Integration of OpenPyXL, .xlxs Geometry Spreadsheet and DiffDRR

Bill Worstell

PicoRad -> MGH

12/26/2023

DRR is a PyTorch module that compues differentiable digitally reconstructed radiographs. The viewing angle for the DRR (known generally in computer graphics as the *camera pose*) is parameterized by the following parameters:

- SDR : Source-to-Detector radius (half of the source-to-detector distance)
- $\mathbf{R} \in \mathrm{SO}(3)$: a rotation
- $\mathbf{t} \in \mathbb{R}^3$: a translation

Tip

`DiffDRR` can take a rotation parameterized in any of the following forms to move the detector plane:

- `axis_angle`
- `euler_angles` (note: also need to specify the `convention` for the Euler angles)
- `matrix`
- `quaternion`
- `rotation_6d` (Zhou et al., 2019)
- `rotation_10d` (Peretroukhin et al., 2021)
- `quaternion_adjugate` (Hanson and Hanson, 2022)

Report an issue

https://vivekg.dev/DiffDRR/api/drr.html#drr

If using Euler angles, the parameters are

- alpha : Azimuthal angle
- beta : Polar angle
- gamma : Plane rotation angle
- bx : X-dir translation
- by : Y-dir translation
- bz : Z-dir translation
- convention : Order of angles (e.g., ZYX)

(bx, by, bz) are translational parameters and (alpha, beta, gamma) are rotational parameters. The rotational parameters are detailed in Spherical Coordiantes Tutorial.

https://vivekg.dev/DiffDRR/api/drr.html#drr

DRR is a PyTorch module that compues differentiable digitally reconstructed radiographs. The viewing angle for the DRR (known generally in computer graphics as the *camera pose*) is parameterized by the following parameters:

- SDR : Source-to-Detector radius (half of the source-to-detector distance)
- $\mathbf{R} \in \mathrm{SO}(3)$: a rotation
- $\mathbf{t} \in \mathbb{R}^3$: a translation

DRR

DRR (volume:numpy.ndarray, spacing:numpy.ndarray, sdr:float, height:int, delx:float, width:int|None=None, dely:float|None=None, x0:float=0.0, y0:float=0.0, p_subsample:float|None=None, reshape:bool=True, reverse_x_axis:bool=False, patch_size:int|None=None, bone_attenuation_multiplier:float=1.0)

PyTorch module that computes differentiable digitally reconstructed radiographs.

	Type	Default	Details
volume	np.ndarray		CT volume
spacing	np.ndarray		Dimensions of voxels in the CT volume
sdr	float		Source-to-detector radius for the C-arm (half of the source-to-detector distance)
height	int		Height of the rendered DRR
delx	float		X-axis pixel size
width	int | None	None	Width of the rendered DRR (if not provided, set to `height`)
dely	float | None	None	Y-axis pixel size (if not provided, set to `delx`)
x0	float	0.0	Principal point X-offset
y0	float	0.0	Principal point Y-offset
p_subsample	float | None	None	Proportion of pixels to randomly subsample
reshape	bool	True	Return DRR with shape (b, 1, h, w)
reverse_x_axis	bool	False	If pose includes reflection (in E(3) not SE(3)), reverse x-axis
patch_size	int | None	None	If the entire DRR can't fit in memory, render patches of the DRR in series
bone_attenuation_multiplier	float	1.0	Contrast ratio of bone to soft tissue

The forward pass of the DRR module generated DRRs from the input CT volume. The pose parameters (i.e., viewing angles) from which the DRRs are generated are passed to the forward call.

DRR.forward

source

```
DRR.forward (rotation:torch.Tensor, translation:torch.Tensor,
             parameterization:str, convention:str=None,
             pose:pytorch3d.transforms.transform3d.Transform3d=None,
             bone_attenuation_multiplier:float=None)
```

Generate DRR with rotational and translational parameters.

