X-ray speckle-based imaging simulation program

DATA SHEET

# Overview:

Objective: compute the intensity images obtained during a random-mask based x-ray phase contrast imaging.

From thickness maps of the geometries of the samples and membranes it computes sample images (with membrane and sample) and reference images (with only the membrane) along with simple propagation image (with only the sample) and a white (nothing other than air, filters and detector in the path of the beam.

There are 2 calculation methods: one based on ray-tracing and one based on wave optics and Fresnel propagator.

All the experiment parameters are set in the xml files (in folder “xmlFiles”).

Some algorithmic parameters are set directly in the code in the ‘main’ part of the main.py file.

Table of content

[Overview: 1](#_Toc74561609)

[User instructions: 2](#_Toc74561610)

[1) Determine the sampling requirements for your experiment 2](#_Toc74561611)

[2) Set the Parameters in the main (main.py) 2](#_Toc74561612)

[3) Set the Experiment global parameters in Experiment.xml 2](#_Toc74561613)

[4) Set the detector parameters in Detector.xml 3](#_Toc74561614)

[5) Set the source parameters in Source.xml 3](#_Toc74561615)

[6) Set all the objects parameters in Sample.xml 4](#_Toc74561616)

[***Additional remarks:*** 5](#_Toc74561617)

[Required Python libraries: 6](#_Toc74561618)

[Class methods description: 7](#_Toc74561619)

[See class Diagram in Apendix 2. 7](#_Toc74561620)

[main.py 7](#_Toc74561621)

[Gets few algorithm parameters to set. 7](#_Toc74561622)

[Initialize the Experiment class. 7](#_Toc74561623)

[Launches the computation of the images for each position of the membrane and saves them. 7](#_Toc74561624)

[Simulation physical models: 10](#_Toc74561625)

# User instructions:

## Determine the sampling requirements for your experiment

If using ray-tracing you simply need a sampling >= 2 (Shannon)

If using Fresnel: Open the script: usefullScripts/getSamplingFactor.py. Enter the correct parameters for your experiment. Note down the minimal oversampling. (Some parameters are explained in the following instructions) /!\ for low resolution experiments, the oversampling factor for the Fresnel method may be very important and induce a longer computation time.

## Set the Parameters in the main (main.py)

- experimentName : [string] corresponding to the one in the csv file

- filepath: [string] which is the path of the folder where you want to store the simulated images

- nbExpPoints: [int] which is the number of pair Is, Ir with different position of the membrane you want to simulate.

- margin: [int] enlarging of the simulation field of view (removed before saving) which allows to get rid of the edge effects (usually 10 pixels is enough).

- nbExpPoints: [integer] number of pairs (reference image, sample image) to simulate.

- simulationType: [string] “Fresnel” or “RayT” too chose the type of calculation method (For polychromatic low coherence siulations ray-tracing is better).

## Set the Experiment global parameters in Experiment.xml

**<experiment>**

**<name>**Clinic\_ContrastPhantom**</name>**

**<distSourceToMembrane** unit**=**"m"**>**1**</distSourceToMembrane>**

**<distMembraneToObject** unit**=**"m"**>**0**</distMembraneToObject>**

**<distObjectToDetector** unit**=**"m"**>**1**</distObjectToDetector>**

**<membraneName>**Mask\_CuSn\_From\_edf**</membraneName>**

**<sampleName>**ContrastPhantomB**</sampleName>**

**<sampleType>**AnalyticalSample**</sampleType>**

**<detectorName>** flatPanelSimap **</detectorName>**

**<sourceName>**clinic**</sourceName>**

**<meanShotCount>**27600**</meanShotCount>** #averaged number of photons received per pixel

**<plaqueName>**C\_plaque**</plaqueName>**

**</experiment>**

For now, **sampleType** must always be “AnalyticalSample” because “voxelized sample” is not implemented.

All element name must correspond to the elements name in their own xml file (sampleName, detectorName…).

“plaqueName” is an optional parameter. If you introduce a plaqueName, you must add a sample corresponding to this window in the Sample.xml file. In the simulation it corresponds to the detector protective plaque (usually a thin carbon layer). (Yeaah I am confusing “plaque” and “plate” in English).

**meanShotCount** is the average number of photon per voxel leaving the source. Because the distance between source and detector is huge in the synchrotron case, the absorption in air is very important and this number should be set very high (~3000 000) I must correct that in the next version to separate conventional sources cases and synchrotron cases.

