

# 1-gVirtualXRay\_vs\_Gate-detector\_realistic\_phantom

September 14, 2022

```
[1]: from IPython.display import display
      from IPython.display import Image
      from utils import * # Code shared across more than one notebook
```

```
[2]: output_path = "1-output_data/"

      if not os.path.exists(output_path):
          os.mkdir(output_path)
```

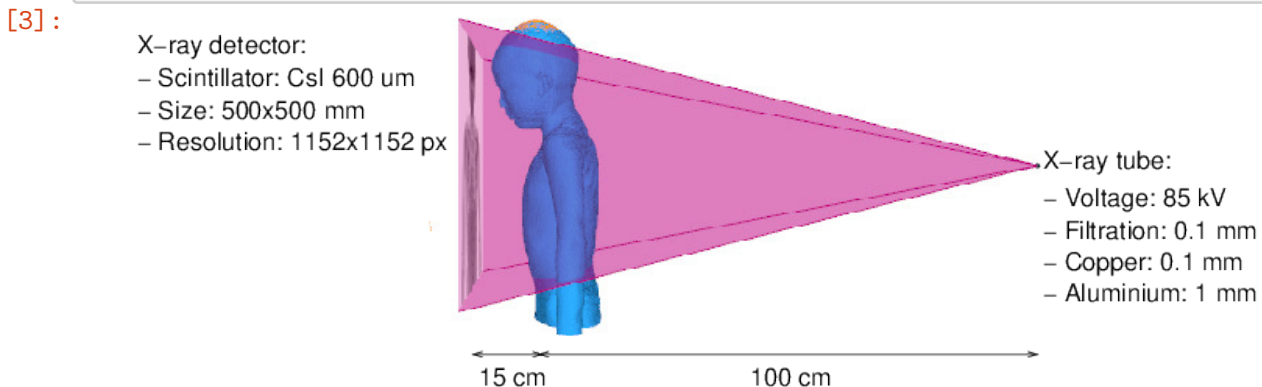
**Main contributors:** F. P. Vidal and J. M. Létang

**Purpose:** In this notebook, we aim to demonstrate that gVirtualXRay is able to generate analytic simulations on GPU comparable to images generated with the state-of-the-art Monte Carlo simulation packages. An anthropomorphic phantom is used. It corresponds to a 5-year old boy. We take into account i) a realistic beam spectrum (tube voltage and filtration) and ii) the energy response of the detector.

**Material and Methods:** We simulate an image with gVirtualXRay and compare it with a ground truth image. For this purpose, we use [Gate](#), a wrapper for CERN's state-of-the-art Monte Carlo simulation tool: [Geant4](#). The number of tracked particles is  $1e9$ .

In our simulation the source-to-object distance (SOD) is 1000mm, and the source-to-detector distance (SDD) is 1125mm. The beam spectrum is polychromatic. The voltage is 85 kV. The filtration is 0.1 mm of copper and 1 mm of aluminium. The energy response of the detector is considered. It mimics a 600-micron thick CsI scintillator.

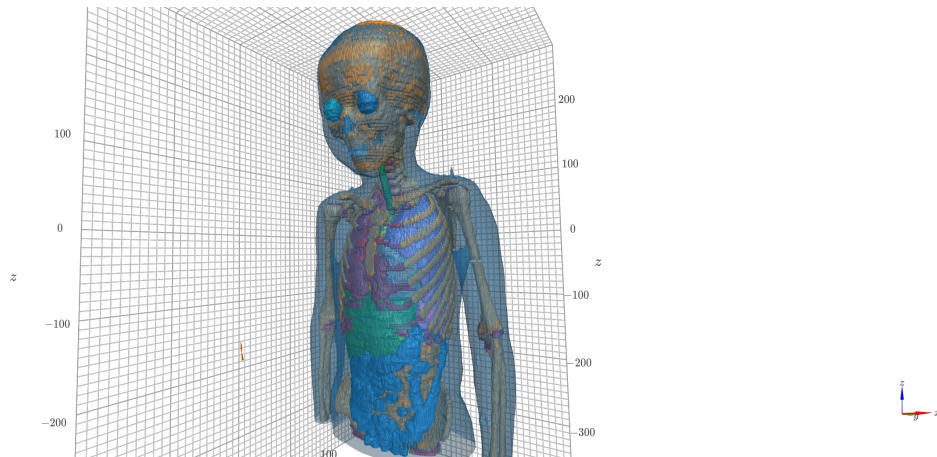
```
[3]: Image(filename="pediatric_phantom_data/pediatric-setup.png")
```



The sample is made of a 70x70x15mm box of water, in which 5 columns of 3 spheres of different radii (2, 3.5, and 5mm) have been inserted. A given material is associated to the spheres of each column (bone (cortical), blood (whole), brain (grey/white matter), breast tissue, and adipose tissue). The columns are ordered in decreasing density. We use the definitions of tissue substitutes provided in the [ICRU Report 44](#) by the [International Commission on Radiation Units and Measurements](#). The material composition is available at <https://physics.nist.gov/PhysRefData/XrayMassCoef/tab2.html>.

```
[4]: Image(filename=output_path+"/pediatric_model.png", width=800)
```

```
[4]:
```



**Results:** The calculations were performed on the following platform:

```
[5]: printSystemInfo()
```

OS:

```
Linux 5.14.21-150400.24.11-default
x86_64
```

CPU:

```
Intel(R) Core(TM) i9-9900K CPU @ 3.60GHz
```

RAM:

```
31 GB
```

GPU:

```
Name: NVIDIA GeForce RTX 2080 Ti
Drivers: 515.48.07
Video memory: 11 GB
```

The Monte Carlo simulation needed 5.36e6 HS06 seconds to complete.

It is equivalent to **8.68E+08** ms (i.e. ~10 days) on the system used. Only  $51 \pm 4$  ms was needed with the GPU used.

The [mean absolute percentage error \(MAPE\)](#), also known as mean absolute percentage deviation (MAPD), between the two simulated images is **MAPE 3.12%**. The [zero-mean normalised cross-correlation](#) is **99.96%**. The [Structural Similarity Index \(SSIM\)](#) is **0.99**.

