9-gVirtualXRay_vs_DR

March 24, 2022

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

1 gVirtualXray vs DR

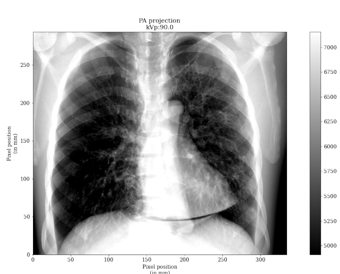
Main contributors: T. Wen, J. Pointon, J. Tugwell-Allsup and F. P. Vidal

Purpose: We aim to reporduce a real digital radiograph taken with a clinically utilised X-ray equipment.

Material and Methods:

1. The CT of a chest phantom has been generated from a real scanner ahead of time.
2. Structures in the reference CT have been segmented and labelled.
3. The resultant surfaces from the segmentations form a virtual lungman model.
4. A digital radiograph was taken with a clinically utilised X-ray equipment.
5. We extract acquisition parameters from the DICOM file and initialise the X-ray simulation parameters.
6. Using a multi-objective optimisation algorithm, the virtual phantom is registered so that its simulated radiograph closely match the real digital radiograph.

Results: The **zero-mean normalised cross-correlation is 93.92%**. The **Structural Similarity Index (SSIM) is 0.91**. The mean absolute percentage error (MAPE) is 3.52%. These results show that the two images are comparable.

Ground truth	Simulation
	

The calculations were performed on the following platform:

[3]:
```
printSystemInfo()
```

```
OS:
        Linux 5.3.18-150300.59.54-default
        x86_64

CPU:
        AMD Ryzen 7 3800XT 8-Core Processor

RAM:
        63 GB
GPU:
        Name: GeForce RTX 2080 Ti
        Drivers: 455.45.01
        Video memory: 11 GB
```

1.1 Import packages

[4]:
```
%matplotlib inline

import os # Locate files
from time import sleep

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

import matplotlib
# old_backend =  matplotlib.get_backend()
# matplotlib.use("Agg")  # Prevent showing stuff

from matplotlib.cm import get_cmap
import matplotlib.pyplot as plt # Plotting
from matplotlib.colors import LogNorm # 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
```

```python
from skimage.util import compare_images # Checkboard comparison between two␣
↪images

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 sklearn.metrics import mean_absolute_error, mean_squared_error
from skimage.metrics import normalized_mutual_information
from sklearn.metrics.cluster import normalized_mutual_info_score
from skimage.filters import gaussian # Implementing the image sharpening filter

import cv2

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

import viewscad # Use OpenSCAD to create STL files

# import pyvista as pv # 3D visualisation
# from pyvista import themes

# import cma # Optimise the parameters of the noise model

import k3d
import random
import base64
from stl import mesh

import urllib, gzip # To download the phantom data, and extract the␣
↪corresponding Z file

import spekpy as sp # Generate a beam spectrum
from scipy import signal # Resampling the beam spectrum

import gvxrPython3 as gvxr # Simulate X-ray images
gvxr.useLogFile()
import json2gvxr # Set gVirtualXRay and the simulation up
from utils import * # Code shared across more than one notebook
import cma # Optimisation

from pymoo.factory import get_problem
from pymoo.optimize import minimize
from pymoo.visualization.scatter import Scatter

import plotly.express as px
import plotly.graph_objects as go
```

```python
from plotly.subplots import make_subplots

import sys
```

```
SimpleGVXR 1.0.1 (2022-03-10T15:28:42) [Compiler: GNU g++] on Linux
gVirtualXRay core library (gvxr) 1.1.5 (2022-03-10T15:28:36) [Compiler: GNU g++]
on Linux
```

[5]:
```python
def standardisation(img):
    return (img - img.mean()) / img.std()
```

2 Preparation of the ground truth image

2.1 Read the real X-ray radiograph from a DICOM file

[6]:
```python
reader = sitk.ImageFileReader()
reader.SetImageIO("GDCMImageIO")
reader.SetFileName("lungman_data/CD3/DICOM/ST000000/SE000000/DX000000")
reader.LoadPrivateTagsOn()
reader.ReadImageInformation()
volume = reader.Execute()
raw_reference_before_cropping = sitk.GetArrayFromImage(volume)[0]
raw_reference_before_cropping.shape = raw_reference_before_cropping.shape

y_min_id = 200
y_max_id = raw_reference_before_cropping.shape[0] - 170
x_min_id = 50
x_max_id = raw_reference_before_cropping.shape[1] - 100

raw_reference = raw_reference_before_cropping[y_min_id:y_max_id, x_min_id:
 ↪x_max_id]

print("The shape was", raw_reference_before_cropping.shape, "| now it is",␣
 ↪raw_reference.shape)
```

```
The shape was (1881, 1871) | now it is (1511, 1721)
```

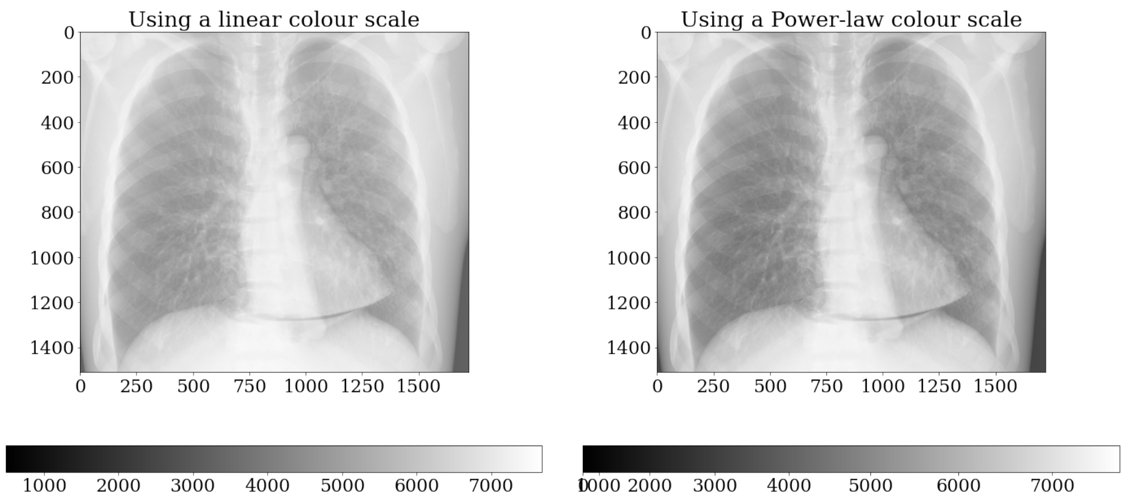
[7]:
```python
imwrite('gVirtualXRay_output_data/real_projection-lungman.tif', raw_reference.
 ↪astype(np.single))
```

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

[8]:
```python
displayLinearPowerScales(raw_reference,
                         "Reference image from the Lungman",
                         "plots/reference-lungman-proj",
```

```
                   log=False,
                   vmin=-93, vmax=89)
```

Reference image from the Lungman



Apply a zero-mean, unit-variance normalisation

[9]:
```
ground_truth = raw_reference
normalised_ground_truth = standardisation(ground_truth)
imwrite('gVirtualXRay_output_data/lungman-normalised_ground_truth.tif', normalised_ground_truth.astype(np.single))
```

[10]:
```
# meta_data_keys = volume.GetMetaDataKeys()

# print("DICOM fields:")
# for key in meta_data_keys:
#     print(key, volume.GetMetaData(key))
```

[11]:
```
# def cleanTags(raw_string):
#     regular_expression = re.compile('<.*?>')
#     clean_text = re.sub(regular_expression, '', raw_string)
#     return clean_text
```

[12]:
```
# field = volume.GetMetaData("0033|1022")

# for item in field.split("\n"):
#     if "KV" in item:
#         kv = float(cleanTags(item))
```

```python
# print(kv)
```

2.2 Extract the image size and pixel spacing from the DICOM file

It will be useful to set the X-ray detector parameters for the simulation, and to display the images in millimetres.

[13]:
```python
spacing = volume.GetSpacing()[0:2]
size = volume.GetSize()[0:2]
```

2.3 Extract the kVp from the DICOM file

It will be useful to generate a realistic beam spectrum.

[14]:
```python
kVp = float(volume.GetMetaData("0018|0060"))
print("Peak kilo voltage output of the x-ray generator used: ", kVp)
```

```
Peak kilo voltage output of the x-ray generator used:  90.0
```

3 Initialise gVirtualXRay

3.1 Set the experimental parameters (e.g. source and detector positions, etc.)

We use known parameters as much as possible, for example we know the size and composition of the sample. Some parameters are extracted from the DICOM file, such as detector size, pixel resolution, and voltage of the X-ray tube.

[15]:
```python
distance_source_to_detector = float(volume.GetMetaData("0018|1110"))
distance_source_to_patient = float(volume.GetMetaData("0018|1111"))

print("Distance Source to Detector: ", distance_source_to_detector, "mm")
print("Distance Source to Patient: ", distance_source_to_patient, "mm")

window_size =  [800, 450]
# source_position = [0.0, 0.0, source_detector_distance_in_cm -␣
 ↪block_thickness_in_cm / 2, "mm"]
# detector_position = [0.0, 0.0, - block_thickness_in_cm / 2, "cm"]
detector_up = [0, 1, 0]
```

```
Distance Source to Detector:  1800.0 mm
Distance Source to Patient:  1751.0 mm
```

3.2 Initialise the simulation engine

[16]:
```
# Create an OpenGL context
print("Create an OpenGL context:",
    str(window_size[0]) + "x" + str(window_size[1])
)

gvxr.createWindow(-1, True, "EGL")

gvxr.setWindowSize(
    window_size[0],
    window_size[1]
)
```

```
Create an OpenGL context: 800x450
gvxrStatus:     Create window
OpenGL renderer:   GeForce RTX 2080 Ti/PCIe/SSE2
OpenGL version:    4.5.0 NVIDIA 455.45.01
OpenGL vender:     NVIDIA Corporation
0 0 500 500
0 0 800 450
```

3.3 Load the scanned object

[17]:
```
json2gvxr.initSamples("notebook-9.json", verbose=0)
```

[18]:
```
number_of_triangles = 0

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

[19]:
```
skin_bbox = gvxr.getNodeOnlyBoundingBox("Skin", "mm")
print(skin_bbox)
```

```
(-159.375, -117.5, -148.40000915527344, 159.375, 107.5, 148.40000915527344)
```

3.4 Set the source position

[20]:
```
# Set up the beam
print("Set up the beam")
print("\tSource position:", (skin_bbox[0] + skin_bbox[3]) / 2,␣
 ↪distance_source_to_detector + skin_bbox[1] + distance_source_to_patient -␣
 ↪distance_source_to_detector, (skin_bbox[2] + skin_bbox[5]) / 2, "mm")
```

