Documentation

Dataset for Manuscript "Fast Simulation of Wide-Angle Coherent Diffractive Imaging"

Paul Tuemmler, Julia Apportin, Thomas Fennel, Christian Peltz

1 General Information

Dataset title:	Dataset for Manuscript "Fast Simulation of Wide-Angle Coherent Diffractive Imaging"
Authors/Creators:	Paul Tuemmler, Julia Apportin, Thomas Fennel, Christian Peltz
Affiliation:	Institute of Physics, University of Rostock, 18051 Rostock, Germany
E-Mail:	christian.peltz@uni-rostock.de
Date:	Nov. 2025
Type:	Simulations, Analysis Results, Source Code
Language:	English
Rights:	CC BY 4.0
DOI:	10.18453/rosdok_id00004923
Funding:	German Research Foundation (DFG) via CRC 1477 "Light-Matter Interactions at Interfaces" (ID: 441234705) European Social Fund and the Ministry of Education, Science and Culture of Mecklenburg-Vorpommern via project NEISS (ID: ESF/14-BM-A55-0007/19)

2 Description

This archive contains the data shown in publication [1] together with a python script to visualize them. It also provides a python script to generate this data using the open source implementation of pMSFT.

3 Data Format

The data is stored in a compressed HDF5 format, which can be loaded using virtually all programming languages. To view the data without extensive programming knowledge, we recommend using `silx view` (`https://www.silx.org/doc/silx/latest/applications/view.html`) or `HDFView` (`https://www.hdfgroup.org/downloads/hdfview/`).

4 Archive Structure

The archive has the following structure:

```
├── README.md
├── LICENSE.txt
├── generate_data.ipynb
├── plot_data.ipynb
├── data
│   ├── fdtd_octahedron_reference.h5
│   ├── multislice_octahedron_simulations.h5
│   ├── refractive_index_scan.h5
│   └── henke_cxro_files
│       ├── ac.nff
│       ├── ag.nff
│       └── ...
└── requirements.txt
```

The contents of the individual files and folders are described below in more detail.

4.1 fdtd_octahedron_reference.h5

Compressed HDF5 file containing the reference FDTD simulation data for the benchmark example shown in Figure 1 of the publication.

```
fdtd_octahedron_reference.h5
├── fdtd_image :  2d array of the FDTD reference image (256x256 pixels; float64)
├── fdtd_kx :  kx grid corresponding to fdtd_image (256x256 pixels; float64)
├── fdtd_ky :  ky grid corresponding to fdtd_image (256x256 pixels; float64)
├── fdtd_kz :  kz grid corresponding to fdtd_image (256x256 pixels; float64)
├── model :  3d array of the sample model (150x150x150 voxels; uint8)
├── model_width_in_nm :  box width of the sample model in nm (float64)
├── wavelength_in_nm :  wavelength used in the simulation in nm (float64)
└── refractive_index :  complex refractive index used in the simulation (complex128)
```

4.2 multislice_octahedron_simulations.h5

Compressed HDF5 file containing the simulation results for the comparison of pMSFT and the other methods with the FDTD reference data (see above) for an octahedron as shown in Figure 1 of the publication. This can also be generated using `generate_data.ipynb`, see below for more details.

```
multislice_octahedron_simulations.h5
└── images
    ├── BORN : 2d array of the Born's approximation result (2048x2048 pixels; float64)
    ├── HARE : 2d array of result using Hare's method (2048x2048 pixels; float64)
    └── MSFT : 2d array of the MSFT result (2048x2048 pixels; float64)
```

```
│
│   ├── PMSFT : 2d array of the pMSFT result (2048x2048 pixels; float64)
│   └── SAXS : 2d array of the SAXS result (2048x2048 pixels; float64)
├── pmsft_kx :  kx grid corresponding to the above images (2048x2048 pixels; float64)
├── pmsft_ky :  ky grid corresponding to the above images (2048x2048 pixels; float64)
└── pol_mask :  linear polarization mask (2048x2048 pixels; float64)
```

