CWI

Universiteit Antwerpen

# The ASTRA tomography toolbox

## 2D tomography

Tristan van Leeuwen

Slides developed by Willem Jan Palenstijn, Tim Elberfeld

# Table of Contents

# Introduction to ASTRA

**What is ASTRA?**

ASTRA provides fast and flexible building blocks for 2D/3D tomographic reconstruction, aimed at algorithm developers and researchers.

| | |
|---|---|
| **Flexible**<br>algorithms and geometries | **Powerful**<br>C++ and CUDA |
| **Easy to use**<br>MATLAB and Python interface | **Free**<br>Open source, cross-platform |

# History

The ASTRA toolbox was started at the Vision Lab of the University of Antwerp in Belgium by PhD students and post-docs

Work started in **October 2007**

- Initial goal: reduced implementation work for internal PhD projects

Later on: interest from external labs and companies

- E.g. ESRF (France) and FEI (The Netherlands, now ThermoFisher)

First open-source release in **August 2012**

Active development continues

- Now jointly by Vision Lab and CWI

2.0 Release **October 2021**

# Features

Geometries:
- 2D parallel and fan beam
- 3D parallel and cone beam
- All with fully flexible source/detector positioning

Basic reconstruction algorithms:
- FBP, FDK reconstruction
- Iterative SIRT, CGLS reconstruction

Primitives for building your own algorithms:
- FP, BP

# Non-Features

Data I/O

Data alignment, preprocessing

Matched GPU FP, BP operators

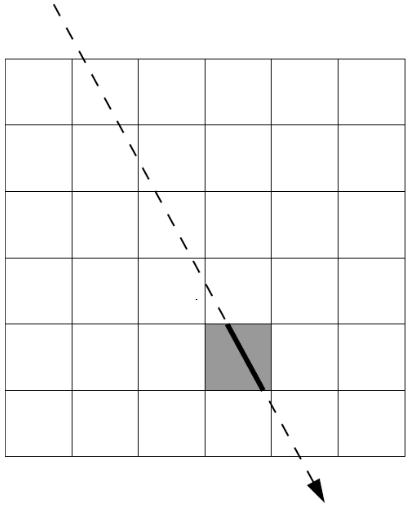
- CPU FP, BP are matched; GPU are not

Advanced iterative reconstruction algorithms

# Tomography as linear equations

Linear (integral) projection model:

- Projection of background (air or vacuum) is zero
- Projection depends linearly on sample thickness and density



$$\sum_{j=1}^{n} w_{ij} x_j = p_i$$

$$Wx = p$$

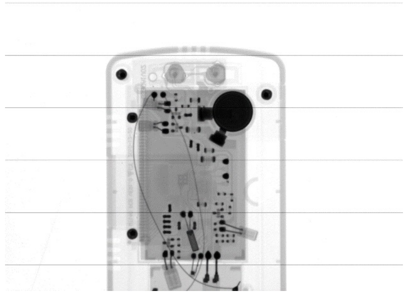
# Tomography as linear equations

ASTRA uses a linear line (or strip) integral model and assumes input projection data has this form.

For HAADF-STEM, raw data satisfies this. For most modalities (TEM, X-ray transmission), input data depends exponentially on material density/thickness (cf. Lambert-Beer's law).

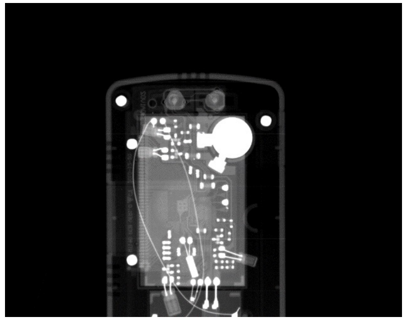
Raw X-ray data:

$$I = I_0 e^{-p}$$

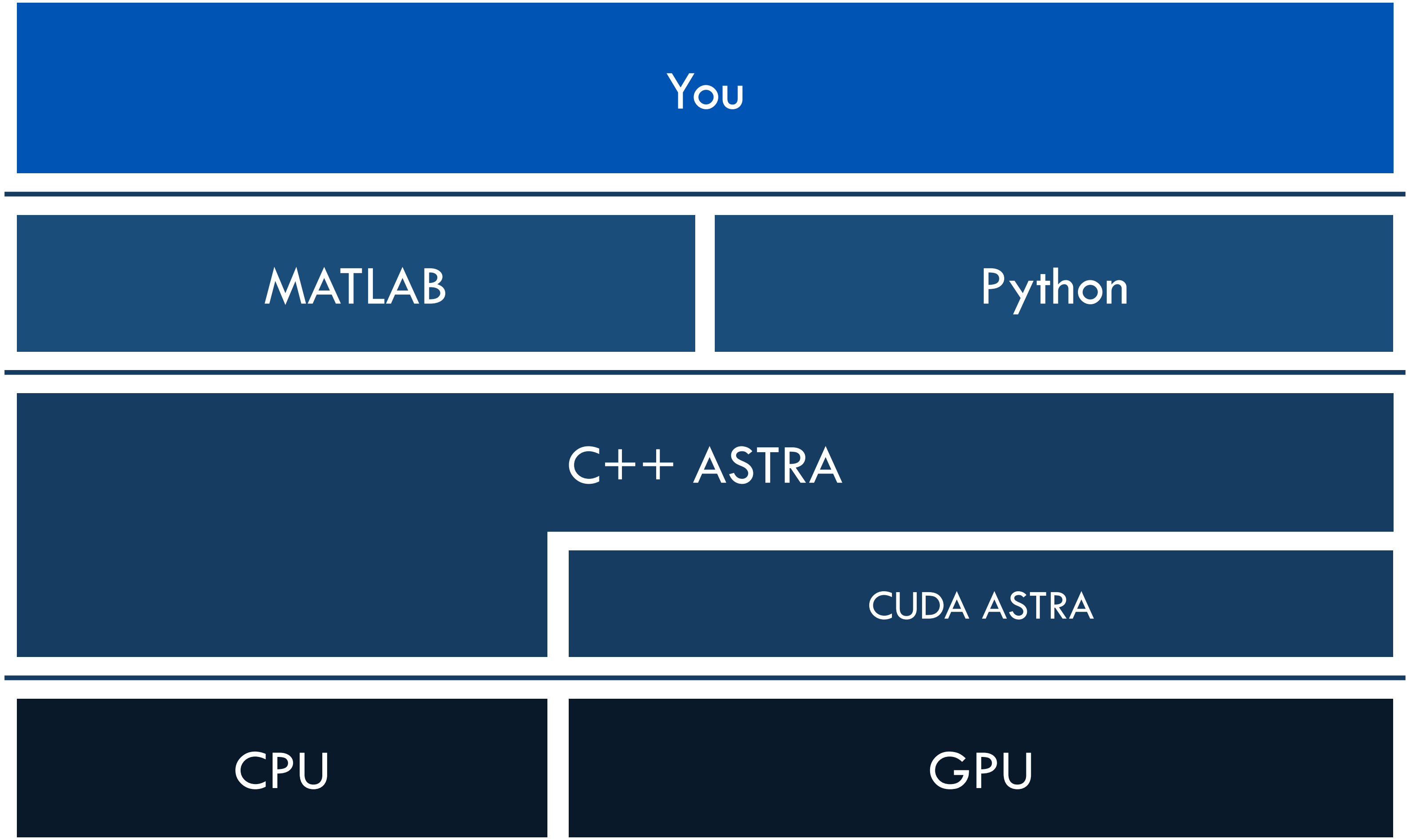


Linearized (or normalized, flat-fielded, …) data:

$$p = -\log\left(\frac{I}{I_0}\right)$$



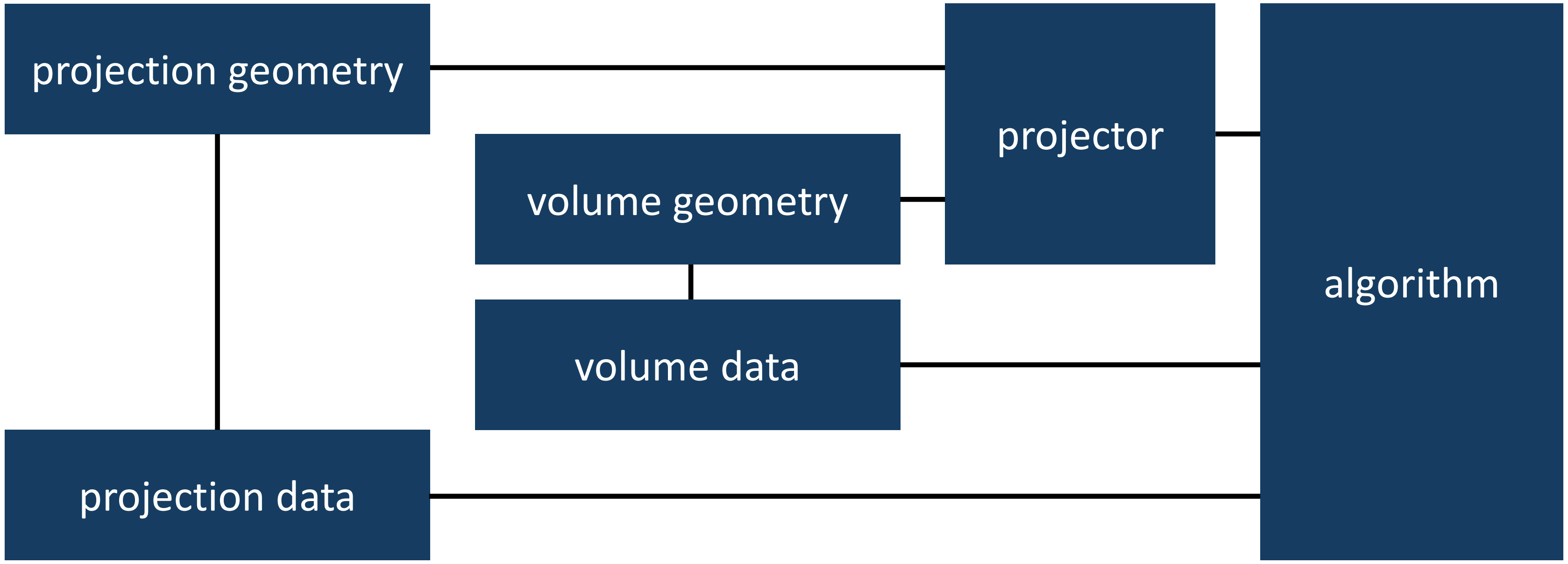
# ASTRA Architecture



# Geometries in ASTRA

Modeling of the x-ray scanning setup

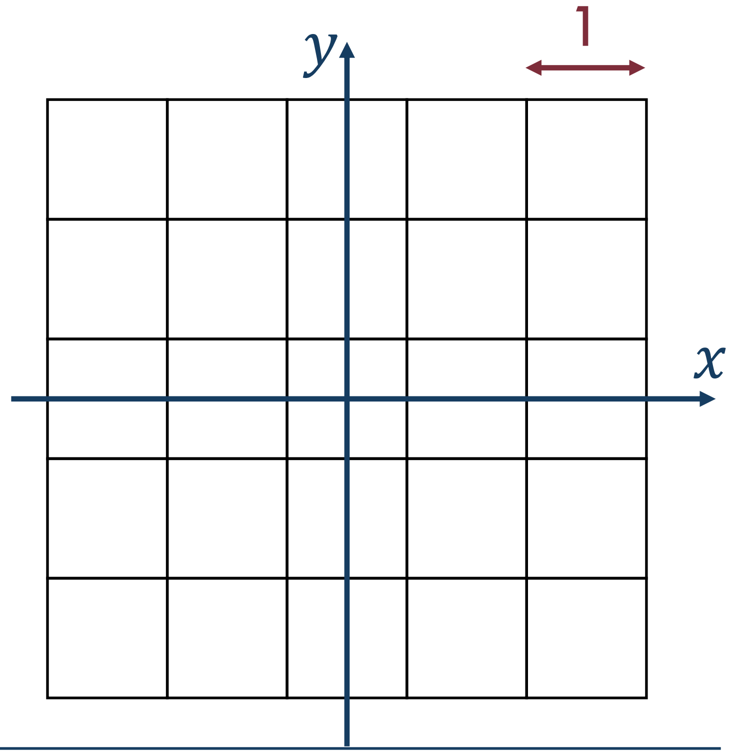
- Phantom / volume
- Source
- Detector



# Geometries in ASTRA

Volume geometry and volume data

```
vol_geom = astra.create_vol_geom(num_rows, num_cols)
```




