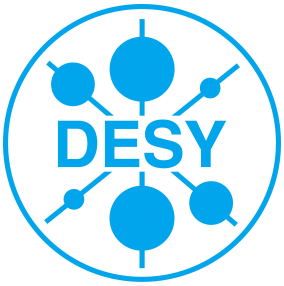

Software Development for the Crystallography Software Suite CrystFEL

Dominik Michels, RWTH Aachen University, Germany

September 6, 2016

Abstract

The following report present the improvements to the crystallography software suite CrystFEL implemented by the summerstudent Dominik Michels. The improvements include a new python geometry parser implemented according to the definition standards of a CrystFEL geometry file [1]. The parser is able to validate given geometry files and outputs the geometry information as a hierarchical python dictionary. Furthermore the functionality to read CrystFEL stream files with the diffraction pattern viewer `cxiview.py` has been implemented. `cxiview.py` is now able to draw found and predicted peaks as well as to display information about the crystal structure. Within `cxiview.py` error management and stability has been improved.

Contents

1 Introduction

In "serial femtosecond crystallography" at free electron lasers, such as the Linac Coherent Light Source at SLAC, protein crystals are illuminated by strong photon pulses for a short amount of time. The photons are diffracted by the crystal structure and measured by a detector located behind the crystal. Due to the the high shot rate of free electron lasers many images can be recorded in a short amount of time resulting in thousands of diffraction patterns which need to be analyzed. This analysis process is automated by the snapshot serial crystallography suite CrystFEL [4].
The software suite CrystFEL provides, besides the functionality to analyze diffraction patterns, the program `hdfsee` to display the images recorded by the detector. This is especially useful in order to check the quality of the recorded diffraction images and to control and manage the analysis process of CrystFEL. Besides `hdfsee` in CrystFEL there is the `cxiview.py` program in the cheetah software suite [3]. `cxiview.py` works similar to `hdfsee` but is written in the interpreted language python and provides a different feature set. It has the capability to display both predicted and found peaks and draw resolution rings.
The first result of this work is the improvement of the code quality and the addition of new features to the program `cxiview.py`.
The detectors at free electron lasers usually consist out of many small modules mounted on a plate. Figure 1 shows for example a photo of the detector at LCLS at SLAC. Each panel can be seen as a single camera. The data received from the detector after a snapshot is the raw pixel data from the single panels. To spatially reconstruct the diffraction information it is therefore necessary to know the position of the panels on the plate. Hence, in order to display the images correctly, the information of the geometrical layout of the detector is needed. This geometry information is saved in a standardized file format described in [1]. The resulting files are called geometry files. For the programs using the geometry file it is essential to have a geometry file parser to get the relevant information out of the geometry file format.
The second goal of this work is the development of a multi-purpose geometry file parser in python.
In section 2 of this report the CrystFEL geometry file format and the implementation of the geometry parser is described in detail. In section 3 the enhancements of the `cxiview.py` program are explained and shown in pictures. This report is written as a technical report providing information about features and implementation details about the programs. For a more general overview over CrystFEL visit `https://www.desy.de/~twhite/crystfel/`.

