4-gVirtualXRay_vs_VHP

September 15, 2022

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

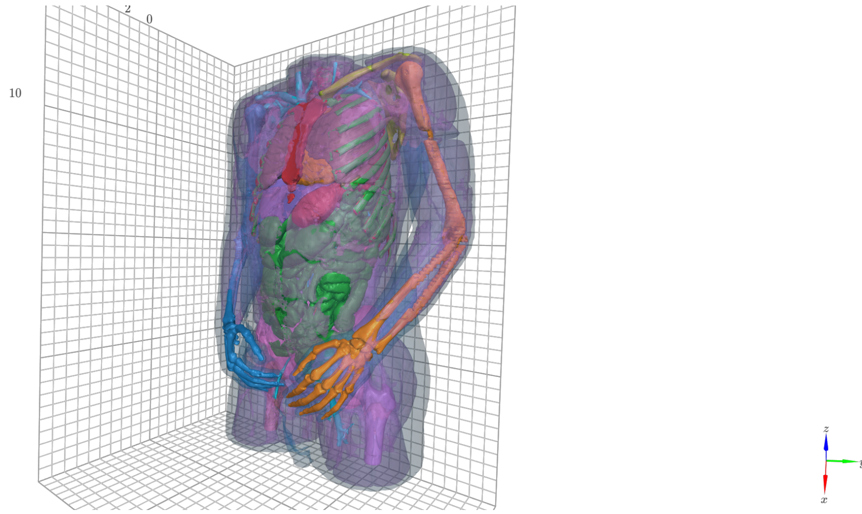
[2]:
```python
output_path = "4-output_data/"

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

if not os.path.exists(output_path + "/NSGA2"):
    os.mkdir(output_path + "/NSGA2")

if not os.path.exists(output_path + "/NSGA3"):
    os.mkdir(output_path + "/NSGA3")
```

Main contributors: T. Wen, A. Sujar and F. P. Vidal

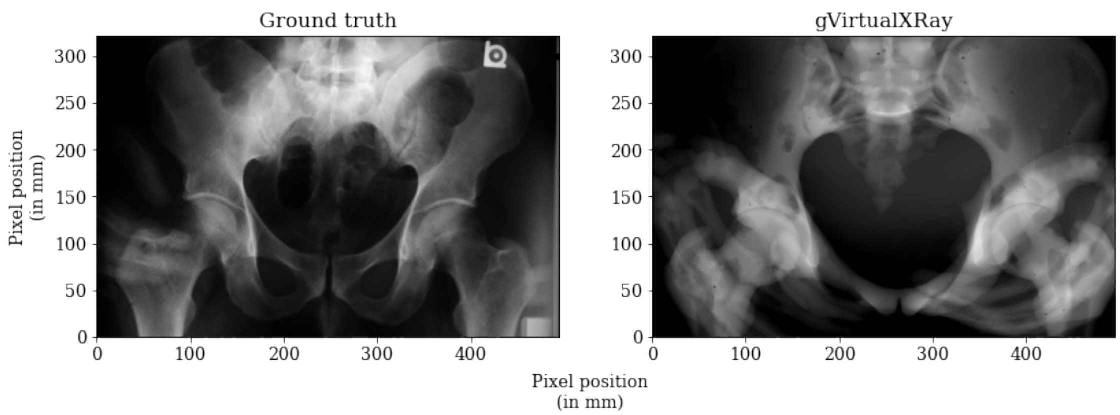
Purpose: In this notebook, we aim to demonstrate that gVirtualXRay is able to generate analytic simulations on GPU comparable to real images from the Visible Human Project (VHP). We register the data provided by VoxelMan segmented from the cadaver's CT and cryosections onto a pelvis radiograph of the human participant. All the acquisition parameters are unknown and would need to be estimated.

Material and Methods: We use the definitions of tissue substitutes provided in the ICRU Report 44 by the International Commission on Radiation Units and Measurements.

[3]:
```python
fname = output_path + "/VHP_model.png"
if os.path.exists(fname):
    display(Image(fname))
```



Results: The **zero-mean normalised cross-correlation** is **59.50%**. The **Structural Similarity Index (SSIM)** is **0.38**.



The calculations were performed on the following platform:

[4]:
```
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
```

1 Import packages

[5]:
```python
%matplotlib inline

import os # Locate files
from time import sleep
from pathlib import Path

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
import matplotlib.image as mpimg # To save PNG files from numpy arrays

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 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
import cv2
from skimage.filters import gaussian # Implementing the image sharpening filter
```

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

import viewscad # Use OpenSCAD to create STL files

from scipy.spatial import distance # Euclidean distance

# 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, unlzw3 # To download the phantom data, and extract the
    ↪corresponding Z file


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

from pymoo.algorithms.soo.nonconvex.cmaes import CMAES
from pymoo.algorithms.moo.nsga2 import NSGA2
from pymoo.factory import get_problem
from pymoo.optimize import minimize
from pymoo.visualization.scatter import Scatter
from pymoo.util.normalization import denormalize

import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots

import sys
```

```
SimpleGVXR 2.0.2 (2022-09-14T12:57:17) [Compiler: GNU g++] on Linux
gVirtualXRay core library (gvxr) 2.0.2 (2022-09-14T12:57:03) [Compiler: GNU g++]
on Linux
```

[6]:
```python
def standardisation(img):
    return (img - img.mean()) / img.std()
```

2 Reference image

We first load the reference image from the Visible Human Project. You can find it at https://data.lhncbc.nlm.nih.gov/public/Visible-Human/Male-Images/radiological/xray8/x_vm_pe.Z. When you download it, make sure to gunzip it!

[7]:
```
if not os.path.exists("VHP"):
    os.mkdir("VHP")

if not os.path.exists("VHP/x_vm_pe.Z"):
    urllib.request.urlretrieve("https://data.lhncbc.nlm.nih.gov/public/
 ↪Visible-Human/Male-Images/radiological/xray14/x_vm_pe.Z", "VHP/x_vm_pe.Z")

if not os.path.exists("VHP/x_vm_pe"):

    file_content = unlzw3.unlzw(Path("VHP/x_vm_pe.Z"))
    f = open('VHP/x_vm_pe', 'wb')
    f.write(file_content)
    f.close()
```

[8]:
```
raw_reference = np.fromfile("VHP/x_vm_pe", dtype='>H') # UINT16 in big endian
raw_reference.shape = (1536,1248)
raw_reference = np.rot90(raw_reference)

# Crop
y_max = 1000
raw_reference = raw_reference[:y_max]
```

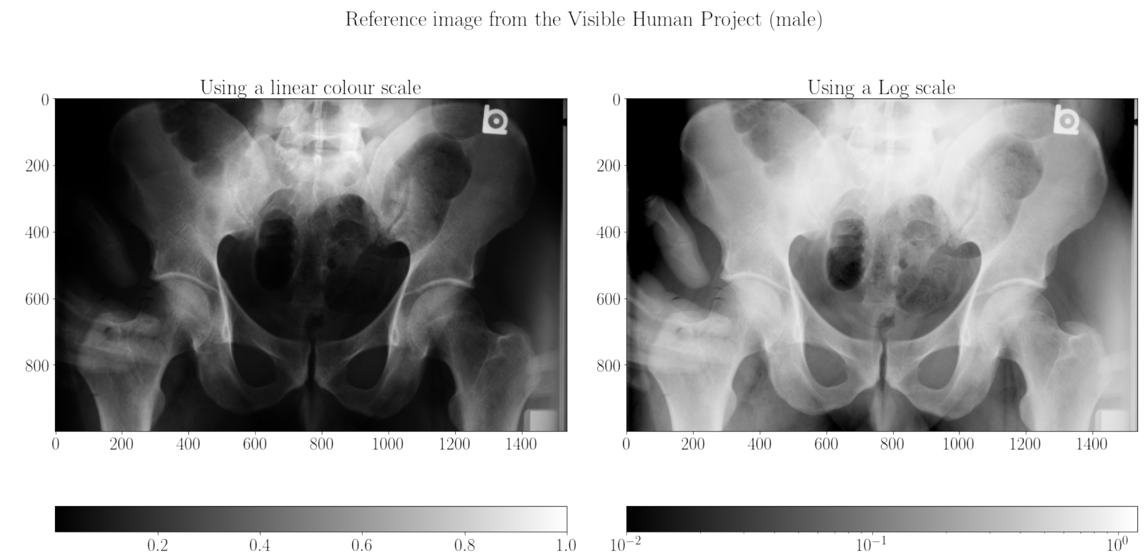
[9]:
```
imwrite(output_path + '/real_projection-VHP.tif', raw_reference.astype(np.
 ↪single))
```

[10]:
```
corrected_real_projection = raw_reference.astype(np.single) / raw_reference.
 ↪max()
```

[11]:
```
imwrite(output_path + '/corrected_real_projection-VHP.tif',␣
 ↪corrected_real_projection.astype(np.single))
```

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

[12]:
```
displayLinearPowerScales(corrected_real_projection,
                         "Reference image from the Visible Human Project␣
 ↪(male)",
                         output_path + "/reference-VHP",
                         log=True)
```



Apply a log transformation

[13]:
```
ground_truth = raw_reference
log_ground_truth = np.log(corrected_real_projection)
normalised_log_ground_truth = standardisation(log_ground_truth)

# imwrite(output_path + '/ground_truth-VHP.tif', ground_truth.astype(np.single))
```

3 Setting up gVirtualXRay

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

[14]:
```
json2gvxr.initGVXR("notebook-4.json", "EGL")
```

```
Create an OpenGL context: 800x450

Thu Sep 15 13:58:53 2022 ---- Create window (ID: -1)
Thu Sep 15 13:58:53 2022 ---- Query the number of EGL devices
Thu Sep 15 13:58:53 2022 ---- Success
Thu Sep 15 13:58:53 2022 ---- Detected 2 EGL devices.
Thu Sep 15 13:58:53 2022 ---- Print the details here of every EGL device.
Thu Sep 15 13:58:53 2022 ---- Success
Thu Sep 15 13:58:53 2022 ---- Device 1/2:
Thu Sep 15 13:58:53 2022 ----   Device Extensions: EGL_NV_device_cuda
EGL_EXT_device_drm EGL_EXT_device_drm_render_node EGL_EXT_device_query_name
EGL_EXT_device_persistent_id
Thu Sep 15 13:58:53 2022 ----   Device vendor: NVIDIA
Thu Sep 15 13:58:53 2022 ----   Failed to retrieve EGL DRM device file.
Thu Sep 15 13:58:53 2022 ---- Device 2/2:
Thu Sep 15 13:58:53 2022 ----   Device Extensions: EGL_MESA_device_software
```

```
Thu Sep 15 13:58:53 2022 ----   Failed to retrieve device vendor.
Thu Sep 15 13:58:53 2022 ----   Failed to retrieve EGL DRM device file.
Thu Sep 15 13:58:53 2022 ---- EGL client extensions: EGL_EXT_platform_base
EGL_EXT_device_base EGL_EXT_device_enumeration EGL_EXT_device_query
EGL_KHR_client_get_all_proc_addresses EGL_EXT_client_extensions EGL_KHR_debug
EGL_KHR_platform_x11 EGL_EXT_platform_x11 EGL_EXT_platform_device
EGL_KHR_platform_wayland EGL_EXT_platform_wayland EGL_KHR_platform_gbm
EGL_MESA_platform_gbm EGL_MESA_platform_xcb EGL_MESA_platform_surfaceless
Thu Sep 15 13:58:53 2022 ---- EGL, find the default display
Thu Sep 15 13:58:53 2022 ----   SUCCESS
Thu Sep 15 13:58:53 2022 ---- Initialise EGL
Thu Sep 15 13:58:53 2022 ---- EGL version: 1.5
Thu Sep 15 13:58:53 2022 ---- Bind the OpenGL API to EGL
Thu Sep 15 13:58:53 2022 ---- Create the context
Thu Sep 15 13:58:53 2022 ---- Create the surface
Thu Sep 15 13:58:53 2022 ---- Make the context current
Thu Sep 15 13:58:53 2022 ---- Initialise GLEW
Thu Sep 15 13:58:53 2022 ---- OpenGL version supported by this platform 4.5.0
NVIDIA 515.48.07
Thu Sep 15 13:58:53 2022 ---- OpenGL vendor:NVIDIA Corporation
Thu Sep 15 13:58:53 2022 ---- OpenGL renderer:NVIDIA GeForce RTX 2080
Ti/PCIe/SSE2
Thu Sep 15 13:58:53 2022 ---- OpenGL version:4.5.0 NVIDIA 515.48.07
Thu Sep 15 13:58:53 2022 ---- Use OpenGL 4.5.
Thu Sep 15 13:58:53 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.