	Type	Default	Details
rotation	torch.Tensor		
translation	torch.Tensor		
parameterization	str		
convention	str	None	
pose	Transform3d	None	If you have a preformed pose, can pass it directly
bone_attenuation_multiplier	float	None	Contrast ratio of bone to soft tissue

How to use `DiffDRR`

https://vivekg.dev/DiffDRR/tutorials/introduction.html

https://github.com/eigenvivek/DiffDRR

In-depth tutorial of the `DRR` module's functionality

```
import matplotlib.pyplot as plt
import torch

from diffdrr.data import load_example_ct
from diffdrr.drr import DRR
from diffdrr.visualization import plot_drr
```

DRR Generation

We demonstrate the speed of `DiffDRR` by timing repeated DRR synthesis. Timing results are on a single NVIDIA RTX 2080 Ti GPU.

```
33.8 ms ± 427 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

`DiffDRR` is implemented as a custom PyTorch module.

All raytracing operations have been formulated in a vectorized function, enabling use of PyTorch's GPU support and autograd. This also means that DRR generation is available as a layer in deep learning frameworks.

> **Tip**
>
> Rotations can be parameterized with numerous conventions (not just Euler angles). See `diffdrr.DRR` for more details.

```
# Read in the volume and get the isocenter
volume, spacing = load_example_ct()
bx, by, bz = torch.tensor(volume.shape) * torch.tensor(spacing) / 2

# Initialize the DRR module for generating synthetic X-rays
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
drr = DRR(
    volume,  # The CT volume as a numpy array
    spacing,  # Voxel dimensions of the CT
    sdr=300.0,  # Source-to-detector radius (half of the source-to-detector distance)
    height=200,  # Height of the DRR (if width is not seperately provided, the generated i
    delx=4.0,  # Pixel spacing (in mm)
).to(device)

# Set the camera pose with rotations (yaw, pitch, roll) and translations (x, y, z)
rotations = torch.tensor([[torch.pi, 0.0, torch.pi / 2]], device=device)
translations = torch.tensor([[bx, by, bz]], device=device)
img = drr(rotations, translations, parameterization="euler_angles", convention="ZYX")
plot_drr(img, ticks=False)
plt.show()
```

```
import matplotlib.pyplot as plt
import torch

from diffdrr.drr import DRR
from diffdrr.data import load_example_ct
from diffdrr.visualization import plot_drr

# Read in the volume and get the isocenter
volume, spacing = load_example_ct()
bx, by, bz = torch.tensor(volume.shape) * torch.tensor(spacing) / 2

# Initialize the DRR module for generating synthetic X-rays
device = "cuda" if torch.cuda.is_available() else "cpu"
drr = DRR(
    volume,      # The CT volume as a numpy array
    spacing,     # Voxel dimensions of the CT
    sdr=300.0,   # Source-to-detector radius (half of the source-to-detector distance)
    height=200,  # Height of the DRR (if width is not seperately provided, the generated image is squar
    delx=4.0,    # Pixel spacing (in mm)
).to(device)

# Set the camera pose with rotation (yaw, pitch, roll) and translation (x, y, z)
rotation = torch.tensor([[torch.pi, 0.0, torch.pi / 2]], device=device)
translation = torch.tensor([[bx, by, bz]], device=device)

# 📸 Also note that DiffDRR can take many representations of SO(3) 📸
# For example, quaternions, rotation matrix, axis-angle, etc...
img = drr(rotation, translation, parameterization="euler_angles", convention="ZYX")
plot_drr(img, ticks=False)
plt.show()
```

https://pytorch3d.org/tutorials/camera_position_optimization_with_differentiable_rendering

RUN IN GOOGLE COLAB | DOWNLOAD TUTORIAL JUPYTER NOTEBOOK | DOWNLOAD TUTORIAL SOURCE CODE

In [ ]:

```
# Copyright (c) Meta Platforms, Inc. and affiliates. All rights reserved.
```

Camera position optimization using differentiable rendering¶

In this tutorial we will learn the [x, y, z] position of a camera given a reference image using differentiable rendering.

We will first initialize a renderer with a starting position for the camera. We will then use this to generate an image, compute a loss with the reference image, and finally backpropagate through the entire pipeline to update the position of the camera.

This tutorial shows how to:

https://colab.research.google.com/github/BillWorstell/DiffDRR/blob/main/notebooks/tutorials/introduction.ipynb

https://colab.research.google.com/github/facebookresearch/pytorch3d/blob/stable/docs/tutorials/camera_position_optimization_with_differentiable_rendering.ipynb#scrollTo=sEVdNGFwripM

introduction.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

+ Code + Text Copy to Drive

title: How to use `DiffDRR` description: In-depth tutorial of the `DRR` module's functionality output-file: introduction.html

https://vivekg.dev/DiffDRR/

install DiffDRR from PyPI:

```
1 !pip install diffdrr
```

```
Collecting diffdrr
  Using cached diffdrr-0.3.8-py3-none-any.whl (34.6 MB)
Collecting fastcore
  Downloading fastcore-1.5.29-py3-none-any.whl (67 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 67.6/67.6 kB 3.1 MB/s eta 0:00:00
Collecting matplotlib
  Downloading matplotlib-3.8.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (11.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.6/11.6 MB 28.0 MB/s eta 0:00:00
Collecting torchvision
  Downloading torchvision-0.16.2-cp310-cp310-manylinux1_x86_64.whl (6.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.8/6.8 MB 18.4 MB/s eta 0:00:00
```

```
[2]  1 import os
     2 import sys
     3 import torch
     4 need_pytorch3d=False
     5 try:
     6     import pytorch3d
     7 except ModuleNotFoundError:
     8     need_pytorch3d=True
     9 if need_pytorch3d:
    10     if torch.__version__.startswith("2.1.") and sys.platform.startswith("linux"):
    11         # We try to install PyTorch3D via a released wheel.
    12         pyt_version_str=torch.__version__.split("+")[0].replace(".", "")
    13         version_str="".join([
    14             f"py3{sys.version_info.minor}_cu",
    15             torch.version.cuda.replace(".",""),
    16             f"_pyt{pyt_version_str}"
    17         ])
    18         !pip install fvcore iopath
    19         !pip install --no-index --no-cache-dir pytorch3d -f https://dl.fbaipublicfiles.com/pytorch3d/packaging/wheels/{version_str}/download.html
    20     else:
    21         # We try to install PyTorch3D from source.
    22         !pip install 'git+https://github.com/facebookresearch/pytorch3d.git@stable'
```

```
1 device = "cuda" if torch.cuda.is_available() else "cpu"
2 drr = DRR(
3     volume,      # The CT volume as a numpy array
4     spacing,     # Voxel dimensions of the CT
5     sdr=300.0,   # Source-to-detector radius (half of the source-to-detector distance)
6     height=200,  # Height of the DRR (if width is not seperately provided, the generated image is square)
7     delx=4.0,    # Pixel spacing (in mm)
8 ).to(device)
```

Set the camera pose with rotation (yaw, pitch, roll) and translation (x, y, z)

```
[25]  1 rotation = torch.tensor([[torch.pi, 0.0, torch.pi / 2]], device=device)
      2 translation = torch.tensor([[bx, by, bz]], device=device)
```

Also note that DiffDRR can take many representations of SO(3)

For example, quaternions, rotation matrix, axis-angle, etc...

```
[26]  1 img = drr(rotation, translation, parameterization="euler_angles", convention="ZYX")
      2 plot_drr(img, ticks=False)
      3 plt.show()
```

Get characteristics of volume

```
[39]  1 ic(volume.dtype)
      2 ic(volume.shape)
      3 ic(volume.size)
```

```
ic| volume.dtype: dtype('float32')
ic| volume.shape: (512, 512, 133)
ic| volume.size: 34865152
34865152
```

Get voxel spacing