```
gvxr.setSourcePosition((skin_bbox[0] + skin_bbox[3]) / 2,␣
↪distance_source_to_detector + skin_bbox[1] + distance_source_to_patient -␣
↪distance_source_to_detector, (skin_bbox[2] + skin_bbox[5]) / 2, "mm")

gvxr.usePointSource()
```

```
Set up the beam
        Source position: 0.0 1633.5 0.0 mm
```

3.5 Get the spectrum from the DICOM file

[21]:
```
spectrum = {};
# filter_material = "Al"      # See email Mon 05/07/2021 15:29
# filter_thickness_in_mm = 3  # See email Mon 05/07/2021 15:29

s = sp.Spek(kvp=kVp)
# s.filter(filter_material, filter_thickness_in_mm) # Filter by 3 mm of Al
unit = "keV"
k, f = s.get_spectrum(edges=True) # Get the spectrum

min_energy = sys.float_info.max
max_energy = -sys.float_info.max

for energy, count in zip(k, f):
    count = round(count)

    if count > 0:

        max_energy = max(max_energy, energy)
        min_energy = min(min_energy, energy)

        if energy in spectrum.keys():
            spectrum[energy] += count
        else:
            spectrum[energy] = count
```

Reformat the data

[22]:
```
# get the integral nb of photons
nbphotons=0.
energy1 = -1.
energy2 = -1.

for energy in spectrum.keys():
```

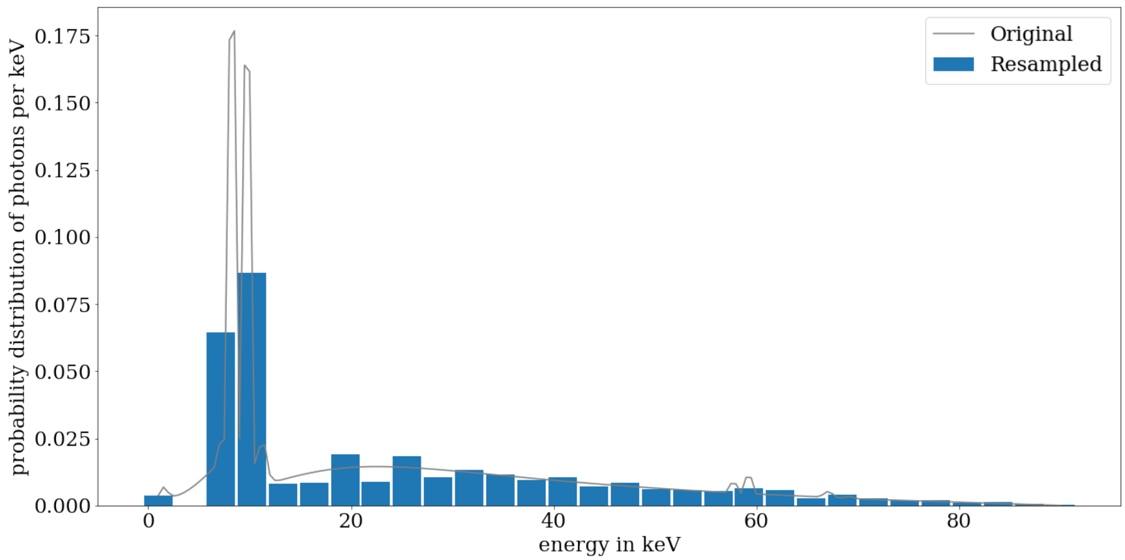
```
    if energy1<0:
        energy1 = float(energy)
    elif energy2<0:
        energy2 = float(energy)
    nbphotons += float(spectrum[energy])
sampling = (energy2-energy1)

# get spectrum
data = []
for energy in spectrum.keys():
    source = [float(energy),float(spectrum[energy])/(nbphotons*sampling)]
    data.append(source)

data_array = np.array(data)

energies, counts = data_array.T
```

Resample the data to reduce the number of bins

[23]:
```
temp_count_set = signal.decimate(counts, 6)
number_of_energy_bins = temp_count_set.shape[0]
temp_energy_set = np.linspace(energies.min(), energies.max(),␣
↪number_of_energy_bins, endpoint=True)

test = temp_count_set > 0
count_set = temp_count_set[test]
energy_set = temp_energy_set[test]
```

Plot the beam spectrum from spekpy and the resampled vervion

[24]:
```
plt.figure(figsize= (20,10))
plt.bar(energy_set, count_set, width=2.8, label="Resampled")
plt.plot(energies, counts, label="Original", color="gray")
plt.xlabel('energy in keV')
plt.ylabel('probability distribution of photons per keV')
plt.legend()
plt.savefig("plots/lungman-projection-spectrum.pdf")
```



3.6 Load the beam spectrum in the simulator

[25]:
```
gvxr.resetBeamSpectrum() # To be on the safe side when debugging
for energy, count in zip(energy_set, count_set):
    gvxr.addEnergyBinToSpectrum(energy, unit, count);
```

3.7 Set the X-ray detector

[26]:
```
# Set up the detector
print("Set up the detector");
print("\tDetector position:", (skin_bbox[0] + skin_bbox[3]) / 2, skin_bbox[1] +␣
 ↪distance_source_to_patient - distance_source_to_detector, (skin_bbox[2] +␣
 ↪skin_bbox[5]) / 2, "mm")
gvxr.setDetectorPosition((skin_bbox[0] + skin_bbox[3]) / 2, skin_bbox[1] +␣
 ↪distance_source_to_patient - distance_source_to_detector, (skin_bbox[2] +␣
 ↪skin_bbox[5]) / 2, "mm");

print("\tDetector up vector:", [0, 0, 1])
gvxr.setDetectorUpVector(0, 0, 1);
```

```
Set up the detector
        Detector position: 0.0 -166.5 0.0 mm
        Detector up vector: [0, 0, 1]
```

[27]:
```python
print("\tDetector number of pixels:", size)
gvxr.setDetectorNumberOfPixels(
    size[0],
    size[1]
);

print("\tPixel spacing:", spacing)
gvxr.setDetectorPixelSize(
    spacing[0],
    spacing[1],
    "mm"
);
```

```
        Detector number of pixels: (1871, 1881)
        Pixel spacing: (0.194556, 0.194556)
```

Load the detector response in energy

[28]:
```python
gvxr.clearDetectorEnergyResponse() # To be on the safe side
gvxr.loadDetectorEnergyResponse("Gate_data/responseDetector.txt",
                                "MeV")
```

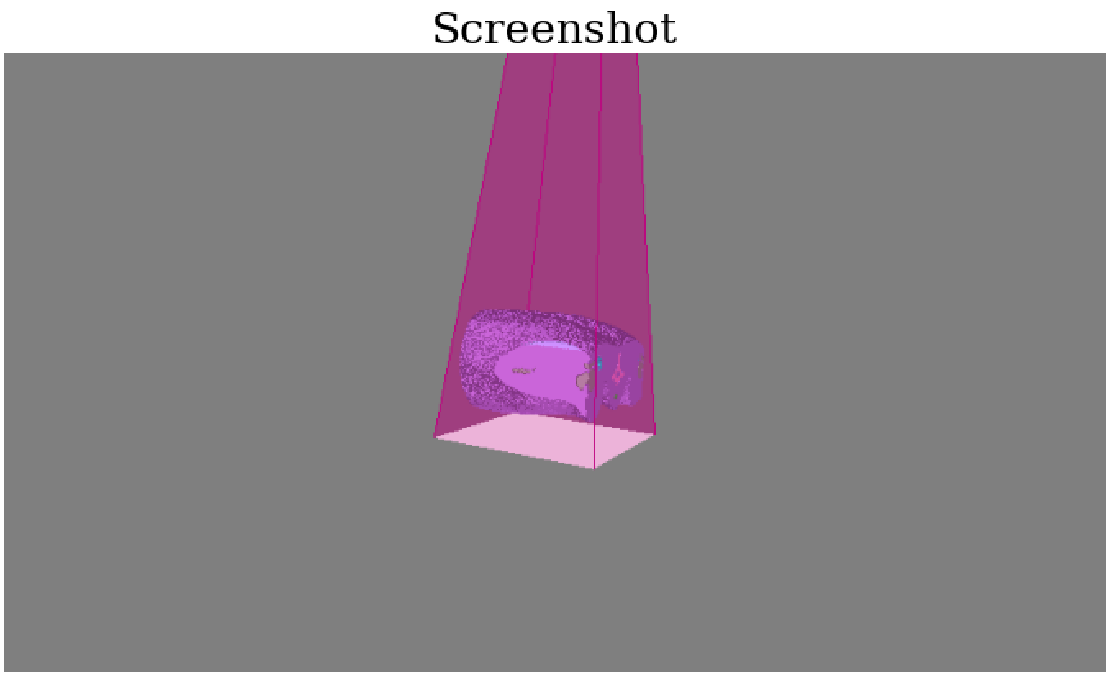
3.8 Take a screenshot of the 3D environment

[29]:
```python
gvxr.displayScene()

gvxr.useNegative()
gvxr.useLighing()
gvxr.useWireframe()
gvxr.setSceneRotationMatrix([0.43813619017601013, 0.09238918125629425, -0.
↪8941444158554077, 0.0,
                             0.06627026945352554, 0.9886708855628967, 0.
↪13463231921195984, 0.0,
                             0.8964602947235107, -0.11824299395084381, 0.
↪4270564019680023, 0.0,
                             0.0, 0.0, 0.0, 1.0])
gvxr.setZoom(1639.6787109375)

gvxr.displayScene()
```

[30]:
```python
screenshot = gvxr.takeScreenshot()
```

[31]:
```python
plt.figure(figsize= (10,10))
plt.title("Screenshot")
plt.imshow(screenshot)
plt.axis('off')
```

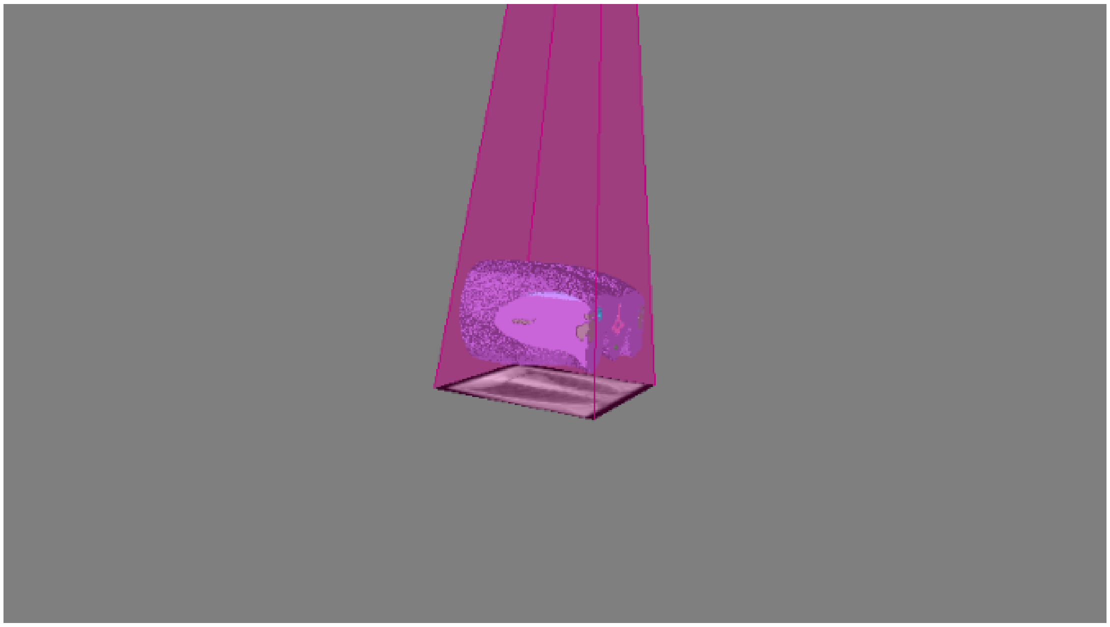
```
plt.tight_layout()

plt.savefig('plots/lungman-projection-screenshot-beam-off.pdf')
plt.savefig('plots/lungman-projection-screenshot-beam-off.png')
```



[32]:
```
gvxr.computeXRayImage()
gvxr.displayScene()
```

[33]:
```
screenshot = gvxr.takeScreenshot()