As MAPE is relatively low (about 3%), SSIM is high (close to 1), and ZNCC is high (close to 100%), we can conclude that this X-ray image simulated with gVirtualXRay on GPU in milliseconds is comparable to the same Monte Carlo simulation that ran for days.

## 1 Import packages

```
[6]: %matplotlib inline

import os # Locate files

import math
import numpy as np # Who does not use Numpy?
import pandas as pd # Load/Write CSV files

import urllib, zipfile

import matplotlib

from matplotlib.cm import get_cmap
import matplotlib.pyplot as plt # Plotting
from matplotlib.colors import LogNorm # Look up table
from matplotlib.colors import PowerNorm # Look up table
import matplotlib.colors as mcolors

font = {'family' : 'serif',
        #'weight' : 'bold',
        'size'   : 22
        }
matplotlib.rc('font', **font)
# matplotlib.rc('text', usetex=True)

from scipy.stats import pearsonr # Compute the correlatio coefficient
from scipy.spatial import distance # Euclidean distance

from skimage.util import compare_images # Checkboard comparison between two
↳ images
from skimage.metrics import structural_similarity as ssim
from sklearn.metrics import mean_absolute_percentage_error as mape
from skimage.metrics import structural_similarity as ssim
from skimage.transform import resize # Resample the images

from tifffile import imread, imwrite # Load/Write TIFF files
```

```

import datetime # For the runtime

import k3d, base64
import SimpleITK as sitk
from stl import mesh
import random
from sitk2vtk import sitk2vtk

from gvxrPython3 import gvxr # Simulate X-ray images
from gvxrPython3 import json2gvxr # Set gVirtualXRay and the simulation up
    ↪ using a JSON file

from utils import * # Code shared across more than one notebook

```

SimpleGVXR 2.0.2 (2022-09-13T19:01:41) [Compiler: GNU g++] on Linux  
gVirtualXRay core library (gvxr) 2.0.2 (2022-09-13T19:01:41) [Compiler: GNU g++]  
on Linux

## 2 Reference image

We first load the reference image that has been simulated using [Gate](#) wrapper for CERN's [Geant4](#). Here we ignore scattering.

```

[7]: raw_reference = imread("pediatric_phantom_data/direct.tif")
Full_field = np.ones(raw_reference.shape) # Perfect full field image
Dark_field = np.zeros(Full_field.shape) # Perfect dark field image

```

Projections are then corrected to account for variations in beam homogeneity and in the pixel-to-pixel sensitivity of the detector. This is the projection with flat-field correction (**Proj**):

$$\mathbf{Proj} = \frac{I - D}{F - D} \quad (1)$$

where  $F$  (full fields) and  $D$  (dark fields) are projection images without sample and acquired with and without the X-ray beam turned on respectively.

We now apply the flat-field correction to Image.

```

[8]: gate_image = (raw_reference - Dark_field) / (Full_field - Dark_field)
    # gate_image = raw_reference / np.mean(Full_field)

```

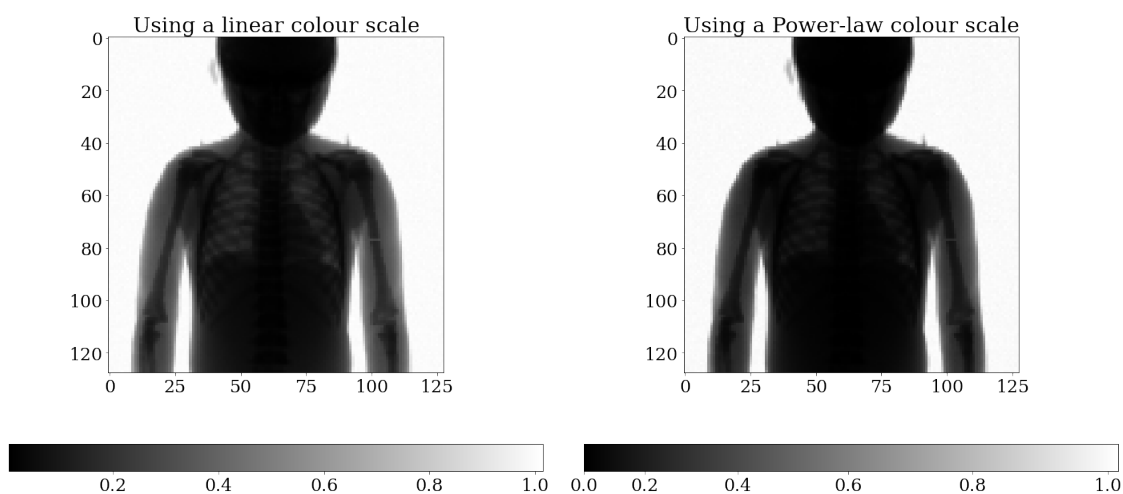
We plot the image using a linear look-up table and a power-law normalisation.

```

[9]: displayLinearPowerScales(gate_image,
    "Image simulated using Gate wrapper for CERN's Geant4",
    output_path + "/reference_from_Gate-paediatrics")

```

Image simulated using Gate wrapper for CERN's Geant4



### 3 Setting up gVirtualXRay

Before simulating an X-ray image using gVirtualXRay, we must create an OpenGL context.

```
[10]: json2gvxr.initGVXR("notebook-1.json", "OPENGL")
```

Create an OpenGL context: 800x450

```
Wed Sep 14 09:36:49 2022 ---- Create window (ID: -1)
Wed Sep 14 09:36:49 2022 ---- Initialise GLFW
Wed Sep 14 09:36:49 2022 ---- Create an OpenGL window with a 3.2 context.
Wed Sep 14 09:36:51 2022 ---- Make the window's context current
Wed Sep 14 09:36:51 2022 ---- Initialise GLEW
Wed Sep 14 09:36:51 2022 ---- OpenGL vendor: NVIDIA Corporation
Wed Sep 14 09:36:51 2022 ---- OpenGL renderer: NVIDIA GeForce RTX 2080
Ti/PCIe/SSE2
Wed Sep 14 09:36:51 2022 ---- OpenGL version: 3.2.0 NVIDIA 515.48.07
Wed Sep 14 09:36:51 2022 ---- Use OpenGL 4.5.
Wed Sep 14 09:36:51 2022 ---- Initialise the X-ray renderer if needed and if
possible
```

#### 3.1 X-ray source

We create an X-ray source. It is a point source.

```
[11]: json2gvxr.initSourceGeometry()
```

Set up the beam

```
Source position: [0.0, 1000.0, 0.0, 'mm']
Source shape: PointSource
```

## 3.2 Spectrum

The spectrum is polychromatic.

