Introduction to X-ray attenuation and its implementation in gVXR

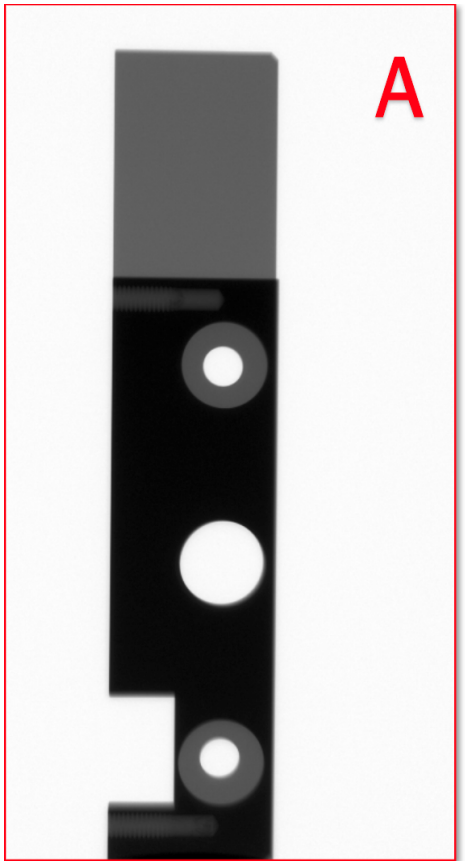
Summer school, ICTMS 2026

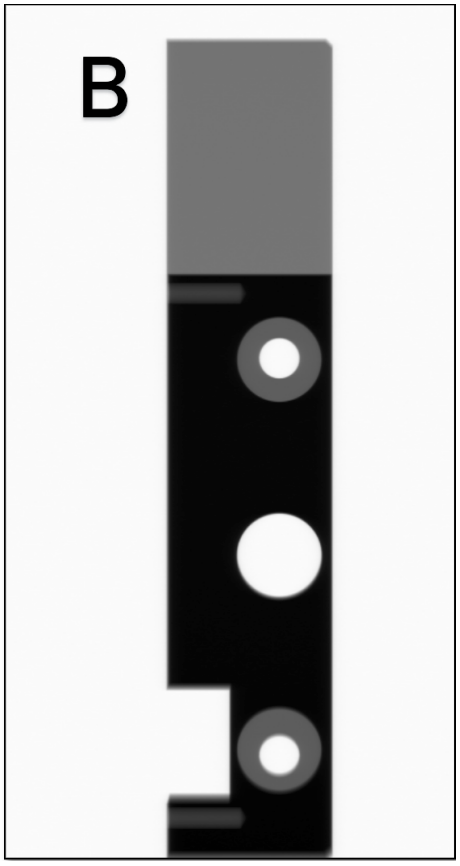
Prof Franck P. Vidal

Scientific Computing, Science and Technology Facilities Council

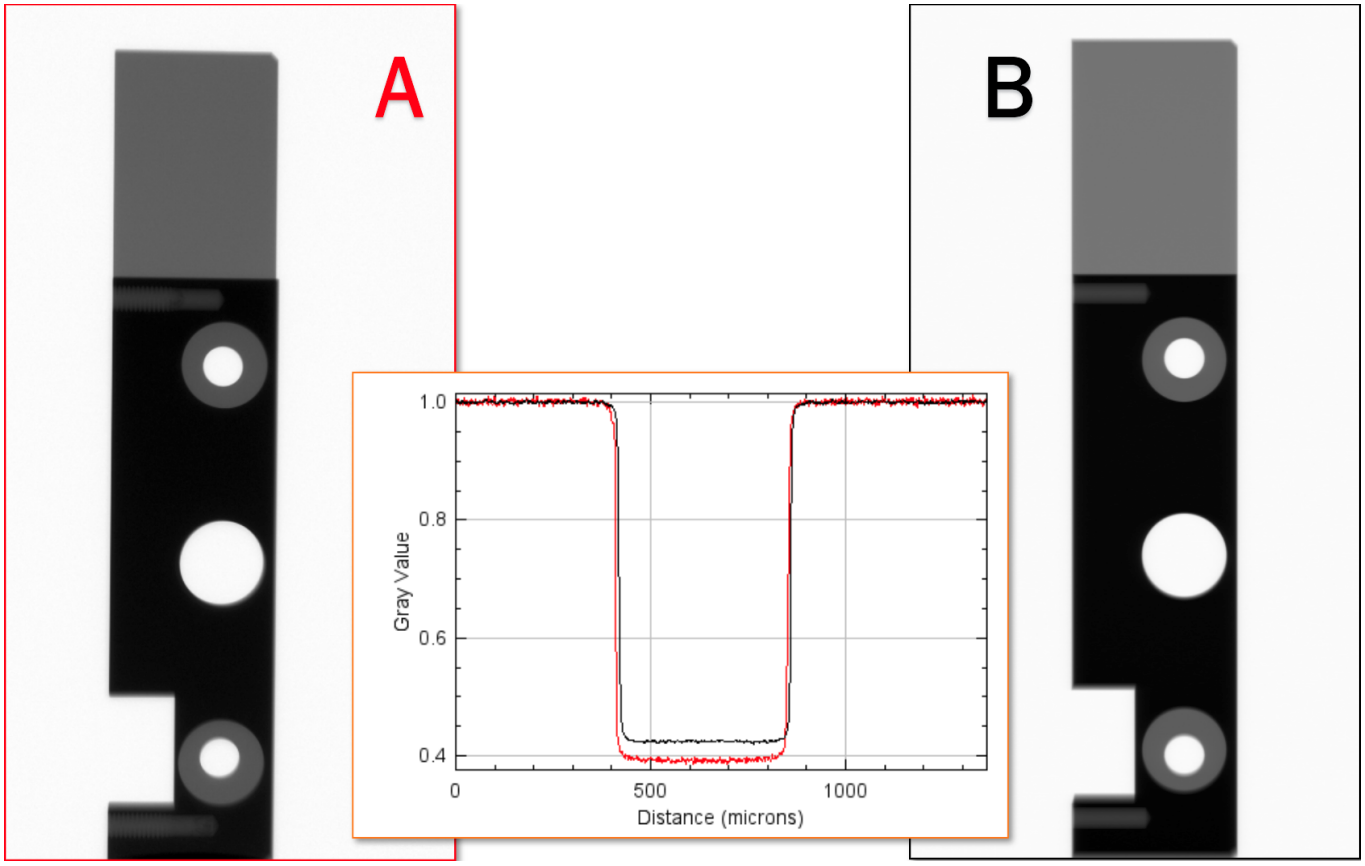
2026-07-03

Real vs simulated?





Real vs simulated?



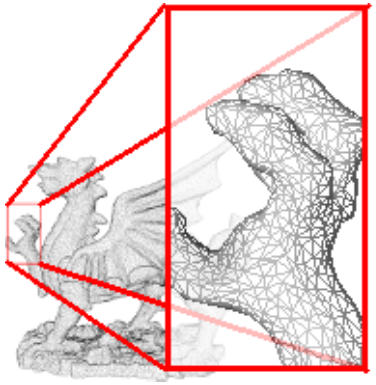
Aims of this session

- Discover what gVXR is;
- Become familiar with the Beer-Lambert law to compute the attenuation of X-rays by matter;
- Describe how the Beer-Lambert law is implemented in gVirtualXray;
- Learn how to generate realistic, XCT projection data using the **gVXR** package; and
- Focus on building simulation setups that mirror practical XCT scenarios.

Contents

1. What is gVirtualXray (gVXR)?

https://gvirtualxray.sourceforge.io/



Example of triangle meshes showed in wireframe.

- API (application programming interface)
- relying on the **Beer-Lambert law**
- to simulate X-ray images in realtime on a GPU (graphics processor unit)
- using triangular meshes.

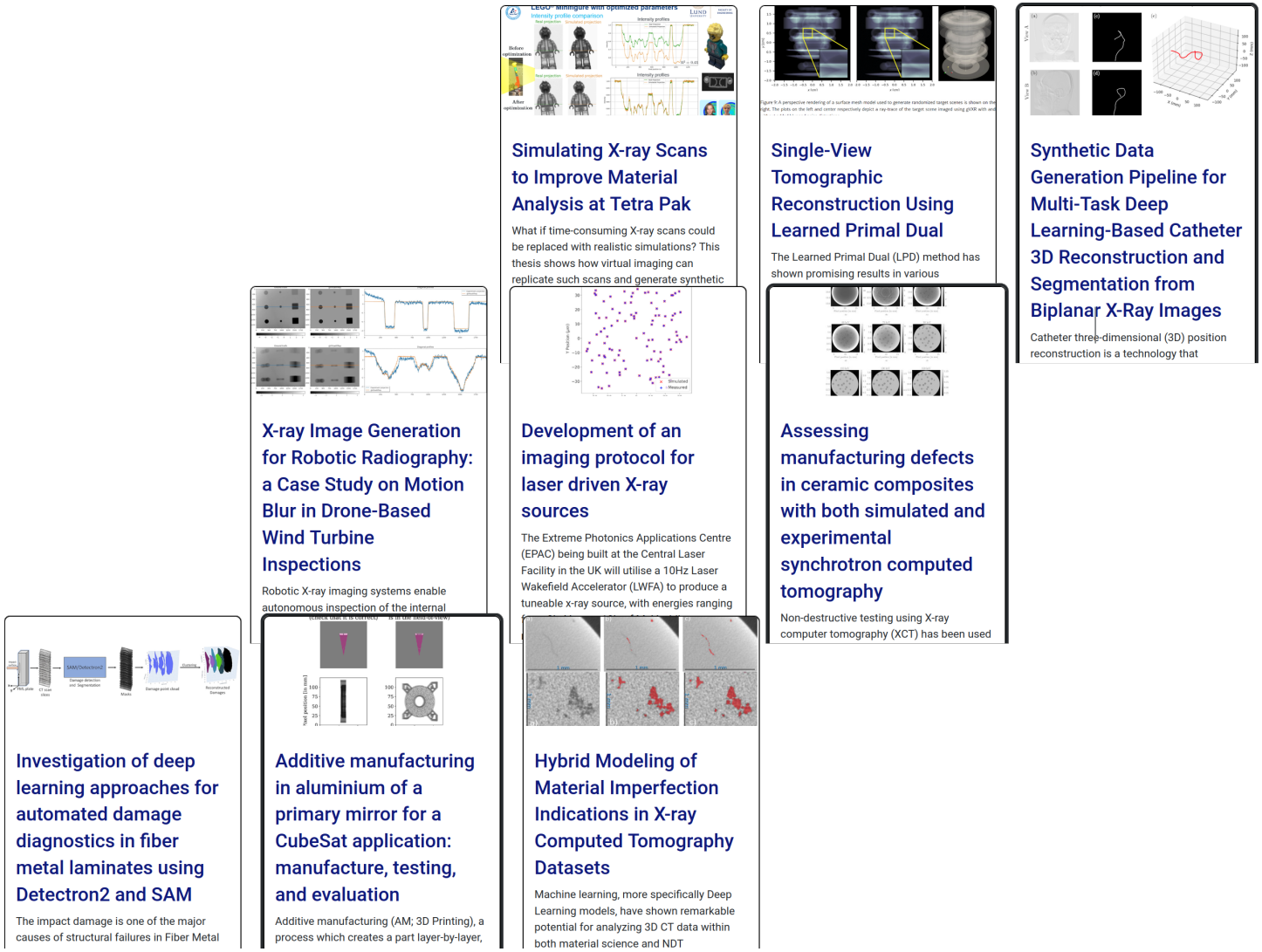
It is an alternative to Monte Carlo methods when scattering[1] can be ignored or speed is required.