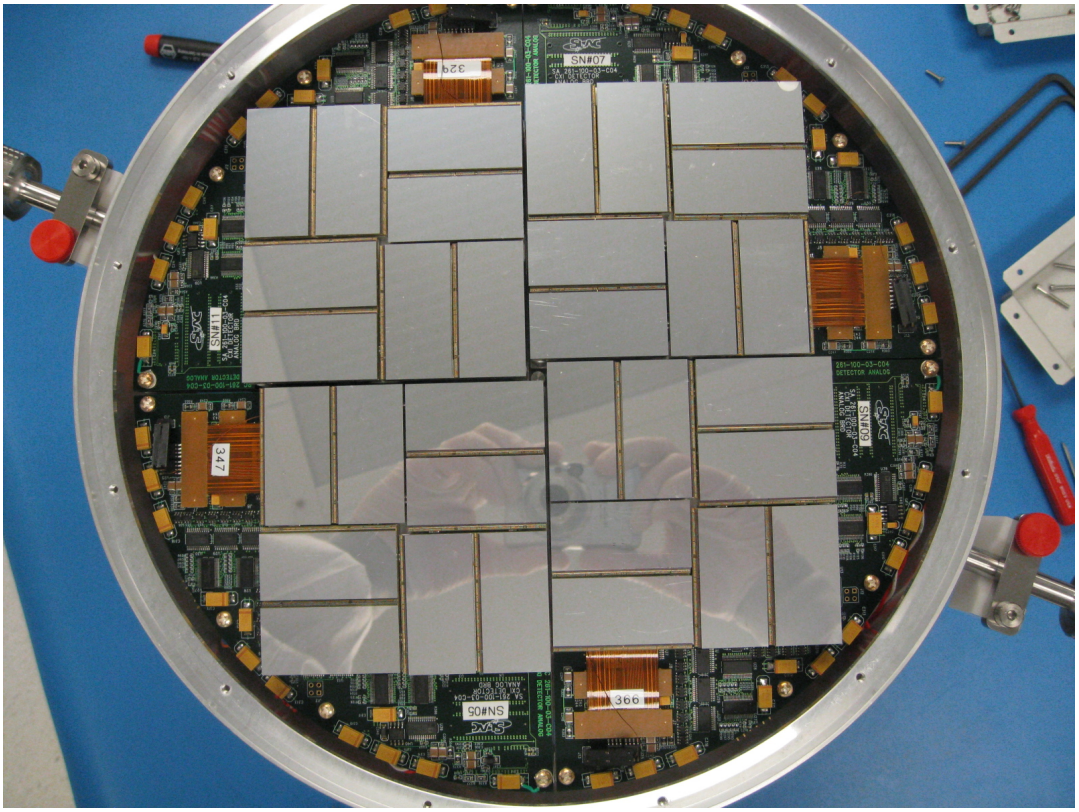

Figure 1: This figure shows a picture from `https://portal.slac.stanford.edu/sites/lcls_public/instruments/cxi/pictures/2013-04-19-CSPAD-DS1-Image-1700.JPG` of the detector at the free electron laser LCLS at SLAC. The detector is subdivided into 32 modules which are mounted on a logical board. The position of the modules is measured before the detector is mounted behind the free electron laser.

2 Implementation of a CrystFEL geometry parser in python

In this section the implementation and usage of the CrystFEL geometry parser in python is described. The CrystFEL geometry file format is defined and explained in detail on the webpage [1]. Thus an explanation about the physical meaning of the parameters in the geometry file is omitted here. The first part of this section focuses on the structural layout of the information and the design of the parser program. The second part of this section is concerned with explaining how to use the parser in later applications.

2.1 Parsing the file

The geometry parser starts to parse the geometry file by reading the file into memory and removing all comments and all resulting empty lines. After that the parser processes every line sequentially.
The geometry parser was designed to be highly adaptable to changes in the definition of the geometry file. Therefore every line of the file is parsed with the help of regular expressions with can easily be extended to include new information without changing the core implementation of the parser. Depending on the structure and the found keywords, also called properties, the category of the information in the line is decided.
The information stored in a line of the geometry file can be divided into five categories: Information about beam characteristics, detector panels, bad regions, rigid groups and rigid group collections. Each of the categories has a slightly different information layout which requires a slightly different parsing process.

2.1.1 Handling beam characteristic information

The information about beam characteristics is stored as an assignment of the form:

$$Property = Value \tag{1}$$

Property is the name of the beam characteristic information and *Value* the actual information. The properties allowed in a geometry file are *photon_energy* and *photon_energy_scale*. The Value can either be of the type string, integer or float.
The beam characteristic information is parsed with the help of the regular expression:

```
^[\s]*
(?!rigid_group)              # must not be a rigid group
(?!rigid_group_collection)   # must not be a rigid group collection
([A-Za-z0-9_]+)              # property name
[\s]*
=
(.*$)                        # value
```

If the matched property name does not match the allowed beam characteristic properties an error is reported.

2.1.2 Handling panel information

The information about detector panels is stored as an assignment in two different forms:

$$Panel/Property = Value \tag{2}$$

$$Property = Value \tag{3}$$

Panel is the name of the panel in the detector. For panel names only alphanumeric characters and {_} are allowed. *Property* and *Value* contain the panel properties and the information about the actual value of the given property. The different representations 2 and 3 of panel information are referred to as local and global information respectively. If the information is of the form 3, i.e. global it is not assigned to a specific panel and therefore considered to be valid for all following panels in the geometry file. If for a following panel the same property is also defined in the local form, the local information is preferred.
The local panel information is parsed with the help of the regular expression:

```
^[\s]*
(?!bad)              # must not be a bad region
([A-Za-z0-9_]+)      # panel name
\/
([A-Za-z0-9_]+)      # property name
[\s]*
=
(.*$)                # property value
```

The global panel information is, due to the same structure as the beam characteristic information, parsed with the same regular expression as the beam characteristic information. The two categories are distinguished by the allowed properties. The list of allowed properties is *data*, *dim0*, *dim1*, *dim2*, *dim3*, *min_fs*, *min_ss*, *max_fs*, *max_ss*, *adu_per_eV*, *badrow_direction*, *res*, *clen*, *coffset*, *fs*, *ss*, *corner_x*, *corner_y*, *max_adu*, *no_index*, *mask*, *mask_file*, *saturation_map*, *saturation_map_file*, *mask_good*, *mask_bad*, *adu_per_photon*.
If the found *Property* does not match a property on the list an error is reported.