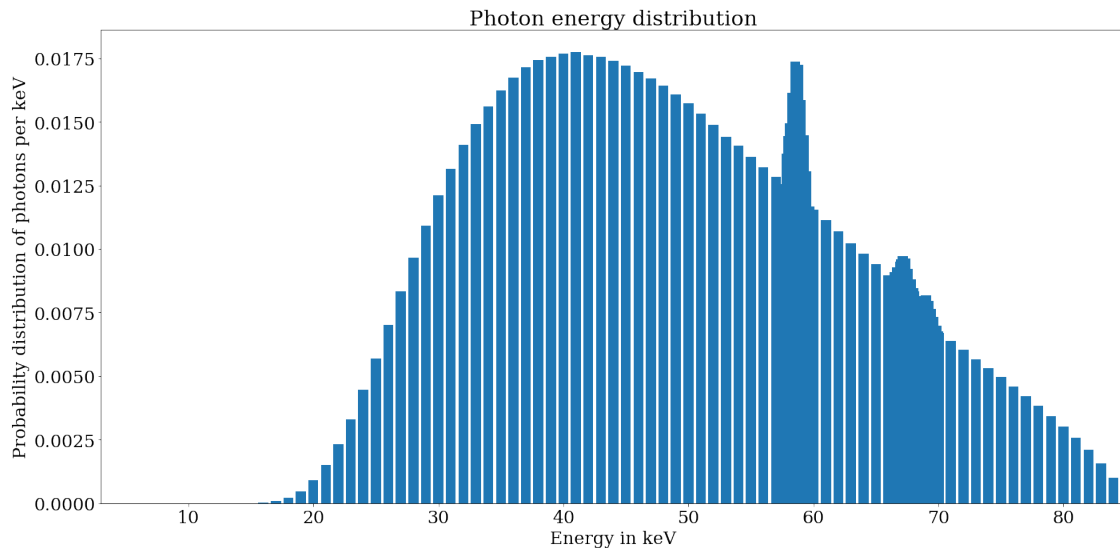
```
[12]: spectrum, unit, k, f = json2gvxr.initSpectrum(verbose=0)
      energy_set = sorted(spectrum.keys())

      count_set = []

      for energy in energy_set:
          count_set.append(spectrum[energy])
```

Plot the spectrum

```
[13]: k *= 1000
      plotSpectrum(k, f, output_path + "/spectrum-paediatrics", xlim=[np.min(k), np.
      ↪max(k)])
```



## 3.3 Detector

Create a digital detector

```
[14]: json2gvxr.initDetector()
```

Set up the detector

```
Detector position: [0.0, -150.0, 0.0, 'mm']
Detector up vector: [0, 0, -1]
Detector number of pixels: [128, 128]
Energy response: Gate_data/responseDetector.txt in MeV
Pixel spacing: [3.90625, 3.90625, 'mm']
```

Wed Sep 14 09:36:52 2022 ---- Initialise the renderer

### 3.4 Model the energy response of the detector

Load the energy response

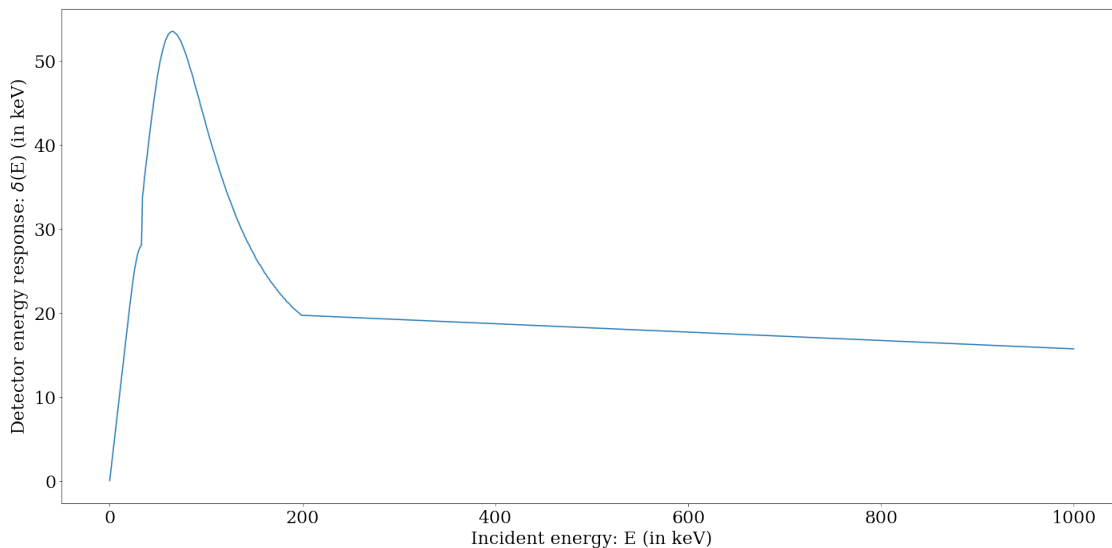
```
[15]: detector_response = np.loadtxt("Gate_data/responseDetector.txt")
```

Display the energy response

```
[16]: plt.figure(figsize= (20,10))
# plt.title("Detector response")
plt.plot(detector_response[:,0] * 1000, detector_response[:,1] * 1000)
plt.xlabel('Incident energy: E (in keV)')
plt.ylabel('Detector energy response:  $\delta(E)$  (in keV)')

plt.tight_layout()

plt.savefig(output_path + '/detector_response.pdf')
plt.savefig(output_path + '/detector_response.png')
```



### 3.5 Converting the voxelised phantom to surface meshes

Download and unzip the phantom

```
[17]: if not os.path.exists("pediatric_phantom_data/Pediatric phantom.zip"):
    urllib.request.urlretrieve("https://drive.uca.fr/f/384a08b5f73244cf9ead/?
    ↪dl=1", "pediatric_phantom_data/Pediatric phantom.zip")
```

```

with zipfile.ZipFile("pediatric_phantom_data/Pediatric phantom.zip", "r") as zip_ref:
    zip_ref.extractall("pediatric_phantom_data")

