should return true, otherwise false should be returned to give the data viewer a chance to handle the event for itself

bool **OnKeyEvent**(QKeyEvent * e, ost::img::gui::DataViewerPanel * dvp)

key event handler

see comments for the mouse events

ReflectionList **GetReflectionList**()

void **SetReflectionList**(const ReflectionList & rlist)

void **AddResolutionRing**(double value)

void **RemoveResolutionRing**(double value)

void **DisplayResolutionRings**(bool flag)

void **HighlightSymmetryRelatedPoints**(const ost::img::Point & index)

3.3.5 Spot list overlay

Python interface

class `iplt.gui.`**`SpotlistOverlay`**

`GetIndexedList`(*(SpotlistOverlay)arg1*) → IndexedList :

C++ signature : iplt::gui::IndexedList GetIndexedList(iplt::gui::SpotlistOverlay {lvalue})

`SetIndexedList`(*(SpotlistOverlay)arg1*, *(IndexedList)arg2*) → None :

C++ signature : void SetIndexedList(iplt::gui::SpotlistOverlay {lvalue},iplt::gui::IndexedList)

`__init__`(*(object)arg1*, *(Lattice)arg2*) → None :

C++ signature : void __init__(_object*,iplt::Lattice)

__init__((object)arg1, (IndexedList)arg2) -> None :

C++ signature : void __init__(_object*,iplt::gui::IndexedList)

C++ interface

class **iplt::gui::SpotlistOverlay**

Public Functions

SpotlistOverlay(const *Lattice* & lat)

SpotlistOverlay(const IndexedList & ilist)

void **OnDraw**(QPainter & pnt, ost::img::gui::DataViewerPanel * dvp, bool is_active)

call when redrawing is necessary

params: the drawing context, the parent data viewer, as well as a flag to indicate wether this overlay is currently active

void **OnMenuEvent**(QAction * e)

bool **OnMouseEvent**(QMouseEvent * e, ost::img::gui::DataViewerPanel * dvp, const QPoint & lastmouse)

mouse event handler

if an overlay is active, it receives the mouse events. if it handles the event for itself, it should return true, otherwise false should be returned to give the data viewer a chance to handle the event for itself

bool **OnKeyEvent**(QKeyEvent * e, ost::img::gui::DataViewerPanel * dvp)

key event handler

see comments for the mouse events

QMenu * **GetMenu**()

return overlay specific pulldown menu

void **SetIndexedList**(const IndexedList & ili)

IndexedList **GetIndexedList**()

3.3.6 Unit cell overlay

Python interface

class iplt.gui.**UnitCellOverlay**

GetPhaseShift (*(UnitCellOverlay)arg1*) → object :

C++ signature : geom::Vec2 GetPhaseShift(iplt::gui::UnitCellOverlay {lvalue})

SetPhaseShift (*(UnitCellOverlay)arg1*, *(object)arg2*) → None :

C++ signature : void SetPhaseShift(iplt::gui::UnitCellOverlay {lvalue},geom::Vec2)

__init__ (*(object)arg1*, *(SpatialUnitCell)arg2*) → None :

C++ signature : void __init__(_object*,iplt::SpatialUnitCell)

C++ interface

class **iplt::gui::UnitCellOverlay**

Public Functions

UnitCellOverlay(const SpatialUnitCell & uc)

void **OnDraw**(QPainter & pnt, ost::img::gui::DataViewerPanel * dvp, bool is_active)

call when redrawing is necessary

params: the drawing context, the parent data viewer, as well as a flag to indicate wether this overlay is currently active

void **OnMenuEvent**(QAction * e)

bool **OnMouseEvent**(QMouseEvent * e, ost::img::gui::DataViewerPanel * dvp, const QPoint & lastmouse)

mouse event handler

if an overlay is active, it receives the mouse events. if it handles the event for itself, it should return true, otherwise false should be returned to give the data viewer a chance to handle the event for itself

bool **OnKeyEvent**(QKeyEvent * e, ost::img::gui::DataViewerPanel * dvp)

key event handler

see comments for the mouse events

QMenu * **GetMenu**()

return overlay specific pulldown menu

void **SetUnitCell**(const SpatialUnitCell & s)

void **SetPhaseShift**(const geom::Vec2 & po)

geom::Vec2 **GetPhaseShift**()

CHAPTER

FOUR

TUTORIALS AND SCRIPTS

4.1 Tutorials

4.1.1 Quickstart

Start up `giplt` and enter the following lines:

```
from ost.img import *
```

This will import the basic image processing classes.

```
im=CreateImage(Size(200,200))
```

This will create an empty, 2D image, with a height and width of 200 pixels, whose origin (ie the pixel with the coordinates <0,0>) is in the top-left corner.

```
v=Viewer(im)
```

A `viewer window` will pop up (see below), showing a white frame on a black background. The inner area of the white frame is the image, which is empty. Let us fill this with random values.

```
rand_alg = alg.Randomize() # create algorithm object
im.ApplyIP( rand_alg ) # apply algorithm object in-place
```

As you can see, applying an algorithm is conceptually a two-step process. First, an instance of an algorithm class is created, yielding an algorithm object (in this case `rand_alg`). In a second step, the algorithm object is applied to an image, either *in-place*, modifying the image, or *out-of-place*, leaving the original image untouched, and returning the result as a new image. Note that the in-place/out-of-place logic is *decoupled* from the algorithm object.

Now that we have some (noisy) data present, let us run another algorithm, this time a gaussian filter with a sigma of 4 pixel.

```
im.ApplyIP( alg.GaussianFilter(4.0) ) # apply temporary algorithm object in-place
```

As you can see, it is not always necessary to create an independent algorithm instance first, in many cases a temporary object will suffice (this applies to the randomization algorithm as well, `im.ApplyIP(alg.Randomize())` would have been fine).

Having algorithms as instances is very useful, however, since it makes the algorithm *stateful*, meaning the algorithm can "remember" values prior to and after applying itself to an image. For example, to gather statistics about the image, the Stat algorithm can be employed as follows:

```
stat=alg.Stat()
im.ApplyIP(stat)
print stat.GetMean()
```

Here, the algorithm object is required to live beyond the `ApplyIP` call, in order for the results to be retrieved from it.

4.1.2 IPLT command line

In this tutorial you will learn how to do basic things on the IPLT command line.

The Command Line Basics

The IPLT command line is built on top of Python[1]. If you want to get more information on any object, function or class, the python help command may be useful.

```
#import classes from geom module
from ost.geom import *
# get list of methods of Rotation3
help(Rotation3)
# get help for method GetPsi
help(Rotation3.GetPsi)
```

Vector Math

IPLT has built-in support for vectors in 2,3 and 4 dimension. They are called Vec2, Vec3 and Vec4 respectively. You can use the vector classes like in a math formula: You can add them together, multiply them by scalar values or calculate the inner[2] and outer[3] product. This short code snippet gives an idea of how you can use vectors.

```
from ost.geom import *

a=Vec2(1,0)
b=Vec2(0,1)
# print the vectors
print a,b

# create a new vector by using components of both a and b.
# The components of a vector can be accessed by using the
# array subscript notation:
c=Vec2(a[0],b[1])
# print length of vector a and b
print Length(a), Length(b)

# calculate dot (inner) product
print Dot(a,b)

# do some fancy calculation
d = a+2*b
print '%s+2*%s is %s' % (a,b,d)
```

Linear Transforms

Linear transforms such as scaling, rotation of vectors can be achieved easily.

Rotation

Rotations are usually carried out using quaternions or rotation matrices. There are several helper classes in IPLT that encapsulate the logic of rotation.The Rotation3 class represents a rotation in 3D space defined by 3 Euler angles[4].

[1] http://www.python.org
[2] http://en.wikipedia.org/wiki/Dot_product
[3] http://en.wikipedia.org/wiki/Cross_product
[4] http://en.wikipedia.org/wiki/Euler_angles/

```
import math
# define rotation around x axis by 180 degress (PI radians)
rot=Rotation3(math.pi,0,0)
v=Vec3(0,1,0)
# rotate v by rot
print rot.GetRotationMatrix()*v
```

In two dimensions, the Rotate function can be used to rotate a vector by a certain amount

```
#rotate by 90 degrees (PI/2 radians)
print Rotate(Vec2(1,0),math.pi/2)
```

Scaling

Uniform vector scaling can be accomplished by multiplication of the vector by a scalar factor. For non-uniform scaling you can use a 3x3 matrix with all but the main diagonal elements set to zero.

```
scale=Mat3(1.0,0.0,0.0,
           0.0,2.0,0.0,
           0.0,0.0,4.0)

print scale*Vec3(1,1,1)
```

Short Example

While you are fitting a lattice to your diffraction pattern *Fit lattice*, you would like to set the angle between the first and second lattice vector to exactly 90 degrees. You decide to do it on the command line. Your a-vector is fitting perfectly, you only want to reorient the b-vector.

```
from ost.geom import *
from ost.io import LoadImage
import iplt.gui, math
# load the image and display it
image=LoadImage('marvelous_diff_pattern.tiff')
v=Viewer(image)
# add lattice overlay
lov=iplt.gui.LatticeOverlay()
v.AddOverlay(lov)

# get snapshot of lattice. note that all changes to lattice will not
# be reflected in the overlay. you need to call  lov.SetLattice(l)
l = lov.GetLattice()
length_b=Length(l.GetSecond())
new_b = Normalize(Rotate(l.GetFirst(),math.pi/2))*length_b
l.SetSecond(new_b)
lov.SetLattice(l)
```

4.1.3 Extracting and Pasting Images

An image can be extracted and pasted into another image using the `Extract()` and `Paste()` member functions:

```
# import the necessary classes
from ost.img import *
from ost.io import LoadImage
# load the image
im=LoadImage("imagename.tif")
# gererate a subimage from the region going from (10,10) to (30,30)
im2=im.Extract(Extent(Point(10,10),Point(30,30)))
```

```
# generate an empty image with the same size as the original image
im3=CreateImage(im.GetExtent())
# paste the subimage into the empty image
im3.Paste(im2)
```

Note that the extent is fully honored for the paste operation, i.e. only the region where the pasted-to and the pasted-in image overlap will be affected.

4.1.4 Applying a Fourier Transform to an image

An image is Fourier-transformed using the `alg.FFT()` algorithm object:

```
#imports
from ost.io import LoadImage
from ost.img.alg import FFT
from iplt.gui import Viewer
#Load the image
im=LoadImage("FILENAME.tif") # load the image
# create an instance of the fft algorithm object
fft=FFT()
# do the actual Fourier transformation
im_ft=im.Apply(fft)
# back-transform
im2 = im_ft.Apply(fft)
# if this is run from within giplt, open viewers to look at the images
Viewer(im)
Viewer(im_ft)
Viewer(im2)
```

It is not really necessary to create the `fft` instance, a temporary object can be used, since the `alg.FFT` algorithm object is stateless. In addition, the algorithm can be applied in-place to avoid the creation of a second image:

```
#imports
from ost.io import LoadImage
from ost.img.alg import FFT
#Load the image
im=LoadImage("FILENAME.tif") # load the image
# do the actual Fourier transformation, in-place using temporary object
im.ApplyIP(FFT())
# repeating this command will do the back-transform
im.ApplyIP(FFT())
```

Please note that the `FFT()` algorithm does not require a direction to be given, this is implicitly determined by the underlying image state: a SPATIAL image will always be transformed to the FREQUENCY domain, and vice-versa.

4.1.5 Fit lattice

Download the example image `test_grid_512.tif`, then start up `giplt`.

First load the image:

```
im = ost.io.LoadImage("test_grid_512.tif")
```

The image has now been loaded and is available via the python variable `im`. In order to display the image, a viewer must be opened:

```
v=Viewer(im)
```

The viewer will open as a separate widget (figure 4.1); it is nevertheless also accessible via the variable `v`, and the importance of this will become clear further below. (From the `Window` menu of the viewer, the `Info`, `Zoom` and `Overlay` helper windows may be turned on).

The manual lattice fitting needs to be performed on the Fourier-transform if this image, and this is most conveniently done with a quick in-place application of the `FFT` algorithm from the `alg` module.

```
im.ApplyIP(ost.img.alg.FFT())
```

The viewer will change to reflect the new content, but since the normalization and centering needs to be updated, the key shortcuts `C` (for **C**entering) and `N` (for **N**ormalization) should be pressed while the focus is in the viewer.

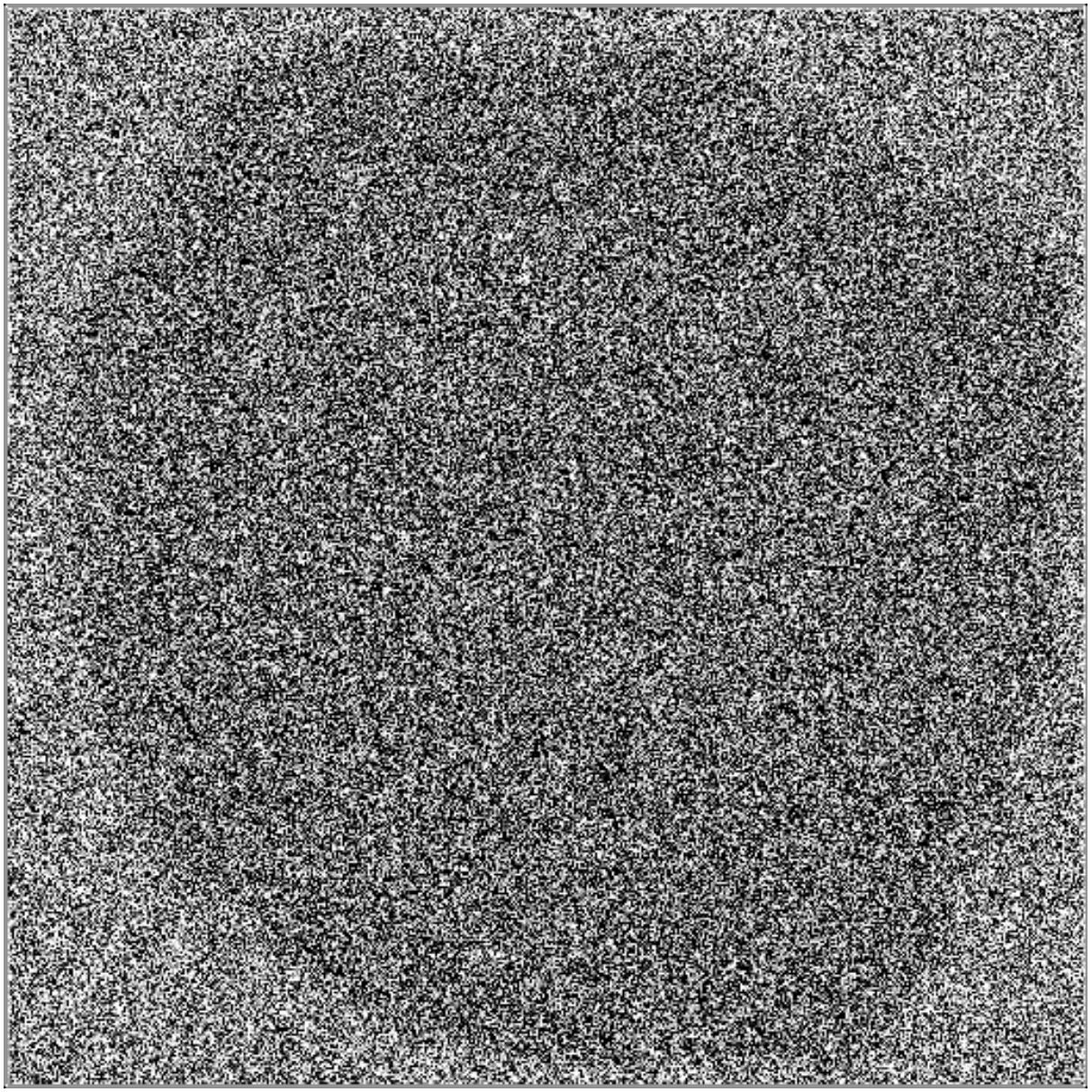

Figure 4.1: The example image after loading

Using the mouse-wheel or the keys `K`/`L` to zoom, the contrast of the Fourier transform is still somewhat bad, this can be fixed by selecting a small region around a peak (using the left mouse button), and then pressing `N` again - the viewer will maximize contrast for the pixels contained within the selected subregion.

Now its time to add the lattice overlay, which is a 2-step process: a lattice overlay object is first created and then added to the viewer. Since the code that will be used from this point on uses the `iplt` module and its GUI components, we need to import the proper modules before anything else:

```
# import the iplt module and the iplt.gui
import iplt
import iplt.gui
```

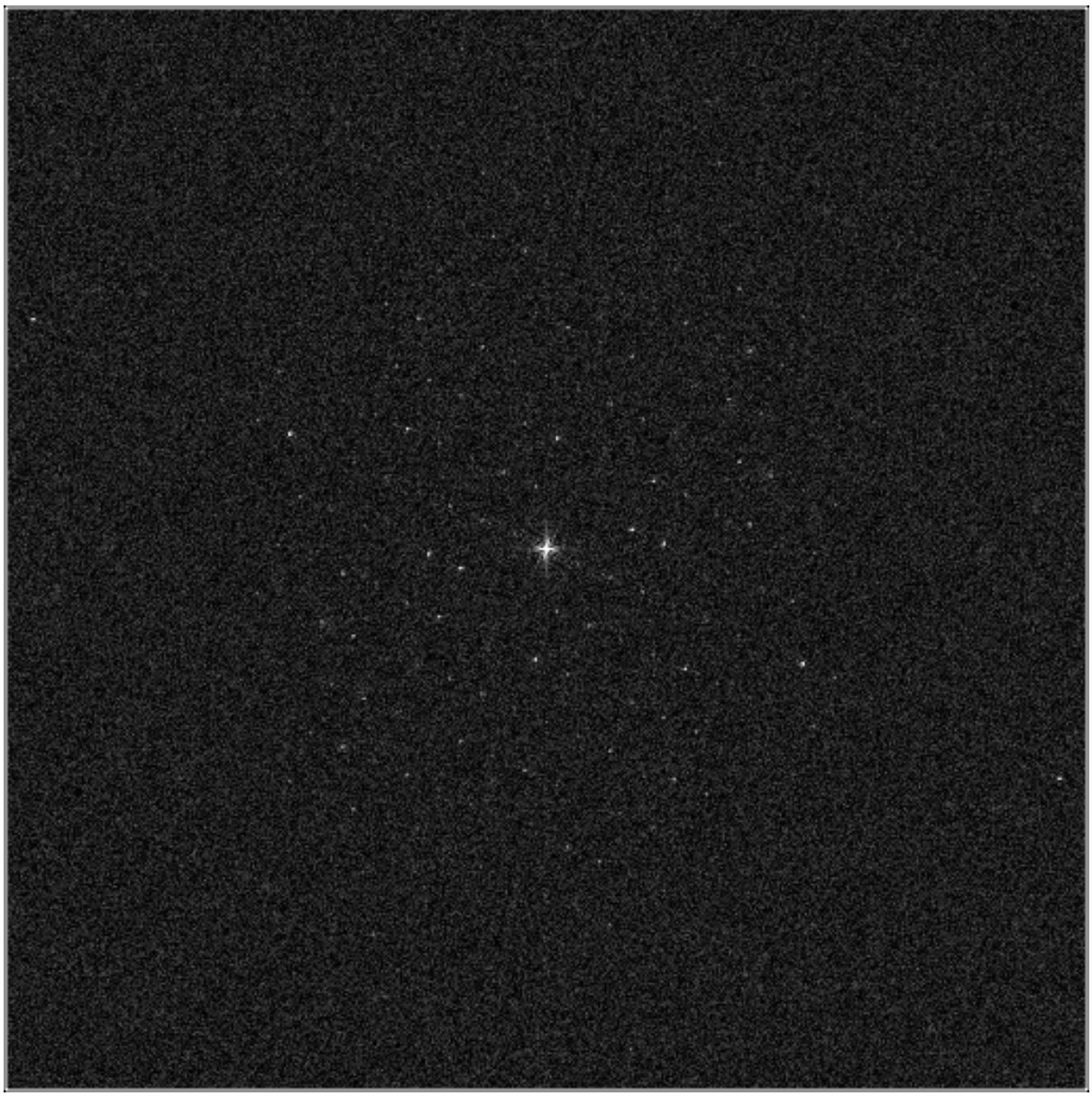

Figure 4.2: FFT of the example image

```
# create lattice overlay
lov=iplt.gui.LatticeOverlay()
# add newly created overlay to viewer
v.AddOverlay(lov)
```

The result is shown on the left side of the figure 4.3. Note that upon editing the lattice, a new one appears in a different color (as shown on the right), while the old one remains as it is. In order to **f**ix the new lattice, use `CTRL F`, and to **r**evert back to the old one use `CTRL R`.