[1] Scattering may be added in the future

What do people do with gVXR?

Used in a wide range of applications, including:

- real-time medical simulators;
- proposing new imaging methods;
- prototype algorithms;
- studying noise removal techniques;
- teaching particle physics and x-ray imaging (250 students / year);
- predicting image quality and artifacts;
- optimise scans; and
- a lot of ML/AI nowadays.



Example of applications.

How to get help

- Email me (Franck P. Vidal, STFC);
- Join the discord: https://discord.gg/eZj3CDBqCz;
- Raise an issue on GitHub: https://github.com/TomographicImaging/gVXR-Tutorials/issues;
- Open a ticket on SourceForge: https://sourceforge.net/p/gvirtualxray/tickets;
- ~~Subscribe to the mailing list: https://sourceforge.net/projects/gvirtualxray/lists/gvirtualxray-discuss;~~
- In Python, check the technical documentation, e.g.
  - *calling `help(gvxr)` for help on the Python package, or*
  - *something like `help (gvxr.function_name)` for a specific function.*
- It is also available at https://gvirtualxray.sourceforge.io/documentation/
  - *It is formatted in C++;*
  - *SimpleGVXR API also applies to Python.*

How to cite?

Vidal, F.P.; Afshari, S.; Ahmed, S.; Albiol, A.; Albiol, F.; Béchet, É.; Bellot, A.C.; Bosse, S.; Burkhard, S.; Chahid, Y.; et al. X-ray simulations with gVXR in education, digital twining, experiment planning, and data analysis. Nucl. Instrum. Methods Phys. Res. Sect. B 2025, 568, 165804. doi: 10.1016/j.nimb.2025.165804

gVXR

Nuclear Inst. and Methods in Physics Research, B 568 (2025) 165804

Contents lists available at ScienceDirect

Nuclear Inst. and Methods in Physics Research, B

journal homepage: www.elsevier.com/locate/nimb

Full Length Article

X-ray simulations with gVXR in education, digital twining, experiment planning, and data analysis

Franck P. Vidal [1,2,*], Shaghayegh Afshari [3], Sharif Ahmed [4], Alberto Albiol [5], Francisco Albiol [6], Éric Béchet [7], Alberto Corbí Bellot [8], Stefan Bosse [9,10], Simon Burkhard [11], Younes Chahid [12], Cheng-Ying Chou [3], Robert Culver [13], Pascal Desbarats [14], Lewis Dixon [2], Johan Friemann [15], Amin Garbout [16], Marcos García-Lorenzo [17], Jean-François Giovannelli [18], Ross Hanna [13], Clémentine Hatton [19], Audrey Henry [19], Graham Kelly [20], Christophe Leblanc [7], Alberto Leonardi [4], Jean Michel Létang [21], Harry Lipscomb [16], Tristan Manchester [4], Bas Meere [22], Claire Michelet [23], Simon Middleburgh [2], Radu P. Mihail [24], Iwan Mitchell [2], Liam Perera [4], Martí Puig [16,25], Malek Racy [26], Ali Rouwane [23], Hervé Seznec [23], Aaron Sújar [17], Jenna Tugwell-Allsup [27], Pierre-Frédéric Villard [28]

[1] UKRI-STFC Scientific Computing, Daresbury Laboratory, UK
[2] School of Computer Science & Engineering, Bangor University, UK
[3] Department of Biomechatronics Engineering, National Taiwan University, Taiwan
[4] Diamond Light Source, UK
[5] PRHLT Research centre, Universitat Politècnica València, Spain
[6] Instituto de Física Corpuscular, CSIC-Universitat de València, Spain
[7] Département d'Aérospatiale et Mécanique, Université de Liège, Belgium
[8] Escuela Superior de Ingeniería y Tecnología - Universidad Internacional de La Rioja, Spain
[9] Department of Computer Science, University of Koblenz, Germany
[10] Department of Mechanical Engineering, University of Siegen, Germany
[11] Federal Institute of Metrology METAS, Bern-Wabern, Switzerland
[12] UK Astronomy Technology Centre, Royal Observatory, UK
[13] The Manufacturing Technology Centre, UK
[14] Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, France
[15] Department of Industrial and Materials Science, Chalmers University of Technology, Sweden
[16] Henry Royce Institute, Henry Moseley X-ray Imaging Facility, Department of Materials, The University of Manchester, UK
[17] Grupo de Modelado y Realidad Virtual, Universidad Rey Juan Carlos, Spain
[18] Univ. Bordeaux, CNRS, Bordeaux INP, IMS, UMR 5218, France
[19] Scalian DS, France
[20] Shrewsbury and Telford Hospital NHS Trust, UK
[21] INSA-Lyon, Université Claude Bernard Lyon 1, CNRS, Inserm, CREATIS UMR 5220 U1294, France
[22] Department of Mechanical Engineering, Eindhoven University of Technology, Netherlands
[23] Univ. Bordeaux, CNRS, LP2I Bordeaux, UMR 5797, France
[24] Department of Computer Science and Engineering Technology, Valdosta State University, USA
[25] Department of Engineering Science, University of Oxford, UK
[26] Salford Royal Hospital, Manchester, UK
[27] Radiology Department, Betsi Cadwaladr University Health Board (BCUHB), Ysbyty Gwynedd, UK
[28] Université de Lorraine, CNRS, Inria, LORIA, France

ARTICLE INFO

Keywords:
X-ray imaging
Computed tomography
Simulation
GPU programming

ABSTRACT

gVirtualXray (gVXR) is an open-source framework that relies on the Beer–Lambert law to simulate X-ray images in real time on a graphics processor unit (GPU) using triangular meshes. A wide range of programming languages is supported (C/C++, Python, R, Ruby, Tcl, C#, Java, and GNU Octave). Simulations generated with gVXR have been benchmarked with clinically realistic phantoms (i.e. complex structures and materials) using Monte Carlo (MC) simulations, real radiographs and real digitally reconstructed radiographs (DRRs),

* Corresponding author at: UKRI-STFC Scientific Computing, Daresbury Laboratory, UK.
E-mail address: franck.vidal@stfc.ac.uk (F.P. Vidal).

https://doi.org/10.1016/j.nimb.2025.165804

Available online 28 August 2025

Front page of user article.png


gVXR

Implementation

- R&D started in the early 2000s, VXI[1] by Nicolas Freud (INSA-Lyon), and
- Its port on GPU when they became programmable (Bangor University, 2007);
- Not a ray tracer, but a rasterizer!
- Reimplementation from scratch as open source project on SOURCEFORGE
  - *Registered on 2013-12-01;*
  - *4544 commits on 2026-09-09 (v2.1.0). (average of 1.5 commit a working day in 12.5 years)*
  - *Implemented in* C++ *using* OpenGL
    - 173 C++ source files, 21036 lines of comments, 89950 lines of code
    - 123 C++ header files, 17482 lines of comments, 61203 lines of code
    - 69 CMake files, 1686 lines of comments, 5273 lines of code
    - 81 GLSL files, 4249 lines of comments, 2965 lines of code
    - 599 source code files, 49881 lines of comments, 199341 lines of code
  - *Wrapper for* python, R, Ruby, Rust, C#, Java, *and* GNU Octave.

[1] Freud et al. (2006), “Fast and robust ray casting algorithms for virtual X-ray imaging”

gVXR

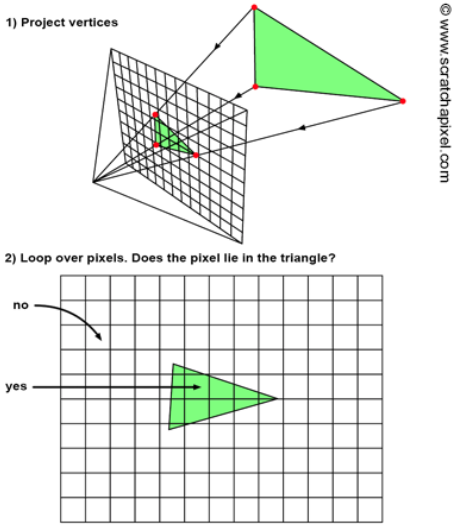
Rasterisation

✅ How it works

- Converts 3D geometry (triangles) into 2D pixels.
- Projects objects onto a screen from the camera's perspective.
- Determines which pixels belong to which triangles.
- Applies shading, textures, and lighting approximations.

⚡ Key strengths

- Very fast → ideal for real-time applications (games)
- Hardware optimized (GPUs have dedicated raster units)
- Efficient for large, complex scenes



(c) www.scratchapixel.com

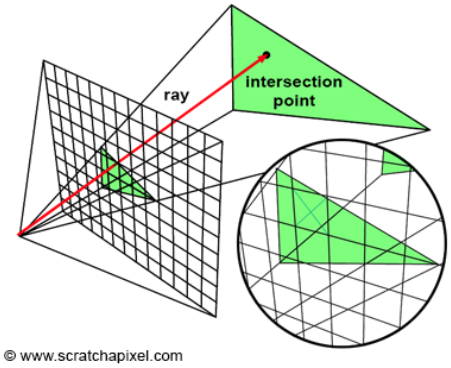
Ray Tracing (Physically-Based Rendering)

✅ How it works

- Simulates rays of light:
  - *Rays shoot from the camera into the scene*
  - *They bounce between surfaces*
  - *Lighting is calculated based on physical interactions*
- Handles reflection, refraction, shadows naturally

⚡ Key strengths

- Highly realistic lighting
- Accurate reflections, shadows, translucency
- Global illumination emerges naturally



(c) www.scratchapixel.com

Ray Tracing vs Rasterisation

Feature	Rasterisation	Ray Tracing
Speed	Very fast (real-time)	Slower (compute heavy)
Realism	Approximate	Highly realistic
Lighting	Precomputed / tricks	Physically accurate
Shadows	Shadow maps	Natural soft shadows
Reflections	Screen-space / cube maps	True reflections
Typical Use	Games, real-time apps	Film, high-end, RTX games

Source: Intel blog titled “Real Time Ray-Tracing: The End of Rasterization?”