```

Load the phantom

```

[18]: phantom = sitk.ReadImage("pediatric_phantom_data/Pediatric phantom/
    ↳ Pediatric_model.mhd")

```

Load the labels

```

[19]: df = pd.read_csv("pediatric_phantom_data/labels.dat")

```

Process every structure of the phantom

```

[20]: if not os.path.exists("pediatric_phantom_data/meshes"):
        os.mkdir("pediatric_phantom_data/meshes")

    if not os.path.exists("pediatric_phantom_data/segmentations"):
        os.mkdir("pediatric_phantom_data/segmentations")

    meshes = []

    for threshold, organ in zip(df["Label"], df["Organs"]):

        # Ignore air
        if organ != "Air":

            print("Process", organ)

            seg_fname = "pediatric_phantom_data/segmentations/" + organ + ".mha"
            mesh_fname = "pediatric_phantom_data/meshes/" + organ + ".stl"
            meshes.append(mesh_fname)

            # Only create the mesh if it does not exist
            if not os.path.exists(mesh_fname):

                # Only segment the image it is not done as yet
                if not os.path.exists(seg_fname):

                    # Threshold the phantom
                    binary_image = (phantom == threshold)

                    # Smooth the binary segmentation
                    smoothed_binary_image = sitk.AntiAliasBinary(binary_image)

                    sitk.WriteImage(smoothed_binary_image, seg_fname)
                else:

```



```

        smoothed_binary_image = sitk.ReadImage(seg_fname)

    # Create a VTK image
    vtkimg = sitk2vtk(smoothed_binary_image, centre=True)

    vtk_mesh = extractSurface(vtkimg, 0)

#         print('Before decimation')
#         print(f'There are {mesh.GetNumberOfPoints()} points.')
#         print(f'There are {mesh.GetNumberOfPolys()} polygons.')

#         decimate = vtk.vtkDecimatePro()
#         decimate.SetInputData(mesh)
#         decimate.SplittingOn()
#         decimate.SetTargetReduction(30)
#         decimate.PreserveTopologyOn()
#         decimate.Update()

#         decimated = vtk.vtkPolyData()
#         decimated.ShallowCopy(decimate.GetOutput())

#         print('After decimation')
#         print(f'There are {decimated.GetNumberOfPoints()} points.')
#         print(f'There are {decimated.GetNumberOfPolys()} polygons.')
#         print(
#             f'Reduction: {(mesh.GetNumberOfPolys() - decimated.
↪GetNumberOfPolys()) / mesh.GetNumberOfPolys()}'
#         )

#         print("\n\n")
#         writeSTL(decimated, mesh_fname)
    writeSTL(vtk_mesh, mesh_fname)

```

Process Muscle  
 Process Bone  
 Process Stomach-Interior  
 Process Cartilage  
 Process Brain  
 Process Bladder  
 Process Gallbladder  
 Process Heart  
 Process Kidneys-right  
 Process Kidneys-left  
 Process Small-Intestine  
 Process Large-Intestine  
 Process Liver  
 Process Lung-right  
 Process Lung-left  
 Process Pancreas

Process Spleen  
Process Stomach  
Process Thymus  
Process Eyes-right  
Process Eyes-left  
Process Skull  
Process Trachea