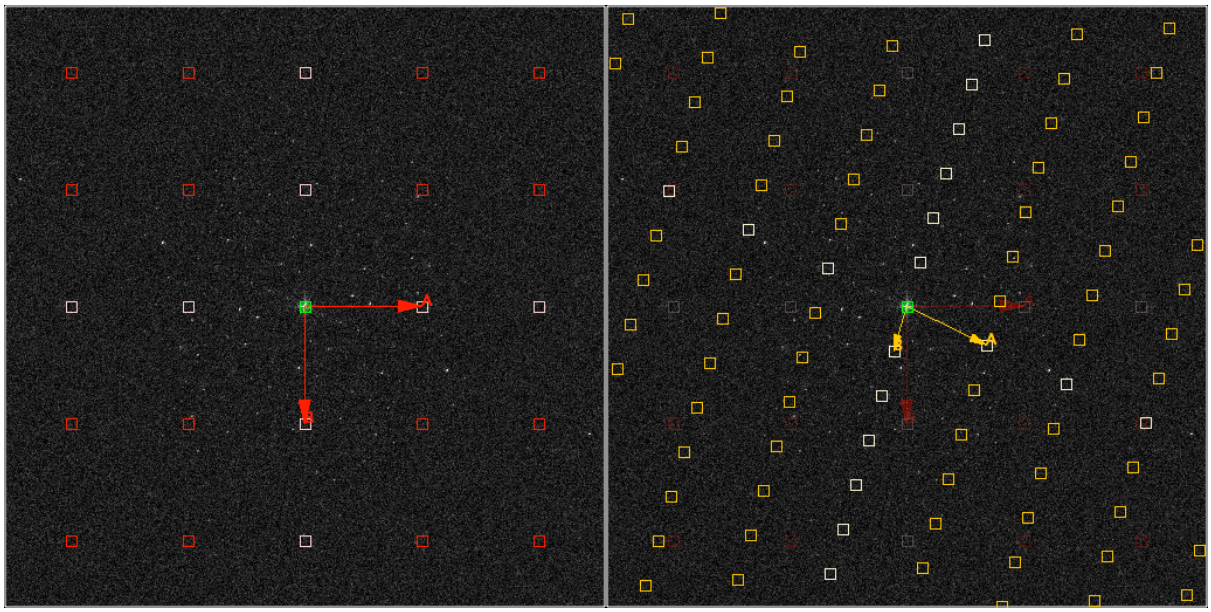

Figure 4.3: FFT with overlayd lattice

The following functionality is now available to to fit the lattice to the underlying pattern:

`CTRL LMB` Will shift the targeted lattice point around, adjusting the remaining lattice on the fly

`CTRL 2` Will fix the targeted lattice point, and subsequent adjustment of the lattice takes this constraint into account. To remove such a fixed point, press `CTRL 2` while hovering with the mouse between lattice points.

`CTRL G` Attempts to perform a local 2D gaussian fitting based on the underlying image, for the lattice point that the mouse is targetting

4.1.6 Fourier Peak Filtering and Correlation Averaging in Iplt

The goal of this exercise is to Fourier peak filter an image of a negatively stained crystal and to further improve the average by correlation averaging.

Starting the Tutorial

First download the sample image `raw.tif` used in the tutorial.

- Linux: Start the tutorial with the command **giplt_corr_avg** (part of the iplt distribution).
- Mac OS X: Double click the **IPLT Correlation Averaging** icon in the Applications folder.

This will start iplt and display the list of commands used for this tutorial.

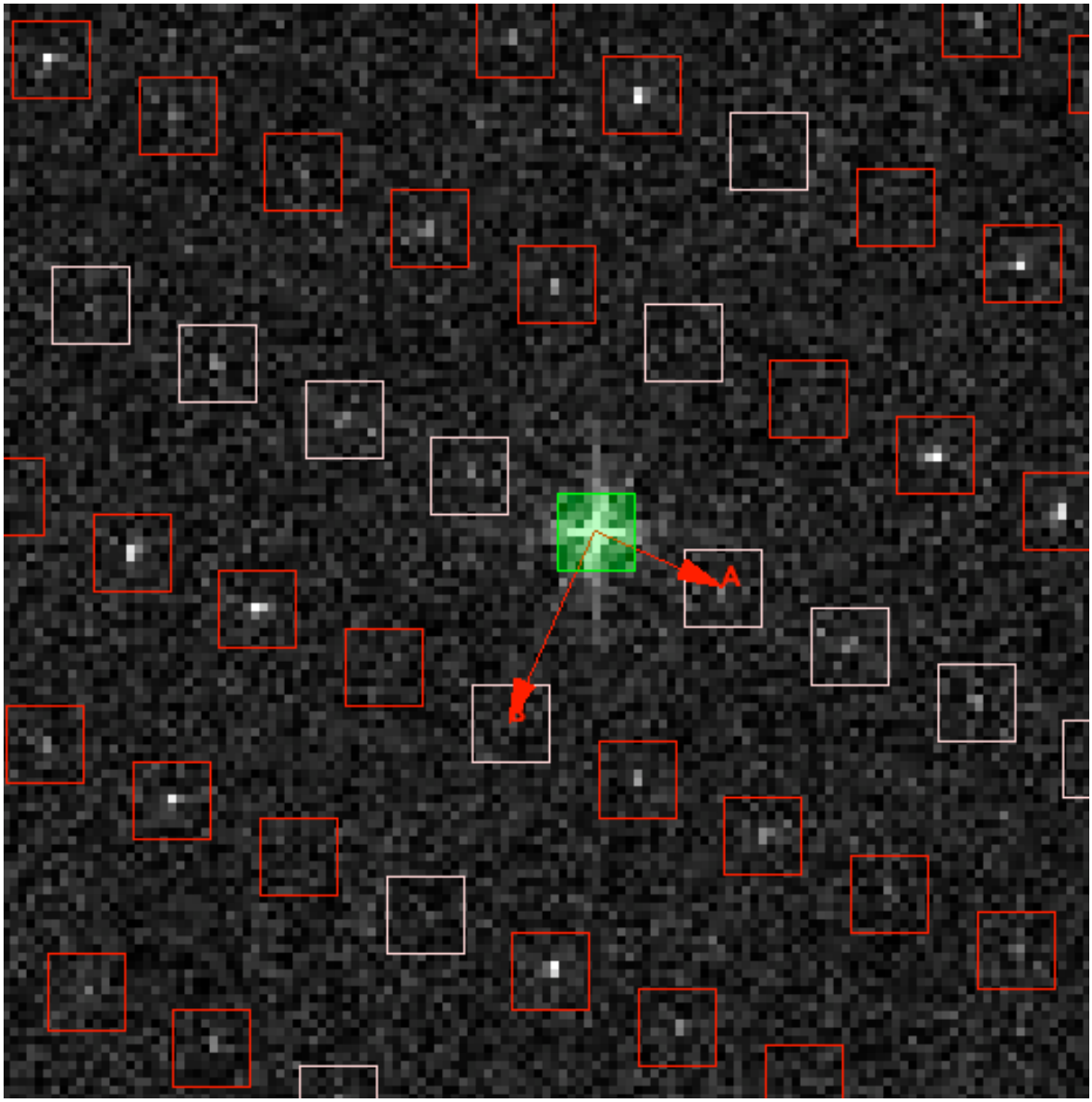

Figure 4.4: FFT with overlayd fitted lattice

Loading the Image

Press “Load raw image” to display a dialog where you can enter the name of the inital image (raw.tif) and the pixel sampling (8 Angstroem). The pixel sampling defines the pixel dimension, i.e. 8 means that the pixel extends 8 Angstroem in x and y direction.

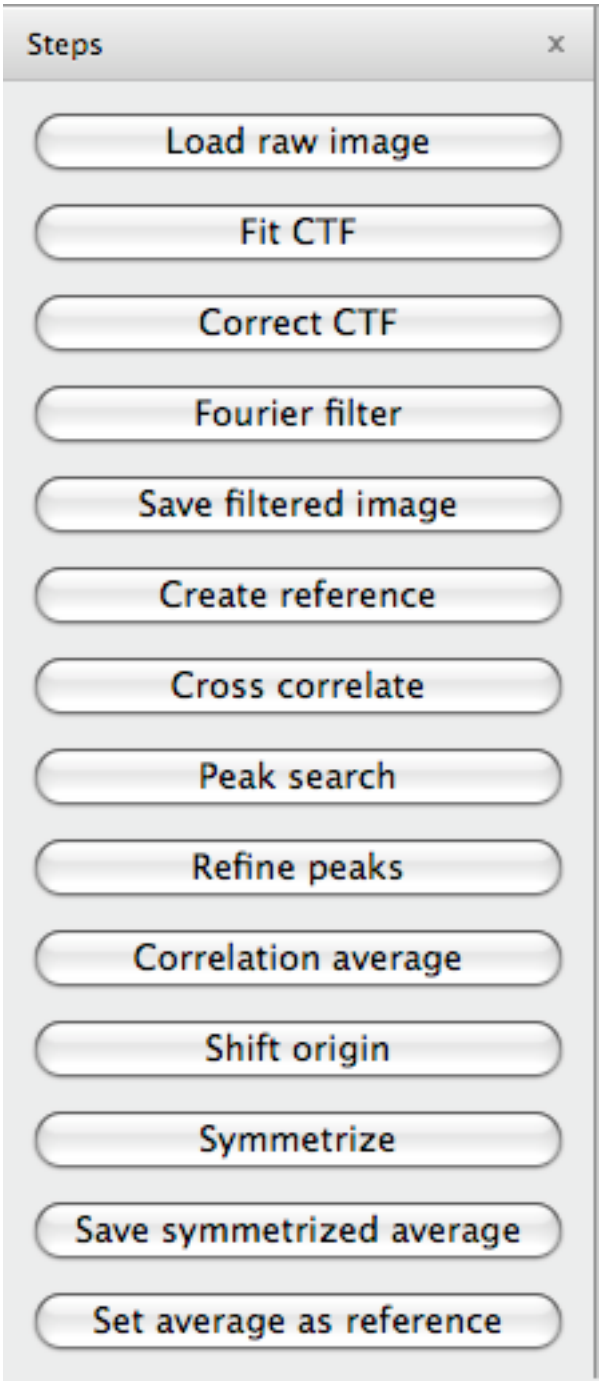

Figure 4.5: List of commands

You should now see two data viewers displaying the inital image and its Fourier transform. The Fourier transform will be mostly black with one bright pixel in the center. To improve the display of the Fourier transform select an area not containing the center with the mouse and press `N` to renormalize the image. By renormalizing the Fourier transfrom you will just modify the display of the image, not the original data.