Cross-platform

- Operating systems:  (binaries for MacOS are no longer distributed)
- CPUs:
  - intel
  - AMD
  - arm
- GPUs:
  - NVIDIA
  - AMD
  - intel
  -  *(software rendering with Mesa 3D)*
- Computers:
  - *On-board computers (OBCs)*
    - 
    -  NVIDIA Jetson
  - *Personal computers (PCs)*
    - 
    - 
  - *Data center and High-performance computing (HPC) systems*
    - STFC SCARF
    - SUPERCOMPUTING WALES UWCHGYFRIFIADURA CYMRU
    -  Google Colab
    - CODE OCEAN (used for reproducible research)
- Containerisation with docker

gVXR

Easy installation for python

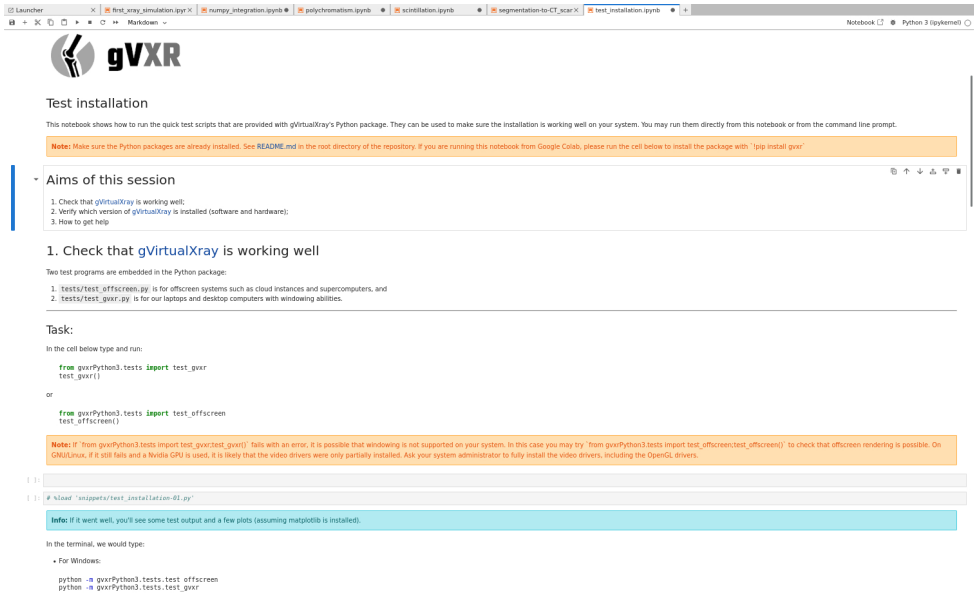
- Available on python Package Index : https://pypi.org/project/gvxr/
- For MS Windows (x86_64 architecture only);
- GNU/Linux (x86_64 and aarch64 architectures)

```
pip install gvxr
```

gVXR

Jupyter Notebook 1: Test installation

- test_installation.ipynb
- Run the quick test scripts provided with gVirtualXray's Python package to make sure the installation is working well on your system.
- Open in Colab



gVXR

2. What can we scan?

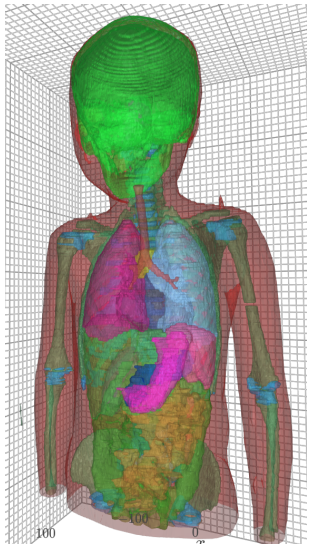
- Surface mesh from files (all common formats are supported, inc. STL)

Y Ddraig Goch, from the Welsh Red Dragon dataset for EG2011.

Corresponding STL file obtained using 3D laser scanning.

2. What can we scan?

- Surface mesh from files (all common formats are supported, inc. STL)
- Multi-part models using multiple different materials

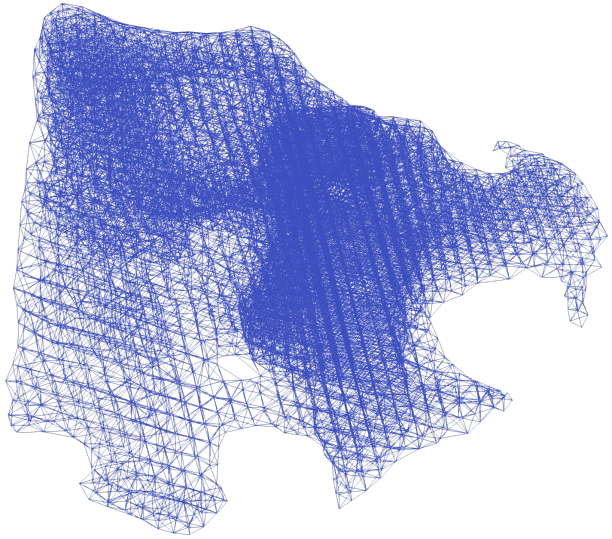


Paediatric phantom from the ERROR project (https://error.upatras.gr/).

2. What can we scan?

- Surface mesh from files (all common formats are supported, inc. STL)
- Multi-part models using multiple different materials
- Volume meshes (INP files from Abaqus, EXPERIMENTAL)

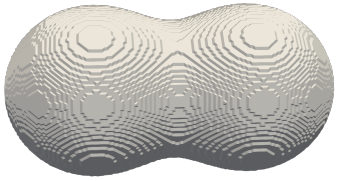
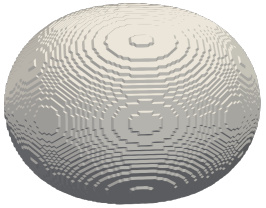
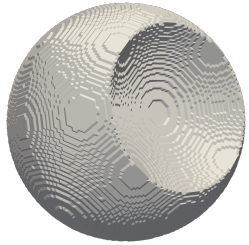
Tooth model



Volumetric mesh made of tetrahedrons.

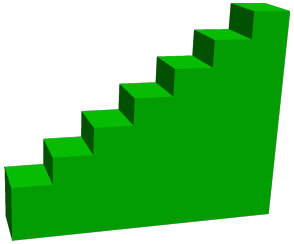
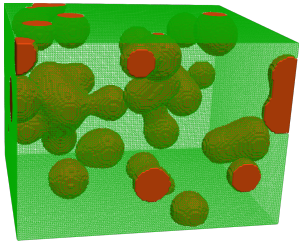
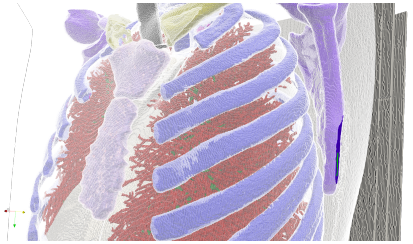
2. What can we scan?

- Surface mesh from files (all common formats are supported, inc. STL)
- Multi-part models using multiple different materials
- Volume meshes (INP files from Abaqus, EXPERIMENTAL)
- Implicit modelling (organic-looking n-dimensional isosurfaces)

Far from each other	Getting closer	Close from each other	Positive + negative density fields
			

2. What can we scan?

- Surface mesh from files (all common formats are supported, inc. STL)
- Multi-part models using multiple different materials
- Volume meshes (INP files from Abaqus, EXPERIMENTAL)
- Implicit modelling (organic-looking n-dimensional isosurfaces)
- Customisable built-in phantoms

Welsh dragon	Step wedge	Foam	Geometric shapes	Lungman
				
	Fully customisable	Fully customisable	Cuboids, spheres, cylinders, inc. boolean operations	To appear

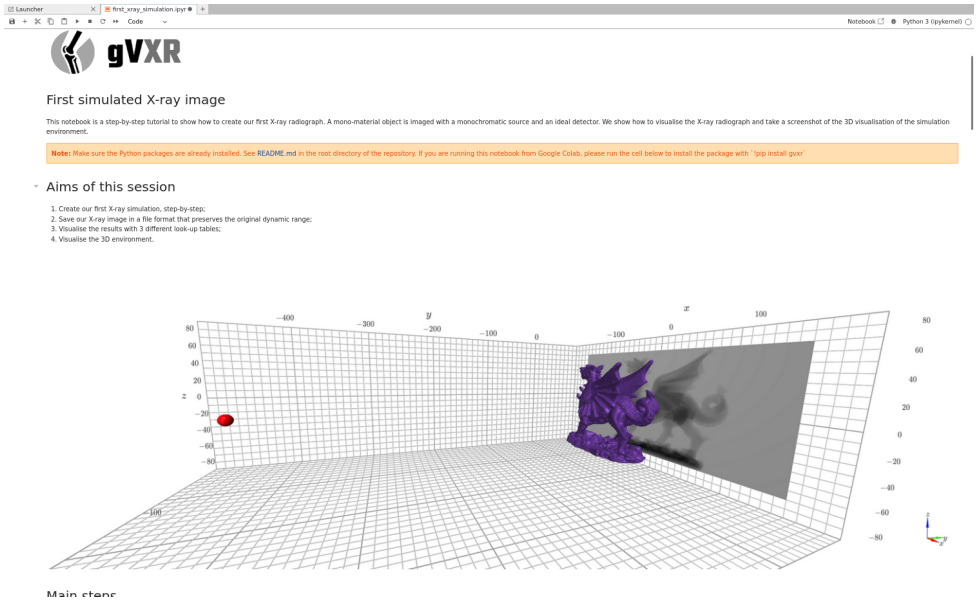
3. Beer-Lambert law (monochromatic)

$$I_{mono}(x,y) = E \times \mathbf{D}(E)\ \exp\left(-\sum_i \mu_i(E)\ \mathbf{d}_i(x,y)\right)$$

- E is the Energy E emitted by the source
- $\mathbf{D}(E)$ is the number of photons of Energy E emitted by the source
- $\mu_i(E)$ is the linear attenuation coefficient at Energy E of the i-th material;
- $\mathbf{d}_i(x,y)$ is the path length of the ray from the X-ray source to pixel (x,y) crossing the i-th material.

Jupyter Notebook 2: First X-ray simulation

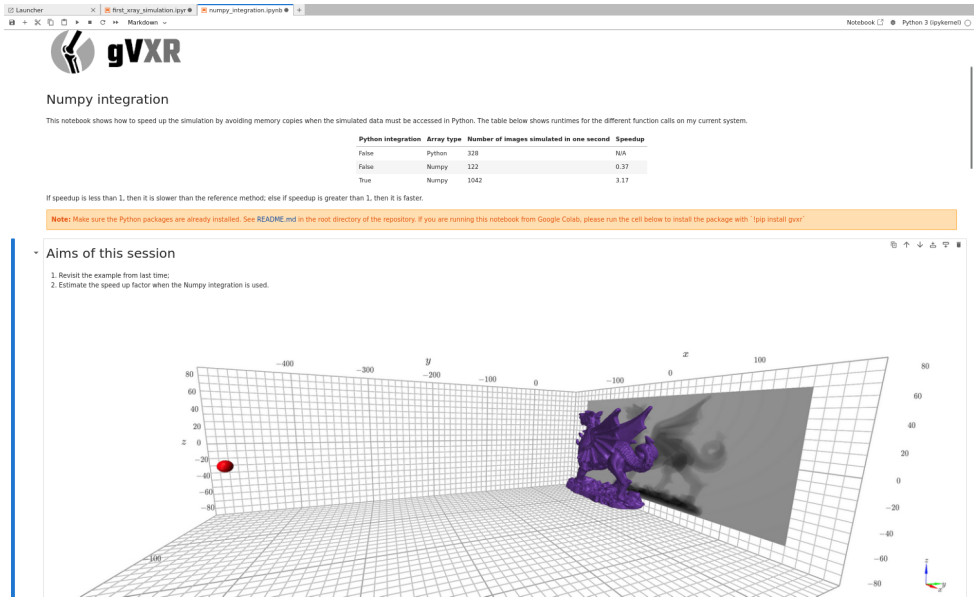
- first_xray_simulation.ipynb
- Explore the step-by-step notebook to create our first X-ray radiograph.
- A mono-material object is imaged with a monochromatic source and an ideal detector.
- We show how to visualise the X-ray radiograph and take a screenshot of the 3D visualisation of the simulation environment.
- Open in Colab