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

plt.tight_layout()

plt.savefig('plots/PMMA_screenshot-beam-on.pdf')
plt.savefig('plots/PMMA_screenshot-beam-on.png')
```

Screenshot



4 Visualise the virtual patient

[34]:
```
plot = k3d.plot()
plot.background_color = 0xffffff

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 == "Skin":
        opacity = 0.2
    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 = [458.4242199518181, -394.5268107574361, 59.58430140683608, 93.
 ↪26420522817403, -15.742963565665017, -45.88423611599179, -0.08892603121323975,␣
 ↪0.11140808541436767, 0.9897880578573034]
```

```
Output()
```

4.1 Simulation with the default values

[35]:
```
# Backup the transformation matrix
global_matrix_backup = gvxr.getSceneTransformationMatrix()
```

[36]:
```
def getXRayImage():
    global total_energy_in_MeV

    # Compute the X-ray image
    xray_image = np.array(gvxr.computeXRayImage())

    # Flat-field
#     xray_image /= total_energy_in_MeV

    # Negative
    # x_ray_image = 1.0 - x_ray_image
    return np.flip(xray_image) #np.ones(xray_image.shape).astype(np.single) -␣
 ↪xray_image
```

[37]:
```
# gvxr.enableArtefactFilteringOnCPU()
gvxr.enableArtefactFilteringOnGPU()
# gvxr.disableArtefactFiltering() # Spere inserts are missing with GPU␣
 ↪integration when a outer surface is used for the matrix
```

[38]:
```
xray_image = getXRayImage()
```

[39]:
```
# total_energy_in_keV = 0.0
# for energy, count in zip(energy_set, count_set):
#     effective_energy = find_nearest(detector_response[:,0], energy / 1000,␣
↪detector_response[:,1])

#     total_energy_in_keV += effective_energy * count

total_energy_in_MeV = gvxr.getTotalEnergyWithDetectorResponse()
```

[40]:
```
gvxr.displayScene()
gvxr.useNegative()

gvxr.setZoom(1339.6787109375)
gvxr.setSceneRotationMatrix([0.8227577805519104, 0.1368587613105774, -0.
↪5516625642776489, 0.0, -0.5680444240570068, 0.23148967325687408, -0.
↪7897683382034302, 0.0, 0.01961756870150566, 0.9631487131118774, 0.
↪26820749044418335, 0.0, 0.0, 0.0, 0.0, 1.0])

gvxr.setWindowBackGroundColour(0.5, 0.5, 0.5)

gvxr.displayScene()
```

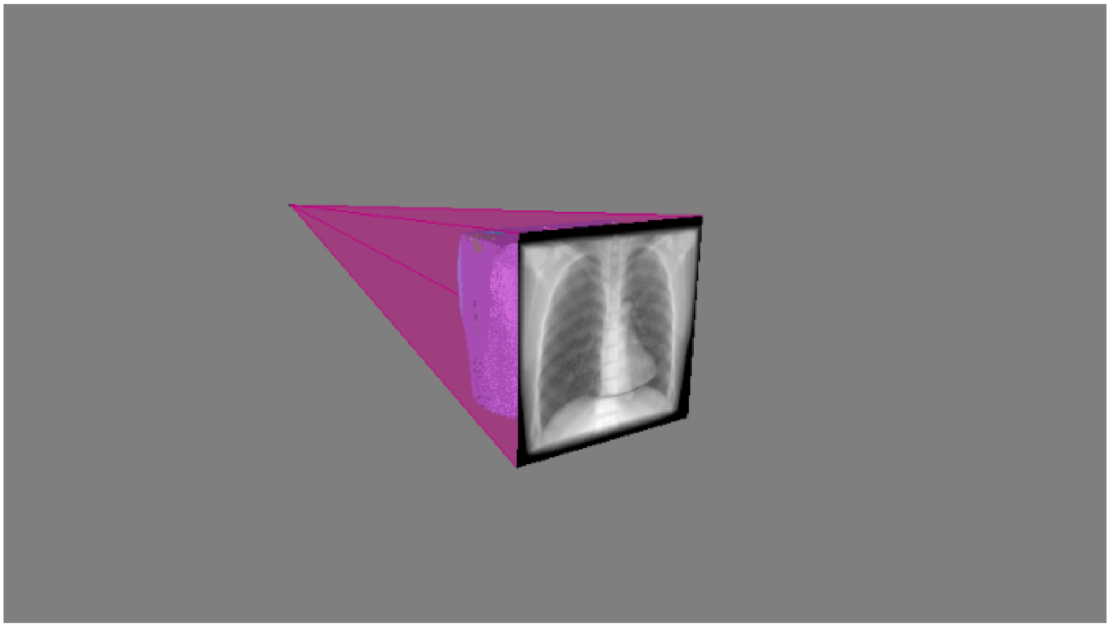
[41]:
```
# gvxr.renderLoop()
```

[42]:
```
# print(gvxr.getZoom())
# print(gvxr.getSceneRotationMatrix())
```

[43]:
```
screenshot = (255 * np.array(gvxr.takeScreenshot())).astype(np.uint8)
```

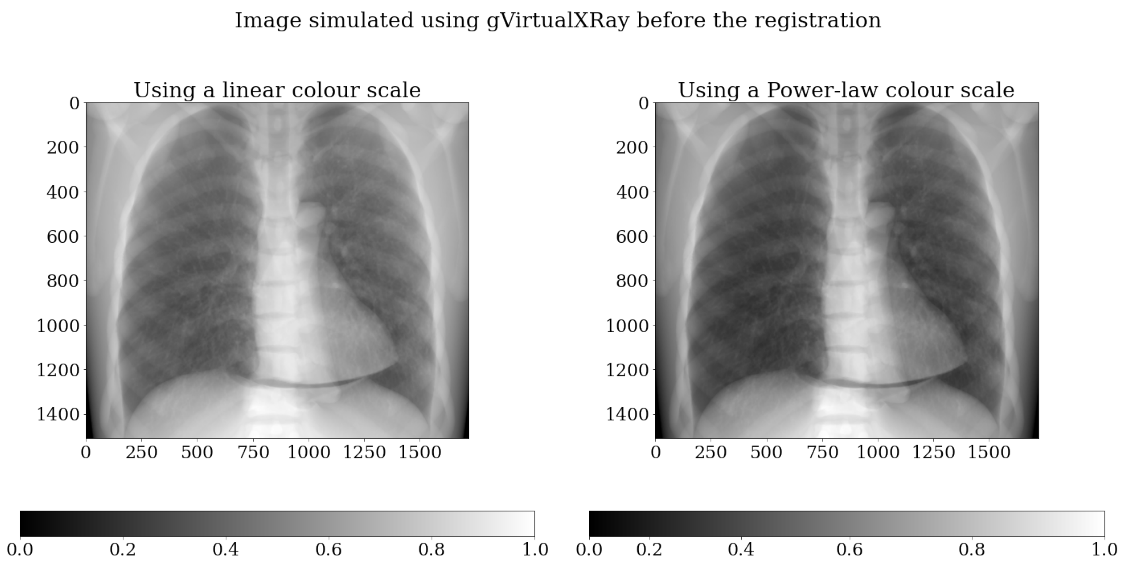
[44]:
```
plt.figure(figsize= (10,10))
plt.title("Screenshot")
plt.imshow(screenshot)
plt.axis('off')

plt.tight_layout()

plt.savefig('plots/default-screenshot-beam-on-lungman.pdf')
plt.savefig('plots/default-screenshot-beam-on-lungman.png')
```

Screenshot



[45]:
```
def logImage(xray_image: np.array, min_val: float, max_val: float) -> np.array:

    log_epsilon = 1.0e-9

    shift_filter = -math.log(min_val + log_epsilon)

    if min_val != max_val:
        scale_filter = 1.0 / (math.log(max_val + log_epsilon) - math.log(min_val + log_epsilon))
    else:
        scale_filter = 1.0

    corrected_image = np.log(xray_image + log_epsilon)

    return (corrected_image + shift_filter) * scale_filter
```

[46]:
```
xray_image_cropped = xray_image[y_min_id:y_max_id, x_min_id:x_max_id]
displayLinearPowerScales(1 - logImage(xray_image_cropped, xray_image_cropped.min(), xray_image_cropped.max()),
                         "Image simulated using gVirtualXRay before the registration",
                         "plots/gVirtualXRay-before_registration-lungman",
                         log=False)
```



4.2 Registration

[47]:

```
roi_ground_truth_min = ground_truth.min()
roi_ground_truth_max = ground_truth.max()
standardised_roi_ground_truth = standardisation(ground_truth)

imwrite('gVirtualXRay_output_data/standardised_roi_ground_truth-lungman.tif',␣
 ↪standardised_roi_ground_truth.astype(np.single))
```

[48]:

```
source_position_bak = gvxr.getSourcePosition("cm")
detector_position_bak = gvxr.getDetectorPosition("cm")

x_init = [
    # Orientation of the sample
    0.0, 0.0,

    # Position of the source
    source_position_bak[0],
    source_position_bak[1],
    source_position_bak[2],

    # Position of the detector
    detector_position_bak[0],
    detector_position_bak[1],
    detector_position_bak[2]#,

    # Orientation of the detector
```

```
    #     det_rotation_angle1 = x[8]
    #     det_rotation_angle2 = x[9]

#         1.0 / 3.0, # c1
#         1.0, # gain1
#         0.0, # bias1

#         1.0 / 3.0, # c2
#         1.0, # gain2
#         0.0#, # bias2

#         1.0 / 3.0, # c3
#         1.0, # gain3
#         0.0, # bias3
#         2.0 # gamma
]
```

[49]:

```
pos_offset = 20
angle_offset = 5

xl = [
            -angle_offset, -angle_offset,
            source_position_bak[0] - pos_offset, source_position_bak[1] -␣
↪pos_offset, source_position_bak[2] - pos_offset,
            detector_position_bak[0] - pos_offset, detector_position_bak[1] -␣
↪pos_offset, detector_position_bak[2] - pos_offset#,
#             -90, -90,
#             -10.0,
#             -10.0,
#             -10.0,