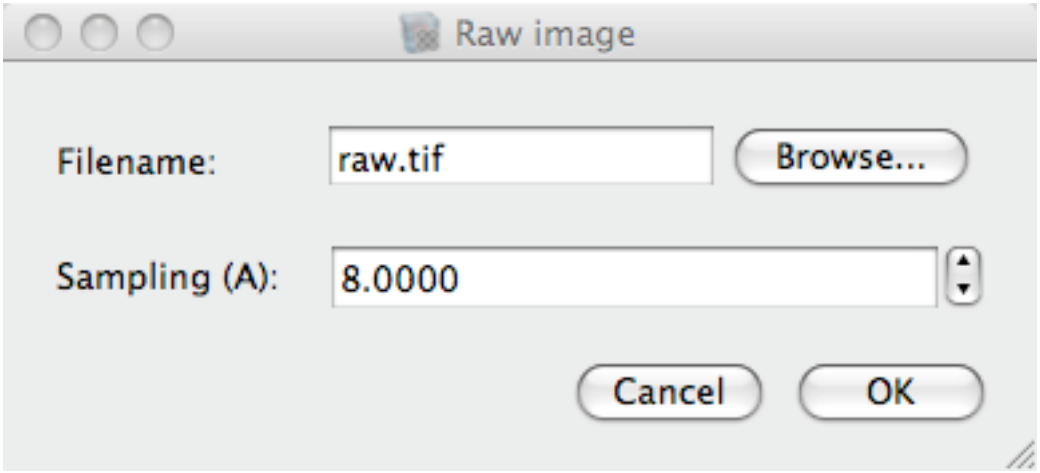

Figure 4.6: Load raw image dialog

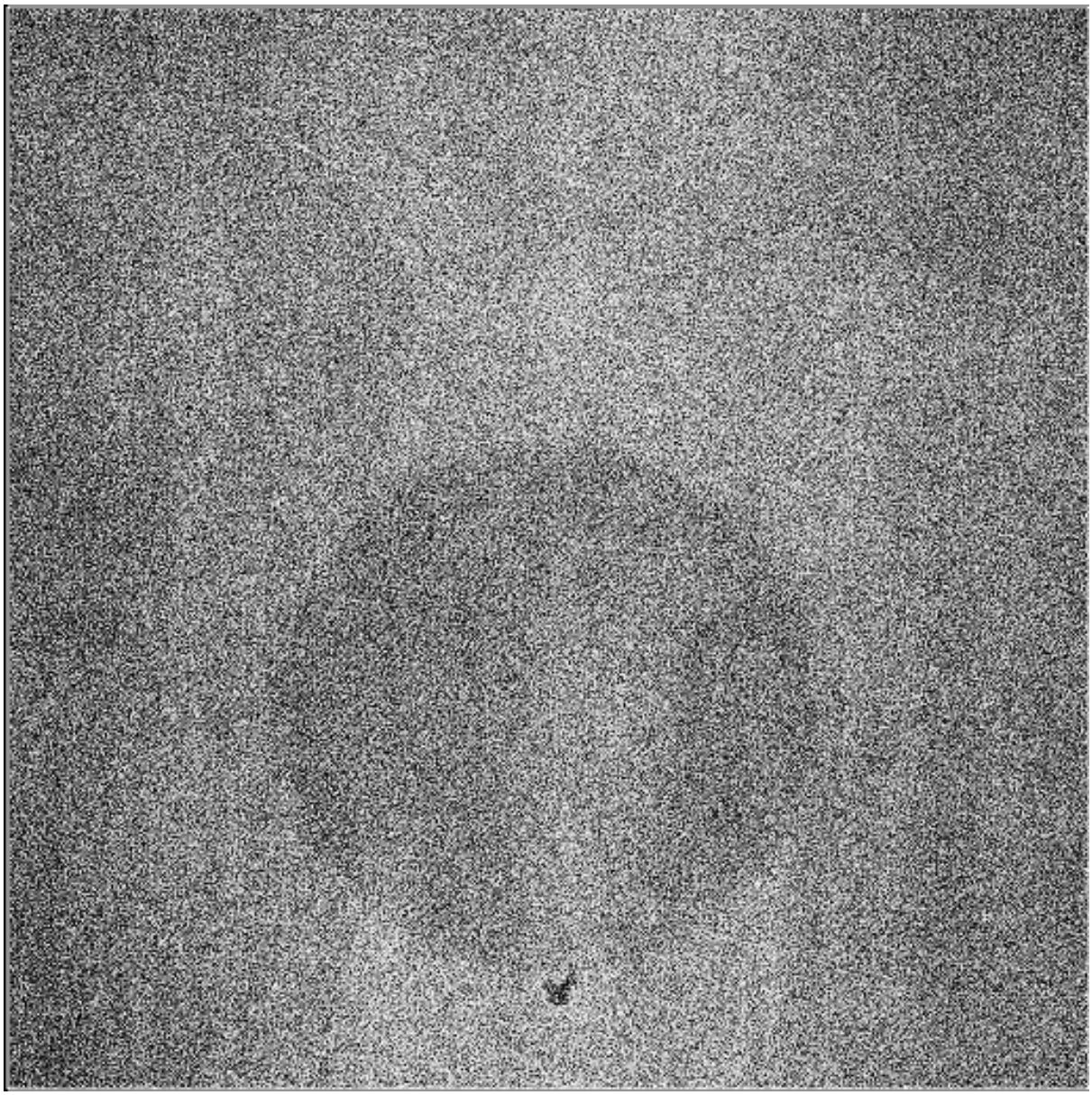

Figure 4.7: initial image

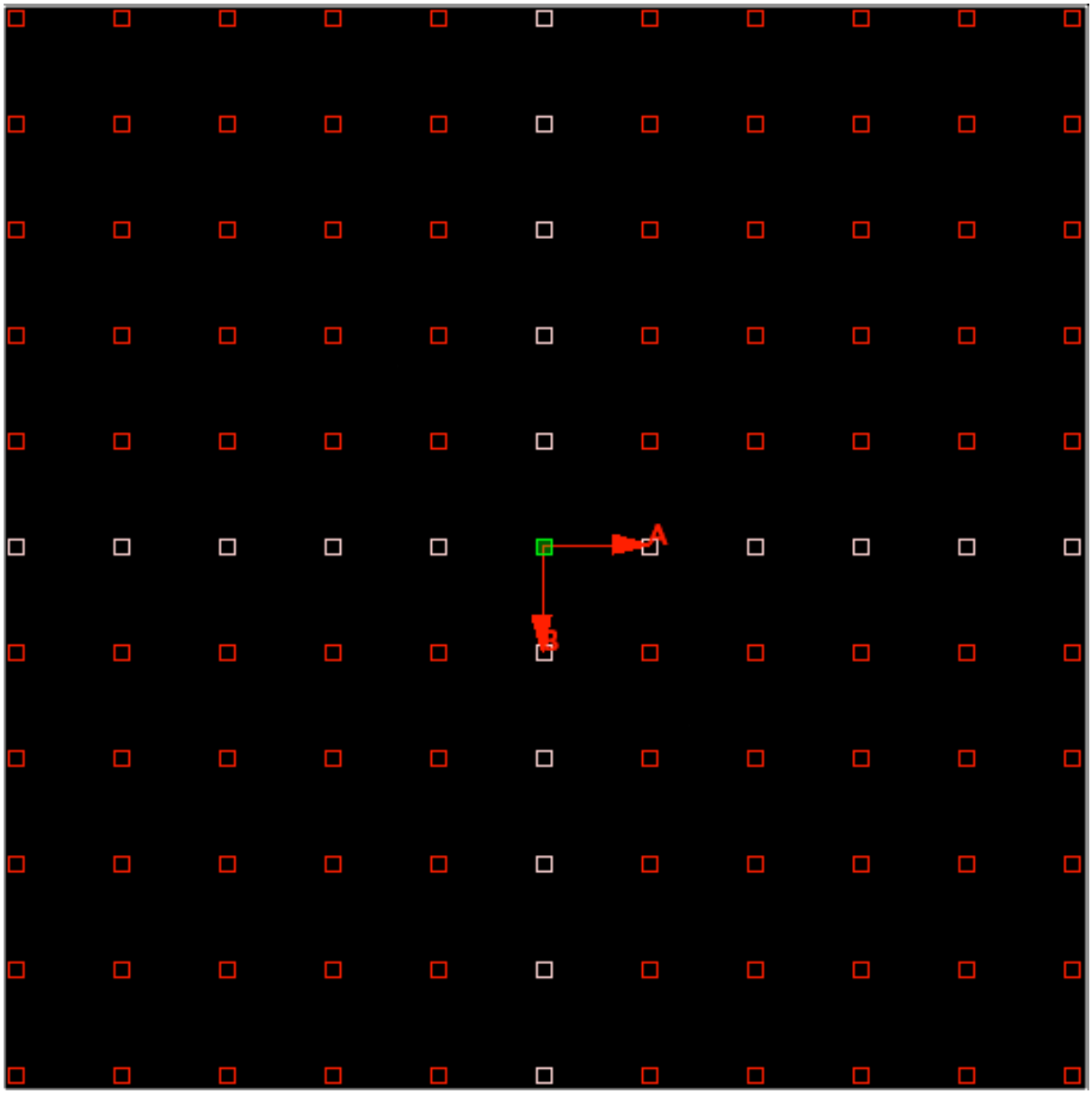

Figure 4.8: Fourier transform

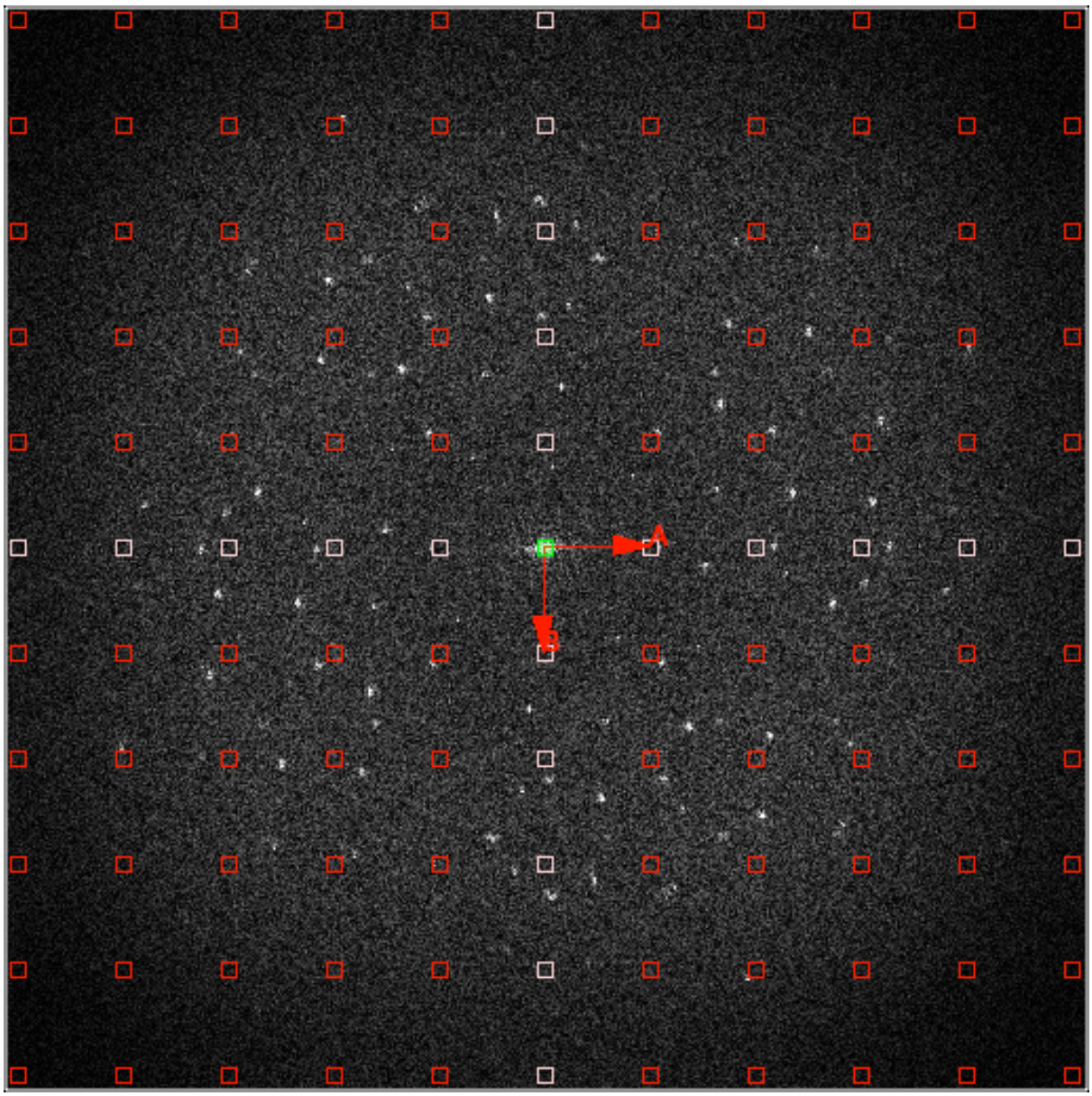

Figure 4.9: Renormalized image

Lattice Indexing

In the Fourier transform you see peaks corresponding to two crystal lattices. We will now fit one of the two lattices manually. Modify the *Lattice Overlay* (Press `Control` and use the LMB, CMB and RMB) to fit as well as possible one of the lattices you see in the Fourier Transform:

`Control-LMB`	Move Lattice Point
`Control-CMB`	Rotate Lattice
`Control-RMB`	Shift Origin
`Control-1`	Set first anchor
`Control-2`	Set second anchor
`Control-F`	fix new lattice
`Control-R`	reject new lattice

On MacOS X use `Command-<key>` instead of `Control-<key>`.

As soon as you start modifying the lattice, you will see two lattices displayed: One in red and one in beige. The lattice in red is the last saved lattice, the beige lattice is the modified version. You can go back to the red lattice by pressing <Control>+<r>. When you are satisfied with your lattice, press <Control>+<f>. This will save your new lattice. Note that for the subsequent Fourier peak filtering step, the red lattice will be used.

Example of one fitted lattice:

Fourier Peak Filtering

Since we know that all the information of the unit cell is concentrated on the spots in the lattice and everything outside is noise, we can now filter the image with a "Fourier Peak Filter". On a perfect crystal, the information would be concentrated on a single pixel. However, due to crystalline imperfections (bending, crystal is not infinite in size), the information is spread out over several pixels. That's why we include not only the central pixel of the lattice but also pixel inside a certain radius.

Press "Fourier Filtering" to Fourier peak filter the Fourier transform. A dialog will pop up to ask for the radius of the mask and for the high and low frequency cutoff values for the subsequent bandpass filter. For this example a mask radius of 5 pixels and a highpass threshold cutoff of 200 Angstroem should be reasonable. For the lowpass threshold cutoff a value of 16 Angstroem should be chosen. The dialog also asks which types of mask and of Fourier filter should be used. For both, Gaussian type should be selected.. After filtering the filtered, backtransformed image will be displayed. The red box indicates the size of a single unit cell.

Saving the filtered image

It is now possible to save the filtered image. This step is not necessary for further processing.

Correlation Averaging

The Correlation Averaging technique combines Fourier filtering and averaging. In the previous step, we created a Fourier filtered image which will serve us now as a reference for a cross-correlation with the original image: In combination with a peak search this will give us the locations of the unit cells of the lattice.

Select an area the size of ca. 2x2 unit cells of the filtered image with the mouse in the viewer to use it as reference. Press "Crosscorrelate" to calculate the cross correlation between inital image and selected reference. the crosscorrelation image will be displayed with the real space lattice overlayed in red.

Peak search

The lattice coincides only with part of the cross correlation maxima. The reason for this are bends in the crystal lattice. With a peaksearch we can never the less find the correct position of every good unit cell and use this position information to average the unit cells. Press "Peak Search" to open the dialog for entering the peak search parameters. The outer radius gives the minimal distance two peaks have to be separated. 8 pixels is a reasonable

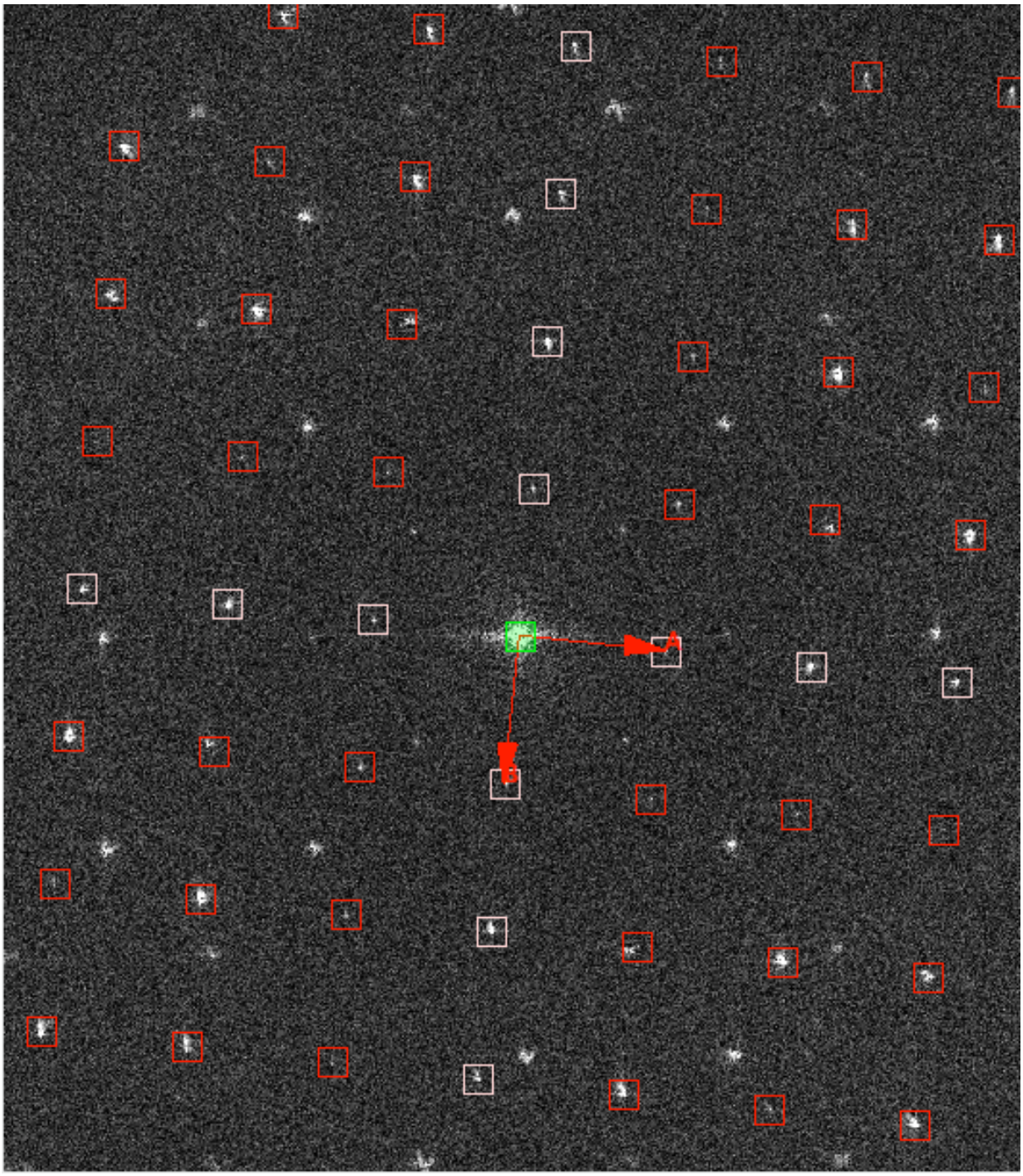

Figure 4.10: Manually fitted lattice

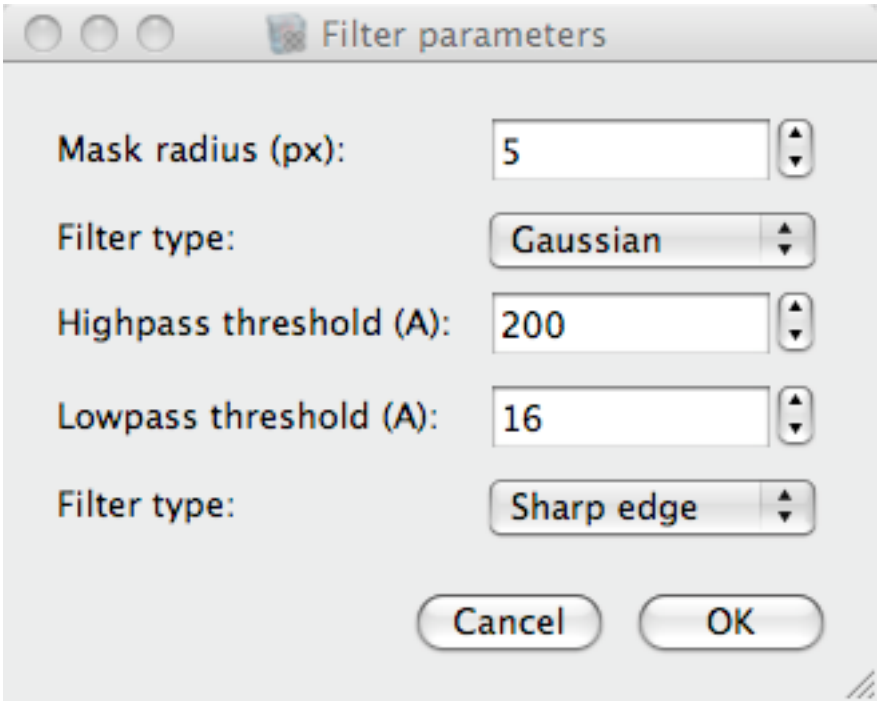

Figure 4.11: Fourier peak filtering dialog

value. For the inner radius (maximal radius of the peak maximum) use 3 pixels and for the sensitivity use 1.2. A lower sensitivity will lead to more spots to be picked up.

If not enough or too much peaks are found, clear all the marks in the Data viewer (Display->Clear All Marks) and redo the last step with slightly different parameters.

Peak refine

The correlation peak positions can be further refined using a 2D gauss fit. For this the size of the image around a peak has to be given on which the gauss fit will be performed.

Correlation averaging

Press "Correlation average" to start the correlation averaging. A dialog will ask about the size of the final image. Depending on the number of peaks and the speed of the computer this step can take several minutes. In the end the averaged image will be displayed. The red box indicates the size of the unit cell.

Saving the correlation average

The correlation average can now be saved by pressing "Save correlation average".

4.2 Short IPLT scripts perform common and useful tasks

4.2.1 Create Split Image

Usage

giplt `create_split_image.py` <input image 1> [<input image 2> <input image 3> ....] <output image>