- all lengths and sizes defined relative to voxel size
- voxel size defaults to 1 unit (square voxels)

# Geometries in ASTRA

Volume geometry and volume data

```
rec_id = astra.data2d.create(`-vol`, vol_geom)
```

Allocates `float32` array in C++ interface

ID for reference in Python

Storage for volume data (e.g., the reconstruction)

Needs volume geometry as initializer, links to it

# Geometries in ASTRA

## Volume data

```
rec_id = astra.data2d.create('-vol', vol_geom)

rec_id = astra.data2d.create('-vol', vol_geom, 0)
astra.data2d.store(rec_id, 0)

rec_id = astra.data2d.create('-vol', vol_geom, V)
astra.data2d.store(rec_id, V)

V = astra.data2d.get(rec_id)

astra.data2d.delete(rec_id)
```

# Geometries in ASTRA

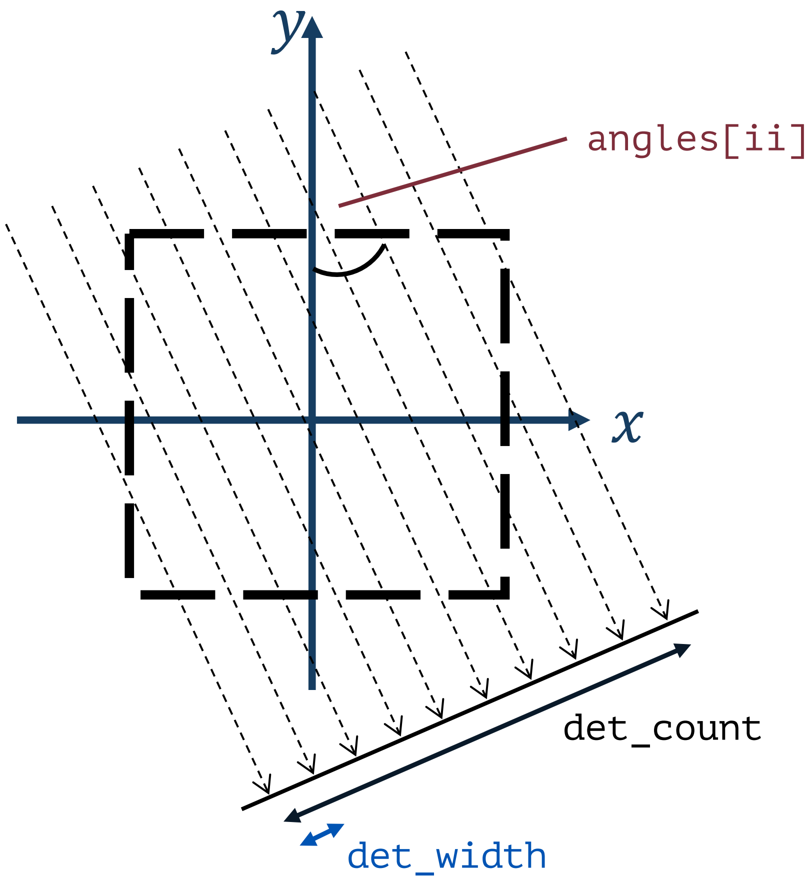
Projection geometry and data

- trajectory of source and detector plane
- number of detector elements
- detector size

1. parallel beam
2. fan-beam

# Geometries in ASTRA

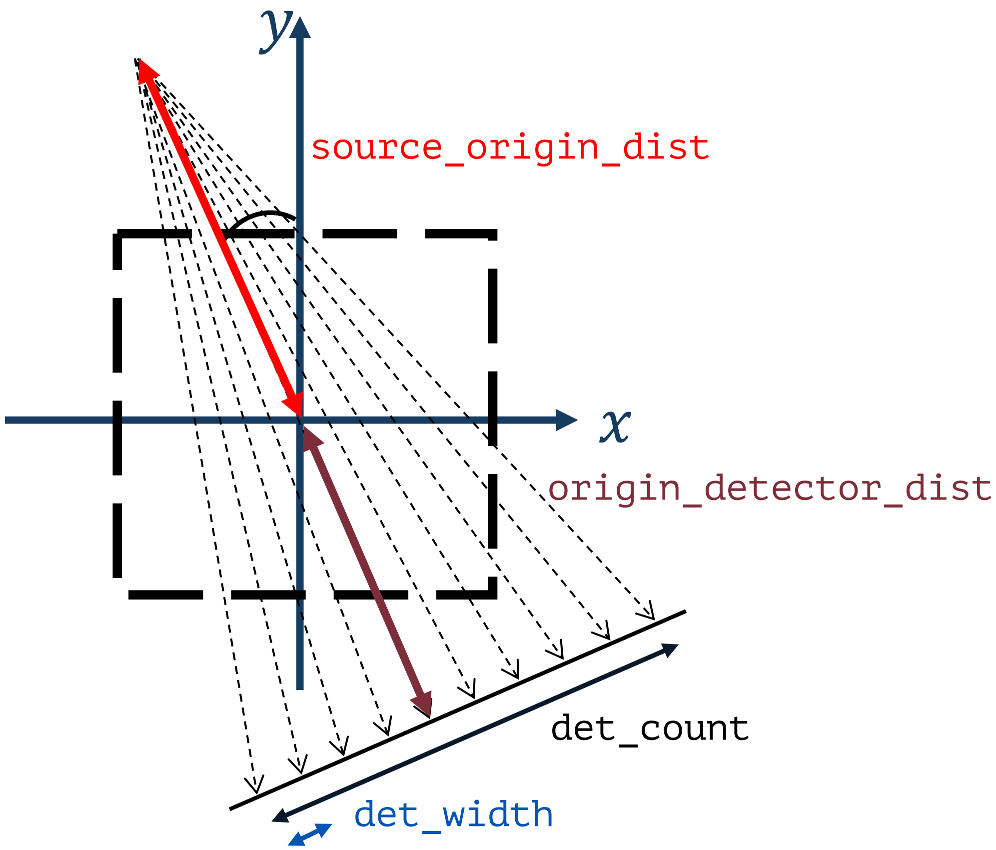
parallel projection geometry




```
angles = np.linspace(0, np.pi, 180, False)
proj_geom = astra.create_proj_geom('parallel', det_width, det_count, angles)
```

# Geometries in ASTRA

fan-beam projection geometry

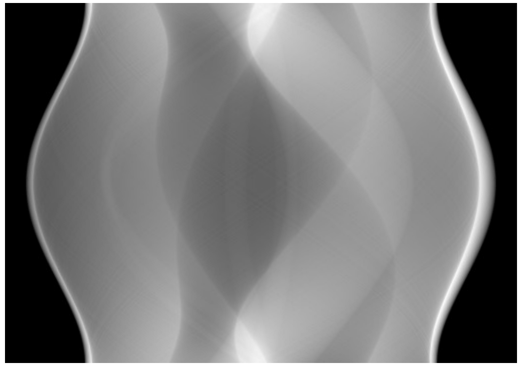



```
angles = np.linspace(0, 2*np.pi, 360, False)
proj_geom = astra.create_proj_geom('fanflat', det_width,
        det_count, angles, source_origin_dist, origin_detector_dist)
```

# Geometries in ASTRA

projection data

- place to store projection (i.e., sinogram)
- similar to volume data
- links to projection geometry



```
proj_id = astra.data2d.create('-sino', proj_geom)
proj_id = astra.data2d.create('-sino', proj_geom, 0)
proj_id = astra.data2d.create('-sino', proj_geom, V)
```

Image from: https://tomopy.readthedocs.io/en/latest/ipynb/astra.html

# Geometries in ASTRA

The projector object describes the way ASTRA computes the (implicit) system matrix.

For CPU algorithms:

- Multiple choices of projector
- Exactly matched forward and backprojection

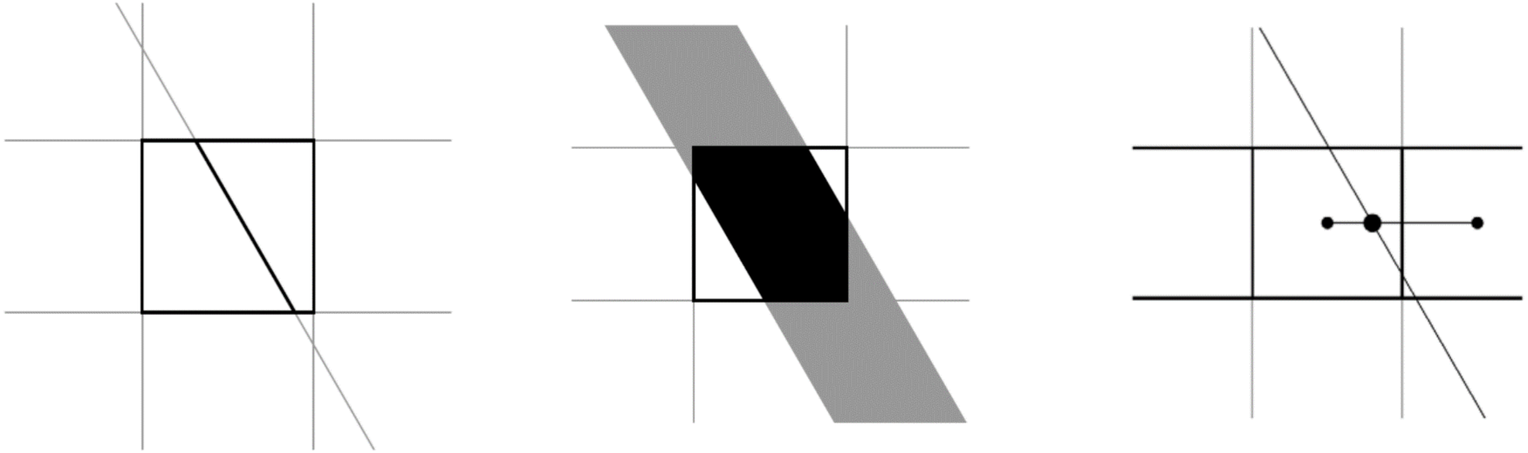
For GPU algorithms:

- A single choice of projector
- Only approximately matched forward and backprojection

# Geometries in ASTRA

For 2D CPU algorithms, ASTRA has different discretizations:

```
proj_id = astra.create_projector('line', proj_geom, vol_geom)
```

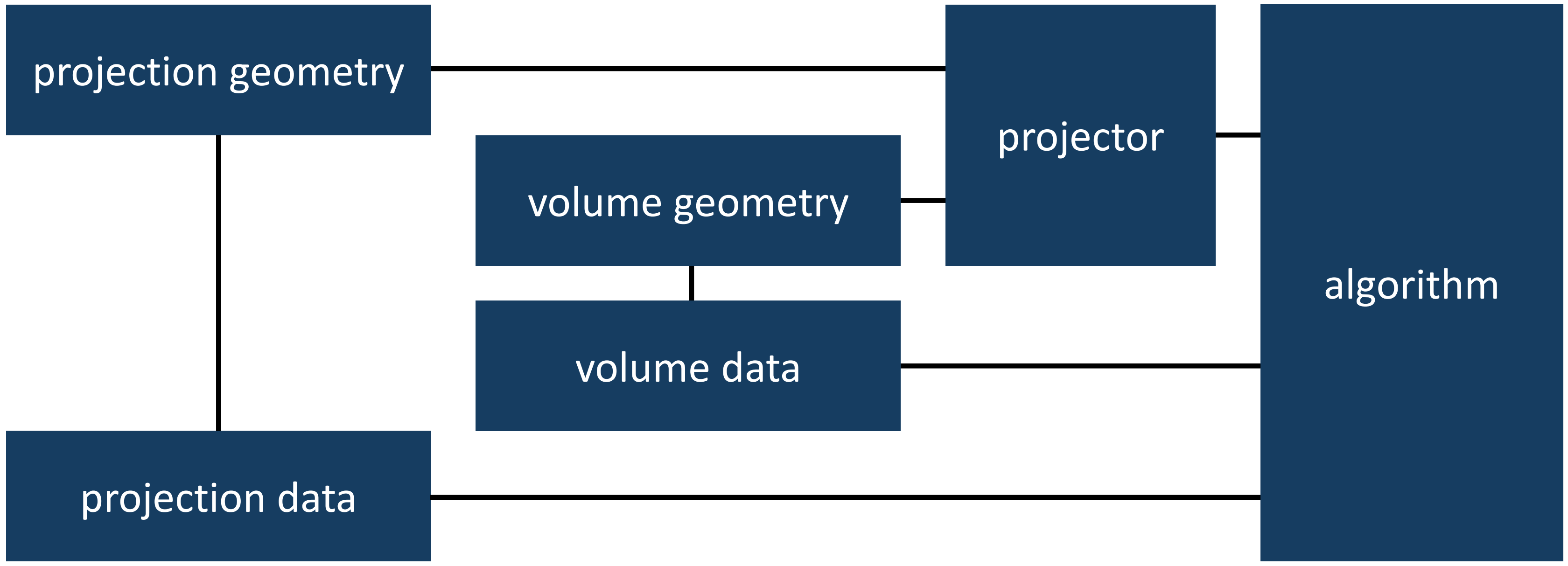


'line'
'line_fanflat'

'strip'
'strip_fanflat'

'linear'
(Joseph kernel)

# Geometries in ASTRA



# Reconstruction Algorithms

Faster

FBP

Iterative least-squares solvers

Variational methods, statistical methods

More flexible

# Reconstruction Algorithms

ASTRA provides

2D CPU

- FP, BP
- FBP
- ART, SART, SIRT, CGLS

2D GPU

- FP_CUDA, BP_CUDA
- FBP_CUDA
- SART_CUDA, SIRT_CUDA, CGLS_CUDA
- EM_CUDA

# Reconstruction – SIRT (CPU)

```
# Configure geometry
angles = np.linspace(0, 2*np.pi, nAngles, False)
proj_geom = astra.create_proj_geom('fanflat', 1.0, detectorCount, angles,
                                  originToSource, originToDetector)
vol_geom = astra.create_vol_geom(vy, vx)

# Data objects for input, output
sino_id = astra.data2d.create('-sino', proj_geom, S)
rec_id = astra.data2d.create('-vol', vol_geom)
proj_id = astra.create_projector('strip_fanflat', proj_geom, vol_geom)

# Configure algorithm
cfg = astra.astra_dict('SIRT')
cfg['ReconstructionDataId'] = rec_id
cfg['ProjectionDataId'] = sino_id
cfg['ProjectorDataId'] = proj_id
alg_id = astra.algorithm.create(cfg)

# Run
astra.algorithm.run(alg_id, 100)
rec = astra.data2d.get(rec_id)
```

# Reconstruction – SIRT (GPU)

```
# Configure geometry
angles = np.linspace(0, 2*np.pi, nAngles, False)
proj_geom = astra.create_proj_geom('fanflat', 1.0, detectorCount, angles,
                                   originToSource, originToDetector)
vol_geom = astra.create_vol_geom(vy, vx)

# Data objects for input, output
proj_id = astra.data2d.create('-sino', proj_geom, S)
rec_id = astra.data2d.create('-vol', vol_geom)

# Configure algorithm
cfg = astra.astra_dict('SIRT_CUDA')
cfg['ReconstructionDataId'] = rec_id
cfg['ProjectionDataId'] = proj_id
alg_id = astra.algorithm.create(cfg)

# Run
astra.algorithm.run(alg_id, 100)
rec = astra.data2d.get(rec_id)
```