#             -10.0,
#             0.0,
#             0.0#,

#             -10.0,
#             -10.0,
#             -10.0,
#             0.0
        ]

xu = [
            angle_offset, angle_offset,
            source_position_bak[0] + pos_offset, source_position_bak[1] +␣
↪pos_offset, source_position_bak[2] + pos_offset,
            detector_position_bak[0] + pos_offset, detector_position_bak[1] +␣
↪pos_offset, detector_position_bak[2] + pos_offset #,
```

```
#              90, 90,
#              10.0,
#              10.0,
#              10.0,

#              10.0,
#              10.0,
#              10.0#,

#              10.0,
#              10.0,
#              10.0,
#              100.0
        ]
```

[50]:

```
def setTransformations(x):
    # Orientation of the sample
    sample_rotation_angle1 = x[0]
    sample_rotation_angle2 = x[1]

    gvxr.rotateScene(sample_rotation_angle1, 1, 0, 0)
    gvxr.rotateScene(sample_rotation_angle2, 0, 1, 0)

    # Position of the source
    source_position_x = x[2]
    source_position_y = x[3]
    source_position_z = x[4]

    gvxr.setSourcePosition(
        source_position_x,
        source_position_y,
        source_position_z,
        "cm"
    )

    # Position of the detector
    det_position_x = x[5]
    det_position_y = x[6]
    det_position_z = x[7]

    gvxr.setDetectorPosition(
        det_position_x,
        det_position_y,
        det_position_z,
        "cm"
    )
```

```
    # Orientation of the detector
#     det_rotation_angle1 = x[8]
#     det_rotation_angle2 = x[9]
```

[51]:
```
def resetToDefaultParameters():
    gvxr.setSourcePosition(source_position_bak[0], source_position_bak[1],␣
↪source_position_bak[2], "cm")
    gvxr.setDetectorPosition(detector_position_bak[0], detector_position_bak[1],␣
↪detector_position_bak[2], "cm")


    # Restore the transformation matrix
    gvxr.setSceneTransformationMatrix(global_matrix_backup)
```

[52]:
```
def updateXRayImage(x, restore_transformation=True):


    # Backup the transformation matrix
    if restore_transformation:
        matrix_backup = gvxr.getSceneTransformationMatrix()

    # Set the transformations
    setTransformations(x)

    # Compute the X-ray image
    xray_image = getXRayImage()

#     gvxr.displayScene()
#     screenshot = gvxr.takeScreenshot()

    # Restore the transformation matrix
    if restore_transformation:
        gvxr.setSceneTransformationMatrix(matrix_backup)

    return xray_image #, screenshot
```

[53]:
```
def applyLogScaleAndNegative(image: np.array) -> np.array:
    temp = logImage(image, image.min(), image.max())
    return 1.0 - temp
```

[54]:
```
timeout_in_sec = 30 * 60 # 20 minutes
```

4.3 NSGA-III

[55]:
```
from sklearn.metrics.cluster import normalized_mutual_info_score
from pymoo.algorithms.moo.nsga3 import NSGA3
from pymoo.factory import get_reference_directions

standardised_roi_ground_truth = standardised_roi_ground_truth.astype(np.single)
gX = cv2.Sobel(standardised_roi_ground_truth, ddepth=cv2.CV_32F, dx=1, dy=0)
gY = cv2.Sobel(standardised_roi_ground_truth, ddepth=cv2.CV_32F, dx=0, dy=1)
gX = cv2.convertScaleAbs(gX)
gY = cv2.convertScaleAbs(gY)
ref_grad_magn = cv2.addWeighted(gX, 0.5, gY, 0.5, 0)
ref_grad_magn = standardisation(ref_grad_magn)

def objectiveFunctions(x):

    global objective_function_string

    global ground_truth, standardised_roi_ground_truth
    global best_fitness, best_fitness_id, fitness_function_call_id,␣
↪evolution_fitness, evolution_parameters

    objectives = []

    for ind in x:
        xray_image = updateXRayImage(ind)
        corrected_xray_image = applyLogScaleAndNegative(xray_image[y_min_id:
↪y_max_id, x_min_id:x_max_id])
        corrected_xray_image = corrected_xray_image.astype(np.single)

        if corrected_xray_image.min() != corrected_xray_image.max():
            standardised_corrected_xray_image =␣
↪standardisation(corrected_xray_image)
        else:
            standardised_corrected_xray_image = corrected_xray_image

        # gX = cv2.Sobel(corrected_xray_image, ddepth=cv2.CV_32F, dx=1, dy=0)
        # gY = cv2.Sobel(corrected_xray_image, ddepth=cv2.CV_32F, dx=0, dy=1)
        # gX = cv2.convertScaleAbs(gX)
        # gY = cv2.convertScaleAbs(gY)
        # test_grad_magn = cv2.addWeighted(gX, 0.5, gY, 0.5, 0)


        ref_image = standardised_roi_ground_truth
        test_image = standardised_corrected_xray_image

        # ref_image = ref_grad_magn
```

```python
        # if test_grad_magn.min() != test_grad_magn.max():
        #     test_grad_magn = standardisation(test_grad_magn)
        # else:
        #     test_image = test_grad_magn

        row = []

        zncc = np.mean(standardised_roi_ground_truth *␣
↪standardised_corrected_xray_image)
        dzncc = (1.0 - zncc) / 2.0
        row.append(dzncc)

        mae = np.mean(np.abs(standardised_roi_ground_truth -␣
↪standardised_corrected_xray_image))
        row.append(mae)

        rmse = math.sqrt(np.mean(np.square(standardised_roi_ground_truth -␣
↪standardised_corrected_xray_image)))
        row.append(rmse)

        ssim_value = ssim(standardised_roi_ground_truth,␣
↪standardised_corrected_xray_image, data_range=standardised_roi_ground_truth.
↪max() - standardised_roi_ground_truth.min())
        dssim = (1.0 - ssim_value) / 2.0
        row.append(dssim)

        # Avoid div by 0
        offset1 = min(standardised_roi_ground_truth.min(),␣
↪standardised_corrected_xray_image.min())
        offset2 = 0.01 * (standardised_roi_ground_truth.max() -␣
↪standardised_roi_ground_truth.min())
        offset = offset2 - offset1

        mape_value = mape(standardised_roi_ground_truth + offset,␣
↪standardised_corrected_xray_image + offset)
        row.append(mape_value)

        mi = normalized_mutual_information(standardised_roi_ground_truth,␣
↪standardised_corrected_xray_image)
        dmi = (1.0 - mi) / 2.0
        row.append(dmi)

        objectives.append(row)

    return objectives
```

[56]:
```
from pymoo.core.problem import Problem

class MyMultiObjectiveProblem(Problem):

    def __init__(self):
        super().__init__(n_var=len(x_init),
                         n_obj=6,
                         n_constr=0,
                         xl=xl,
                         xu=xu)

    def _evaluate(self, x, out, *args, **kwargs):
        out["F"] = objectiveFunctions(x)
```

[57]:
```
import warnings
warnings.filterwarnings("ignore", category=UserWarning)

from pymoo.optimize import minimize
from pymoo.factory import get_termination
from pymoo.util.termination import collection

resetToDefaultParameters()

problem = MyMultiObjectiveProblem()

pop_size = 210

x_tol_termination = get_termination("x_tol", 1e-5)
f_tol_termination = get_termination("f_tol", 1e-5)
time_termination = get_termination("time", "00:20:00")

termination = collection.TerminationCollection(x_tol_termination,␣
↪f_tol_termination, time_termination)

if os.path.exists("gVirtualXRay_output_data/lungman-res-nsga3-X.dat") and os.
↪path.exists("gVirtualXRay_output_data/lungman-res-nsga3-F.dat"):

    res_nsga3_X = np.loadtxt("gVirtualXRay_output_data/lungman-res-nsga3-X.dat")
    res_nsga3_F = np.loadtxt("gVirtualXRay_output_data/lungman-res-nsga3-F.dat")

else:

    n_objs = 6
    n_partitions = 6

    ref_dirs = get_reference_directions("das-dennis", n_objs,␣
↪n_partitions=n_partitions)
```

```python
    problem = MyMultiObjectiveProblem()

    pop_size = 462 #2 * ref_dirs.shape[0]

    algorithm = NSGA3(
        pop_size=pop_size,
#         n_offsprings=int(pop_size*0.05),
        eliminate_duplicates=True,
        ref_dirs=ref_dirs
    )


    res_nsga3 = minimize(problem,
                   algorithm,
                   termination,
                   seed=1,
                   save_history=True,
                   verbose=True)

    res_nsga3_X = res_nsga3.X
    res_nsga3_F = res_nsga3.F

    np.savetxt("gVirtualXRay_output_data/lungman-res-nsga3-X.dat", res_nsga3_X)
    np.savetxt("gVirtualXRay_output_data/lungman-res-nsga3-F.dat", res_nsga3_F)
```

```
=======================================================
n_gen |  n_eval |  n_nds  |     eps      |  indicator
=======================================================
    1 |     462 |       1 |            - |            -
    2 |     924 |       1 |  0.00000E+00 |            f
    3 |    1386 |       1 |  0.00000E+00 |            f
    4 |    1848 |       3 |  1.000000000 |        ideal
    5 |    2310 |       1 |  0.191668663 |        ideal
```

[58]:
```python
if len(res_nsga3_F.shape) == 1:
    res_nsga3_F.shape = [1,6]

if len(res_nsga3_X.shape) == 1:
    res_nsga3_X.shape = [1,8]

best_dzncc_id = np.argmin(res_nsga3_F[:,0])
best_mae_id = np.argmin(res_nsga3_F[:,1])
best_rmse_id = np.argmin(res_nsga3_F[:,2])
best_dssim_id = np.argmin(res_nsga3_F[:,3])
best_mape_id = np.argmin(res_nsga3_F[:,4])
best_dmi_id = np.argmin(res_nsga3_F[:,5])
```

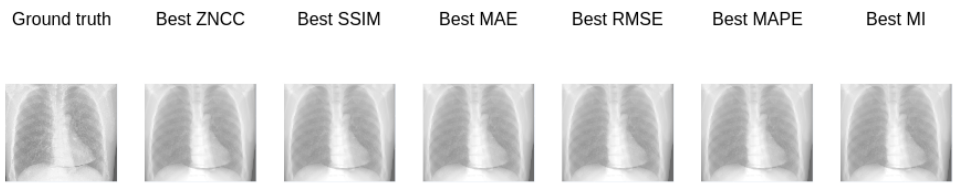
```
best_dzncc_X = res_nsga3_X[best_dzncc_id]
best_mae_X = res_nsga3_X[best_mae_id]
best_rmse_X = res_nsga3_X[best_rmse_id]
best_dssim_X = res_nsga3_X[best_dssim_id]
best_mape_X = res_nsga3_X[best_mape_id]
best_dmi_X = res_nsga3_X[best_dmi_id]

print("Lowest DZNCC:", res_nsga3_F[:,0].min(), best_dzncc_id,␣
 ↪res_nsga3_X[best_dzncc_id])
print("Lowest DSSIM:", res_nsga3_F[:,3].min(), best_dssim_id,␣
 ↪res_nsga3_X[best_dssim_id])
print("Lowest MAE:",   res_nsga3_F[:,1].min(), best_mae_id,  ␣
 ↪res_nsga3_X[best_mae_id])
print("Lowest RMSE:",  res_nsga3_F[:,2].min(), best_rmse_id, ␣
 ↪res_nsga3_X[best_rmse_id])
print("Lowest MAPE:",  res_nsga3_F[:,4].min(), best_mape_id, ␣
 ↪res_nsga3_X[best_mape_id])
print("Lowest DMI:",  res_nsga3_F[:,5].min(), best_dmi_id, ␣
 ↪res_nsga3_X[best_dmi_id])
```