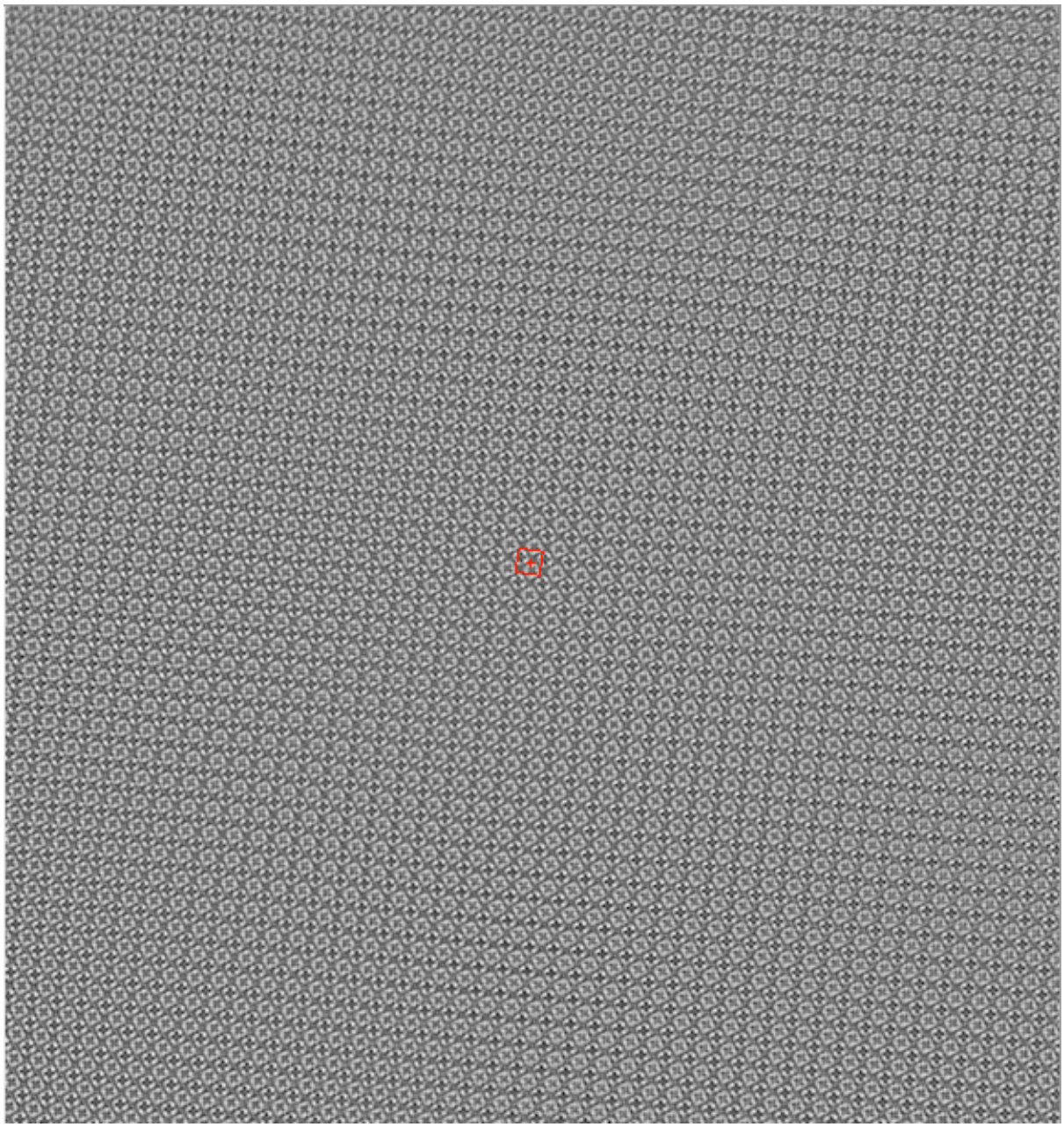

Figure 4.12: Fourier peak filtered image

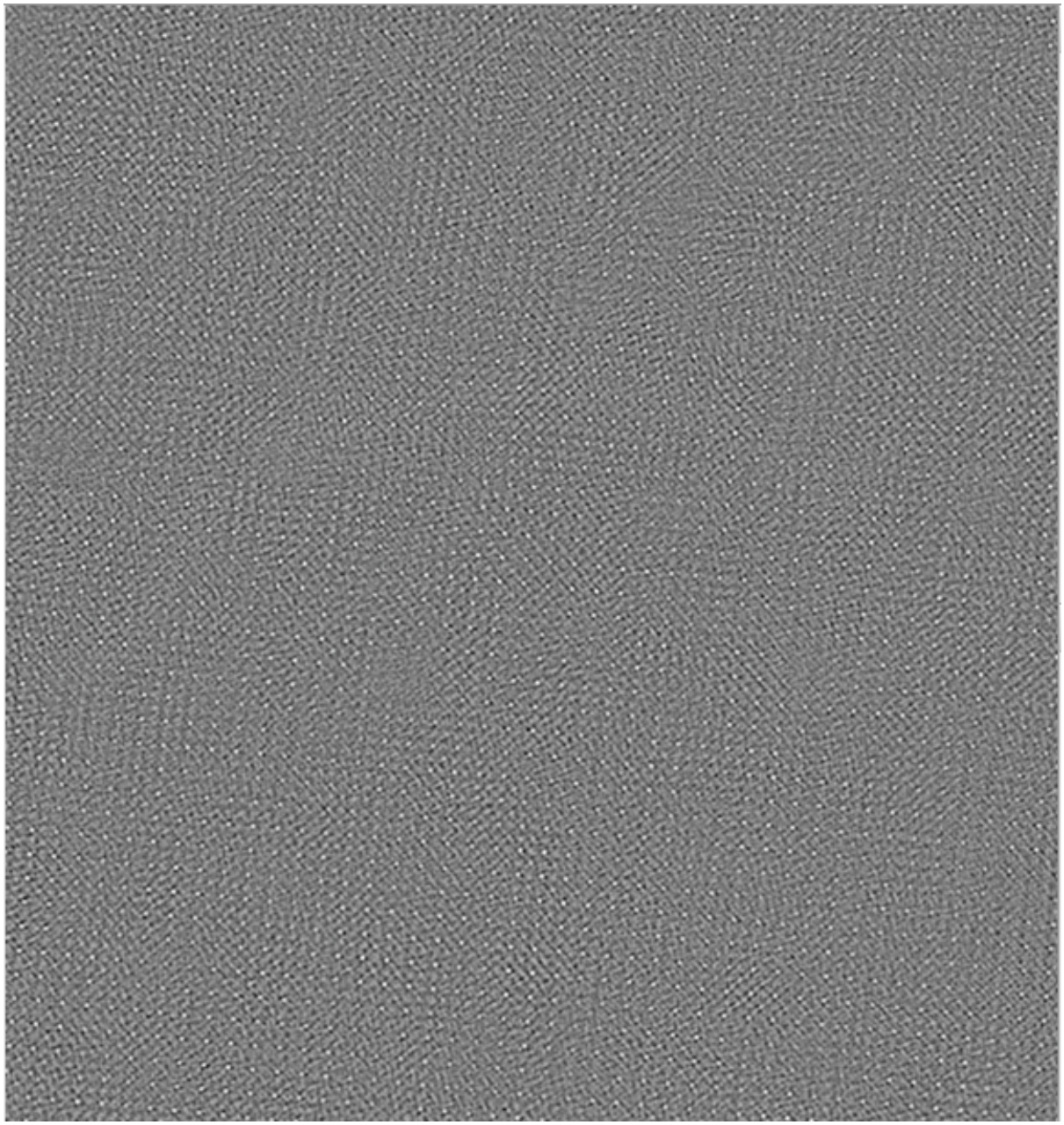

Figure 4.13: Cross correlation image

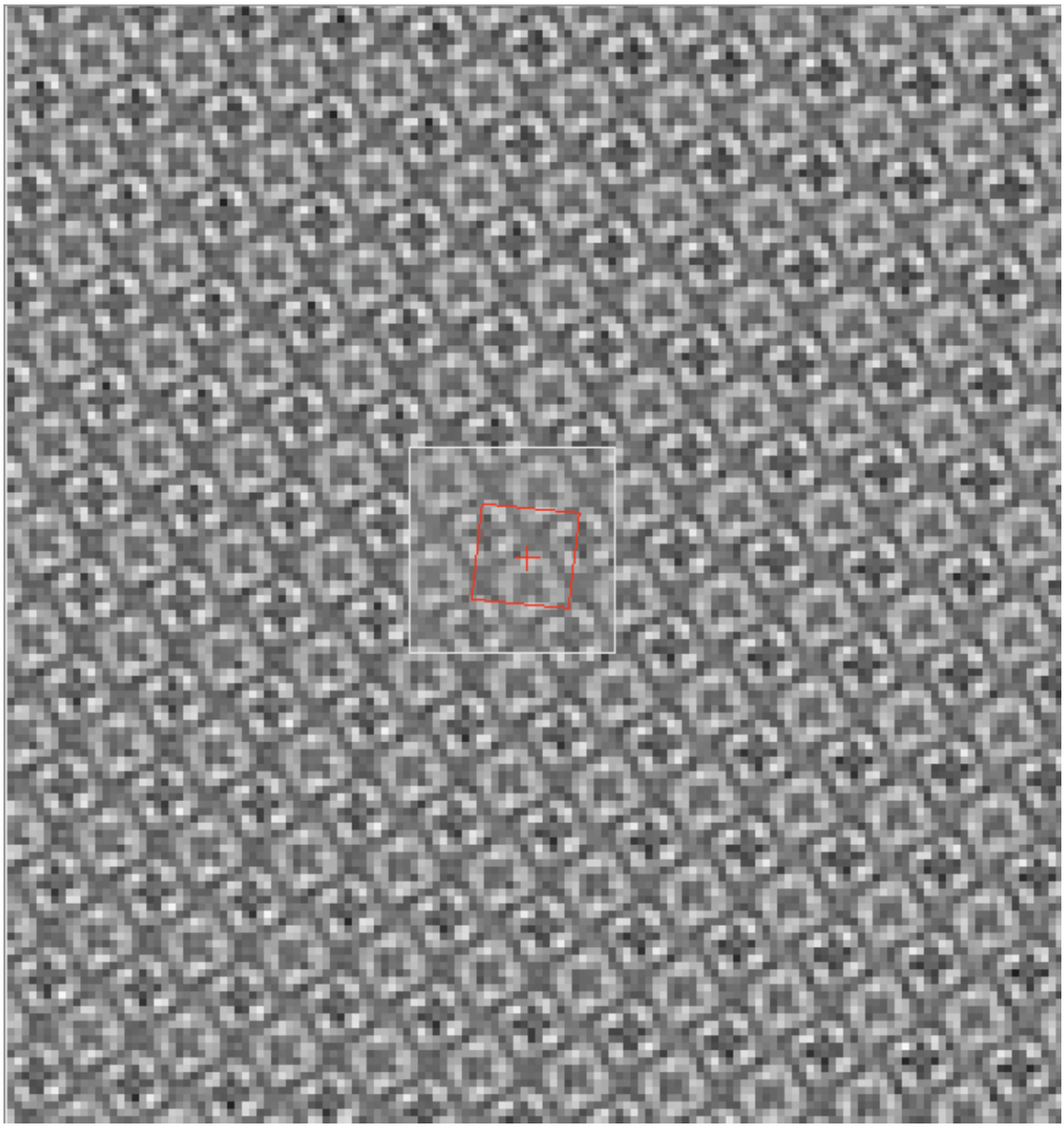

Figure 4.14: Selected reference

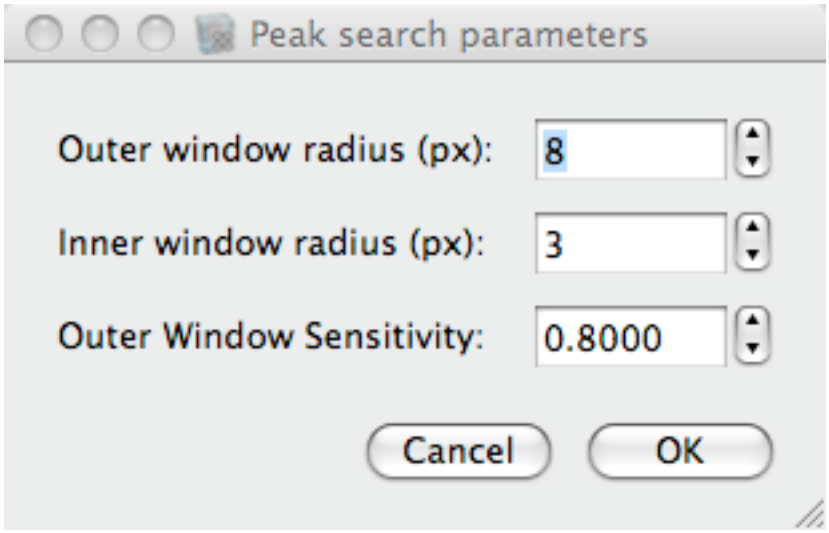

Figure 4.15: Peak search dialog

Description

This scripts loads one or more images and creates a split image, where each of the images is displayed in a cone of equal angle. See figure 4.19 for a sample image.

Script

```
import sys
from math import *
from ost.geom import *
from ost.img.alg import *
from ost.img import *
from ost.io import LoadImageList, SaveImage

def CreateSplitImage(imagelist, start_at_y=True):
    result=imagelist[0].Copy(False)
    result.CenterSpatialOrigin()
    extent=imagelist[0].GetExtent()
    if start_at_y:
            startpoint=Vec2(0.0,-extent.GetSize()[0])
    else:
            startpoint=Vec2(extent.GetSize()[0],0.0)
    angle=2*pi/len(imagelist)
    count=0
    for image in imagelist:
            image.CenterSpatialOrigin()
            start_angle=angle*count
            end_angle=angle*(count+1)
            pol=Polygon2()
            pol.AddNode(Vec2(0,0))
            pol.AddNode(Rotate(startpoint,start_angle))
            if(start_angle<pi/4.0 and end_angle>pi/4.0):
                    pol.AddNode(Rotate(startpoint,pi/4.0))
            if(start_angle<3.0*pi/4.0 and end_angle>3.0*pi/4.0):
                    pol.AddNode(Rotate(startpoint,3.0*pi/4.0))
            if(start_angle<5.0*pi/4.0 and end_angle>5.0*pi/4.0):
                    pol.AddNode(Rotate(startpoint,5.0*pi/4.0))
            if(start_angle<7.0*pi/4.0 and end_angle>7.0*pi/4.0):
                    pol.AddNode(Rotate(startpoint,7.0*pi/4.0))
            pol.AddNode(Rotate(startpoint,end_angle))
```

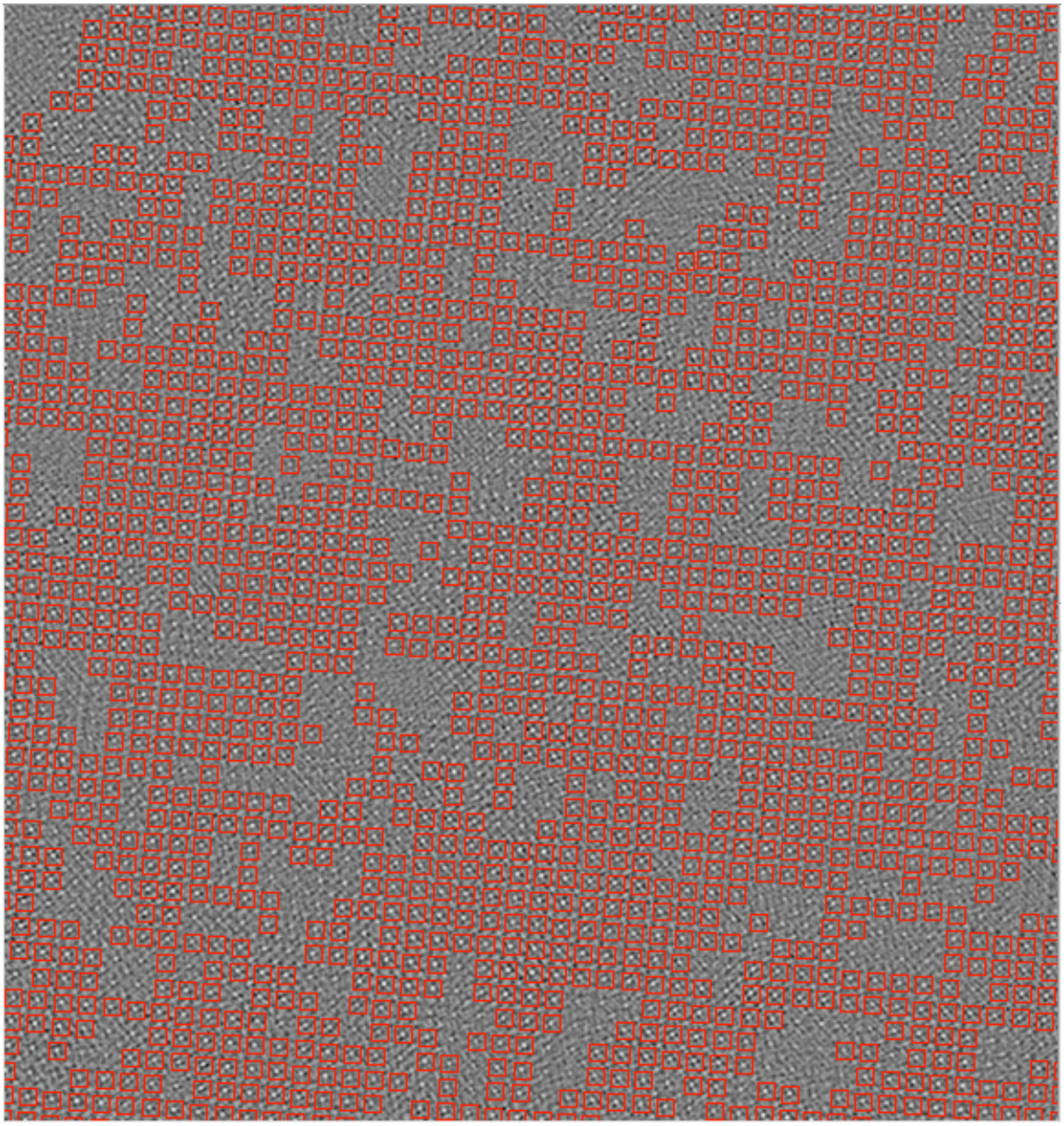

Figure 4.16: Cross correlation image showing the found peaks in red

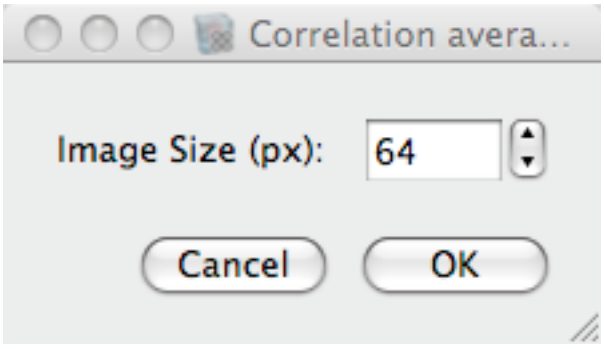

Figure 4.17: Correlation average dialog

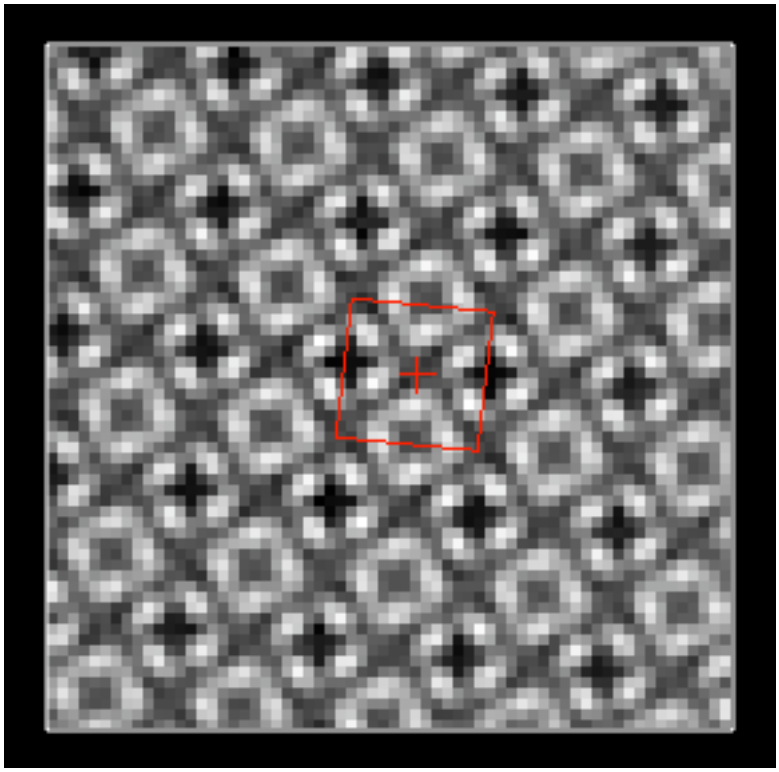

Figure 4.18: Correlation average

Figure 4.19: Split image

```
            m=Mask(pol)
            result+=image.Apply(MaskImage(m))
            count+=1
    return result

imagelist=LoadImageList(sys.argv[1:-1])
result=CreateSplitImage(imagelist)
SaveImage(result,sys.argv[-1])
```

You can download the script here.

4.2.2 Modulate Image by Python Function

This snippet shows how to define a python algorithm that modulates an image by a sine. Note the explicit call of ConstModIPAlgorithm.__init__ in the modulator class.

```
from math import sin

class SinusModulator(img.ConstModIPAlgorithm):
        def __init__(self,v):
                img.ConstModIPAlgorithm.__init__(self,"SinusModulator")
                self.v_=v

        def VisitImage(self,image):
                for point in image:
                        image.SetReal(point,sin(point[0]*self.v_))

image = img.CreateImage(img.Size(400,200))
sin_modulator=SinusModulator(5.3)
image.ApplyIP(sin_modulator)
v=Viewer(image)
```

4.2.3 View phase difference

Usage

giplt view_phase_difference.py <image1> <image2>

Description

This scripts displays the phase difference (in degrees) between corresponding pixels in the Fourier Transforms of the two input images, which must be of the same size. The Fourier Transforms honor the origin of the reference system, which is assumed to be at the center of the two images.

Script

```
import sys
import math
from ost.io import LoadImage
from ost.img import alg

image1=LoadImage(sys.argv[1])
image2=LoadImage(sys.argv[2])
if image1.GetExtent() != image2.GetExtent():
  print 'Error: The input images should have the same size.'
  print sys.exit(0)
image1.CenterSpatialOrigin()
```

```
image2.CenterSpatialOrigin()
image1.ApplyIP(alg.DFT())
image2.ApplyIP(alg.DFT())
diff_image=ost.img.CreateImage(image1.GetExtent())
for pixel in image1.GetExtent():
  phase1=ost.img.Phase(image1.GetComplex(pixel))
  phase2=ost.img.Phase(image2.GetComplex(pixel))
  phase_diff=phase1-phase2
  diff_image.SetReal(pixel,180.0*float(phase_diff)/math.pi)
v=Viewer(diff_image)
v.SetName("Phase difference (in degrees)")
```

4.2.4 Extracting a subimage

Usage

giplt extract.py

Description

This script can be used to extract a subimage either using the graphical selection in the image viewer or by entering the top-left and bottom-right coordinates of the subimage manually.