[15]:
```
json2gvxr.initSourceGeometry()
```

```
Set up the beam
        Source position: [0.0, -30.5, 150.0, 'cm']
        Source shape: PointSource
```

3.2 Spectrum

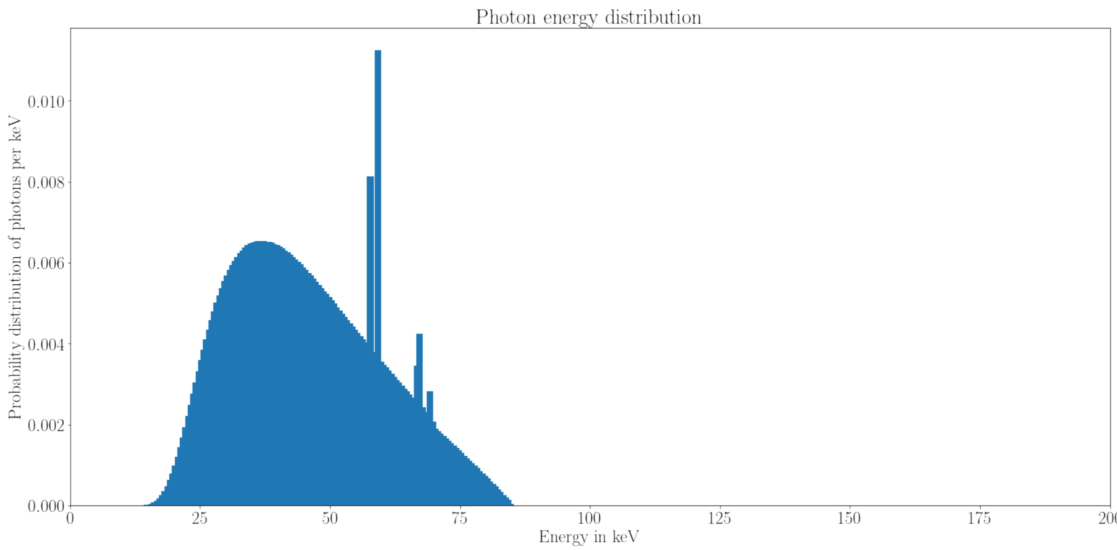
The spectrum is polychromatic.

[16]:
```
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])
```

```
params["Source"]["Beam"] {'kvp': 85, 'tube angle': 12, 'filter': [['Al', 3.2]]}
['Al', 3.2]
```

Plot the spectrum

[17]:
```python
plotSpectrum(k, f, output_path + '/spectrum-VHP')
```



3.3 Detector

Create a digital detector

[18]:
```python
json2gvxr.initDetector()
```

```
Set up the detector
        Detector position: [0.0, -30.5, -20.5, 'cm']
        Detector up vector: [0, -1, 0]
        Detector number of pixels: [1536, 1248]
        Pixel spacing: [0.026347682927083334, 0.026347683, 'cm']
Thu Sep 15 13:58:54 2022 ---- Initialise the renderer
```

3.4 Sample

We now load the models segmented from the Visible Human.

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

[19]:
```python
json2gvxr.initSamples(verbose=0)
```

```
Thu Sep 15 13:58:56 2022 ---- file_name:        VHP/meshes/armR.stl
nb_faces:       33838   nb_vertices:    101514  bounding_box (in cm):
```

```
(-10.0264, -10.1334, -2.8384)   (0.294166, 10.694, 6.13302)
Thu Sep 15 13:58:56 2022 ---- file_name:        VHP/meshes/armL.stl
nb_faces:       33475   nb_vertices:    100425  bounding_box (in cm):
(1.14313, -10.7634, -3.28228)   (9.46749, 11.1713, 5.74204)
Thu Sep 15 13:58:57 2022 ---- file_name:        VHP/meshes/chest.stl
nb_faces:       89235   nb_vertices:    267705  bounding_box (in cm):
(-5.19216, -2.22134, -4.18652)  (5.28961, 12.0778, 3.51821)
Thu Sep 15 13:58:57 2022 ---- file_name:        VHP/meshes/chest2.stl
nb_faces:       75408   nb_vertices:    226224  bounding_box (in cm):
(-5.20763, -2.213, -4.16399)    (5.26581, 12.0682, 4.06685)
Thu Sep 15 13:58:57 2022 ---- file_name:        VHP/meshes/femurL.stl
nb_faces:       9660    nb_vertices:    28980   bounding_box (in cm):   (2.2332,
-14.6571, -1.851)       (5.5281, -8.86371, 0.57499)
Thu Sep 15 13:58:58 2022 ---- file_name:        VHP/meshes/femurR.stl
nb_faces:       9660    nb_vertices:    28980   bounding_box (in cm):
(-5.60176, -14.6576, -1.48193)  (-2.25352, -8.82717, 0.79766)
Thu Sep 15 13:58:58 2022 ---- file_name:        VHP/meshes/hips.stl
nb_faces:       56575   nb_vertices:    169725  bounding_box (in cm):
(-4.86403, -12.1307, -4.17882)  (4.97615, -4.45668, 1.91861)
Thu Sep 15 13:58:58 2022 ---- file_name:        VHP/meshes/bladder.stl
nb_faces:       2706    nb_vertices:    8118    bounding_box (in cm):
(-1.47306, -10.5873, -0.88731)  (0.947542, -9.09488, 1.8016)
Thu Sep 15 13:58:58 2022 ---- file_name:        VHP/meshes/bronch.stl
nb_faces:       1658    nb_vertices:    4974    bounding_box (in cm):
(-1.94132, 5.95348, -1.32576)   (1.82832, 8.46616, 0.070288)
Thu Sep 15 13:58:58 2022 ---- file_name:        VHP/meshes/circulatory.stl
nb_faces:       28012   nb_vertices:    84036   bounding_box (in cm):
(-4.15496, -13.5437, -2.42441)  (3.83071, 11.9091, 2.33955)
Thu Sep 15 13:58:58 2022 ---- file_name:        VHP/meshes/gallbladder.stl
nb_faces:       1172    nb_vertices:    3516    bounding_box (in cm):
(-3.57645, 0.788212, 0.717928)  (-2.14436, 2.12213, 3.22025)
Thu Sep 15 13:58:58 2022 ---- file_name:        VHP/meshes/heart.stl
nb_faces:       19290   nb_vertices:    57870   bounding_box (in cm):
(-1.82649, 3.34164, -0.817277)  (3.04628, 8.98106, 3.46046)
Thu Sep 15 13:58:59 2022 ---- file_name:        VHP/meshes/intestine.stl
nb_faces:       111010  nb_vertices:    333030  bounding_box (in cm):
(-4.22381, -9.67662, -1.68918)  (4.91206, 3.18448, 4.39048)
Thu Sep 15 13:58:59 2022 ---- file_name:        VHP/meshes/kidney.stl
nb_faces:       18283   nb_vertices:    54849   bounding_box (in cm):
(-3.52353, -2.96092, -2.65465)  (3.8139, 1.46367, 0.675229)
Thu Sep 15 13:59:00 2022 ---- file_name:        VHP/meshes/liver.stl
nb_faces:       29688   nb_vertices:    89064   bounding_box (in cm):
(-4.89368, -1.17826, -3.13392)  (2.02827, 5.00414, 3.76076)
Thu Sep 15 13:59:00 2022 ---- file_name:        VHP/meshes/lungs.stl
nb_faces:       68674   nb_vertices:    206022  bounding_box (in cm):
(-4.61199, 2.87393, -3.52451)   (4.57452, 10.5346, 3.27177)
Thu Sep 15 13:59:04 2022 ---- file_name:        VHP/meshes/muscles.stl
nb_faces:       553157  nb_vertices:    1659471 bounding_box (in cm):
```

```
(-10.1643, -14.6703, -4.70584)  (9.63791, 12.0829, 5.65443)
Thu Sep 15 13:59:04 2022 ---- file_name:        VHP/meshes/pancreas.stl
nb_faces:       5638    nb_vertices:    16914   bounding_box (in cm):
(-1.25805, -0.822349, -1.27119) (2.71833, 2.29143, 1.78095)
Thu Sep 15 13:59:04 2022 ---- file_name:        VHP/meshes/shoulderL.stl
nb_faces:       12548   nb_vertices:    37644   bounding_box (in cm):
(0.437246, 5.94084, -4.08889)   (6.5641, 12.0494, 1.94009)
Thu Sep 15 13:59:04 2022 ---- file_name:        VHP/meshes/shoulderR.stl
nb_faces:       12454   nb_vertices:    37362   bounding_box (in cm):
(-7.24934, 5.67127, -4.06742)   (-0.804483, 11.5342, 1.83824)
Thu Sep 15 13:59:05 2022 ---- file_name:        VHP/meshes/skin.stl
nb_faces:       64068   nb_vertices:    192204  bounding_box (in cm):
(-10.2747, -14.6631, -5.1876)   (9.70979, 12.1115, 6.36175)
Thu Sep 15 13:59:05 2022 ---- file_name:        VHP/meshes/spine.stl
nb_faces:       22205   nb_vertices:    66615   bounding_box (in cm):
(-1.63024, -5.87568, -3.46047)  (1.92212, 0.863915, 0.495364)
Thu Sep 15 13:59:05 2022 ---- file_name:        VHP/meshes/spleen.stl
nb_faces:       6096    nb_vertices:    18288   bounding_box (in cm):
(0.76985, -0.12949, -3.09199)   (4.73059, 3.94038, 0.045474)
Thu Sep 15 13:59:05 2022 ---- file_name:        VHP/meshes/stomach.stl
nb_faces:       8928    nb_vertices:    26784   bounding_box (in cm):
(-0.162305, 0.659165, -2.26369) (3.41186, 4.32818, 3.96684)
```

[20]:
```
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

    print(label, \
          "has", \
          f"{number_of_triangles_in_mesh:,}", \
          "triangles.")

print("\nThere are", f"{number_of_triangles:,}", "triangles in total")
```

```
armR has 33,838 triangles.
armL has 33,475 triangles.
rib cage has 89,235 triangles.
sternum has 75,408 triangles.
femurL has 9,660 triangles.
femurR has 9,660 triangles.
hips has 56,575 triangles.
bladder has 2,706 triangles.
bronch has 1,658 triangles.
circulatory has 28,012 triangles.
gallbladder has 1,172 triangles.
```

```
heart has 19,290 triangles.
intestine has 111,010 triangles.
kidney has 18,283 triangles.
liver has 29,688 triangles.
lungs has 68,674 triangles.
muscles has 553,157 triangles.
pancreas has 5,638 triangles.
shoulderL has 12,548 triangles.
shoulderR has 12,454 triangles.
skin has 64,068 triangles.
spine has 22,205 triangles.
spleen has 6,096 triangles.
stomach has 8,928 triangles.

There are 1,273,438 triangles in total
```

3.5 Visualise the virtual patient

First, apply the scaling factor from voxels to cm.

[21]:
```
for anatomy in json2gvxr.params["Samples"]:
    label = anatomy["Label"]
    gvxr.scaleNode(label, 3.3, 3.3, 3.3)
    gvxr.applyCurrentLocalTransformation(label)
```

Visualise the virtual patient

[22]:
```
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
    elif label == "muscles":
        opacity = 0.3
    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 = [17.757085561576094, -0.5941481609904308, 25.327896522760597, -0.
 ↪2824745178222656, -1.2793889045715332, 0.587073564529419, -0.
 ↪00450313545724278, 0.9990136450056702, 0.04417531962005856]
```

Output()

[23]:
```
fname = output_path + '/VHP_model.png'
if not os.path.isfile(fname):

    sleep(10)
    plot.fetch_screenshot() # Not sure why, but we need to do it twice to get␣
 ↪the right screenshot
    sleep(10)

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

3.6 Simulation with the default values

[24]:
```
# Backup the transformation matrix
global_matrix_backup = gvxr.getSceneTransformationMatrix()
```

[25]:
```
def getXRayImage():
    global total_energy_in_MeV

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

```
    # Apply the ROI
    xray_image = xray_image[:y_max]

    # Flat-field
#     xray_image /= total_energy_in_MeV

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

[26]:
```
xray_image = getXRayImage()
```

[27]:
```
# gvxr.enableArtefactFilteringOnCPU()
gvxr.enableArtefactFilteringOnGPU()
# gvxr.disableArtefactFiltering() # Spere inserts are missing with GPU␣
↪integration when a outer surface is used for the matrix
```

[28]:
```
# 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()
```

[29]:
```
xray_image = getXRayImage()
```

[30]:
```
gvxr.displayScene()
gvxr.useNegative()

gvxr.setZoom(1569.6787109375)
gvxr.setSceneRotationMatrix([-0.3190782964229584,  -0.15100032091140747, -0.
↪9356207251548767,  0.0,
                              0.002036974299699068, 0.987101674079895,   -0.
↪16000667214393616, 0.0,
                              0.9477221369743347,  -0.05296054854989052, -0.
↪31466084718704224, 0.0,
                              0.0,                  0.0,                  0.0, ␣
↪               1.0])

gvxr.setWindowBackGroundColour(0.5, 0.5, 0.5)

gvxr.displayScene()
```

[31]:
```
# gvxr.renderLoop()
```

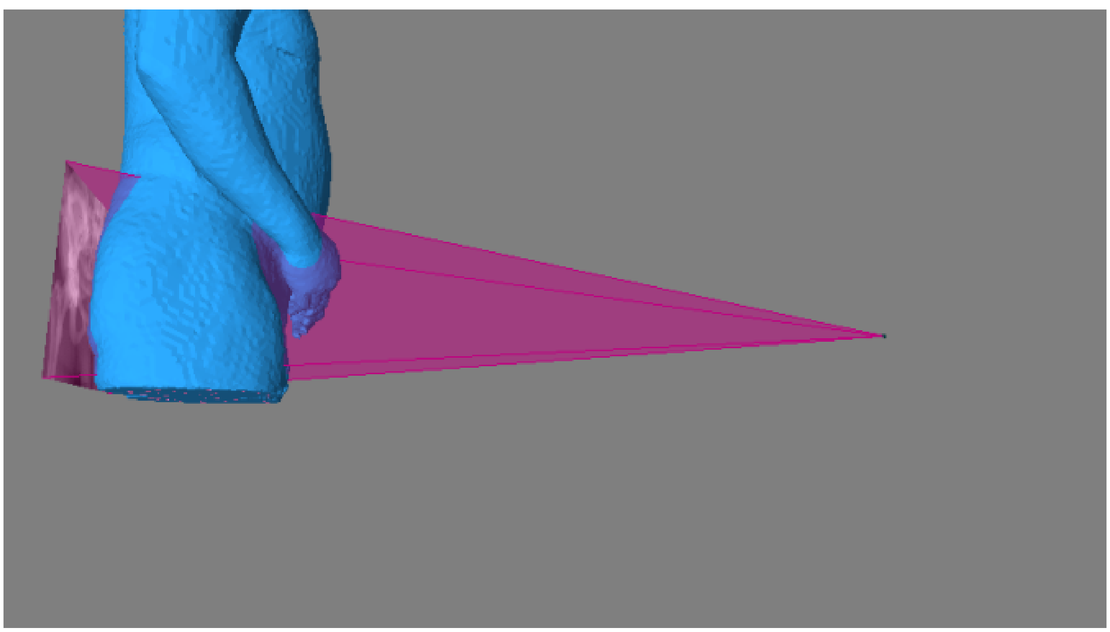
[32]:
```
# print(gvxr.getZoom())
# print(gvxr.getSceneRotationMatrix())
```

[33]:
```
screenshot = (255 * np.array(gvxr.takeScreenshot())).astype(np.uint8)
```

[34]:
```
fname = 'VHP/default-screenshot.png'
if True:#not os.path.isfile(fname):