```
Lowest DZNCC: 0.03275948762893677 0 [ -0.46911404  -0.41254088  -9.09671268
181.15806747 -18.39449859
   1.78487687 -15.99704297   1.6970333 ]
Lowest DSSIM: 0.1295748646164261 0 [ -0.46911404  -0.41254088  -9.09671268
181.15806747 -18.39449859
   1.78487687 -15.99704297   1.6970333 ]
Lowest MAE: 0.27272987365722656 0 [ -0.46911404  -0.41254088  -9.09671268
181.15806747 -18.39449859
   1.78487687 -15.99704297   1.6970333 ]
Lowest RMSE: 0.3619910062138976 0 [ -0.46911404  -0.41254088  -9.09671268
181.15806747 -18.39449859
   1.78487687 -15.99704297   1.6970333 ]
Lowest MAPE: 0.04024988412857056 0 [ -0.46911404  -0.41254088  -9.09671268
181.15806747 -18.39449859
   1.78487687 -15.99704297   1.6970333 ]
Lowest DMI: -0.08161550666052719 0 [ -0.46911404  -0.41254088  -9.09671268
181.15806747 -18.39449859
   1.78487687 -15.99704297   1.6970333 ]
```

[59]:
```
xray_image_dzncc_nsga3 = applyLogScaleAndNegative(updateXRayImage(best_dzncc_X))
xray_image_mae_nsga3   = applyLogScaleAndNegative(updateXRayImage(best_mae_X))
xray_image_rmse_nsga3  = applyLogScaleAndNegative(updateXRayImage(best_rmse_X))
xray_image_dssim_nsga3 = applyLogScaleAndNegative(updateXRayImage(best_dssim_X))
xray_image_mape_nsga3  = applyLogScaleAndNegative(updateXRayImage(best_mape_X))
xray_image_dmi_nsga3   = applyLogScaleAndNegative(updateXRayImage(best_dmi_X))
```

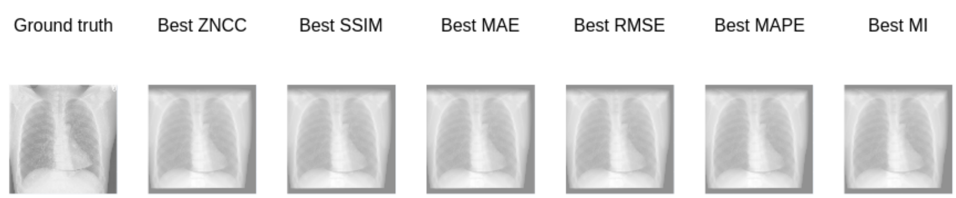
```
[60]: fig = make_subplots(rows=1, cols=7,
                    start_cell="bottom-left",
                    subplot_titles=("Ground truth", "Best ZNCC", "Best SSIM",␣
 ↪"Best MAE", "Best RMSE", "Best MAPE", "Best MI"))

nsga3_img_set = [standardised_roi_ground_truth,
                 standardisation(xray_image_dzncc_nsga3[y_min_id:y_max_id,␣
 ↪x_min_id:x_max_id]),
                 standardisation(xray_image_dssim_nsga3[y_min_id:y_max_id,␣
 ↪x_min_id:x_max_id]),
                 standardisation(xray_image_mae_nsga3[y_min_id:y_max_id,␣
 ↪x_min_id:x_max_id]),
                 standardisation(xray_image_rmse_nsga3[y_min_id:y_max_id,␣
 ↪x_min_id:x_max_id]),
                 standardisation(xray_image_mape_nsga3[y_min_id:y_max_id,␣
 ↪x_min_id:x_max_id]),
                 standardisation(xray_image_dmi_nsga3[y_min_id:y_max_id,␣
 ↪x_min_id:x_max_id])]

for n, image in enumerate(nsga3_img_set):

    im = px.imshow(image, aspect="equal", binary_string=True,␣
 ↪zmin=standardised_roi_ground_truth.min(), zmax=standardised_roi_ground_truth.
 ↪max())
    fig.add_trace(im.data[0], 1, n + 1)

fig.update_xaxes(showticklabels=False) # hide all the xticks
fig.update_yaxes(showticklabels=False) # hide all the yticks
fig.update_layout(coloraxis_showscale=False)

fig.update_layout(
    font_family="Arial",
    font_color="black",
    title_font_family="Arial",
    title_font_color="black",
    legend_title_font_color="black"
)

fig.update_layout(
    height=300,
    width=1200
)

fig.write_image("plots/lungman-NSGA3-objectives-cropped.pdf", engine="kaleido")
fig.write_image("plots/lungman-NSGA3-objectives-cropped.png", engine="kaleido")
```

```
fig.show()
```



[61]:
```
fig = make_subplots(rows=1, cols=7,
                    start_cell="bottom-left",
                    subplot_titles=("Ground truth", "Best ZNCC", "Best SSIM",␣
 ↪"Best MAE", "Best RMSE", "Best MAPE", "Best MI"))

standardised_ground_truth = standardisation(raw_reference_before_cropping)
nsga3_img_set = [standardised_ground_truth,
                 standardisation(xray_image_dzncc_nsga3),
                 standardisation(xray_image_dssim_nsga3),
                 standardisation(xray_image_mae_nsga3),
                 standardisation(xray_image_rmse_nsga3),
                 standardisation(xray_image_mape_nsga3),
                 standardisation(xray_image_dmi_nsga3)]

for n, image in enumerate(nsga3_img_set):

    im = px.imshow(image, aspect="equal", binary_string=True,␣
 ↪zmin=standardised_roi_ground_truth.min(), zmax=standardised_roi_ground_truth.
 ↪max())
    fig.add_trace(im.data[0], 1, n + 1)

fig.update_xaxes(showticklabels=False) # hide all the xticks
fig.update_yaxes(showticklabels=False) # hide all the yticks
fig.update_layout(coloraxis_showscale=False)

fig.update_layout(
    font_family="Arial",
    font_color="black",
    title_font_family="Arial",
    title_font_color="black",
    legend_title_font_color="black"
```

```
)

fig.update_layout(
    height=300,
    width=1200
)

fig.write_image("plots/lungman-NSGA3-objectives.pdf", engine="kaleido")
fig.write_image("plots/lungman-NSGA3-objectives.png", engine="kaleido")

fig.show()
```



[62]:
```
temp_res_nsga3_F = np.copy(res_nsga3_F[:, [0, 3, 5]])
temp_res_nsga3_F[:,0] = 1.0 - (2.0 * temp_res_nsga3_F[:,0])
temp_res_nsga3_F[:,1] = 1.0 - (2.0 * temp_res_nsga3_F[:,1])
temp_res_nsga3_F[:,2] = 1.0 - (2.0 * temp_res_nsga3_F[:,2])
```

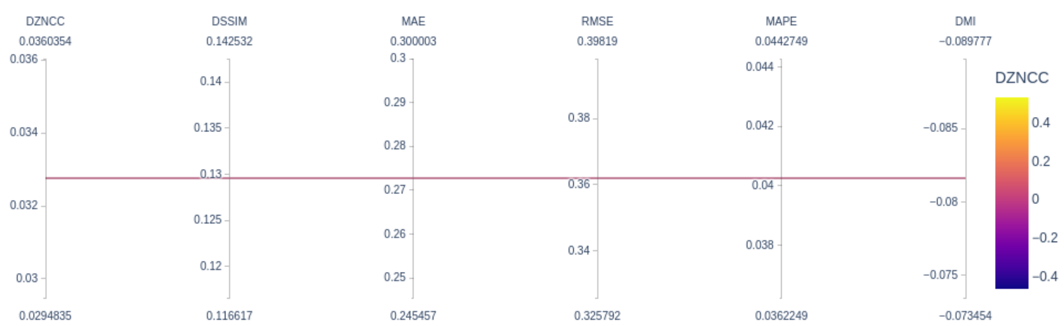
[63]:
```
new_array = np.append(res_nsga3_X, res_nsga3_F, axis=1)
new_array = np.append(new_array, temp_res_nsga3_F, axis=1)

columns=["sample_rotation_angle1", "sample_rotation_angle2", "src_pos_x",␣
 ↪"src_pos_y", "src_pos_z", "det_pos_x", "det_pos_y", "det_pos_z", "DZNCC",␣
 ↪"MAE", "RMSE", "DSSIM", "MAPE", "DMI", "ZNCC", "SSIM", "MI"]

print(new_array.shape)
print(len(columns))

df_nsga3 = pd.DataFrame(data=new_array,
                  columns=columns)

df_nsga3["Optimiser"] = "NSGA-III"
df_nsga3["Optimiser_code"] = 2
df_nsga3.to_csv("gVirtualXRay_output_data/lungman-optimiser-nsga3.csv")
```

```
(1, 17)
17
```

[64]:
```
display(df_nsga3)
```

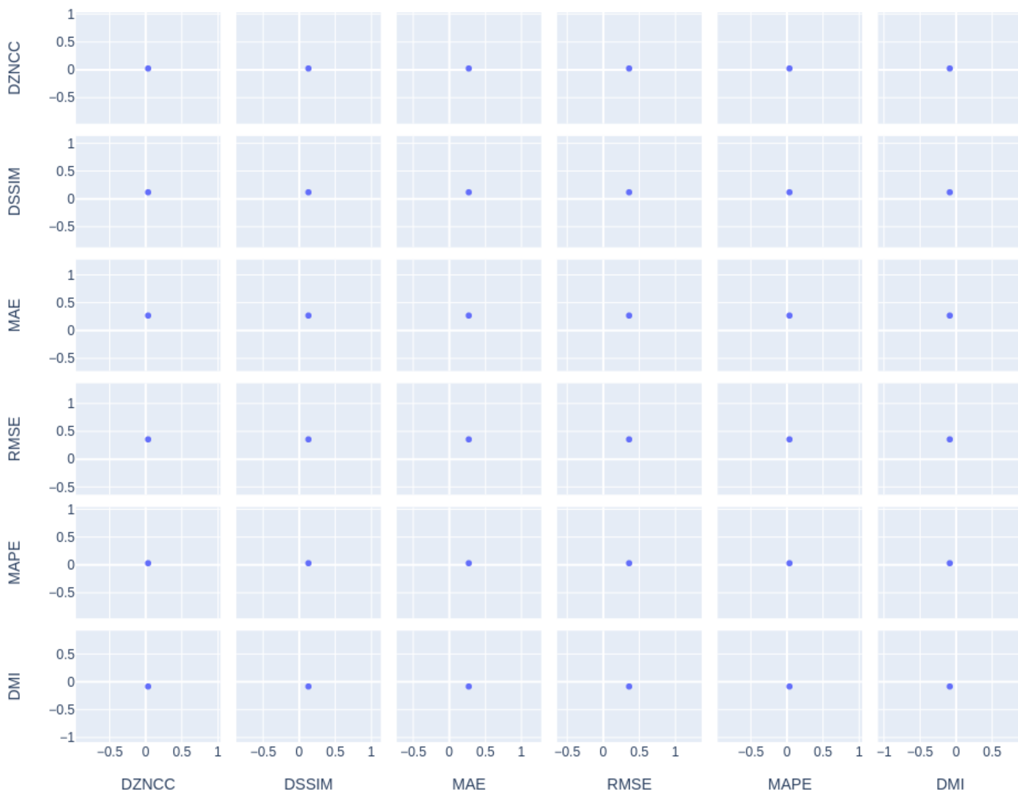
```
   sample_rotation_angle1  sample_rotation_angle2  src_pos_x   src_pos_y  \
0               -0.469114               -0.412541  -9.096713  181.158067

   src_pos_z  det_pos_x  det_pos_y  det_pos_z     DZNCC      MAE      RMSE  \
0 -18.394499   1.784877 -15.997043   1.697033  0.032759  0.27273  0.361991