Script

```
from iplt.gui.inputdialog import *


def load_image():
    global image,viewer
    load_dialog=InputDialog("Load image")
    load_dialog.AddPath("Filename:","open")
    if load_dialog.exec_():
        image=ost.io.LoadImage(load_dialog.GetData()[0])
        viewer=Viewer(image)
        return True
    return False

def extract_graphical():
    global image,viewer
    image=image.Extract(viewer.GetSelection())
    viewer=Viewer(image)
    return True

def extract_numerical():
    global image,viewer
    ext=image.GetExtent()
    range_dialog=InputDialog("Extract range:")
    range_dialog.AddInt("x1: ",ext.GetStart()[0],ext.GetEnd()[0],ext.GetStart()[0])
    range_dialog.AddInt("y1: ",ext.GetStart()[1],ext.GetEnd()[1],ext.GetStart()[1])
    range_dialog.AddInt("x2: ",ext.GetStart()[0],ext.GetEnd()[0],ext.GetEnd()[0])
    range_dialog.AddInt("y2: ",ext.GetStart()[1],ext.GetEnd()[1],ext.GetEnd()[1])
    if range_dialog.exec_():
        data=range_dialog.GetData()
        image=image.Extract(ost.img.Extent(ost.img.Point(data[0],data[1]),ost.img.Point(data[2],da
        viewer=Viewer(image)
        return True
    return False
```

```
def save_image():
    global image,viewer
    save_dialog=InputDialog("Save image")
    save_dialog.AddPath("Filename: ","save")
    if save_dialog.exec_():
        ost.io.SaveImage(image,save_dialog.GetData()[0])
        return True
    return False

dock = QWidget()
vb=QVBoxLayout()

pb=QPushButton('Load image')
QObject.connect(pb,SIGNAL("clicked()"),load_image)
vb.addWidget(pb)

pb=QPushButton('Take subimage extent form selection')
QObject.connect(pb,SIGNAL("clicked()"),extract_graphical)
vb.addWidget(pb)

pb=QPushButton('Enter subimage extent manually')
QObject.connect(pb,SIGNAL("clicked()"),extract_numerical)
vb.addWidget(pb)

pb=QPushButton('Save image')
QObject.connect(pb,SIGNAL("clicked()"),save_image)
vb.addWidget(pb)

dock.setLayout(vb)
dockwidget=ost.gui.Widget(dock)
app=ost.gui.GostyApp.Instance()
app.perspective.panels.AddWidgetToPool("Subimage extraction",dockwidget)
app.perspective.panels.AddWidget(ost.gui.PanelPosition.LEFT_PANEL, dockwidget)
dock.show()
```

4.2.5 Generate Spoke Pattern image

Usage

Edit spoke_pattern.py using any text editor

then

giplt `spoke_pattern.py` <output image>

Description

This `script` can be used to create a "spoke pattern" image in the style of the ones featured in the following paper:

Downing K.H., Glaeser R.M., Restoration of weak phase-contrast images recorded with a high degree of defocus: The "twin image" problem associated with CTF correction, **Ultramicroscopy 108 (2008)**, 921-928, PMID: 18508199[5], PMCID: PMC2694513[6]

All tweakable parameters are defined in the first lines of the script, which can be edited before the script is executed. The script generates the image, opens a viewer to show it and saves it in the current folder with the name specified by the user. See figure 4.20 for a sample image.

[5] http://www.ncbi.nlm.nih.gov/pubmed/18508199
[6] http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2694513

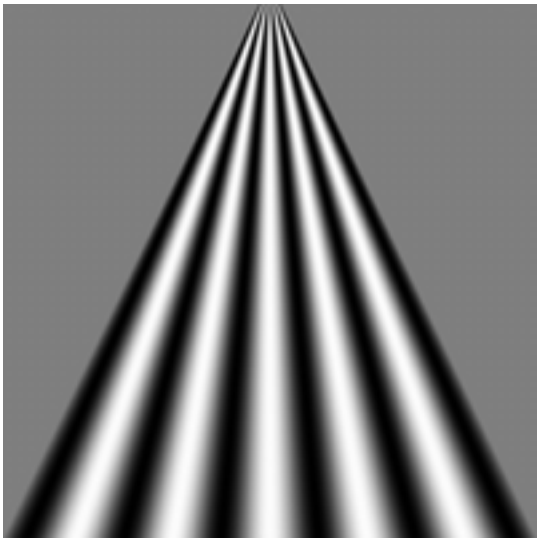

Figure 4.20: Spoke pattern

Script

```
import math
import sys
from ost.img import *

# Tweakable parameters
wedge_start = 80 # Width of wedge at the narrower point
wedge_end = 2000 # Width of wedge at the broader point
number_of_bands = 11 # Number of alternating bands (must be odd)
size_of_image_x = 2000 # Size of the image (X)
size_of_image_y = 2000 # Size of the image (Y)
pixel_sampling = 1.0 * Units.A # Pixel width
threshold = 2.0 * Units.A # Threshold for low pass filtering
amplitude = 255 # amplitude of the obscillations

# Image is created
image=CreateImage(Size(size_of_image_x,size_of_image_y))
image.CenterSpatialOrigin()
image.SetSpatialSampling(pixel_sampling)

# For a cleaner code, 4 shorthands are defined
image_extent=image.GetExtent()
start_y=image_extent.GetStart()[1]
end_y=image_extent.GetEnd()[1]
start_x=image_extent.GetStart()[0]
end_x=image_extent.GetEnd()[0]

# Wedge is written in the image
for y in range (start_y,end_y+1):
  wedge_width=wedge_start+(y-start_y)*(wedge_end-wedge_start)/(end_y-start_y)
  for x in range (start_x,end_x+1):
    half_wedge=wedge_width/2.0
    outer_bands=(number_of_bands-1)/2
    factor=(0.5+outer_bands)*math.pi/half_wedge
    if float(x)>-half_wedge and float(x)<half_wedge:
      value=255*math.cos(float(x)*factor)
      image.SetReal(Point(x,y),value)

# Image is low-pass filtered
filter=alg.GaussianLowPassFilter(threshold)
```

```
image.ApplyIP(filter)

# Image is saved
ost.io.SaveImage(image,sys.argv[1])

# Viewer is launched to show the result
Viewer(image)
```

4.2.6 FFT search

Howto search a diffracting region with the live FFT, and then save that region as a new image.

- Start up `giplt` and then use the following two commands to load the image and display it in a viewer:

```
i=ost.io.LoadImage("imagename.tif")
v=Viewer(i)
```

- In the viewer, right click on the "Overlays" title and select the *FFT* subwindow.
- In the smaller, *FFT* viewer, make sure to select a subregion (not containing the origin!) and re-normalize (left-click and drag with the mouse, then press *N*, just as you would in the main image viewer). This may need to be repeated after the following steps.
- Right-click on the *FFT* image to access the submenu, where you can set its size (bottom menu point). Note that depending on your local workstation, sizes above 256 or 512 may slow things down.
- You can convert the *FFT* viewer into an independent top-level window by dragging it on its title frame, and then resize this window if your FFT size is too large.
- As you move the mouse around in the main viewer, and keep the *Shift* key pressed, the *FFT* window will update accordingly. Or, simply move the mouse to a region, and press *Shift* briefly to get an update of the *FFT* window only at that time.
- Once you have found the region you would like to extract from the original image, make a single click with the left mouse button at the center of that region, then use this command sequence to cut out this subregion and save it (adjusting the size and filename as required)

```
s = i.Extract(Extent(Size(1024,1024),Point(v.GetClickedPosition())))
SaveImage(s,"cut_out.tif")
```

4.3 User contributed examples

4.3.1 SIRA calibration

Author: John Minter (jrminter at rochester dot rr dot com)

Precise calibration of electron images is greatly aided by IPLT. The following script was developed to calibrate images of the SIRA 2160 line/mm grating. The basic script was patterned after Andreas Schenk's extract subimage tutorial and the examples in the ex module. Execute the script with 'giplt analyze_sira.py' to display the main dialog (figure 4.21).

Notice the task-bar on the left. Press the 'Load Image' button and choose the desired image file from the dialog. The routine displays the image (figure 4.22).

The routine displays the full image and computes the largest FFT that will fit in the image. Pressing the 'Get Spacing' dialog (figure 4.23) displays a dialog that presents the default spacing for the 2160line/mm grating and lets the user change it if a different spacing was used.

Pressing the "Analyze Image" crops a region from the center of the image (figure 4.24)

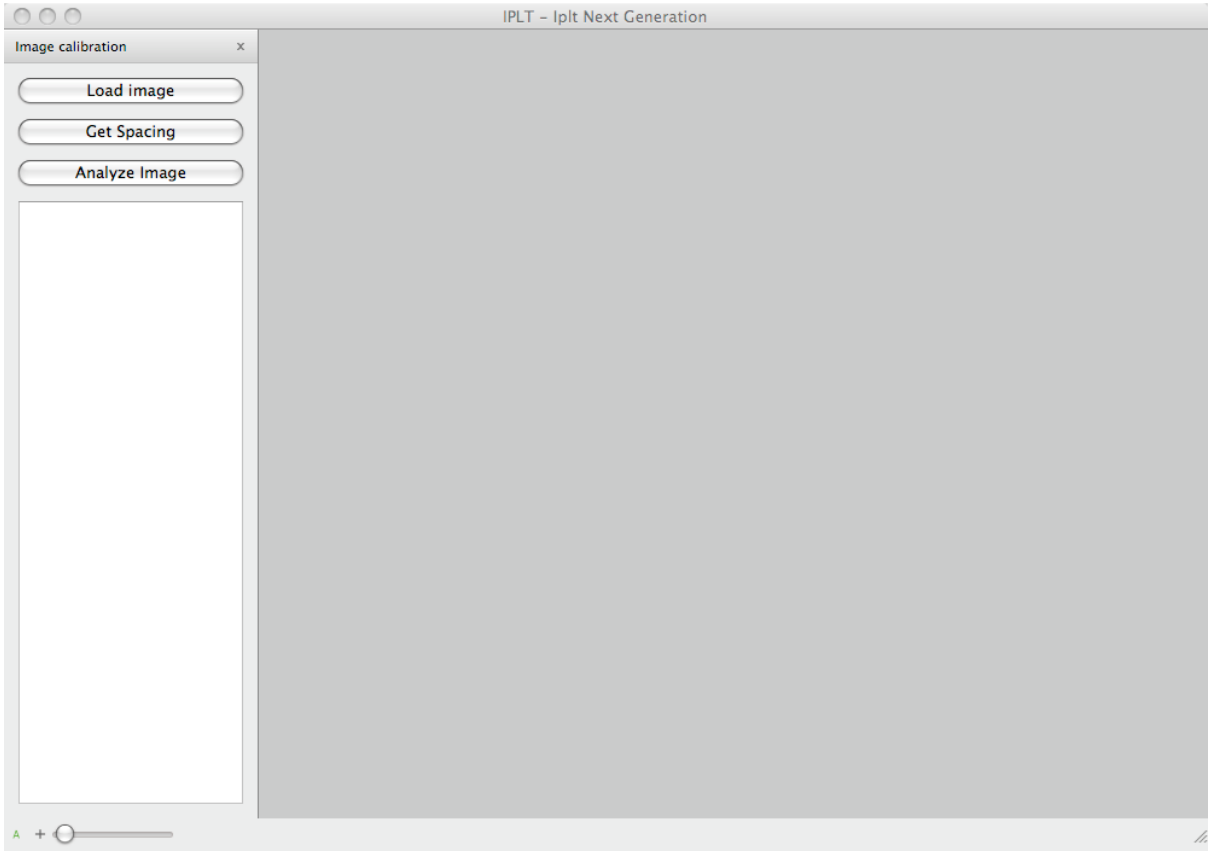

Figure 4.21: Main dialog

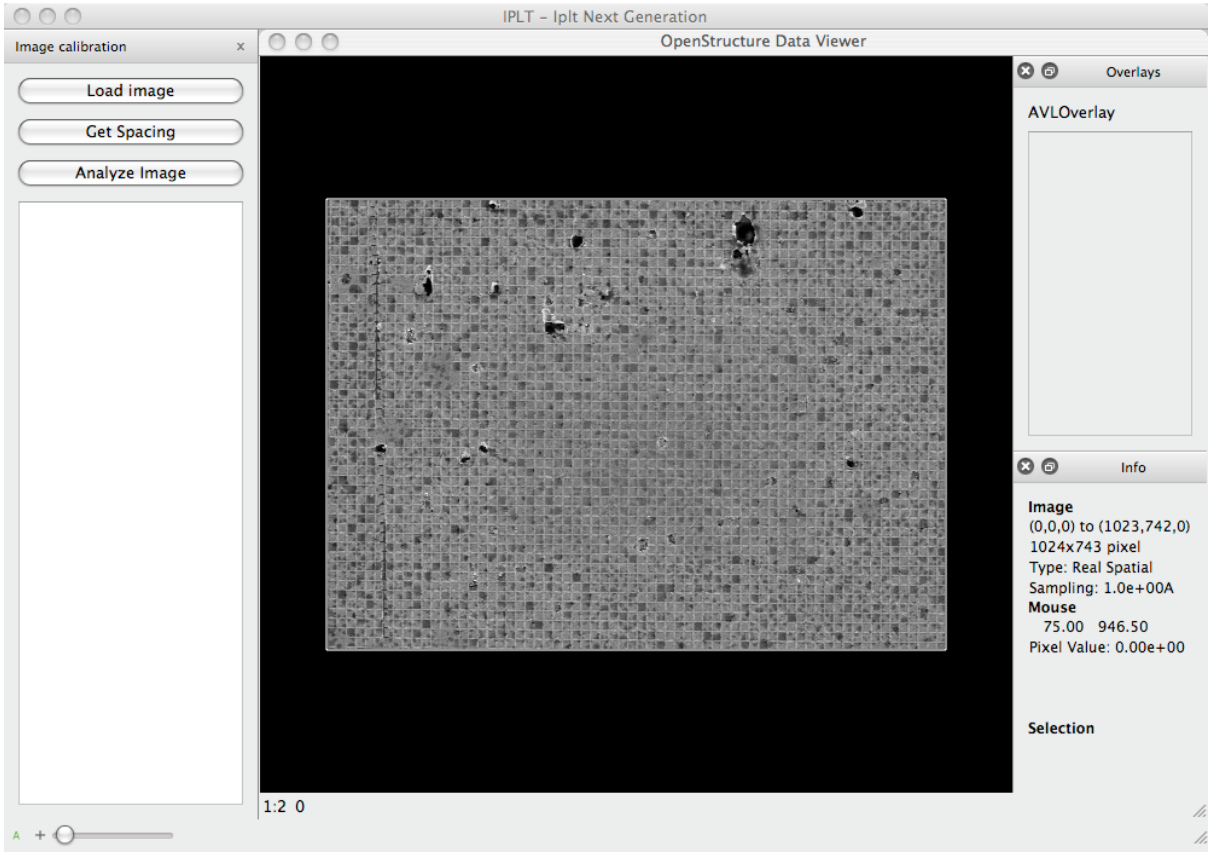

Figure 4.22: SIRA displaying image

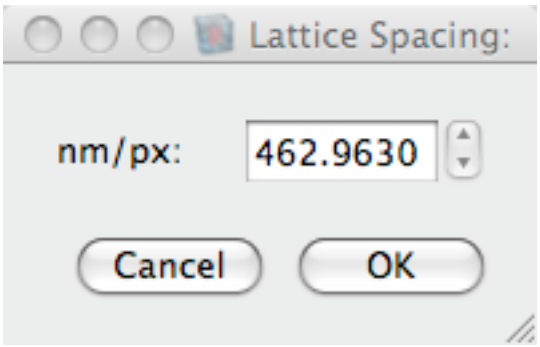

Figure 4.23: Get spacing dialog

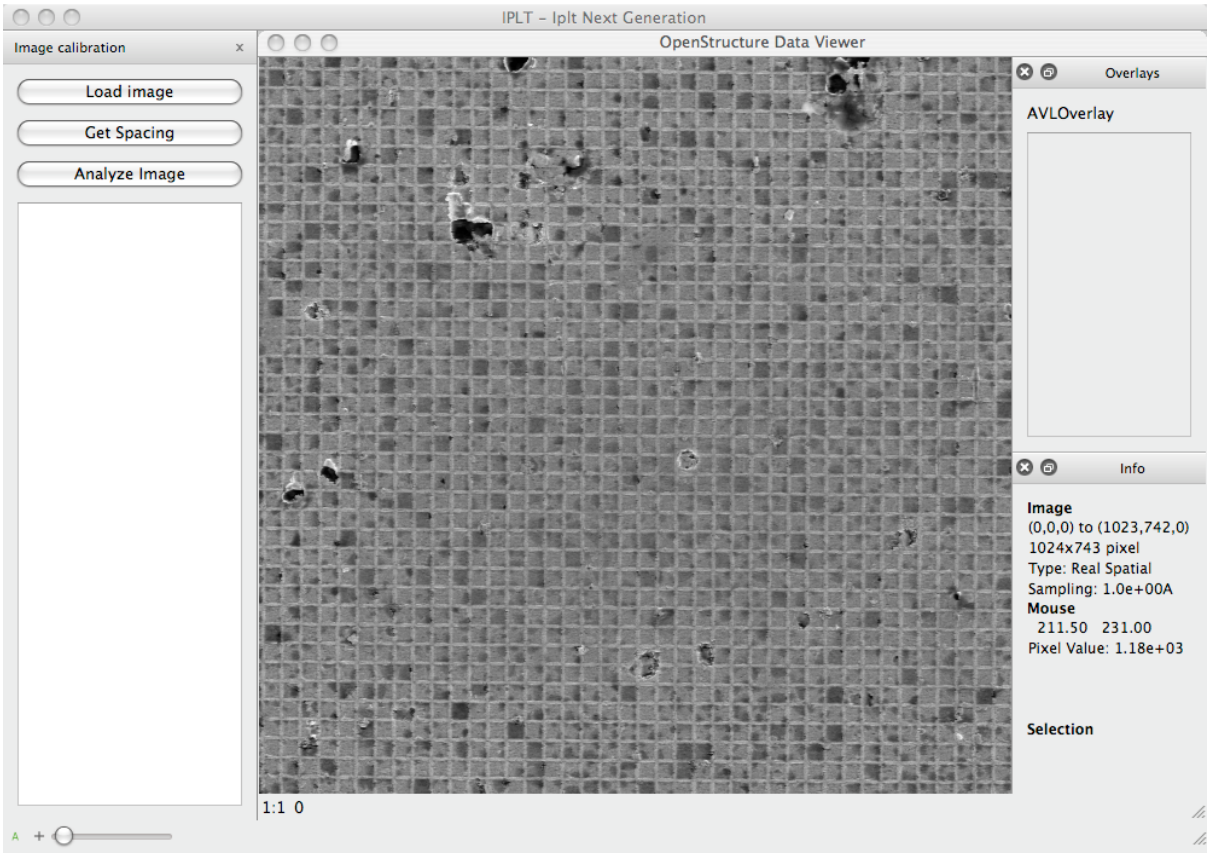

Figure 4.24: Cropped image

and computes the power spectrum, and fits a lattice (figure 4.25). One can normalize the display with the usual commands by selecting a region and pressing N. It also outputs the results to the result window window and to a file with the image file name with '_ana.txt' appended.