2.1.3 Handling bad region information

The information about bad regions is stored as an assignment of the form:

$$Badregion/Property = Value \tag{4}$$

The name of the bad region is given by *Badregion*. It has to begin with the three characters {*bad*}. Otherwise the bad region will be detected as a new detector panel. The bad region and the property of the bad region are allowed to consist out of all alphanumeric characters and {_}. The geometry parser imposes no restriction on the allowed properties.
The bad region information is parsed with a regular expression similar to the regular expression for detector panels:

```
^[\s]*
(?=bad)              # must have bad at the beginning
([A-Za-z0-9_]+)      # bad region name
\/
([A-Za-z0-9_]+)      # property name
[\s]*
=
(.*$)                # property value
```

2.1.4 Handling rigid group and rigid group collection information

The information about rigid groups and rigid group collections is stored as an assignment in the following form:

$$Property = Value1, \ldots, ValueN \tag{5}$$

Property is the name of the rigid group or the rigid group collection. It has to start with *rigid_group* or *rigid_group_collection* respectively. Otherwise it is treated as a global panel information. The list of *Value*s is supposed to contain panel names. The geometry parser does not check if the *Value* is a panel name used in the geometry file. For both the *Property* and the list of *Value*s alphanumeric characters and {_} are allowed.
The rigid group and rigid group collection information is parsed with the help of the regular expression:

```
^[\s]*
((rigid_group)       # rigid_group_collection respectively
(?!_collection)      # this line is omitted for
                     # rigid_group_collections
[A-Za-z0-9_]+)       # name of the rigid_group or
                     # rigid_group_collection
[\s]*
=
[\s]*
(([A-Za-z0-9_]+)[\s]*  (,[\s]*[A-Za-z0-9_]+[\s]*)*)$
```

2.2 Storing the information

The information retrieved from the geometry file is stored in a hierarchical python dictionary, called `dictionary` in the program. The first level of the dictionary has the keys *panels*, *beam_characteristics*, *bad_regions*, *rigid_groups* and *rigid_group_collections* corresponding to the information categories present in the geometry file. The information storage is explained using the example of the panel information. The panel information is stored behind the key *panels*. The keys of the dictionary `dictionary['panels']` are the panel names and keys of the dictionary `dictionary['panels']['panel1']` are the panel properties. The assigned value to the specific panel properties is given by the value stored in the dictionary.

The information of the other keys is stored analogously in the python dictionary. The geometry parser tries to store the values in the dictionary as integer or float, if a conversion is possible. If not the values are stored as string objects.

2.3 Usage of the `GeometryFileParser` class

The main class implementing the features of the geometry parser is called `GeometryFileParser`. The parser can be easily used by the creating a `GeometryFileParser` object. The parser only needs the path to the geometry file on disk and has the functionality to parse the geometry file, print the resulting python dictionary, validate the given geometry file and generate the information needed for the `cxiview.py` program. The resulting dictionary is stored in the global class variable `dictionary`. In the following the docstrings from the implementation are given to describe the functionality of the methods in more detail.

2.3.1 GeometryFileParser.__init__

```
The constructor of the class.

Args:
    filename (string): Path to the geometry file

Note:
    The filename is optional. If a filename is given it is stored
    in self.filename.
```

2.3.2 GeometryFileParser.check

```
This method checks if the given geometry file or the geometry
file stored in the self.filename fulfils the definition
standards of a valid CrystFEL geometry file.

Args:
    filename (string): Path to the geometry file

Returns:
    bool: True if the geometry file fulfills the standards, False
        otherwise

Note:
    The filename is optional. If no filename is given the filename
    stored in self.filename is used.
```

2.3.3 GeometryFileParser.parse

```
This methods parses the geometry file and saves the information in
self.dictionary.

Args:
    filename (string): Path to the geometry file

Note:
    The filename is optional. If no filename is given the filename
    stored in self.filename is used.
```

2.3.4 GeometryFileParser.pixel_map_for_cxiview

```
This method returns the information needed for the cxiviewer.py

Returns:
    dict: Dictionary the cxiviewer.py needs to display the
        information correctly. The keys are: x, y, r, dx, coffset,
        shape, clen
```

2.3.5 GeometryFileParser.dump

```
This methods dumps the contents of the geometry dictionary
self.dictionary.
```

3 Enhancements to the `cxiview.py` program

In this section the enhancements to the `cxiview.py` program are explained. The `cxiview.py` program was originally a part of the *cheetah* software suite [3] and is designed to display diffraction snapshots from free electron lasers. A screenshot of the program is shown in Figure 2. The main part of the work on the `cxiview.py` program was dedicated to implementing the functionality to display the contents of a CrystFEL stream file. This functionality is especially useful for scientists using CrystFEL because they can easily visualize and control the evaluation process of diffraction patterns.