Load the samples. `verbose=2` is used to print the material database for Gate. To disable it, use `verbose=0` or `verbose=1`.

```
[21]: json2gvxr.initSamples(verbose=0)
```

```
Wed Sep 14 09:36:56 2022 ---- file_name:
pediatric_phantom_data/meshes/Muscle.stl      nb_faces:      1756726
nb_vertices:      5270178 bounding_box (in cm):  (-17.9687, -10.8887, -30.9017)
(16.6016, 11.1799, 28.6986)
Wed Sep 14 09:36:56 2022 ---- file_name:
pediatric_phantom_data/meshes/Bone.stl  nb_faces:      541826  nb_vertices:
1625478 bounding_box (in cm):  (-16.7969, -23.6577, -30.9017) (15.2152,
9.88865, 16.3501)
Wed Sep 14 09:36:56 2022 ---- file_name:
pediatric_phantom_data/meshes/Stomach-Interior.stl  nb_faces:      9452
nb_vertices:      28356  bounding_box (in cm):  (-1.34334, -2.38867, -17.0041)
(4.16143, 3.05231, -8.50205)
Wed Sep 14 09:36:57 2022 ---- file_name:
pediatric_phantom_data/meshes/Cartilage.stl  nb_faces:      163322
nb_vertices:      489966 bounding_box (in cm):  (-16.7615, -4.32288, -30.9017)
(15.5041, 8.717, 16.6771)
Wed Sep 14 09:36:57 2022 ---- file_name:
pediatric_phantom_data/meshes/Brain.stl nb_faces:      124028  nb_vertices:
372084 bounding_box (in cm):  (-7.32082, -9.98695, 16.3501) (7.50031,
5.78681, 28.1222)
Wed Sep 14 09:36:57 2022 ---- file_name:
pediatric_phantom_data/meshes/Bladder.stl  nb_faces:      3712
nb_vertices:      11136  bounding_box (in cm):  (-3.78536, 2.11808, -30.9017)
(0.175804, 5.49461, -29.7572)
Wed Sep 14 09:36:57 2022 ---- file_name:
pediatric_phantom_data/meshes/Gallbladder.stl  nb_faces:      4308
nb_vertices:      12924  bounding_box (in cm):  (-5.07422, -1.68659, -17.9851)
(-2.54188, 1.49065, -14.3881)
Wed Sep 14 09:36:57 2022 ---- file_name:
pediatric_phantom_data/meshes/Heart.stl nb_faces:      48172  nb_vertices:
144516 bounding_box (in cm):  (-3.78536, -3.07617, -9.15606) (6.32529,
5.68903, 1.30801)
Wed Sep 14 09:36:57 2022 ---- file_name:
pediatric_phantom_data/meshes/Kidneys-right.stl nb_faces:      17512
nb_vertices:      52536  bounding_box (in cm):  (-7.69363, 1.73117, -18.9661)
(-2.47349, 7.23954, -10.4641)
```

```

Wed Sep 14 09:36:57 2022 ---- file_name:
pediatric_phantom_data/meshes/Kidneys-left.stl  nb_faces:      16388
nb_vertices:  49164  bounding_box (in cm):  (1.37053, 3.46679, -17.9851)
(6.44388, 7.74184, -8.82905)
Wed Sep 14 09:36:58 2022 ---- file_name:
pediatric_phantom_data/meshes/Small-Intestine.stl  nb_faces:      118532
nb_vertices:  355596  bounding_box (in cm):  (-7.48809, -2.95731, -30.9017)
(7.59416, 8.32697, -12.0991)
Wed Sep 14 09:36:58 2022 ---- file_name:
pediatric_phantom_data/meshes/Large-Intestine.stl  nb_faces:      94336
nb_vertices:  283008  bounding_box (in cm):  (-4.66426, -1.67902, -30.4112)
(7.11153, 6.16473, -13.4071)
Wed Sep 14 09:36:58 2022 ---- file_name:
pediatric_phantom_data/meshes/Liver.stl nb_faces:      87800  nb_vertices:
263400  bounding_box (in cm):  (-9.35286, -3.73856, -19.2931) (5.43096,
7.83896, -6.21304)
Wed Sep 14 09:36:58 2022 ---- file_name:
pediatric_phantom_data/meshes/Lung-right.stl  nb_faces:      80364
nb_vertices:  241092  bounding_box (in cm):  (-9.47265, -3.16992, -8.82905)
(0.0788746, 8.15358, 6.54004)
Wed Sep 14 09:36:58 2022 ---- file_name:
pediatric_phantom_data/meshes/Lung-left.stl  nb_faces:      70736
nb_vertices:  212208  bounding_box (in cm):  (0.397666, -2.26504, -9.81006)
(8.28139, 8.52371, 6.21304)
Wed Sep 14 09:36:58 2022 ---- file_name:
pediatric_phantom_data/meshes/Pancreas.stl  nb_faces:      14592
nb_vertices:  43776  bounding_box (in cm):  (-2.8088, -0.240234, -17.0041)
(5.6632, 4.32215, -10.1371)
Wed Sep 14 09:36:58 2022 ---- file_name:
pediatric_phantom_data/meshes/Spleen.stl  nb_faces:      25468
nb_vertices:  76404  bounding_box (in cm):  (1.48829, -0.611202, -14.7151)
(8.10404, 7.94215, -6.86704)
Wed Sep 14 09:36:58 2022 ---- file_name:
pediatric_phantom_data/meshes/Stomach.stl  nb_faces:      28680
nb_vertices:  86040  bounding_box (in cm):  (-3.47804, -2.58413, -17.0041)
(5.05955, 4.0295, -7.84805)
Wed Sep 14 09:36:58 2022 ---- file_name:
pediatric_phantom_data/meshes/Thymus.stl  nb_faces:      3136
nb_vertices:  9408  bounding_box (in cm):  (-0.846352, -1.87282, -1.30801)
(1.53113, 1.18326, 2.28901)
Wed Sep 14 09:36:58 2022 ---- file_name:
pediatric_phantom_data/meshes/Eyes-right.stl  nb_faces:      3956
nb_vertices:  11868  bounding_box (in cm):  (-3.88504, -9.01112, 14.7151)
(-1.28679, -6.41928, 17.6581)
Wed Sep 14 09:36:58 2022 ---- file_name:
pediatric_phantom_data/meshes/Eyes-left.stl  nb_faces:      4116
nb_vertices:  12348  bounding_box (in cm):  (1.66718, -8.8147, 14.7151)
(4.47449, -6.12631, 17.6581)

```

```

Wed Sep 14 09:36:58 2022 ---- file_name:
pediatric_phantom_data/meshes/Skull.stl nb_faces:      327028  nb_vertices:
981084  bounding_box (in cm):   (-7.59598, -10.476, 7.84805)   (7.79064,
6.17931, 29.1032)
Wed Sep 14 09:36:58 2022 ---- file_name:
pediatric_phantom_data/meshes/Trachea.stl      nb_faces:      8588
nb_vertices:      25764  bounding_box (in cm):   (-3.48031, -0.996257, -2.61602)
(3.39865, 5.09486, 10.1371)

```

Get the total number of triangles

```

[22]: number_of_triangles = 0

for sample in json2gvxr.params["Samples"]:
    label = sample["Label"]
    number_of_triangles += gvxr.getNumberOfPrimitives(label)

```

Visualise the phantom

```

[23]: plot = k3d.plot()
plot.background_color = 0xffffffff

for sample in json2gvxr.params["Samples"]:

    label = sample["Label"]

    fname = sample["Path"]

    r, g, b, a = gvxr.getAmbientColour(label)
    R = math.floor(255*r)
    G = math.floor(255*g)
    B = math.floor(255*b)
    A = math.floor(255*a)

    k3d_color = 0;
    k3d_color |= (R & 255) << 16;
    k3d_color |= (G & 255) << 8;
    k3d_color |= (B & 255);

    mesh_from_stl_file = mesh.Mesh.from_file(fname)

    if label == "Muscle":
        opacity = 0.4
    else:
        opacity = 1
    geometry = k3d.mesh(mesh_from_stl_file.vectors.flatten(),
                        range(int(mesh_from_stl_file.vectors.flatten().
↳shape[0] / 3))),

```

```

        color=k3d_color,
        wireframe=False,
        flat_shading=False,
        name=fname,
        opacity=opacity)

    plot += geometry

plot.display()
plot.camera = [321.6678075002728, -461.4855245196105, -34.86613985320561,
               0, 0, -1.635009765625,
               0.08017827340927154, -0.083269170696295, 0.9932963755519574]

```

Output()

```

[24]: fname = output_path + '/pediatric_model.png'
      if not os.path.isfile(fname):

          plot.fetch_screenshot() # Not sure why, but we need to do it twice to get
          ↳ the right screenshot
          plot.fetch_screenshot()

          data = base64.b64decode(plot.screenshot)
          with open(fname, 'wb') as fp:
              fp.write(data)