gVXR

Jupyter Notebook 3: Numpy integration

- numpy_integration.ipynb
- Experiment with the Numpy integration to speed up the simulation by avoiding memory copies when the simulated data must be accessed in Python.
- Open in Colab



gVXR

Why was this notebook important: Advantage of NumPy arrays over Python Lists

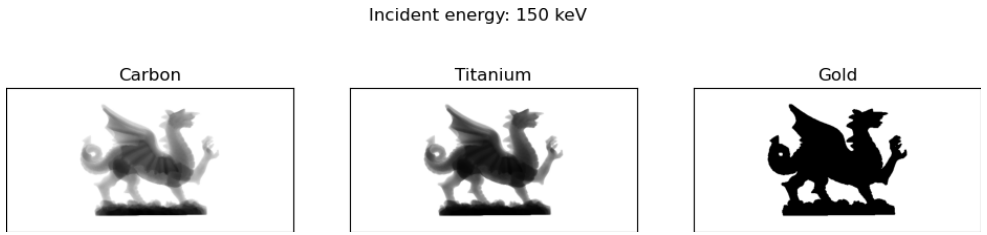
- Memory preallocation;
- Memory efficiency;
- Associated speed of reads and writes;
- Benefit of all the Numpy functionalities on arrays (e.g. `min()`, `max()`, `average()`, `std()`, `+`, `-`, `*`, `/`, etc.).

Python integration	Array type	Number of images simulated in one second	Compute time for one image	Compute time for 1609 images	Speedup
False	Python lists	320	3.12 ms	5 s	N/A
False	Python lists to Numpy arrays	124	8.09 ms	13 s	0.39
True	Numpy arrays	1269	0.79 ms	1 s	3.96

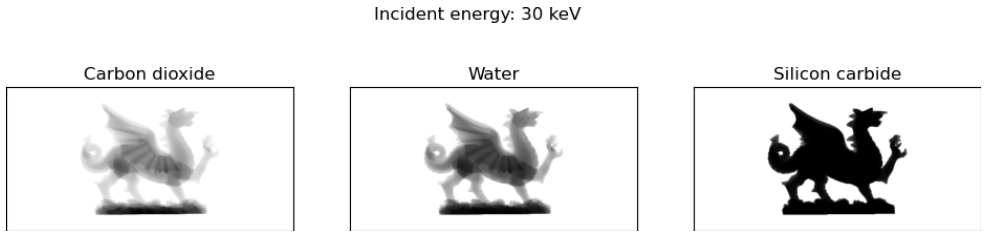
- Computed on:
  - *OS: GNU/Linux – openSUSE Leap 16.0*
  - *GPU: NVIDIA GeForce RTX 4060 Ti/PCIe/SSE2*
  - *OpenGL version: 4.3.0 NVIDIA 595.80*

4. Built-in functionalities

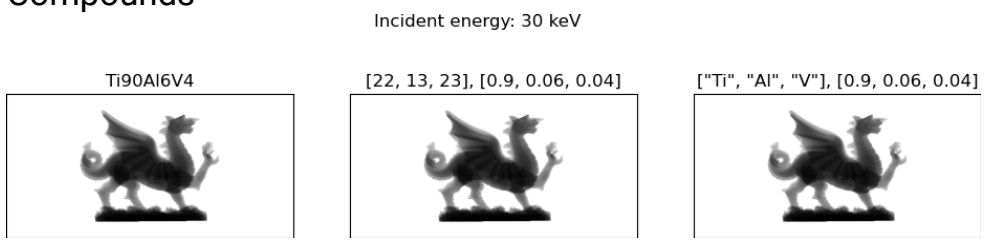
1. **Flexible material composition**;



Chemical elements



Compounds

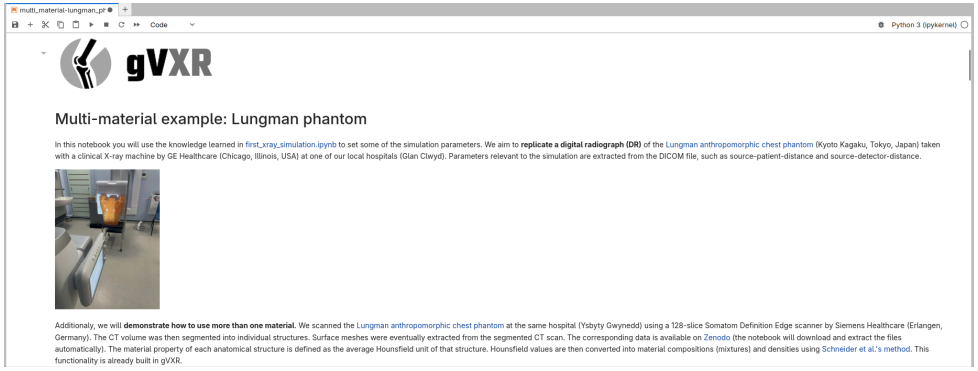


Mixtures

[1] Text in **bold characters** marks components validated against Monte Carlo simulations}

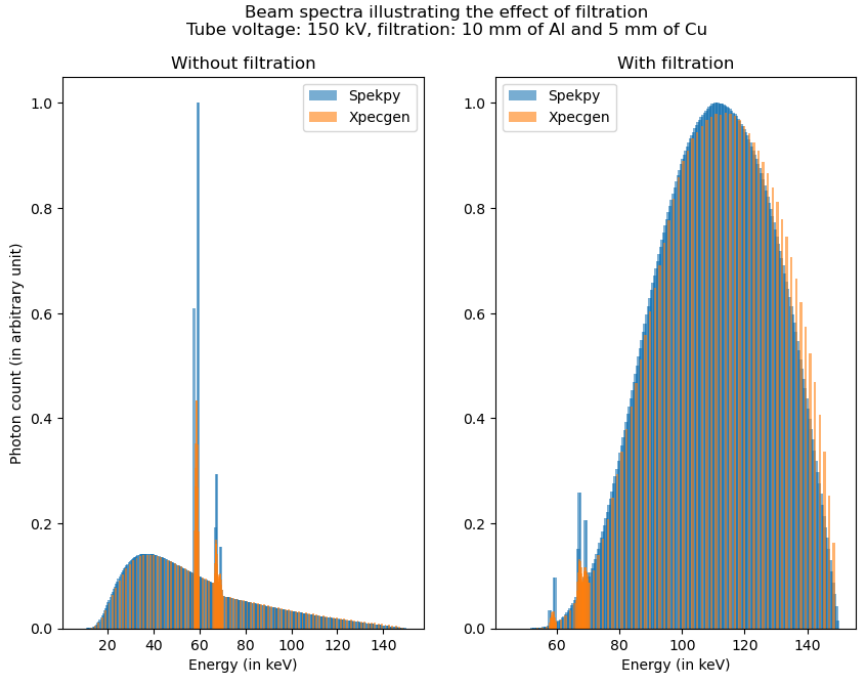
Jupyter Notebook 4: Multi-material sample

- multi_material-lungman_phantom.ipynb
- In this notebook we practice, amongst other things, how to set simulation parameters related to the X-ray source and detector; and
- demonstrate how to load several STL files and use them all in the simulation (multi-material).
- We aim to **replicate a digital radiograph (DR)** of the Lungman anthropomorphic chest phantom (Kyoto Kagaku, Tokyo, Japan) taken with a clinical X-ray machine by GE Healthcare (Chicago, Illinois, USA) at one of our local hospitals (Glan Clwyd).
- Parameters relevant to the simulation are extracted from the DICOM file, such as source-patient-distance and source-detector-distance.
- Open in Colab



4. Built-in functionalities

1. **Flexible material composition**;
2. **Monochromatic** & **polychromatic** spectra:



Spectrum comparision

[1] Text in **bold characters** marks components validated against Monte Carlo simulations}

4.2) Beer-Lambert law (polychromatic)

$$I_{poly}(x,y) = \sum_j E_j \times \mathbf{D}(E_j) \ \exp\left(-\sum_i \mu_i(E_j)\ \mathbf{d}_i(x,y)\right)$$

- **Polychromatism ($\sum_j$):** Images are integrated over J energy bins.

4. Built-in functionalities

1. **Flexible material composition**;
2. **Monochromatic** & **polychromatic** spectra:

- including kV and beam filtration;

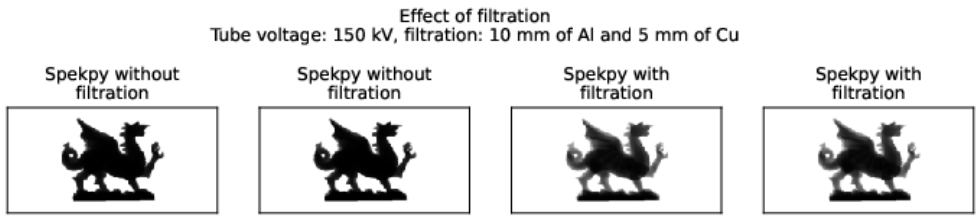
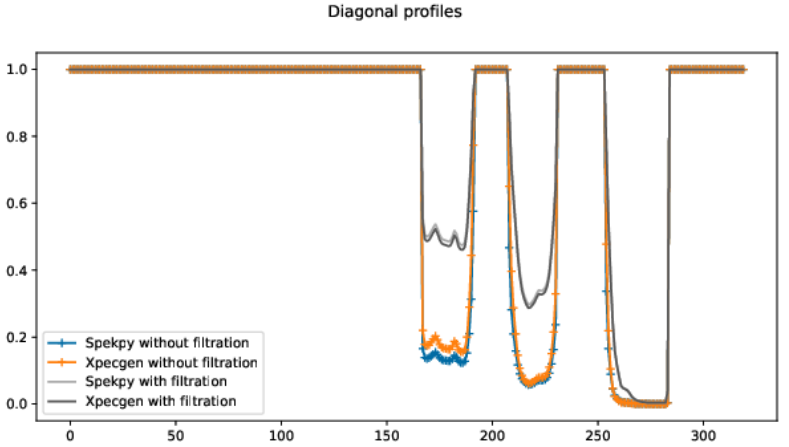


Image comparision (with/without filtration)