## Set the detector parameters in Detector.xml

**<detector>**

**<name>**flatPanelSimap**</name>**

**<myDimensions>**

**<dimX>**500**</dimX>**

**<dimY>**1500**</dimY>**

**</myDimensions>**

**<myPixelSize** unit**=**"um"**>**127**</myPixelSize>**

**<myPSF** unit**=**"pixel"**>**1**</myPSF>**

**<myBinsThersholds>**20,30,40**</myBinsThersholds>**

**<myScintillatorMaterial>**CsI**</myScintillatorMaterial>**

**<myScintillatorThickness** unit**="**um**">**600**</myScintillatorThickness>**

**</detector>**

dimX and dimY will correspond to the dimensions of your final image.

**myBinsThersholds** are the thresholds of a spectral detector. They must be in the range of the source spectrum. They must be floats separated by only “,”. If not specified, the detector is not a spectral one. (It is an optional parameter).

The detector spectrum efficiency can now be calculated from its scintillator parameters. It is calculated as the attenuation in the thickness of the scintillator plaque using its material absorption coefficient. This coefficient is the beta coefficient that must be present in the TablesDeltaBeta.xls file of the Sample/DeltaBeta folder. (See the part about objects definition below for more details about this file). For the program to take the scintillator into account you simply add the **myScintillatorMaterial** and **myScintillatorThickness** parameters in the xml file. (They are optional parameters).

The PSF value corresponds to the standard deviation of the point spread function of the detector. For a photon counting detector with no PSF set it to 0.

The pixel size assumes square pixels.

## Set the source parameters in Source.xml

Sources.xml: the source can be monochromatic or polychromatic /!\ they don’t have the same parameters

**<source>**

**<name>** simap**</name>**

**<myType>**Polychromatic**</myType>**

**<mySize** unit**=**"um"**>**8**</mySize>** #size of the emitting spot

**<sourceVoltage** unit**=**"kVp"**>**40**</sourceVoltage>**

**<myTargetMaterial>**Mo**</myTargetMaterial>**

**<exitingWindowMaterial>**Be**</exitingWindowMaterial>**

**<exitingWindowThickness** unit**=**"mm"**>**0.2**</exitingWindowThickness>**

**<myEnergySampling** unit**=**"keV"**>**5**</myEnergySampling>**

**</source>**

The source size corresponds to the FWHM of the spot on the anode. It will be projected on the detector plane and approximated by a gaussian shape.

The source voltage allows to get its spectrum from the SpekPy library.

The source target material can be Tungsten: W (value by default if not defined), Molybdenum: Mo or rhodium Rh.

The **exitingWindowMaterial** is optional too. It corresponds to a filter placed right after the source with thickness **exitingWindowThickness**. If those parameters are not specified there is no filter.

**myEnergySampling** is for under-sampling the spectrum for the calculation (if we calculate for every energy the computation time gets quite long). It must be >=0.5keV.

Monochromatic case:

**<source>**

**<name>**id17**</name>**

**<myType>**Monochromatic**</myType>**

**<mySize** unit**=**"um"**>**10**</mySize>**

**<myEnergy** unit**=**"kev"**>**27**</myEnergy>**

**</source>**

## Set all the objects parameters in Sample.xml

This is the trickiest part.

There are 2 main types of sample: the sample of interest and the membrane. (The air volume is there by default and must not be modified).

**<sample>**

**<name>**filNylon**</name>**

**<myType>**sample\_of\_interest**</myType>**

**<myMaterials>**Nylon**</myMaterials>**

**<myGeometryFunction>**CreateSampleNylonWire**</myGeometryFunction>**

**<myRadius** unit**=**"um"**>**70**</myRadius>**

**<myOrientation** unit**=**"degree"**>**81.56**</myOrientation>**

**</sample>**

**<sample>**

**<name>**Mask\_CuSn\_From\_txt**</name>**

**<myType>**membrane**</myType>**

**<myGeometryFunction>**getMembraneSegmentedFromFile**</myGeometryFunction>**

**<myMembraneFile>**Samples/Membranes/CuSn.txt**</myMembraneFile>**

**<myPMMAThickness** unit**=**"um"**>**4000**</myPMMAThickness>**

**<myMaterials>**CuSn,PMMA**</myMaterials>**

**<myMeanSphereRadius>**6**</myMeanSphereRadius>**

**<myNbOfLayers>**2**</myNbOfLayers>**

**</sample>**

Every sample has:

- **name**

- **myType** which is either sample\_of\_interest, membrane or thin\_film (this one corresponding to the detector protective plate)

- **myMaterials** is either 1 material name or a list of materials separated by comas “,” (no spaces). This material will be used in the getDeltaBeta function. Important precisions about those materials bellow.