      DSSIM      MAPE       DMI      ZNCC     SSIM        MI Optimiser  \
0  0.129575  0.04025 -0.081616  0.934481  0.74085  1.163231  NSGA-III

   Optimiser_code
0               2
```

[65]:
```
fig = px.parallel_coordinates(df_nsga3[["DZNCC", "DSSIM", "MAE", "RMSE", "MAPE",␣
↪"DMI"]], color="DZNCC")
fig.show()

fig.write_image("plots/lungman-nsga3-parallel_coordinates.pdf", engine="kaleido")
fig.write_image("plots/lungman-nsga3-parallel_coordinates.png", engine="kaleido")
```



[66]:
```
fig = px.scatter_matrix(df_nsga3[["DZNCC", "DSSIM", "MAE", "RMSE", "MAPE",␣
↪"DMI"]])

fig.update_layout(
    height=800,
    width=800
)

fig.show()
```

```python
fig.write_image("plots/lungman-nsga3-scatter_matrix.pdf", engine="kaleido")
fig.write_image("plots/lungman-nsga3-scatter_matrix.png", engine="kaleido")
```



4.4 Select a solution

[67]:
```python
standardised_corrected_xray_image =␣
 ↪standardisation(xray_image_dzncc_nsga3[y_min_id:y_max_id, x_min_id:x_max_id])
# standardised_corrected_xray_image =␣
 ↪standardisation(xray_image_dssim_nsga3[y_min_id:y_max_id, x_min_id:x_max_id])
# standardised_corrected_xray_image =␣
 ↪standardisation(xray_image_mae_nsga3[y_min_id:y_max_id, x_min_id:x_max_id])
# standardised_corrected_xray_image =␣
 ↪standardisation(xray_image_rmse_nsga3[y_min_id:y_max_id, x_min_id:x_max_id])
# standardised_corrected_xray_image =␣
 ↪standardisation(xray_image_mape_nsga3[y_min_id:y_max_id, x_min_id:x_max_id])
# standardised_corrected_xray_image =␣
 ↪standardisation(xray_image_dmi_nsga3[y_min_id:y_max_id, x_min_id:x_max_id])
```

```
imwrite("standardised_corrected_xray_image.tif",␣
↪standardised_corrected_xray_image.astype(np.single))
hist_ref = np.histogram(standardised_roi_ground_truth, 100)[0]

standardised_roi_ground_truth = standardised_roi_ground_truth.astype(np.single)
gX = cv2.Sobel(standardised_roi_ground_truth, ddepth=cv2.CV_32F, dx=1, dy=0)
gY = cv2.Sobel(standardised_roi_ground_truth, ddepth=cv2.CV_32F, dx=0, dy=1)
gX = cv2.convertScaleAbs(gX)
gY = cv2.convertScaleAbs(gY)
ref_grad_magn = cv2.addWeighted(gX, 0.5, gY, 0.5, 0)

hist_ref = np.histogram(ref_grad_magn, 100)[0]
```

[68]:
```
# Apply the transformation
updateXRayImage(res_nsga3_X[best_dzncc_id], restore_transformation=False);
```

[69]:
```
gvxr.displayScene()
gvxr.useNegative()

gvxr.setZoom(1339.6787109375)
gvxr.setSceneRotationMatrix([0.8227577805519104, 0.1368587613105774, -0.
↪5516625642776489, 0.0, -0.5680444240570068, 0.23148967325687408, -0.
↪7897683382034302, 0.0, 0.01961756870150566, 0.9631487131118774, 0.
↪26820749044418335, 0.0, 0.0, 0.0, 0.0, 1.0])

gvxr.setWindowBackGroundColour(0.5, 0.5, 0.5)

gvxr.displayScene()
```

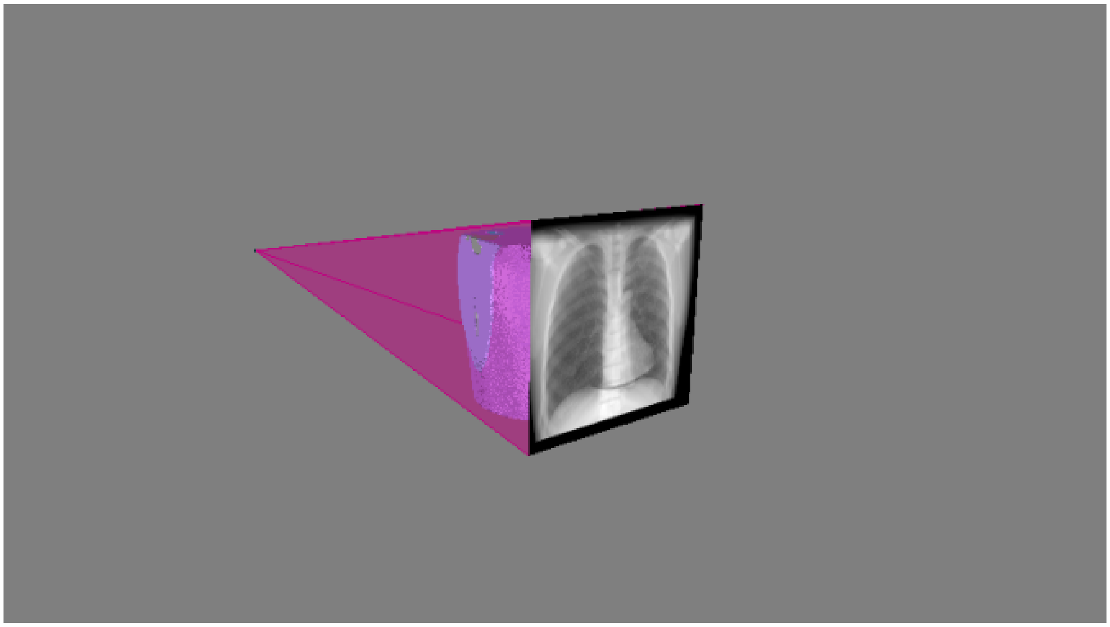
[70]:
```
# gvxr.renderLoop()
```

[71]:
```
# print(gvxr.getZoom())
# print(gvxr.getSceneRotationMatrix())
```

[72]:
```
screenshot = (255 * np.array(gvxr.takeScreenshot())).astype(np.uint8)
```

[73]:
```
plt.figure(figsize= (10,10))
plt.title("Screenshot")
plt.imshow(screenshot)
plt.axis('off')

plt.tight_layout()

plt.savefig('plots/lungman-optimised-screenshot-beam-on.pdf')
plt.savefig('plots/lungman-optimised-screenshot-beam-on.png')
```

Screenshot



5 Post-processing using image sharpening

We can see from the real image that an image sharpening filter was applied. We will implement one and optimise its parameters.

[74]:
```
def sharpen(image, ksize, shift, scale):
    details = image - gaussian(image, ksize)

    return (details + shift) * scale
```

5.1 Define an objective function

[75]:
```
# Compute the X-ray image
xray_image = np.flip(gvxr.computeXRayImage())

# Flat-field
xray_image_flat = xray_image[y_min_id:y_max_id, x_min_id:x_max_id] /
↪total_energy_in_MeV
```

[76]:
```
def objectiveFunctionSharpen(parameters):

    global xray_image_flat
```

```
    global standardised_roi_ground_truth

    # Retrieve the parameters
    shift1, scale1, sigma1, sigma2, shift2, scale2 = parameters

    # Process the image
    contrast = (xray_image_flat + shift1) * scale1
    details = sharpen(xray_image_flat, (sigma1, sigma2), shift2, scale2)
    sharpened = contrast + details
    sharpened[sharpened < 1e-9] = 1e-9
    log_image = np.log(contrast + details)
    negative = log_image * -1
    normalised = standardisation(negative)

    # Return the objective
    objective = math.sqrt(mean_squared_error(standardised_roi_ground_truth,␣
↪normalised))
    # objective = math.sqrt(mean_squared_error(ref_grad_magn, test_grad_magn))
    # objective = math.sqrt(mean_squared_error(hist_ref, hist_test))

    # objective = ref_grad_magn.sum() - test_grad_magn.sum()
    # objective *= objective
    # objective *= -1

    # mi = normalized_mutual_info_score(hist_ref, hist_test)
    # dmi = (1.0 - mi) / 2.0
    # objective = dmi

    return objective
```

5.2 Minimise the objective function

[77]:
```
sigma1 = 2
sigma2 = 2
alpha = 10.5
shift = 0
scale = 1

xl = [0, 0, 0, -5, 0]
xu = [10, 10, 15, 5, 2]
x_init = [sigma1, sigma2, alpha, shift, scale]

#0.003937458431052107 4012.600582499311 3.916470602237863 0.28374201341640476 6.
↪851503972136956 6.8658686847669045e-09
```

```python
xl = [0, 1e-9, 1, 1, 0, 0.5]
xu = [10000, 10000, 10, 10, 10000, 10000]
x_init = [0, 1, 3, 3, 0, 1]
```

[78]:
```python
# The registration has already been performed. Load the results.
if os.path.isfile("gVirtualXRay_output_data/lungman_postprocess.dat"):

    shift1, scale1, sigma1, sigma2, shift2, scale2 = np.
 ↪loadtxt("gVirtualXRay_output_data/lungman_postprocess.dat")

else:
    # Optimise
    timeout_in_sec = 20 * 60 # 20 minutes
    opts = cma.CMAOptions()
    opts.set('tolfun', 1e-10)
    opts['tolx'] = 1e-10
    opts['timeout'] = timeout_in_sec
    opts['bounds'] = [xl, xu]
    opts['CMA_stds'] = []

    for min_val, max_val in zip(opts['bounds'][0], opts['bounds'][1]):
        opts['CMA_stds'].append(abs(max_val - min_val) * 0.05)