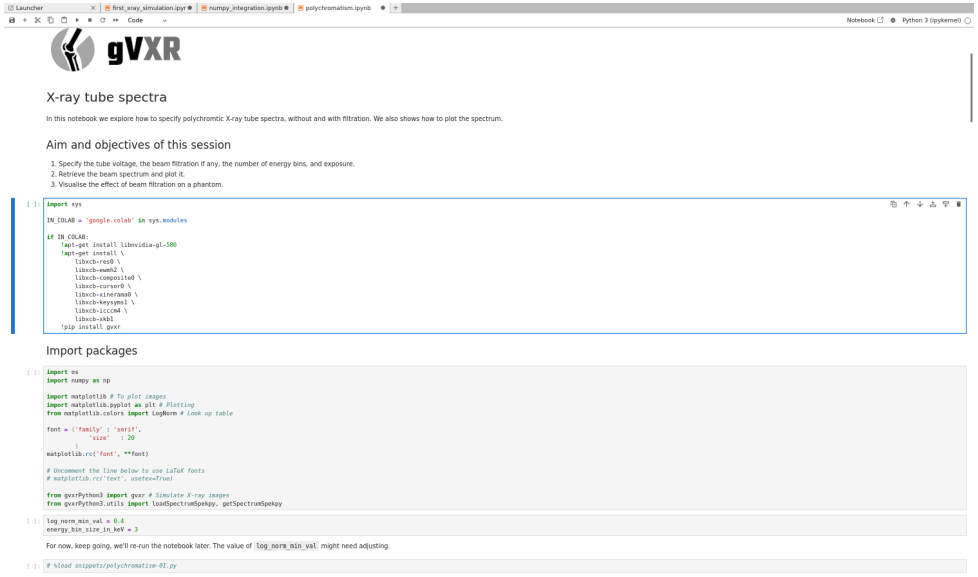


Profile comparision (with/without filtration)

[1] Text in **bold characters** marks components validated against Monte Carlo simulations}

Jupyter Notebook 5: Polychromtic X-ray tube spectra

- polychromatism.ipynb
- In this notebook we explore how to specify polychromtic X-ray tube spectra, without and with filtration.
- We also shows how to plot the spectrum.
- Open in Colab



X-ray tube spectra

In this notebook we explore how to specify polychromtic X-ray tube spectra, without and with filtration. We also shows how to plot the spectrum.

Aim and objectives of this session

1. Specify the tube voltage, the beam filtration if any, the number of energy bins, and exposure.
2. Retrieve the beam spectrum and plot it.
3. Visualise the effect of beam filtration on a phantom.

```
import sys

IN_COLAB = 'google.colab' in sys.modules

if IN_COLAB:
    !apt-get install libnvidia-gl-580
    !apt-get install \
        libxcb-res0 \
        libxcb-xwm0 \
        libxcb-composite0 \
        libxcb-cursor0 \
        libxcb-xinerama0 \
        libxcb-keysyms1 \
        libxcb-icccm4 \
        libxcb-xkb1
    !pip install gvxr
```

Import packages

```
import os
import numpy as np

import matplotlib # To plot images
import matplotlib.pyplot as plt # Plotting
from matplotlib.colors import LogNorm # Look up table

font = {'family' : 'serif',
        'size'   : 20
        }
matplotlib.rc('font', **font)

# Uncommant the line below to use LaTeX fonts
# matplotlib.rc('text', usetex=True)

from gvxrPython3 import gvxr # Simulate X-ray images
from gvxrPython3.utils import loadSpectrumSpekpy, getSpectrumSpekpy
```

```
log_norm_min_val = 0.4
energy_bin_size_in_keV = 3
```

For now, keep going, we'll re-run the notebook later. The value of `log_norm_min_val` might need adjusting.

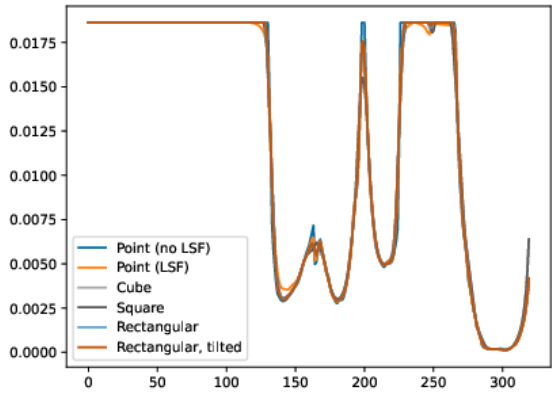
```
# %load snippets/polychromatism-01.py
```

Where to save the plots

4. Built-in functionalities

1. **Flexible material composition**;
2. **Monochromatic** & **polychromatic** spectra:
   - including kV and beam filtration;
3. **Parallel beams**, **point sources**, & **focal spots**[1];

Focal spot type	Size
Point	0.0 mm
Cube	0.25 × 0.25 × 0.25 mm
Square	0.25 × 0.25 mm
Rectangle	0.25 × 0.5 mm
Rectangle rotated by 25°	0.25 × 0.5 mm



Focal spot type comparision

[1] Text in **bold characters** marks components validated against Monte Carlo simulations}


gVXR

4.3) Beer-Lambert law (polychromatic + focal spot)

$$I_{FS}(x,y) = \sum_k \sum_j E_j \times \mathbf{D}(E_j) \ \exp\left(- \sum_i \mu_i(E_j) \ \mathbf{d}_{i,k}(x,y) \right)$$

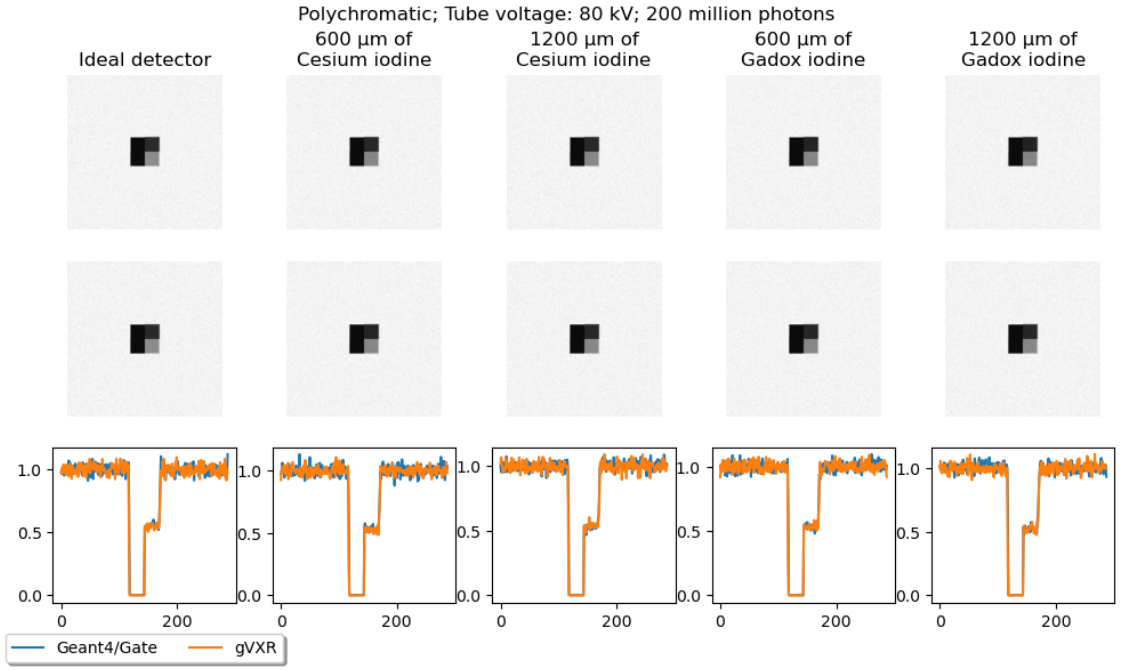
- **Focal spot** $(\sum_k)$**:** Images are integrated over K point sources.

4. Built-in functionalities

1. **Flexible material composition**;
2. **Monochromatic** & **polychromatic** spectra:

- including kV and beam filtration;

3. **Parallel beams**, **point sources**, & **focal spots**[1];
4. **Scintillation**;



Scintillator comparison (gVXR vs. Geant4)

[1] Text in **bold characters** marks components validated against Monte Carlo simulations}

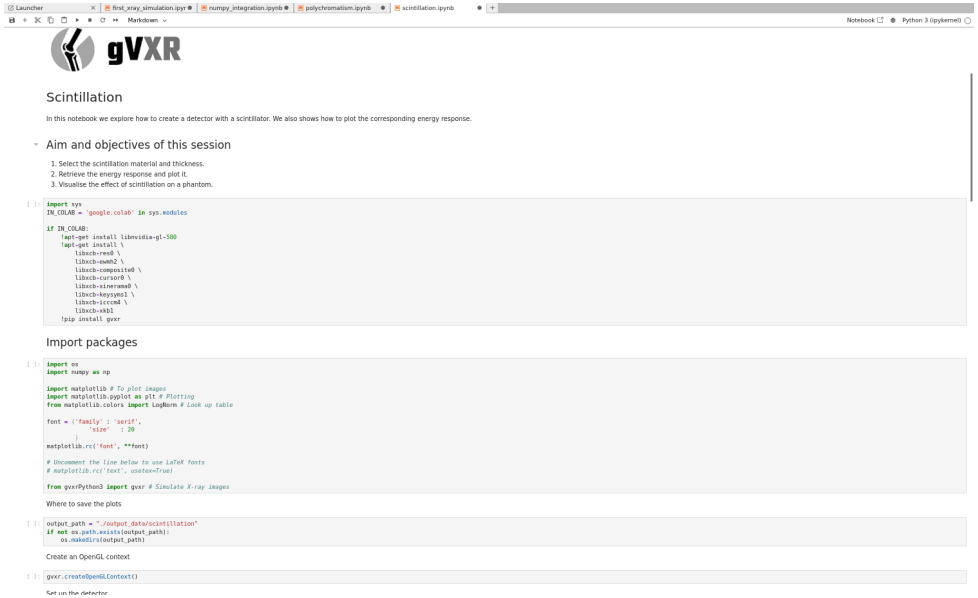
4.4) Beer-Lambert law (polychromatic + focal spot + scintillation)

$$I(x,y) = \sum_k \sum_j \mathrm{R}(E_j) \times \mathbf{D}(E_j) \ \exp\left(- \sum_i \mu_i(E_j) \ \mathbf{d}_{i,k}(x,y) \right)$$

- **Scintillator** $(\mathbf{R}(E_j))$**:** energy response of the detector, a lookup table.

Jupyter Notebook 6: Scintillation

- scintillation.ipynb
- In this notebook we explore how to create a detector with a scintillator.
- We also shows how to plot the corresponding energy response.
- Open in Colab



Launcher | first_xray_simulation.ipyr | numpy_integration.ipynb | polychromatism.ipynb | scintillation.ipynb

Markdown | Notebook | Python 3 (ipykernel)

gVXR

Scintillation

In this notebook we explore how to create a detector with a scintillator. We also shows how to plot the corresponding energy response.