- **myGeometryFunction** is the function that you use to get the thickness map of your sample (or thickness mapS if there are several materials) more details bellow.

Any other parameters are related to the way you generate the samples geometries.

**Delta Beta parameters:**

/!\ For each material of the considered samples, a delta-beta table must exist in the TablesDeltaBeta.xls document (in Samples/deltabeta/) in the same way as the other already entered data.

A list of the already existing materials is in the first column of the sheet.

To generate a new table of data use the website: <http://ts-imaging.science.unimelb.edu.au/Services/Simple/ICUtilAbsorb.aspx>

Then copy and paste the obtained values in the ArangeValue sheet of the TablesDeltaBeta.xls doc. To separate the columns, use Data>Convert the remove all the additional spaces with the replace tool. Then copy and use “paste Values” on the main sheet.

**Membrane geometry:**

The membrane geometry is based on a CuSn membrane microscopic image segmented. The mean radius of those spheres was 4.57 but the distribution of those radia is not uniform. (See Appendix 1).

As a parameter, you can choose to change the mean sphere radius (it is adviced to start by simulating 1 point and verify that the grains are visible on the reference image). We observe best results when they appear to be between 5 and 10 pixels wide.

You can choose the grains material (first material of your list) and the material of the support (usually PMMA).

You can choose the number of layers of spheres you want to generate.

You can choose the thickness of the support on which the spheres are placed (in practice it is a 2mm thick plate of PMMA per layer of spheres).

/!\ the grains geometry is independent from the PMMA thickness on which they are fixated. The PMMA thickness is the total thickness of all considered layers : if we use 3 layers, the PMMA thickness should be 6000um.

To creat the membrane there are 2 options:

1. Generate it directly in the simulation from the segmented coordinates of the spheres using the getMembraneSegmentedFromFile geometry function. This is quite time consuming. If you want to try changing other experimental parameters it is advised to use the second option.
2. Start by saving thickness maps of the spheres using the Samples/﻿getMembraneFromFile.py script. You must enter all the parameters in the main corresponding to your experiment. Then launch it and copy the path of the folder where they are saved in the xml file corresponding to your membrane for the simulation under **myMembraneFile**. Then in the simulation you will use the getMembraneFromFile geometry function.

Careful, in that case, you must remember the number of layers to adapt the thickness of PMMA to add.

Remark: the segmented surface is limited so depending on the size of the simulated surface you may see the stitching edge appear on the geometry. It does not disturb the phase retrieval.

**Samples geometries:**

Some geometry functions are already implemented: a cylinder, a sphere, two spheres in a tube and a contrast phantom used for the test of micro-tomography tools. You can choose them by setting their geometry functions. You can also load thickness maps if those have been previously created.

**<myGeometryFunction>**CreateSampleCylindre**</myGeometryFunction>**

For that function you must precise <myRadius> in um, <myOrientation> in degree and <myMaterial>. Be sure that the diameter of your wire fits your final image size (defined by the size of your detector) when multiplied by the magnification (defined by your distances).

**<myGeometryFunction>**CreateSampleSphere**</myGeometryFunction>**

Set <myRadius> in um and <myMaterials>. Check also the size of your sample.

**<myGeometryFunction>**CreateSampleSpheresInTube**</myGeometryFunction>**

This sample has a fixed size. Tube diameter is 3mm, spheres radius is 0.5mm. The tube height is the final image height.

You only need to set <myMaterials> with 3 materials for this sample: Sphere1Material,Sphere2Material,TubeMaterial. Once again, please check that the sample fits your geometry.

**<myGeometryFunction>**generateContrastPhantom**</myGeometryFunction>**

Same as the previous one, this phantom geometry is fixed. It is a phantom of materials equivalent to human tissues which materials are already set in the xml files but can be changed at your convenience. The final geometry will be calculated for the phantom seen from the side which will be more interesting for tomography.

The function is stored in the generateContrastPhantom.py file. This script can be launched directly with its main to create your geometry beforehand and save the thickness maps as .edf files. Then you will have to use the **<myGeometryFunction>**openContrastPhantom**</myGeometryFunction>** to load the geometry (faster than generating the phantom fitting your geometry everytime you launch the code.