3.1 Description of the new features

The `cxiview.py` program has to be used via command line. It features many different options which are accessible by different command line flags. In order to use the new stream file reading functionality the option `-s` has been introduced. The usage of the new feature is as follows.

```
python cxiview.py -s <path to streamfile>
```

`cxiview.py` is dependent on python 3.5 and packages which are distributed with the *cheetah* software suite. Hence, before using `cxiview.py`, please make sure that the installation instructions on `http://www.desy.de/~barty/cheetah/Cheetah/Cheetah_GUI.html` has been executed.
The new `-s` option parses the CrystFEL stream file. The parser has a similar layout as the CrystFEL geometry parser described in section 2 but implementation details are not given here because it is not for general purpose use. A CrystFEL stream file contains the CrystFEL geometry information and many different chunks which themselves contain the analysis information about one diffraction snapshot. This includes, besides the link to the snapshot image, information about found peaks, possible crystals and predicted peaks . With the new parser all this information can be displayed.
The functionality is explained at the example of Figure 2. When the `cxiview.py` program is started it displays the snapshot from the first chunk in the CrystFEL stream file. It is possible to navigate between the snapshots with the buttons *Previous*, *Next*, *Play*, *Random*, *Shuffle* or by entering a frame number. If information about a crystal in a snapshot is present the crystal unit cell information is displayed in the lower left corner. By activating the tickboxes *Found peaks* and *Predicted peaks* found and predicted peaks are drawn on the image. Found peaks are displayed by red squares and predicted peaks by blue circles. Example pictures can be found in Figure 3 and Figure 4. Information about a specific predicted peak, including the resolution and hkl indices, is shown in the lower left corner if the user clicks on it. This is shown in Figure 5. If the tickbox *Resolution rings* is activated and information about the photon energy and the detector distance is present in the stream file the resolution rings are drawn. An example is show in Figure 6. The information about photon energy and detector distance could be missing in corrupted stream files.

3.2 Description of the code improvements

The major code improvements are explained here in a list style format. The complete development log of the program can be found in the git log in the branch `michelsd` under the url `https://stash.desy.de/projects/CCPHASE/repos/cheetah.py/browse`.

- The geometry parser of `cxiview.py` is almost completely replaced by the new geometry parser described in section 2. The new geometry parser is well documented and failsafe. Furthermore it can parse all the geometry files which fulfill the geometry files definition standards given on the webpage [1].
- Error checking of the parsed command line options in `cxiview.py` added.
- User defined exceptions to handle errors in `cxiview.py` added.
- Bug which did not allow to jump to image number 0 with the frame box while opening `.cxi` files fixed.
- Bug displaying the wrong image number in the window title while opening `.cxi` files fixed.
- Dependency on numpy.nan changed to float('nan') to support native python language instead of library features.