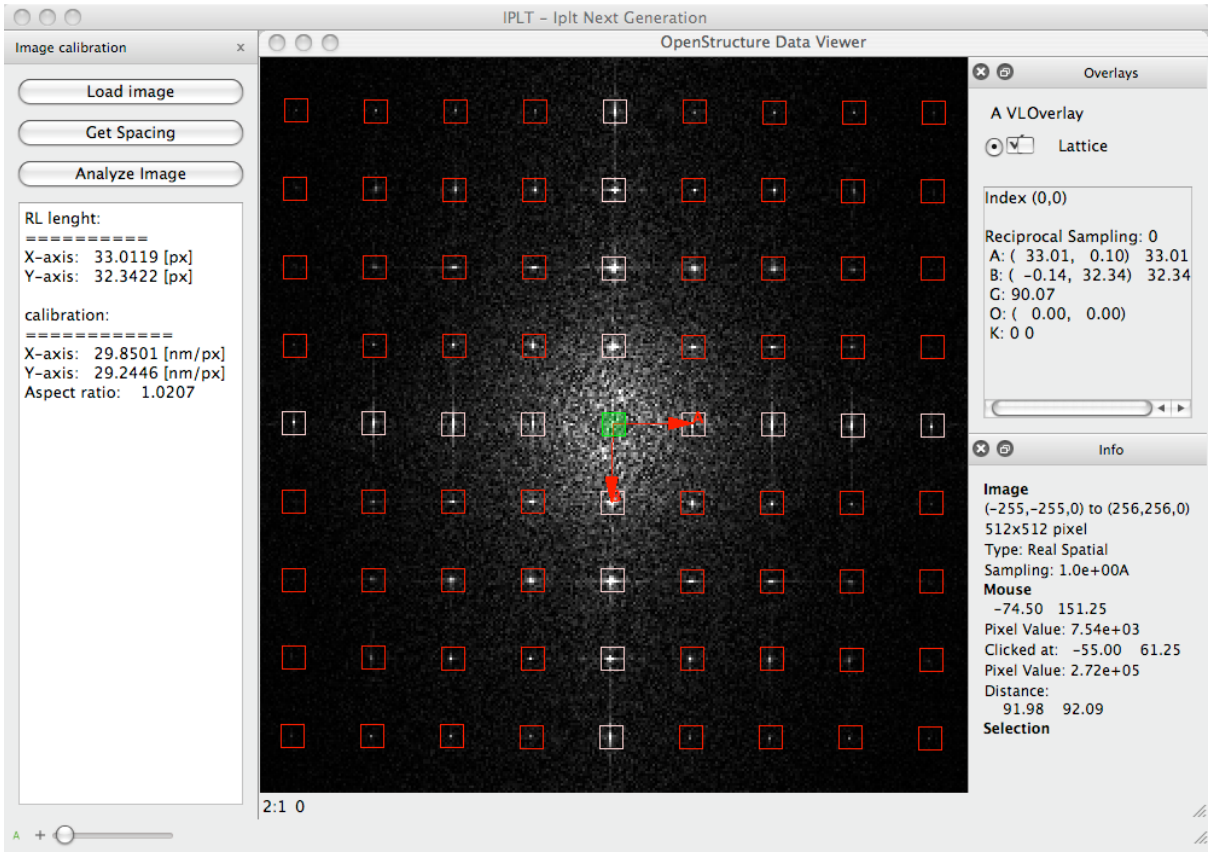

Figure 4.25: Fourier transform

see the source code `analyze_sira.py` and the test image `sira.tif`.

CHAPTER

FIVE

DIFFRACTION PROCESSING

5.1 Introduction

The IPLT diffraction processing pipeline provides an automated way to process and merge electron diffraction data sets. I can be accessed either from the command line using the two command *iplt_diff* and *iplt_diff_merge* or the GUI using the command *giplt_diff_manager*.

The following sections will give a introcution of the diffraction data extraction and merging. A more in depth explanation of the algorithms involved in the diffraction processing can be found in:

- Schenk AD, Philippsen A, Engel A, Walz T. *A pipeline for comprehensive and automated processing of electron diffraction data in IPLT.* **J Struct Biol.** 2013 link[1]

5.2 Project creation

5.2.1 Welcome screen

If the diffraction manager (giplt_diff_manager) is started within a directory containg an IPLT diffraction project, it displays the project. Otherwise it display a welcome screen allowing the user to either select an existing project or create a project.

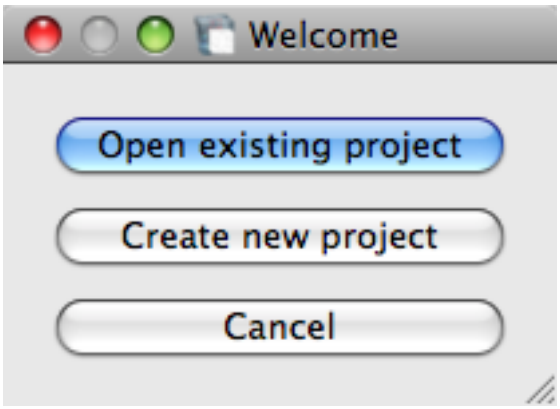

Figure 5.1: Welcome screen

5.2.2 Project creation

The user is guided through the project creation by a wizard. the following parameters have to be provided by the user:

[1] http://dx.doi.org/10.1016/j.jsb.2013.02.017

- **Path:** Directory where all the project data will be stored.
- **Raw image format:** File format of the recorded raw images.
- **Project prefix:** Three letter code identifying the project.
- **Spacegroup:** Crystallographic spacegroup of the recorded crystals.
- **A:** Length of the first unit cell axis.
- **B:** Length of the second unit cell axis.
- **Gamma:** Angle between the unit cell axes in degrees.
- **Thickness:** Thickness of the unit cell.

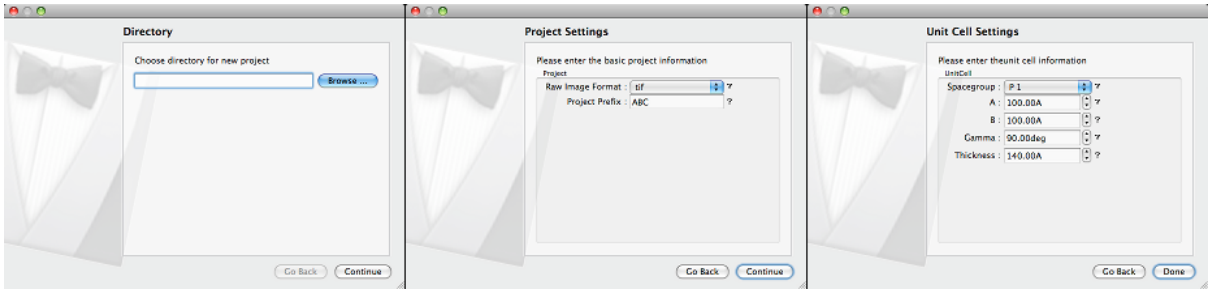

Figure 5.2: Project creation wizard

5.3 Project management

The IPLT diffraction manager consists of three main panels. The first panel (figure 5.3) contains all project-wide parameters and information. It also contains the controls for the very high-level project-wide data processing.

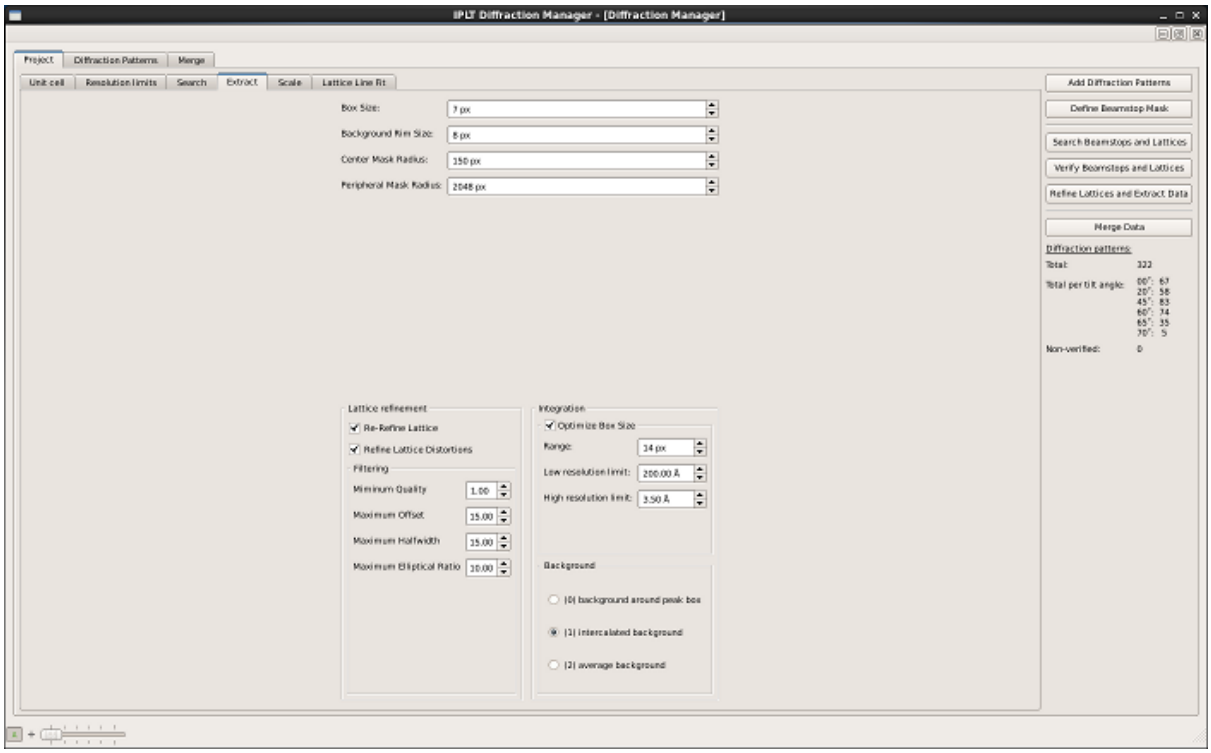

Figure 5.3: Project panel

5.3.1 Add diffraction patterns

New diffraction patterns can be addded at any time during a project by using the "Add diffraction patterns button". The file browser allows to select single or multiple (by pressing `Control` and `Shift`) diffration patterns. Once a set of diffraction patterns are opened, the import dialog (see figure 5.4) will be shown, where one can check if the nominal tilt angle and image number were correctly determined for all files. IPLT uses the file name format AAAXXYYYY, where AAA is a three-letter prefix unique for each project, XX is the nominal tilt angle of the corresponding pattern and YYYY is the image number. If the original files don't follow this naming convention, IPLT will try to guess the correct nominal tilt angle and image number. The tilt angle and image number for a pattern can be modified by double clicking on the numbers in the tilt angle and image number column. Alternatively the nominal tilt angle can be changed for all diffraction patterns and they can be renumbered using the buttons on the top.

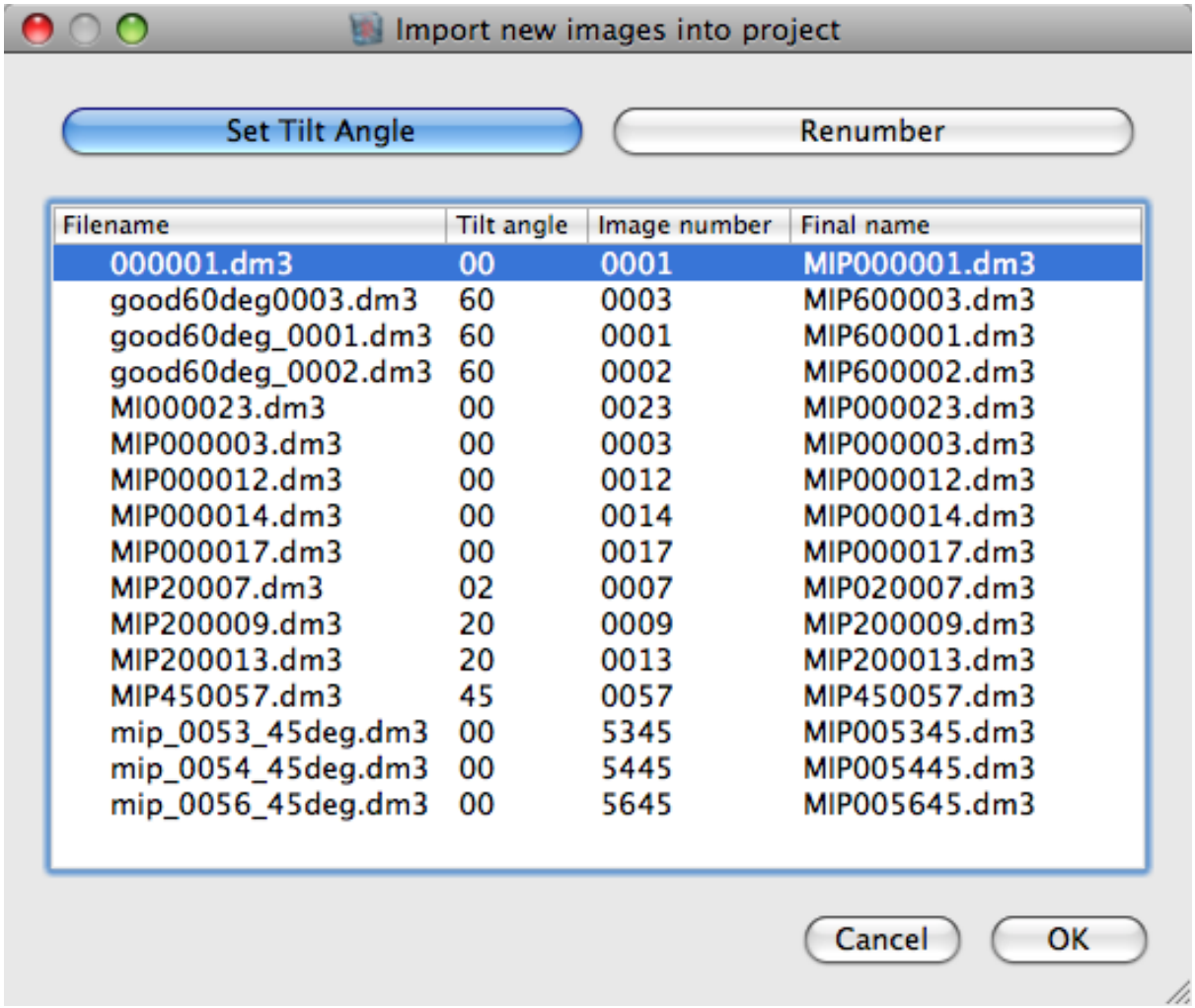

Figure 5.4: Adding diffraction patterns

5.3.2 Define beamstop mask

The shape of the beam stop has to be defined manually. Later on this information is used to automatically detect the beamstop position for each diffraction pattern. Selecting "Define beamstop mask" opens a Data Viewer displaying the first diffraction pattern and a mask overlay. The mask overlay can be used to create new polygon mask or to modify and move an existing polygon mask. Every interaction with the mask overlay is done by pressing `Control` together with different keys or mouse buttons. On OS X the `Control` key is replaced by the `Command` key.

1. Creating a new mask
   - `Control+Shift+Left Mouse Button`: Add a new point to the masking polygon
   - `Control+A`: Accept the new mask
   - `Control+R`: Reject the new mask
2. Modifying an existing mask
   - `Control+Left Mouse Button`: Move a point of the polygon
   - `Control+Right Mouse Button`: Move the whole mask

The mask overlay can be used to create a new polygon mask tracing the contour of the beamstop. The mask is saved by selecting "Save global beamstop mask".

5.3.3 Search beamstops and lattices

"Search beamstops and lattices" allows the automatic determination of the beam stop position and the crystal lattice. The diffraction manager automatically detects and skips patterns, for which these were already determined.

5.3.4 Verify beamstops and lattices

"Verify beamstops and lattices" displays all patterns with non-verified lattices for visual inspection and manual correction (if necessary).

5.3.5 Refine lattices and extract data

"Refine lattices and extract data" refines the lattice and extracts the reflection data for all non-processed diffraction patterns.

5.3.6 Merge data

Merge data creates an initial merged data set that can be further refined using the *merging panel*.

5.4 Diffraction data extraction

5.4.1 Overview

The diffraction data extraction process has four distinct steps. It comprises positioning of a beam stop mask, search for reflection peaks, assignment and subsequent refinement of the lattice, and background-corrected integration of the intensity of each reflection peak (figure 5.5).

5.4.2 Determination of the beam stop postion and masking of the beam stop

The polygonal beam stop shape that was defined during project setup is automatically positioned for each diffraction pattern to cover the beam stop area and exlude refleciton peaks too close to the beam stop from further processing.

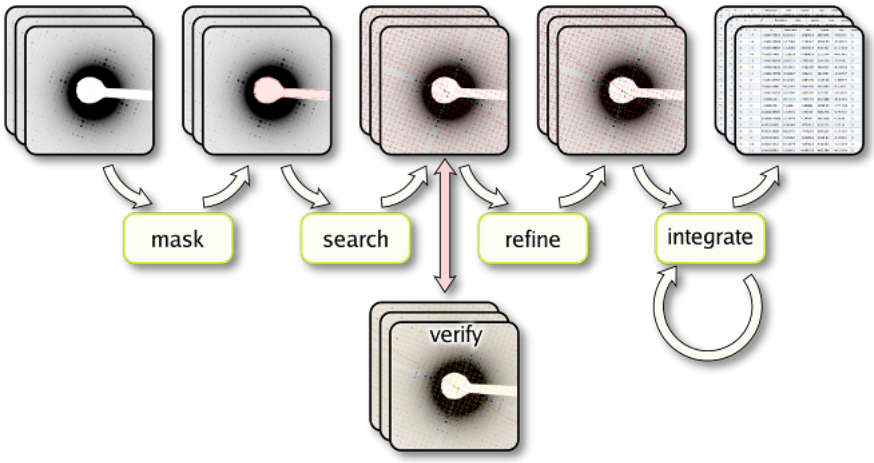

Figure 5.5: Data extraction overview

5.4.3 Lattice search

The second step in the diffraction data extraction is the automatic search of the lattice in the form $\vec{l} = \vec{o} + h\vec{a} + k\vec{b}$. Based on the the strongest diffraction reflections an initial lattice is determined automatically. The correct indexing of the lattice is determined based on the information about the nominal tilt parameters known from data aquisition at the electron microscope. The lattice origin is determined based on the minimization of the difference between Friedel related reflection peaks.The correct indexing of the lattice can be verified at this point using the graphical user interface.

5.4.4 Lattice refinement

The initial lattice can further be refined by taking the peak position for all reflection peaks into account. In addition to the refinement of the lattice vectors and the lattice origin, the algorithm also provides the option to determine and refine parameters for the correction of barrel and spiral distortions affecting the lattice (see *lattice documentation* for explanation of the distortion parameters).

5.4.5 Data extraction

After lattice refinement, the reflection intensities are extracted by integration over the diffraction reflections and correction for the background contribution. The radius of the integration area is automatically optimized to minimize the RFriedel factor for each diffraction pattern.

5.5 Diffraction data merging

5.5.1 Overview

Once all individual diffraction patterns have been processed, each reflection of each pattern is characterized by a duplet of **(h,k)**, the crystallographic lattice index of the reflection. The integration step assigned an intensity **(iobs)** and an associated error estimation **(sigiobs)** to each reflection; a subsequent step used the tilt geometry to calculate the **z* (zstar)** value as well as its error estimation **(sigmazstar)**.

The merge process is responsible for first combining reflections from the individual patterns and then discretizing the irregularly spaced reflections to obtain a full crystallographic dataset with reflections organized in **(h,k,l)** triplets. At this point, it is very useful to think of the combined data as they distribute into lattice lines- ultimately, it will be those lattice lines which will allow the **z*** to **l** discretization. Generally speaking, each diffraction pattern

contributes to each lattice line, as explained in figure 5.6. The merge processing goes back and forth between the lattice line and individual pattern representation, and thus it is important to understand how they relate to each other.