    plt.imsave(fname, screenshot)
```

[35]:
```
plt.figure(figsize= (10,10))
plt.title("Screenshot")
plt.imshow(screenshot)
plt.axis('off')

plt.tight_layout()

plt.savefig(output_path + '/default-screenshot-beam-on-VHP.pdf')
plt.savefig(output_path + '/default-screenshot-beam-on-VHP.png', bbox_inches =␣
 ↪'tight')
```

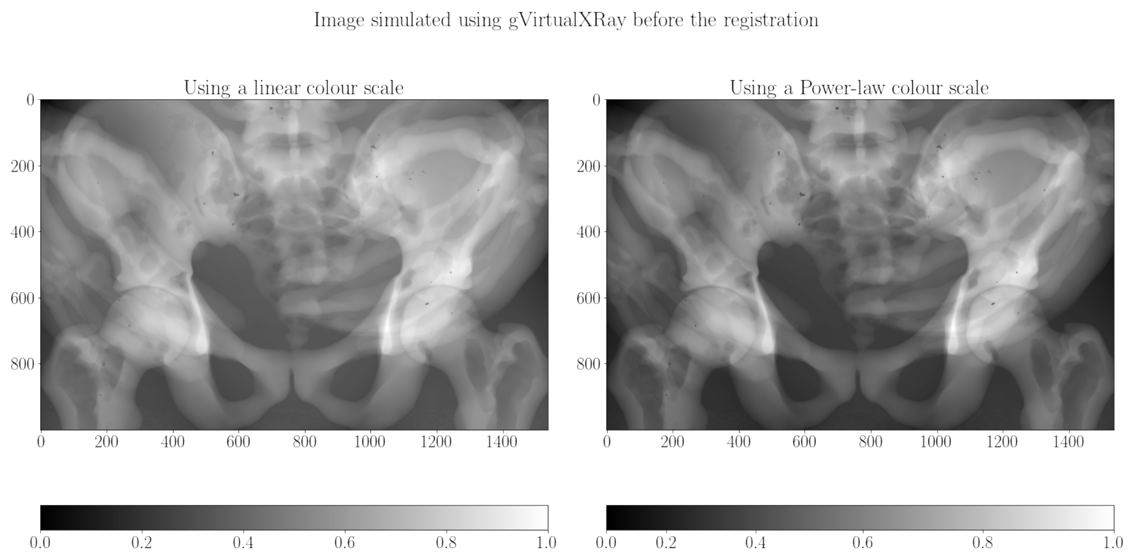
Screenshot



[36]:
```
def logImage(xray_image: np.array, min_val: float, max_val: float) -> np.array:

    log_epsilon = 1.0e-9
```

```python
    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
```

[37]:
```python
displayLinearPowerScales(1 - logImage(xray_image, xray_image.min(), xray_image.
 ↪max()),
                         "Image simulated using gVirtualXRay before the␣
 ↪registration",
                         output_path + "/gVirtualXRay-before_registration-VHP",
                         log=False)
```



4 Registration

4.1 Single-objective optimisation with CMA-ES

1st using CMA-ES with 5 different fitness functions, then using NSGA2 and NSGA3.

[38]:
```python
roi_ground_truth_min = ground_truth.min()
roi_ground_truth_max = ground_truth.max()
standardised_roi_ground_truth = standardisation(ground_truth)
```

```
imsave(output_path + '/standardised_roi_ground_truth-VHP.tif',␣
 ↪standardised_roi_ground_truth.astype(np.single))
```

```
/tmp/ipykernel_3809/1922444729.py:5: DeprecationWarning: <tifffile.imsave> is
deprecated. Use tifffile.imwrite
  imsave(output_path + '/standardised_roi_ground_truth-VHP.tif',
standardised_roi_ground_truth.astype(np.single))
```

[39]:

```
source_position = gvxr.getSourcePosition("cm")
detector_position = gvxr.getDetectorPosition("cm")

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

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

    # Position of the detector
    detector_position[0],
    detector_position[1],
    detector_position[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
]
```

[40]:

```
pos_offset = 20
angle_offset = 15

xl = [
```

```
            -angle_offset, -angle_offset,
            source_position[0] - pos_offset, source_position[1] - pos_offset,␣
↪source_position[2] - pos_offset,
            detector_position[0] - pos_offset, detector_position[1] -␣
↪pos_offset, detector_position[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[0] + pos_offset, source_position[1] + pos_offset,␣
↪source_position[2] + pos_offset,
            detector_position[0] + pos_offset, detector_position[1] +␣
↪pos_offset, detector_position[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
        ]
```

[41]:
```
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)
```

```python
    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]
```

[42]:
```python
def resetToDefaultParameters():
    json2gvxr.initDetector("notebook-4.json")
    json2gvxr.initSourceGeometry("notebook-4.json")
    source_position = gvxr.getSourcePosition("cm")
    detector_position = gvxr.getDetectorPosition("cm")

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

[43]:
```python
def updateXRayImage(x):



    # Backup the transformation matrix
    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
    gvxr.setSceneTransformationMatrix(matrix_backup)

    return xray_image #, screenshot
```

[44]:
```
def applyLogScaleAndNegative(image: np.array) -> np.array:
    temp = logImage(image, image.min(), image.max())
    return 1.0 - temp
```

[45]:
```
timeout_in_sec = 20 * 60 # 20 minutes
```

4.2 Define an objective function

[46]:
```
def objectiveFunction(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


    xray_image = updateXRayImage(x)
    corrected_xray_image = applyLogScaleAndNegative(xray_image)
    standardised_corrected_xray_image = standardisation(corrected_xray_image)

    if objective_function_string == "zncc":
        zncc = np.mean(standardised_roi_ground_truth *␣
 ↪standardised_corrected_xray_image)
        dzncc = (1.0 - zncc) / 2.0
        objective = dzncc
    elif objective_function_string == "mae":
        mae = np.mean(np.abs(standardised_roi_ground_truth -␣
 ↪standardised_corrected_xray_image))
        objective = mae
    elif objective_function_string == "rmse":
        rmse = math.sqrt(np.mean(np.square(standardised_roi_ground_truth -␣
 ↪standardised_corrected_xray_image)))
        objective = rmse
    elif objective_function_string == "ssim":
```

```
        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
        objective = dssim
    elif objective_function_string == "mape":
        # 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)
        objective = mape_value

    if best_fitness > objective:

        evolution_fitness.append([fitness_function_call_id, objective])

        row = [fitness_function_call_id]
        for i in x:
            row.append(i)
        evolution_parameters.append(row)

        best_fitness = objective

    fitness_function_call_id += 1

    return objective
```

[47]:

```
def optimiseWithCMAES(objective_function_str: str):

    global objective_function_string
    global best_fitness
    global best_fitness_id
    global fitness_function_call_id
    global evolution_fitness
    global evolution_parameters

    resetToDefaultParameters()

    objective_function_string = objective_function_str

    source_position = [0.0, 0.0, 0.0]
    detector_position = [0.0, 0.0, 0.0]
```

```
if os.path.exists(output_path + "/HIPS-" + objective_function_string + ".
↪dat") and \
    os.path.exists(output_path + "/HIPS_evolution-" +␣
↪objective_function_string + ".dat") and \
    os.path.exists(output_path + "/HIPS_evolution_parameters-" +␣
↪objective_function_string + ".dat"):

    temp = np.loadtxt(output_path + "/HIPS-" + objective_function_string +␣
↪".dat")

    sample_rotation_angle1 = temp[0]
    sample_rotation_angle2 = temp[1]
    source_position[0] = temp[2]
    source_position[1] = temp[3]
    source_position[2] = temp[4]
    detector_position[0] = temp[5]
    detector_position[1] = temp[6]
    detector_position[2] = temp[7]

    evolution_fitness = np.loadtxt(output_path + "/HIPS_evolution-" +␣
↪objective_function_string + ".dat")
    evolution_parameters = np.loadtxt(output_path + "/
↪HIPS_evolution_parameters-" + objective_function_string + ".dat")

# CMA-ES
else:

    opts = cma.CMAOptions()
    opts.set('tolfun', 1e-5)
    opts['tolx'] = 1e-5
    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) * 0.5)

    best_fitness = sys.float_info.max
    best_fitness_id = 0
    fitness_function_call_id = 0
    evolution_fitness = []
    evolution_parameters = []

    res = cma.fmin(objectiveFunction,
```

```python
                  x_init,
                  0.5,
                  opts,
                  restarts=0)

        # Save the best individual
        sample_rotation_angle1 = res[0][0]
        sample_rotation_angle2 = res[0][1]
        source_position[0] = res[0][2]
        source_position[1] = res[0][3]
        source_position[2] = res[0][4]
        detector_position[0] = res[0][5]
        detector_position[1] = res[0][6]
        detector_position[2] = res[0][7]

        # Save best parameters from the optimiser
        answer = np.array([sample_rotation_angle1, sample_rotation_angle2,␣
↪source_position[0], source_position[1], source_position[2],␣
↪detector_position[0], detector_position[1], detector_position[2]])
        answer = answer.reshape(1, answer.shape[0])
        np.savetxt(output_path + "/HIPS-" + objective_function_string + ".dat",
                   answer,
                   header='sample_rotation_angle1,␣
↪sample_rotation_angle2,source_pos_x,source_pos_y,source_pos_z,detector_pos_x,detector_pos_y

        # Save the list of zncc for plotting
        evolution_fitness = np.array(evolution_fitness)
        np.savetxt(output_path + "/HIPS_evolution-" + objective_function_string␣
↪+ ".dat",
                   evolution_fitness,
                   header='t,' + objective_function_string)

        # Save the list of parameters for plotting
        evolution_parameters = np.array(evolution_parameters)
        print(evolution_parameters.shape)

        np.savetxt(output_path + "/HIPS_evolution_parameters-" +␣
↪objective_function_string + ".dat",
                   evolution_parameters,
                  ␣
↪header='t,sample_rotation_angle1,sample_rotation_angle2,source_pos_x,source_pos_y,source_po

    return [sample_rotation_angle1, sample_rotation_angle2, source_position[0],␣
↪source_position[1], source_position[2], detector_position[0],␣
↪detector_position[1], detector_position[2]], \
        evolution_fitness, \
```

```
        evolution_parameters
```

4.3 Run the optimisation for each image comparison method

[48]:
```
objective_function_string = "zncc"
x_zncc, evolution_fitness_zncc, evolution_parameters_zncc =␣
↪optimiseWithCMAES("zncc")

resetToDefaultParameters()
xray_image_zncc = applyLogScaleAndNegative(updateXRayImage(x_zncc))
```

```
Set up the detector
        Detector position: [0.0, -30.5, -20.5, 'cm']
        Detector up vector: [0, -1, 0]
        Detector number of pixels: [1536, 1248]
        Pixel spacing: [0.026347682927083334, 0.026347683, 'cm']
Set up the beam
        Source position: [0.0, -30.5, 150.0, 'cm']
        Source shape: PointSource
Set up the detector
        Detector position: [0.0, -30.5, -20.5, 'cm']
        Detector up vector: [0, -1, 0]

Thu Sep 15 13:59:30 2022 ---- Initialise the renderer
Thu Sep 15 13:59:32 2022 ---- Initialise the renderer

        Detector number of pixels: [1536, 1248]
        Pixel spacing: [0.026347682927083334, 0.026347683, 'cm']
Set up the beam
        Source position: [0.0, -30.5, 150.0, 'cm']
        Source shape: PointSource
```

[49]:
```
objective_function_string = "mae"
x_mae, evolution_fitness_mae, evolution_parameters_mae =␣
↪optimiseWithCMAES("mae")

resetToDefaultParameters()
xray_image_mae = applyLogScaleAndNegative(updateXRayImage(x_mae))
```

```
Set up the detector
        Detector position: [0.0, -30.5, -20.5, 'cm']
        Detector up vector: [0, -1, 0]
        Detector number of pixels: [1536, 1248]
        Pixel spacing: [0.026347682927083334, 0.026347683, 'cm']
Set up the beam
        Source position: [0.0, -30.5, 150.0, 'cm']
        Source shape: PointSource
Set up the detector
        Detector position: [0.0, -30.5, -20.5, 'cm']
```

```
        Detector up vector: [0, -1, 0]
        Detector number of pixels: [1536, 1248]
        Pixel spacing: [0.026347682927083334, 0.026347683, 'cm']
Set up the beam
        Source position: [0.0, -30.5, 150.0, 'cm']
        Source shape: PointSource

Thu Sep 15 13:59:33 2022 ---- Initialise the renderer
Thu Sep 15 13:59:35 2022 ---- Initialise the renderer
```