```
[42]  1 ic(spacing)
```

```
ic| spacing: [0.703125, 0.703125, 2.5]
[0.703125, 0.703125, 2.5]
```

https://github.com/BillWorstell/DiffDRR/blob/main/notebooks/tutorials/introduction.ipynb

MDSL.excel80M10RFR.cut-plate.008.150roi.2.30pin.105ellipse .XLSX

File Edit View Insert Format Data Tools Help

Menus 100% $ % .0 .00 123 Calibri 11 B I A

A1

	A	B	C	D	E	F	G	H	I	J	K
1		x1	y1	z1	b	S	l	l + b+Td	x2	y2	z2
2	Definitions	x coordinate value at center of hole	y coordinate value at center of hole	z coordinate value at center of hole	distance from center of hole to center of heart	3D slope	length of collimator		x coordinate value at end of detector	y coordinate value at end of detector	z coordinate value at end of detector
3	1	277.002	70.37124	-70.0363	294.2572051	1.062292709	89.07020616	393.3274112	370.2627419	94.06375503	-93.61604779
4	2	242.8831	107.2608	-70.2602	274.6517344	1.130798044	82.70081149	367.3525458	324.8613206	143.4636051	-93.97451431
5	3	201.3326	136.87	-70.5519	253.4675193	1.258949218	75.77607567	339.2435949	269.4656704	183.1882482	-94.42740535
6	4	155.0181	158.4473	-70.9056	232.7310944	1.501315617	68.94258232	311.6736767	207.6003695	212.192757	-94.95683896
7	5	106.2709	171.9591	-71.3542	214.3708425	2.017211132	62.83224619	287.2030887	142.3763156	230.3820057	-95.59670705
8	6	56.74056	177.7703	-71.7895	199.9387482	3.523735899	57.97763045	267.9163786	76.03191225	238.2108293	-96.19737564

ReadGeometryXLS.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
[12]  2 print(wb.active.title)
      3 ws = wb.active
```

```
Coordinates
```

x coordinate value at center of hole

```
[13]  1 ic(ws.cell(2,2).value)
      2 x1=np.zeros(80)
      3 for i in range(3,83):
      4   x1[i-3]=(ws.cell(i,2).value)
      5 print(x1)
```

```
ic| ws.cell(2,2).value: 'x coordinate value at center of hole'
[ 277.002    242.8831   201.3326   155.0181   106.2709    56.74056
    7.351419  -41.7583   -89.0623  -132.361   -170.499   -201.258
 -221.948    -230.423   -225.927   -209.229   -182.273   -147.473
 -107.125     -63.1068   277.002    242.8831   201.3326   155.0181
  106.2709    56.74056    7.351419  -41.7583   -89.0623  -132.361
```

ReadGeometryXLS.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

y coordinate value at center of hole

```
[14]  1 ic(ws.cell(2,3).value)
      2 y1=np.zeros(80)
      3 for i in range(3,83):
      4   y1[i-3]=(ws.cell(i,3).value)
      5 print(y1)
```

```
ic| ws.cell(2,3).value: 'y coordinate value at center of hole'
[  70.37124  107.2608   136.87     158.4473   171.9591   177.7703
  176.2461   167.3279   150.8126   126.955     95.57272   56.97774
   12.56229  -35.1111   -82.9365  -127.899   -167.656   -200.839
 -226.898   -245.785    70.37124  107.2608   136.87     158.4473
```