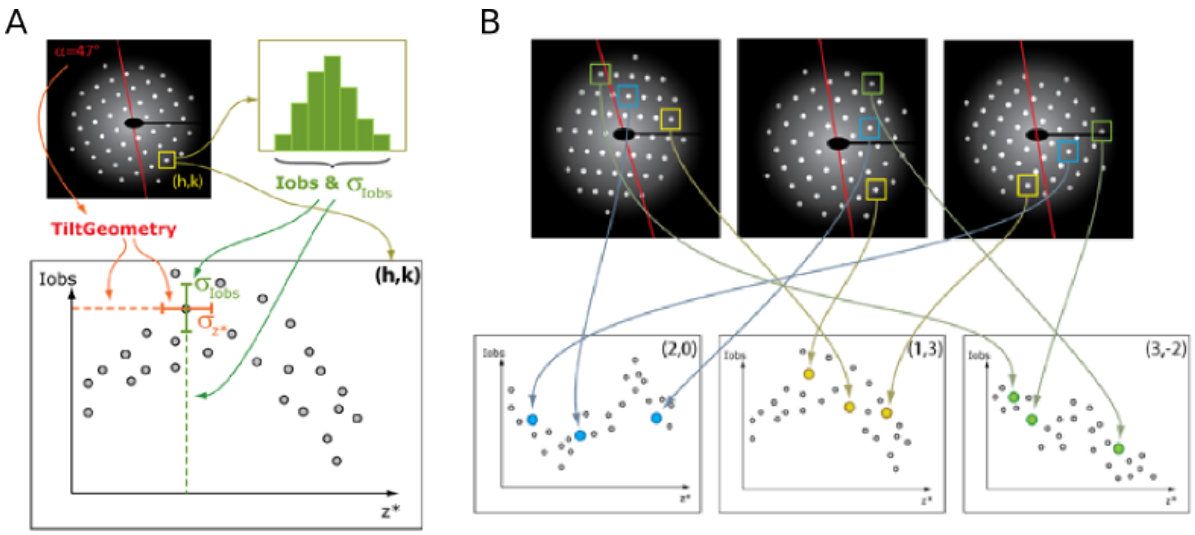

Figure 5.6: The relationship between diffraction patterns and lattice lines. (A) The contribution of a single diffraction pattern (top left) to a lattice line (bottom). A lattice line is a 2D plot, based on a particular (h,k) duplet. Its abscissa represents the z* value, which is based on the tilt geometry of the pattern. Its ordinate represents the intensity (iobs), which is determined during the integration step in the pattern processing. Estimated errors, both in x and y, play an important role in the merge processing. From the integration, an measure of uncertainty is obtained for the intensity (sigiobs). For estimating the z* error (sigmazstar), the uncertainty of the precise lattice vectors is used. (B)Each diffraction pattern (top row) will usually contribute to each lattice line (bottom row), as indicated with the three reflections at indices (2,0), (1,3) and (3,-2), in blue, yellow and green, respectively.

The merge process has four distinct steps. In the beginning, reflections from all datasets are pooled, and the crystallographic symmetry is applied. Then, a scaling algorithm takes account of the fact that each pattern is from an individual measurements, meaning that the integrated intensities cannot be just thrown together. Following this step is the tilt geometry refinement, which tweaks the z* values. Finally, the lattice line discretization models the scattered points of each lattice line as a continuous curve, and yields reference reflections at discretized values.

An overview over the merging process is given in figure 5.7 and a summary of this process from the perspective of a single lattice line is given in figure 5.8.

5.6 Graphical user interface

The graphical user interface (GUI) for the IPLT diffraction processing pipeline is started by using the iplt_diff_manager command. If the GUI is started in a folder containing a IPLT diffraction project, the project settings are automatically loaded. Otherwise the GUI presents the options to create a new project or to choose a project in a different folder.

The GUI is organized into three panels labeled project, diffraction pattern, and merge. At the bottom of the GUI a slider allows to set the verbosity level for the logging.

5.6.1 Project panel

The project panel displays information relevant to the entire project and allows changing project-wide parameters. It also allows addition of new diffraction patterns, definition of the global beam stop shape and project wide processing and merging of diffraction patterns.

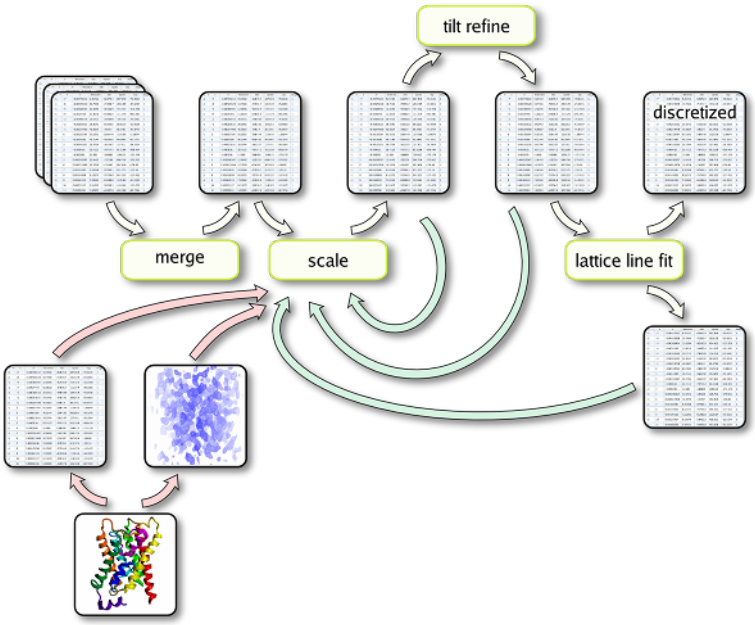

Figure 5.7: Merging overview

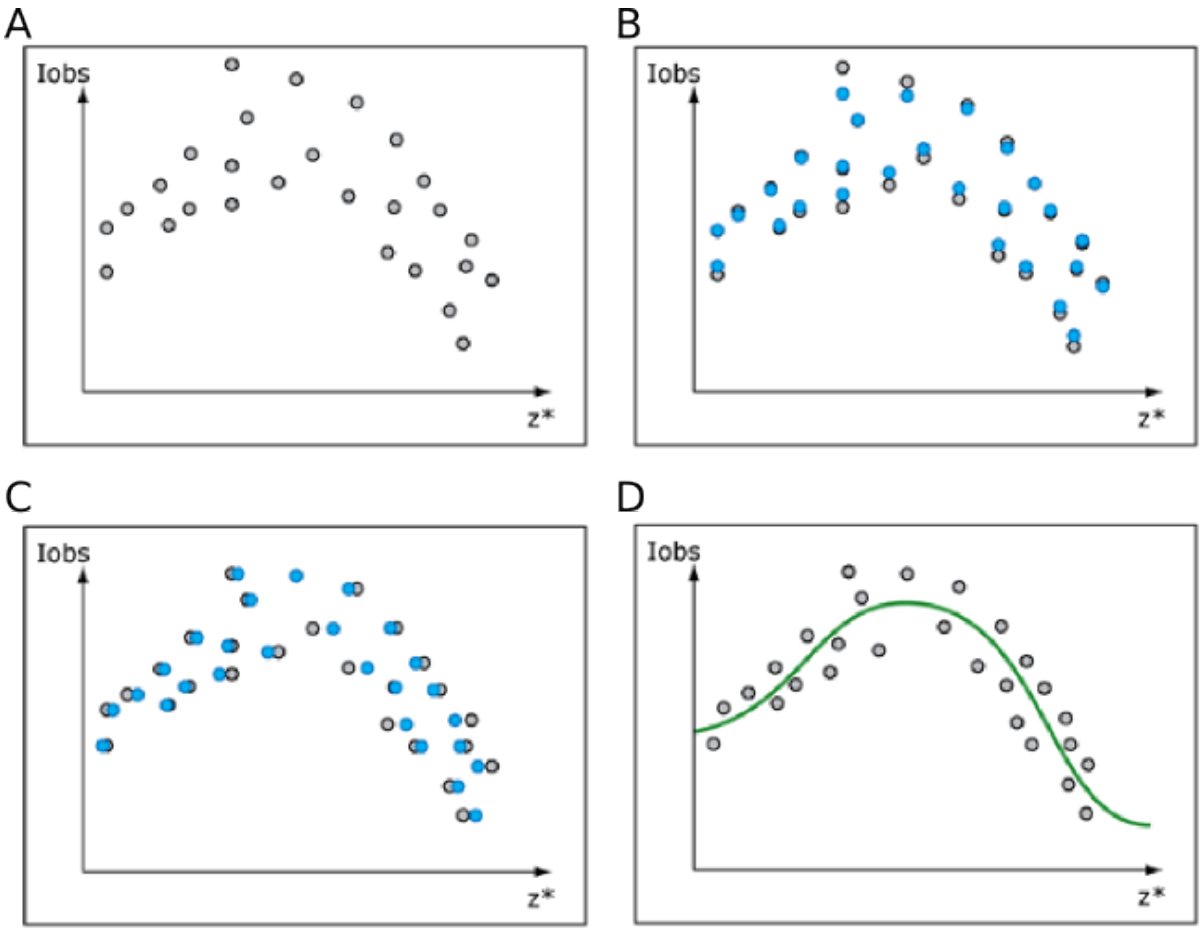

Figure 5.8: **Figure 3: Merge processing steps as seen from the lattice line perspective.** (A) First merge: scattering of lattice line values in both x- and y-direction. (B) Scaling: only Iobs are affected (y-axis). The change in the intensities (grey to blue points) may be different for each value, as they originate from different patterns. (C) Tilt geometry refinement: only z* values are affected (x-axis). (D) Lattice line fitting: the scaled and tilt-geometry-refined values are modeled by a curve, which then gives the discretized values.

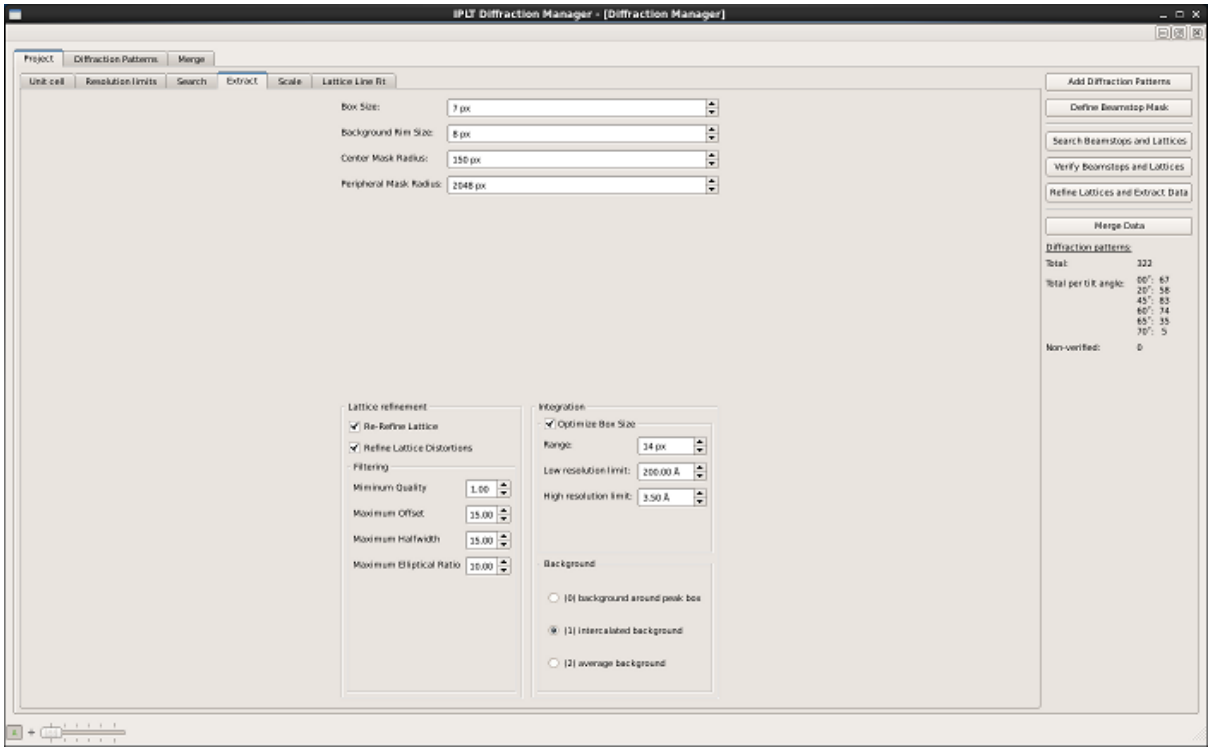

Figure 5.9: Project panel

5.6.2 Diffraction pattern panel

The diffraction panel displays a list of diffraction patterns on the left side, shows indicators for their processing status and displays RFriedel factors for fully processed diffration patterns. The checkboxes in the first column of the diffraction pattern list can be used to exclude patterns from processing and merging. The center part of the panel allows to override the default processing parameters for single diffraction patterns. The list of buttons to the right allows the user to run individual processing steps manually and to assess the processing results using the data viewer. The list on the bottom right gives access to the log files for the individual processing steps.

5.6.3 Merging panel

The merging panel allows the merging of the extracted diffraction data into a 3D data set. A new merge can be created clicking **New Merge Directory** and providing an unique suffix identifying the folder. The currently active merge and refinement round can be selected using the pulldown menus on the top left. The list of diffraction patterns on the left side displays the merging R-factors, the scale and B factors and the tilt geometry for each pattern for each round. The checkboxes allow the inclusion or exclusion of diffraction patterns from merging and refinement. The center part of the merging panel allows to override the processing parameters for the currently active merge. The list on the bottom right allows access to the merging and refienment log files. Graphical plots of the results from the scaling and lattice line fitting can be viewed by pressing **Check Scaling** and **Check Lattice Line Fit** respectively.

5.7 Command line interface

5.7.1 iplt_diff

This executable handles all steps of the diffraction processing, indicated by different *modes* or *commands*, as described in more detail below. It is run from the top-level directory that contains the diffraction patterns in individual subdirectories, as explained in `Project Setup`.