[50]:
```
objective_function_string = "rmse"
x_rmse, evolution_fitness_rmse, evolution_parameters_rmse =␣
 ↪optimiseWithCMAES("rmse")

resetToDefaultParameters()
xray_image_rmse = applyLogScaleAndNegative(updateXRayImage(x_rmse))
```

```
Set up the detector
        Detector position: [0.0, -30.5, -20.5, 'cm']
        Detector up vector: [0, -1, 0]
        Detector number of pixels: [1536, 1248]
        Pixel spacing: [0.026347682927083334, 0.026347683, 'cm']
Set up the beam
        Source position: [0.0, -30.5, 150.0, 'cm']
        Source shape: PointSource
Set up the detector
        Detector position: [0.0, -30.5, -20.5, 'cm']
        Detector up vector: [0, -1, 0]
        Detector number of pixels: [1536, 1248]
        Pixel spacing: [0.026347682927083334, 0.026347683, 'cm']
Set up the beam
        Source position: [0.0, -30.5, 150.0, 'cm']
        Source shape: PointSource

Thu Sep 15 13:59:36 2022 ---- Initialise the renderer
Thu Sep 15 13:59:38 2022 ---- Initialise the renderer
```

[51]:
```
objective_function_string = "ssim"
x_ssim, evolution_fitness_ssim, evolution_parameters_ssim =␣
 ↪optimiseWithCMAES("ssim")

resetToDefaultParameters()
xray_image_ssim = applyLogScaleAndNegative(updateXRayImage(x_ssim))
```

```
Set up the detector
        Detector position: [0.0, -30.5, -20.5, 'cm']
        Detector up vector: [0, -1, 0]
        Detector number of pixels: [1536, 1248]
        Pixel spacing: [0.026347682927083334, 0.026347683, 'cm']
```

```
Thu Sep 15 13:59:39 2022 ---- Initialise the renderer
Thu Sep 15 13:59:40 2022 ----
```

```
Set up the beam
        Source position: [0.0, -30.5, 150.0, 'cm']
        Source shape: PointSource
Set up the detector
        Detector position: [0.0, -30.5, -20.5, 'cm']
        Detector up vector: [0, -1, 0]
        Detector number of pixels: [1536, 1248]
        Pixel spacing: [0.026347682927083334, 0.026347683, 'cm']
Set up the beam
        Source position: [0.0, -30.5, 150.0, 'cm']
        Source shape: PointSource
```

```
Initialise the renderer
```

[52]:

```
objective_function_string = "mape"
x_mape, evolution_fitness_mape, evolution_parameters_mape =␣
 ↪optimiseWithCMAES("mape")

resetToDefaultParameters()
xray_image_mape = applyLogScaleAndNegative(updateXRayImage(x_mape))
```

```
Set up the detector
        Detector position: [0.0, -30.5, -20.5, 'cm']
        Detector up vector: [0, -1, 0]
        Detector number of pixels: [1536, 1248]
        Pixel spacing: [0.026347682927083334, 0.026347683, 'cm']
Set up the beam
        Source position: [0.0, -30.5, 150.0, 'cm']
        Source shape: PointSource
```

```
Thu Sep 15 13:59:42 2022 ---- Initialise the renderer
```

```
Set up the detector
        Detector position: [0.0, -30.5, -20.5, 'cm']
        Detector up vector: [0, -1, 0]
        Detector number of pixels: [1536, 1248]
        Pixel spacing: [0.026347682927083334, 0.026347683, 'cm']
Set up the beam
        Source position: [0.0, -30.5, 150.0, 'cm']
        Source shape: PointSource
```

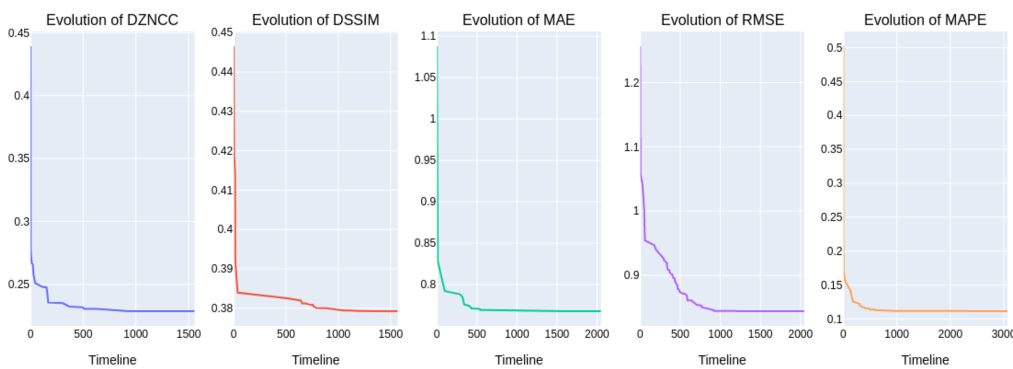
```
Thu Sep 15 13:59:43 2022 ---- Initialise the renderer
```

[53]:

```
fig = make_subplots(rows=1, cols=5, start_cell="bottom-left",
                    subplot_titles=("Evolution of DZNCC", "Evolution of DSSIM",␣
 ↪"Evolution of MAE", "Evolution of RMSE", "Evolution of MAPE"))
```

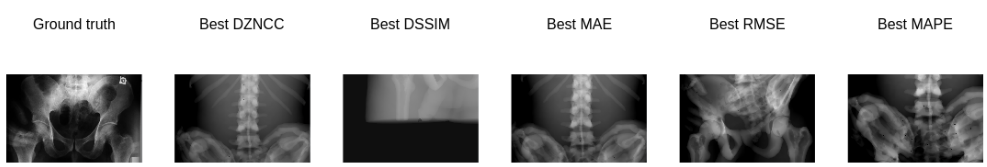
```python
# fig.add_trace(go.Scatter(x=evolution_fitness_zncc[:,0], y=1.0 - (2.0 * evolution_fitness_zncc[:,1])),
fig.add_trace(go.Scatter(x=evolution_fitness_zncc[:,0], y=evolution_fitness_zncc[:,1]),
              row=1, col=1)

# fig.add_trace(go.Scatter(x=evolution_fitness_ssim[:,0], y=1.0 - (2.0 * evolution_fitness_ssim[:,1])),
fig.add_trace(go.Scatter(x=evolution_fitness_ssim[:,0], y=evolution_fitness_ssim[:,1]),
              row=1, col=2)

fig.add_trace(go.Scatter(x=evolution_fitness_mae[:,0], y=evolution_fitness_mae[:,1]),
              row=1, col=3)

fig.add_trace(go.Scatter(x=evolution_fitness_rmse[:,0], y=evolution_fitness_rmse[:,1]),
              row=1, col=4)

fig.add_trace(go.Scatter(x=evolution_fitness_mape[:,0], y=evolution_fitness_mape[:,1]),
              row=1, col=5)

# Update xaxis properties
fig.update_xaxes(title_text="Timeline", row=1, col=1)
fig.update_xaxes(title_text="Timeline", row=1, col=2)
fig.update_xaxes(title_text="Timeline", row=1, col=3)
fig.update_xaxes(title_text="Timeline", row=1, col=4)
fig.update_xaxes(title_text="Timeline", row=1, col=5)

# Update yaxis properties
# fig.update_yaxes(title_text="Ojective function: ZNCC", row=1, col=1)
# fig.update_yaxes(title_text="Ojective function: MAE", row=1, col=2)
# fig.update_yaxes(title_text="Ojective function: RMSE", row=1, col=3)
# fig.update_yaxes(title_text="Ojective function: SSIM", row=1, col=4)
# fig.update_yaxes(title_text="Ojective function: MAPE", row=1, col=5)

# fig.update_yaxes(title_text="yaxis 2 title", range=[40, 80], row=1, col=2)
# fig.update_yaxes(title_text="yaxis 3 title", showgrid=False, row=2, col=1)
# fig.update_yaxes(title_text="yaxis 4 title", row=2, col=2)

fig.update_layout(showlegend=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=500,
    width=1000
)

fig.write_image(output_path + "/HIPS_evolution-objectives.pdf",␣
 ↪engine="kaleido")
fig.write_image(output_path + "/HIPS_evolution-objectives.png",␣
 ↪engine="kaleido")

fig.show()
```



[54]:
```
fig = make_subplots(rows=1, cols=6,
                    start_cell="bottom-left",
                    subplot_titles=("Ground truth", "Best DZNCC", "Best DSSIM",␣
 ↪"Best MAE", "Best RMSE", "Best MAPE"))

cmaes_img_set = [standardised_roi_ground_truth,
                 standardisation(xray_image_zncc),
                 standardisation(xray_image_ssim),
                 standardisation(xray_image_mae),
                 standardisation(xray_image_rmse),
                 standardisation(xray_image_mape)]

for n, image in enumerate(cmaes_img_set):
```

```python
    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(output_path + "/HIPS_cmaes-objectives.pdf", engine="kaleido")
fig.write_image(output_path + "/HIPS_cmaes-objectives.png", engine="kaleido")

fig.show()
```



[55]:
```python
cmaes_x_set   =  [x_zncc, x_mae, x_rmse, x_ssim, x_mape]
cmaes_img_set = [xray_image_zncc, xray_image_mae, xray_image_rmse, xray_image_ssim, xray_image_mape]

temp_res_cmaes = []
```

```
for n, [x, image] in enumerate(zip(cmaes_x_set, cmaes_img_set)):

    row = copy.deepcopy(x)

    xray_image = updateXRayImage(x)
    corrected_xray_image = applyLogScaleAndNegative(xray_image)
    standardised_corrected_xray_image = standardisation(corrected_xray_image)

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

    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())
    row.append((1.0 - ssim_value) / 2.0)

    # 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)

    temp_res_cmaes.append(row)
```

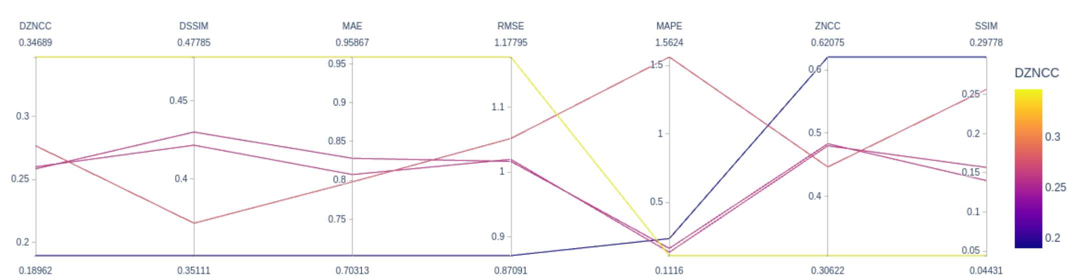
[56]:
```
df_cmaes = pd.DataFrame(data=temp_res_cmaes,
                  columns=["sample_rotation_angle1", "sample_rotation_angle2",␣
 ↪"src_pos_x", "src_pos_y", "src_pos_z", "det_pos_x", "det_pos_y",␣
 ↪"det_pos_z", "DZNCC", "MAE", "RMSE", "DSSIM", "MAPE"])

df_cmaes["ZNCC"] = 1.0 - (df_cmaes["DZNCC"] * 2.0)
df_cmaes["SSIM"] = 1.0 - (df_cmaes["DSSIM"] * 2.0)

df_cmaes["Optimiser"] = "CMA-ES"
```

```python
df_cmaes["Optimiser_code"] = 1
df_cmaes.to_csv(output_path + "/hips-optimiser-cmaes.csv")
```

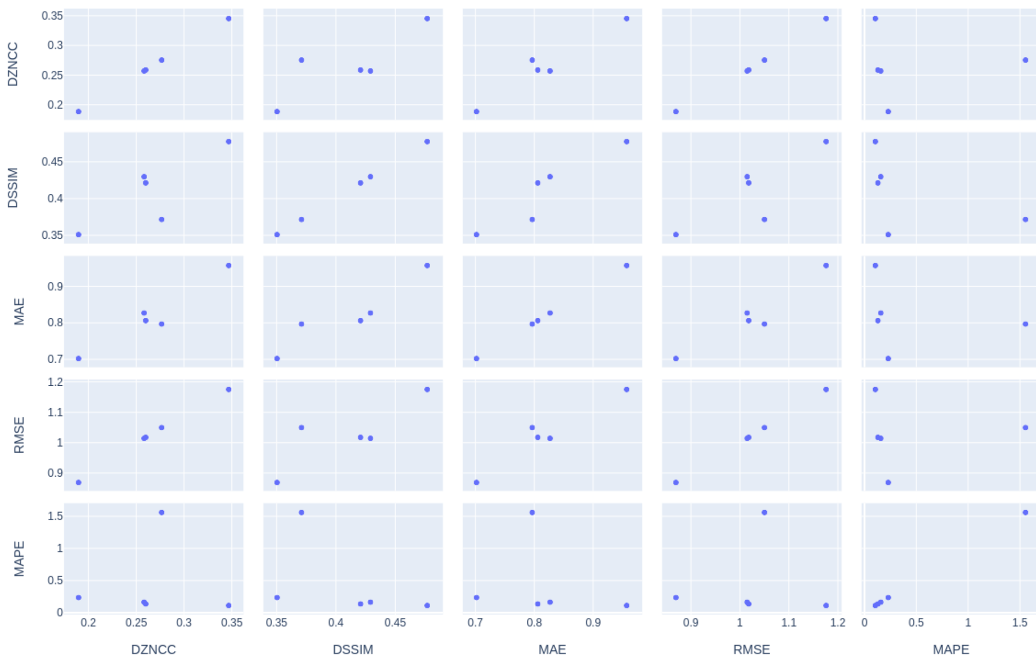
[57]:
```python
display(df_cmaes)
```

```
   sample_rotation_angle1  sample_rotation_angle2  src_pos_x  src_pos_y  \
0               -4.999974                2.942652   7.521376 -50.499428
1               -4.999836                4.999441  16.973382 -50.067071
2               -4.999989                4.999922 -19.999830 -50.497748
3                4.999735                4.999826 -19.996561 -50.499921
4               -4.999659               -4.995986 -10.132157 -50.433492

    src_pos_z  det_pos_x  det_pos_y  det_pos_z     DZNCC       MAE      RMSE  \
0  130.000302   1.592841 -10.507583 -11.218151  0.258320  0.828300  1.016504
1  130.000067  -0.630155 -10.500383 -15.360448  0.259959  0.807418  1.019724
2  130.000344   1.091760 -30.833317  -7.976479  0.189623  0.703128  0.870914
3  130.000954  13.155220 -50.499978 -12.451590  0.276675  0.798011  1.051998
4  130.041014   7.996906 -12.703856 -40.499699  0.346892  0.958673  1.177950

      DSSIM      MAPE      ZNCC      SSIM Optimiser  Optimiser_code
0  0.429956  0.164314  0.483360  0.140088    CMA-ES               1
1  0.421568  0.136231  0.480081  0.156864    CMA-ES               1
2  0.351112  0.236018  0.620755  0.297775    CMA-ES               1
3  0.371779  1.562402  0.446650  0.256442    CMA-ES               1
4  0.477846  0.111581  0.306217  0.044308    CMA-ES               1
```