**Create a new geometry function**

Samples geometry functions are called in the Sample.py script in the function: ﻿getMyGeometry(self,studyDimensions, studyPixelSize,oversamp, pointNum=0)

Where self.studyDimensions is a tuple (dimX, dimY) corresponding to (detectorDim+2\*margins)\*oversampling. The studyPixelSize corresponds to the voxel size in the sample plane or to the membrane pixel size for the membrane.

The pointNum is used to change the position of the membrane for each point but should not be used for any other sample.

When creating a new geometry function, you can load its specific parameters as for the others in the Sample.py file in the getCorrectValues function. Then your function must be called in the getMyGeometry function in a condition to select your geometry function (basically use the same syntax as for the other functions).

## Additional remarks:

1. For now the code does not calculates the number of photons per pixel as a function of voltage, current intensity and exposition time, you have to chose the value by yourself. If you know a simple and efficient way to calculate that, please let me know.
2. Both calculation methods give similar results for synchrotron simulations but they diverge for less coherent systems. It is then preferable to use the ray-tracing one.
3. The refraction calculation is greatly accelerated by numba.jit

# Required Python libraries:

* OpenCV, <https://pypi.org/project/opencv-python/>
* Numba, <http://numba.pydata.org/>
* Xml.dom, <https://docs.python.org/3/library/xml.dom.minidom.html>
* Xlrd, <https://pypi.org/project/xlrd/>
* spekpy

<https://bitbucket.org/spekpy/spekpy_release/wiki/Further%20information#markdown-header-specify-a-target-material-other-than-tungsten>

R Bujila, A Omar and G Poludniowski, *A validation of SpekPy: a software toolkit for modelling x-ray tube spectra*. Phys Med. 2020; 75:44-54.

A Omar, P Andreo and G Poludniowski, *A model for the energy and angular distribution of x rays emitted from an x-ray tube. Part I. Bremsstrahlung production*. Med Phys. 2020; 47(10):4763-4774

A Omar, P Andreo and G Poludniowski, *A model for the energy and angular distribution of x rays emitted from an x-ray tube. Part II. Validation of x-ray spectra from 20 to 300 kV*. Med Phys. 2020; 47(9):4005-4019

A Omar, P Andreo and G Poludniowski, *A model for the emission of K and L x rays from an x-ray tube*. NIM B 2018; 437:36-47.

G Poludniowski, *Calculation of x-ray spectra emerging from an x-ray tube. Part II. X-ray production and filtration in x-ray targets*. Med Phys 2007; 34(6):2175-86.

G Poludniowski and PM Evans, *Calculation of x-ray spectra emerging from an x-ray tube. Part I. electron penetration characteristics in x-ray targets*. Med Phys 2007; 34(6):2164-74.

G Poludniowski, et al., *SpekCalc: a program to calculate photon spectra from tungsten anode x-ray tubes*. Phys Med Biol 2009; 54(19):N433-8.

* Scipy, <https://pypi.org/project/scipy/>

Class methods description:

See class Diagram in Apendix 2.

**main.py**

Gets few algorithm parameters to set.

Initialize the Experiment class.

Launches the computation of the images for each position of the membrane and saves them.

**Experiment.py**

**Class Experiment**

+\_\_init\_\_

﻿Experiment class constructor.

Args:

exp\_dict (dict): dictionnary of simulation algorithm parameters

+﻿getStudyDimensions(self):

Calculates the study dimensions considereing the geometry of the set up, the field of view and the sample pixels oversampling

Returns:

None.

+﻿wavePropagation(self, waveToPropagate, propagationDistance, Energy, magnification):

Propagation of the wave

Args:

waveToPropagate (2d numpy array): incident wave.

propagationDistance (Float): propagation distance in m.

Energy (Float): considered eneregy in keV.

magnification (Float): magnification on the considered segment from the source.

Returns:

TYPE: DESCRIPTION.

**﻿**

+refraction(self,intensityRefracted, phi, propagationDistance,Energy, magnification):

Computes the intensity after propagation with ray-tracing

Args:

intensityRefracted (2d numpy array): intensity before propagation.

phi (2d numpy array): phase.

propagationDistance (Float): propagation distance in m.

Energy (Float): considered energy of the spectrum (in keV).

magnification (Float): magnification of the considered segment from the source.

Returns:

intensityRefracted2 (2d numpy array): DESCRIPTION.

Dx (2d numpy array): displacements along x.

Dy (2d numpy array): displacements along y.

+ defineCorrectValues(self, exp\_dict):

Initializes every compound parameters before calculations

Args:

exp\_dict (dictionnary): algorithm parameters.

Raises:

Exception: sample type not defined.