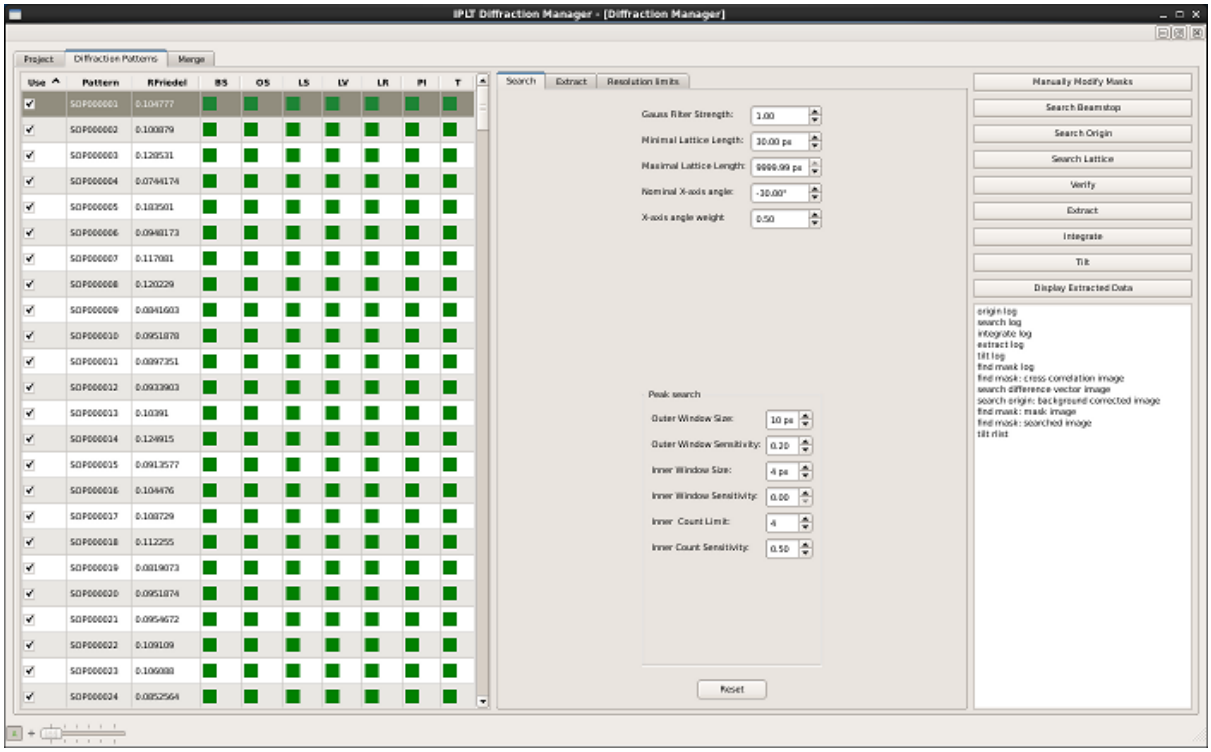

Figure 5.10: Diffraction pattern panel

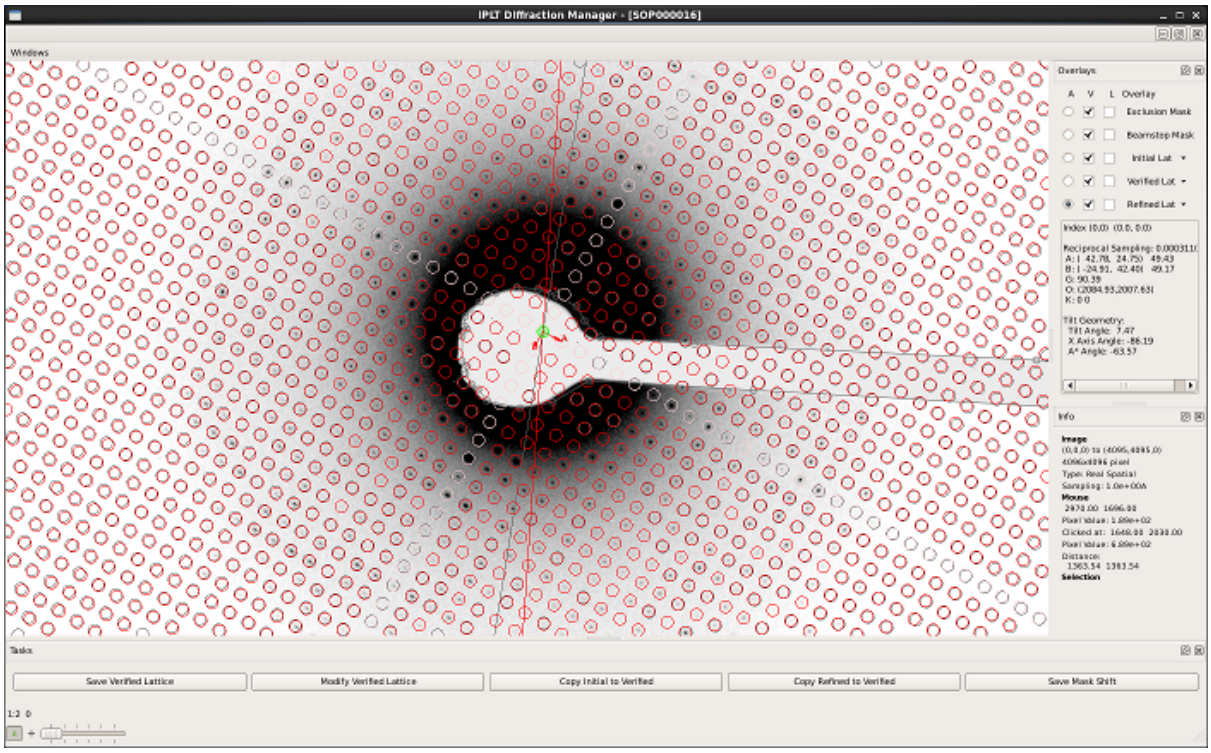

Figure 5.11: Data viewer

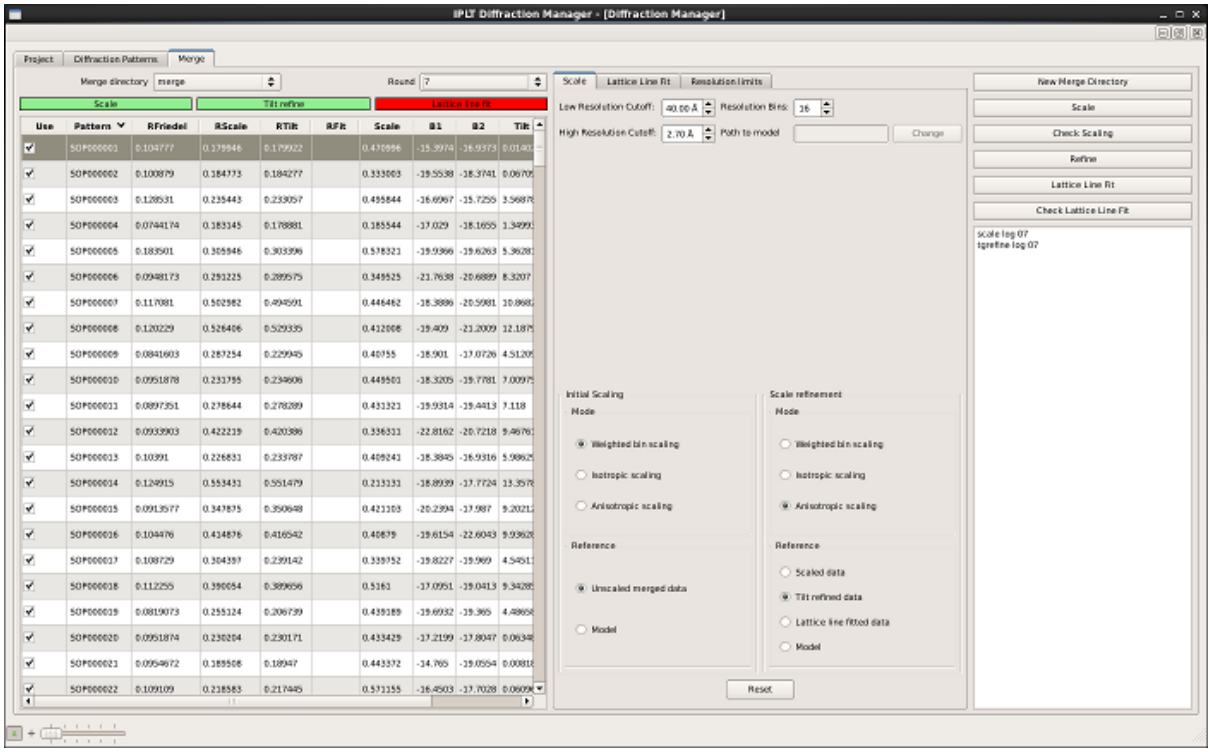

Figure 5.12: Merging panel

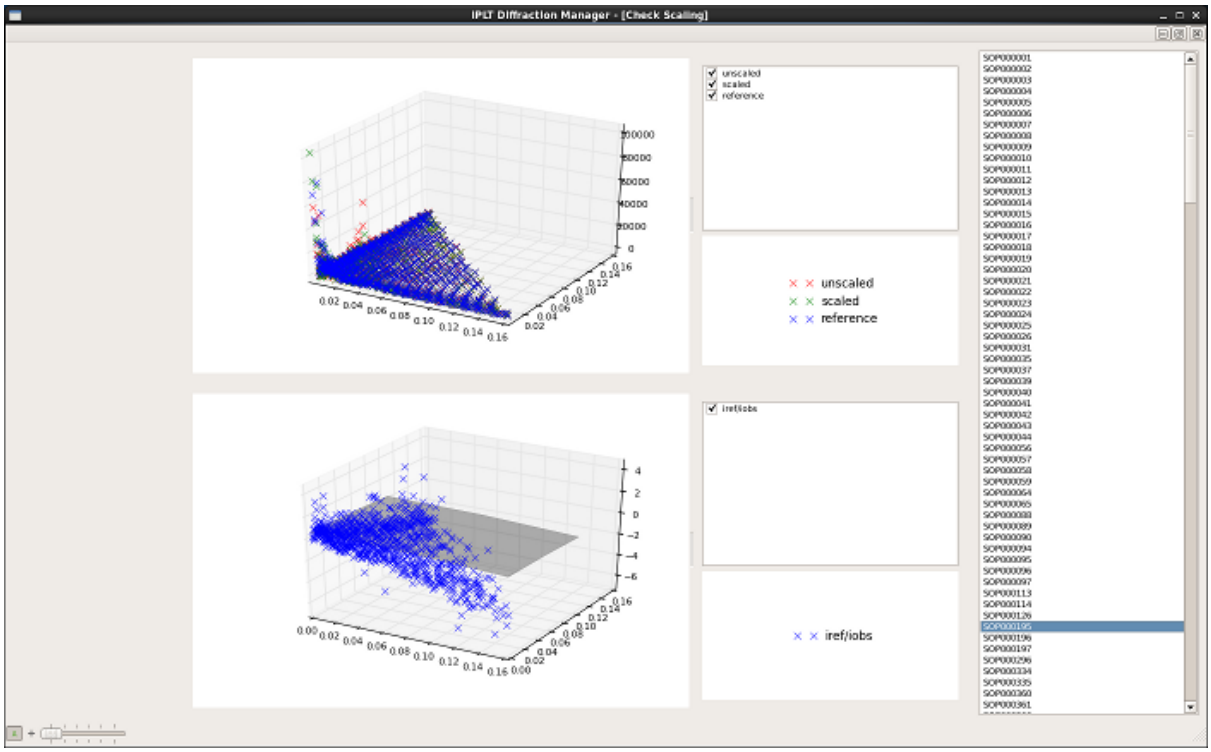

Figure 5.13: Scaling widget

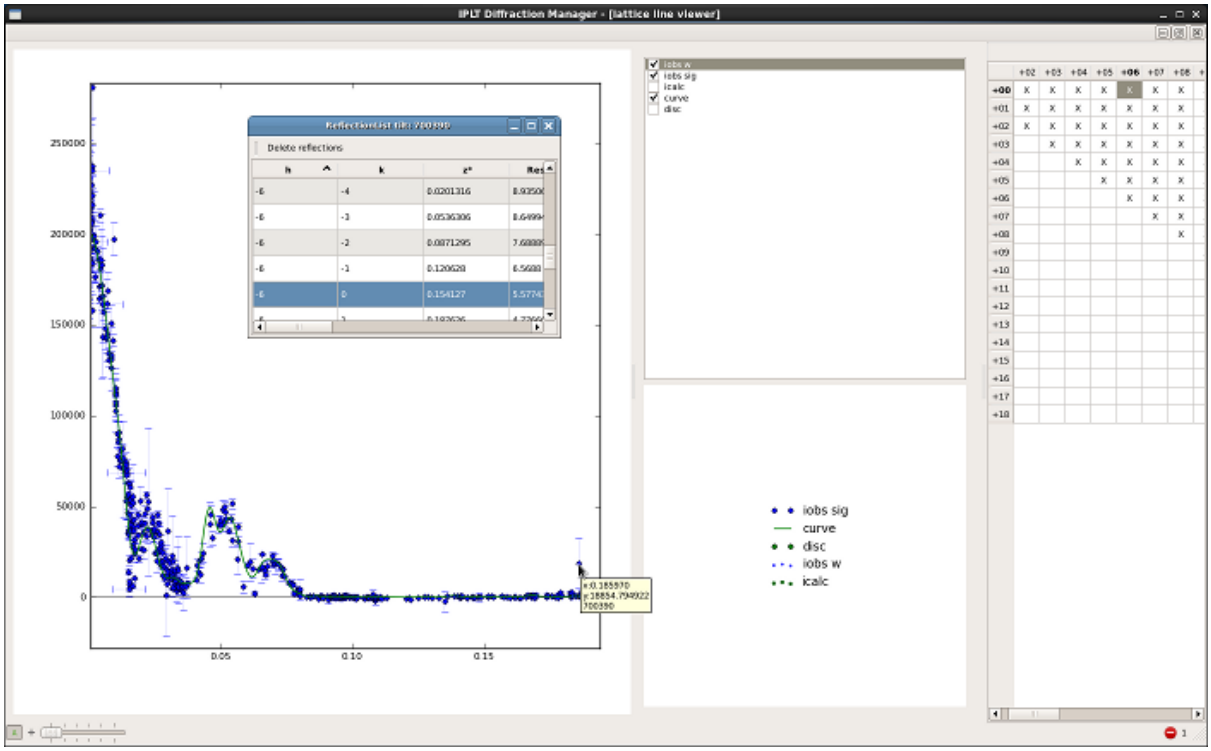

Figure 5.14: Lattice line viewer

NOTE: Prior to the extraction, a manual lattice verification step is recommended using the giplt_diff_manager.

Syntax

```
iplt_diff [OPTIONS] MODE [MODE_OPTIONS]
```

Iterate over *all* image sub-directories and run the particular processing step indicated by `MODE` if one of the following is true:

- that step hasn't been run yet (the resulting data files are not present)
- its input files are newer than its result files, or the input parameters in `info.xml` are newer than the output parameters.
- the processing step is forced with `-f` (see below)

If a required input files is missing, or an empty file named `ignore` is present in the image directory, that particular directory will always be skipped, irrespective if one or more of the above conditions holds.

Info Files

All modes will retrieve their options from either the top-level `project.xml` or the image specific `info.xml` files (in the image directory), under the top-level tag `DiffProcessing`. If an option appears twice, the one in `info.xml` takes precedence. Most modes also record their main results within `info.xml`, under the top-level tag `DiffProcessingResults`.

The item required by all modes is `DiffProcessing/ImageMask` in `project.xml`, which defines the regular expression mask to identify the image directories. Only those directories that match this mask will be traversed during processing.

General options

`-v N` or `--verbosity=N` set verbosity level to N (usually 0-5)

`-i MASK` or `--image-mask=MASK` use only image(s) that fit(s) MASK

`-f` or `--force` force re-processing of the particular step, even if the data files already exist (combine with `-i` to restrict this to some directories, otherwise *all* image directories will be re-processed)

Diffraction Processing Steps

A single diffraction pattern is processed in various sequential steps. These are applied with the various commands, listed here in order of logical usage:

1. init: setup directory
2. mask: beamstop mask handling
3. origin: determine origin
4. search: determine lattice
5. manual lattice verification step with `giplt_diff_manager`
6. extract: extract reflections
7. integrate: integrate reflections
8. tilt: determine tilt geometry

All general options apply, but some steps have additional options given below.

init

Syntax: `iplt_diff [GENERAL OPTIONS] init`

Creates info.xml file in all image directories and fills it with default options. If `info.xml` already exists, this step will be skipped, unless `-f` is used; in this case, the default options will be written into the existing file, leaving the rest of the content untouched.

mask

Syntax: `iplt_diff [GENERAL OPTIONS] mask`

Searches beamstop mask in diffraction pattern, based on the polygon definition in `project.xml`.

Output: `diff_mask.log` and mask shift in `info.xml`

origin

Syntax: `iplt_diff [GENERAL OPTIONS] [--fast_origin] origin`

Determines the origin, i.e. the location of the (0,0) reflection. The `--fast_origin` option uses a reduced set of background values to speed up the algorithm, but may deliver less reliable results. *NOTE: This algorithm is currently borked, please expect wrong identification of the origin, you can fix this during the manual lattice verification step.*

Output: `diff_ori.log` and origin in `info.xml`

search

Syntax: `iplt_diff [GENERAL OPTIONS] search`

Searches for a lattice, using the previously determined origin, and saves it as the initial lattice.

Input: diffraction pattern as `IMGID.tif` file in `IMGID` directory and origin from previous step.

Output: `diff_search.log` and an `InitialLattice` in `info.xml`

manual lattice verification step

Use `giplt_diff_manager` to convert initial lattices into verified ones.

extract

Syntax: `iplt_diff [GENERAL OPTIONS] [--assume-verified-lattice] [--ignore-refined-lattice] extract`

Runs the main reflection extraction routine, based on a manually verified lattice or previously refined lattice. If neither a verified or refined lattice is present, or the initial lattice is newer than these two, the extraction will not be run. This can be overruled with the `--assume-verified-lattice` option, which will accept the initial lattice as the verified one (not recommended). As the name suggests, `--ignore-refined-lattice` will cause the extract routine to ignore a refined lattice and always use the verified one instead.

Input: diffraction pattern as `IMGID.tif` file in `IMGID` directory and `VerifiedLattice` or `RefinedLattice` in `info.xml`

Output: `IMGID_ext.mtz` `diff_extract.log`, and refined lattice in `info.xml`

integrate

Syntax: `iplt_diff [GENERAL OPTIONS] integrate`

Converts the information collected during the extract run into `iobs` and `sigiobs`, i.e. integrates the peak intensities.

Output: `IMGID_int.mtz` and `diff_integrate.log`

tilt

Syntax: `iplt_diff [GENERAL OPTIONS] [--ignore-verified-tilt] tilt`

Determines the reciprocal z values for each reflection, either based on a tilt geometry calculated from the unit cell and the refined lattice, or taken from a verified tilt geometry in `info.xml`. The option `--ignore-verified-tilt` will deactive the latter behaviour, and cause the tilt geometry to be always re-calculated from the verified lattice and unit cell.

Input: `IMGID_int.mtz`

Output: `IMGID_tilt.mtz` `diff_tilt.log` and `InitialTiltGeometry` in `info.xml`

5.7.2 iplt_diff_merge

Syntax

```
iplt_diff_merge COMMAND [-w DIR] [-c CYCLE] [-l POINT] [-v VERB] [-d SDIR1
[-d SDIR2]]
```

Iterate over all image directories in the current dir, plus all directories added with the `-d` option, and perform one of the commands for merging (see below). The parameters are:

`-w DIR`: store merging results in `DIR` (defaults to `./merge`). An `info.xml` file in this directory will be imported and overrides parameters from `project.xml`.

`-c CYCLE`: choose refinement cycle, defaults to `0`

`-v VERB`: verbosity level, default is `3`

COMMAND is one of:

scale

Scale each data set against the merged one. The merged data set is created on the fly.

Output: `merge_mergedNN.mtz` `merge_scaledNN.mtz`

where `NN` is the cycle number

tgrefine

Refine scaling and/or tilt geometry.

Input: `merge_scaledNN.mtz`

Output: `merge_tgrefNN.mtz`

where `NN` is the cycle number

llfit

Run the fitting and discretization algorithm for each lattice line.

Input: `merge_tgrefNN.mtz`

Output: `merge_llfitNN.mtz` `merge_discNN.mtz` `merge_curveNN.mtz`

where `NN` is the cycle number

all

all of the above, in their logical order `scale` `tgrefine` `llfit`

xml parameters

The following xml tags are recognized in the `project.xml` file, or in an `info.xml` file in the merge destination dir

```
<Scale>
  <!-- resolution limits -->
  <Resolution>
    <low value="40" type="float"/>
    <high value="2.5" type="float"/>
  </Resolution>
  <!-- number of bins for the weighted bin scaling -->
  <BinCount value="16" type="int"/>
  <!-- cutoffs for the filtering step after scaling, based on the weighted bins -->
  <WBinISigICutoff value="0.1" type="float"/>
  <WBinSigmaCutoff value="10.0" type="float"/>
  <!-- minimum tilt angle (in degree) for which to invoke an anisotropic pre-scaling-->
```

```
  <AnisoMinTilt value="90" type="float"/>
</Scale>

<Refine>
   <!-- lattice line curve fitting mode 0 (based on Imax and z*max) or 1 (bases on sigmaI and sig
   <CurveFitMode value="0" type="int"/>
   <!-- zstar fudge factor, experimental -->
   <ZWeight value="8.0" type="float"/>
   <!-- flag to turn intensity scale refinement on or off -->
   <RefineScale value="1" type="bool"/>
   <!-- flag to turn tilt geometry refinement on or off -->
   <RefineTilt value="1" type="bool"/>
   <!-- resolution limits to determine new tilt geometry -->
   <RefineTiltRLow value="10" type="float"/>
   <RefineTiltRHigh value="4" type="float"/>
 </Refine>

<LLFit>
  <!-- experimental weighting algorithms, use mode 1 and cutoff 0.0 for all practical purposes --
  <Weighting>
    <T1 value="10.0" type="float"/>
    <T2 value="40.0" type="float"/>
    <Mode value="1" type="int"/>
    <Cutoff value="0.0" type="float"/>
  </Weighting>
  <!-- iteration limit for the internal non-linear fitting routine -->
  <MaxIterations value="1000" type="int"/>
  <!-- iteration precision for the internal non-linear fitting routine -->
  <IterationPrecision value="1e-2" type="float"/>
  <!-- bootstrapping mode for improved sigma generation, recommended -->
  <BootstrapFlag value="1" type="bool"/>
  <BootstrapIterations value="20" type="int"/>
  <!-- initial guess mode prior to fitting; 0=from weighted bins, 1=random phase, unit amplitude,
  <GuessMode value="0" type="int"/>
  <!-- multiple iterations with different starting values, not as powerful as bootstrapping, best
  <Iterations value="1" type="int"/>
  <!-- phantom mode includes two additional values at +- sigma z* -->
  <PhantomMode value="0" type="bool"/>
</LLFit>
```

CHAPTER

SIX

PUBLICATIONS

There are two publications describing the motivation, architecture, and current state of IPLT:

- Philippsen A, Schenk AD, Signorell GA, Mariani V, Berneche S, Engel A. *Collaborative EM image processing with the IPLT image processing library and toolbox* **J Struct Biol.** 2007 Jan;157(1):28-37 link[1]
- Philippsen A, Schenk AD, Stahlberg H, Engel A. *IPLT: image processing library and toolkit for the electron microscopy community.* **J Struct Biol.** 2003 Oct-Nov;144(1-2):4-12. link[2]

A third publication describes the IPLT diffraction processing pipeline:

- Schenk AD, Philippsen A, Engel A, Walz T. *A pipeline for comprehensive and automated processing of electron diffraction data in IPLT.* **J Struct Biol.** 2013 link[3]

[1] http://dx.doi.org/10.1016/j.jsb.2006.06.009
[2] http://dx.doi.org/10.1016/j.jsb.2003.09.032
[3] http://dx.doi.org/10.1016/j.jsb.2013.02.017

CHAPTER

SEVEN

CONTACT

7.1 Main developer

- Andreas D. Schenk [1]

7.2 Contributors

- Ansgar Philippsen [2]
- Valerio Mariani [3]
- Marco Biasini [3]
- Jeff Lovelace [4]

1. Walz Laboratory, Department of Cell Biology, Harvard Medical School, Boston
2. D.E. Shaw, New York
3. Schwede Laboratory, Swiss Institute of Bioinformatics, University of Basel, Basel
4. Epley Institute, Structural Biology Facility, University of Nebraska Medical Center, Omaha

7.3 Former Contributors

- Giani Signorell
- Johan Hebert
- Remco Wouts
- Simon Bernèche

7.4 Mailing Lists

We have a user mailing list[1] for general questions and discussions concerning IPLT, as well as a dedicated developers mailing list[2].

[1] https://www.maillist.unibas.ch/mailman/listinfo/iplt-user
[2] https://www.maillist.unibas.ch/mailman/listinfo/iplt-devel

7.5 Financial support

7.6 Contributions

IPLT has been made possible with financial and personnel contributions from:

- Harvard Medical School[3] (2009-present)
- Swiss National Science Foundation[4] (2001-2012)
- 3DEM Network of Excellence (2003-2009)
- Maurice E. Müller Institute (2001-2009)

7.7 Website and Hosting

The hosting of the source code and the website was kindly provided by the:

- Swiss Institute of Bioinformatics[5]

Navigation icons created by:

- Glyphish[6]

[3] http://www.hms.harvard.edu
[4] http://www.snf.ch
[5] http://www.isb-sib.ch
[6] http://glyphish.com

PYTHON MODULE INDEX

i

iplt, 7
iplt.alg, 26
iplt.gui, 126