[58]:
```python
fig = px.parallel_coordinates(df_cmaes[["DZNCC", "DSSIM", "MAE", "RMSE",␣
 ↪"MAPE", "ZNCC", "SSIM"]], color="DZNCC")
fig.show()

fig.write_image(output_path + "/HIPS-cmaes-parallel_coordinates.pdf",␣
 ↪engine="kaleido")
fig.write_image(output_path + "/HIPS-cmaes-parallel_coordinates.png",␣
 ↪engine="kaleido")
```



[59]:
```
fig = px.scatter_matrix(df_cmaes[["DZNCC", "DSSIM", "MAE", "RMSE", "MAPE"]])

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

fig.show()

fig.write_image(output_path + "/HIPS-cmaes-scatter_matrix.pdf",␣
 ↪engine="kaleido")
fig.write_image(output_path + "/HIPS-cmaes-scatter_matrix.png",␣
 ↪engine="kaleido")
```



4.4 Multi-objective optimisation with NSGA-II

[60]:
```
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)
        standardised_corrected_xray_image =␣
↪standardisation(corrected_xray_image)

        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)

        objectives.append(row)

    return objectives
```

[61]:
```
from pymoo.core.problem import Problem
```

```
class MyMultiObjectiveProblem(Problem):

    def __init__(self):
        super().__init__(n_var=len(x_init),
                         n_obj=5,
                         n_constr=0,
                         xl=xl,
                         xu=xu)

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

[62]:

```
from pymoo.algorithms.moo.nsga2 import NSGA2
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", "01:00:00")

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

if os.path.exists(output_path + "/VHP-res-nsga2-X.dat") and os.path.
 ↪exists(output_path + "/VHP-res-nsga2-F.dat"):

    res_nsga2_X = np.loadtxt(output_path + "/VHP-res-nsga2-X.dat")
    res_nsga2_F = np.loadtxt(output_path + "/VHP-res-nsga2-F.dat")

else:

    algorithm = NSGA2(
        pop_size=pop_size,
#         n_offsprings=int(pop_size*0.05),
        eliminate_duplicates=True
    )

    res_nsga2 = minimize(problem,
                   algorithm,
                   termination,
```

```
                    seed=1,
                    save_history=True,
                    verbose=True)

    res_nsga2_X = res_nsga2.X
    res_nsga2_F = res_nsga2.F

    np.savetxt(output_path + "/VHP-res-nsga2-X.dat", res_nsga2_X)
    np.savetxt(output_path + "/VHP-res-nsga2-F.dat", res_nsga2_F)
```

```
Set up the detector
        Detector position: [0.0, -30.5, -20.5, 'cm']
        Detector up vector: [0, -1, 0]
        Detector number of pixels: [1536, 1248]
        Pixel spacing: [0.026347682927083334, 0.026347683, 'cm']
Set up the beam
        Source position: [0.0, -30.5, 150.0, 'cm']
        Source shape: PointSource
```

```
Thu Sep 15 13:59:55 2022 ---- Initialise the renderer
```

[63]:

```
best_dzncc_id = np.argmin(res_nsga2_F[:,0])
best_mae_id = np.argmin(res_nsga2_F[:,1])
best_rmse_id = np.argmin(res_nsga2_F[:,2])
best_dssim_id = np.argmin(res_nsga2_F[:,3])
best_mape_id = np.argmin(res_nsga2_F[:,4])

print("Lowest DZNCC:", res_nsga2_F[:,0].min(), best_dzncc_id,␣
 ↪res_nsga2_X[best_dzncc_id])
print("Lowest DSSIM:", res_nsga2_F[:,3].min(), best_dssim_id,␣
 ↪res_nsga2_X[best_dssim_id])
print("Lowest MAE:",   res_nsga2_F[:,1].min(), best_mae_id,  ␣
 ↪res_nsga2_X[best_mae_id])
print("Lowest RMSE:",  res_nsga2_F[:,2].min(), best_rmse_id, ␣
 ↪res_nsga2_X[best_rmse_id])
print("Lowest MAPE:",  res_nsga2_F[:,4].min(), best_mape_id, ␣
 ↪res_nsga2_X[best_mape_id])
```

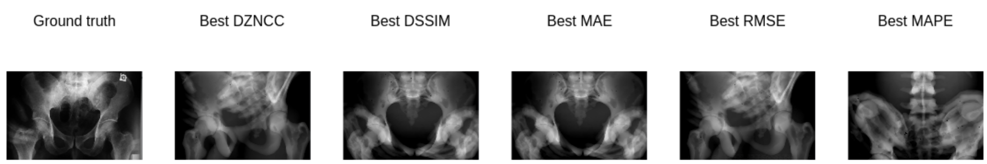
```
Lowest DZNCC: 0.185465780596681 83 [-11.64023327  13.48139382 -17.14343476
-29.99029926 158.16326652
   1.88062885 -32.45039691 -15.35413903]
Lowest DSSIM: 0.3096520655444412 94 [ 1.48943256e+01 -1.65730637e+00
7.90603291e+00 -1.08250512e+01
  1.30818746e+02  1.02683377e-01 -3.07267514e+01 -2.64178174e+01]
Lowest MAE: 0.6892712813113362 94 [ 1.48943256e+01 -1.65730637e+00
7.90603291e+00 -1.08250512e+01
  1.30818746e+02  1.02683377e-01 -3.07267514e+01 -2.64178174e+01]
Lowest RMSE: 0.8613147638272104 83 [-11.64023327  13.48139382 -17.14343476
```

```
-29.99029926 158.16326652
   1.88062885 -32.45039691 -15.35413903]
Lowest MAPE: 0.10874328879996342 106 [-13.8488119   -9.19608541 -14.85216699
-26.29165615 131.39221507
   6.66066609 -20.64174421 -27.26753763]
```

[64]:
```
xray_image_dzncc_nsga2 =␣
 ↪applyLogScaleAndNegative(updateXRayImage(res_nsga2_X[best_dzncc_id]))
xray_image_mae_nsga2   =␣
 ↪applyLogScaleAndNegative(updateXRayImage(res_nsga2_X[best_mae_id]))
xray_image_rmse_nsga2  =␣
 ↪applyLogScaleAndNegative(updateXRayImage(res_nsga2_X[best_rmse_id]))
xray_image_dssim_nsga2 =␣
 ↪applyLogScaleAndNegative(updateXRayImage(res_nsga2_X[best_dssim_id]))
xray_image_mape_nsga2  =␣
 ↪applyLogScaleAndNegative(updateXRayImage(res_nsga2_X[best_mape_id]))
```

[65]:
```
fig = make_subplots(rows=1, cols=6,
                    start_cell="bottom-left",
                    subplot_titles=("Ground truth", "Best DZNCC", "Best DSSIM",␣
 ↪"Best MAE", "Best RMSE", "Best MAPE"))

nsga2_img_set = [standardised_roi_ground_truth,
                 standardisation(xray_image_dzncc_nsga2),
                 standardisation(xray_image_dssim_nsga2),
                 standardisation(xray_image_mae_nsga2),
                 standardisation(xray_image_rmse_nsga2),
                 standardisation(xray_image_mape_nsga2)]

for n, image in enumerate(nsga2_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(output_path + "/HIPS-NSGA2-objectives.pdf", engine="kaleido")
fig.write_image(output_path + "/HIPS-NSGA2-objectives.png", engine="kaleido")

fig.show()
```



[66]:
```
df_nsga2 = pd.DataFrame(data=np.append(res_nsga2_X, res_nsga2_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"])

df_nsga2["ZNCC"] = 1.0 - (df_nsga2["DZNCC"] * 2.0)
df_nsga2["SSIM"] = 1.0 - (df_nsga2["DSSIM"] * 2.0)

df_nsga2["Optimiser"] = "NSGA-II"
df_nsga2["Optimiser_code"] = 2
df_nsga2.to_csv(output_path + "/hips-optimiser-nsga2.csv")
```

[67]:
```
display(df_nsga2)
```

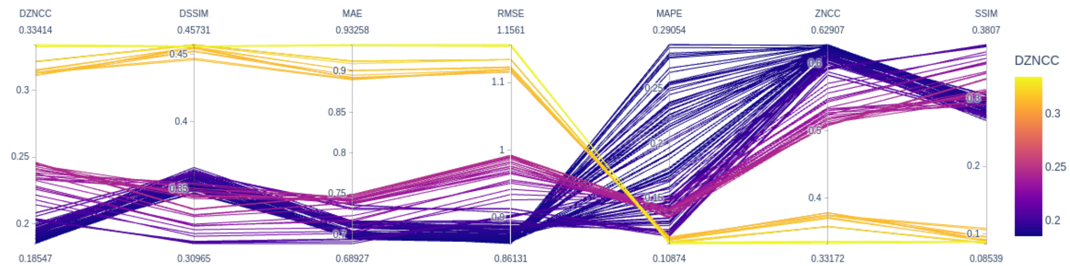
```
     sample_rotation_angle1  sample_rotation_angle2  src_pos_x  src_pos_y  \
0                -12.214140               12.965309 -15.733508 -29.994005
1                 -6.679212               12.749666 -10.455286 -40.386098
2                 -5.611060               12.076054  -6.852205 -44.679958
3                 -8.642622               12.053705  -7.005473 -40.159503
4                 -5.789463               10.842973  -6.249077 -35.802147
..                      …                       …          …          …
137              -14.713646               -8.434451 -14.834482 -24.899000
138               14.972113                0.982936   7.086566 -10.930021
139               -8.540964               13.604303 -12.021694 -29.894818
```

```
140              -12.162269             11.048147 -15.847851 -26.619016
141              -13.972585            -13.621481 -14.614618 -25.469657

      src_pos_z  det_pos_x  det_pos_y  det_pos_z     DZNCC       MAE  \
0    155.349984   1.888944 -31.440861 -15.541124  0.186939  0.699252
1    141.666040   0.068312 -30.933368  -6.245937  0.193168  0.710965
2    142.110323  -0.210794 -30.803209  -5.446876  0.195350  0.714085
3    163.978172  -0.084172 -30.944482  -6.228267  0.195167  0.714214
4    142.119278   0.140307 -30.287829  -8.241393  0.205830  0.735937
..          …          …          …          …         …         …
137  131.417358   6.661959 -19.806614 -26.979297  0.321453  0.909179
138  130.922329   0.611846 -29.802588 -25.669618  0.225963  0.723332
139  140.864281   0.155587 -31.697596  -6.244817  0.192468  0.708184
140  142.577841   1.852858 -32.877586 -14.126720  0.185974  0.698526
141  131.534365   6.660531 -18.903100 -27.088419  0.313319  0.890422

         RMSE     DSSIM      MAPE      ZNCC      SSIM Optimiser  \
0    0.864728  0.351507  0.281040  0.626123  0.296987   NSGA-II
1    0.879018  0.359267  0.147263  0.613664  0.281467   NSGA-II
2    0.883967  0.356973  0.145874  0.609301  0.286053   NSGA-II
3    0.883555  0.355683  0.146494  0.609666  0.288634   NSGA-II
4    0.907371  0.362099  0.115170  0.588339  0.275803   NSGA-II
..        …         …         …         …         …         …
137  1.133936  0.456594  0.110550  0.357095  0.086813   NSGA-II
138  0.950711  0.324019  0.143241  0.548074  0.351962   NSGA-II
139  0.877424  0.363057  0.143966  0.615064  0.273886   NSGA-II
140  0.862493  0.350632  0.290234  0.628053  0.298737   NSGA-II
141  1.119498  0.447150  0.112160  0.373362  0.105700   NSGA-II

     Optimiser_code
0                 2
1                 2
2                 2
3                 2
4                 2
..              …
137               2
138               2
139               2
140               2
141               2