ReadGeometryXLS.ipynb

File Edit View Insert Runtime Tools Help All changes saved

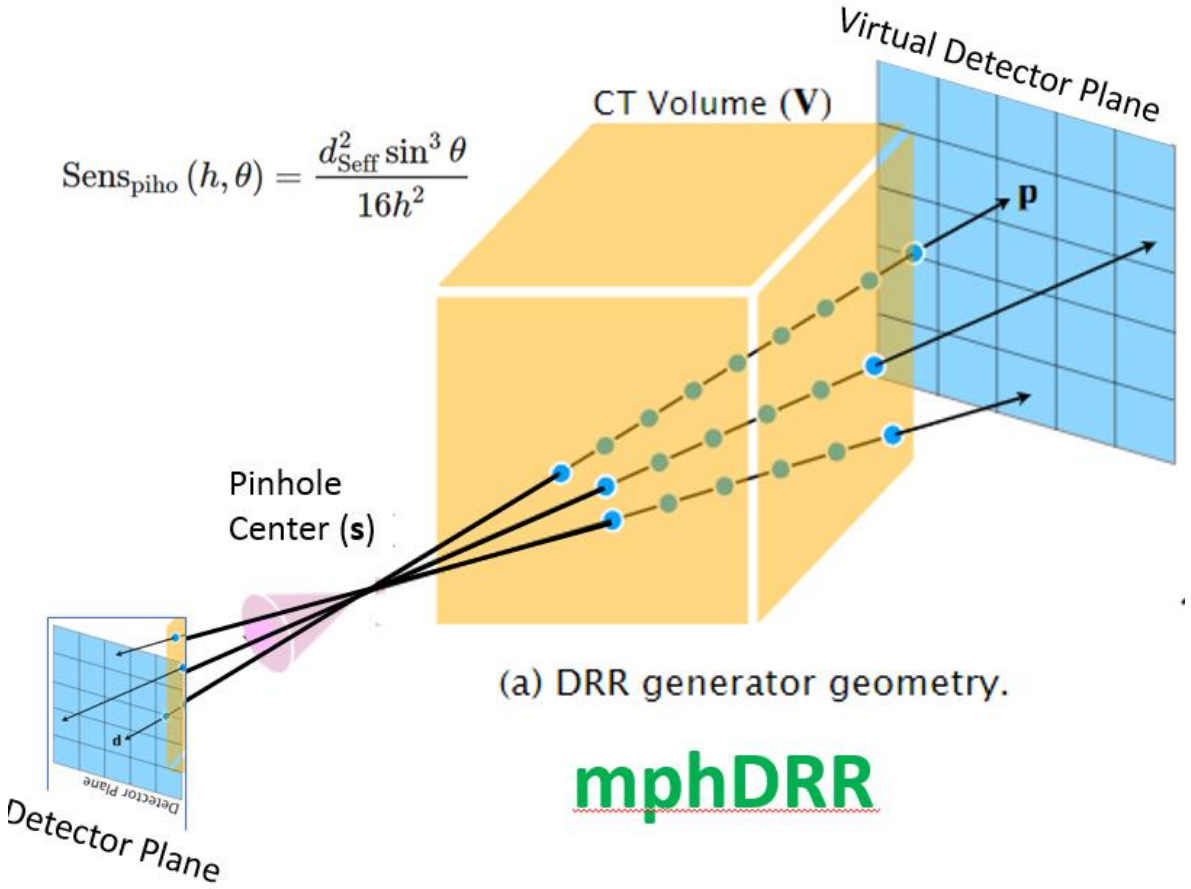
+ Code + Text

z coordinate value at center of hole

```
1 ic(ws.cell(2,4).value)
2 z1=np.zeros(80)
3 for i in range(3,83):
4   z1[i-3]=(ws.cell(i,4).value)
5 print(z1)
```

```
ic| ws.cell(2,4).value: 'z coordinate value at center of hole'
[-70.0363 -70.2602 -70.5519 -70.9056 -71.3542 -71.7895 -72.1241 -72.2656
 -72.1683 -71.8898 -71.5318 -71.1818 -70.8925 -70.7108 -70.5931 -70.5263
 -70.4918 -70.4707 -70.4467 -70.4085 -23.4255 -23.5096 -23.62   -23.7553
 -23.9071 -24.0565 -24.1728 -24.2224 -24.1883 -24.0912 -23.9678 -23.8485
```

https://vivekg.dev/DiffDRR/api/drr.html#drr



	Type	Default	Details
volume	np.ndarray		CT volume
spacing	np.ndarray		Dimensions of voxels in the CT volume
sdr	float		Source-to-detector radius for the C-arm (half of the source-to-detector distance)

Choose sdr = 5 * [length of collimator]
Magnification M=10

=(ATAN2(B3,C3))

	T	U	V
axis		pinhole range	pinhole azimuth (radians)
4725		285.8010137	0.2487829888
1903		265.5128612	0.4158590487

=(ATAN2(U3,D3))

	T	U	V	W
axis		pinhole range	pinhole azimuth (radians)	pinhole elevation (radians)
14725		285.8010137	0.2487829888	-0.2403169548
91903		265.5128612	0.4158590487	-0.2586912886

=E3+G3+Main!T3

	B	C	D	E	F	G	H
	x1	y1	z1	b	S	l	l + b+Td
	x coordinate value at center of hole	y coordinate value at center of hole	z coordinate value at center of hole	distance from center of hole to center of heart	3D slope	length of collimator	
1	277.002	70.37124	-70.0363	294.2572051	1.062292709	89.07020616	393.3274112
2	242.8831	107.2608	-70.2602	274.6517344	1.130798044	82.70081149	367.3525458

=SQRT(SUMSQ(B3,C3))

	T	U
axis		pinhole range
725		285.8010137
903		265.5128612

If using Euler angles, the parameters are

- alpha : Azimuthal angle
- beta : Polar angle
- gamma : Plane rotation angle
- bx : X-dir translation
- by : Y-dir translation
- bz : Z-dir translation
- convention : Order of angles (e.g., ZYX)

(bx, by, bz) are translational parameters and (alpha, beta, gamma) are rotational parameters. The rotational parameters are detailed in Spherical Coordiantes Tutorial.