```

## 4 Run the simulation

Update the 3D visualisation and take a screenshot

```

[25]: gvxr.displayScene()

gvxr.computeXRayImage()
gvxr.useLighting()
gvxr.useWireframe()
gvxr.setZoom(1549.6787109375)

angle = math.pi / 2.0
rotation_matrix_x = np.array([ 1, 0, 0, 0,
                               0, math.cos(angle), -math.sin(angle), 0,
                               0, math.sin(angle),  math.cos(angle), 0,
                               0, 0, 0, 1])

rotation_matrix_z = np.array([ math.cos(angle), -math.sin(angle), 0, 0,
                               math.sin(angle),  math.cos(angle), 0, 0,
                               0, 0, 1, 0,
                               0, 0, 0, 1])

```

```

rotation_matrix_x.shape = [4,4]
rotation_matrix_z.shape = [4,4]

transformation_matrix = np.identity(4)

transformation_matrix = np.matmul(rotation_matrix_x, transformation_matrix)
transformation_matrix = np.matmul(rotation_matrix_z, transformation_matrix)

gvxr.setSceneRotationMatrix(transformation_matrix.flatten())

gvxr.setWindowBackGroundColour(1, 1, 1)

gvxr.displayScene()

```

```
[26]: screenshot = (255 * np.array(gvxr.takeScreenshot())).astype(np.uint8)
```

```

[27]: fname = output_path + 'screenshot.png'
      if not os.path.isfile(fname):

          plt.imshow(screenshot)

```

```

[28]: gvxr.setZoom(1549.6787109375)
      gvxr.setSceneRotationMatrix([-0.19267332553863525, -0.06089369207620621, 0.
      ↪ 9793692827224731, 0.0,
      ↪ 0.9809651970863342, -0.03645244985818863, 0.
      ↪ 19072122871875763, 0.0,
      ↪ 0.02408679760992527, 0.9974713325500488, 0.
      ↪ 06675821542739868, 0.0,
      ↪ 0.0, 0.0, 0.0,
      ↪ 1.0])

      gvxr.setWindowBackGroundColour(0.5, 0.5, 0.5)

      gvxr.useNegative()

      gvxr.displayScene()

```

```
[29]: screenshot = (255 * np.array(gvxr.takeScreenshot())).astype(np.uint8)
```

```

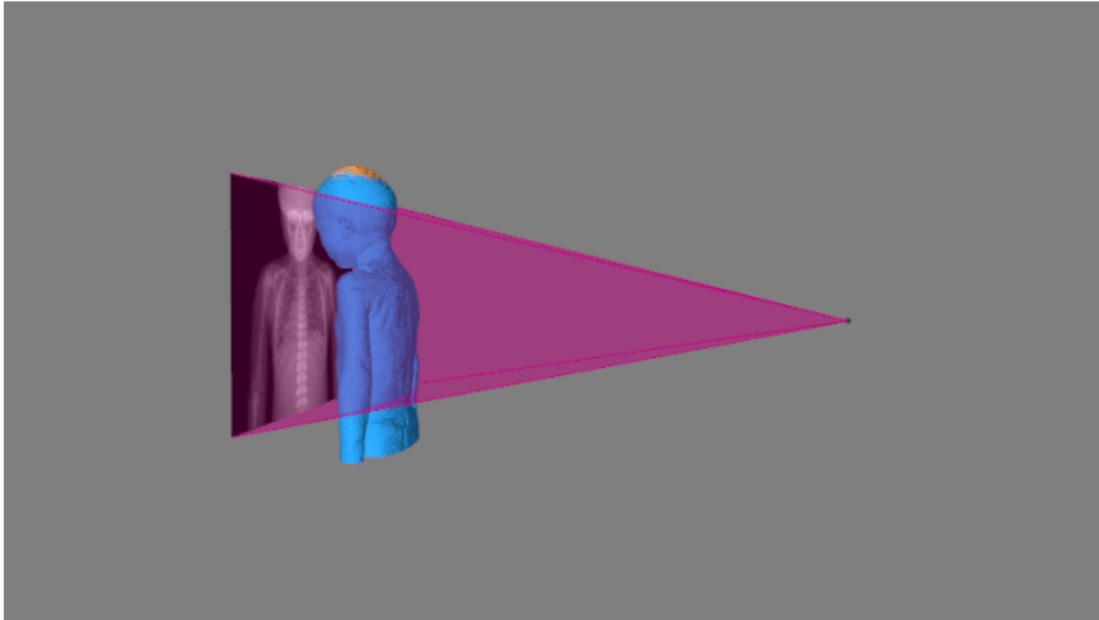
[30]: plt.figure(figsize= (10,10))
      plt.title("Screenshot")
      plt.imshow(screenshot)
      plt.axis('off')

      plt.tight_layout()

```

```
plt.savefig(output_path + '/screenshot-beam-on-paediatrics.pdf')
plt.savefig(output_path + '/screenshot-beam-on-paediatrics.png')
```

## Screenshot



Compute an X-ray image 50 times (to gather performance statistics)

```
[31]: # gvxr.enableArtefactFilteringOnCPU()
gvxr.enableArtefactFilteringOnGPU()
# gvxr.disableArtefactFiltering() # Spere inserts are missing with GPU
↳ integration when a outer surface is used for the matrix

runtimes = []

for i in range(50):
    start_time = datetime.datetime.now()
    gvxr.computeXRayImage()
    end_time = datetime.datetime.now()
    delta_time = end_time - start_time
    runtimes.append(delta_time.total_seconds() * 1000)
```

Save an X-ray image

```
[32]: # Compute the L-buffers on the GPU and integrate on the GPU
gvxr_image = np.array(gvxr.computeXRayImage())
gvxr_image = resize(gvxr_image, gate_image.shape)
```

```
imwrite(output_path + '/gvxr_image-raw-paediatrics.tif', gvxr_image.astype(np.
↪single))
```

Flat-field correction

```
[33]: total_energy_in_keV = 0.0
      for energy, count in zip(energy_set, count_set):
          total_energy_in_keV += energy * count

      total_energy_in_MeV = gvxr.getTotalEnergyWithDetectorResponse()
```

```
[34]: white = np.ones(gvxr_image.shape) * total_energy_in_MeV
      dark = np.zeros(gvxr_image.shape)

      gvxr_image = (gvxr_image - dark) / (white - dark)
```

