**X-ray speckle-based imaging simulation program**

Overview:

Objective: compute the intensity images obtained when an x-ray beam goes through one or several objects and propagates to a detector.

There are 2 calculation methods: 0ne 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 des matières

[Overview: 1](#_Toc67643627)

[Input parameters: 1](#_Toc67643628)

[Class methods description: (Not up to date) 3](#_Toc67643629)

[USER INSTRUCTIONS: 4](#_Toc67643630)

[Simulation physical models: 5](#_Toc67643631)

**Input parameters:**

To modify directly in the main (main.py):

- experimentName : [string]

- filepath: [string] containing the path of the folder for storing the outputs

- ﻿sampleSampling: [integer] How many times you want to over sample the object for a mor precise result. (Usually between 2 and 10).

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

- simulationType: [string] “Fresnel” or “RayT” too chose the type of calculation method

To modify in Experiment.xml

**<experiment>**

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

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

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

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

**<membraneName>**RandomSpheresSegmentedFromFileFe90**</membraneName>**

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

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

**<detectorName>**flatPanelClinic**</detectorName>**

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

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

**<windowName>**Be\_window**</windowName>**

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

**</experiment>**

For now sample type must always be “AnalyticalSample” because “voxelized sample” is not implemented.

All elementName must correspond to the elements name in their own xml file.

“windowName” and “plaqueName” are optional parameters. If you introduce a windowName, you must add a sample corresponding to this window in the Sample.xml file. In the simulation it corresponds to a thin film (filter) just at the exit of the source. plaqueName works the same and corresponds to the detector protective plaque (usually a thin carbon layer).

If you don’t want those 2 elements to be simulated simply don’t declare them in the experiment.

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

**<source>**

**<name>**clinic**</name>**

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

**<mySize** unit**=**"um"**>**50**</mySize>** #size of the emitting spot

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

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

**<myEnergySampling** unit**=**"keV"**>**5**</myEnergySampling>** #sampling for the calculation (if we calculate for every energy the computation time gets quite long)

**</source>**

/!\ the sourceVoltage parameter is independent from the path of the spectrum actually containing the spectrum values (i.e. changing that parameter wont change the considered spectrum).

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

**<source>**

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

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

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

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

**</source>**

To modify in Samples.xml: informations about the sample, the membrane and any other element present in the path of the beam. /!\ for every sample (or membrane) geometry stored in a file, the algorithm parameter “sampling” and the pixel size must agree with the one of the files.

**<sample>**

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

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

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

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

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

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

**</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>**

Some parameters can be changed or added depending on the way the geometry of the sample is defined.

The Type of the sample can be “sample\_of\_interest” or “membrane” or “thin\_film” or “volume”.

More details in “user instructions”

To modify in Detectors.xml:

**<detector>**

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

**<myDimensions>**

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

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

**</myDimensions>**

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

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

**<myEnergyLimit** unit**=**"keV"**>**30**</myEnergyLimit>**

**</detector>**

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

myEnergyLimit is the maximum energy that the detector will detect (raw estimation of scintillator efficiency limit or first bin of an energy resolved photon counting detector).

**USER INSTRUCTIONS:**

1. Determine the sampling requirements for your experiment
2. Set the Parameters in the main
3. Set the experiment global parameters in Experiment.xml in the different xml files
4. Set the detector parameters in Detector.xml
5. Set the source parameters in Source.xml
6. Set all the objects parameters in Sample.xml
7. Launch main.py
8. **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)

1. **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).

1. **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>**RandomSpheresSegmentedFromFileFe90**</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 received by 1 pixel of the detector.

1. **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>**

**<myEnergyLimit** unit**=**"keV"**>**30**</myEnergyLimit>**

**</detector>**

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

myEnergyLimit is the maximum energy that the detector will detect (raw estimation of scintillator efficiency limit or first bin of an energy resolved photon counting detector).

The detectors efficiency spectrum is not implemented yet because I do not have this information but could be “easily” added.

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.

1. **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>**

1. **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:**

2 Geometry functions are already implemented: a cylinder and a sphere. For each you can choose a radius and for the cylinder an orientation.

**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 mebrane for each point but should not be used for any other sample.

Additionnal 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.

Class methods description: (This part is NOT up to date)

The code also needs to be commented.

**Experiment**

**+getText(Node):** returns the content of the xmlfile corresponding to the node

**+getStudyDimensions():**

Computes the sampling factor from the given precision chosen for the angle of refraction in um, the propagation distance and the pixel size of the theoretical detector

sampling=myDetector.pixelSize/(precision\*distObjectToDetector)

Computes the dimensions of the study corresponding to the dimensions of the detector (ie the simulated image) times the sampling factor.

studyDimensions=myDetector.myDimensions\*sampling

Computes the study pixel size

studyPixelSize=myDetector.pixelSize/sampling

**+wavePropagation(waveToPropagate,propagationDist,Energy)**

Computes the wave function after propagation over a distance from an initial wave.

**+defineCorrectValues()**

Searches through the experiments xml file and gets the correct values for all the attributes

**Source**

**+getMySpectrum()**

Gets a list of tuples with each energy and their weights.

The weights correspond to flux and is not normalized yet.

**+defineCorrectValuesSource()**

**Sample**

**+defineCorrectValuesSample()**

**AnalyticalSample**

**+getMyGeometry()**

Depends a lot on the object

**+getDeltaBeta()**

Returns two 2D arrays containing tuples (energy, delta) for each energy for each material. Same for beta. Data are extracted from the excel doc /Volumes/ID17/speckle2/quenot/Simulations/CodePython/Samples/DeltaBeta/TablesDeltaBeta.xlsx. The data must be entered in the table with the same form!!!

**+setWave(incidentWave)**

Returns the wave function after the incidentWave went through the sample

For now only with the projection approximation.

**VoxelizedSample**

Not implemented yet

**Detector**

**+defineCorrectValuesDetector()**

**+detection(incidentWave)**

Computes the detected image from an incident wave function taking into account the detector’s parameters.

The oversampling is dealt with by summing the pixels of the oversized image to fill the detected one.

**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.

Appendix 1:

Sphere radius distribution:

