Learned Multi-Perspective 3-D CT and SPECT/CT Reconstruction using the PyTorch3D Library

Bill Worstell

PicoRad -> PDSI + MGH

2/20/2024

Learned Visual Depth Perception is familiar from human experience, monocular and binocular

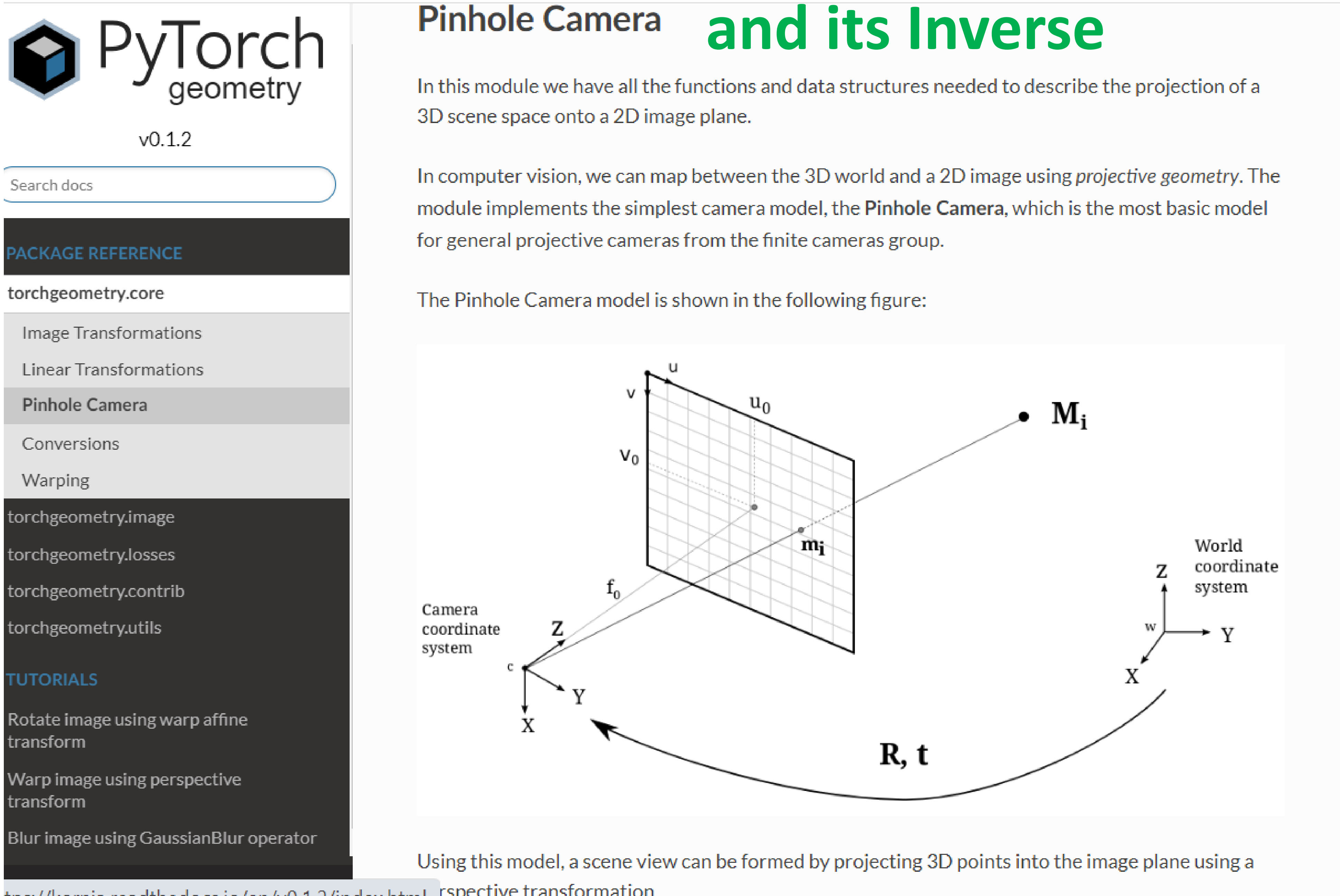
- Most of us have the equivalent of two high-resolution pinhole cameras, with parallax provided by binocular input image data
- The scene we are viewing contains generally opaque objects at varying depths. After we have learned object permanence and the like, we will have formed a mental model of our 3D environment and its features that we can use to predict from an observed scene were it to be observed from an alternative perspective.
- In the CT and SPECT/CT 3D imaging case, the scene contains “translucent” objects with well understood attenuation and background projection space effects, and we seek an accurate high resolution and ideally isotropic image with good MTF from projection space perturbations at high frequencies.
- In our learned multi-perspective 3D imaging case we seek a neural network architecture which embeds the (presumed known) geometry of a multiple pinhole camera system (or equivalent) when assessing “parallax” by having each camera predict observations from the other cameras’ perspectives to infer depth distributions along each line of sight.

https://kornia.readthedocs.io/en/v0.1.2/pinhole.html

Projection Transform and its Inverse

Each point in the Pinhole Camera 2D projection space [u,v] has a magnification factor k given the depth of field of the source voxel M,

The signal measured at mi is a line integral through a set of voxels in the 3D homogeneous coordinate projection space volume, which is equivalent to the source volume with a known image-to-image transform for each camera



PyTorch geometry

v0.1.2

Search docs

PACKAGE REFERENCE

torchgeometry.core

- Image Transformations
- Linear Transformations
- Pinhole Camera
- Conversions
- Warping

torchgeometry.image

torchgeometry.losses

torchgeometry.contrib

torchgeometry.utils

TUTORIALS

Rotate image using warp affine transform

Warp image using perspective transform

Blur image using GaussianBlur operator

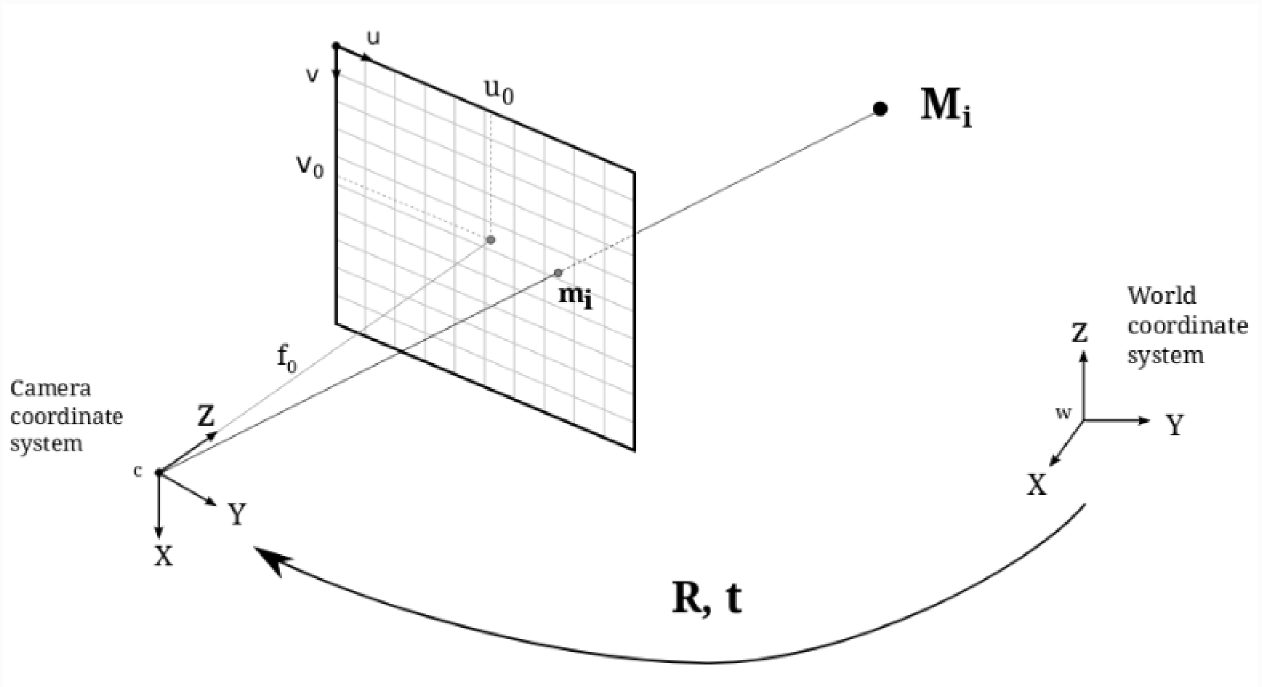
tps://kornia.readthedocs.io/en/v0.1.2/index.html

Pinhole Camera

In this module we have all the functions and data structures needed to describe the projection of a 3D scene space onto a 2D image plane.

In computer vision, we can map between the 3D world and a 2D image using *projective geometry*. The module implements the simplest camera model, the **Pinhole Camera**, which is the most basic model for general projective cameras from the finite cameras group.

The Pinhole Camera model is shown in the following figure:



Using this model, a scene view can be formed by projecting 3D points into the image plane using a
rspective transformation

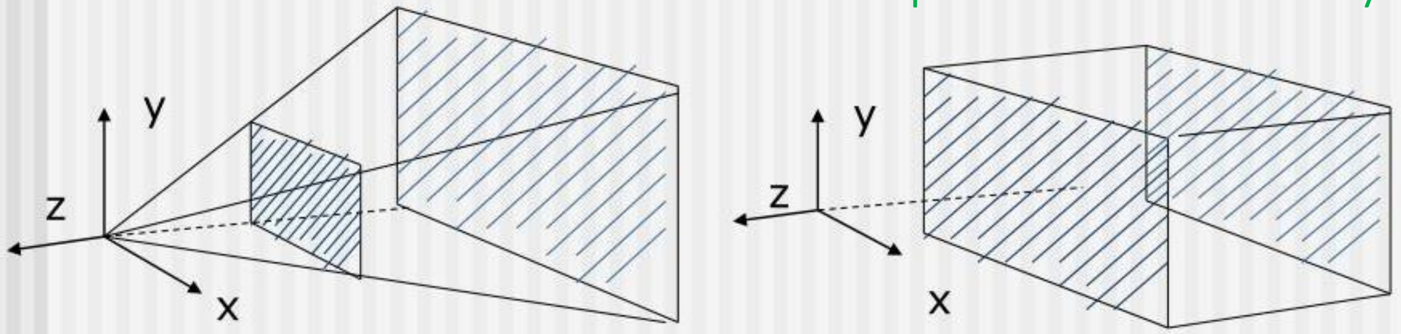
There is a one-to-one continuous mapping between a source volume in Cartesian space coordinate representation and Homogeneous Projection coordinates with scale factor inverse with distance to screen.

Projection Transformation

- Projection – map the object from 3D space to 2D screen

K>0 with k=f/z

K=0 -> view from point at z = infinity

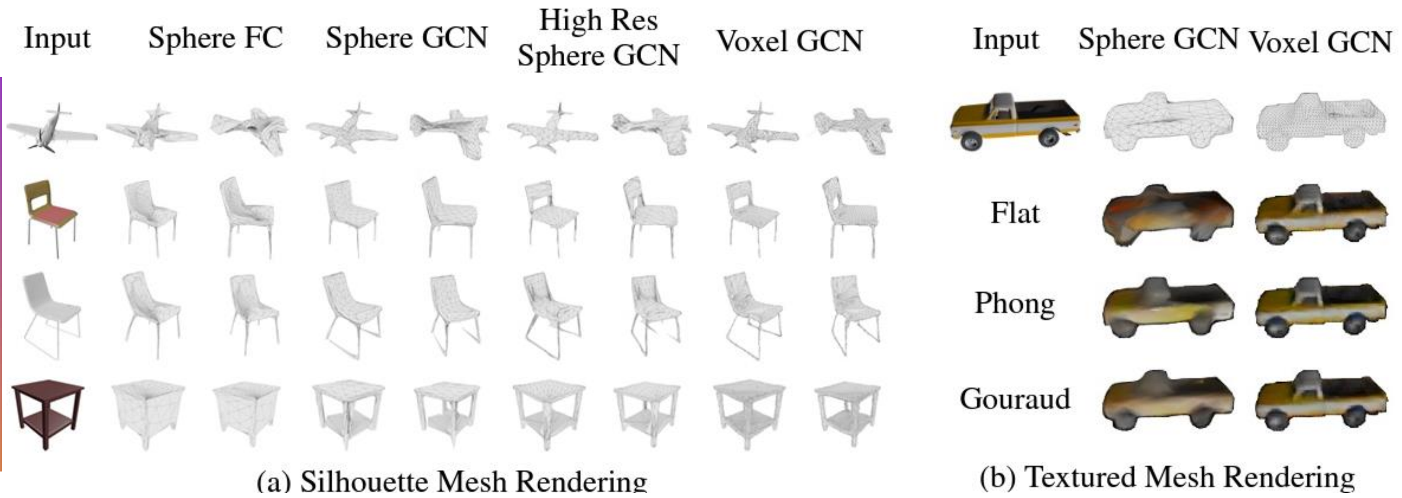


Perspective: **gluPerspective()**

Parallel: **glOrtho()**

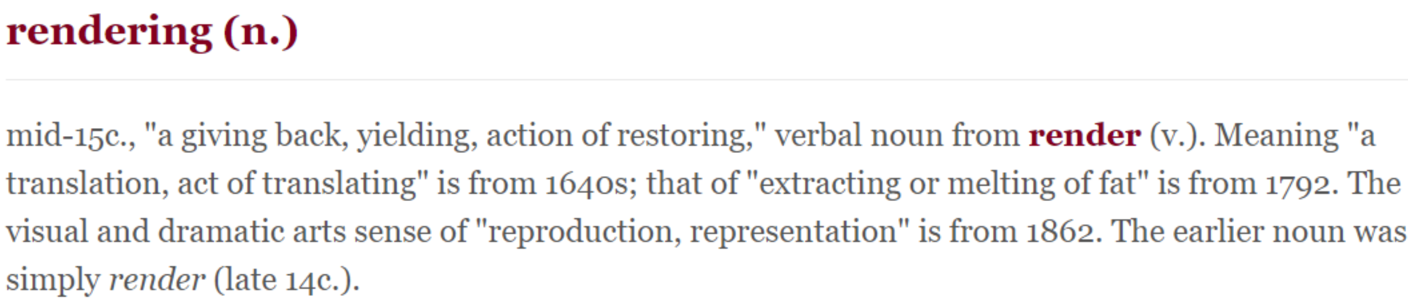
Mapping from Source to a camera orthogonal parallel reference frame can be represented as a sequence of rotations and a translation, with inverses in opposite sequence to transform to voxel points in lab frame from equivalent homogeneous coordinates continuously one-to-one.

Scale by k at each z to map from camera system Orthogonal coordinates onto one-to-one equivalent Perspective projected homogeneous coordinates https://pytorch3d.org/



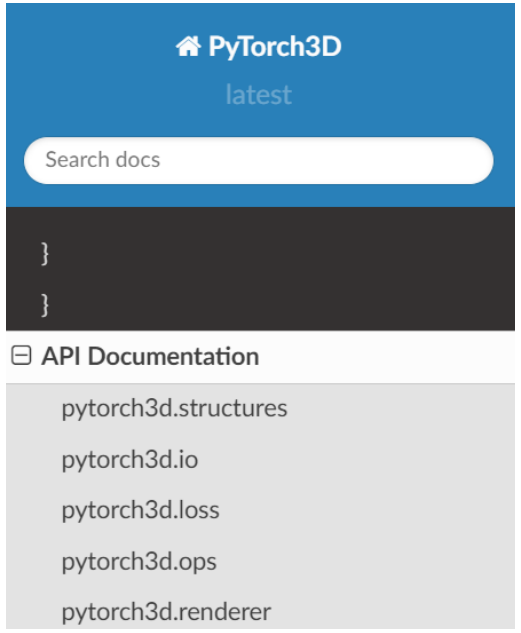
(a) Silhouette Mesh Rendering (b) Textured Mesh Rendering

Ravi, N., Reizenstein, J., Novotny, D., Gordon, T., Lo, W.Y., Johnson, J. and Gkioxari, G., 2020. Accelerating 3d deep learning with pytorch3d. *arXiv preprint arXiv:2007.08501*.



rendering (n.)

mid-15c., "a giving back, yielding, action of restoring," verbal noun from **render** (v.). Meaning "a translation, act of translating" is from 1640s; that of "extracting or melting of fat" is from 1792. The visual and dramatic arts sense of "reproduction, representation" is from 1862. The earlier noun was simply *render* (late 14c.).



https://pytorch3d.readthedocs.io/en/latest/modules/transforms.html

Heterogeneous Batching

Supports batching of 3D inputs of different sizes such as meshes

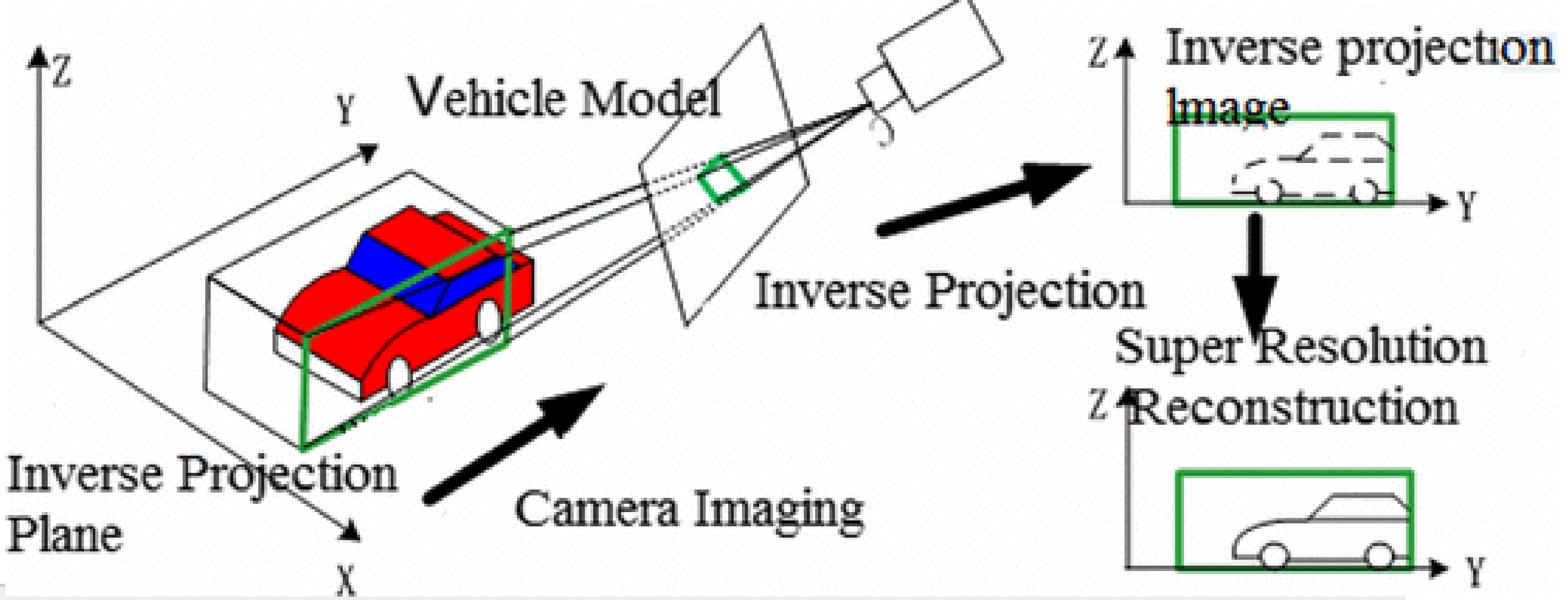
Fast 3D Operators

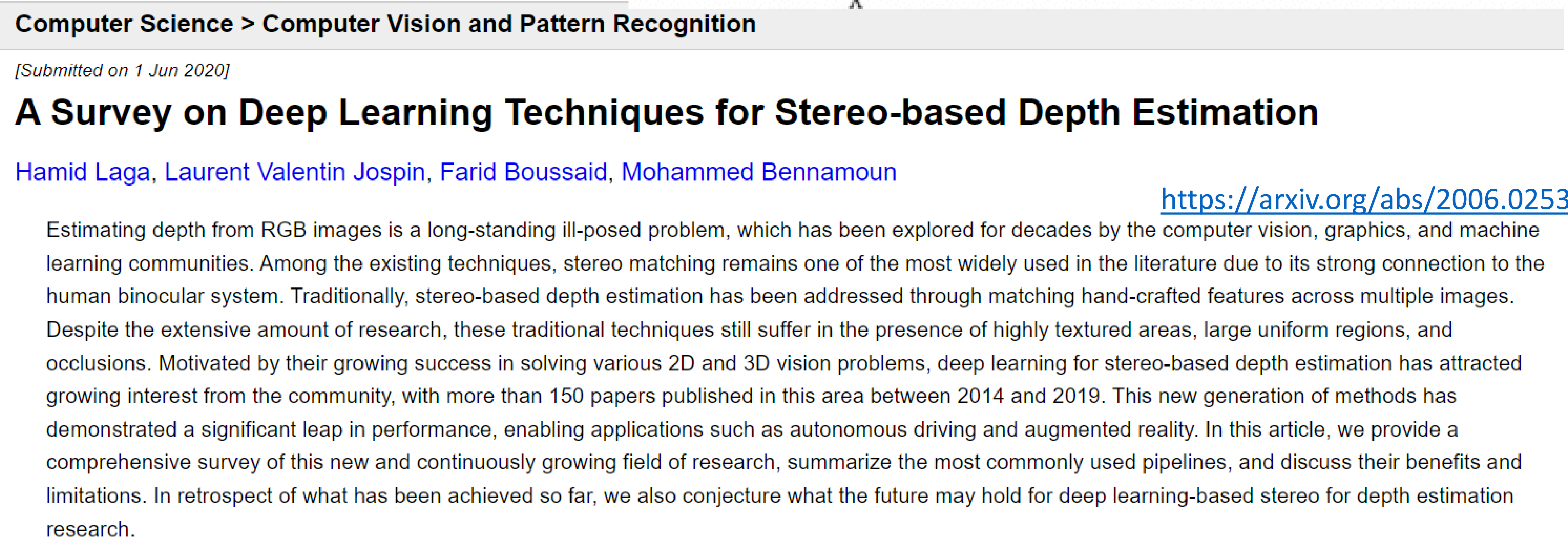
Supports optimized implementations of several common functions for 3D data

Differentiable Rendering

Modular differentiable rendering API with parallel implementations in PyTorch, C++ and CUDA

"Deep learning has significantly improved 2D image recognition. Extending into 3D may advance many new applications including autonomous vehicles, virtual and augmented reality, authoring 3D content, and even improving 2D recognition"
[Accelerating 3D Deep Learning with PyTorch3d]




Computer Science > Computer Vision and Pattern Recognition

[Submitted on 1 Jun 2020]

A Survey on Deep Learning Techniques for Stereo-based Depth Estimation

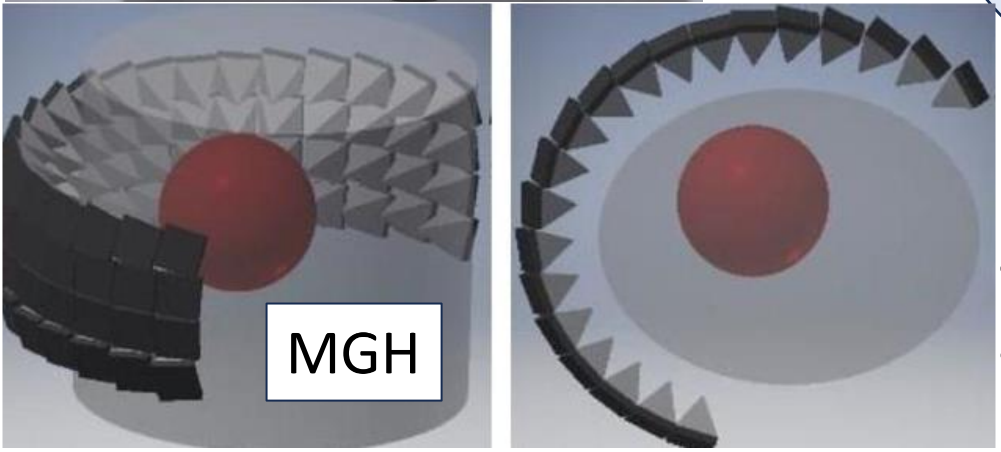
Hamid Laga, Laurent Valentin Jospin, Farid Boussaid, Mohammed Bennamoun

https://arxiv.org/abs/2006.02535

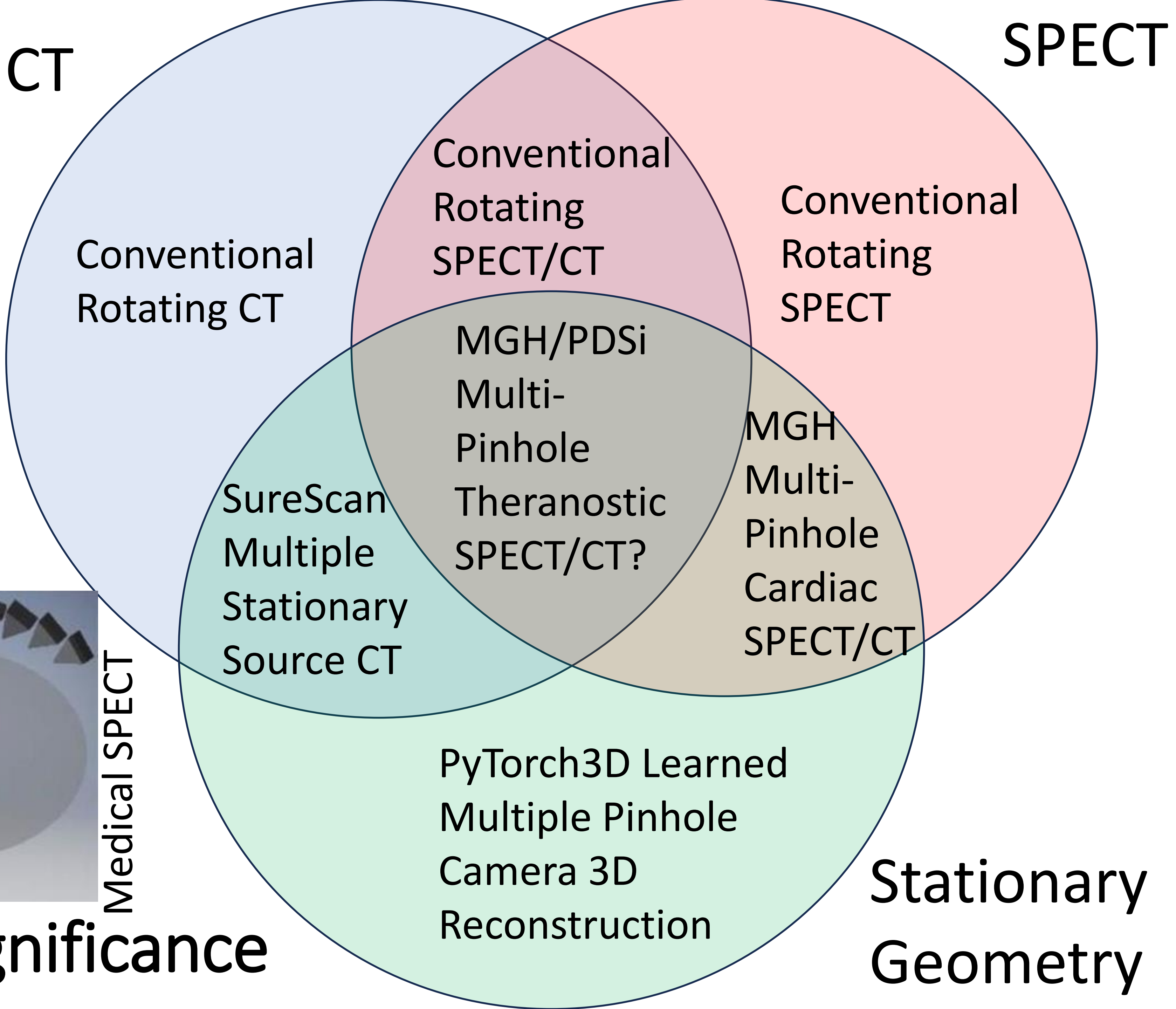
Estimating depth from RGB images is a long-standing ill-posed problem, which has been explored for decades by the computer vision, graphics, and machine learning communities. Among the existing techniques, stereo matching remains one of the most widely used in the literature due to its strong connection to the human binocular system. Traditionally, stereo-based depth estimation has been addressed through matching hand-crafted features across multiple images. Despite the extensive amount of research, these traditional techniques still suffer in the presence of highly textured areas, large uniform regions, and occlusions. Motivated by their growing success in solving various 2D and 3D vision problems, deep learning for stereo-based depth estimation has attracted growing interest from the community, with more than 150 papers published in this area between 2014 and 2019. This new generation of methods has demonstrated a significant leap in performance, enabling applications such as autonomous driving and augmented reality. In this article, we provide a comprehensive survey of this new and continuously growing field of research, summarize the most commonly used pipelines, and discuss their benefits and limitations. In retrospect of what has been achieved so far, we also conjecture what the future may hold for deep learning-based stereo for depth estimation research.