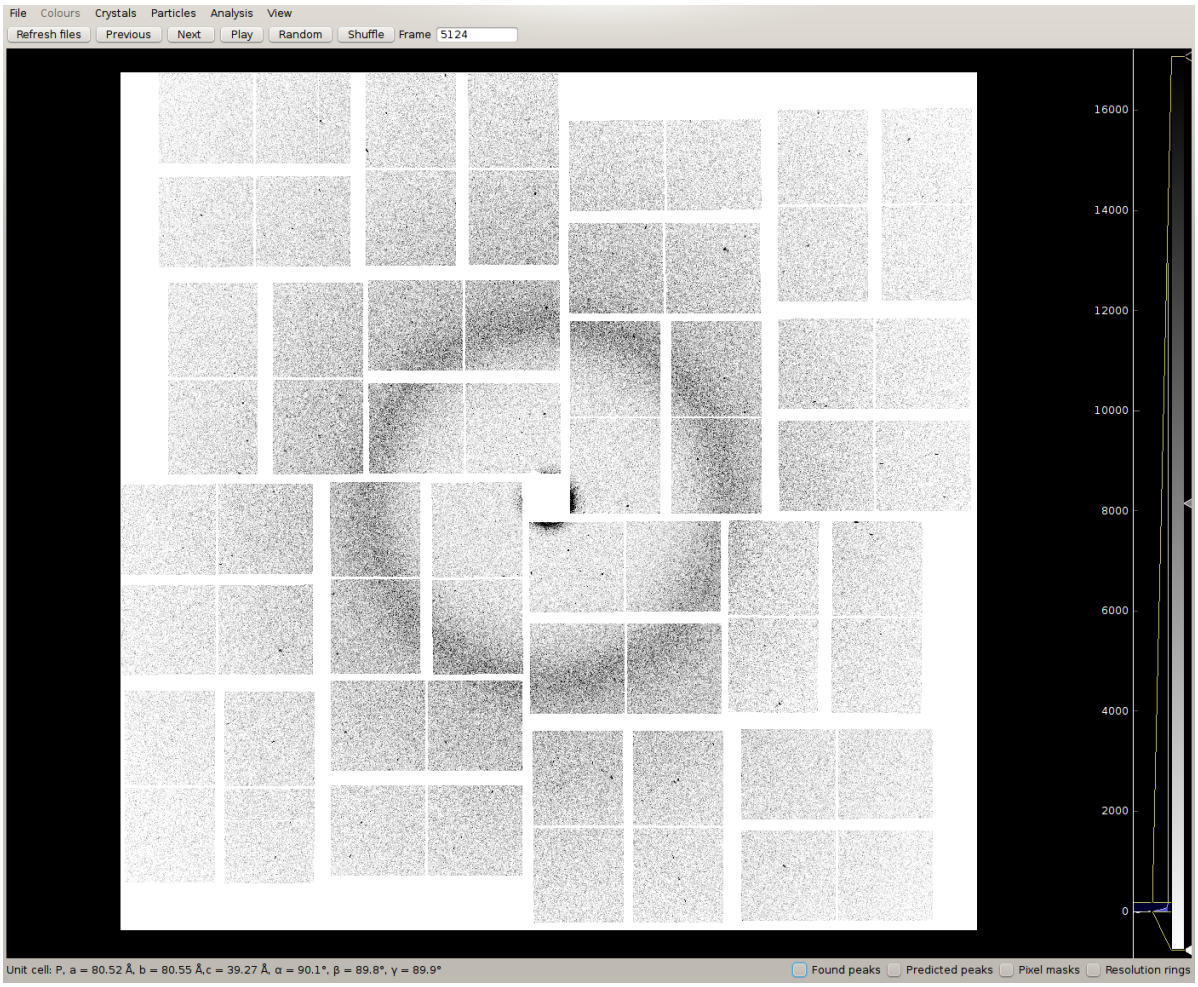

Figure 2: Figure displaying a screenshot of the `cxiview.py` after right after the program was started.

4 Summary and Outlook

A new geometry parser and the functionality to display the contents of CrystFEL stream files in the `cxiview.py` program have been implemented successfully. The new features are well documented and include error handling.
As an outlook there are still things worth working on. The code quality in the untouched parts of the program is from a software engineering perspective not good. There is almost no error handling present. If error handling is present by try and except blocks in most cases generic exceptions are caught. Furthermore PEP style documentation in the code is missing. Therefore, to improve the stability and usability, improvements and reimplementation of some parts of the `cxiview.py` program are recommended.
New features which are worth adding to the program include the implementation of the functionality of the *Pixel masks* tickbox and better, high resolution image saving.