    # Optimise
    es = cma.CMAEvolutionStrategy(x_init, 0.5, opts)
    es.optimize(objectiveFunctionSharpen)

    # Save the parameters
    shift1, scale1, sigma1, sigma2, shift2, scale2 = es.result.xbest
    np.savetxt("gVirtualXRay_output_data/lungman_postprocess.dat", [shift,␣
 ↪scale], header='shift,scale')

    # Release memory
    del es;
```

```
(4_w,9)-aCMA-ES (mu_w=2.8,w_1=49%) in dimension 6 (seed=312978, Thu Mar 24
20:46:50 2022)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      9 8.510483622118171e-01 1.0e+00 5.03e-01  2e-01  3e+02 0:00.9
    2     18 8.512543418819045e-01 1.3e+00 5.16e-01  2e-01  3e+02 0:01.8
    3     27 8.509555665870406e-01 1.5e+00 5.43e-01  2e-01  3e+02 0:02.7
    7     63 8.517665056499197e-01 2.0e+00 3.66e-01  2e-01  2e+02 0:06.2
   12    108 8.349227986393447e-01 2.4e+00 2.04e-01  8e-02  1e+02 0:10.7
   18    162 8.246925378578255e-01 2.8e+00 1.15e-01  4e-02  6e+01 0:16.1
   25    225 7.888171013126792e-01 3.8e+00 5.03e-02  2e-02  2e+01 0:22.4
   33    297 7.664114051503951e-01 5.9e+00 3.48e-02  1e-02  2e+01 0:29.5
   42    378 6.942725808298136e-01 1.5e+01 2.83e-02  9e-03  2e+01 0:37.6
```

```
  53    477 3.490412482511431e-01 4.5e+01 1.44e-02  4e-03  1e+01 0:47.5
  65    585 3.523158801706301e-01 1.1e+02 8.77e-03  3e-03  6e+00 0:58.3
  78    702 3.490626945364066e-01 2.2e+02 5.98e-03  2e-03  4e+00 1:10.0
  92    828 3.488153538226098e-01 4.8e+02 3.20e-03  1e-03  2e+00 1:22.5
 100    900 3.487955213794939e-01 9.5e+02 2.06e-03  1e-03  1e+00 1:29.6
 116   1044 3.487926920278356e-01 3.3e+03 1.49e-03  7e-04  1e+00 1:44.0
 133   1197 3.487916909208646e-01 5.7e+03 7.74e-04  3e-04  5e-01 1:59.3
 151   1359 3.487916733365316e-01 1.1e+04 4.20e-04  1e-04  2e-01 2:15.7
 170   1530 3.487916530177975e-01 4.8e+04 3.42e-04  1e-04  3e-01 2:32.8
 191   1719 3.487915302336575e-01 1.7e+05 2.92e-03  1e-03  5e+00 2:51.6
 200   1800 3.487913678120540e-01 2.3e+05 3.10e-03  8e-04  7e+00 2:59.6
NOTE (module=cma, iteration=201):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 7.9e+10 to 4.4e+03
 222   1998 3.487910360116451e-01 8.6e+01 3.87e-03  3e-04  1e+01 3:19.9
 245   2205 3.487901393947447e-01 2.3e+02 6.92e-03  8e-05  3e+01 3:41.4
 268   2412 3.487886626741367e-01 8.9e+02 4.68e-02  1e-04  6e+02 4:04.5
 286   2574 3.487884400288988e-01 4.4e+03 7.53e-02  5e-05  1e+03 4:28.9
 300   2700 3.487884196117332e-01 6.2e+03 1.67e-02  7e-06  2e+02 4:49.1
 318   2862 3.487884182250678e-01 5.3e+03 7.40e-03  2e-06  4e+01 5:15.1
 337   3033 3.487884157988346e-01 5.4e+03 2.77e-02  1e-05  8e+01 5:42.6
 356   3204 3.487884046740885e-01 6.5e+03 1.28e-01  4e-05  2e+02 6:09.8
 379   3411 3.487881788184747e-01 8.7e+03 1.23e+00  2e-04  2e+03 6:38.2
 400   3600 3.487880809762774e-01 1.3e+04 9.06e-01  2e-04  9e+02 6:58.2
 440   3960 3.487880001910424e-01 7.8e+03 3.27e-01  5e-05  2e+02 7:28.2
 483   4347 3.487879843176845e-01 1.3e+04 3.15e-01  8e-05  3e+02 7:59.3
 500   4500 3.487879835415664e-01 7.1e+03 3.57e-01  4e-05  1e+02 8:11.6
 546   4914 3.487879737564441e-01 5.9e+03 8.89e-01  4e-05  1e+02 8:44.8
 594   5346 3.487879710506298e-01 7.4e+03 1.04e+00  4e-05  6e+01 9:19.1
 600   5400 3.487879710135983e-01 6.4e+03 1.03e+00  4e-05  5e+01 9:23.4
 651   5859 3.487879706219256e-01 6.8e+03 7.41e-01  1e-05  2e+01 9:59.7
 653   5877 3.487879706201411e-01 6.8e+03 7.36e-01  1e-05  2e+01 10:01.2
```

5.3 Apply the result of the optimisation

[79]:
```
print(shift1, scale1, sigma1, sigma2, shift2, scale2) #    return scale * (shift␣
↪+ image) + alpha * details

# Process the image
contrast = (xray_image_flat + shift1) * scale1
details = sharpen(xray_image_flat, (sigma1, sigma2), shift2, scale2)
sharpened = contrast + details
sharpened[sharpened < 1e-9] = 1e-9
log_image = np.log(contrast + details)
negative = log_image * -1
normalised = standardisation(negative)
```

```
0.004478964839408227 9999.994574454577 1.0000041157105053 1.0000189315650312
13.28768841817979 0.500038491169309
```

[80]:
```python
font = {'size'   : 12.5
       }
matplotlib.rc('font', **font)
```

[81]:
```python
new_image_without_post_process = standardised_corrected_xray_image *␣
 ↪raw_reference.std()
new_image_without_post_process -= standardised_corrected_xray_image.mean()
new_image_without_post_process += raw_reference.mean()

new_image_with_post_process = normalised * raw_reference.std()
new_image_with_post_process -= normalised.mean()
new_image_with_post_process += raw_reference.mean()
```

[82]:
```python
old_zncc = np.mean(standardised_roi_ground_truth *␣
 ↪standardised_corrected_xray_image)
new_zncc = np.mean(standardised_roi_ground_truth * normalised)
```

[83]:
```python
old_mape = mape(raw_reference, new_image_without_post_process)
new_mape = mape(raw_reference, new_image_with_post_process)
```

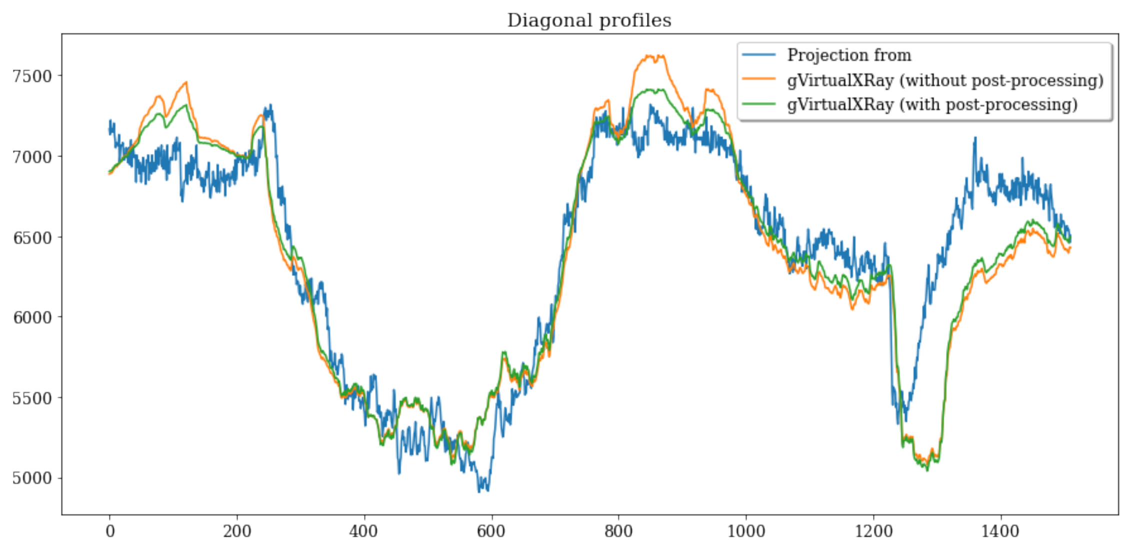
[84]:
```python
old_ssim = ssim(raw_reference, new_image_without_post_process,␣
 ↪data_range=raw_reference.max() - raw_reference.min())
new_ssim = ssim(raw_reference, new_image_with_post_process,␣
 ↪data_range=raw_reference.max() - raw_reference.min())
```

[85]:
```python
print("ZNCC before sharpening:", str(100 * old_zncc) + "%")
print("ZNCC after sharpening:", str(100 * new_zncc) + "%")

print("MAPE before sharpening:", str(100 * old_mape) + "%")
print("MAPE after sharpening:", str(100 * new_mape) + "%")

print("SSIM before sharpening:", str(old_ssim))
print("SSIM after sharpening:", str(new_ssim))
```

```
ZNCC before sharpening: 93.4481250501381%
ZNCC after sharpening: 93.91734741059904%
MAPE before sharpening: 3.7314359267254957%
MAPE after sharpening: 3.5164271567829983%
SSIM before sharpening: 0.9143491020519106
SSIM after sharpening: 0.914014019499426
```

[86]:
```python
ref_diag = np.diag(raw_reference)
test_diag = np.diag(new_image_without_post_process)
sharpen_test_diag = np.diag(new_image_with_post_process)
```

```
plt.figure(figsize=(15, 7))

ax = plt.subplot(111)

ax.set_title("Diagonal profiles")

ax.plot(ref_diag, label="Projection from ")
ax.plot(test_diag, label="gVirtualXRay (without post-processing)")
ax.plot(sharpen_test_diag, label="gVirtualXRay (with post-processing)")