ReadGeometryXLS.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

Length of Collimator

```
1 ic(ws.cell(2,7).value)
2 lcoll=np.zeros(80)
3 for i in range(3,83):
4   lcoll[i-3]=(ws.cell(i,7).value)
5 print(lcoll)
```

V
pinhole azimuth (radians)
0.2487829888
0.4158590487
0.5970538066
0.7963373714
1.017245821
1.26183823
1.52910939

Choose vsdr = 5 * length of collimator

```
[ ]  1 vsdr=5.*lcoll
```

alpha: Azimuthal angle (radians)

```
1 ic(ws.cell(2,22).value)
2 alpha=np.zeros(80)
3 for i in range(3,83):
4   alpha[i-3]=(ws.cell(i,22).value)
5 print(alpha)
```

```
ic| ws.cell(2,22).value: 'pinhole azimuth (radians)'
[ 0.24878299  0.41585905  0.59705381  0.79633737  1.01724582  1.26183823
  1.52910939  1.81536054  2.10423792  2.37703862  2.63068794  2.86570428
  3.08505283 -2.9903791  -2.78977101 -2.59291509 -2.39794155 -2.20416353
```

ReadGeometryXLS.ipynb

File Edit View Insert Runtime Tools Help All changes saved

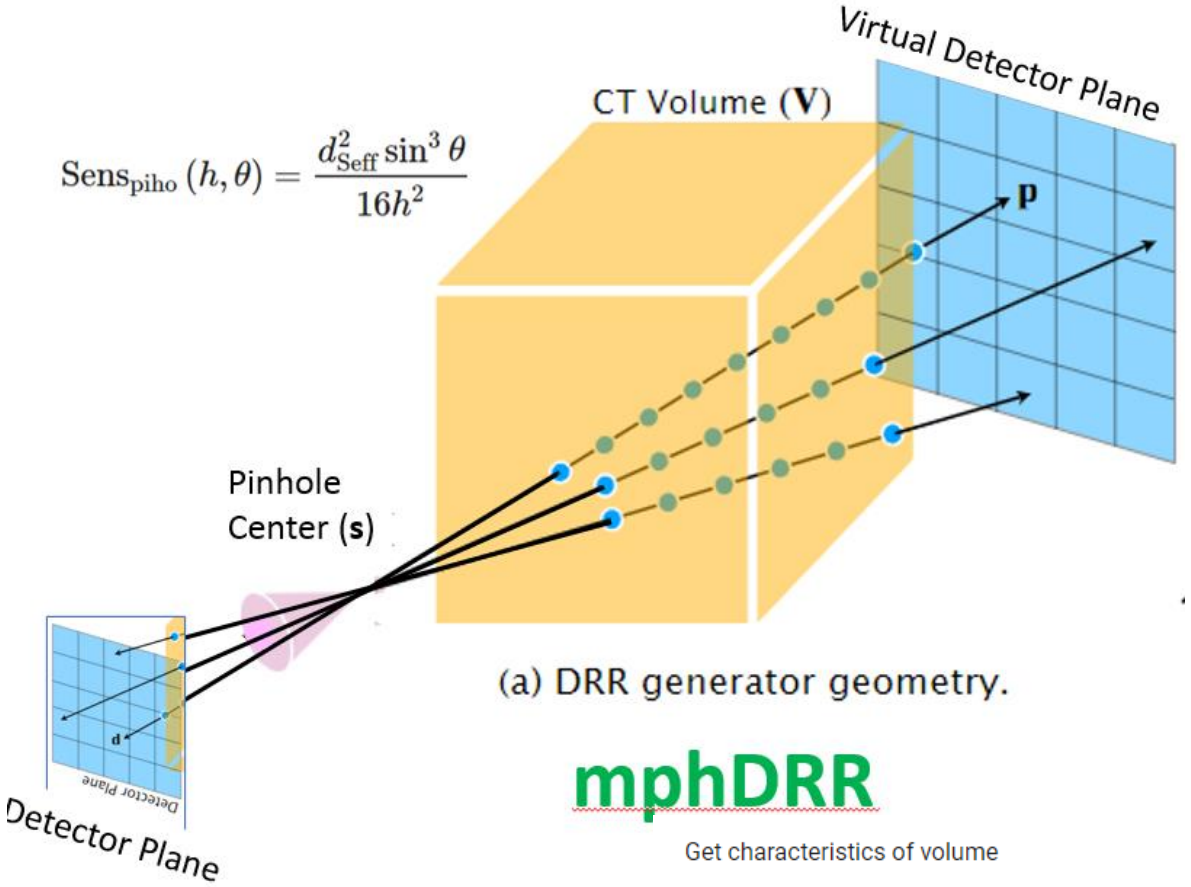
+ Code + Text

beta = altitude (radians)

```
1 ic(ws.cell(2,233).value)
2 beta=np.zeros(80)
3 for i in range(3,83):
4   beta[i-3]=(ws.cell(i,23).value)
5 print(beta)
```

```
ic| ws.cell(2,233).value: None
[-0.24031695 -0.25869129 -0.28207257 -0.30958931 -0.33932858 -0.36725782
 -0.38812799 -0.39680203 -0.39084561 -0.37356937 -0.35082977 -0.32801569
 -0.3087048  -0.29454762 -0.28531763 -0.28004146 -0.27730567 -0.27562572
 -0.27371232 -0.27065554 -0.08178156 -0.0883138  -0.096719   -0.10675926
 -0.11771907 -0.12820893 -0.13618629 -0.13953966 -0.13723464 -0.13060819
 -0.1220143  -0.11352578 -0.10643635 -0.10125    -0.09789449 -0.09598459
 -0.09499655 -0.09439062 -0.09370115 -0.09260112  0.08178156  0.0883138
```

W
pinhole elevation (radians)
-0.2403169548
-0.2586912886
-0.2820725687
-0.3095893101
-0.3393285829



mphDRR

https://vivekg.dev/DiffDRR/api/drr.html#drr

height	int		Height of the rendered DRR
delx	float		X-axis pixel size
width	int | None	None	Width of the rendered DRR (if not provided, set to `height`)
dely	float | None	None	Y-axis pixel size (if not provided, set to `delx`)
x0	float	0.0	Principal point X-offset
y0	float	0.0	Principal point Y-offset

Volume [X Y Z] is
360 x 360 x 332.5 mm

Get size of image

```
1 ic(img.shape)
```

```