5 Appendix

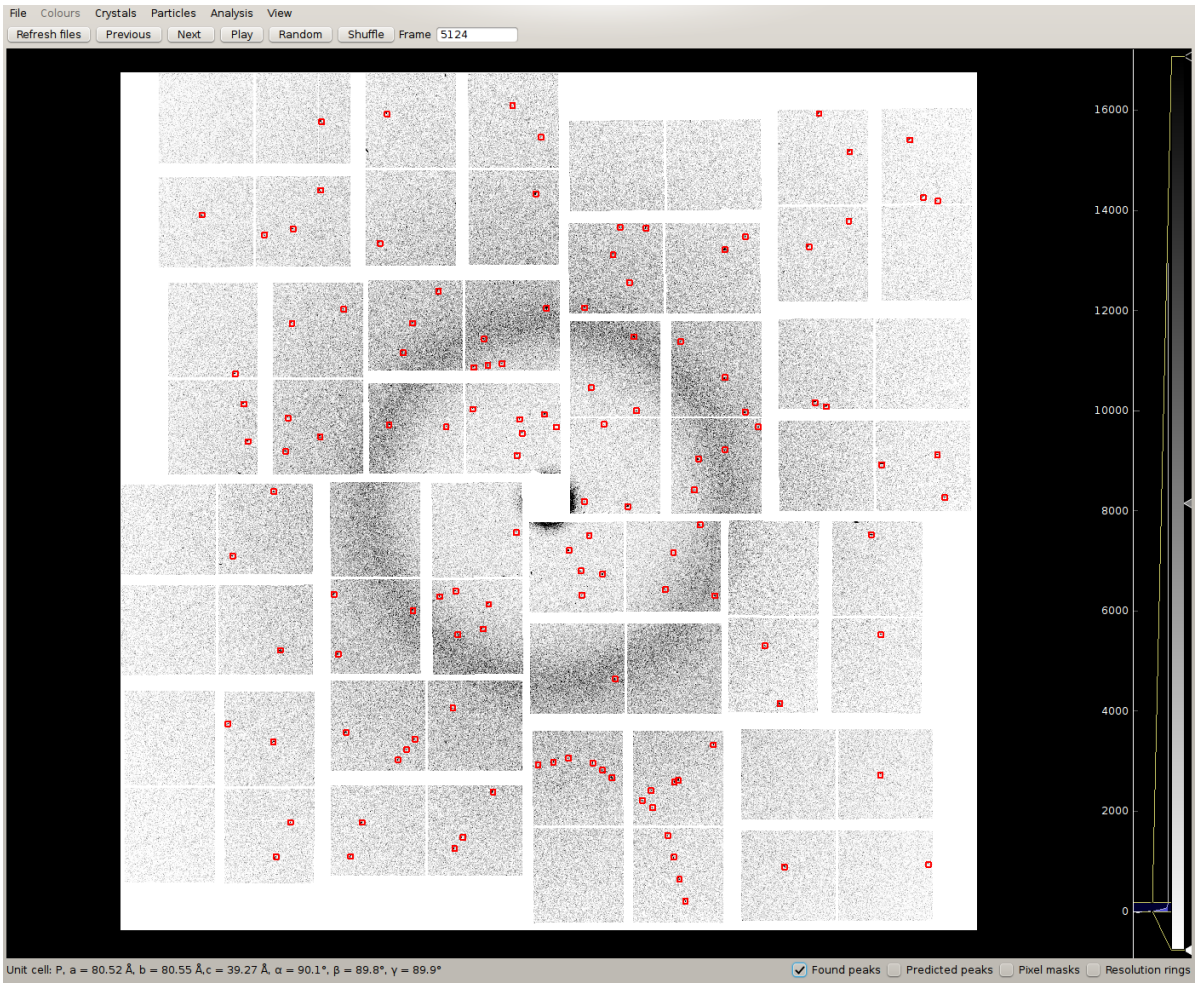

Figure 3: Figure displaying a screenshot of the `cxiview.py` program when the *Found peak* tickbox is activated.

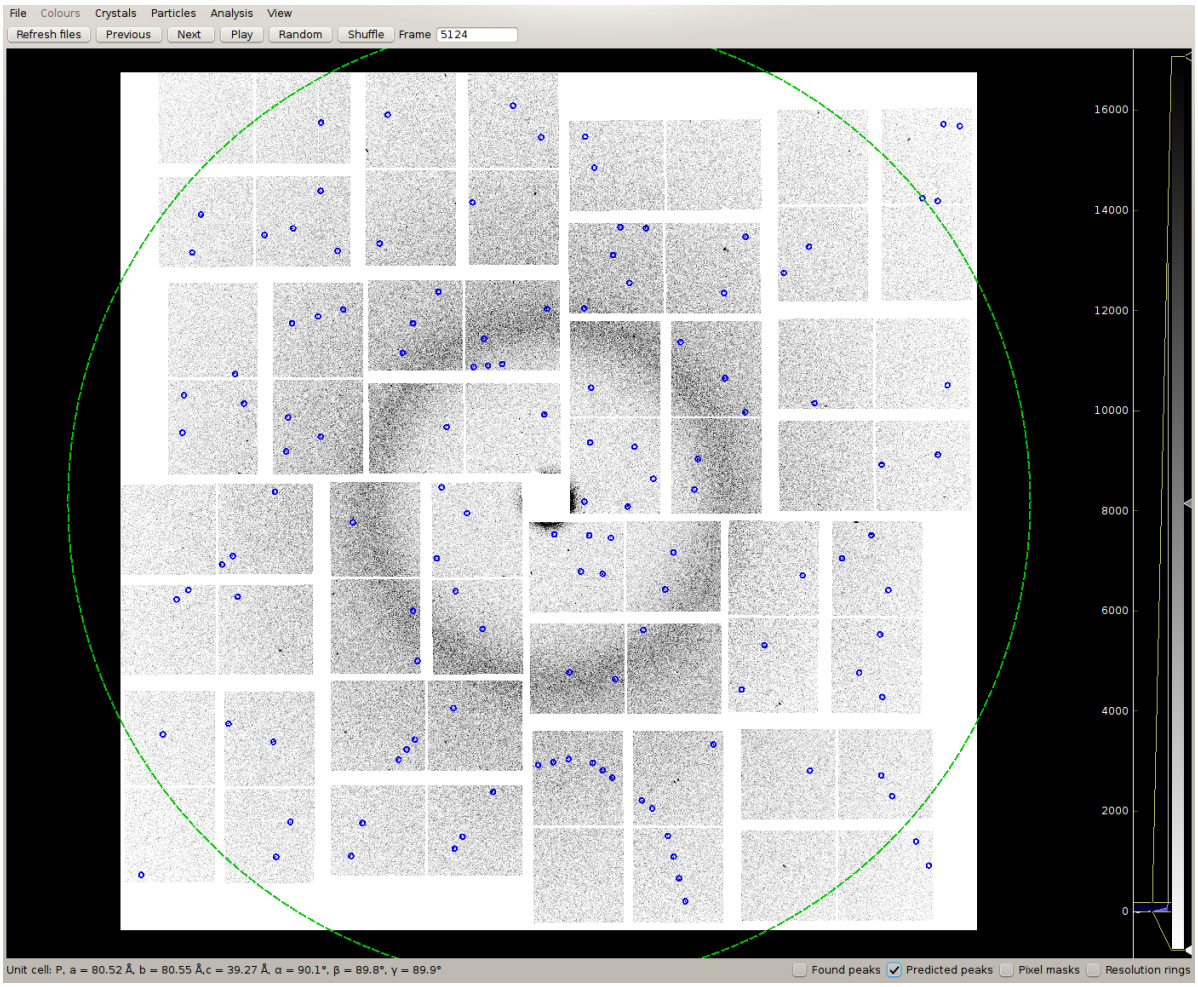

Figure 4: Figure displaying a screenshot of the `cxiview.py` program when the *Predicted peak* tickbox is activated and a crystal was found by CrystFEL in the snapshot