[142 rows x 17 columns]
```

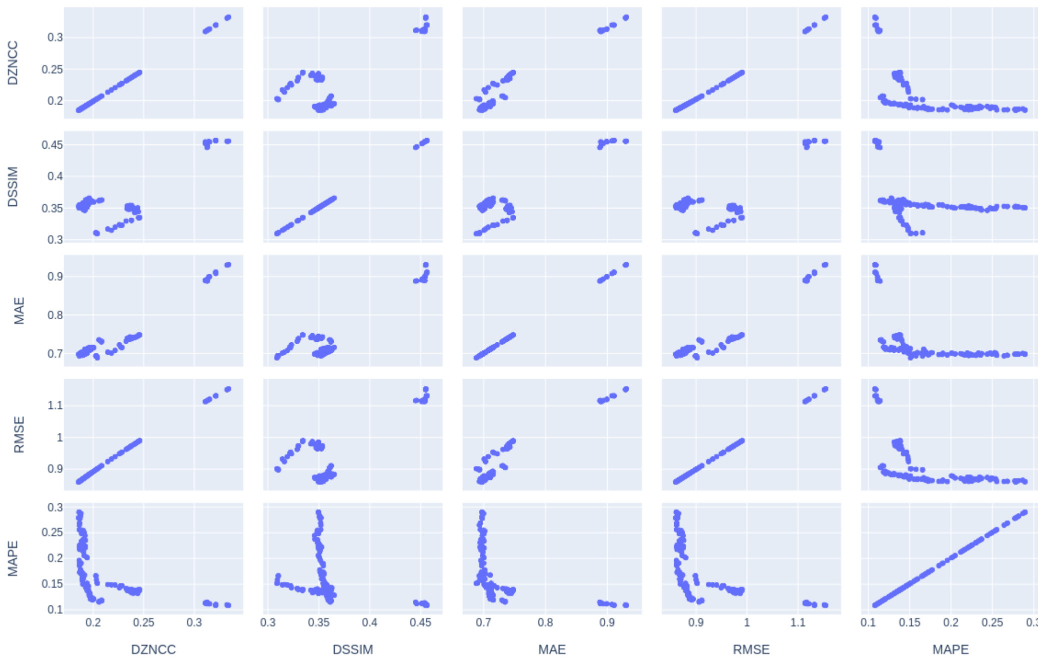
[68]:

```
fig = px.parallel_coordinates(df_nsga2[["DZNCC", "DSSIM", "MAE", "RMSE",
    "MAPE", "ZNCC", "SSIM"]], color="DZNCC")
fig.show()
```

```
fig.write_image(output_path + "/HIPS-NSGA2-parallel_coordinates.pdf",␣
↪engine="kaleido")
fig.write_image(output_path + "/HIPS-NSGA2-parallel_coordinates.png",␣
↪engine="kaleido")
```



[69]:
```
fig = px.scatter_matrix(df_nsga2[["DZNCC", "DSSIM", "MAE", "RMSE", "MAPE"]])

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

fig.show()

fig.write_image(output_path + "/HIPS-NSGA2-scatter_matrix.pdf",␣
↪engine="kaleido")
fig.write_image(output_path + "/HIPS-NSGA2-scatter_matrix.png",␣
↪engine="kaleido")
```



[70]:
```
for i, x in enumerate(res_nsga2_X):
    img = applyLogScaleAndNegative(updateXRayImage(x))
    mpimg.imsave(output_path + "/NSGA2/img_" + str(i) + ".png", (255 * (img -␣
↪img.min()) / (img.max() - img.min())).astype(np.uint8), cmap="gray")
```

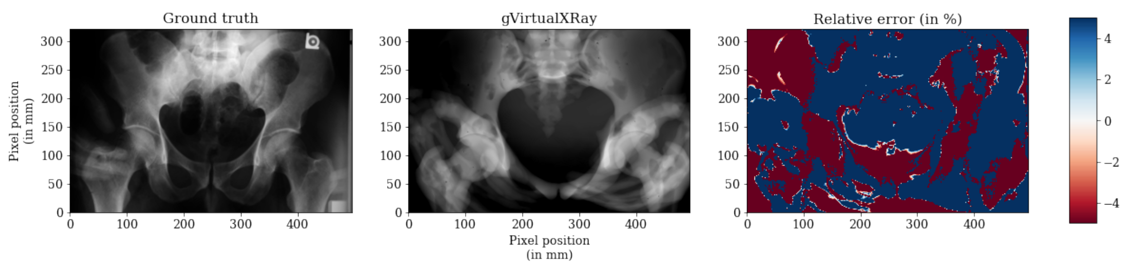
[71]:
```
best_nsga2_id = 109
x = res_nsga2_X[best_nsga2_id]

nsga2_raw_x_ray_image = xray_image_dzncc_nsga2 =␣
↪applyLogScaleAndNegative(updateXRayImage(x))
nsga2_raw_x_ray_image = np.array(nsga2_raw_x_ray_image)

mpimg.imsave(output_path + "/NSGA2-img_" + str(best_nsga2_id) + ".png", (255 *␣
↪(nsga2_raw_x_ray_image - nsga2_raw_x_ray_image.min()) /␣
↪(nsga2_raw_x_ray_image.max() - nsga2_raw_x_ray_image.min())).astype(np.
↪uint8), cmap="gray")
```

[72]:
```
source_position = gvxr.getSourcePosition("mm")
detector_position = gvxr.getDetectorPosition("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)

detector_size = np.array(gvxr.getDetectorSize("mm"))
number_of_pixels = gvxr.getDetectorNumberOfPixels()

detector_element_spacing = (detector_size / number_of_pixels)
spacing = source_imager_distance * detector_element_spacing /␣
 ↪source_object_distance

print("spacing in the object plane (in mm):", spacing)
```

```
spacing in the object plane (in mm): [0.32236026 0.32236027]
```

[73]:
```
matplotlib.rc('text', usetex=False)

font = {'size'   : 12.5
       }
matplotlib.rc('font', **font)

vmin = standardised_roi_ground_truth.min()
vmax = standardised_roi_ground_truth.max()

fullCompareImages(standardised_roi_ground_truth,
                  standardisation(nsga2_raw_x_ray_image),
                  "gVirtualXRay",
                  output_path + "/full_comparison_VHP-NSGA2", spacing,␣
 ↪vmin=vmin, vmax=vmax, log=False)
```



[74]:
```
runtimes = []

resetToDefaultParameters()
setTransformations(res_nsga2_X[best_nsga2_id])

```

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

    temp = gvxr.computeXRayImage()

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

```
Set up the detector
        Detector position: [0.0, -30.5, -20.5, 'cm']
        Detector up vector: [0, -1, 0]
        Detector number of pixels: [1536, 1248]
        Pixel spacing: [0.026347682927083334, 0.026347683, 'cm']
Set up the beam
        Source position: [0.0, -30.5, 150.0, 'cm']
        Source shape: PointSource

Thu Sep 15 14:01:31 2022 ---- Initialise the renderer
```

[75]:
```
runtime_avg = round(np.mean(runtimes))
runtime_std = round(np.std(runtimes))
```

[76]:
```
ZNCC = 1.0 - (2.0 * res_nsga2_F[best_nsga2_id,0])
SSIM = 1.0 - (2.0 * res_nsga2_F[best_nsga2_id,3])
MAPE = res_nsga2_F[best_nsga2_id,4]

print("Registration VHP & Real image & " +
      "{0:0.2f}".format(100 * MAPE) + "\\%    &    " +
      "{0:0.2f}".format(100 * ZNCC) + "\\%    &    " +
      "{0:0.2f}".format(SSIM) + "    &    $" +
      str(nsga2_raw_x_ray_image.shape[1]) + " \\times " +␣
 ↪str(nsga2_raw_x_ray_image.shape[0]) + "$    &    " +
      str(number_of_triangles) + "    &    " +
      "$" + str(runtime_avg) + " \\pm " + str(runtime_std) + "$ \\\\")
```

```
Registration VHP & Real image & 16.64\%    &    59.50\%    &    0.38    &
$1536 \times 1000$    &    1273438    &    $360 \pm 17$ \\
```

5 Compute the magnification

$magnification = \frac{SID}{SOD}$ with SID the source to imager distance and SOD the source to object distance.

[77]:
```
source_position = gvxr.getSourcePosition("mm")
detector_position = gvxr.getDetectorPosition("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
```

[78]:
```
print("SID:", source_imager_distance, "mm")
print("SOD:", source_object_distance, "mm")
print("magnification:", magnification)
```

```
SID: 1611.6312885195107 mm
SOD: 1317.2450758504572 mm
magnification: 1.2234862882132909
```

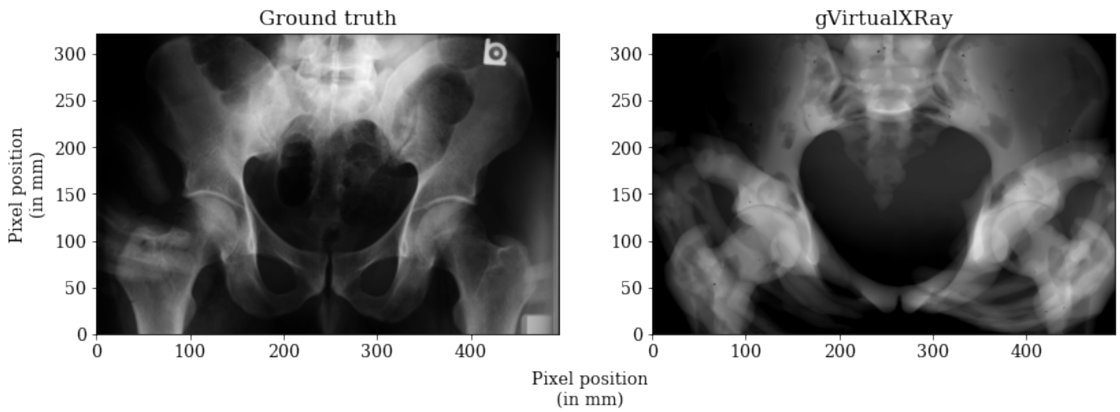
6 Compute the pixel size in the object plane

[79]:
```
detector_size = np.array(gvxr.getDetectorSize("mm"))
number_of_pixels = gvxr.getDetectorNumberOfPixels()

detector_element_spacing = (detector_size / number_of_pixels)
spacing = source_imager_distance * detector_element_spacing /␣
 ↪source_object_distance

print("spacing in the object plane (in mm):", spacing)
```

```
spacing in the object plane (in mm): [0.32236026 0.32236027]
```

[80]:
```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(20 * 2 / 3, 20))

vmin = standardised_roi_ground_truth.min()
vmax = standardised_roi_ground_truth.max()

im1 = axes.flat[0].imshow(standardised_roi_ground_truth, cmap="gray",␣
 ↪vmin=vmin, vmax=vmax,
                          extent=[0,(standardised_roi_ground_truth.
 ↪shape[1]-1)*spacing[0],0,(standardised_roi_ground_truth.
 ↪shape[0]-1)*spacing[1]])
axes.flat[0].set_title("Ground truth")
# axes.flat[0].set_xticks([])
# axes.flat[0].set_yticks([])

im2 = axes.flat[1].imshow(standardisation(nsga2_raw_x_ray_image), cmap="gray",␣
 ↪vmin=vmin, vmax=vmax,
```

```
                                    extent=[0,(nsga2_raw_x_ray_image.
↪shape[1]-1)*spacing[0],0,(nsga2_raw_x_ray_image.shape[0]-1)*spacing[1]])
axes.flat[1].set_title("gVirtualXRay")
# axes.flat[1].set_xticks([])
# axes.flat[1].set_yticks([])

fig.text(0.5, 0.39, 'Pixel position\n(in mm)', ha='center')

axes.flat[0].set_ylabel("Pixel position\n(in mm)")

plt.savefig(output_path + 'full_comparison_VHP-NSGA2.pdf')
plt.savefig(output_path + 'full_comparison_VHP-NSGA2.png', bbox_inches =␣
↪'tight')
```



6.1 Multi-objective optimisation with NSGA-3

[81]:
```
from pymoo.algorithms.moo.nsga3 import NSGA3
from pymoo.factory import get_reference_directions

resetToDefaultParameters()

if os.path.exists(output_path + "/VHP-res-nsga3-X.dat") and os.path.
↪exists(output_path + "/VHP-res-nsga3-F.dat"):

    res_nsga3_X = np.loadtxt(output_path + "/VHP-res-nsga3-X.dat")
    res_nsga3_F = np.loadtxt(output_path + "/VHP-res-nsga3-F.dat")

else:
    n_objs = 5
    n_partitions = 6
```

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

    problem = MyMultiObjectiveProblem()

    pop_size = 210 #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(output_path + "/VHP-res-nsga3-X.dat", res_nsga3_X)
    np.savetxt(output_path + "/VHP-res-nsga3-F.dat", res_nsga3_F)
```

```
Set up the detector
        Detector position: [0.0, -30.5, -20.5, 'cm']
        Detector up vector: [0, -1, 0]
        Detector number of pixels: [1536, 1248]
        Pixel spacing: [0.026347682927083334, 0.026347683, 'cm']
Set up the beam
        Source position: [0.0, -30.5, 150.0, 'cm']
        Source shape: PointSource

Thu Sep 15 14:01:42 2022 ---- Initialise the renderer
```

[82]:
```
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])

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])
```

```
Lowest DZNCC: 0.1838638014108661 2 [-14.88682174   9.8471118  -19.22286359
-23.74979691 133.69563762
   1.59534161 -31.75989139  -8.40887855]
Lowest DSSIM: 0.3165490218991922 6 [ 14.87391702 -14.45643058  -6.80768805
-12.47571275 133.60351888
   1.54743178 -29.71054245 -27.26163087]
Lowest MAE: 0.6909638079923718 2 [-14.88682174   9.8471118  -19.22286359
-23.74979691 133.69563762
   1.59534161 -31.75989139  -8.40887855]
Lowest RMSE: 0.8575868048599764 2 [-14.88682174   9.8471118  -19.22286359
-23.74979691 133.69563762
   1.59534161 -31.75989139  -8.40887855]
Lowest MAPE: 0.11604876439643708 16 [-13.16732094  13.30030525  -3.8189956
-16.01018574 164.29324952
  -0.31206772 -31.45082966  -6.16443664]
```