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

# plt.legend()

plt.savefig('plots/lungman-profiles-projection-postprocessing.pdf')
plt.savefig('plots/lungman-profiles-projection-postprocessing.png')
```

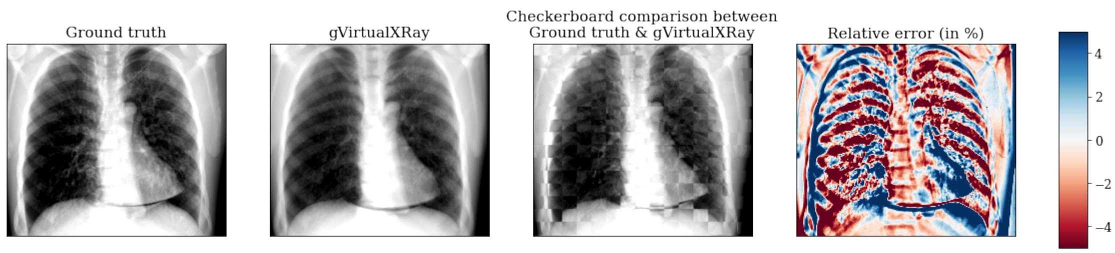


[87]:
```
window_centre = int(volume.GetMetaData("0028|1050").split("\\")[1]) # Use 0 for␣
↪normal, 1 for harder, 2 for softer
window_width = int(volume.GetMetaData("0028|1051").split("\\")[1]) # Use 0 for␣
↪normal, 1 for harder, 2 for softer

print("Window Center used: ", window_centre)
print("Window Width used: ", window_width)

vmin = window_centre - window_width / 2
vmax = window_centre + window_width / 2
```

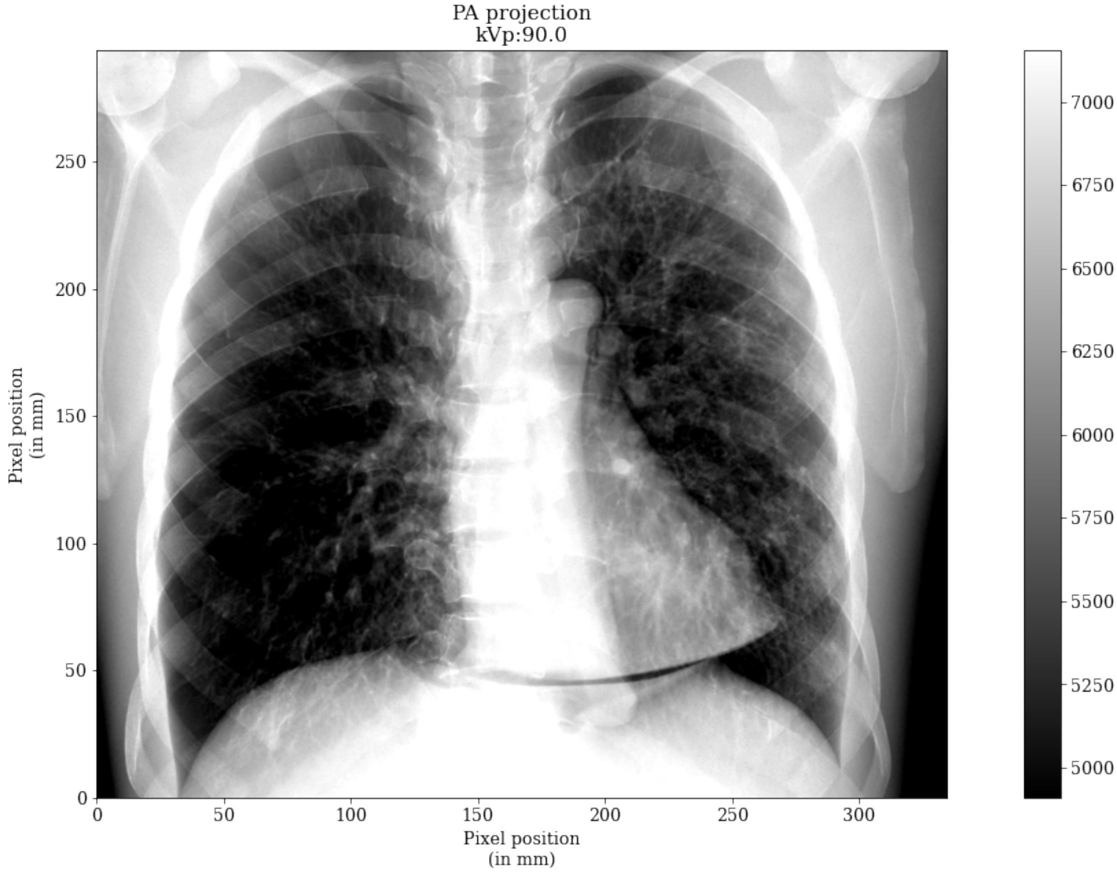
```
view_position = volume.GetMetaData("0018|5101")
```

```
Window Center used:  6032
Window Width used:  2245
```

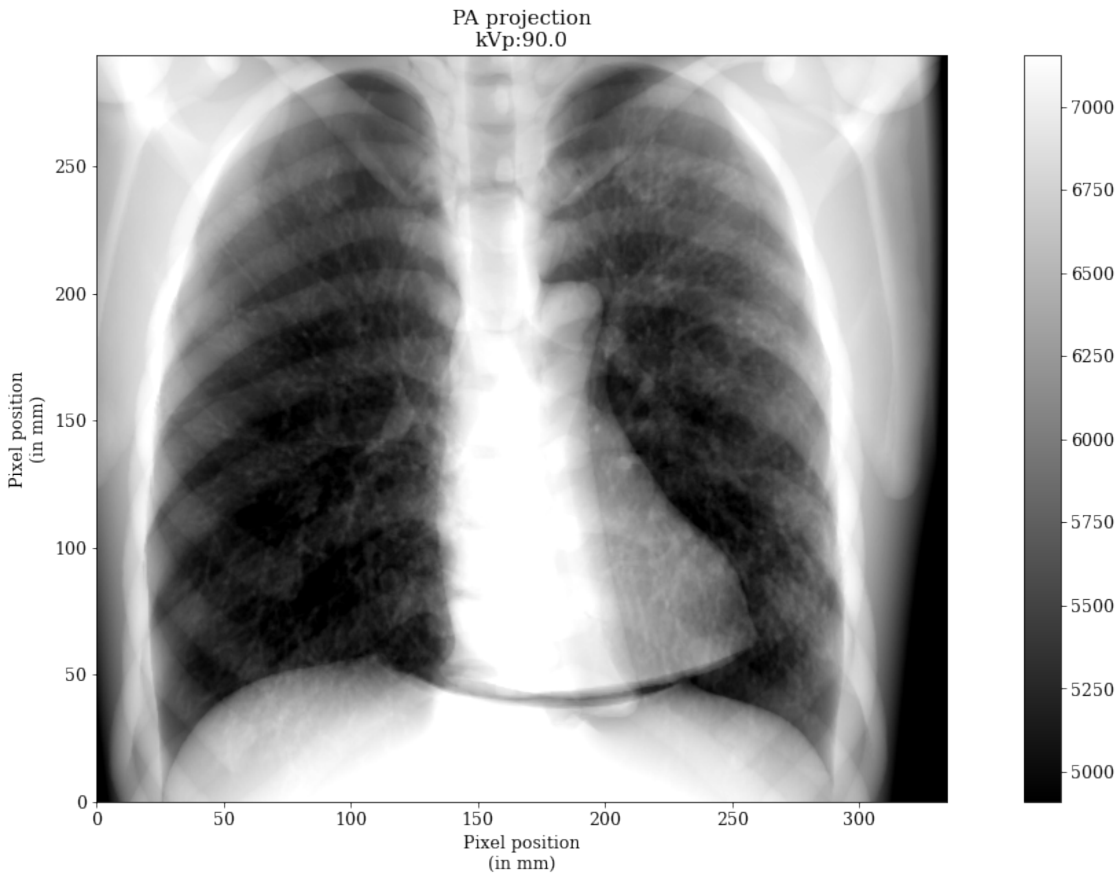
[88]:
```
fullCompareImages(raw_reference,
                  new_image_with_post_process,
                  "gVirtualXRay",
                  "plots/lungman-projection-harder", vmin=vmin, vmax=vmax,
                 avoid_div_0=False)
```



[89]:
```
plt.figure(figsize= (20,10))
xrange=range(raw_reference.shape[1])
yrange=range(raw_reference.shape[0])

plt.xlabel("Pixel position\n(in mm)")
plt.ylabel("Pixel position\n(in mm)")
plt.title(view_position + " projection\nkVp:" + str(kVp))
plt.imshow(raw_reference, cmap="gray",
           vmin=vmin, vmax=vmax,
           extent=[0,(raw_reference.shape[1]-1)*spacing[0],0,(raw_reference.
 ↪shape[0]-1)*spacing[1]])
plt.colorbar(orientation='vertical')

plt.savefig('plots/lungman_experimental_DX_image-harder.pdf')
plt.savefig('plots/lungman_experimental_DX_image-harder.png')
```



[90]:
```
plt.figure(figsize= (20,10))
xrange=range(raw_reference.shape[1])
yrange=range(raw_reference.shape[0])

plt.xlabel("Pixel position\n(in mm)")
plt.ylabel("Pixel position\n(in mm)")
plt.title(view_position + " projection\nkVp:" + str(kVp))
plt.imshow(new_image_with_post_process, cmap="gray",
           vmin=vmin, vmax=vmax,
           extent=[0,(raw_reference.shape[1]-1)*spacing[0],0,(raw_reference.
 ↪shape[0]-1)*spacing[1]])
plt.colorbar(orientation='vertical')

plt.savefig('plots/lungman_simulated_DX_image-harder.pdf')
plt.savefig('plots/lungman_simulated_DX_image-harder.png')
```



[91]:
```
runtimes = []

resetToDefaultParameters()
setTransformations(res_nsga3_X[best_dzncc_id])

for i in range(25):
    start_time = datetime.datetime.now()

    raw_x_ray_image = np.array(gvxr.computeXRayImage())

    # Process the image
    xray_image_flat = raw_x_ray_image / total_energy_in_MeV
    contrast = (xray_image_flat + shift1) * scale1
    details = sharpen(xray_image_flat, (sigma1, sigma2), shift2, scale2)
    sharpened = contrast + details
    sharpened[sharpened < 1e-9] = 1e-9
    log_image = np.log(contrast + details)
    negative = log_image * -1
```

```
    end_time = datetime.datetime.now()
    delta_time = end_time - start_time
    runtimes.append(delta_time.total_seconds() * 1000)
```

[92]:
```
runtime_avg = round(np.mean(runtimes))
runtime_std = round(np.std(runtimes))

raw_x_ray_image = np.array(raw_x_ray_image)
```

[93]:
```
ZNCC = df_nsga3["ZNCC"].max()
SSIM = df_nsga3["SSIM"].max()
MAPE = df_nsga3["MAPE"].max()

print("Lungman PA view & Real digital radiograph  & " +
      "{0:0.2f}".format(100 * new_mape) + "\\%    &    " +
      "{0:0.2f}".format(100 * new_zncc) + "\\%    &    " +
      "{0:0.2f}".format(new_ssim) + "    &    $" +
      str(raw_x_ray_image.shape[1]) + " \\times " + str(raw_x_ray_image.
 ↪shape[0]) + "$    &    N/A    &" +
      str(number_of_triangles) + "    &    " +
      "$" + str(runtime_avg) + " \\pm " + str(runtime_std) + "$  & N/A \\\\")
```

```
Lungman PA view & Real digital radiograph  & 3.52\%    &    93.92\%    &    0.91
&    $1871 \times 1881$    &    N/A    &16528580    &    $471 \pm 68$  & N/A \\
```

5.4 All done

Destroy the window

[94]:
```
gvxr.destroyAllWindows()
```