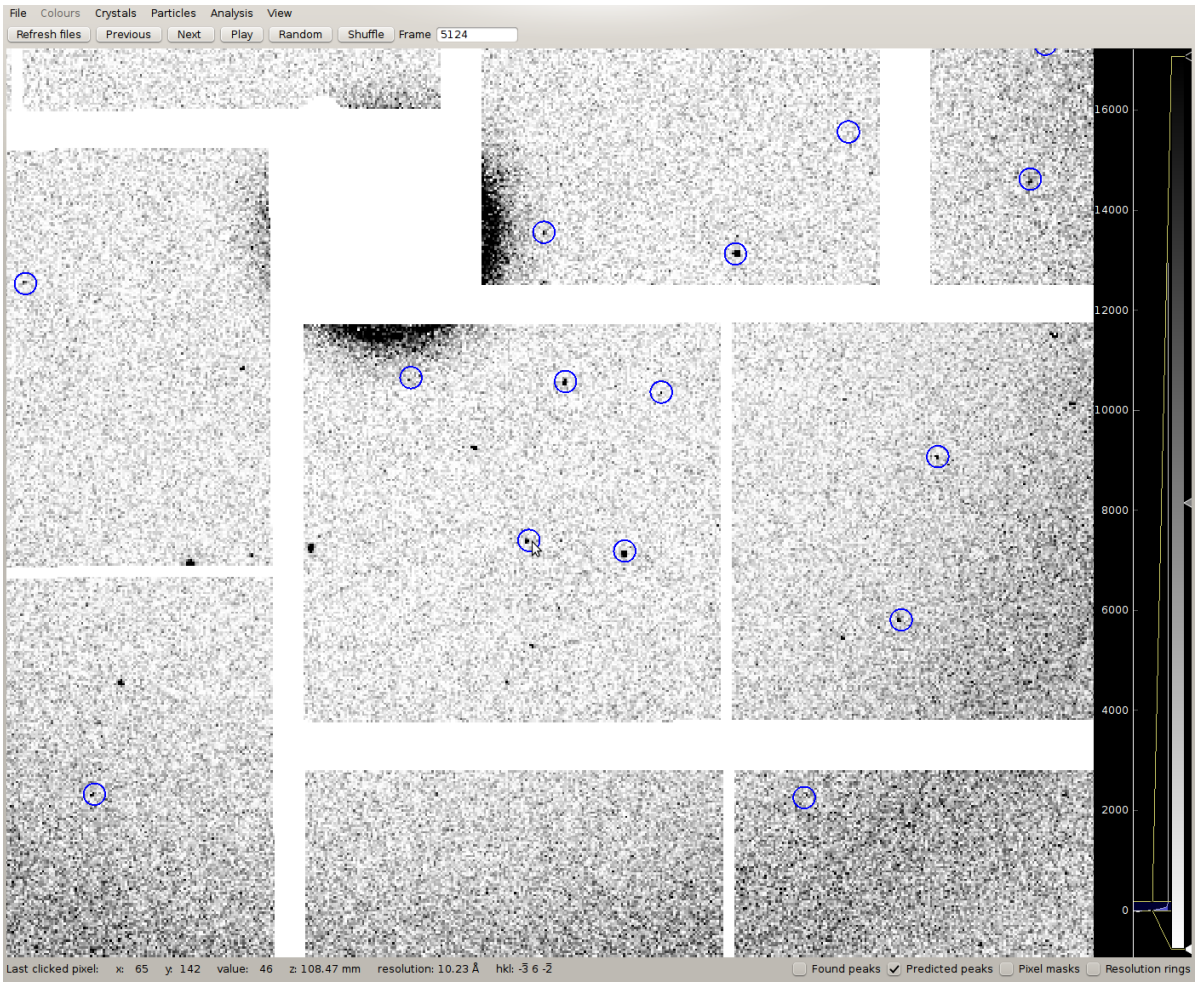

Figure 5: Figure displaying a screenshot of the `cxiview.py` program when the *Predicted peak* tickbox is activated, a crystal was found by CrystFEL in the snapshot and the user has clicked on a predicted peak. Information about the predicted peak is displayed in the lower left corner.