Aim and objectives of this session

1. Select the scintillation material and thickness.
2. Retrieve the energy response and plot it.
3. Visualise the effect of scintillation on a phantom.

```
import sys
IN_COLAB = 'google.colab' in sys.modules

if IN_COLAB:
    !apt-get install libnvidia-gl-580
    !apt-get install \
        libxcb-res0 \
        libxcb-ewmh2 \
        libxcb-composite0 \
        libxcb-cursor0 \
        libxcb-xinerama0 \
        libxcb-keysyms1 \
        libxcb-icccm4 \
        libxcb-xkb1
    !pip install gvxr
```

Import packages

```
import os
import numpy as np

import matplotlib # To plot images
import matplotlib.pyplot as plt # Plotting
from matplotlib.colors import LogNorm # Look up table

font = {'family' : 'serif',
        'size'   : 20
        }
matplotlib.rc('font', **font)

# Uncomment the line below to use LaTeX fonts
# matplotlib.rc('text', usetex=True)

from gvxrPython3 import gvxr # Simulate X-ray images
```

Where to save the plots

```
output_path = "./output_data/scintillation"
if not os.path.exists(output_path):
    os.makedirs(output_path)
```

Create an OpenGL context

```
gvxr.createOpenGLContext()
```

Set up the detector

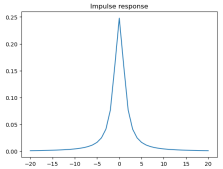
gVXR

4. Built-in functionalities

1. **Flexible material composition**;
2. **Monochromatic** & **polychromatic** spectra:

- including kV and beam filtration;

3. **Parallel beams**, **point sources**, & **focal spots**[1];
4. **Scintillation**;
5. Impulse response of detectors;



LSF

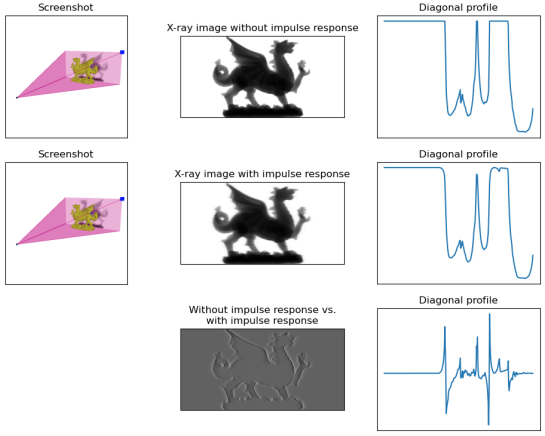


Image comparision (with/without blur)

4.5) Beer-Lambert law (polychromatic + focal spot + scintillation + PSF)

$$I_{PSF}(x,y) = PSF * \sum_k \sum_j \mathbf{R}(E_j) \times \mathbf{D}(E_j) \ \exp\left(- \sum_i \mu_i(E_j) \ \mathbf{d}_{i,k}(x,y) \right)$$

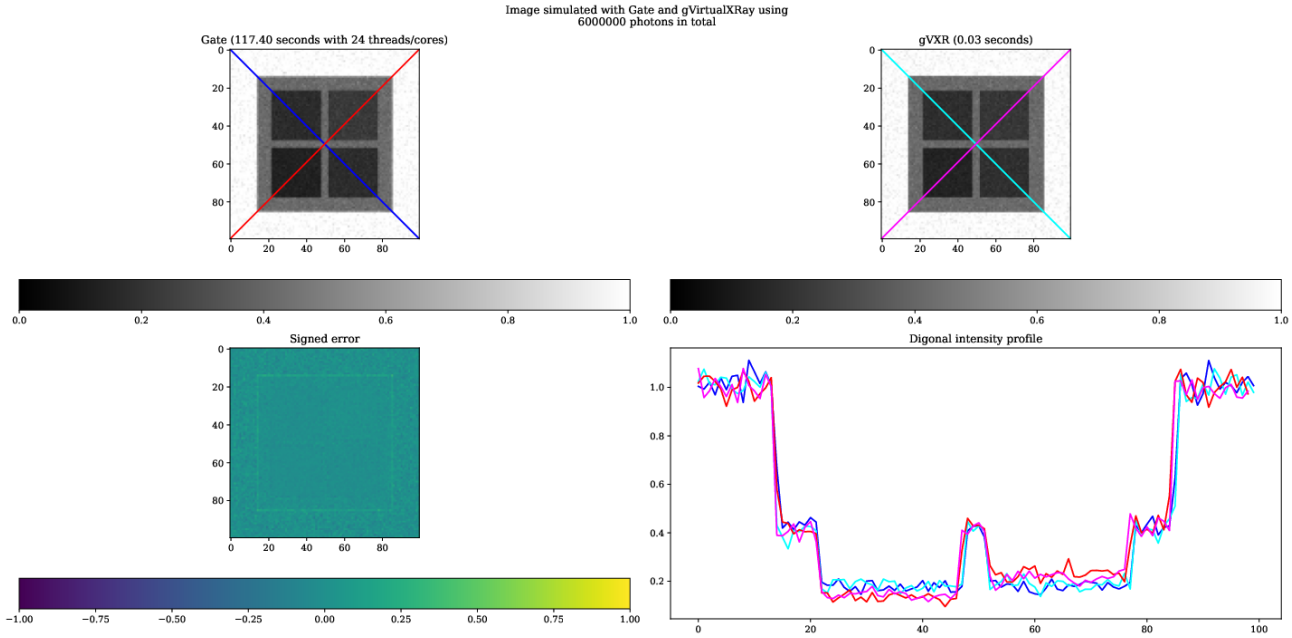
- **Detector blur** (PSF)**:** 2D impulse response of the detector, a low-pass convolution filter;
- $*$: spatial convolution operator.

4. Built-in functionalities

1. **Flexible material composition**;
2. **Monochromatic** & **polychromatic** spectra:

- including kV and beam filtration;

3. **Parallel beams**, **point sources**, & **focal spots**[1];
4. **Scintillation**;
5. Impulse response of detectors;
6. Noise (electronic & **photonic**).



Poisson noise comparison (gVXR vs. Geant4)

[1] Text in **bold characters** marks components validated against Monte Carlo simulations}

4.6) Beer-Lambert law (polychromatic + focal spot + scintillation + PSF + electronic noise & photonic noise)

$$I_{noisy}(x,y) = \text{Gauss}(\theta,\sigma) + gain \times \left(PSF * \sum_k \sum_j \mathbf{R}(E_j) \times \text{Poisson}\left(\mathbf{D}(E_j)\ \exp\left(-\sum_i \mu_i(E_j)\ \mathbf{d}_{i,k}(x,y) \right) \right) \right)$$

- **Electronic noise** $(\text{Gauss}(\theta,\sigma))$**:** additive Gaussian noise of average θ and standard deviation σ corresponding to the dark field image
- **Detector gain** $(gain)$**:** a multiplicative factor
- **Photonic noise** (Poisson)**:** Poisson noise that depends on the number of photons.

4.6) Final approximated model

- **Models of Poisson noise: computationally slow**;
- Complexity for previous slide: $\mathcal{O}(n^3)$:
  - *one call of* $\mathrm{Poisson}()$
  - *for every pixel*
    - for every energy bin
      - for every focal spot source point
- Approximation: $\mathcal{O}(n)$: one call for every pixel only.

$$e2p = \frac{\sum_j \mathbf{D}(E_j)}{\sum_j \mathbf{R}(E_j)\,\mathbf{D}(E_j)} = \frac{1}{p2e}$$

- **Scaling factor ($e2p$):** convert an integrated energy (I) in into an approximated number of photons;
- **Scaling factor ($p2e$):** convert an approximated number of photons in an integrated energy.

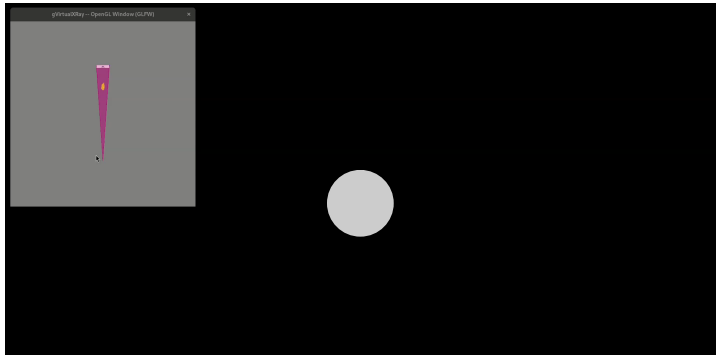
$$I_{final}(x,y) = \mathrm{Gauss}(\theta,\sigma) + gain \times (PSF * (p2e \times \mathrm{Poisson}\left(I(x,y) \times e2p\right)))$$

4. Built-in functionalities

1. **Flexible material composition**;
2. **Monochromatic** & **polychromatic** spectra:

- including kV and beam filtration;

3. **Parallel beams**, **point sources**, & **focal spots**[1];
4. **Scintillation**;
5. Impulse response of detectors;
6. Noise (electronic & **photonic**).
7. Interactive 3D visualization

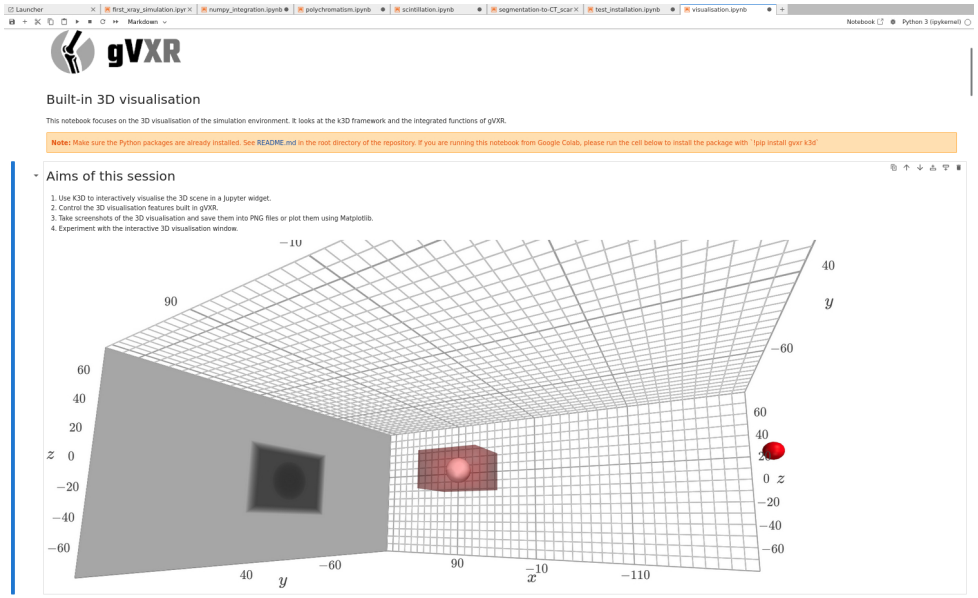


3D visualisation

[1] Text in **bold characters** marks components validated against Monte Carlo simulations}

Jupyter Notebook 7: 3D visualisation

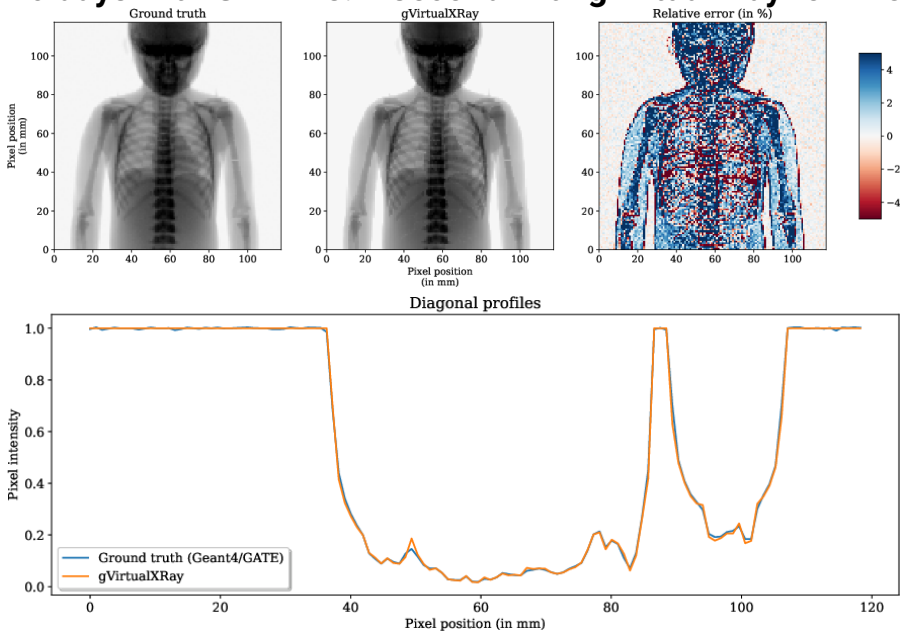
- visualisation.ipynb
- Get familiar with the three different 3D visualisation methods provided with gVXR, including:
  1. K3D to interactively visualise the 3D scene in a Jupyter widget,
  2. a customisable static 3D visualisation, and
  3. an interactive 3D visualisation window.
- In this notebook you will also create a multi-material sample.
- Open in Colab



5. Is gVXR validated? [1]

- Against state-of-the art **Monte Carlo simulation**, namely Geant4/Gate

10 days with GATE vs. 1 second with gVirtualXray for 128 × 128 pixels

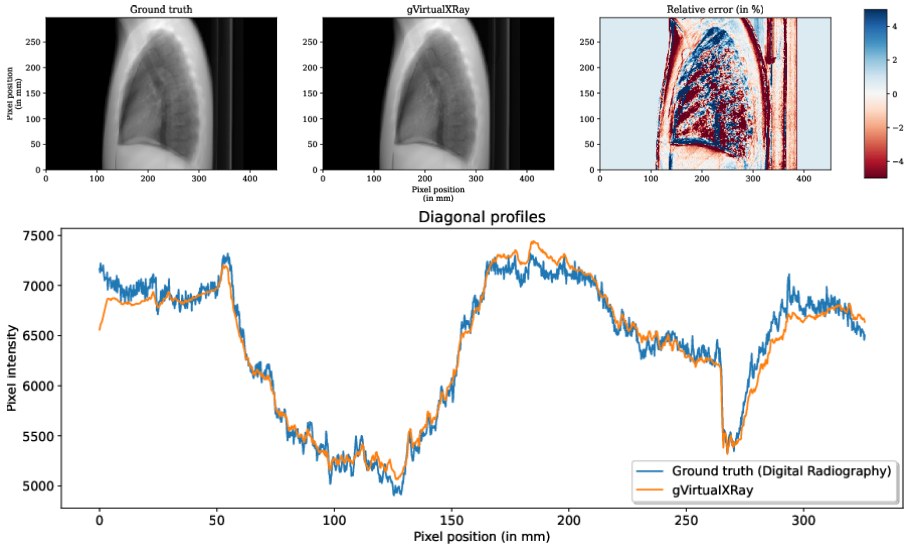


MAPE: 3.12%, ZNCC: 99.96%, SSIM: 0.99

[1] Pointon et al. (2023), “Simulation of X-ray projections on GPU: Benchmarking gVirtualXray with clinically realistic phantoms”

5. Is gVXR validated? [1]

- Against state-of-the art **Monte Carlo simulation**, namely Geant4/Gate
- Against **DRRs** computed from experimental data acquired with a **clinically utilized device**

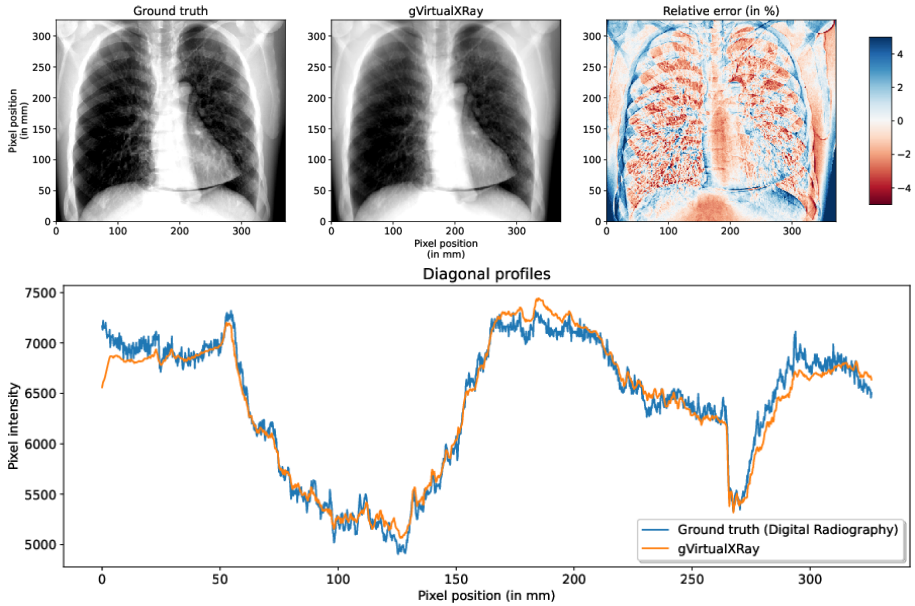


MAPE: 1.76%, ZNCC: 99.66%, SSIM: 0.98

[1] Pointon et al. (2023), “Simulation of X-ray projections on GPU: Benchmarking gVirtualXray with clinically realistic phantoms”

5. Is gVXR validated? [1]

- Against state-of-the art **Monte Carlo simulation**, namely Geant4/Gate
- Against **DRRs** computed from experimental data acquired with a **clinically utilized device**
- Against **digital radiographs** acquired with a **clinically utilized device**

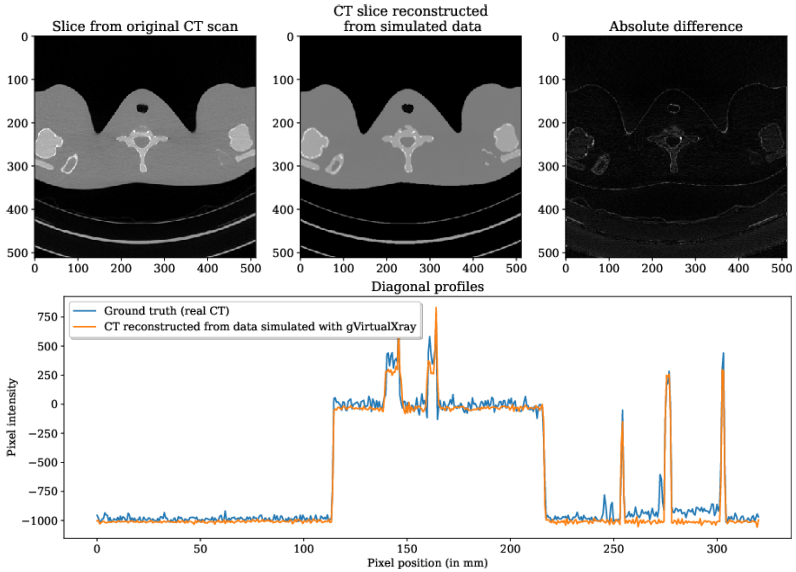


MAPE: 1.56%, ZNCC: 98.91%, SSIM: 0.94

[1] Pointon et al. (2023), “Simulation of X-ray projections on GPU: Benchmarking gVirtualXray with clinically realistic phantoms”

5. Is gVXR validated? [1]

- Against state-of-the art **Monte Carlo simulation**, namely Geant4/ Gate
- Against **DRRs** computed from experimental data acquired with a **clinically utilized device**
- Against **digital radiographs** acquired with a **clinically utilized device**
- Against **CT slices** from experimental data acquired with a **clinically utilized device**



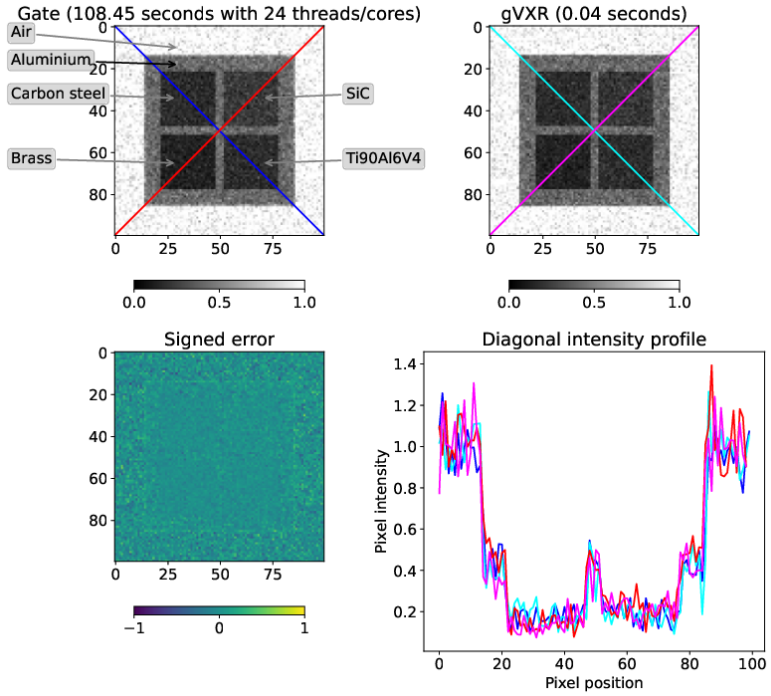
MAPE: 4.46%, ZNCC: 99.05%, SSIM: 0.82

[1] Pointon et al. (2023), “Simulation of X-ray projections on GPU: Benchmarking gVirtualXray with clinically realistic phantoms”

Is gVXR fast?

It was initially used in realtime medical VR for training purposes. So speed and accuracy are both requirements and tradeoffs must be found!