ic| img.shape: torch.Size([1, 1, 200, 200])
torch.Size([1, 1, 200, 200])
```

Get characteristics of volume

```
[39]  1 ic(volume.dtype)
      2 ic(volume.shape)
      3 ic(volume.size)
```

```
ic| volume.dtype: dtype('float32')
ic| volume.shape: (512, 512, 133)
ic| volume.size: 34865152
34865152
```

Get voxel spacing

```
[42]  1 ic(spacing)
```

```
ic| spacing: [0.703125, 0.703125, 2.5]
[0.703125, 0.703125, 2.5]
```

Choose sdr = 5 * [length of collimator] -> ~450mm
Magnification M=10, Source-to-detector -> ~900mm

Length of side of virtual detector crystal = 500 mm

height= 500
delx = 500/200 = 2.5
width = 500
dely = 500/200=2.5
x0=0
y0=0

s	
Ld	T
length of side of detector crystal	th of d cr
50	
50	
50	
50	

```
# Initialize the DRR module for generating synthetic X-rays
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
drr = DRR(
    volume,  # The CT volume as a numpy array
    spacing,  # Voxel dimensions of the CT
    sdr=300.0,  # Source-to-detector radius (half of the source-to-detector distance)
    height=200,  # Height of the DRR (if width is not seperately provided, the generated image is square)
    delx=4.0,  # Pixel spacing (in mm)
).to(device)
# Set the camera pose with rotations (yaw, pitch, roll) and translations (x, y, z)
rotations = torch.tensor([[torch.pi, 0.0, torch.pi / 2]], device=device)
translations = torch.tensor([[bx, by, bz]], device=device)
img = drr(rotations, translations, parameterization="euler_angles", convention="ZYX")
```

```
# Initialize the DRR module for generating synthetic X-rays
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
drr = DRR(
    volume,  # The CT volume as a numpy array
    spacing,  # Voxel dimensions of the CT
    sdr=vsdr[imod],  # Source-to-virtual-detector radius (half of the source-to-virtual-detector distance)
    height=500,  # Height of the DRR (if width is not seperately provided, the generated image is square)
    delx=2.5,  # Pixel spacing (in mm)
).to(device)
# Set the camera pose with rotations (yaw, pitch, roll) and translations (x, y, z)
rotations = torch.tensor([alpha[imod],beta[imod],0.], device=device)
translations = torch.tensor([[bx, by, bz]], device=device)
img = drr(rotations, translations, parameterization="euler_angles", convention="ZYX")
```