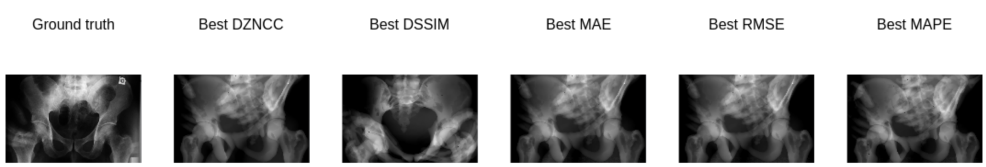
[83]:

```
xray_image_dzncc_nsga3 =␣
 ↪applyLogScaleAndNegative(updateXRayImage(res_nsga3_X[best_dzncc_id]))
xray_image_mae_nsga3   =␣
 ↪applyLogScaleAndNegative(updateXRayImage(res_nsga3_X[best_mae_id]))
xray_image_rmse_nsga3  =␣
 ↪applyLogScaleAndNegative(updateXRayImage(res_nsga3_X[best_rmse_id]))
xray_image_dssim_nsga3 =␣
 ↪applyLogScaleAndNegative(updateXRayImage(res_nsga3_X[best_dssim_id]))
xray_image_mape_nsga3  =␣
 ↪applyLogScaleAndNegative(updateXRayImage(res_nsga3_X[best_mape_id]))
```

[84]:

```
import plotly.express as px

fig = make_subplots(rows=1, cols=6,
                    start_cell="bottom-left",
                    subplot_titles=("Ground truth", "Best DZNCC", "Best DSSIM",␣
 ↪"Best MAE", "Best RMSE", "Best MAPE"))

nsga3_img_set = [standardised_roi_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)]
```

```python
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(output_path + "/HIPS-NSGA3-objectives.pdf", engine="kaleido")
fig.write_image(output_path + "/HIPS-NSGA3-objectives.png", engine="kaleido")

fig.show()
```



[85]:
```python
df_nsga3 = pd.DataFrame(data=np.append(res_nsga3_X, 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"])
```

```python
df_nsga3["ZNCC"] = 1.0 - (df_nsga3["DZNCC"] * 2.0)
df_nsga3["SSIM"] = 1.0 - (df_nsga3["DSSIM"] * 2.0)

df_nsga3["Optimiser"] = "NSGA3"
df_nsga3["Optimiser_code"] = 3
df_nsga3.to_csv(output_path + "/hips-optimiser-nsga3.csv")
```

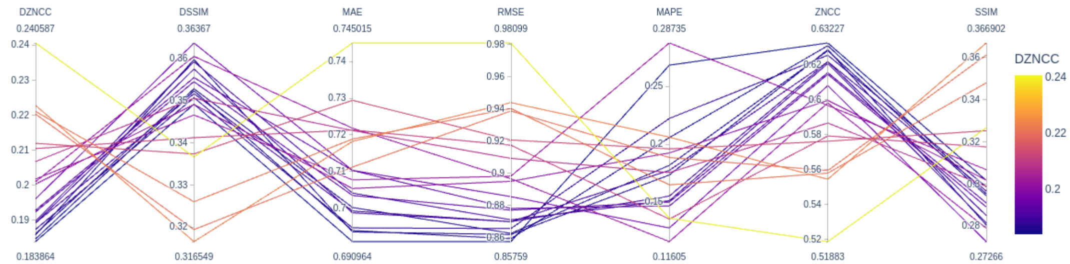
[86]:
```python
display(df_nsga3)
```

```
    sample_rotation_angle1  sample_rotation_angle2  src_pos_x  src_pos_y  \
0               -13.767274               10.017851  -9.680849 -32.501776
1                -7.730357               13.902997 -11.630203 -31.048287
2               -14.886822                9.847112 -19.222864 -23.749797
3                13.791372                5.568700  16.927914 -10.650334
4                14.847119              -14.932843 -11.562707 -10.529268
5               -14.964182               13.653620 -19.809022 -14.439679
6                14.873917              -14.456431  -6.807688 -12.475713
7                -9.652774               11.252956  -6.684295 -34.186217
8               -11.424566               10.352680 -15.442996 -30.972590
9               -11.372741               10.405394 -19.057956 -33.250807
10               -9.071985                9.812244 -12.564103 -31.843025
11              -11.448917                9.617288 -15.422520 -31.257372
12              -11.452484                8.797409 -15.481450 -31.175622
13               -8.089549               10.405394 -19.057956 -33.250807
14              -13.197806                9.617642 -15.422520 -31.133197
15              -13.508427                9.842279 -19.142787 -33.250807
16              -13.167321               13.300305  -3.818996 -16.010186
17               14.955251              -14.991797  -7.436388 -12.492031
18               14.955885              -14.999974 -19.058945 -12.677918
19              -14.998922              -14.390508  11.267749 -42.277089
20              -10.701581               10.336439 -19.256195 -31.162960

      src_pos_z  det_pos_x  det_pos_y  det_pos_z     DZNCC       MAE      RMSE  \
0    167.066001   1.432355 -33.871095  -7.399829  0.206717  0.721199  0.909323
1    165.779843  -0.285504 -31.661314  -6.160507  0.196035  0.710351  0.885517
2    133.695638   1.595342 -31.759891  -8.408879  0.183864  0.690964  0.857587
3    141.027338  -0.614831 -30.213381 -18.934932  0.221029  0.718866  0.940274
4    141.890920   1.574921 -30.541483 -26.259704  0.220303  0.711239  0.938728
5    141.674091   1.305043 -32.900129  -8.186302  0.185758  0.693589  0.861993
6    133.603519   1.547432 -29.710542 -27.261631  0.222766  0.718240  0.943962
7    133.859078  -0.312498 -31.839862  -3.421650  0.189202  0.699289  0.869948
8    167.871297   0.124351 -31.469508  -4.776229  0.192342  0.703518  0.877135
9    131.962232   1.060463 -31.323359  -7.473167  0.184633  0.693973  0.859378
10   133.659536   0.430109 -31.478868  -4.990511  0.192797  0.710318  0.878173
11   134.975189   1.417110 -34.588735  -4.803633  0.200276  0.705412  0.895044
12   135.431311   0.211491 -31.672448  -4.871572  0.187351  0.694694  0.865680
13   131.962232   1.060463 -31.323359  -7.648098  0.189721  0.704161  0.871141
```

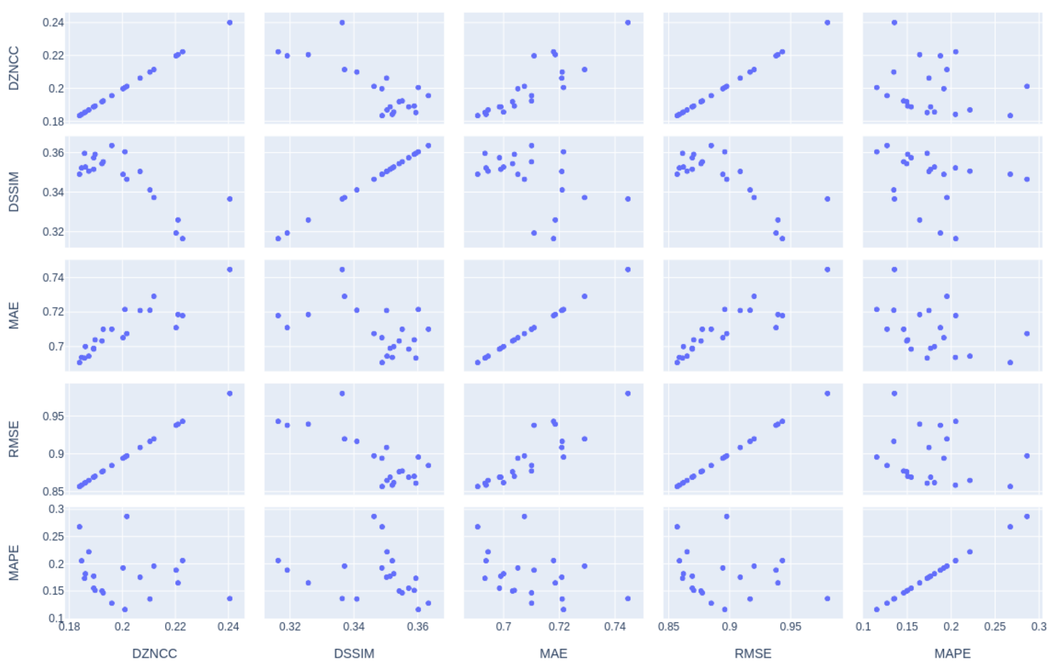
14	143.700316	1.425417	-34.413561	-8.226943	0.201713	0.707772	0.898250
15	131.865533	1.586051	-31.410235	-7.473607	0.186061	0.700260	0.862697
16	164.293250	-0.312068	-31.450830	-6.164437	0.200987	0.721837	0.896631
17	142.505164	2.591582	-30.940398	-27.456934	0.211933	0.729417	0.920724
18	133.054313	1.557569	-28.673396	-18.484314	0.240587	0.745015	0.980994
19	130.375138	0.945429	-33.069176	-1.231457	0.210425	0.721346	0.917442
20	131.706446	0.216550	-31.302922	-4.653586	0.189211	0.698794	0.869968

	DSSIM	MAPE	ZNCC	SSIM	Optimiser	Optimiser_code
0	0.350544	0.175546	0.586566	0.298911	NSGA3	3
1	0.363670	0.127784	0.607929	0.272660	NSGA3	3
2	0.349156	0.268253	0.632272	0.301689	NSGA3	3
3	0.326000	0.164993	0.557943	0.347999	NSGA3	3
4	0.319400	0.188592	0.559394	0.361199	NSGA3	3
5	0.359729	0.173506	0.628484	0.280541	NSGA3	3
6	0.316549	0.206132	0.554468	0.366902	NSGA3	3
7	0.351650	0.177467	0.621595	0.296700	NSGA3	3
8	0.354493	0.150204	0.615317	0.291014	NSGA3	3
9	0.352339	0.205736	0.630735	0.295322	NSGA3	3
10	0.355455	0.146767	0.614406	0.289089	NSGA3	3
11	0.349087	0.192564	0.599448	0.301827	NSGA3	3
12	0.350694	0.222262	0.625299	0.298612	NSGA3	3
13	0.359214	0.151321	0.620557	0.281572	NSGA3	3
14	0.346600	0.287349	0.596573	0.306800	NSGA3	3
15	0.352800	0.181863	0.627877	0.294399	NSGA3	3
16	0.360493	0.116049	0.598027	0.279013	NSGA3	3
17	0.337356	0.195996	0.576134	0.325289	NSGA3	3
18	0.336635	0.136212	0.518825	0.326730	NSGA3	3
19	0.341204	0.135472	0.579151	0.317592	NSGA3	3
20	0.357490	0.155304	0.621578	0.285019	NSGA3	3

[87]:
```
fig = px.parallel_coordinates(df_nsga3[["DZNCC", "DSSIM", "MAE", "RMSE",␣
↪"MAPE", "ZNCC", "SSIM"]], color="DZNCC")
fig.show()

fig.write_image(output_path + "/HIPS-NSGA3-parallel_coordinates.pdf",␣
↪engine="kaleido")
fig.write_image(output_path + "/HIPS-NSGA3-parallel_coordinates.png",␣
↪engine="kaleido")
```




```
[88]: fig = px.scatter_matrix(df_nsga3[["DZNCC", "DSSIM", "MAE", "RMSE", "MAPE"]])

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

fig.show()

fig.write_image(output_path + "/HIPS-NSGA3-scatter_matrix.pdf",␣
 ↪engine="kaleido")
fig.write_image(output_path + "/HIPS-NSGA3-scatter_matrix.png",␣
 ↪engine="kaleido")
```



[89]:
```
for i, x in enumerate(res_nsga3_X):
    img = xray_image_dzncc_nsga3 = applyLogScaleAndNegative(updateXRayImage(x))
    mpimg.imsave(output_path + "/NSGA3/img_" + str(i) + ".png", (255 * (img -␣
↪img.min()) / (img.max() - img.min())).astype(np.uint8), cmap="gray")
```

[90]:
```
best_nsga3_id = 3
x = res_nsga3_X[best_nsga3_id]

nsga3_raw_x_ray_image = applyLogScaleAndNegative(updateXRayImage(x))
nsga3_raw_x_ray_image = np.array(nsga3_raw_x_ray_image)

mpimg.imsave(output_path + "/NSGA3-img_" + str(best_nsga3_id) + ".png", (255 *␣
↪(nsga3_raw_x_ray_image - nsga3_raw_x_ray_image.min()) /␣
↪(nsga3_raw_x_ray_image.max() - nsga3_raw_x_ray_image.min())).astype(np.
↪uint8), cmap="gray")
```

[91]:
```
runtimes = []

resetToDefaultParameters()
setTransformations(res_nsga3_X[best_nsga3_id])

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

    temp = gvxr.computeXRayImage()

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

```
Set up the detector
        Detector position: [0.0, -30.5, -20.5, 'cm']
        Detector up vector: [0, -1, 0]
        Detector number of pixels: [1536, 1248]
        Pixel spacing: [0.026347682927083334, 0.026347683, 'cm']
Set up the beam
        Source position: [0.0, -30.5, 150.0, 'cm']
        Source shape: PointSource

Thu Sep 15 14:02:04 2022 ---- Initialise the renderer
```

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

[93]:
```
ZNCC = 1.0 - (2.0 * res_nsga3_F[best_nsga3_id,0])
SSIM = 1.0 - (2.0 * res_nsga3_F[best_nsga3_id,3])
MAPE = res_nsga3_F[best_nsga3_id,4]

print("Registration VHP & Real image & " +
      "{0:0.2f}".format(100 * MAPE) + "\\%    &    " +
      "{0:0.2f}".format(100 * ZNCC) + "\\%    &    " +
      "{0:0.2f}".format(SSIM) + "    &    $" +
      str(nsga3_raw_x_ray_image.shape[1]) + " \\times " +␣
 ↪str(nsga3_raw_x_ray_image.shape[0]) + "$    &    " +
      str(number_of_triangles) + "    &    " +
      "$" + str(runtime_avg) + " \\pm " + str(runtime_std) + "$ \\\\")
```