ValueError: experiment not found in xml file.

Returns:

None.

+﻿computeSampleAndReferenceImages(self, pointNum):

Compute intensity changes on the path of the previously difined experiment

to create all the images of the SBI experiment with Fresnel propagator

Returns:

SampleImage (2d numpy array): sample image simulated with sample and membrane.

ReferenceImage (2d numpy array): reference image simulated with membrane.

PropagImage (2d numpy array): propagation image simulated with only sample.

detectedWhite (2d numpy array): white image without membrane and sample.

﻿ +computeSampleAndReferenceImagesRT(self, pointNum):

Compute intensity changes on the path of the previously difined experiment

to create all the images of the SBI experiment with ray-tracing

Returns:

SampleImage (2d numpy array): sample image simulated with sample and membrane.

ReferenceImage (2d numpy array): reference image simulated with membrane.

PropagImage (2d numpy array): propagation image simulated with only sample.

detectedWhite (2d numpy array): white image without membrane and sample.

2d numpy array: real Dx from sample to detector.

2d numpy array: real Dy from sample to detector.

+﻿saveAllParameters(self,time0,expDict):

Saves all the experimental and algorithm parameters in a txt file

Args:

time0 (float): time at the beginning of the calculation.

expDict (dictionnary): dictionnary containing algorithm parameters.

Returns:

None.

**Source.py**

**Class Source**

+﻿setMySpectrum(self):

sets the source spectrum from xml file value for monochromatic or Spekpy for polychromatic

Returns:

None.

+﻿defineCorrectValuesSource(self):

gets all the source parameters from the xml file

Raises:

ValueError: Source not found in the xml file.

Returns:

None.

**Sample.py**

**Class Sample**

﻿ +defineCorrectValuesSample(self):

gets all sample parameters from xml file

Raises:

ValueError: Sample not found in the xml file.

Returns:

None.

+﻿getDeltaBeta(self, sourceSpectrum):

gets each materials delta and beta parameters for every energy of the spectrum

Args:

sourceSpectrum (list of tuples): spectrum of the source containing energy (in keV) and weights.

Returns:

None

**Class AnalyticalSample**

+﻿getMyGeometry(self,studyDimensions, studyPixelSize,oversamp, pointNum=0, number\_of\_positions=0):

Sets sample geometry maps for each of their materials geometry[material, x, y]

Args:

studyDimensions (tuple): (Dimx, Dimy).

studyPixelSize (float): voxel size in the considered plane in um (taking into account the oversampling).

oversamp (int): oversampling compared to detector pixels.

pointNum (int, optional): number of the calculated point (only useful to move the membrane between each point). Defaults to 0.

Raises:

ValueError: Could not define sample geometry means that the case was not found.

Returns:

None.

+﻿setWave(self,incidentWave, energy):

sets waves through the sample

Args:

incidentWave (2d numpy array): wave before the sample.

energy (float): considered energy in keV.

Raises:

Exception: Sample Geometry has the wrong nb of dim [material, x, y].

Returns:

disturbedWave (2d numpy array): wave after the sample.

+﻿setWaveRT(self,incidentIntensity,incidentphi, energy):

sets intensity and phase through the sample

Args:

incidentIntensity (2d numpy array): DESCRIPTION.

incidentphi (2d numpy array): DESCRIPTION.

energy (float): current energy in keV.

Returns:

disturbedIntensity (2d numpy array): intensity after attenuation of the sample.

disturbedPhi (2d numpy array): phase shift due to the sample.

**Detector.py**

**Class Detector**

+﻿defineCorrectValuesDetector(self):

Define detector parameters from the xml file

Raises:

ValueError: detector not found in xml file.

Returns:

None.

﻿ +detection(self,incidentWave,effectiveSourceSize):

Adds source and PSF blurrings, resamples to detector pixel size and add shot noise

Args:

incidentWave (2d numpy array): Intensity arriving to the detector.

effectiveSourceSize (float): source projected FWHM.

Returns:

detectedImage (2d numpy array): detected image.

**Simulation physical models:**

[text from the validation article to be published soon]

RAY TRACING

We start by considering the projection approximation which is most of the time valid when the object is small in comparison to the propagation distance and states that the description of the wave through an object can be described as phase shift and attenuation following straight lines through the object. It means that we can calculate the phase shift and the absorption B through the entire object at once with the following relations:

where x and y are the spatial coordinates in the plane perpendicular to the propagation direction z. and are the indexes for refraction and absorption introduced, is the wavelength and is the wave number.