Save the corresponding image

```
[35]: imwrite(output_path + '/gvxr_image-flat.tif', gvxr_image.astype(np.single))
```

```
[36]: plt.figure(figsize= (20,10))

      plt.suptitle("Image simulated using gVirtualXRay,\nintegration on GPU", y=1.02)

      plt.subplot(121)
      plt.imshow(gvxr_image, cmap="gray")
      plt.colorbar(orientation='horizontal')
      plt.title("Using a linear colour scale")

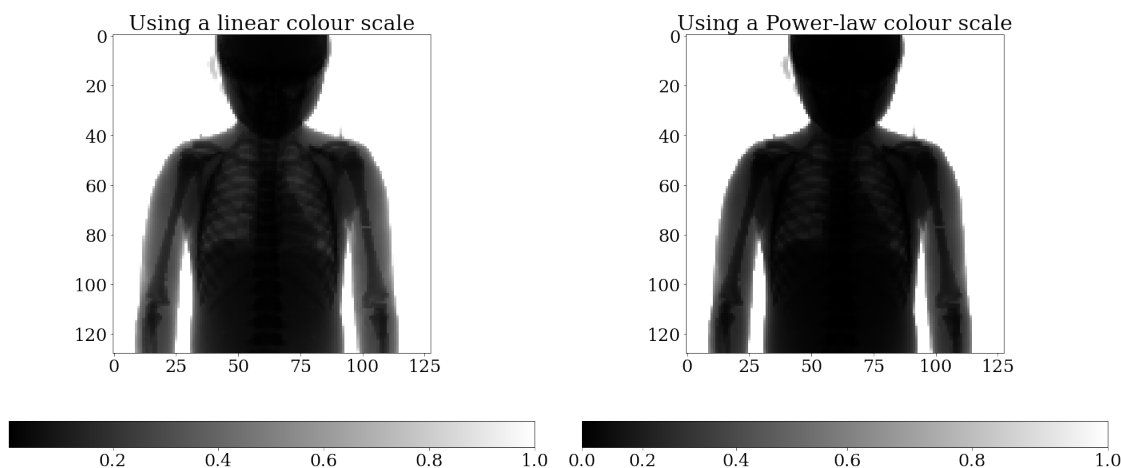
      plt.subplot(122)
      plt.imshow(gvxr_image, norm=PowerNorm(gamma=1./0.75), cmap="gray")
      plt.colorbar(orientation='horizontal')
      plt.title("Using a Power-law colour scale")

      plt.tight_layout()

      plt.savefig(output_path + '/gvxr_image-paediatrics.pdf')
      plt.savefig(output_path + '/gvxr_image-paediatrics.png')
```



Image simulated using gVirtualXRay,  
integration on GPU



## 5 Comparison the analytic simulation with the Monte Carlo simulation

### 5.1 Quantitative validation

Compute image metrics between the two simulated images:

1. [mean absolute percentage error \(MAPE\)](#), also known as mean absolute percentage deviation (MAPD),
2. [zero-mean normalised cross-correlation \(ZNCC\)](#), and
3. [Structural Similarity Index \(SSIM\)](#).

We use these three metrics as one is a dissimilarity measurement (MAPE), two are similarity measurement (ZNCC & SSIM). MAPE and ZNCC can be expressed as a percentage, which eases the interpretation of the numerical values. SSIM is a number between 0 and 1. A good value of MAPE is 0%; of ZNCC 100%, and SSIM 1.

```
[37]: MAPE = mape(gate_image, gvxr_image)
ZNCC = np.mean((gate_image - gate_image.mean()) / gate_image.std() *
    ↪(gvxr_image - gvxr_image.mean()) / gvxr_image.std())
SSIM = ssim(gate_image, gvxr_image, data_range=gate_image.max() - gate_image.
    ↪min())

print("MAPE:", "{0:0.2f}".format(100 * MAPE) + "%")
print("ZNCC:", "{0:0.2f}".format(100 * ZNCC) + "%")
print("SSIM:", "{0:0.2f}".format(SSIM))
```

MAPE: 3.12%

ZNCC: 99.96%

SSIM: 0.99

```
[38]: runtime_avg = round(np.mean(runtimes))
runtime_std = round(np.std(runtimes))
```

Print a row of the table for the paper

```
[39]: print("Paediatric -- polychromatic (85 kV), detector energy response & Gate & "\n
↪+
    "{0:0.2f}".format(100 * MAPE) + "\\%    &    " +
    "{0:0.2f}".format(100 * ZNCC) + "\\%    &    " +
    "{0:0.2f}".format(SSIM) + "    &    $" +
    str(json2gvxr.params["Detector"]["NumberOfPixels"][0]) + " \\pm " +
↪str(json2gvxr.params["Detector"]["NumberOfPixels"][1]) + "$    &    " +
    str(number_of_triangles) + "    &    " +
    "8.68E+08    &    " +
    "$" + str(runtime_avg) + " \\pm " + str(runtime_std) + "$ \\\n")
```

```
Paediatric -- polychromatic (85 kV), detector energy response & Gate & 3.12%
& 99.96%    & 0.99    & $128 \pm 128$    & 3552778    & 8.68E+08
& $51 \pm 4$ \n
```

In both cases, MAPE is very small (about 3%), ZNCC is very high (almost 100%), and SSIM is very high (almost 1). We can conclude that the two images are similar. The main difference lie in the Poisson noise affecting the Monte Carlo simulation.