```
Registration VHP & Real image & 16.50\%    &    55.79\%    &    0.35    &
$1536 \times 1000$    &    1273438    &    $352 \pm 12$ \\
```

7 Compute the magnification

$magnification = \frac{SID}{SOD}$ with SID the source to imager distance and SOD the source to object distance.

[94]:
```
source_position = gvxr.getSourcePosition("mm")
detector_position = gvxr.getDetectorPosition("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
```

[95]:
```
print("SID:", source_imager_distance, "mm")
print("SOD:", source_object_distance, "mm")
print("magnification:", magnification)
```

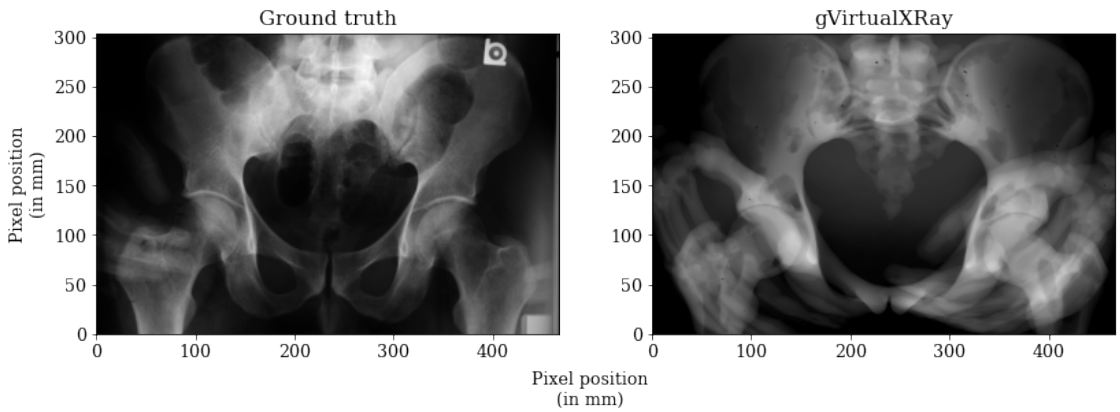
```
SID: 1621.0609642184588 mm
SOD: 1403.7924106361056 mm
magnification: 1.154772566040517
```

8 Compute the pixel size in the object plane

[96]:
```
detector_size = np.array(gvxr.getDetectorSize("mm"))
number_of_pixels = gvxr.getDetectorNumberOfPixels()

detector_element_spacing = (detector_size / number_of_pixels)
spacing = source_imager_distance * detector_element_spacing /␣
↪source_object_distance

print("spacing in the object plane (in mm):", spacing)
```

```
spacing in the object plane (in mm): [0.30425579 0.30425579]
```

[97]:
```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(20 * 2 / 3, 20))

vmin = standardised_roi_ground_truth.min()
vmax = standardised_roi_ground_truth.max()

im1 = axes.flat[0].imshow(standardised_roi_ground_truth, cmap="gray",␣
↪vmin=vmin, vmax=vmax,
                          extent=[0,(standardised_roi_ground_truth.
↪shape[1]-1)*spacing[0],0,(standardised_roi_ground_truth.
↪shape[0]-1)*spacing[1]])
axes.flat[0].set_title("Ground truth")
# axes.flat[0].set_xticks([])
# axes.flat[0].set_yticks([])

im2 = axes.flat[1].imshow(standardisation(nsga3_raw_x_ray_image), cmap="gray",␣
↪vmin=vmin, vmax=vmax,
                          extent=[0,(nsga3_raw_x_ray_image.
↪shape[1]-1)*spacing[0],0,(nsga3_raw_x_ray_image.shape[0]-1)*spacing[1]])
axes.flat[1].set_title("gVirtualXRay")
# axes.flat[1].set_xticks([])
# axes.flat[1].set_yticks([])

fig.text(0.5, 0.39, 'Pixel position\n(in mm)', ha='center')

axes.flat[0].set_ylabel("Pixel position\n(in mm)")

plt.savefig(output_path + 'full_comparison_VHP-NSGA3.pdf')
plt.savefig(output_path + 'full_comparison_VHP-NSGA3.png', bbox_inches =␣
↪'tight')
```



9 Compare the results obtained with the 3 different optimisation algorithms

[98]:
```
df = pd.concat([df_cmaes, df_nsga2, df_nsga3])
df = df.reindex(columns=["Optimiser", "Optimiser_code",␣
↪"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", "ZNCC", "SSIM"])

df.to_csv(output_path + "/hips-optimiser.csv")
```

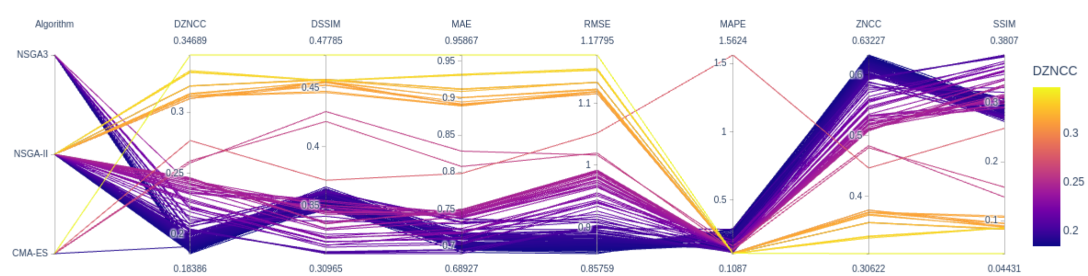
[99]:
```
display(df)
```

```
   Optimiser  Optimiser_code  sample_rotation_angle1  sample_rotation_angle2  \
0     CMA-ES               1               -4.999974                2.942652
1     CMA-ES               1               -4.999836                4.999441
2     CMA-ES               1               -4.999989                4.999922
3     CMA-ES               1                4.999735                4.999826
4     CMA-ES               1               -4.999659               -4.995986
..       …               …                     …                     …
16     NSGA3               3              -13.167321               13.300305
17     NSGA3               3               14.955251              -14.991797
18     NSGA3               3               14.955885              -14.999974
19     NSGA3               3              -14.998922              -14.390508
20     NSGA3               3              -10.701581               10.336439

    src_pos_x  src_pos_y   src_pos_z  det_pos_x  det_pos_y  det_pos_z  \
0    7.521376 -50.499428  130.000302   1.592841 -10.507583 -11.218151
1   16.973382 -50.067071  130.000067  -0.630155 -10.500383 -15.360448
2  -19.999830 -50.497748  130.000344   1.091760 -30.833317  -7.976479
3  -19.996561 -50.499921  130.000954  13.155220 -50.499978 -12.451590
4  -10.132157 -50.433492  130.041014   7.996906 -12.703856 -40.499699
```

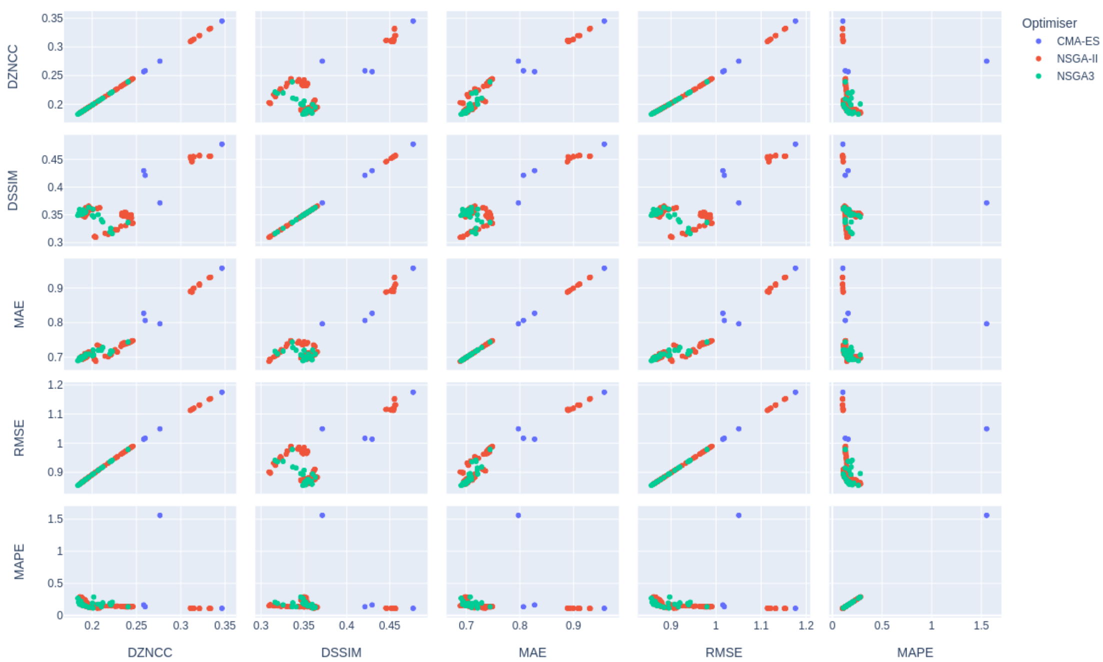
```
..         ...        ...         ...         ...        ...        ...
16   -3.818996 -16.010186  164.293250   -0.312068 -31.450830   -6.164437
17   -7.436388 -12.492031  142.505164    2.591582 -30.940398 -27.456934
18  -19.058945 -12.677918  133.054313    1.557569 -28.673396 -18.484314
19   11.267749 -42.277089  130.375138    0.945429 -33.069176   -1.231457
20  -19.256195 -31.162960  131.706446    0.216550 -31.302922   -4.653586

       DZNCC       MAE      RMSE     DSSIM      MAPE      ZNCC      SSIM
0   0.258320  0.828300  1.016504  0.429956  0.164314  0.483360  0.140088
1   0.259959  0.807418  1.019724  0.421568  0.136231  0.480081  0.156864
2   0.189623  0.703128  0.870914  0.351112  0.236018  0.620755  0.297775
3   0.276675  0.798011  1.051998  0.371779  1.562402  0.446650  0.256442
4   0.346892  0.958673  1.177950  0.477846  0.111581  0.306217  0.044308
..       ...       ...       ...       ...       ...       ...       ...
16  0.200987  0.721837  0.896631  0.360493  0.116049  0.598027  0.279013
17  0.211933  0.729417  0.920724  0.337356  0.195996  0.576134  0.325289
18  0.240587  0.745015  0.980994  0.336635  0.136212  0.518825  0.326730
19  0.210425  0.721346  0.917442  0.341204  0.135472  0.579151  0.317592
20  0.189211  0.698794  0.869968  0.357490  0.155304  0.621578  0.285019

[168 rows x 17 columns]
```

[100]:

```python
fig = px.parallel_coordinates(df, dimensions=['Optimiser_code', 'DZNCC',
    'DSSIM', 'MAE', 'RMSE', 'MAPE', "ZNCC", "SSIM"], color="DZNCC")
fig.data[0]["dimensions"][0]["label"] = "Algorithm"
fig.data[0]["dimensions"][0]["ticktext"] = ["CMA-ES", "NSGA-II", "NSGA3"]
fig.data[0]["dimensions"][0]["tickvals"] = [1, 2, 3]
fig.show()

fig.write_image(output_path + "/HIPS-ALL-parallel_coordinates.pdf",
    engine="kaleido")
fig.write_image(output_path + "/HIPS-ALL-parallel_coordinates.png",
    engine="kaleido")
```



[101]:
```
fig = px.scatter_matrix(df,
    dimensions=["DZNCC", "DSSIM", "MAE", "RMSE", "MAPE"],
    color="Optimiser")

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

fig.show()

fig.write_image(output_path + "/HIPS-ALL-scatter_matrix.pdf", engine="kaleido")
fig.write_image(output_path + "/HIPS-ALL-scatter_matrix.png", engine="kaleido")
```



10 Comparison of the analytic simulation with the real radiograph

[102]:
```
data = [
    [
        "CMA-ES",
        np.max(1.0 - (2.0 * evolution_fitness_zncc[:,1])),
        np.max(1.0 - (2.0 * evolution_fitness_ssim[:,1])),
        np.min(evolution_fitness_mae[:,1]),
        np.min(evolution_fitness_rmse[:,1]),
```

```
        np.min(evolution_fitness_mape[:,1])
    ],

    [
        "NSGA-II",
        np.max(1.0 - (2.0 * res_nsga2_F[:,0])),
        np.max(1.0 - (2.0 * res_nsga2_F[:,3])),
        np.min(res_nsga2_F[:,1]),
        np.min(res_nsga2_F[:,2]),
        np.min(res_nsga2_F[:,4])
    ],

    [
        "NSGA-3",
        np.max(1.0 - (2.0 * res_nsga3_F[:,0])),
        np.max(1.0 - (2.0 * res_nsga3_F[:,3])),
        np.min(res_nsga3_F[:,1]),
        np.min(res_nsga3_F[:,2]),
        np.min(res_nsga3_F[:,4])
    ]
]

df = pd.DataFrame(data=data,
                  columns=["Optimisation algorithm", "ZNCC", "SSIM", "MAE",␣
 ↪"RMSE", "MAPE"])

df.to_csv(output_path + "/hips-results.csv")

print(df)
```

```
  Optimisation algorithm      ZNCC      SSIM       MAE      RMSE      MAPE
0                 CMA-ES  0.542674  0.241476  0.767844  0.843934  0.111374
1                NSGA-II  0.629068  0.380696  0.689271  0.861315  0.108743
2                 NSGA-3  0.632272  0.366902  0.690964  0.857587  0.116049
```

11 All done

Destroy the window

[103]:
```
gvxr.destroyAllWindows()
```

```
Thu Sep 15 14:02:20 2022 ---- Destroy all the windows
Thu Sep 15 14:02:20 2022 ---- Destroy window 0(0x5645b8cb5e70)
```