Applications and Significance

Security CT



Medical SPECT

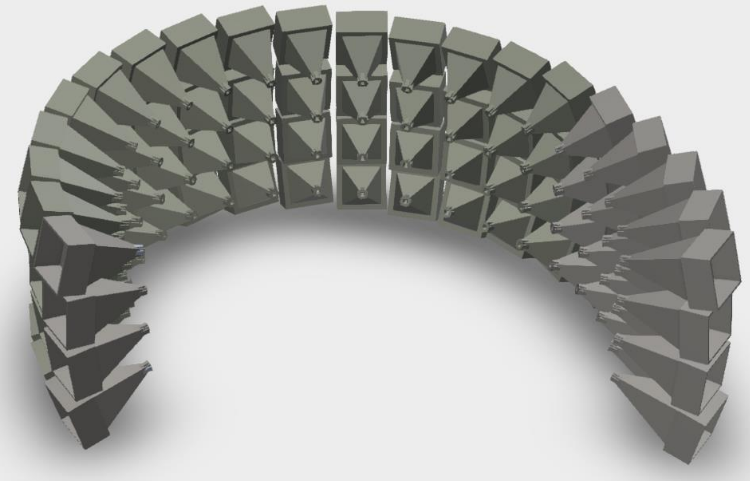

CT
SPECT
Stationary Geometry
Conventional Rotating CT
Conventional Rotating SPECT/CT
Conventional Rotating SPECT
MGH/PDSi Multi-Pinhole Theranostic SPECT/CT?
SureScan Multiple Stationary Source CT
MGH Multi-Pinhole Cardiac SPECT/CT
PyTorch3D Learned Multiple Pinhole Camera 3D Reconstruction

MGH/PDSi Multi-Pinhole Theranostic SPECT/CT Concept

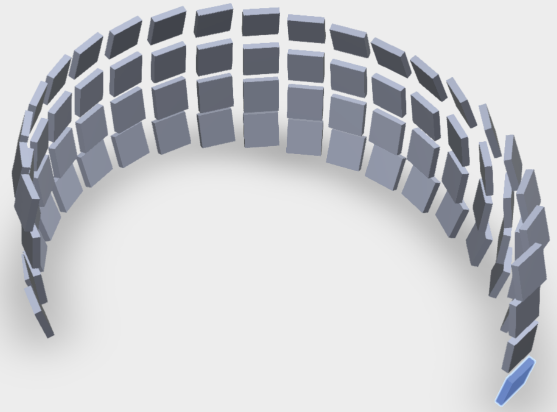
Advantages:

- Real-time 3D X-ray imaging for change from reference images and for motion
- Photon counting stationary spectral multisource CT for dose efficiency and accuracy SPECT-based internal dosimetry.

MGH Multi-Pinhole Cardiac SPECT

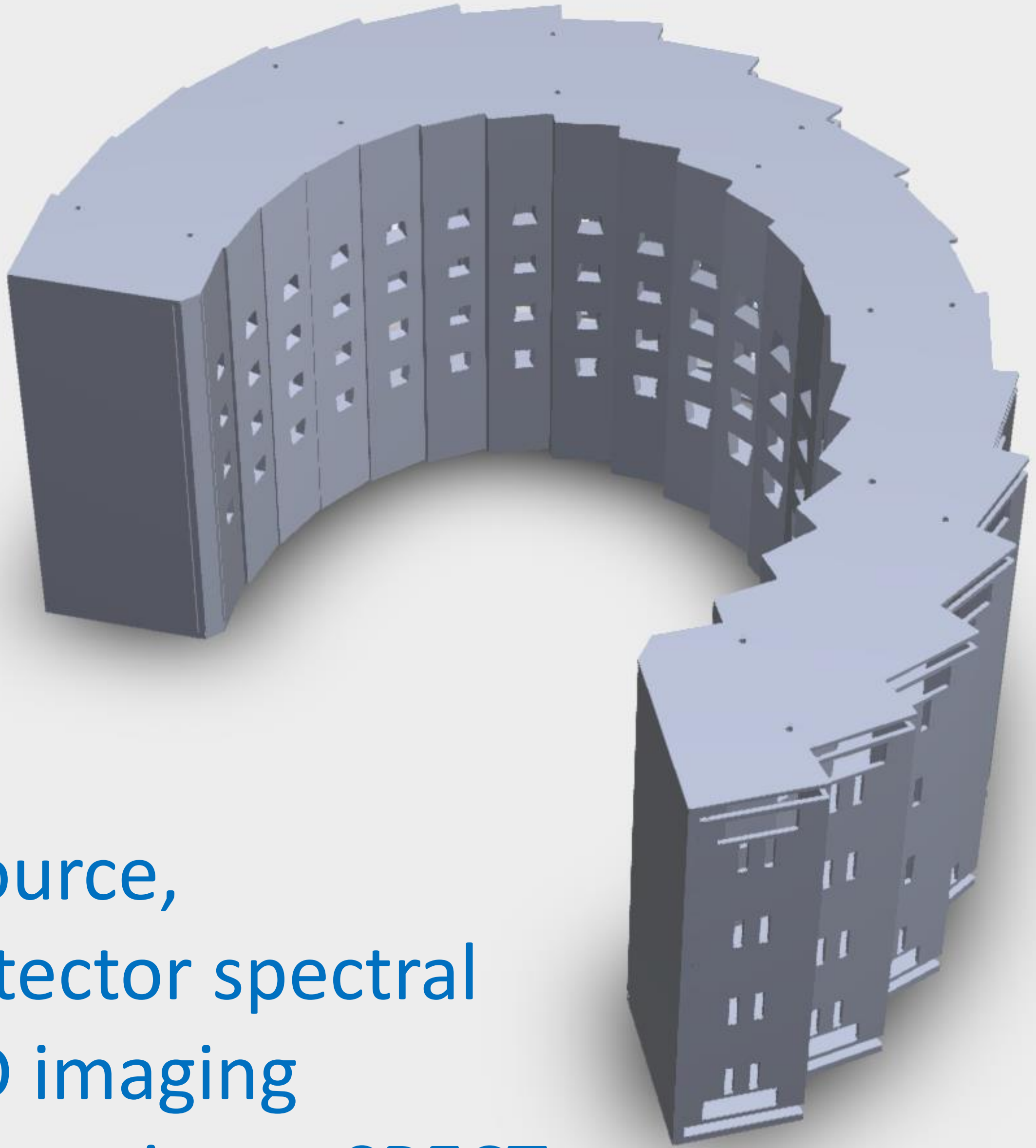


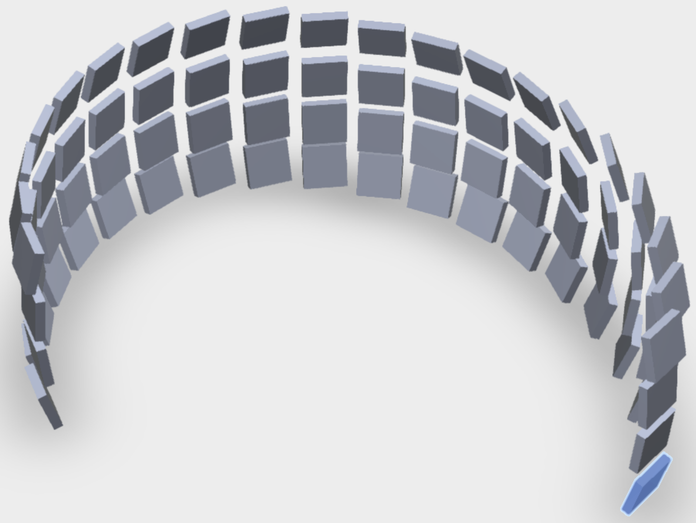
Multi-source, multidetector spectral X-ray 3D imaging system interior to SPECT



MGH/PDSi Multi-Pinhole Theranostic SPECT/CT Concept

MGH Multi-Pinhole Cardiac SPECT





Multi-source, multidetector spectral X-ray 3D imaging system interior to SPECT

Excessive Hand-crafting, or helpful application of deep domain knowledge?

- When designing a neural network architecture for 3D imaging from a set of projection measurements in a stationary multi-pinhole camera (multi-perspective) projection space, i.e. using supervised learning to teach depth perception, is it preferable to:

a) Let the neural net make associations to infer a 3D volumetric image without prior knowledge of the geometric arrangement of the camera array (presumed fixed), or

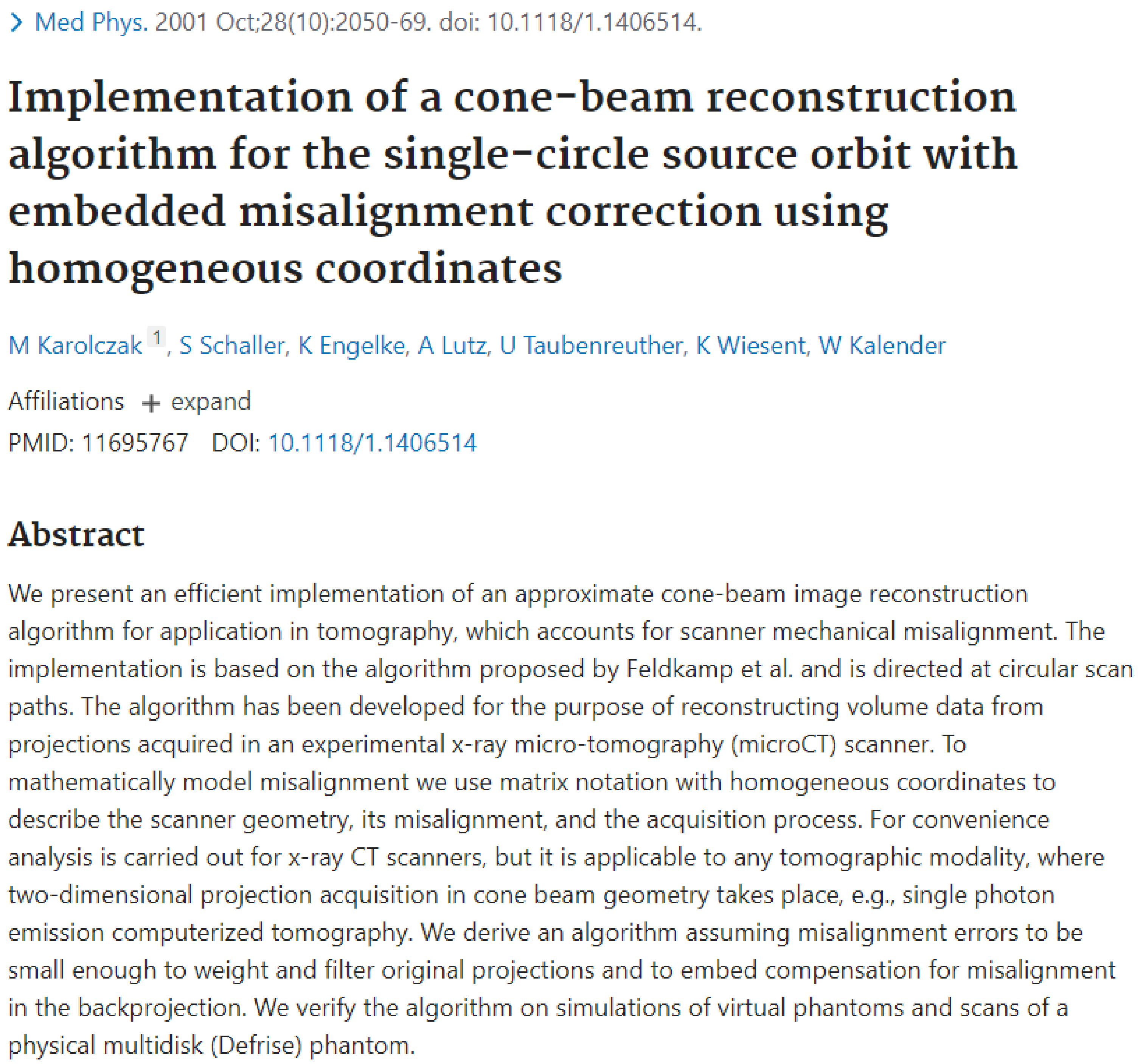
b) Use known multiple camera projection geometry to have each camera "inform" the others of their "opinions" as to the source distribution, based on their learned depth estimation from their perspectives, transformed (using an Inverse Projection Transform and a Projection Transform) to the frame of reference of the camera receiving the "opinion" image estimates?

Imagine a stationary, multi-camera projective imaging system where:

- Each pinhole camera collects integrals of observed source activity (in the emission case) or accumulated attenuation (in the transmission case) along a set of lines of response according to a Projection Transform. There are [M] cameras.
- For the ideal case of point-like pinholes: for any given source distribution we can use homogeneous coordinates and an Inverse Projection Transform to map rays through the image volume which contribute to each camera’s view of the scene from its perspective. [N,N] images. The projection tensor is [M,N,N], with M views.
- We can sum over all cameras and assign equal probability to all possible depth positions along each of the lines of response rays (i.e. summed Inverse Projection Transformed projections for 3-D back-projection, introducing parallax to encode depth distributions).
- Using supervised learning and Projection Transformed data tensors from known truth volumetric data as labels, we can train a set of M 3D U-Net Neural Networks on known truth back-projections to generate M 3D volumetric image estimates, and their mean. This is a learned back-projection filtering algorithm with the geometry embedded.
- Another block of U-Net Convolution Neural Networks could then be used to synthesize the set of M 3D volumetric images and to differentially weight the contribution of each voxel differently for the different cameras based on the image content.

Application of homogeneous coordinates to 3D CT reconstruction is far from new – Willi Kalender was and is a name to conjure with in the CT reconstruction expert community

Karolczak, M., Schaller, S., Engelke, K., Lutz, A., Taubenreuther, U., Wiesent, K. and Kalender, W., 2001. Implementation of a cone-beam reconstruction algorithm for the single-circle source orbit with embedded misalignment correction using homogeneous coordinates. *Medical physics*, *28*(10), pp.2050-2069.



› Med Phys. 2001 Oct;28(10):2050-69. doi: 10.1118/1.1406514.

Implementation of a cone-beam reconstruction algorithm for the single-circle source orbit with embedded misalignment correction using homogeneous coordinates

M Karolczak[1], S Schaller, K Engelke, A Lutz, U Taubenreuther, K Wiesent, W Kalender

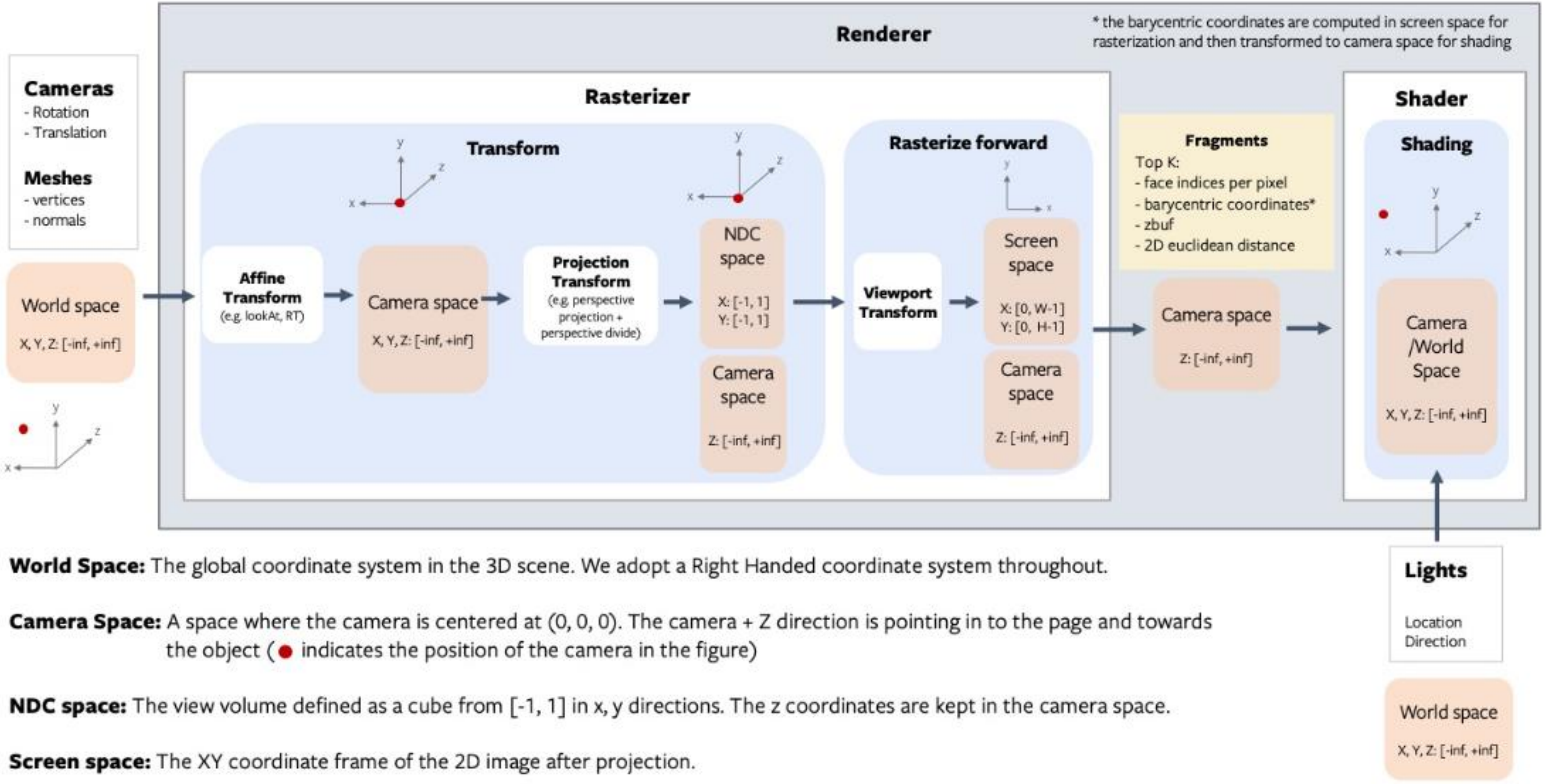
Affiliations + expand

PMID: 11695767 DOI: 10.1118/1.1406514

Abstract

We present an efficient implementation of an approximate cone-beam image reconstruction algorithm for application in tomography, which accounts for scanner mechanical misalignment. The implementation is based on the algorithm proposed by Feldkamp et al. and is directed at circular scan paths. The algorithm has been developed for the purpose of reconstructing volume data from projections acquired in an experimental x-ray micro-tomography (microCT) scanner. To mathematically model misalignment we use matrix notation with homogeneous coordinates to describe the scanner geometry, its misalignment, and the acquisition process. For convenience analysis is carried out for x-ray CT scanners, but it is applicable to any tomographic modality, where two-dimensional projection acquisition in cone beam geometry takes place, e.g., single photon emission computerized tomography. We derive an algorithm assuming misalignment errors to be small enough to weight and filter original projections and to embed compensation for misalignment in the backprojection. We verify the algorithm on simulations of virtual phantoms and scans of a physical multidisk (Defrise) phantom.

https://github.com/facebookresearch/pytorch3d/blob/main/docs/notes/renderer_getting_started.md

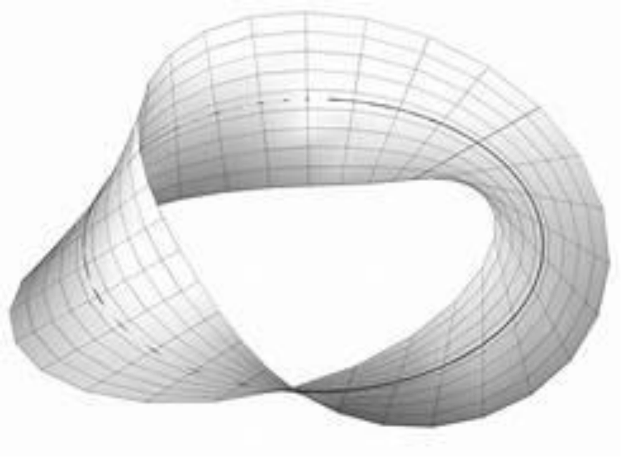


World Space: The global coordinate system in the 3D scene. We adopt a Right Handed coordinate system throughout.

Camera Space: A space where the camera is centered at (0, 0, 0). The camera + Z direction is pointing in to the page and towards the object (● indicates the position of the camera in the figure)

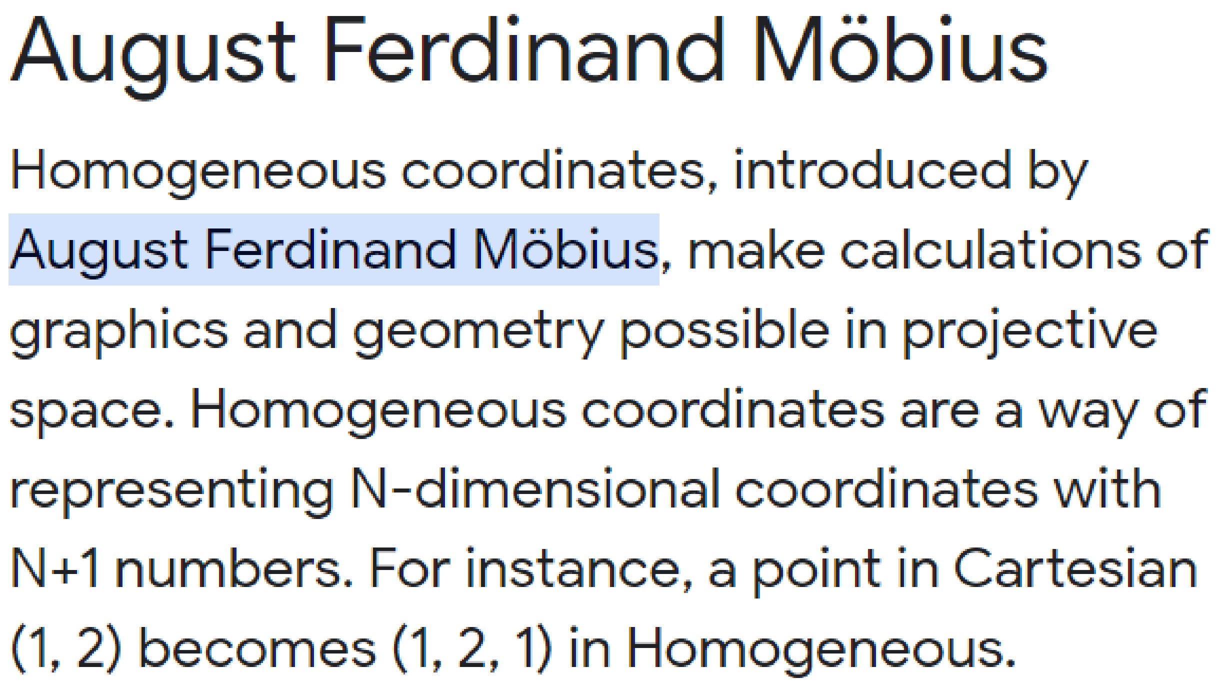
NDC space: The view volume defined as a cube from [-1, 1] in x, y directions. The z coordinates are kept in the camera space.

Screen space: The XY coordinate frame of the 2D image after projection.



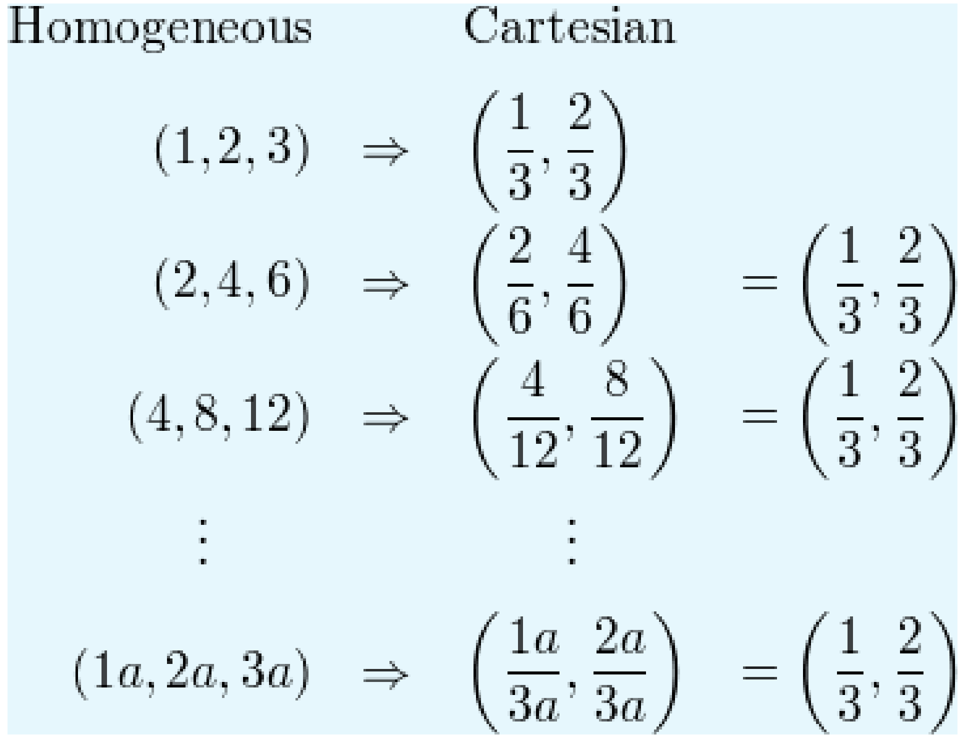
What are Homogeneous Coordinates?

In mathematics, homogeneous coordinates or projective coordinates, introduced by August Ferdinand Möbius in his 1827 work Der barycentrische Calcul, are a system of coordinates used in projective geometry, just as Cartesian coordinates are used in Euclidean geometry.



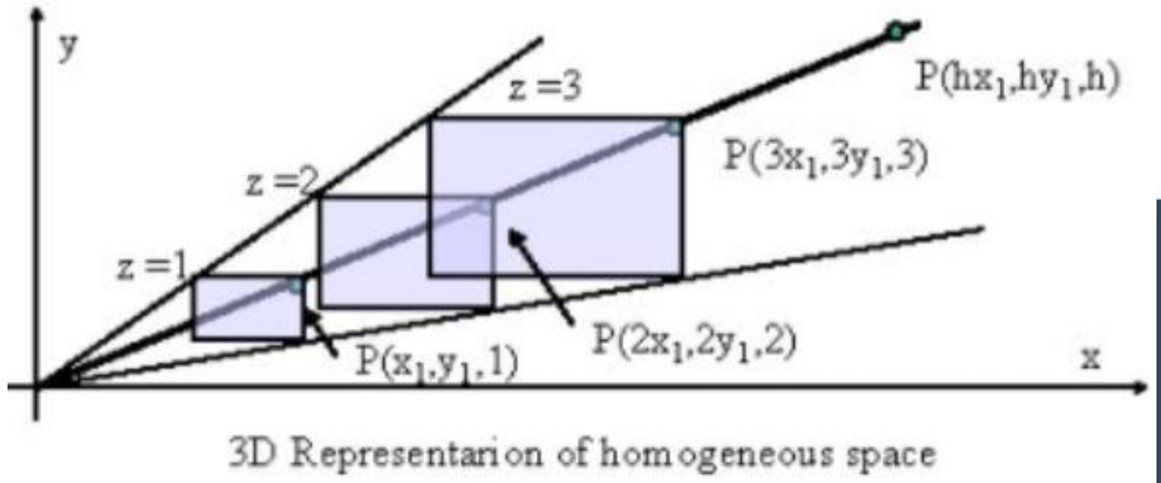
August Ferdinand Möbius

Homogeneous coordinates, introduced by August Ferdinand Möbius, make calculations of graphics and geometry possible in projective space. Homogeneous coordinates are a way of representing N-dimensional coordinates with N+1 numbers. For instance, a point in Cartesian (1, 2) becomes (1, 2, 1) in Homogeneous.

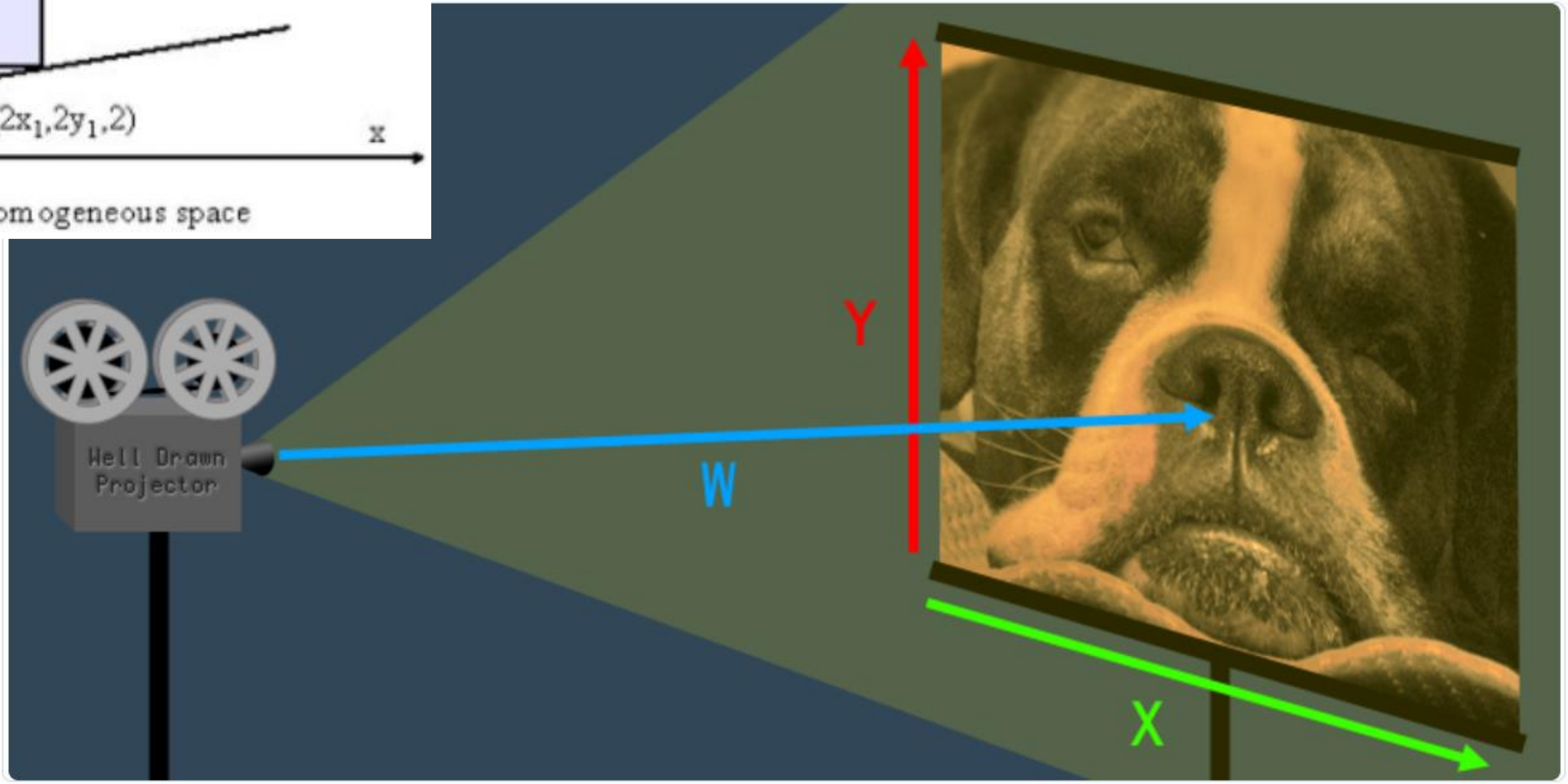


Homogeneous		Cartesian		
$(1,2,3)$	$\Rightarrow$	$\left(\frac{1}{3},\frac{2}{3}\right)$		
$(2,4,6)$	$\Rightarrow$	$\left(\frac{2}{6},\frac{4}{6}\right)$	$=$	$\left(\frac{1}{3},\frac{2}{3}\right)$
$(4,8,12)$	$\Rightarrow$	$\left(\frac{4}{12},\frac{8}{12}\right)$	$=$	$\left(\frac{1}{3},\frac{2}{3}\right)$
$\vdots$		$\vdots$		
$(1a,2a,3a)$	$\Rightarrow$	$\left(\frac{1a}{3a},\frac{2a}{3a}\right)$	$=$	$\left(\frac{1}{3},\frac{2}{3}\right)$

https://www.songho.ca/math/homogeneous/homogeneous.html https://www.tomdalling.com/blog/modern-opengl/explaining-homogenous-coordinates-and-projective-geometry/



Points in the 3D volume being imaged by the pinhole camera scale in their projected image by a factor which is inversely proportional to the distance from the pinhole to a plane perpendicular to the principal axis passing through the point.



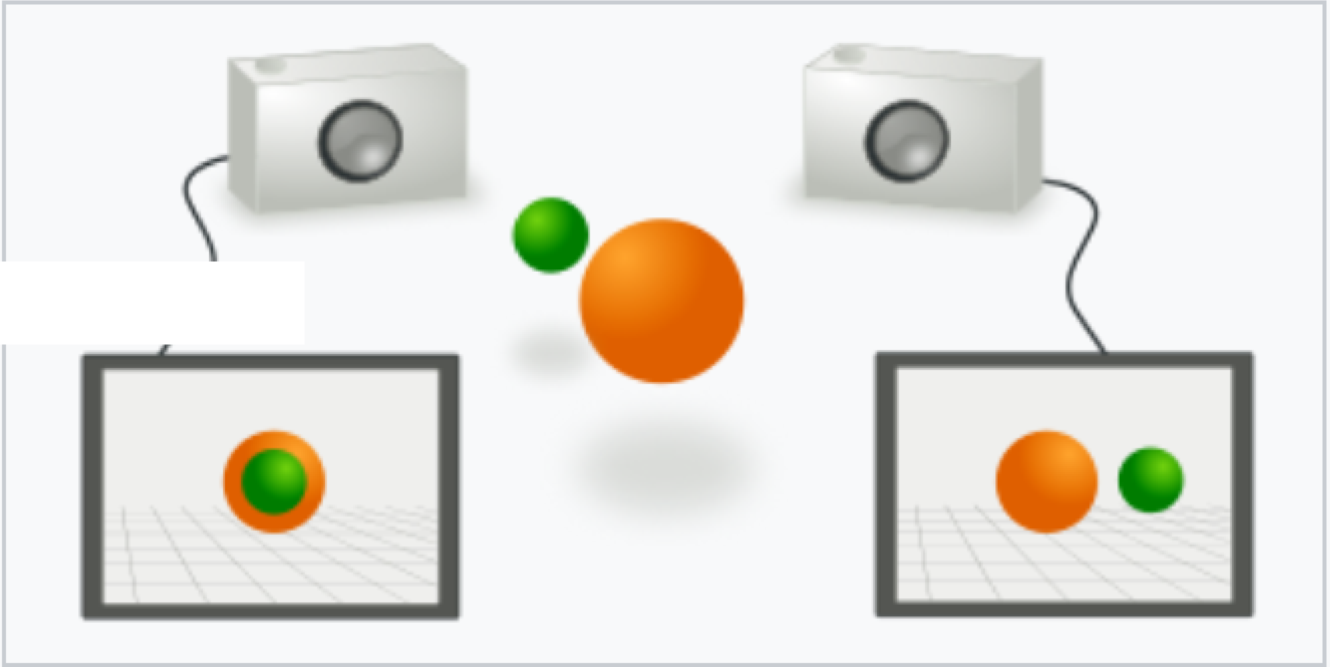
So what does the W dimension do, exactly? Imagine what would happen to the 2D image if you increased or decreased W – that is, if you increased or

The value of W affects the size (a.k.a. scale) of the image.

https://en.wikipedia.org/wiki/Epipolar_geometry

Epipolar geometry

Epipolar geometry is the geometry of stereo vision. When two cameras view a 3D scene from two distinct positions, there are a number of geometric relations between the 3D points and their projections onto the 2D images that lead to constraints between the image points. These relations are derived based on the assumption that the cameras can be approximated by the pinhole camera model.



Typical use case for epipolar geometry

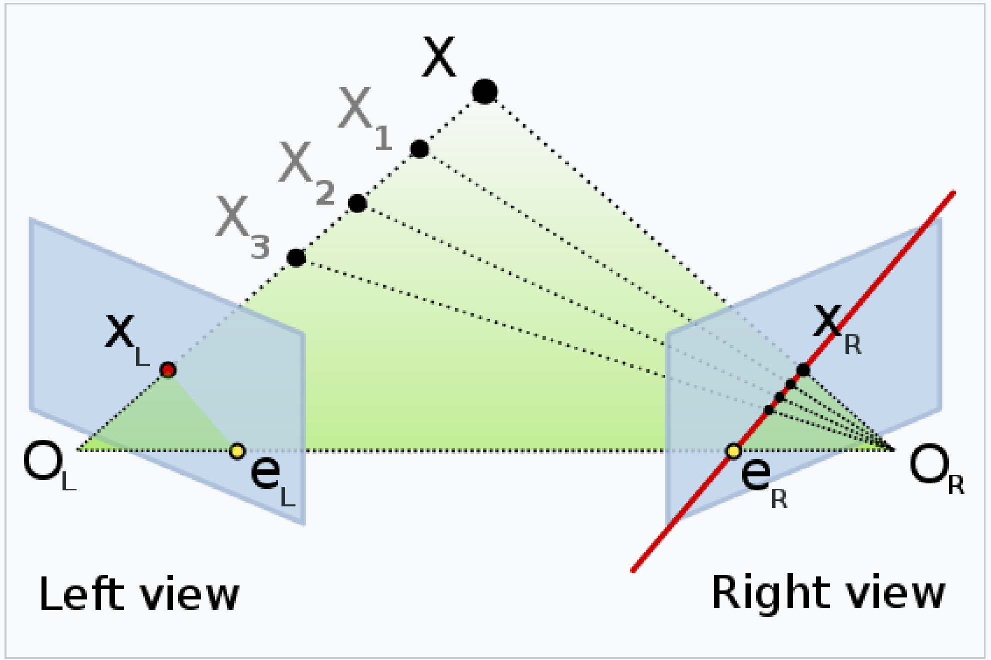
Two cameras take a picture of the same scene from different points of view. The epipolar geometry then describes the relation between the two resulting views.

Definitions [edit]

https://en.wikipedia.org/wiki/Epipolar_geometry

The figure below depicts two pinhole cameras looking at point **X**. In real cameras, the image plane is actually behind the focal center, and produces an image that is symmetric about the focal center of the lens. Here, however, the problem is simplified by placing a *virtual image plane* in front of the focal center i.e. optical center of each camera lens to produce an image not transformed by the symmetry. $\mathbf{O}_L$ and $\mathbf{O}_R$ represent the centers of symmetry of the two cameras lenses. **X** represents the point of interest in both cameras. Points $\mathbf{x}_L$ and $\mathbf{x}_R$ are the projections of point **X** onto the image planes.

Each camera captures a 2D image of the 3D world. This conversion from 3D to 2D is referred to as a perspective projection and is described by the pinhole camera model. It is common to model this projection operation by rays that emanate from the camera, passing through its focal center. Each emanating ray corresponds to a single point in the image.



Epipolar geometry

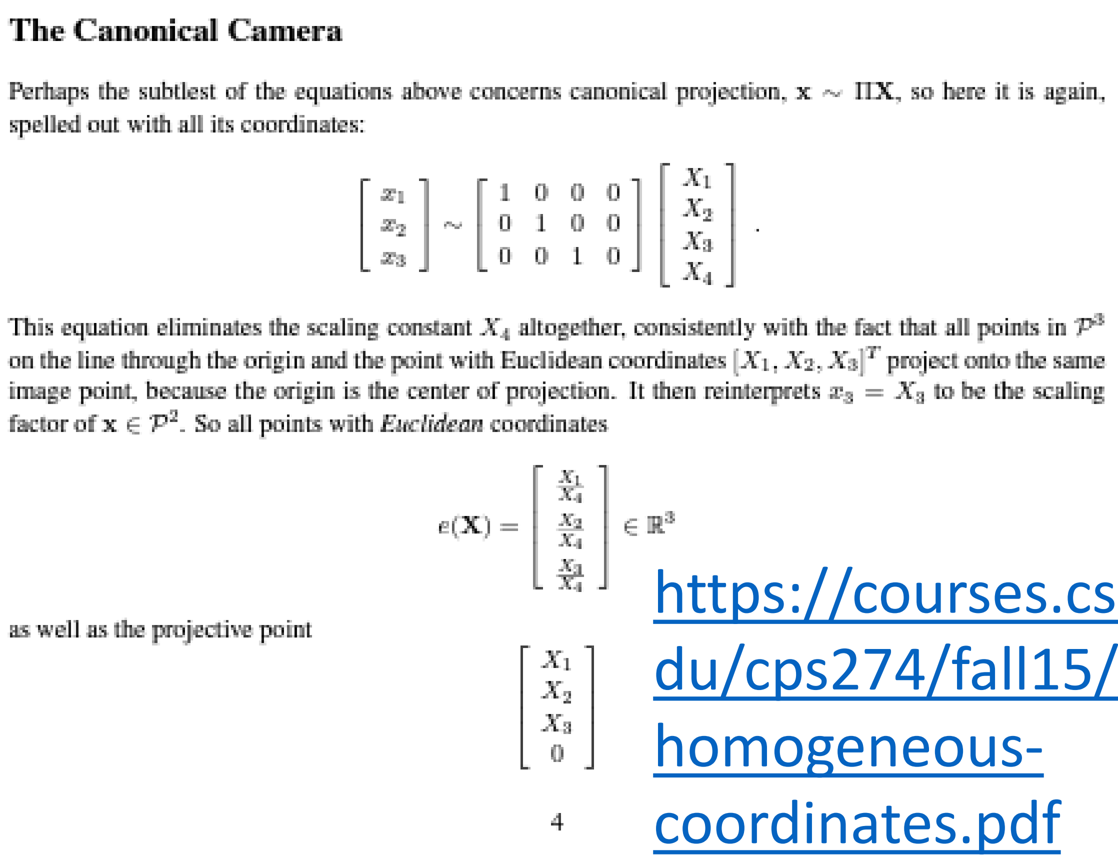
Epipole or epipolar point [edit]

Since the optical centers of the cameras lenses are distinct, each center projects onto a distinct point into the other camera's image plane. These two image points, denoted by $\mathbf{e}_L$ and $\mathbf{e}_R$, are called *epipoles* or *epipolar points*. Both epipoles $\mathbf{e}_L$ and $\mathbf{e}_R$ in their respective image planes and both optical centers $\mathbf{O}_L$ and $\mathbf{O}_R$ lie on a single 3D line.

If the points $\mathbf{x}_L$ and $\mathbf{x}_R$ are known, their projection lines are also known. If the two image points correspond to the same 3D point **X** the projection lines must intersect precisely at **X**. This means that **X** can be calculated from the coordinates of the two image points, a process called *triangulation*

The main advantage of using homogeneous coordinates is that both affine transformations and projections become linear.

Straight lines in the image volume map onto straight lines in the homogeneous coordinate space, for each perspective.



The Canonical Camera

Perhaps the subtlest of the equations above concerns canonical projection, $\mathbf{x} \sim \Pi\mathbf{X}$, so here it is again, spelled out with all its coordinates:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \sim \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} .$$

This equation eliminates the scaling constant X_4 altogether, consistently with the fact that all points in $\mathcal{P}^3$ on the line through the origin and the point with Euclidean coordinates $[X_1, X_2, X_3]^T$ project onto the same image point, because the origin is the center of projection. It then reinterprets $x_3 = X_3$ to be the scaling factor of $\mathbf{x} \in \mathcal{P}^2$. So all points with *Euclidean* coordinates

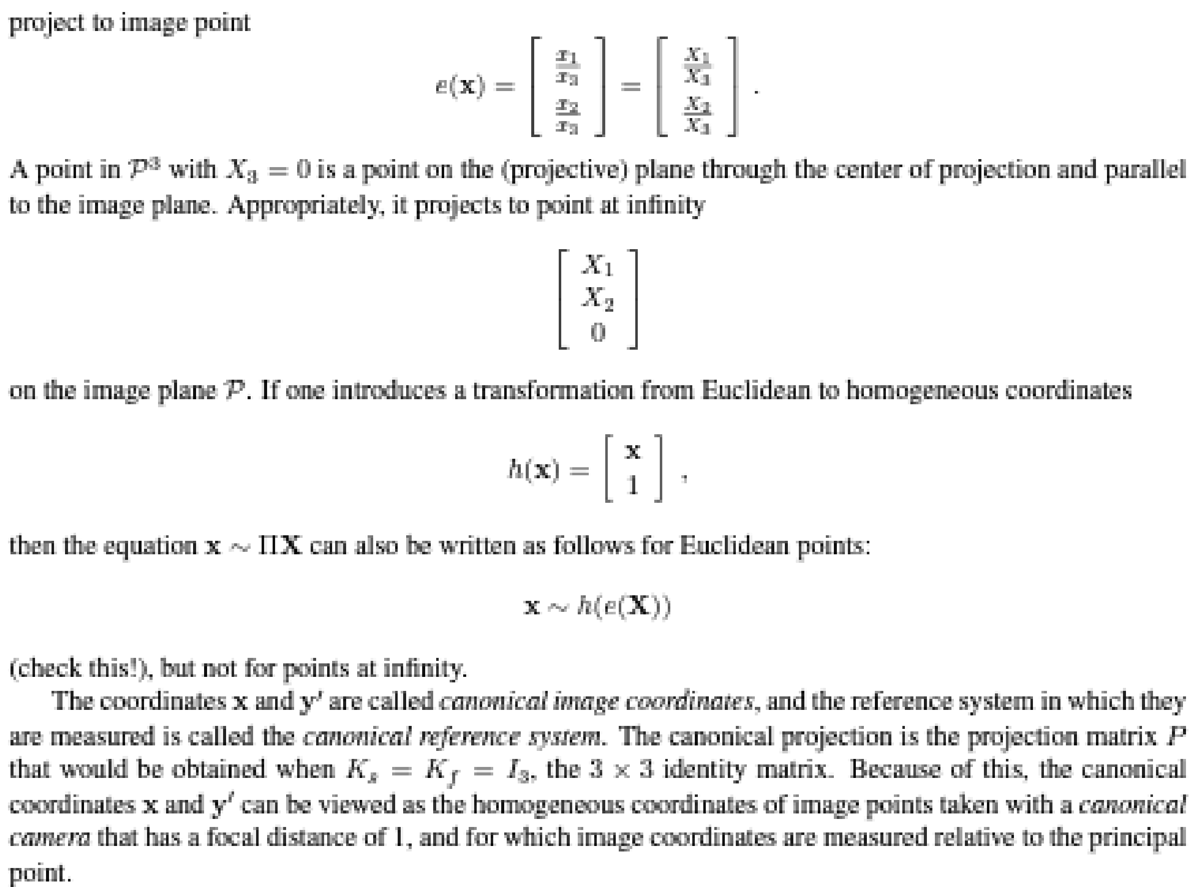
$$e(\mathbf{X}) = \begin{bmatrix} \frac{X_1}{X_4} \\ \frac{X_2}{X_4} \\ \frac{X_3}{X_4} \end{bmatrix} \in \mathbb{R}^3$$

as well as the projective point

$$\begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ 0 \end{bmatrix}$$

4

https://courses.cs.duke.edu/cps274/fall15/notes/homogeneous-coordinates.pdf



project to image point

$$e(\mathbf{x}) = \begin{bmatrix} \frac{x_1}{x_3} \\ \frac{x_2}{x_3} \end{bmatrix} = \begin{bmatrix} \frac{X_1}{X_3} \\ \frac{X_2}{X_3} \end{bmatrix} .$$

A point in $\mathcal{P}^3$ with $X_3 = 0$ is a point on the (projective) plane through the center of projection and parallel to the image plane. Appropriately, it projects to point at infinity

$$\begin{bmatrix} X_1 \\ X_2 \\ 0 \end{bmatrix}$$

on the image plane $\mathcal{P}$. If one introduces a transformation from Euclidean to homogeneous coordinates

$$h(\mathbf{x}) = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} ,$$

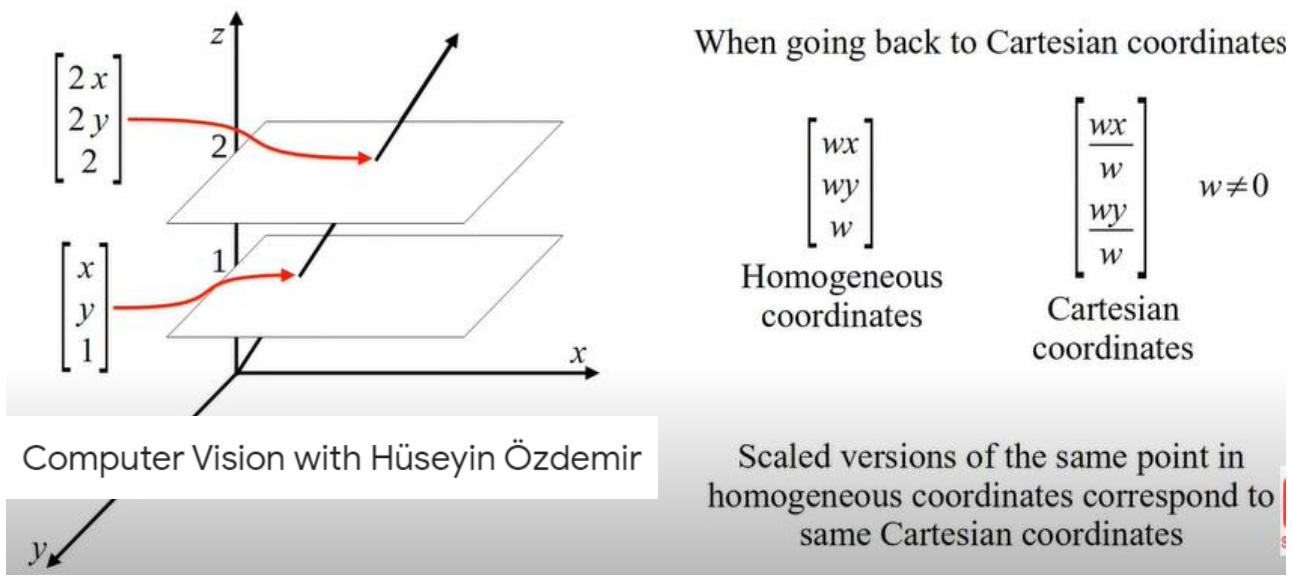
then the equation $\mathbf{x} \sim \Pi\mathbf{X}$ can also be written as follows for Euclidean points:

$$\mathbf{x} \sim h(e(\mathbf{X}))$$

(check this!), but not for points at infinity.

The coordinates $\mathbf{x}$ and $\mathbf{y}'$ are called *canonical image coordinates*, and the reference system in which they are measured is called the *canonical reference system*. The canonical projection is the projection matrix P that would be obtained when $K_s = K_f = I_3$, the 3×3 identity matrix. Because of this, the canonical coordinates $\mathbf{x}$ and $\mathbf{y}'$ can be viewed as the homogeneous coordinates of image points taken with a *canonical camera* that has a focal distance of 1, and for which image coordinates are measured relative to the principal point.

https://www.youtube.com/watch?v=2BIzmFD_pRQ



$\begin{bmatrix} 2x \\ 2y \\ 2 \end{bmatrix}$ $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

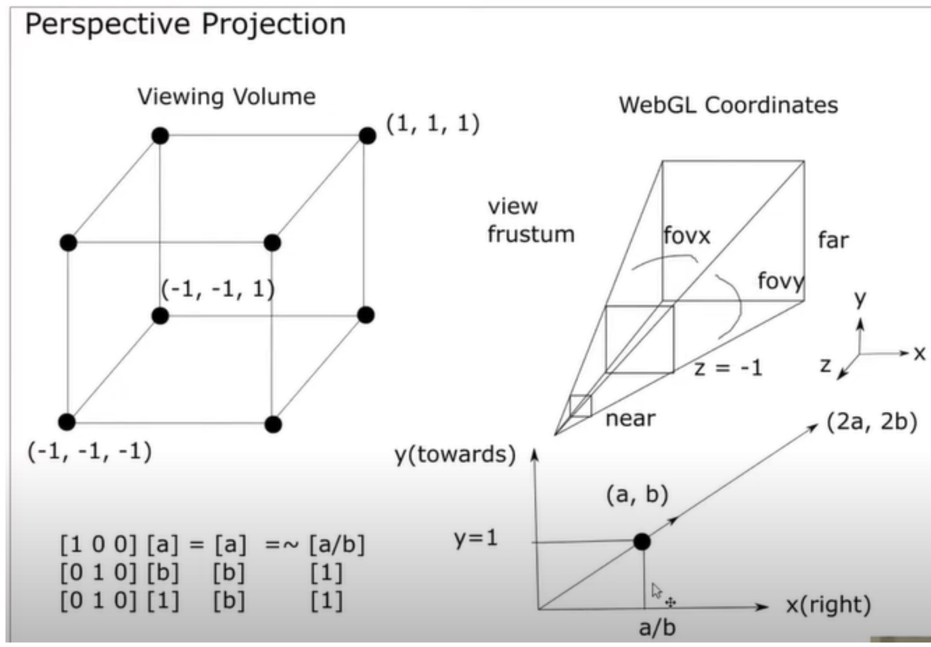
When going back to Cartesian coordinates

$\begin{bmatrix} wx \\ wy \\ w \end{bmatrix}$ Homogeneous coordinates

$\begin{bmatrix} \frac{wx}{w} \\ \frac{wy}{w} \end{bmatrix}$ Cartesian coordinates $w \neq 0$

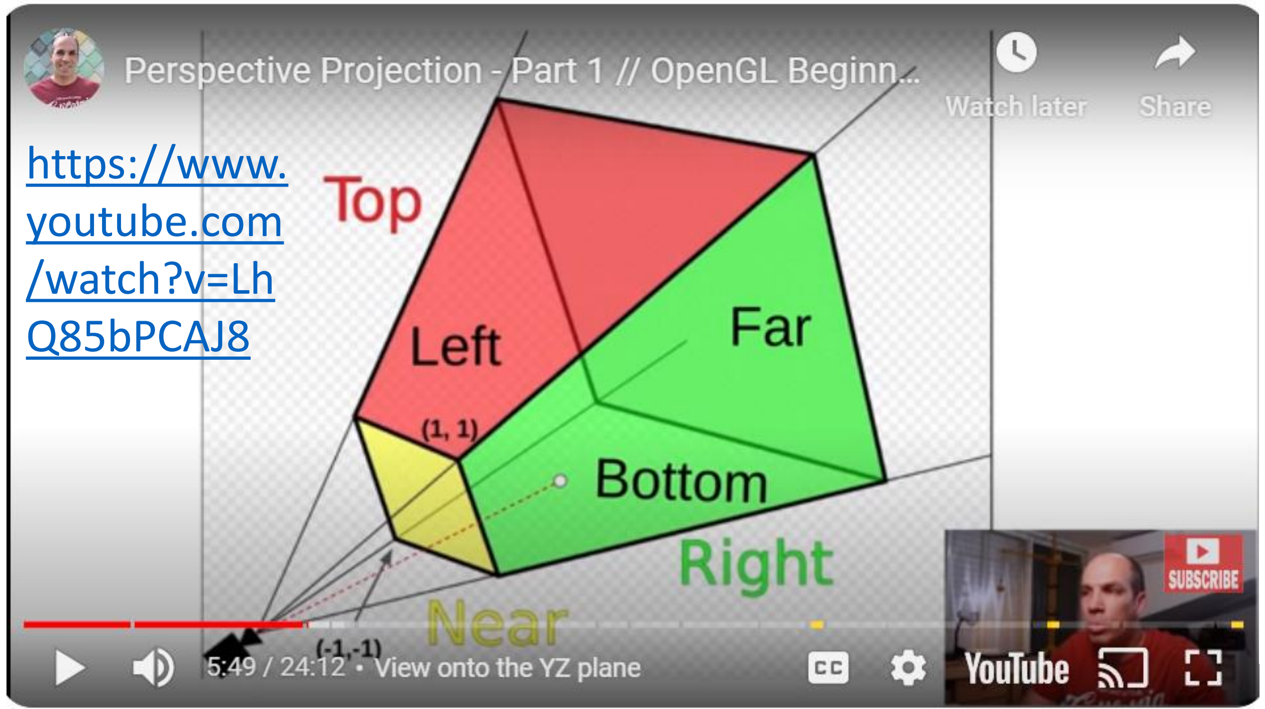
Scaled versions of the same point in homogeneous coordinates correspond to same Cartesian coordinates

https://www.youtube.com/watch?v=vhrJBrQRucw

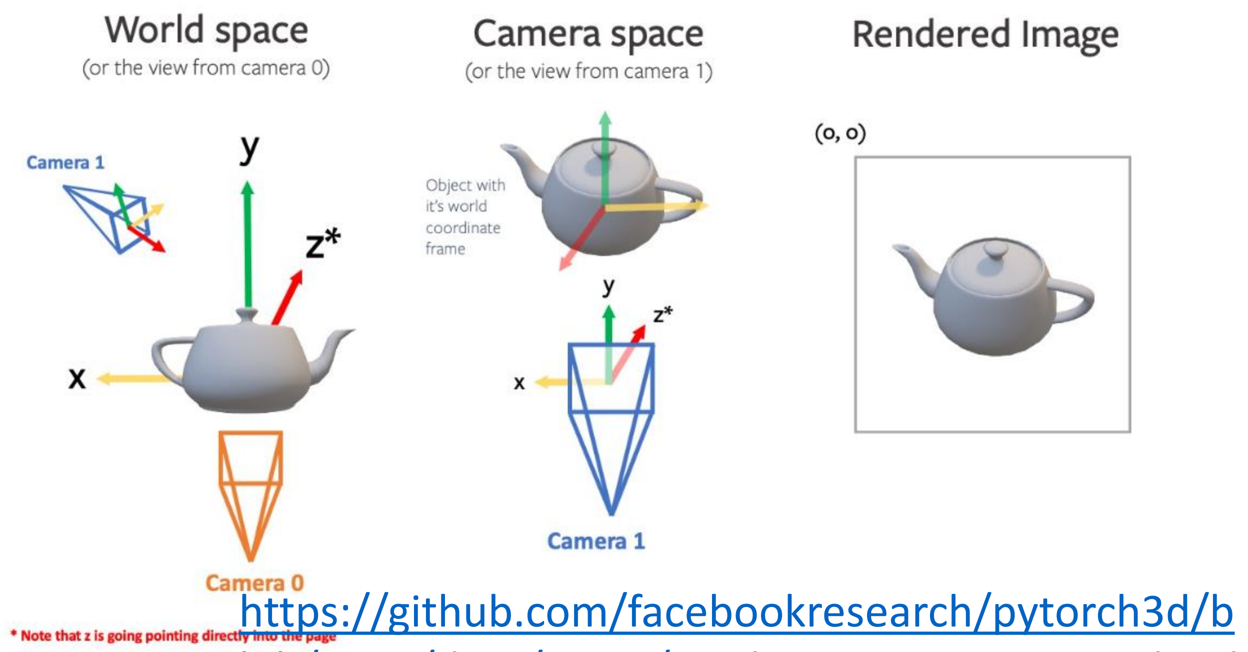


```
[1 0 0] [a] = [a]  =~ [a/b]
[0 1 0] [b]   [b]      [1]
[0 1 0] [1]   [b]      [1]
```

CS 476 Module 13: Perspective Projection Matrices


Perspective Projection - Part 1 // OpenGL Beginn...
Watch later
Share
Top
Left
Far
Bottom
Right
Near
(1, 1)
(-1,-1)
SUBSCRIBE
5:49 / 24:12 • View onto the YZ plane
YouTube

https://www.youtube.com/watch?v=LhQ85bPCAJ8


World space
(or the view from camera 0)
Camera space
(or the view from camera 1)
Rendered Image
Camera 1
y
z*
x
Camera 0
Object with it's world coordinate frame
y
z*
x
Camera 1
(o, o)
* Note that z is going pointing directly into the page

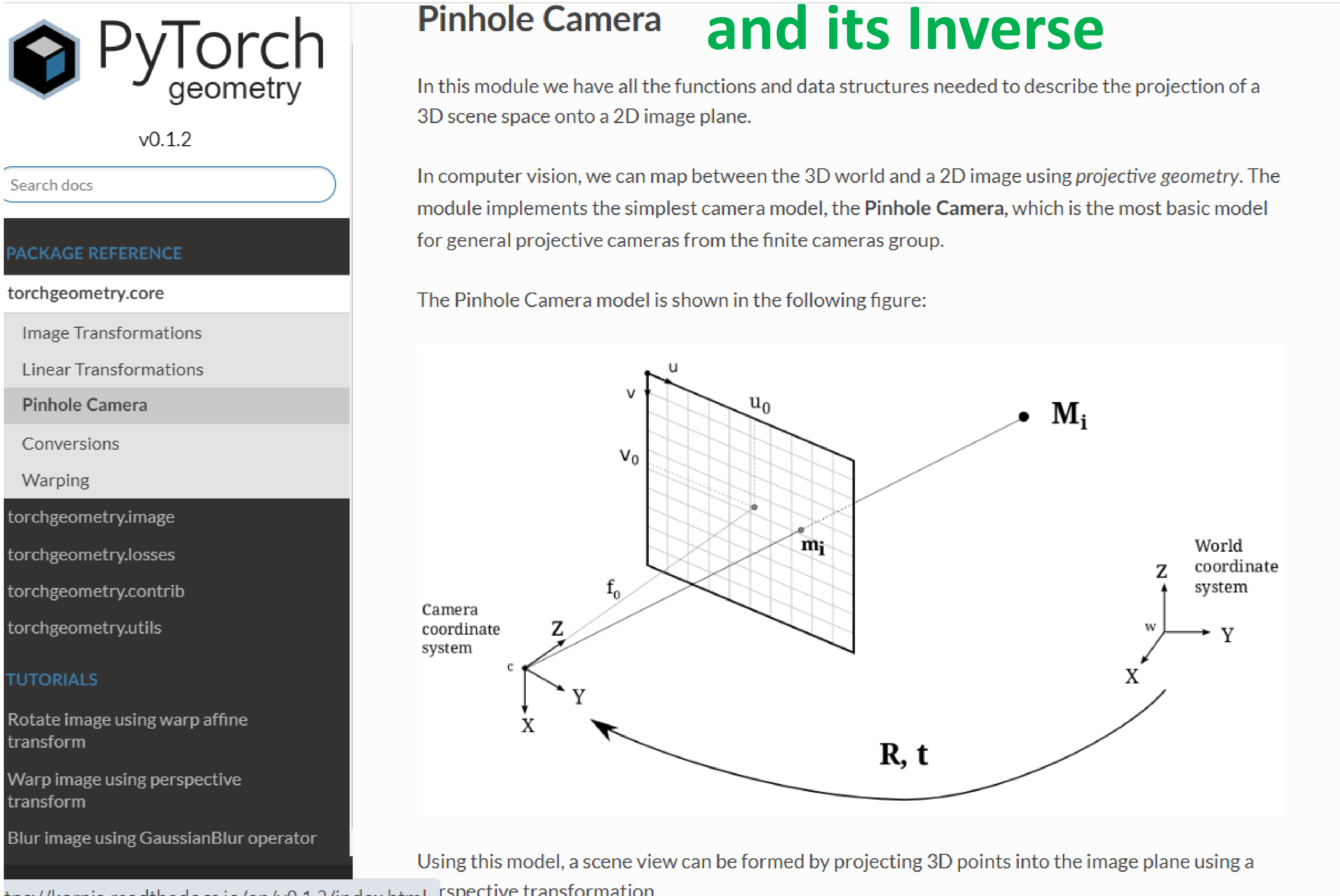
https://github.com/facebookresearch/pytorch3d/blob/main/docs/notes/renderer_getting_started.md

https://kornia.readthedocs.io/en/v0.1.2/pinhole.html

Projection Transform and its Inverse

Each point in the Pinhole Camera 2D projection space [u,v] has a magnification factor k given the depth of field of the source voxel M,

The signal measured at mi is a line integral through a set of voxels in the 3D homogeneous coordinate projection space volume, which is equivalent to the source volume with a known image-to-image transform for each camera



Pinhole Camera

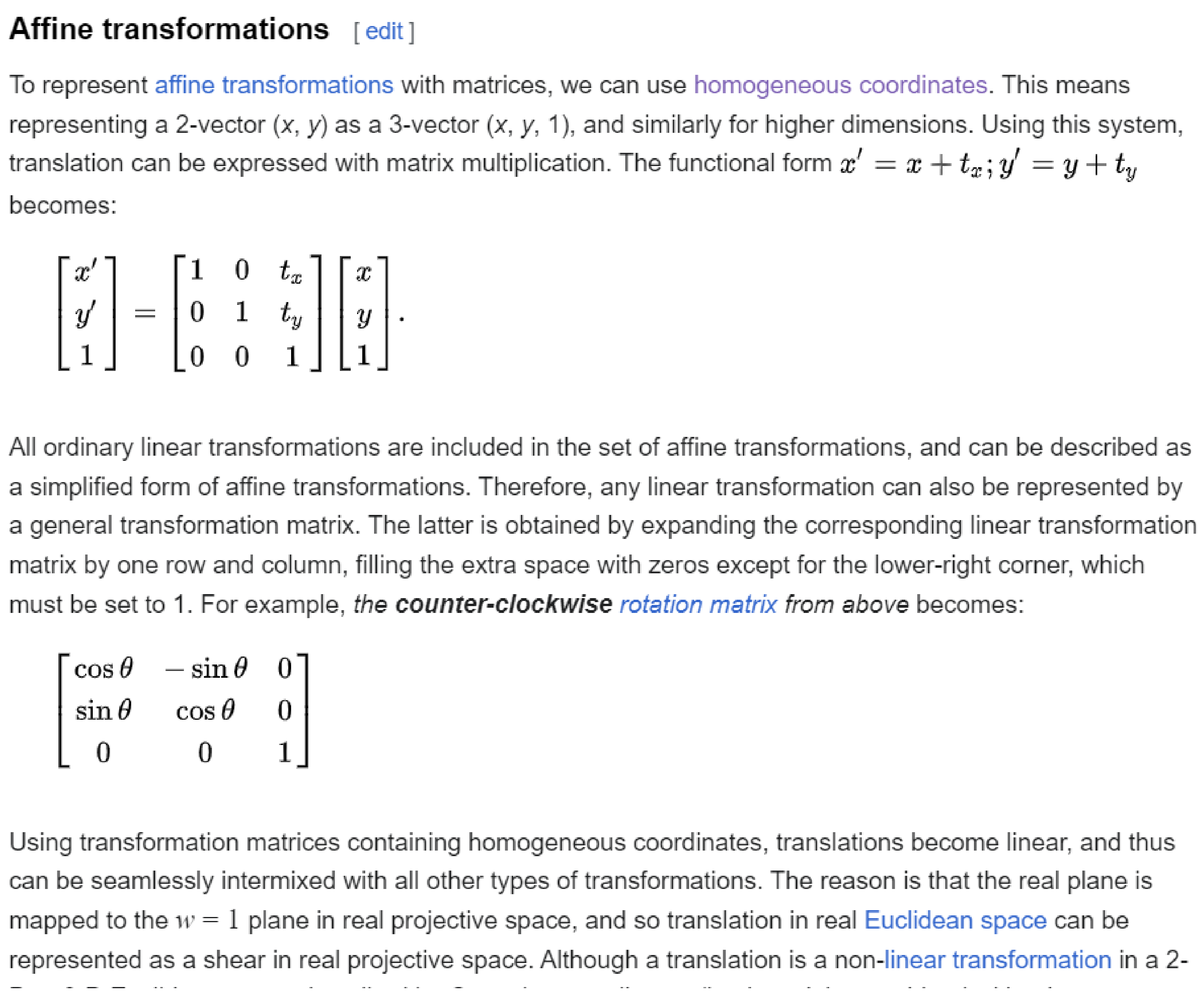
In this module we have all the functions and data structures needed to describe the projection of a 3D scene space onto a 2D image plane.

In computer vision, we can map between the 3D world and a 2D image using *projective geometry*. The module implements the simplest camera model, the **Pinhole Camera**, which is the most basic model for general projective cameras from the finite cameras group.

The Pinhole Camera model is shown in the following figure:

Using this model, a scene view can be formed by projecting 3D points into the image plane using a
rspective transformation

https://en.wikipedia.org/wiki/Transformation_matrix



Affine transformations [edit]

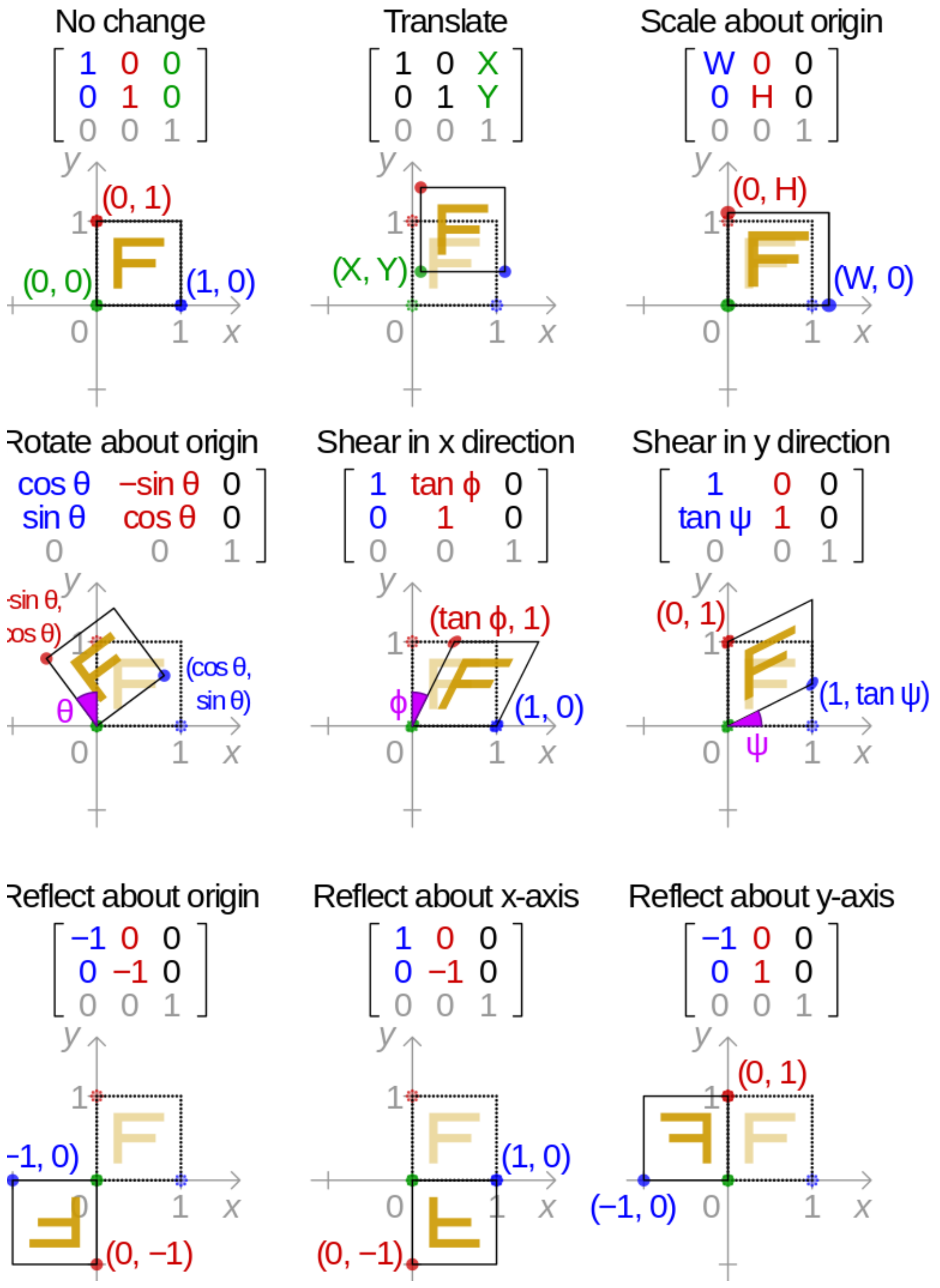
To represent affine transformations with matrices, we can use homogeneous coordinates. This means representing a 2-vector (*x*, *y*) as a 3-vector (*x*, *y*, 1), and similarly for higher dimensions. Using this system, translation can be expressed with matrix multiplication. The functional form $x' = x + t_x; y' = y + t_y$ becomes:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

All ordinary linear transformations are included in the set of affine transformations, and can be described as a simplified form of affine transformations. Therefore, any linear transformation can also be represented by a general transformation matrix. The latter is obtained by expanding the corresponding linear transformation matrix by one row and column, filling the extra space with zeros except for the lower-right corner, which must be set to 1. For example, *the* ***counter-clockwise*** *rotation matrix from above* becomes:

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Using transformation matrices containing homogeneous coordinates, translations become linear, and thus can be seamlessly intermixed with all other types of transformations. The reason is that the real plane is mapped to the $w = 1$ plane in real projective space, and so translation in real Euclidean space can be represented as a shear in real projective space. Although a translation is a non-linear transformation in a 2-


No change
$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
(0, 1)
(0, 0)
(1, 0)
Translate
$\begin{bmatrix} 1 & 0 & X \\ 0 & 1 & Y \\ 0 & 0 & 1 \end{bmatrix}$
(X, Y)
Scale about origin
$\begin{bmatrix} W & 0 & 0 \\ 0 & H & 0 \\ 0 & 0 & 1 \end{bmatrix}$
(0, H)
(W, 0)
Rotate about origin
$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$
(-sin θ, cos θ)
(cos θ, sin θ)
θ
Shear in x direction
$\begin{bmatrix} 1 & \tan\phi & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
(tan ϕ, 1)
ϕ
(1, 0)
Shear in y direction
$\begin{bmatrix} 1 & 0 & 0 \\ \tan\psi & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
(0, 1)
(1, tan ψ)
ψ
Reflect about origin
$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
(-1, 0)
(0, −1)
Reflect about x-axis
$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
(1, 0)
(0, −1)
Reflect about y-axis
$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
(0, 1)
(−1, 0)
x
y
0
1

https://en.wikipedia.org/wiki/Transformation_matrix

Perspective projection [edit]

Main article: Perspective projection

Further information: Pinhole camera model

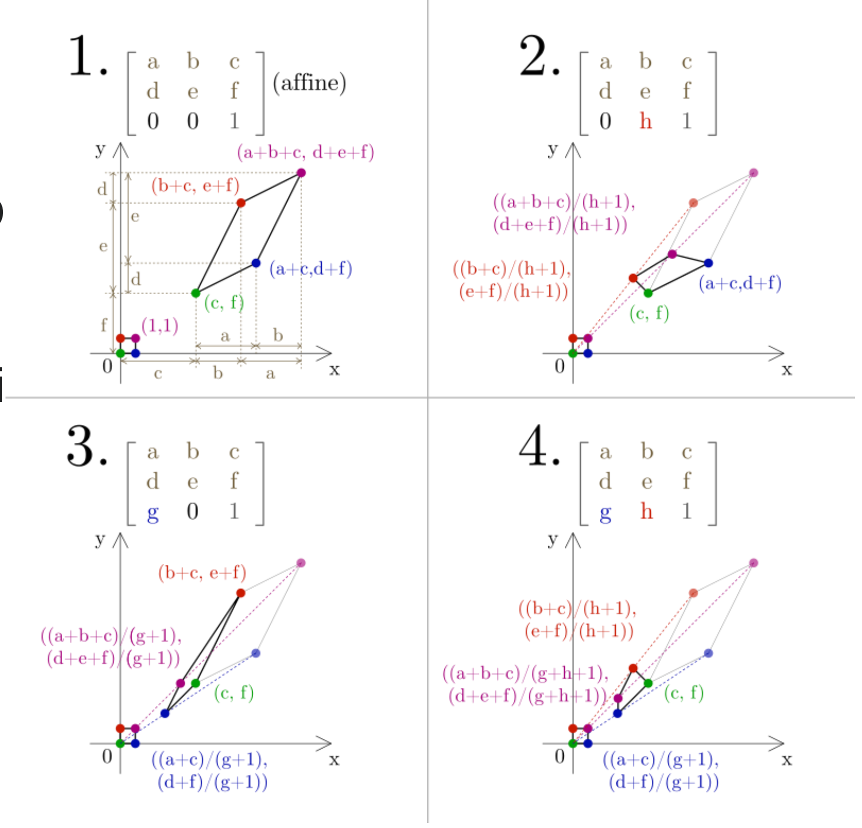
Another type of transformation, of importance in 3D computer graphics, is the perspective projection. Whereas parallel projections are used to project points onto the image plane along parallel lines, the perspective projection projects points onto the image plane along lines that emanate from a single point, called the center of projection. This means that an object has a smaller projection when it is far away from the center of projection and a larger projection when it is closer (see also reciprocal function).

The simplest perspective projection uses the origin as the center of projection, and the plane at $z = 1$ as the image plane. The functional form of this transformation is then $x' = x/z$; $y' = y/z$. We can express this in homogeneous coordinates as:

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z \end{bmatrix}$$

We can also invert the matrix, and can back-project onto the volume along the projection space lines of response

Comparison of the effects of applying 2D affine and perspective transformation matrices on a unit square.

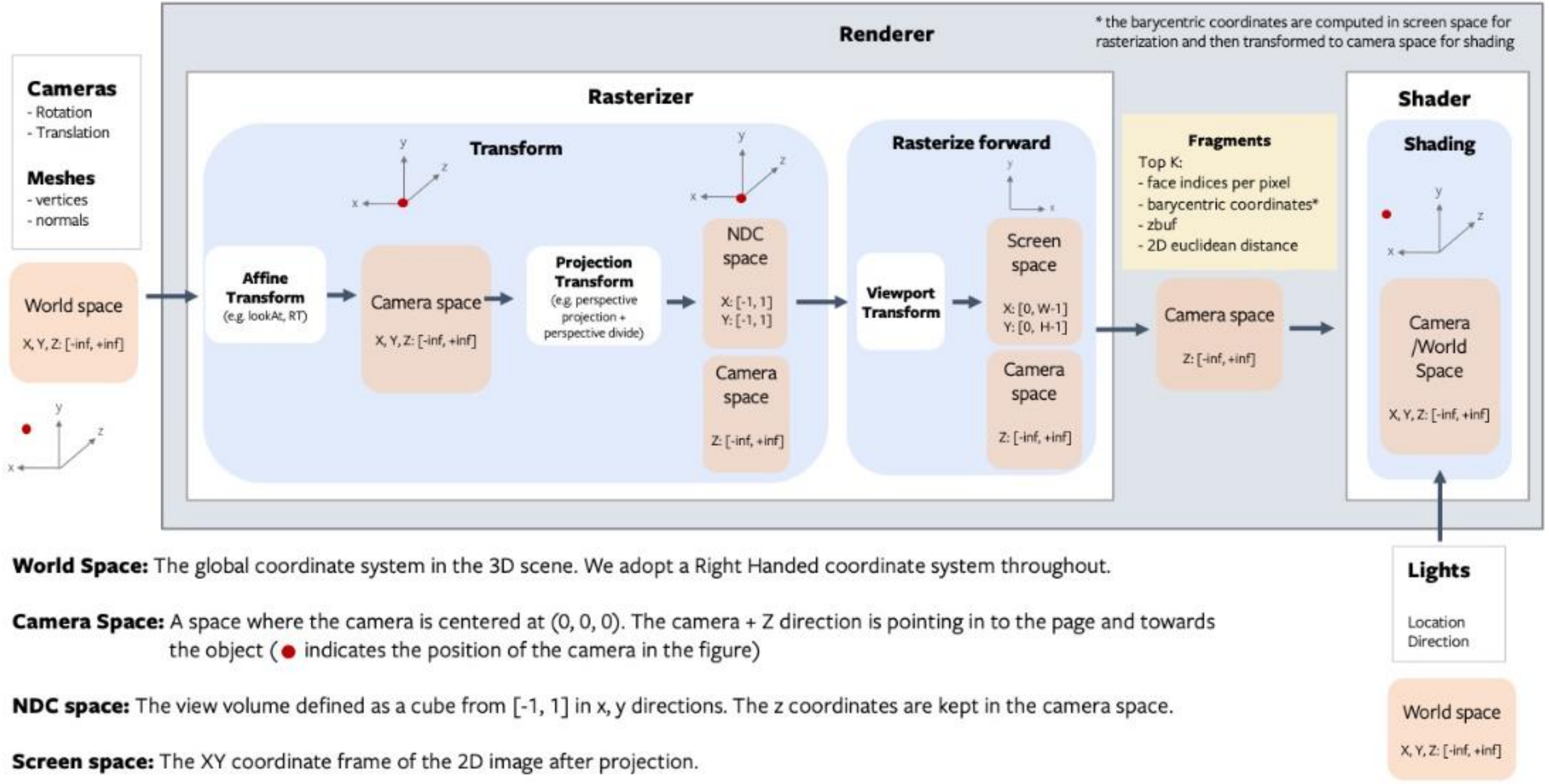

1.
$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$ (affine)
(a+b+c, d+e+f)
(b+c, e+f)
(a+c,d+f)
(c, f)
(1,1)
2.
$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & h & 1 \end{bmatrix}$
((a+b+c)/(h+1), (d+e+f)/(h+1))
((b+c)/(h+1), (e+f)/(h+1))
(a+c,d+f)
(c, f)
3.
$\begin{bmatrix} a & b & c \\ d & e & f \\ g & 0 & 1 \end{bmatrix}$
(b+c, e+f)
((a+b+c)/(g+1), (d+e+f)/(g+1))
(c, f)
((a+c)/(g+1), (d+f)/(g+1))
4.
$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix}$
((b+c)/(h+1), (e+f)/(h+1))
((a+b+c)/(g+h+1), (d+e+f)/(g+h+1))
(c, f)
((a+c)/(g+1), (d+f)/(g+1))
x
y
0

After carrying out the matrix multiplication, the homogeneous component w_c will be equal to the value of z and the other three will not change. Therefore, to map back into the real plane we must perform the **homogeneous divide** or **perspective divide** by dividing each component by w_c:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \frac{1}{w_c} \begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} = \begin{bmatrix} x/z \\ y/z \\ 1 \\ 1 \end{bmatrix}$$

More complicated perspective projections can be composed by combining this one with rotations, scales, translations, and shears to move the image plane and center of projection wherever they are desired.

https://github.com/facebookresearch/pytorch3d/blob/main/docs/notes/renderer_getting_started.md



World Space: The global coordinate system in the 3D scene. We adopt a Right Handed coordinate system throughout.

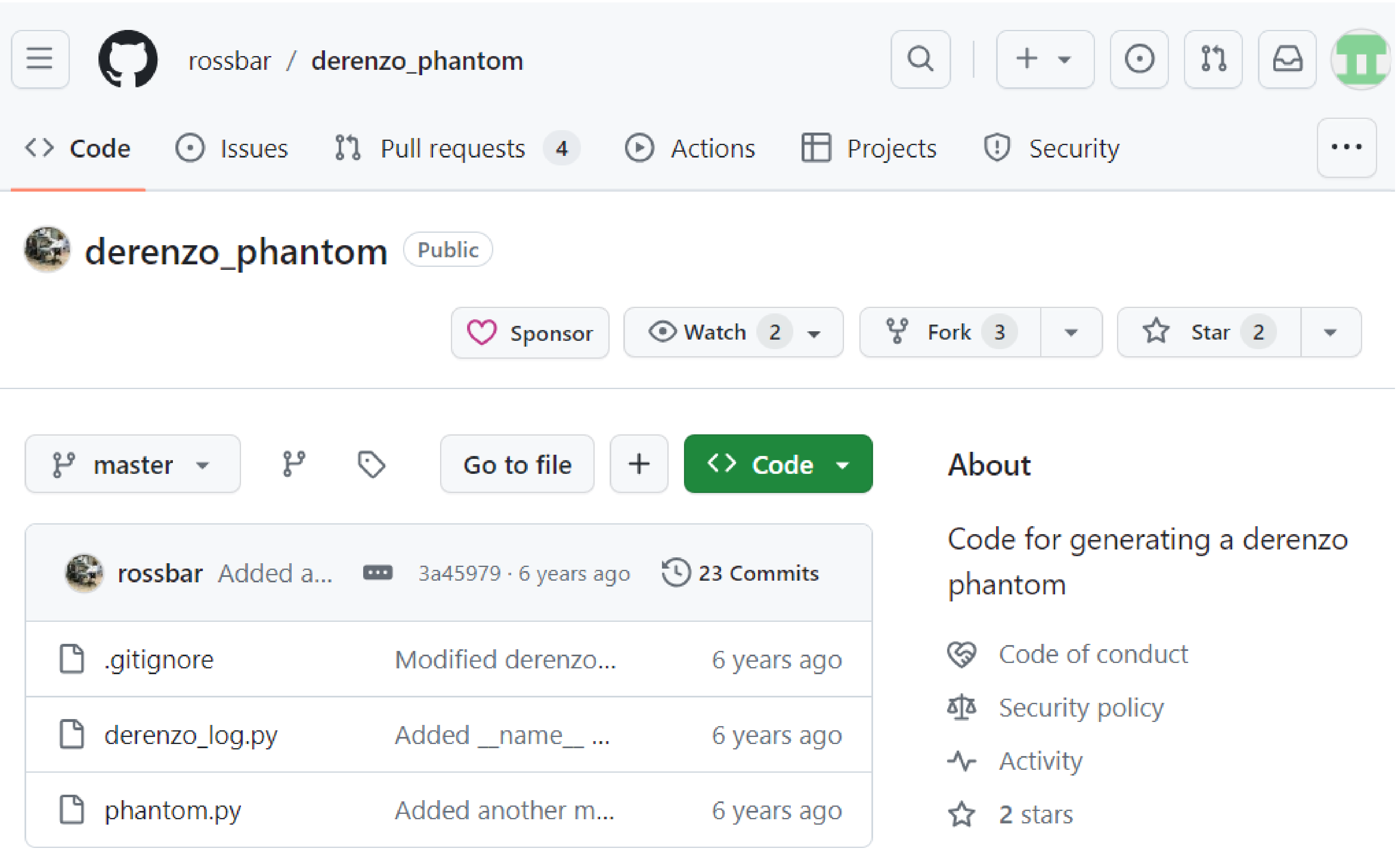
Camera Space: A space where the camera is centered at (0, 0, 0). The camera + Z direction is pointing in to the page and towards the object (● indicates the position of the camera in the figure)

NDC space: The view volume defined as a cube from [-1, 1] in x, y directions. The z coordinates are kept in the camera space.

Screen space: The XY coordinate frame of the 2D image after projection.

Found Open Source on Github:

Python code to generate derenzo phantom displays and GEANT macro inputs for resolution phantoms of arbitrary diameter and well size in each sector, with patterns/coordinates auto-generated



rossbar / derenzo_phantom

derenzo_phantom Public

rossbar Added a... 3a45979 · 6 years ago 23 Commits

.gitignore	Modified derenzo...	6 years ago
derenzo_log.py	Added __name__ ...	6 years ago
phantom.py	Added another m...	6 years ago

About

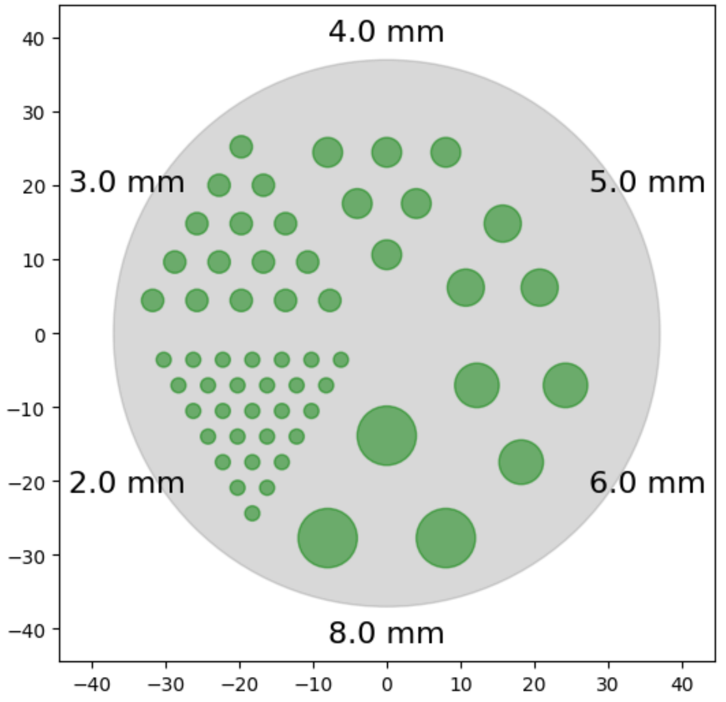
Code for generating a derenzo phantom

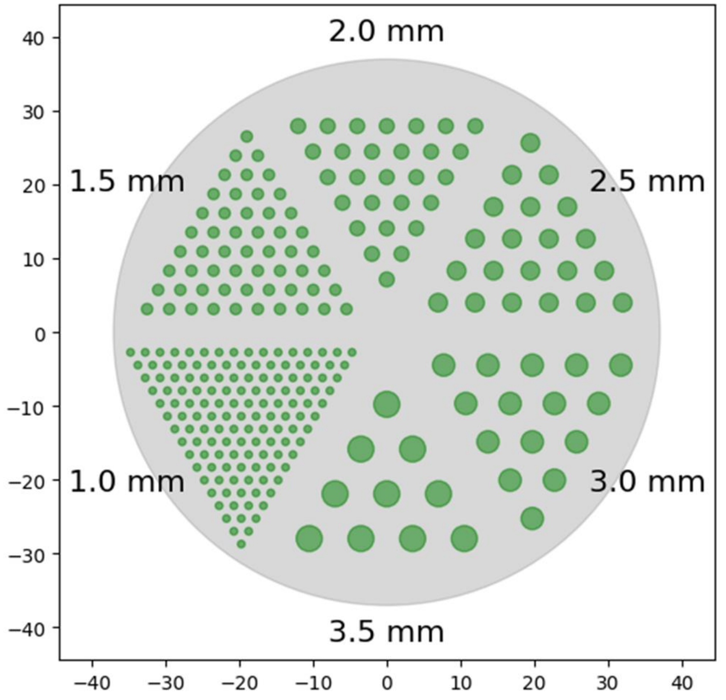
Code of conduct

Security policy

Activity

2 stars





Posted Open Source on Github:

Python code to generate derenzo phantom volumes suitable as input to PyTorch forward projector and simulation/reconstruction codes

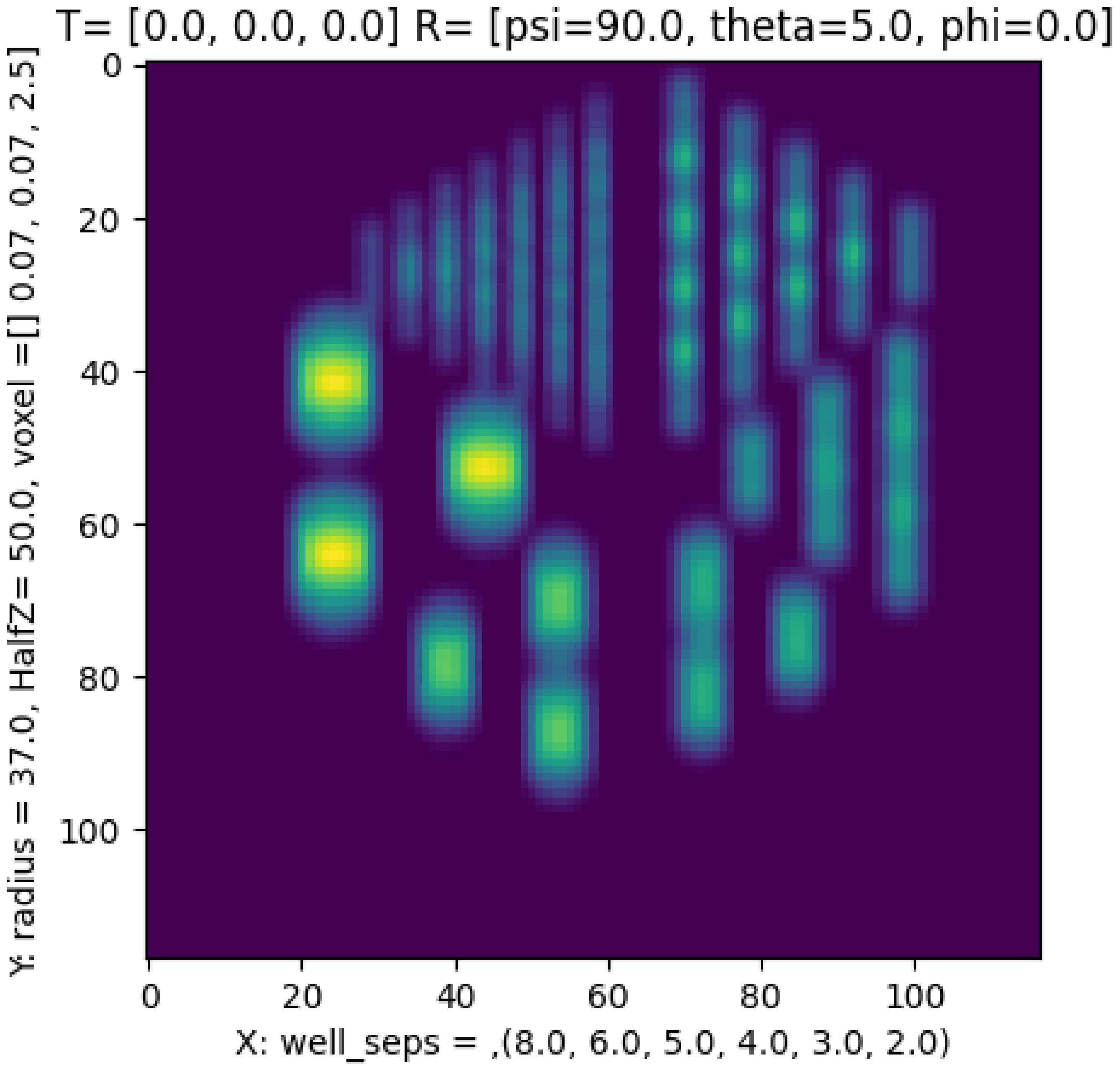
Inputs:

- Well sizes and phantom radius
- Phantom isocenter coordinates
- Phantom orientation Euler angles
- CT-like output volume dimensions
- CT-like output voxels sizes

Output:

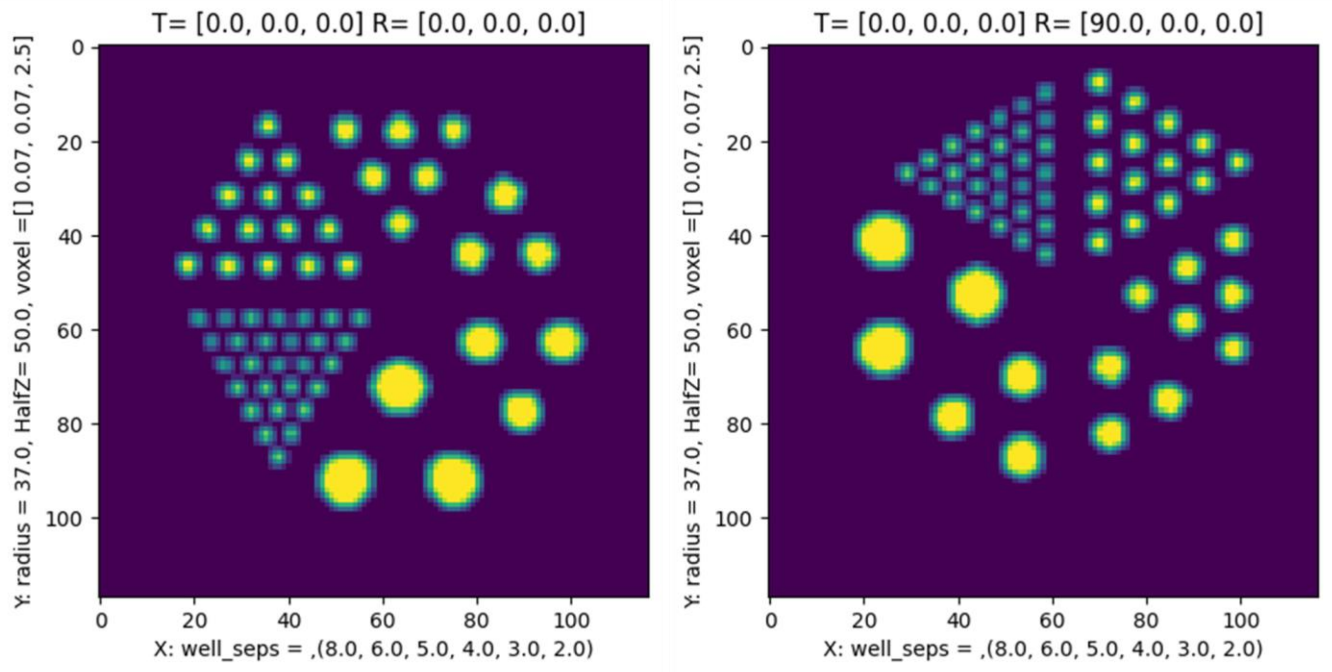
- Phantom object with methods
- Python/Pytorch Image Volume

BillWorstell / derenzo_phantom



iCheckRotations.ipynb

First rotation in lab frame is about phantom axis



Modified to generate 2D 1024x1024 mask of phantom

iphantom1.ipynb is a playground, with subroutines spelled out explicitly for modification as desired

iphantom2.ipynb uses a version of derenzo_phantom loaded from Google drive

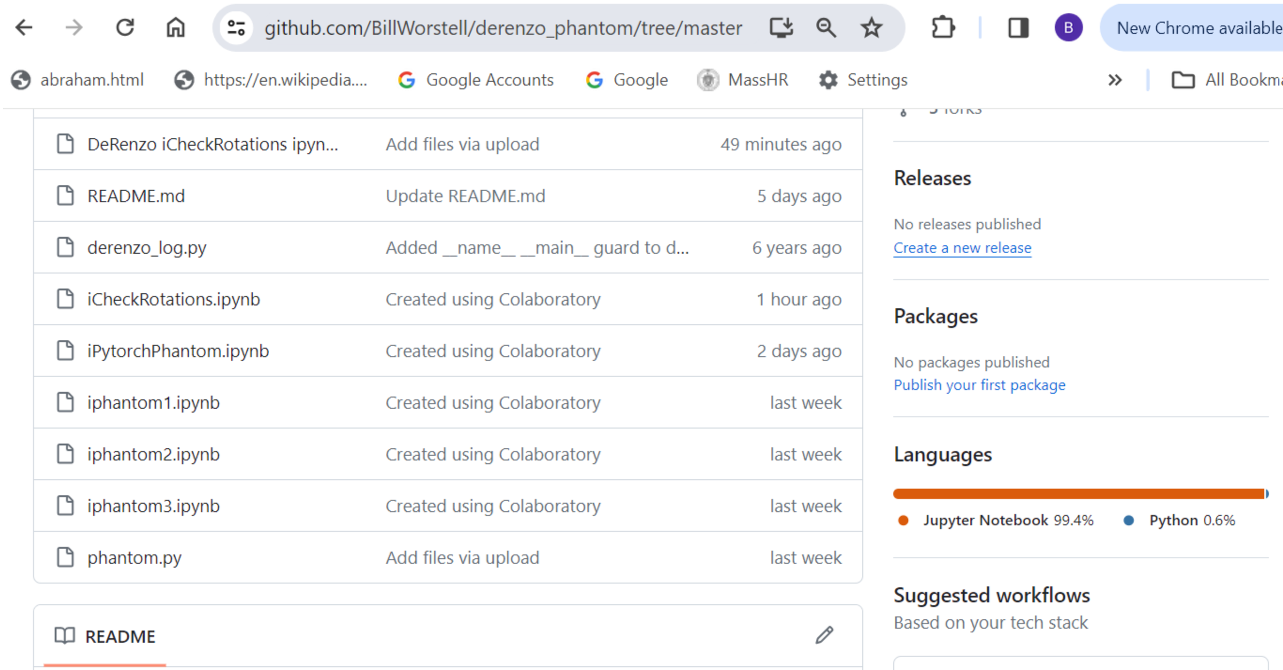
iphantom3.ipynb clones derenzo_phantom from github and uses that

Posted Open Source on Github:

JuPyter notebooks and modified Python functions tested using Google Colab, with output displays and report pdfs

Validation exercises:

- Generation of Phantom objects with arbitrary parameters using JuPyter notebooks with debugging displays
- iCheckRotations.ipynb -> 3D rotation control tested
- Initially written “script-style” with function and documentation to be further developed and posted



DeRenzo iCheckRotations ipyn...	Add files via upload	49 minutes ago
README.md	Update README.md	5 days ago
derenzo_log.py	Added __name__ __main__ guard to d...	6 years ago
iCheckRotations.ipynb	Created using Colaboratory	1 hour ago
iPytorchPhantom.ipynb	Created using Colaboratory	2 days ago
iphantom1.ipynb	Created using Colaboratory	last week
iphantom2.ipynb	Created using Colaboratory	last week
iphantom3.ipynb	Created using Colaboratory	last week
phantom.py	Add files via upload	last week

README

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

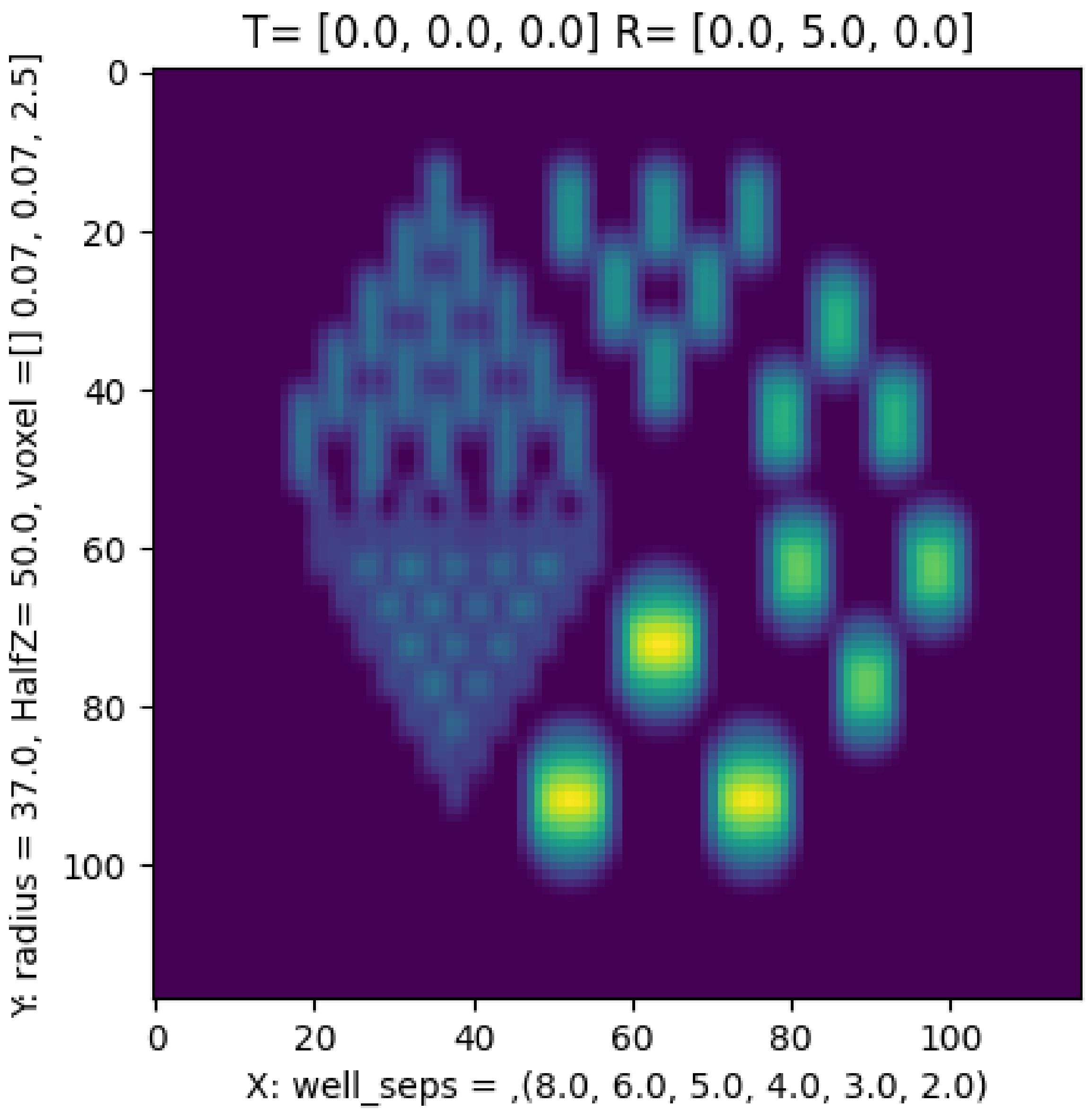
● Jupyter Notebook 99.4% ● Python 0.6%

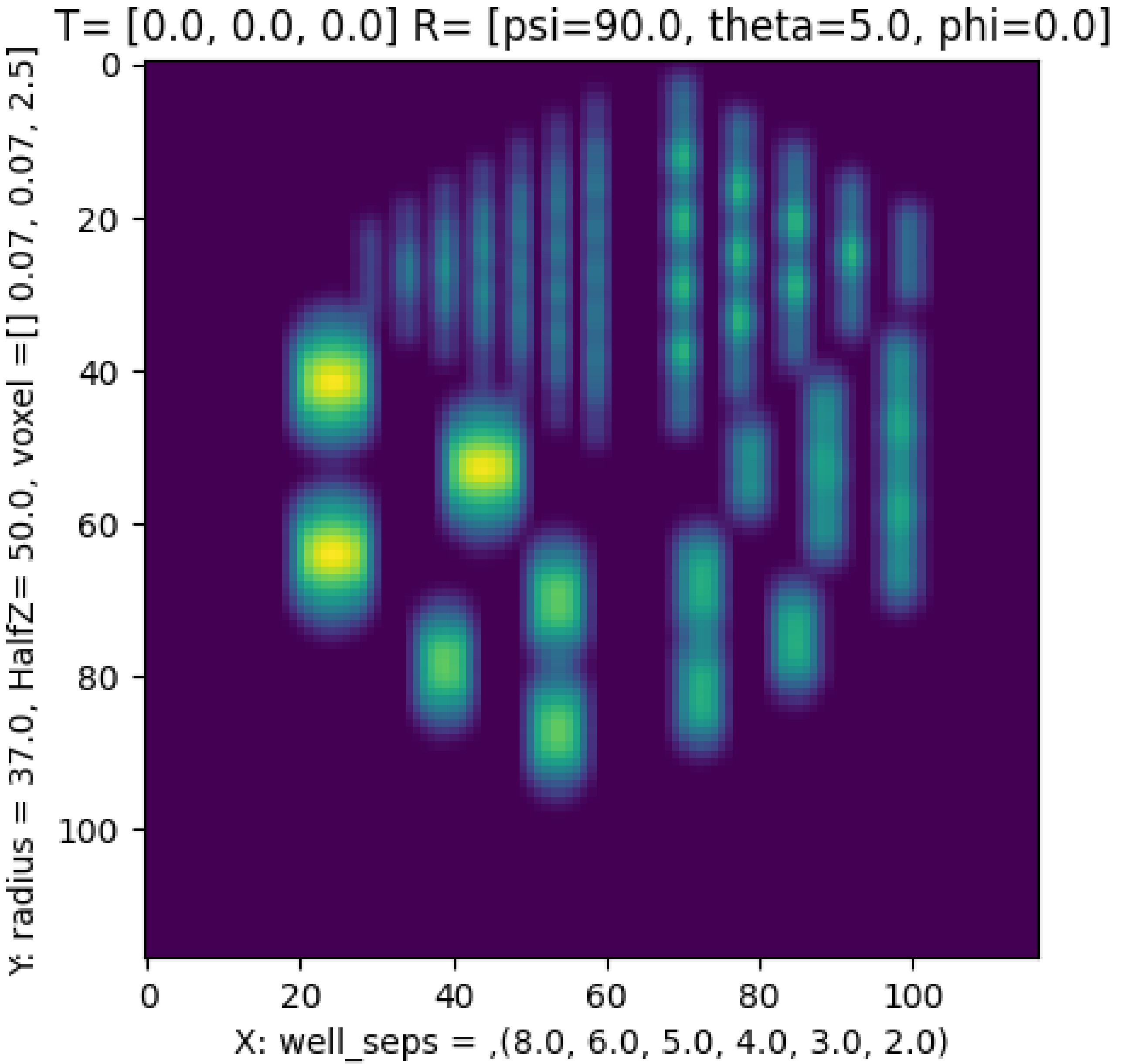
Suggested workflows
Based on your tech stack

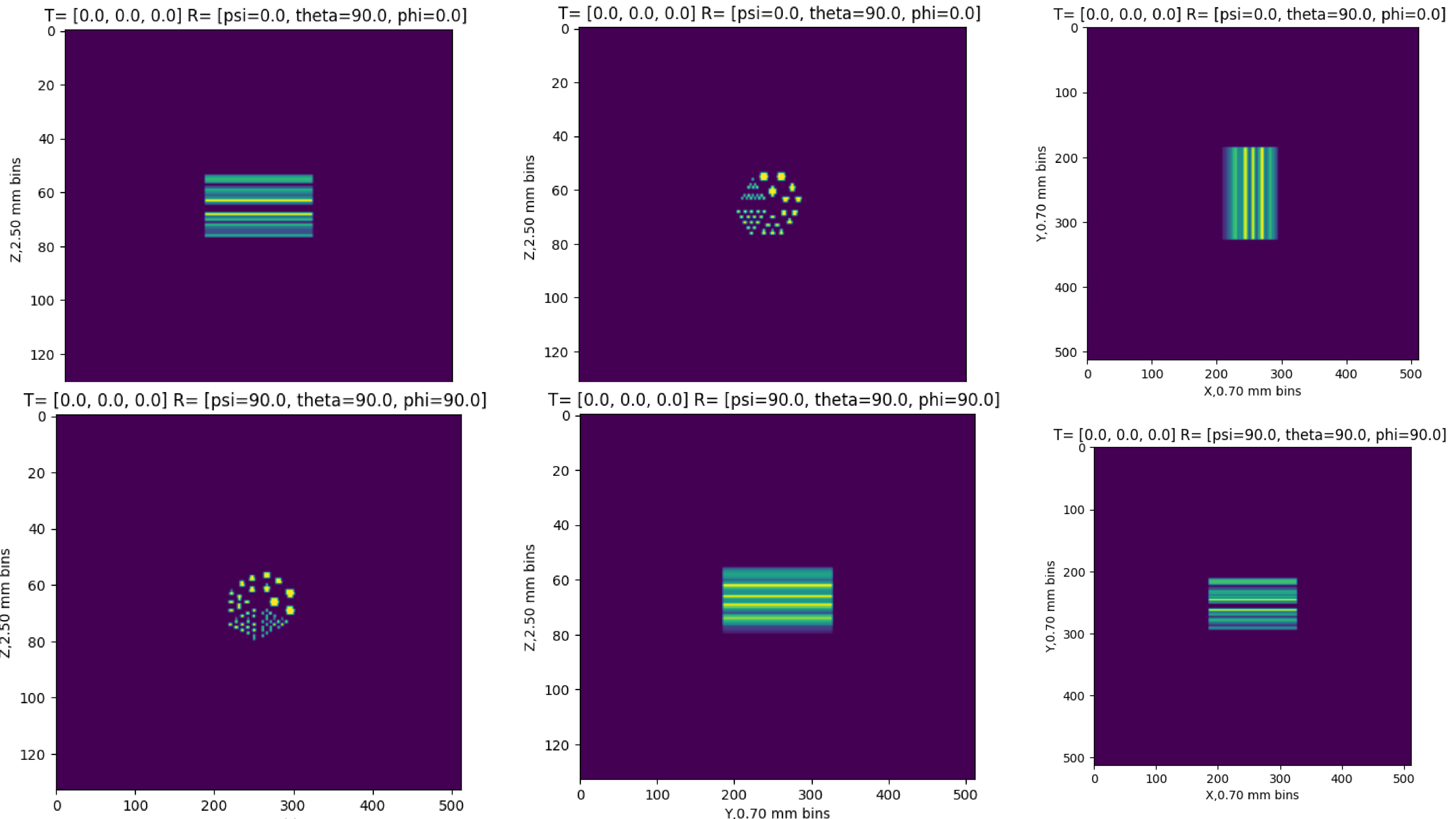
iCheckRotations.ipynb

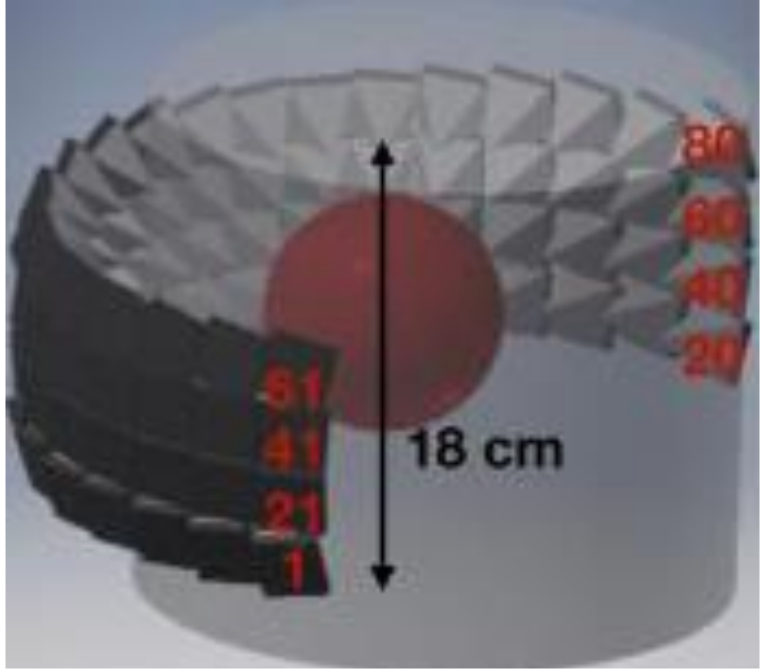
First rotation in lab frame is about phantom axis

Second rotation in lab frame sets phantom axis altitude. Third rotation sets azimuth.



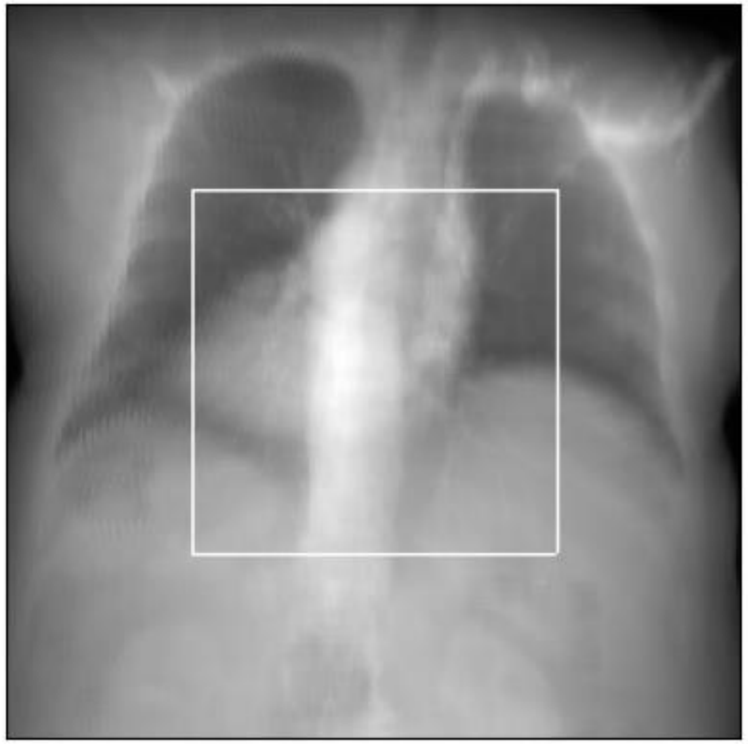


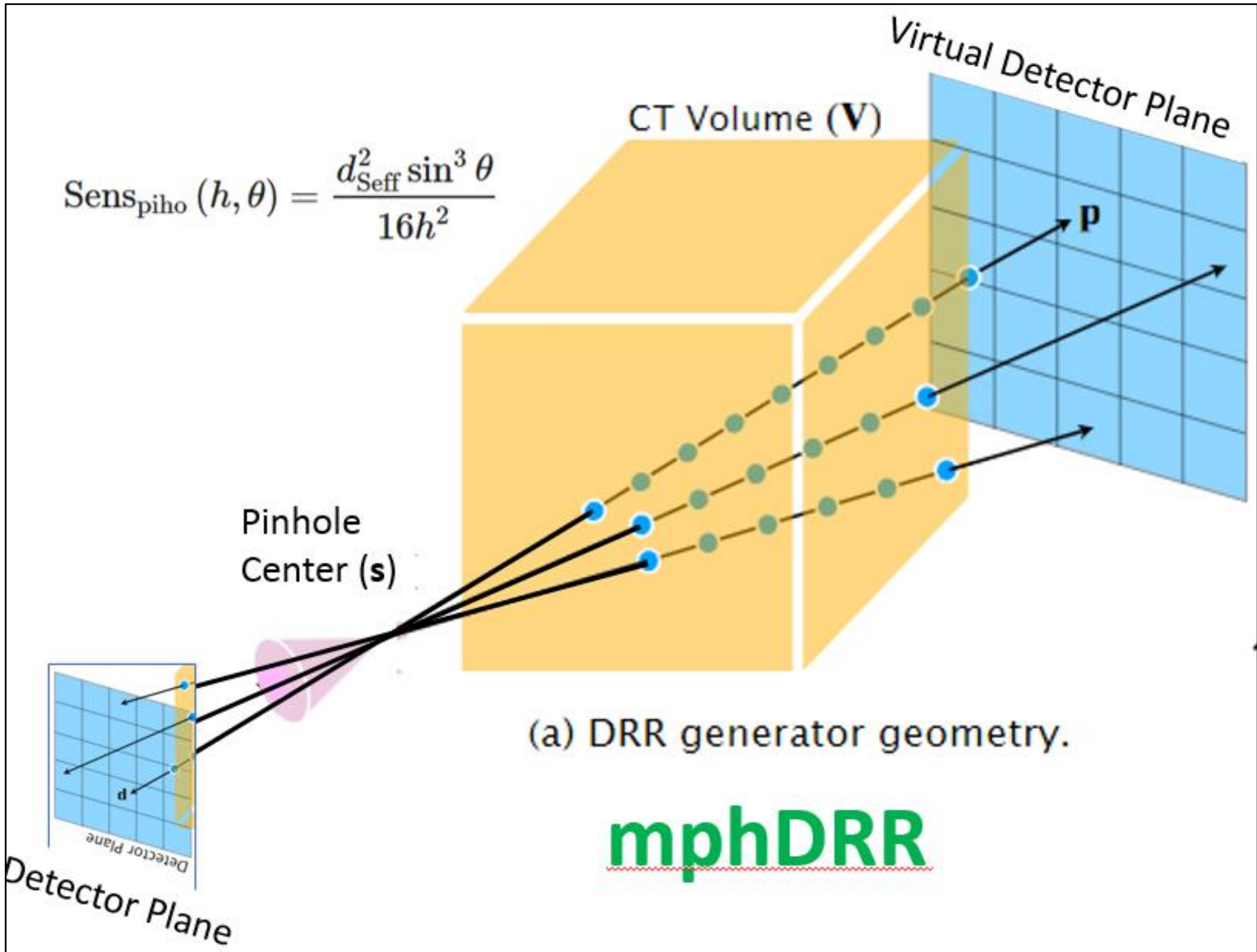

T= [0.0, 0.0, 0.0] R= [psi=0.0, theta=90.0, phi=0.0]
Z,2.50 mm bins
T= [0.0, 0.0, 0.0] R= [psi=0.0, theta=90.0, phi=0.0]
Z,2.50 mm bins
T= [0.0, 0.0, 0.0] R= [psi=0.0, theta=90.0, phi=0.0]
Y,0.70 mm bins
X,0.70 mm bins
T= [0.0, 0.0, 0.0] R= [psi=90.0, theta=90.0, phi=90.0]
Z,2.50 mm bins
T= [0.0, 0.0, 0.0] R= [psi=90.0, theta=90.0, phi=90.0]
Z,2.50 mm bins
Y,0.70 mm bins
T= [0.0, 0.0, 0.0] R= [psi=90.0, theta=90.0, phi=90.0]
Y,0.70 mm bins
X,0.70 mm bins



Posted Open Source on Github (diffDRR):

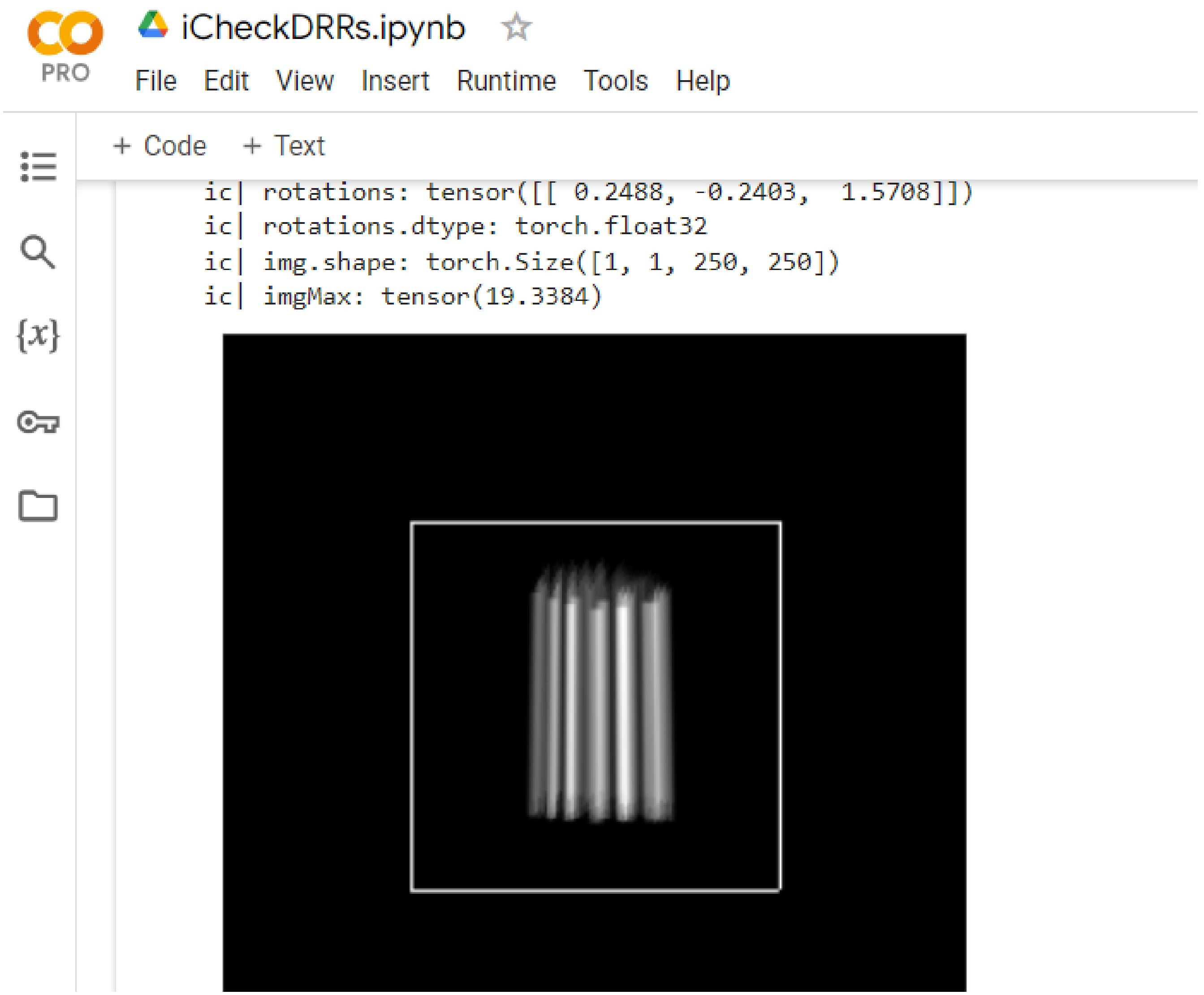
Python code to generate derenzo phantom volumes which are compatible with mphDRR codes



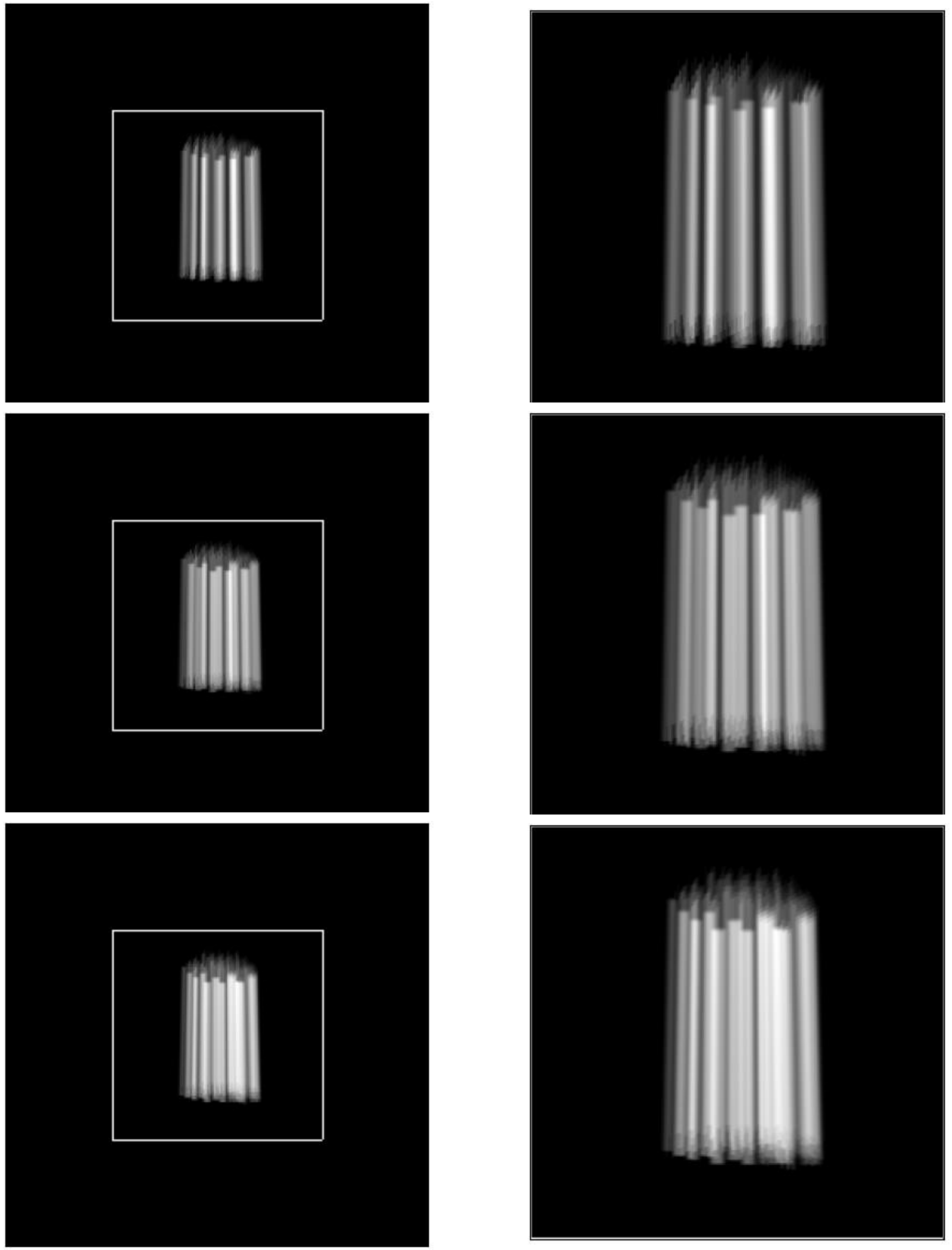


(a) DRR generator geometry.

mphDRR

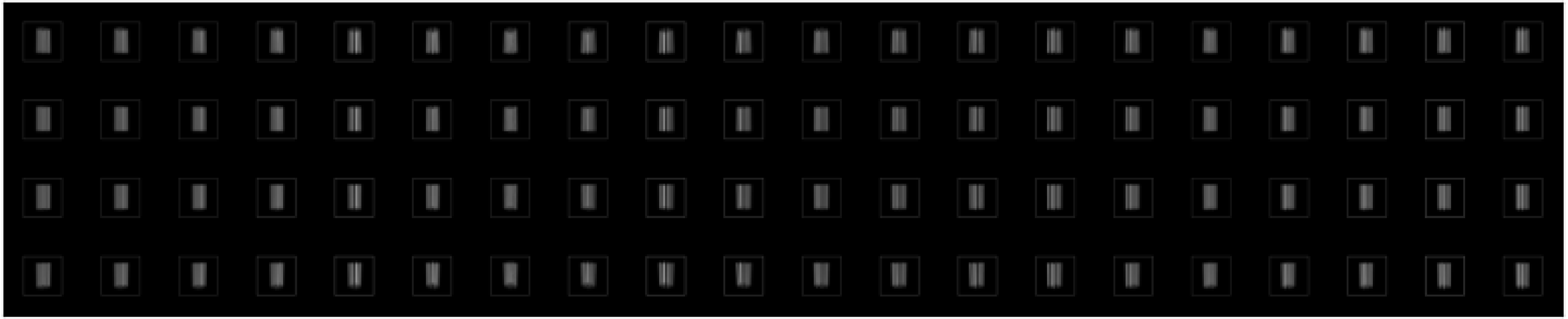


```
ic| rotations: tensor([[ 0.2488, -0.2403,  1.5708]])
ic| rotations.dtype: torch.float32
ic| img.shape: torch.Size([1, 1, 250, 250])
ic| imgMax: tensor(19.3384)
```



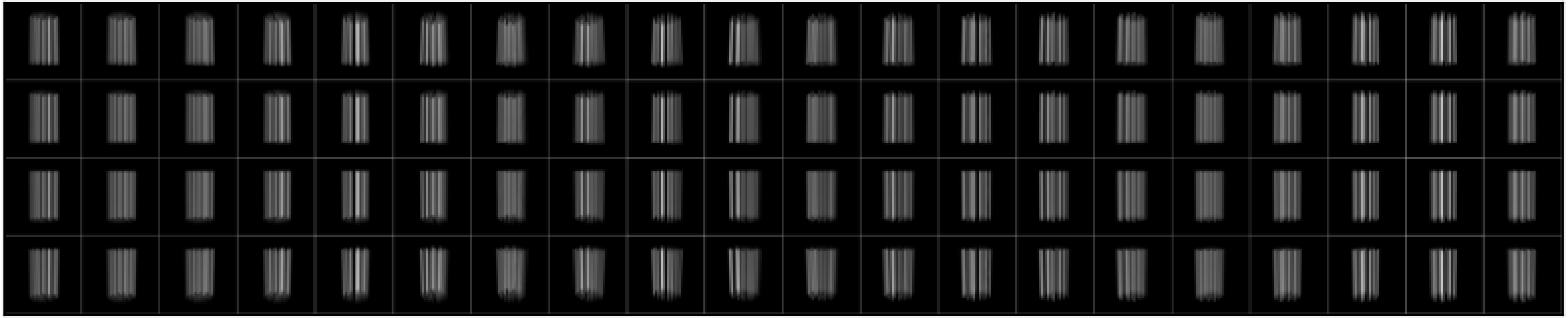
Posted Open Source on Github (diffDRR):

Python code to generate derenzo phantom volumes forward projected by mphDRR codes

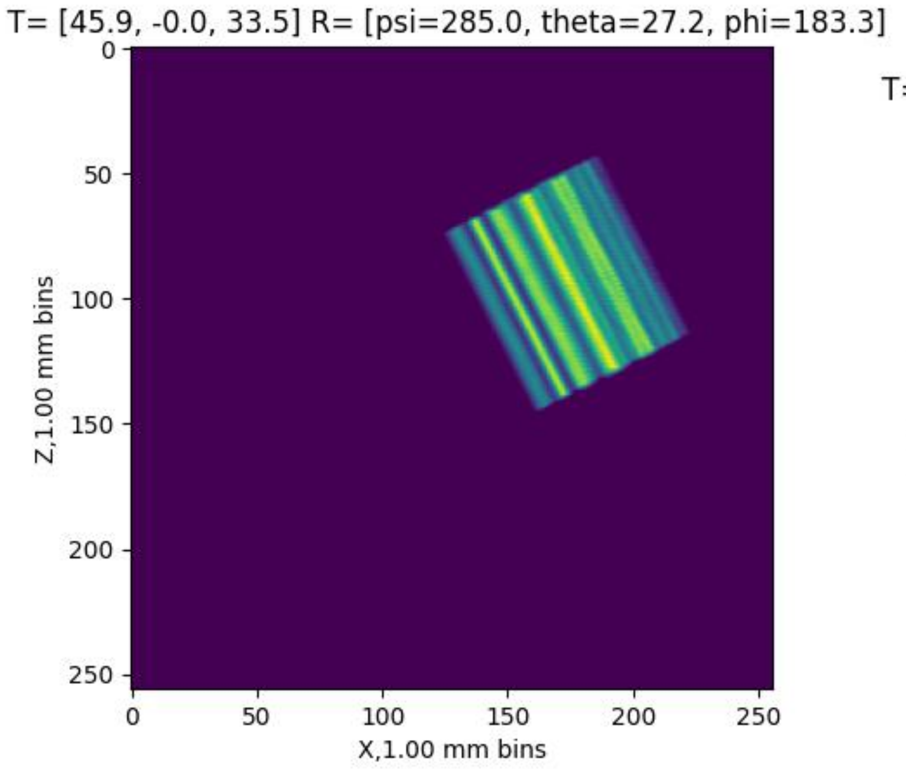


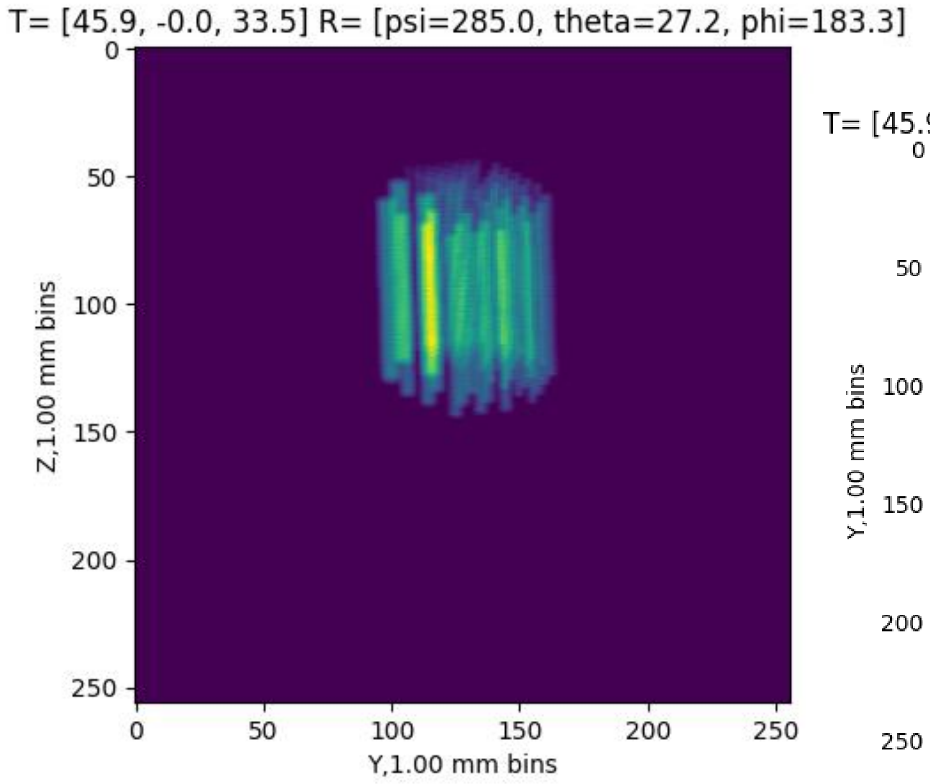
Posted Open Source on Github (diffDRR):

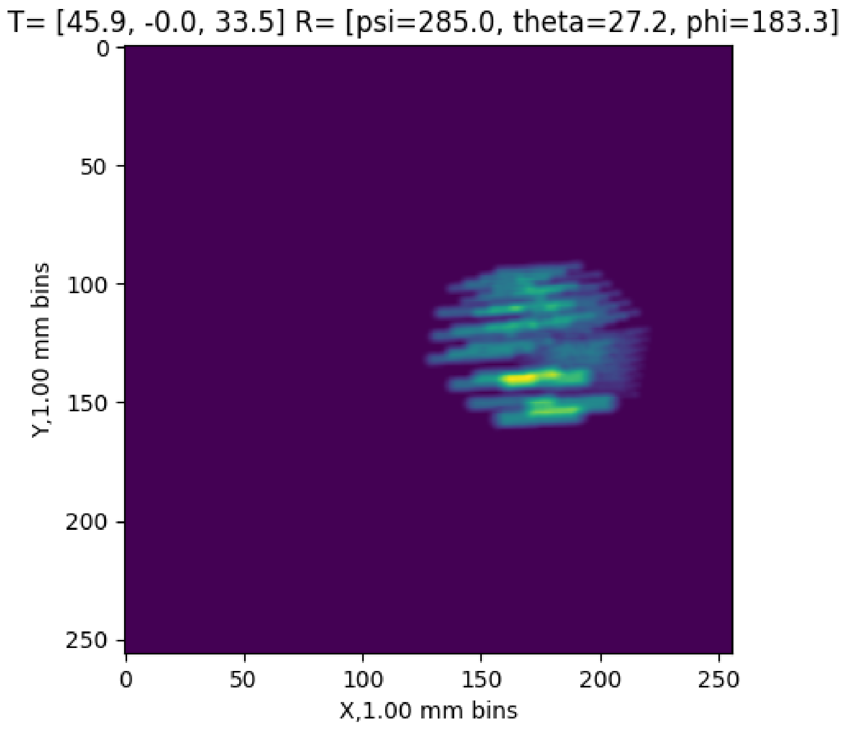
Python code to generate derenzo phantom volumes
forward projected by mphDRR codes



Orthogonal summed projection (Euclidean volume) for first random event, with translations and rotations indicated

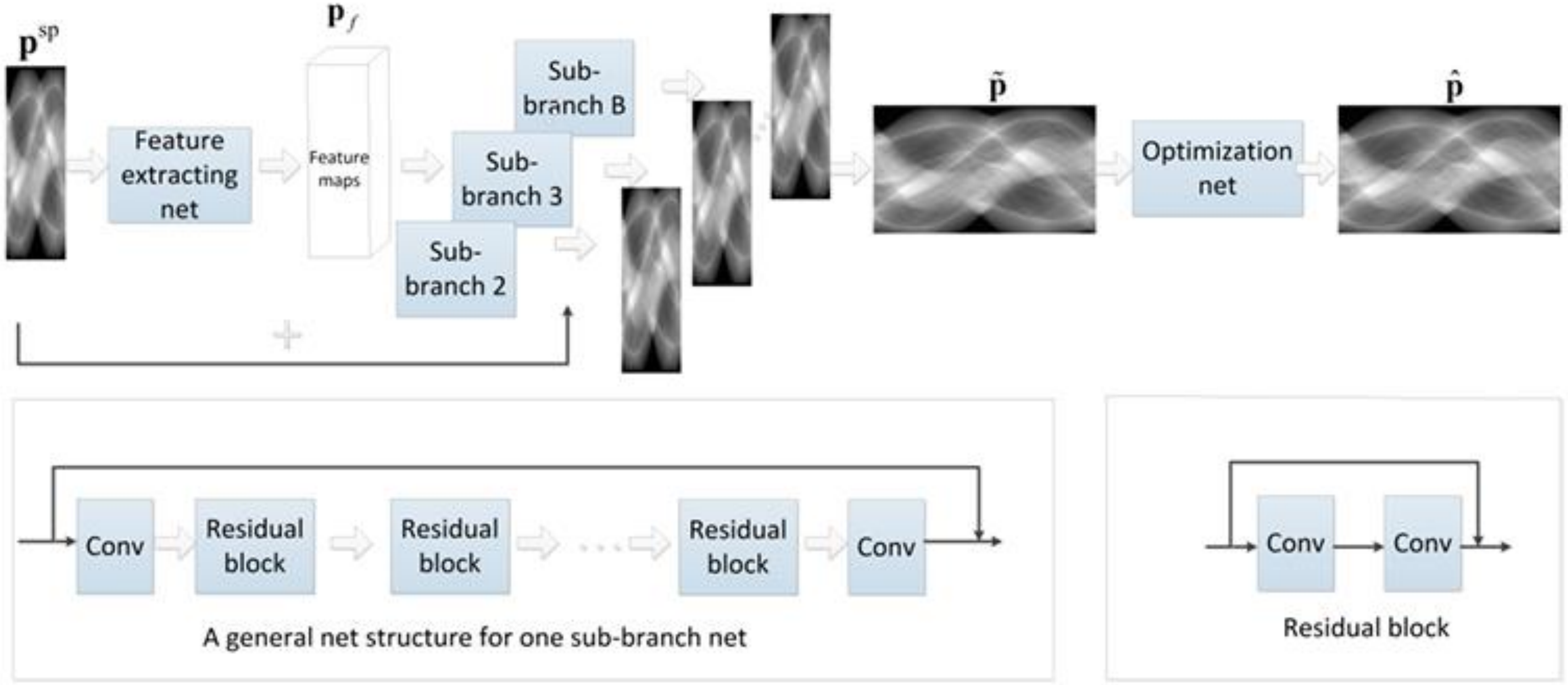


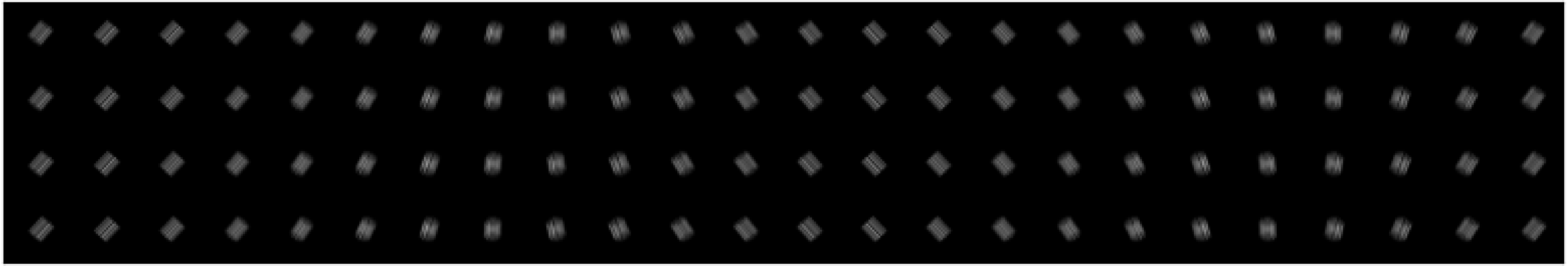




Liang, K., Yang, H. and Xing, Y., 2018. Comparison of projection domain, image domain, and comprehensive deep learning for sparse-view X-ray CT image reconstruction. *arXiv preprint arXiv:1804.04289*.

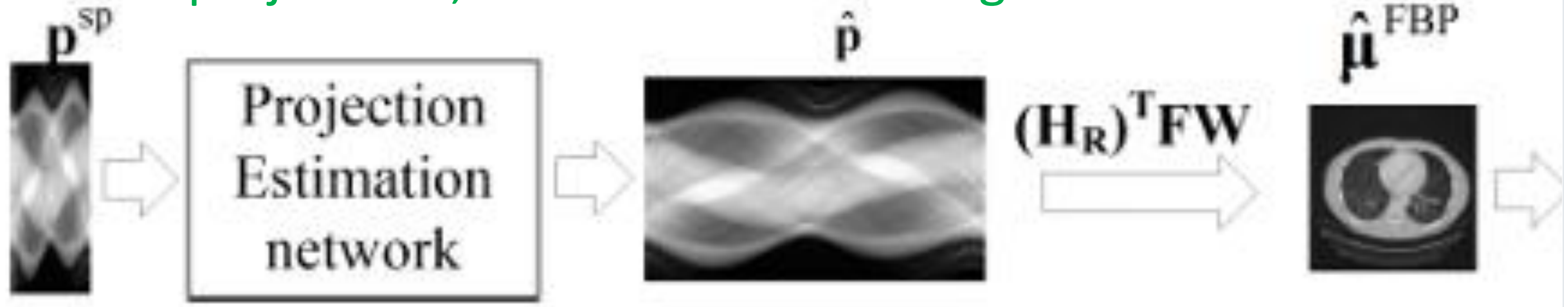
Method for learned reconstruction of high-resolution sparse-view X-ray CT which we can adapt to our multiple pinhole SPECT case

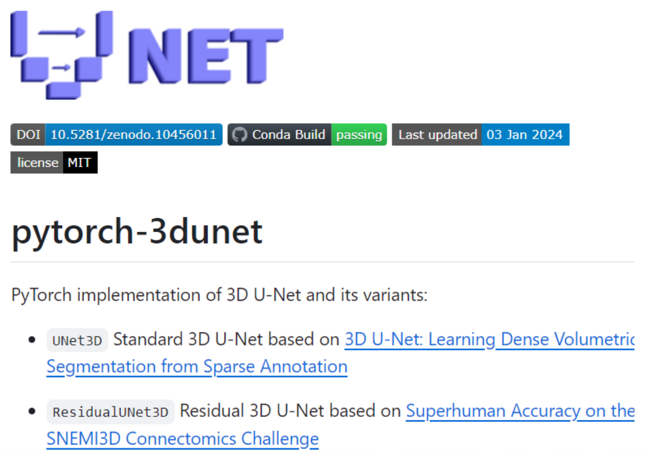




Next will try using 3D U-net to infer ideal CT projections from mphSPECT projections, then reconstruct using FBP for all slices

https://github.com/wolny/pytorch-3dunet



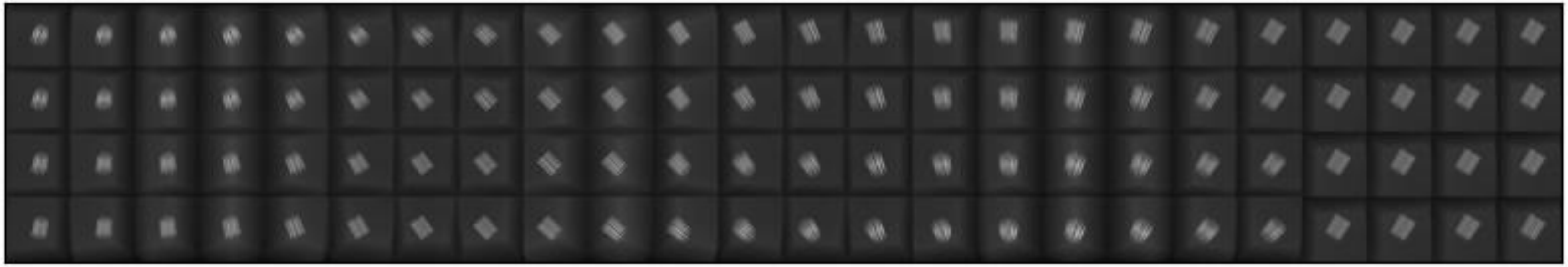

NET

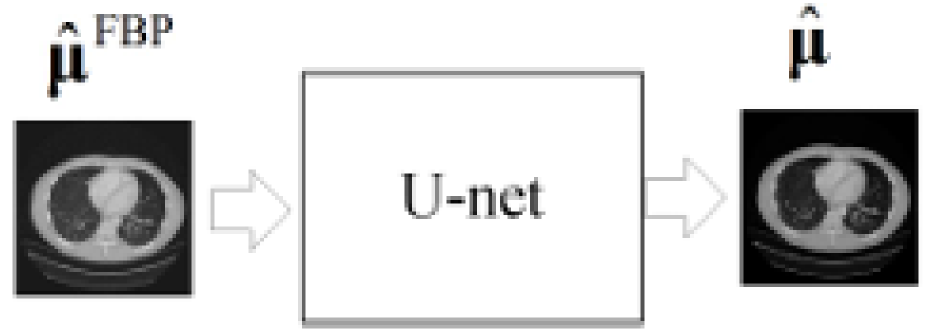
DOI 10.5281/zenodo.10456011 | Conda Build passing | Last updated 03 Jan 2024 | license MIT

pytorch-3dunet

PyTorch implementation of 3D U-Net and its variants:

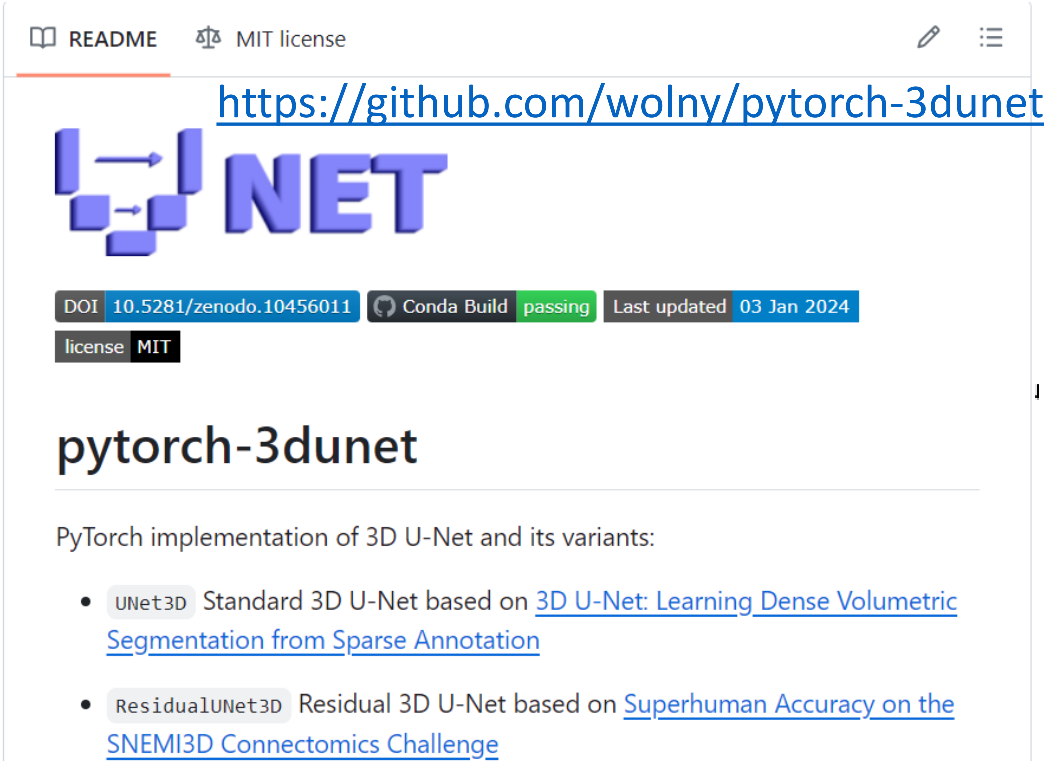
- `UNet3D` Standard 3D U-Net based on 3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation
- `ResidualUNet3D` Residual 3D U-Net based on Superhuman Accuracy on the SNEMI3D Connectomics Challenge





256-angle radon projection tensors are also saved for the same events, for possible future end-to-end learned reconstruction as in the source paper

https://github.com/wolny/pytorch-3dunet

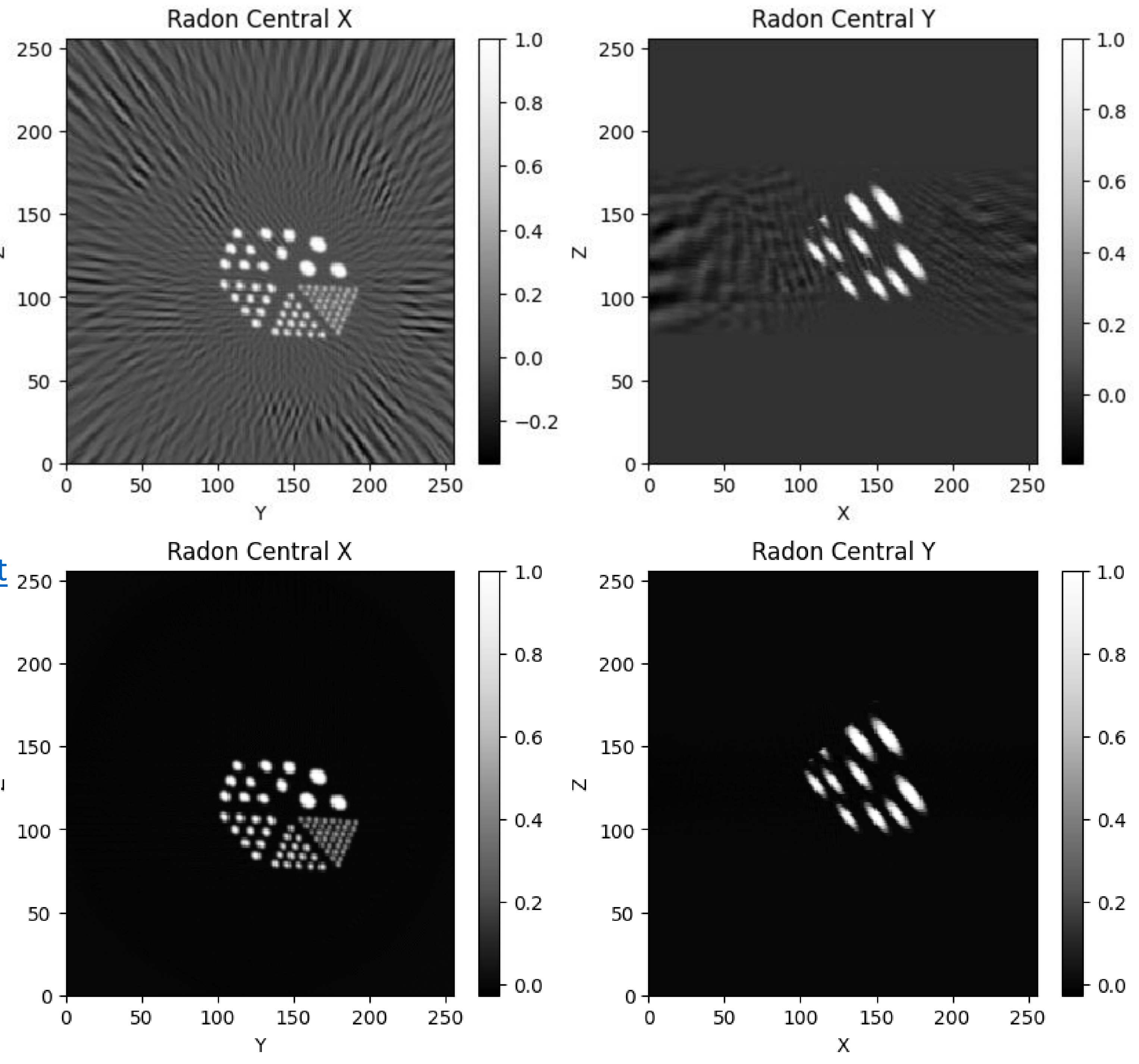

README MIT license

DOI 10.5281/zenodo.10456011 | Conda Build passing | Last updated 03 Jan 2024 | license MIT

pytorch-3dunet

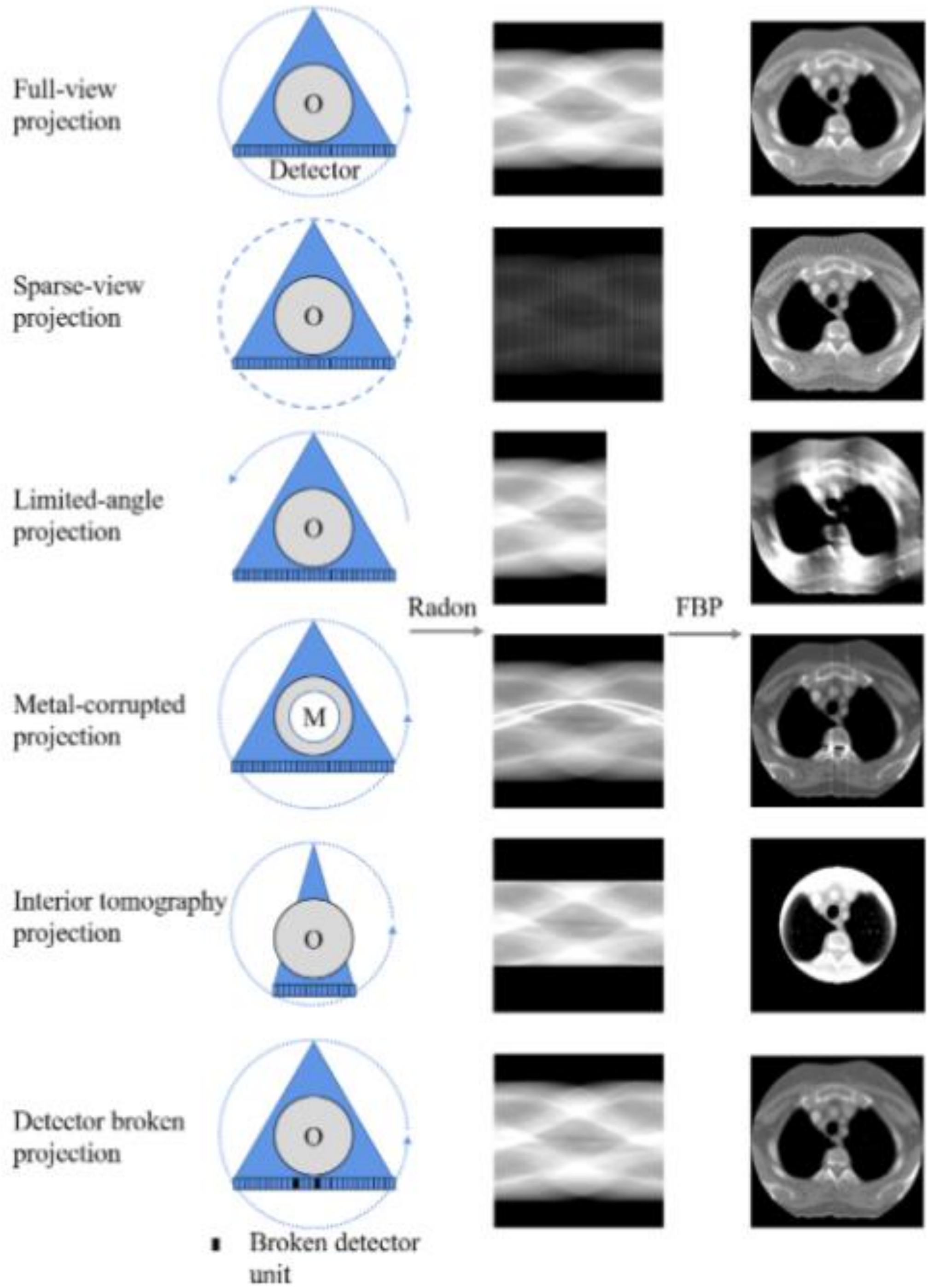
PyTorch implementation of 3D U-Net and its variants:

- `UNet3D` Standard 3D U-Net based on 3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation
- `ResidualUNet3D` Residual 3D U-Net based on Superhuman Accuracy on the SNEMI3D Connectomics Challenge

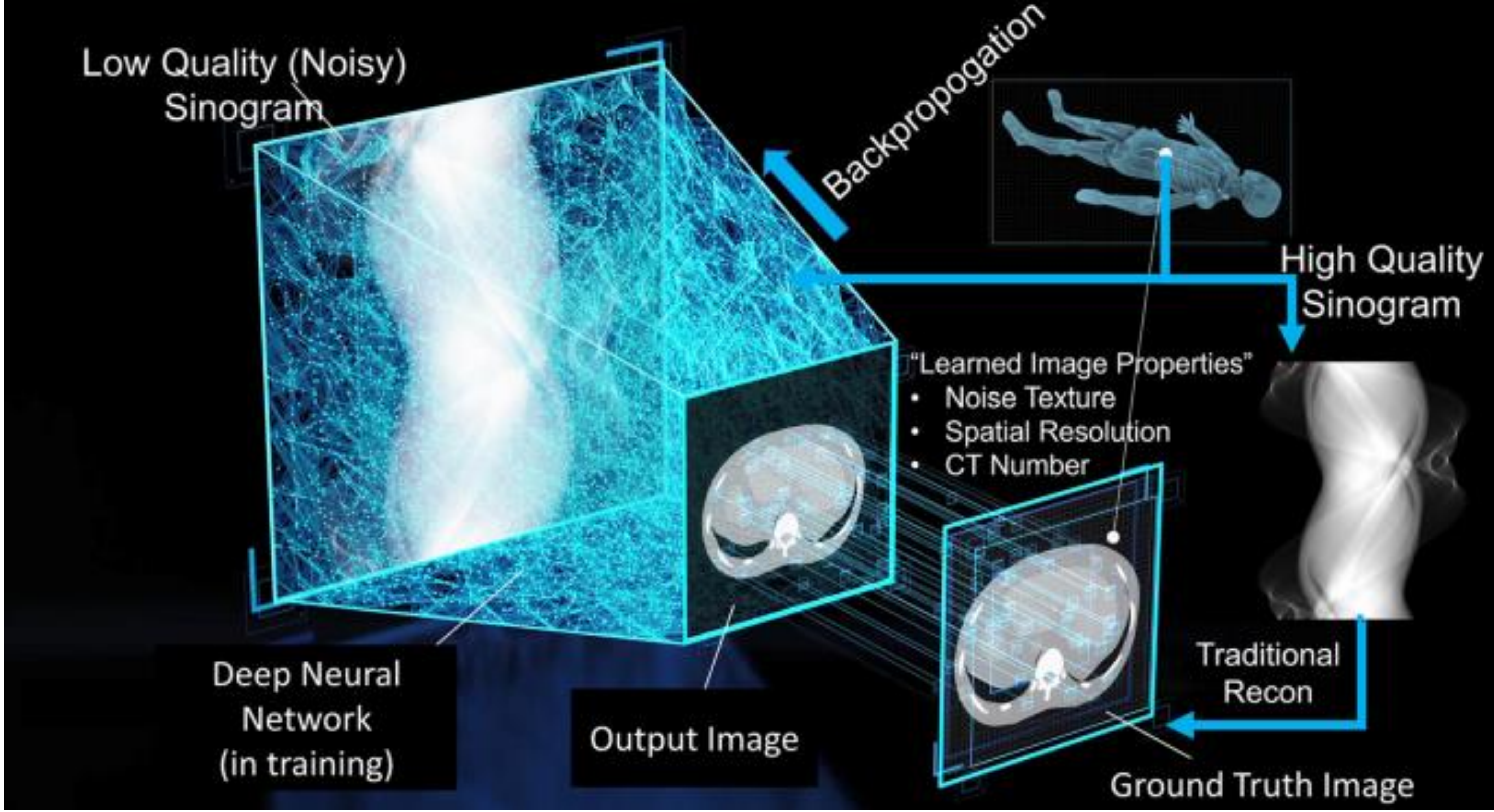


Wang, T., Xia, W., Lu, J. and Zhang, Y., 2023. A review of deep learning ct reconstruction from incomplete projection data. *IEEE Transactions on Radiation and Plasma Medical Sciences*.

Sidky, E.Y. and Pan, X., 2022. Report on the AAPM deep-learning sparse-view CT grand challenge. *Medical Physics*, *49*(8), pp.4935-4943.

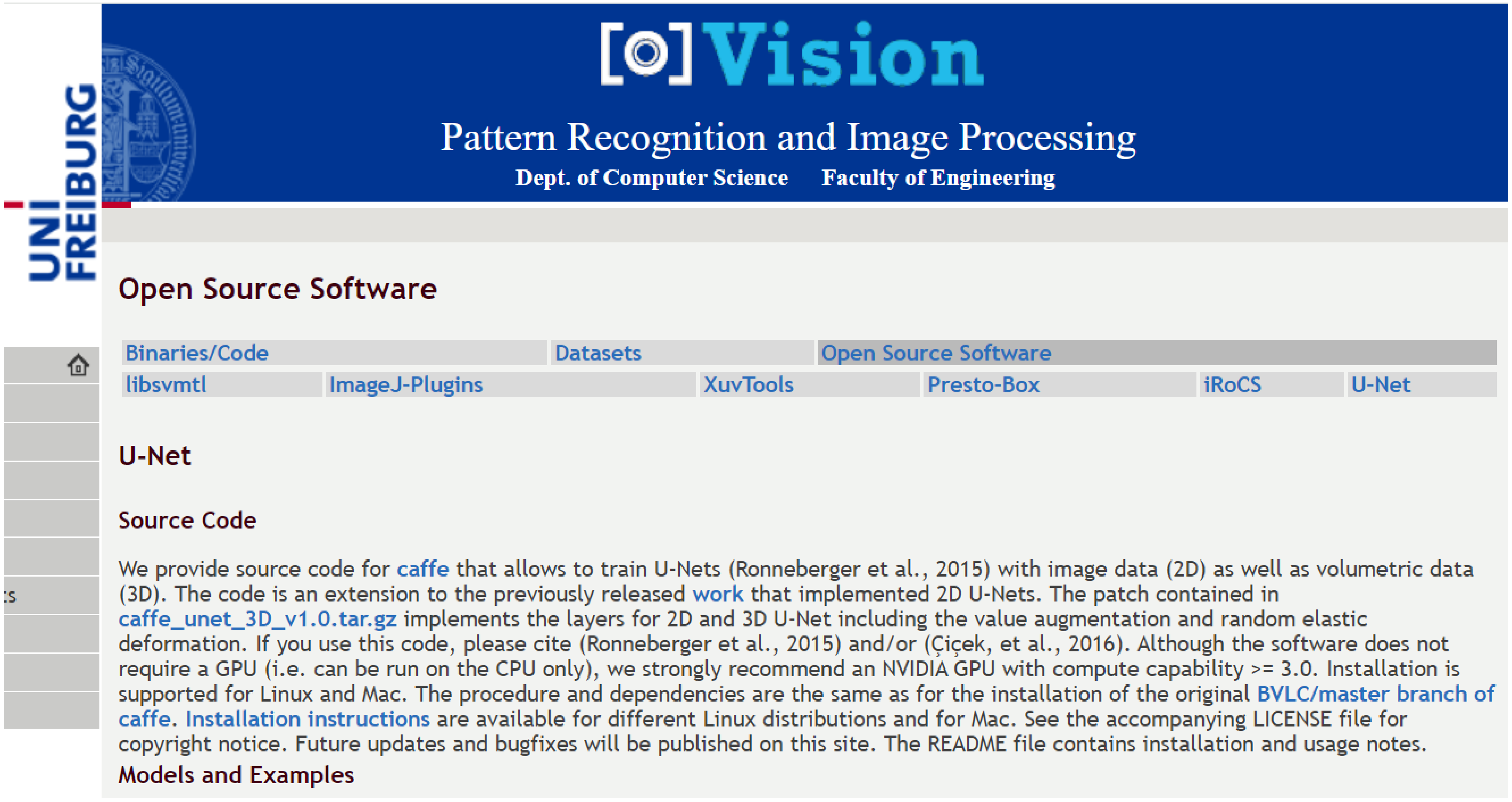


Szczykutowicz, T.P., Toia, G.V., Dhanantwari, A. and Nett, B., 2022. A review of deep learning CT reconstruction: concepts, limitations, and promise in clinical practice. *Current Radiology Reports*, *10*(9), pp.101-115.



Çiçek, Ö., Abdulkadir, A., Lienkamp, S.S., Brox, T. and Ronneberger, O., 2016. [3D U-Net: learning dense volumetric segmentation from sparse annotation]. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2016: 19th International Conference, Athens, Greece, October 17-21, 2016, Proceedings, Part II 19* (pp. 424-432). Springer International Publishing. Cited by 6731

https://lmb.informatik.uni-freiburg.de/resources/opensource/unet.en.html



Open Source Software

U-Net

Source Code

We provide source code for caffe that allows to train U-Nets (Ronneberger et al., 2015) with image data (2D) as well as volumetric data (3D). The code is an extension to the previously released work that implemented 2D U-Nets. The patch contained in caffe_unet_3D_v1.0.tar.gz implements the layers for 2D and 3D U-Net including the value augmentation and random elastic deformation. If you use this code, please cite (Ronneberger et al., 2015) and/or (Çiçek, et al., 2016). Although the software does not require a GPU (i.e. can be run on the CPU only), we strongly recommend an NVIDIA GPU with compute capability >= 3.0. Installation is supported for Linux and Mac. The procedure and dependencies are the same as for the installation of the original BVLC/master branch of caffe. Installation instructions are available for different Linux distributions and for Mac. See the accompanying LICENSE file for copyright notice. Future updates and bugfixes will be published on this site. The README file contains installation and usage notes.

Models and Examples

Çiçek, Ö., Abdulkadir, A., Lienkamp, S.S., Brox, T. and Ronneberger, O., 2016. 3D U-Net: learning dense volumetric segmentation from sparse annotation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2016: 19th International Conference, Athens, Greece, October 17-21, 2016, Proceedings, Part II 19* (pp. 424-432). Springer International Publishing

The input to the network is a 132 × 132 × 116 voxel tile of the image with 3 channels. Our output in the final layer is 44×44×28 voxels in x, y, and z directions respectively.

Figure 2 illustrates the network architecture. Like the standard u-net, it has an analysis and a synthesis path each with four resolution steps. In the analysis path, each layer contains two 3 × 3 × 3 convolutions each followed by a rectified linear unit (ReLu), and then a 2 × 2 × 2 max pooling with strides of two in each dimension. In the synthesis path, each layer consists of an up convolution of 2 × 2 × 2 by strides of two in each dimension, followed by two 3 × 3 × 3 convolutions each followed by a ReLu. Shortcut connections from layers of equal resolution in the analysis path provide the essential high-resolution features to

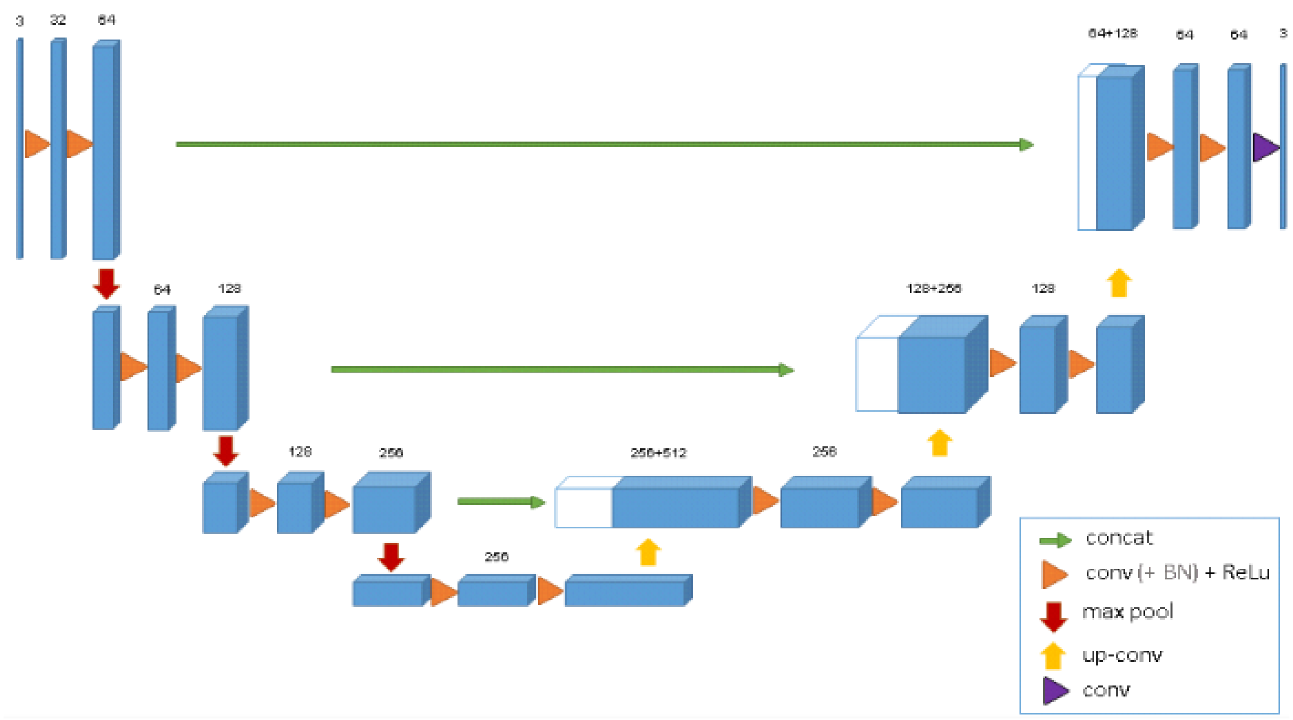


Fig. 2: The 3D u-net architecture. Blue boxes represent feature maps. The number of channels is denoted above each feature map.

the synthesis path. In the last layer a 1×1×1 convolution reduces the number of output channels to the number of labels which is 3 in our case. The architecture has 19069955 parameters in total. Like suggested in [13] we avoid bottlenecks by doubling the number of channels already before max pooling. We also adopt this scheme in the synthesis path.