The intensity after the object can then be calculated as :

From the phase shift, the refraction angle of each ray can be computed as:

And using that angle map, the redistribution of intensities on the detector plane after propagation over a distance z2 can be calculated as:

The intensity received in the detector plane is then convoluted with the source projected size and sampled at the detector pixel size. Then a convolution with the source blurring and point spread function of the detector was applied. Finally, shot noise is modeled using a Poisson distribution for a more realistic result.

FRESNEL PROPAGATOR

The second simulation tool developed is based on the wave optics model for x-rays, describing propagation as a linear operator usually called the Fresnel propagator.

Wave models can be derived starting from Maxwell's equation. From that physical model, wave descriptions for coherent x-rays have been developed and are detailed in \parencite{Paganin2006}.

The starting point of the model is to write the wave equation of the beam:

where is the wavelength, z is the coordinate along the propagation axis, x and y are the coordinate on the perpendicular plane, kx, ky and kz are the vectorial components of the wave vector. Before the wave reaches any object, it can be considered that the wave vector is along the propagation direction therefore, the previous equation becomes:

Then, when a wave goes through an object, it will be partly absorbed and phase shifted. A simple way to calculate those changes is if we use the projection approximation as in the ray tracing model.

Under that approximation, the wave after the object will be modified by a transmittance function that can be described as follows:

However this approximation has limits as it was shown in \parencite{morgan2010projection} and for larger objects or coherent sources, it may be biased.

Once the wave carries the object information, it will undergo other modification during propagation to the detector plane. The experiments of interest here being mainly limited to the Fresnel domain under the paraxial approximation, that propagation can be described by the ``Fresnel Propagator'' \parencite{Paganin2006}.

There are two formulations of that propagator, one in the real space and the second one in the Fourier space. The first one including a convolution product, it has a big computation cost while the second one is a simple multiplication in the Fourier space as shown in eq. \ref{eq:FresnelPropagator}

where is the Fresnel propagator in Fourier space and z is the distance of propagation.

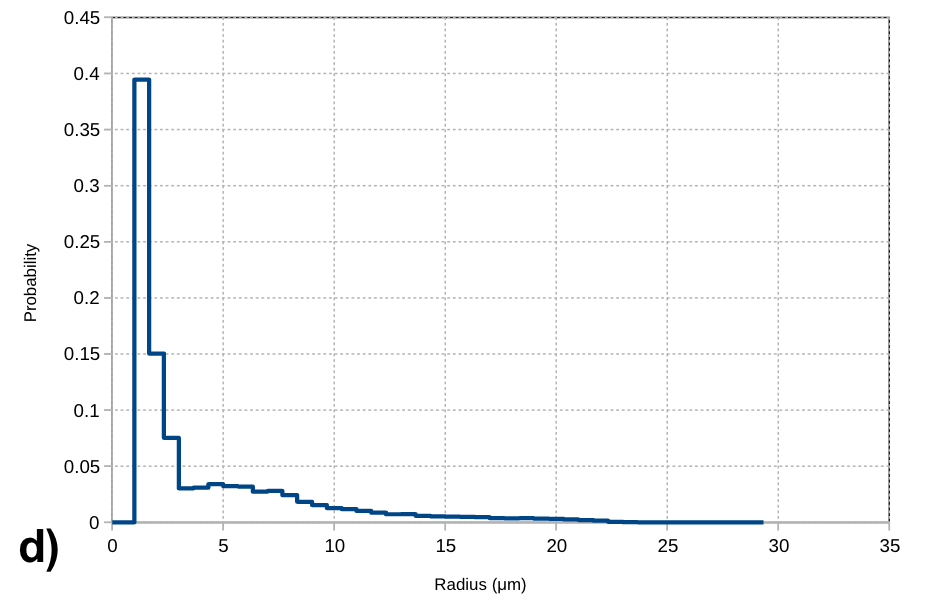
After propagation to the detector plane, the wave function is converted to intensity information with the following relationship:

Then, re-sampling and convolution with the source blurring and the detector's PSF were applied as well as the introduction of the shot noise the same way as for the ray-tracing code.

In the case of indirect detectors, the scintillator spectrum efficiency can be taken into account. It is then calculated as the proportion of absorbed photons in the plaque for each energy considered with the law: with k the wavenumber, the absorption coefficient of the scintillator material at the considered energy and t the thickness of the scintillator plaque.

Appendix 1:

Sphere radius distribution:



Appendix 2:

Class Diagram of the code architecture