## 5.2 Qualitative validation

Checkboard comparison

```
[40]: font = {'size' : 12.5
    }
matplotlib.rc('font', **font)
```

```
[41]: source_position = [json2gvxr.params["Source"]["Position"][0] * gvxr.
↪getUnitOfLength(json2gvxr.params["Source"]["Position"][3]) / gvxr.
↪getUnitOfLength("mm"),
    json2gvxr.params["Source"]["Position"][1] * gvxr.
↪getUnitOfLength(json2gvxr.params["Source"]["Position"][3]) / gvxr.
↪getUnitOfLength("mm"),
    json2gvxr.params["Source"]["Position"][2] * gvxr.
↪getUnitOfLength(json2gvxr.params["Source"]["Position"][3]) / gvxr.
↪getUnitOfLength("mm")
    ]

detector_position = [json2gvxr.params["Detector"]["Position"][0] * gvxr.
↪getUnitOfLength(json2gvxr.params["Detector"]["Position"][3]) / gvxr.
↪getUnitOfLength("mm"),
    json2gvxr.params["Detector"]["Position"][1] * gvxr.
↪getUnitOfLength(json2gvxr.params["Detector"]["Position"][3]) / gvxr.
↪getUnitOfLength("mm"),
```

```

        json2gvxr.params["Detector"]["Position"][2] * gvxr.
        ↪getUnitOfLength(json2gvxr.params["Detector"]["Position"][3]) / gvxr.
        ↪getUnitOfLength("mm")
    ]

object_bbox = gvxr.getNodeAndChildrenBoundingBox("root", "mm")
object_position = [(object_bbox[0] + object_bbox[3]) / 2,
                    (object_bbox[1] + object_bbox[4]) / 2,
                    (object_bbox[2] + object_bbox[5]) / 2
                    ]

source_imager_distance = distance.euclidean(source_position, detector_position)
source_object_distance = distance.euclidean(source_position, object_position)

magnification = source_imager_distance / source_object_distance

```

```

[42]: print("SID:", source_imager_distance, "mm")
      print("SOD:", source_object_distance, "mm")
      print("magnification:", magnification)

```

```

SID: 1150.0 mm
SOD: 1062.4489886902375 mm
magnification: 1.082404908133701

```

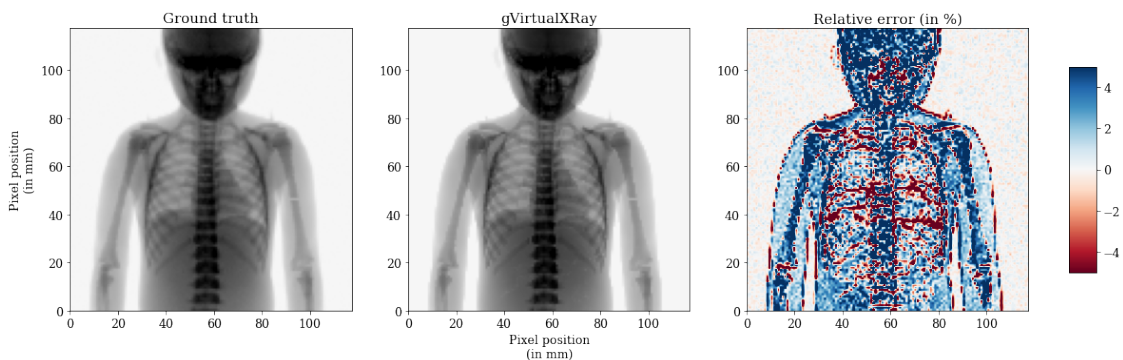
```

[43]: detector_resolution = json2gvxr.params["Detector"]["NumberOfPixels"]
      detector_size = json2gvxr.params["Detector"]["Size"]

      pixel_pitch = np.array([detector_size[0] / detector_size[0] / gvxr.
        ↪getUnitOfLength("mm"),
        detector_size[1] / detector_size[1] / gvxr.getUnitOfLength("mm"),
      ])

      fullCompareImages(gate_image,
        gvxr_image,
        "gVirtualXRay\n with integration on GPU",
        output_path + "/full_comparison-paediatrics", pixel_pitch / ↵
        ↪magnification, log=True)

```



Plot the profiles

```
[44]: font = {'size' : 12.5
        }
matplotlib.rc('font', **font)

[45]: ground_truth_diag = np.diag(gate_image)
gvxr_diag = np.diag(gvvr_image)

x = np.linspace(0, len(ground_truth_diag), len(ground_truth_diag)) *
    ↪ (pixel_pitch[0] / magnification)
plt.figure(figsize=(15, 5))

ax = plt.subplot(111)

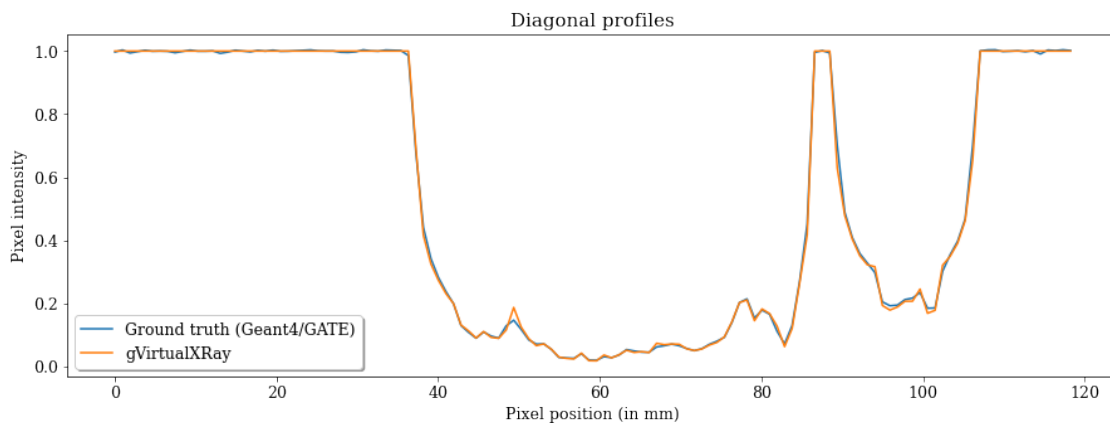
ax.set_title("Diagonal profiles")

ax.plot(x, ground_truth_diag, label="Ground truth (Geant4/GATE)")
ax.plot(x, gvvr_diag, label="gVirtualXRay")

ax.legend(loc='best',
          ncol=1, fancybox=True, shadow=True)

plt.xlabel("Pixel position (in mm)")
plt.ylabel("Pixel intensity")

plt.savefig(output_path + '/profiles-paediatrics.pdf')
plt.savefig(output_path + '/profiles-paediatrics.png')
```



## 6 All done

Destroy the window

```
[46]: gvxr.destroyAllWindows()
```

```
Wed Sep 14 09:37:09 2022 ---- Destroy all the windows
```

```
Wed Sep 14 09:37:09 2022 ---- Destroy window 0(0x5599339750a0)
```