module 1-80 version 008

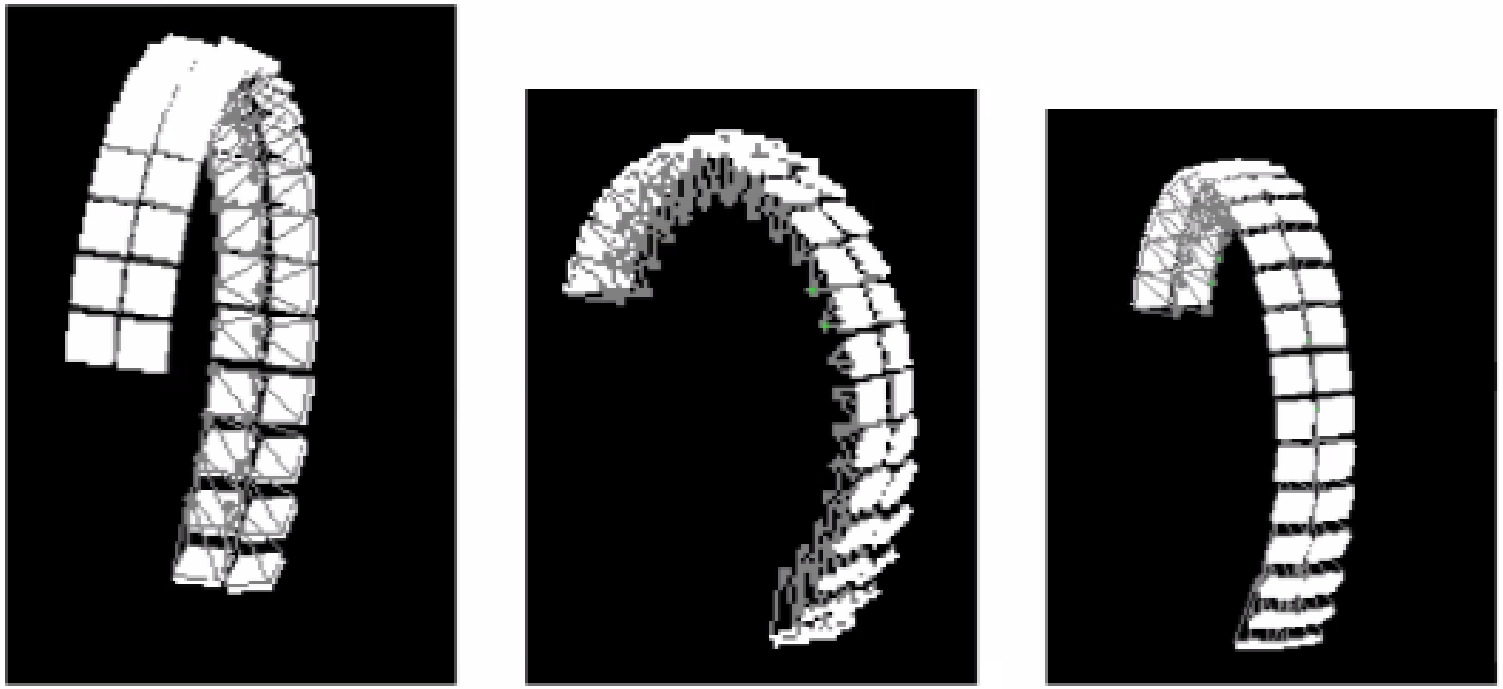


Figure 12: Left: module 41-80, central: module 1-40, right: module 21-60

Issue remaining: Gate fails when involving 80 heads in a long macro.