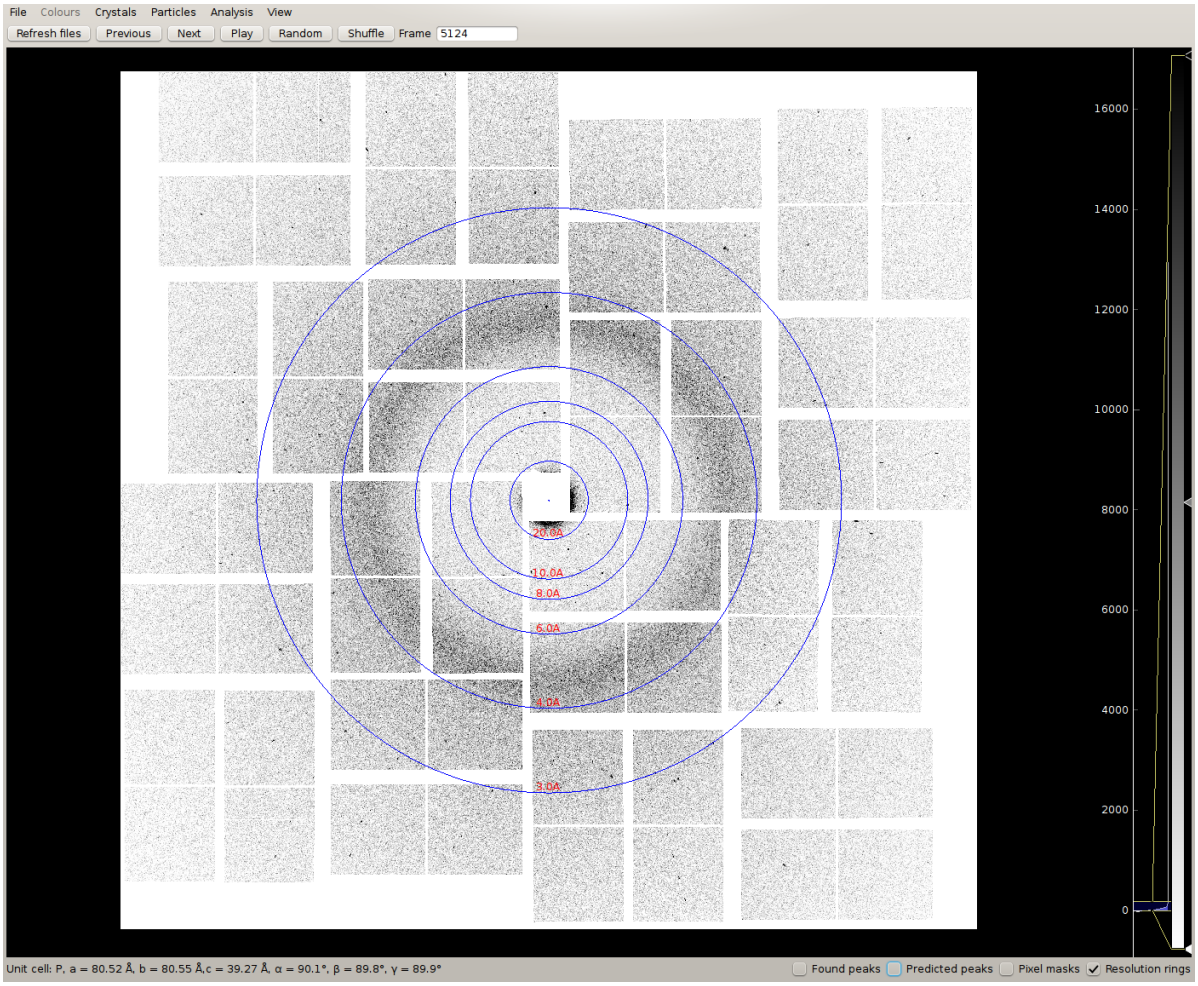

Figure 6: Figure displaying a screenshot of the `cxiview.py` program when the *Resolution ring* tickbox is activated and sufficient information to calculate resolution rings is present in the stream file.

References

[1] CrystFEL geometry specification. `http://www.desy.de/~twhite/crystfel/manual-crystfel_geometry.html`. Accessed: 2016-08-24.

[2] LCLS detector image. `https://portal.slac.stanford.edu/sites/lcls_public/instruments/cxi/pictures/2013-04-19-CSPAD-DS1-Image-1700.JPG`. Accessed: 2016-08-24.

[3] Anton Barty, Richard A. Kirian, Filipe R. N. C. Maia, Max Hantke, Chun Hong Yoon, Thomas A. White, and Henry Chapman. *Cheetah*: software for high-throughput reduction and analysis of serial femtosecond X-ray diffraction data. *Journal of Applied Crystallography*, 47(3):1118–1131, Jun 2014.

[4] Thomas A. White, Richard A. Kirian, Andrew V. Martin, Andrew Aquila, Karol Nass, Anton Barty, and Henry N. Chapman. *CrystFEL*: a software suite for snapshot serial crystallography. *Journal of Applied Crystallography*, 45(2):335–341, Apr 2012.