4.3 refractive_index_scan.h5

Compressed HDF5 file containing the simulation results for the refractive index scan shown in Figure 3 of the publication.

```
refractive_index_scan.h5
├── real_space :  axis corresponding to real parts of the refractive index (512 pixels; float64)
├── imag_space :  axis corresponding to imaginary parts of the refractive index (256 pixels; float64)
├── difference_map_Q
│   ├── BORN : 2d array of Q value in Born's approximation based on the above grid (256x512 pixels;
│   │   float64)
│   ├── HARE : 2d array of Q value using Hare's method based on the above grid (256x512 pixels; float64)
│   ├── MSFT : 2d array of the MSFT Q value based on the above grid (256x512 pixels; float64)
│   └── PMSFT : 2d array of the pMSFT Q value based on the above grid (256x512 pixels; float64)
├── difference_map_R
│   ├── BORN : 2d array of R value in Born's approximation based on the above grid (256x512 pixels;
│   │   float64)
│   ├── HARE : 2d array of R value using Hare's method based on the above grid (256x512 pixels; float64)
│   ├── MSFT : 2d array of the MSFT R value based on the above grid (256x512 pixels; float64)
│   └── PMSFT : 2d array of the pMSFT R value based on the above grid (256x512 pixels; float64)
├── difference_map_R_mask
│   ├── BORN : 2d array of R value (with mask) in Born's approximation based on the above grid (256x512
│   │   pixels; float64)
│   ├── HARE : 2d array of R value (with mask) using Hare's method based on the above grid (256x512
│   │   pixels; float64)
│   ├── MSFT : 2d array of the MSFT R value (with mask) based on the above grid (256x512 pixels;
│   │   float64)
│   └── PMSFT : 2d array of the pMSFT R value (with mask) based on the above grid (256x512 pixels;
│       float64)
└── highlighted_points
    └── 0
        ├── ref_index :  refractive index used for this point (complex128)
        ├── image_BORN : 2d array of the Born's approximation result for point 0 (2048x2048 pixels;
        │   float64)
        ├── image_HARE : 2d array of result using Hare's method for point 0 (2048x2048 pixels; float64)
        ├── image_MSFT : 2d array of the MSFT result for point 0 (2048x2048 pixels; float64)
        ├── image_PMSFT : 2d array of the pMSFT result for point 0 (2048x2048 pixels; float64)
        └── mie_perp :  1d array of the Mie result in perpendicular direction (2048 pixels; float64)
```

```
│    │
│    │    └── mie_polar :  1d array of the Mie result in polarization direction (2048 pixels; float64)
│    ├── 1 ...
│    └── 2 ...
├── kx :  kx grid corresponding to the above images (2048x2048 pixels; float64)
├── ky :  ky grid corresponding to the above images (2048x2048 pixels; float64)
├── angles :  grid of the angles corresponding to the images (2048x2048 pixels; float64)
├── pol_mask :  linear polarization mask (2048x2048 pixels; float64)
├── max_angle :  maximum considered scattering angle in degrees (float64)
├── wavelength :  wavelength used in the simulation in nm (float64)
└── Rsphere :  radius of the spherical sample in nm (float64)
```

4.4 Other files

- `data/henke_cxro_files/`:
  Folder containing tabulated complex refractive index data from CXRO Henke database (`https://henke.lbl.gov/optical_constants/asf.html`) (state as of 30.09.2025), used for Figure 3 of the publication.

- `generate_data.ipynb`:
  Jupyter notebook containing the source code to generate the `refractive_index_scan.h5` and `multislice_octahedron_simulations.h5` data files.

- `plot_data.ipynb`:
  Jupyter notebook containing the source code to generate the plots shown in the publication using the provided data files.

4.5 Using the Jupyter Notebooks

We strongly recommend setting up a virtual environment for the execution of these scripts, i.e. using conda (https://docs.conda.io/en/latest/). This code has been tested for python 3.12. Create and activate a conda environment using

```
conda create -n pymsft python=3.12
conda activate pymsft
```

At this point you can install the package requirements using pip. If you want to calculate the data yourself you should install the full requirements using

```
pip install -r requirements.txt
```

If, however, you are only interested in plotting the data, you can instead use a reduced set of required packages using

```
pip install -r requirements_plotting.txt
```

To take advantage of GPU acceleration for the calculation scripts, we are using CuPy (`https://cupy.dev/`). See the installation instructions (`https://docs.cupy.dev/en/stable/install.html`) for more details. The simplest way to install it is currently via conda:

```
conda install -c conda-forge cupy==13.6.0
```

Note that an NVIDIA GPU with the corresponding driver installed is required.

References

[1] Paul Tuemmler, Julia Apportin, Thomas Fennel and Christian Peltz, *Fast Simulation of Wide-Angle Coherent Diffractive Imaging*, Laser & Photonics Reviews (https://doi.org/10.1002/lpor.202502001).