- Spectrum: polychromatic with 50 energy bins
- Resolution: 100 × 100 pixels
- Flat field images: 10
- Number of threads for the Monte Carlo simulation: 24
- CPU: AMD Ryzen 5900X
- GPU: NVIDIA GeForce RTX 4060 Ti



Comparison: Geant4 vs gVXR

::::::
::::::

Runtimes

Photon count	Execution time with Gate [in hh:mm:ss]	Execution time with gVXR [in ms]	Speed-up factor
100,000 × 11 00	:01:47	32	3,378
275,000 × 11 00	:01:47	32	3,307
775,000 × 11 00	:01:48	35	3,090
2,150,000 × 11 00	:01:51	34	3,273
6,000,000 × 11 00	:01:57	31	3,812
16,000,000 × 11 00	:02:22	36	3,958
46,000,000 × 11 00	:05:24	40	8,059
130,000,000 × 11 00	:13:24	36 2	2,433
360,000,000 × 11 00	:36:34	36 6	0,783
1,000,000,000 × 11 01	:41:07	31 1	94,208

6. CT simulation

```
void computeCTAcquisition(const std::string &   aProjectionOutputPath,
        const std::string &     aScreenshotOutputPath,
        unsigned int    aNumberOfProjections,
        float   aFirstAngle,
        bool    anIncludeLastAngleFlag,
        float   aLastAngle,
        unsigned int    aNumberOfWhiteImagesInFlatField,
        float   aPositionOfCentreOfRotationX,
        float   aPositionOfCentreOfRotationY,
        float   aPositionOfCentreOfRotationZ,
        const std::string &     aUnitOfLength,
        float   aAxisOfRotationX,
        float   aAxisOfRotationY,
        float   aAxisOfRotationZ,
        bool    anIntegrateEnergyFlag = true,
        unsigned int    aVerboseLevel = 0
    )
```

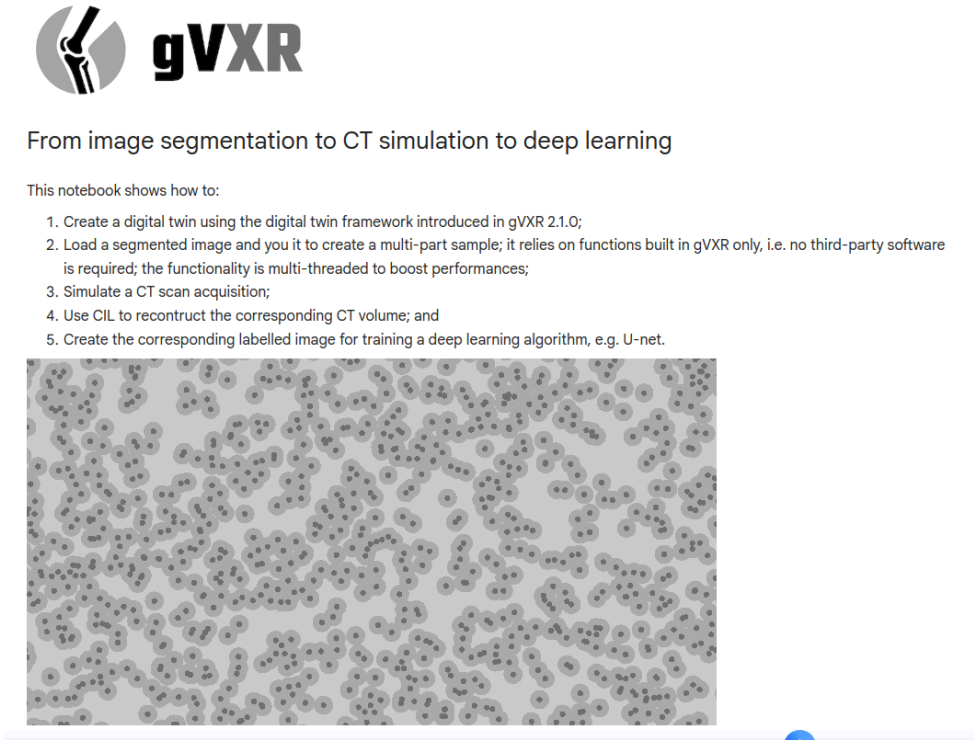
Jupyter Notebooks 8 & 9: Segmentation to simulation

- magnification-simulation
  - *We explore how to exploit the magnification to compute the pixel size in the object plane rather than flat panel plane,*
  - *how the source-to-detector distance affects the image quality in terms of noise, and*
  - *the impact of sample material, noise and polychromatism on CT reconstructions.*
  - Open in Colab
- magnification-reconstruction
  - *We will exploit the simulation data generated using `notebooks/magnification-simulation.ipynb`*
  - *to reconstruct the CT slices using the core imaging library (CIL),*
  - *use the zero-mean, unit-variance normalisation to rescale pixel values,*
  - *identify imaging artefacts by comparing pairs of images, and*
  - *extract intensity profiles to assess noise and cupping artefacts.*
  - Open in Colab

gVXR

Jupyter Notebook 10: Segmentation to simulation

- segmentation-to-CT_scan-simulation
- Create a CT reconstruction from data simulated using a segmented image to model the sample.
- Open in Colab



gVXR

Python script: Simulation to ML

- simulated_data_generation.py
- This script is very similar to the previous notebook. It makes use of the “XT H 225” twin.
- It will, however, run loops over:
  - *6 samples*
  - *SOD: 150 mm +/- 20%*
  - *Current: 160 uA +/- 30%*
  - *Exposures: [0.5, 1.0, 1.42, 2.0] seconds, and*
  - *Voltages in the range [180, 225].*
- In total, 3600 CT slices and corresponding labels will be generated.

7. Other ways to use gVXR

- See
  - *https://gvirtualxray.sourceforge.io/tutorials/*
  - *https://github.com/TomographicImaging/gVXR-Tutorials*
- Write the code to describe the previous parameters in pure python, C++, R, C#, Java, and GNU Octave.
- Or use a more human friendly {json} file.
- To simplify the python code.

JSON file (visualisation window)

```
{
    "Window size": [640, 480],
```

JSON file (X-ray source)

```
"Source": {
        "Position": [0.0, -1500.0, 0.0, "mm"],
        "Shape": "POINT",
        "Beam": {
            "Peak kilo voltage": 160.0,
            "Tube angle": 12.0,
            "mAs": 0.5,
            "filter": [
                ["Cu", 1.0, "mm"]
            ]
        }
    },
```

JSON file (X-ray detector)

```
"Detector": {
        "Position": [0.0, 400.0, 0.0, "mm"],
        "UpVector": [0, 0, -1],
        "RightVector": [-1.0, 0.0, 0.0],
        "NumberOfPixels": [512, 512],
        "Size": [256.0, 256.0, "mm"],
        "LSF": [0.0002, 0.0060, 0.0606, 0.2417,
            0.3829, 0.2417, 0.0606, 0.0060, 0.0002],
        "Scintillator": {
            "Material": "CsI",
            "Thickness": 500.0,
            "Unit": "um"
        }
    },
```

JSON file (Sample)

```
"Samples": [
        {
            "Label": "Dragon",
            "Path": "welsh-dragon-small.stl",
            "Unit": "mm",
            "Material": ["Mixture", "Al6Ti90V4"],
            "Density": 4.43,
            "Type": "inner"
        }
    ],
```

JSON file (CT scan acquisition)

```
"Scan": {
        "OutFolder": "projections",
        "GifPath": "screenshots",
        "NumberOfProjections": 200,
        "StartAngle": 0,
        "FinalAngle": 360,
        "IncludeLastAngle": true,
        "Flat-Field Correction": true,
        "NumberOfWhiteImages": 60,
        "CentreOfRotation": [0, 0, 0, "mm"],
        "RotationAxis": [0, 0, -1]
    }
}
```

python file (CT simulation)

```python
from gvxrPython3 import json2gvxr # Simulate X-ray images

json2gvxr.initGVXR("filename.json", renderer="OPENGL")
json2gvxr.initDetector()
json2gvxr.initSourceGeometry()
json2gvxr.initSpectrum()
json2gvxr.initSamples(verbose=0)
json2gvxr.initScan()
json2gvxr.doCTScan()
```

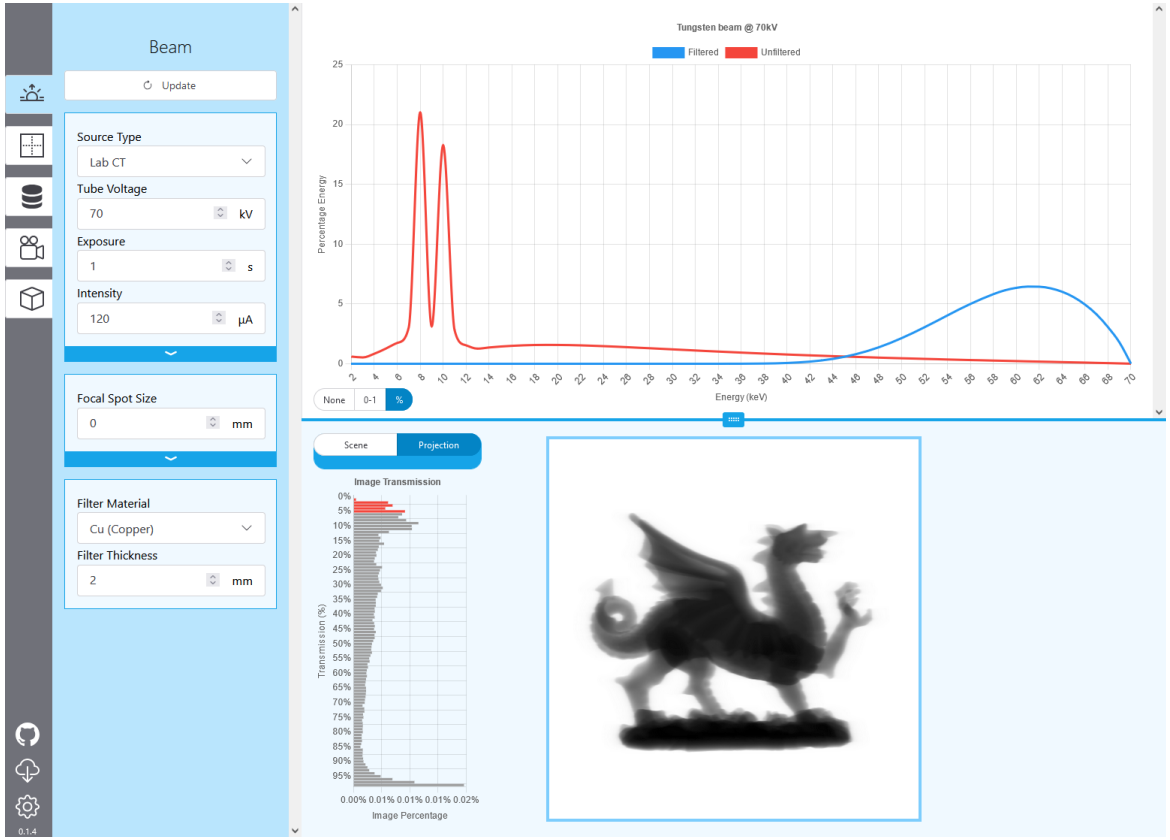
gVXR

python™ file (CT reconstruction)

```python
from gvxrPython3.JSON2gVXRDataReader import *
from cil.processors import TransmissionAbsorptionConverter
from cil.framework import AcquisitionData
from cil.recon import FDK
from cil.io import TIFFWriter

reader = JSON2gVXRDataReader(file_name="filename.json")

data_absorption = TransmissionAbsorptionConverter(
    white_level=data_original.max())(data_original)

acquisition_data = AcquisitionData(data_absorption,
    geometry=data_absorption.geometry)

acquisition_data.reorder(order="tigre")

ig = acquisition_data.geometry.get_ImageGeometry()

fdk =  FDK(acquisition_data, ig)
recon = fdk.run()
TIFFWriter(data=recon, file_name="slices", "out")).write()
```

gVXR

Modern Web-based GUI



Screenshot of WebCT

::::::: ::::::::
::::::: ::::::::

See video on YouTube

Download the latest release

Or visit https://webct.io/

gVXR

Thanks

- For your attentions;
- Our sponsors

gVXR