# SciPy linear operator

```
vol_geom = astra.create_vol_geom(256, 256)
proj_geom = astra.create_proj_geom('parallel', 1.0, 256, np.linspace2(0, np.pi, 180, False))
proj_id = astra.create_projector('strip', proj_geom, vol_geom)


# Create OpTomo operator
W = astra.OpTomo(proj_id)


# Forward projection
s = W * P
s = s.reshape(astra.geom_size(proj_geom))


# Reconstruction using scipy's lsqr
output = scipy.sparse.linal.lsqr(W, s.ravel(), iter_lim=100)
rec = output[0].reshape(astra.geom_size(vol_geom))
```

# Flexible geometries

Example uses (2D and 3D):

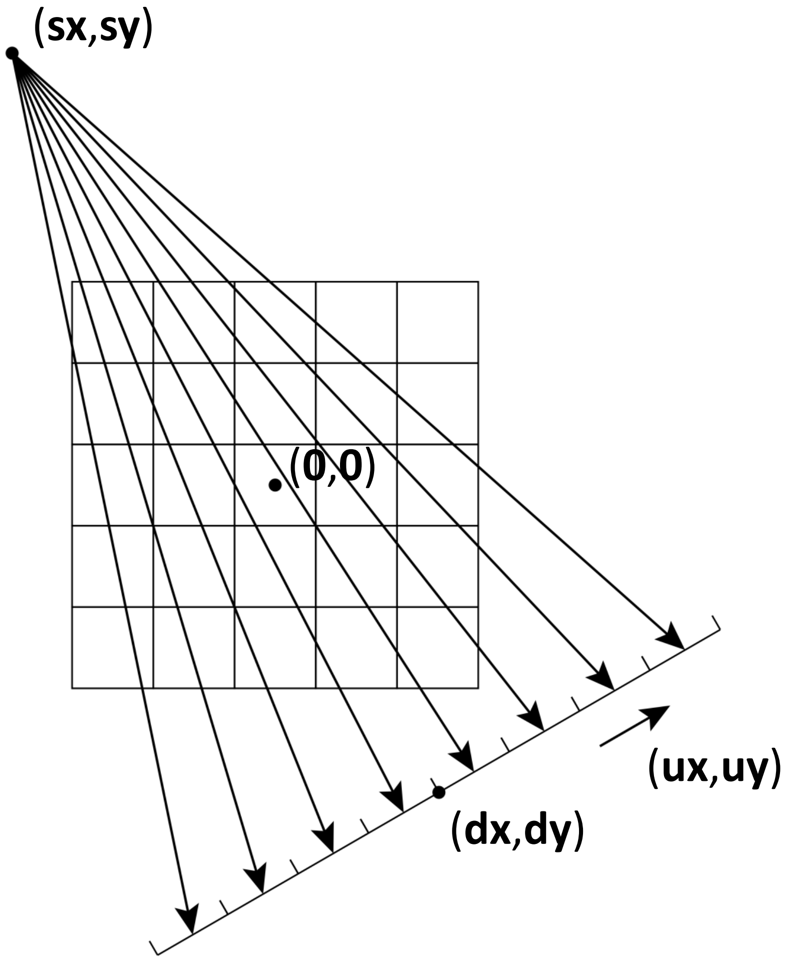
- Misalignment in experimental setup
- Dual axis datasets
- Laminography
- Single particle analysis
- Diffraction contrast tomography

# Fan beam – vector form

Three 2D parameters per projection:

**s, d, u**

These form a 6 element row vector.



# Fan beam – vector form

```
# One single projection angle
vectors = np.zeros((1, 6))
angle = 0.1
source_dist = 2000
# source
vectors[0,0] = np.sin(angle) * source_dist
vectors[0,1] = -np.cos(angle) * source_dist
# center of detector
vectors[0,2] = 0
vectors[0,3] = 0
# vector from detector pixel 0 to 1
vectors[0,4] = np.cos(angle) * 1.0
vectors[0,5] = np.sin(angle) * 1.0

proj_geom = astra.create_proj_geom('fanflat_vec', 256, vectors)
```

# Parallel beam – vector form

```
# Parallel beam - vector form
proj_geom = astra.create_proj_geom('parallel_vec', 256, vectors)
```

`vectors` consists of a 6 element row vector per projection.

Three 2D parameters per projection:

- **r**: Ray direction
- **d**: detector center
- **u**: detector pixel basis vector

# Conversion to vector form

Using utility function:

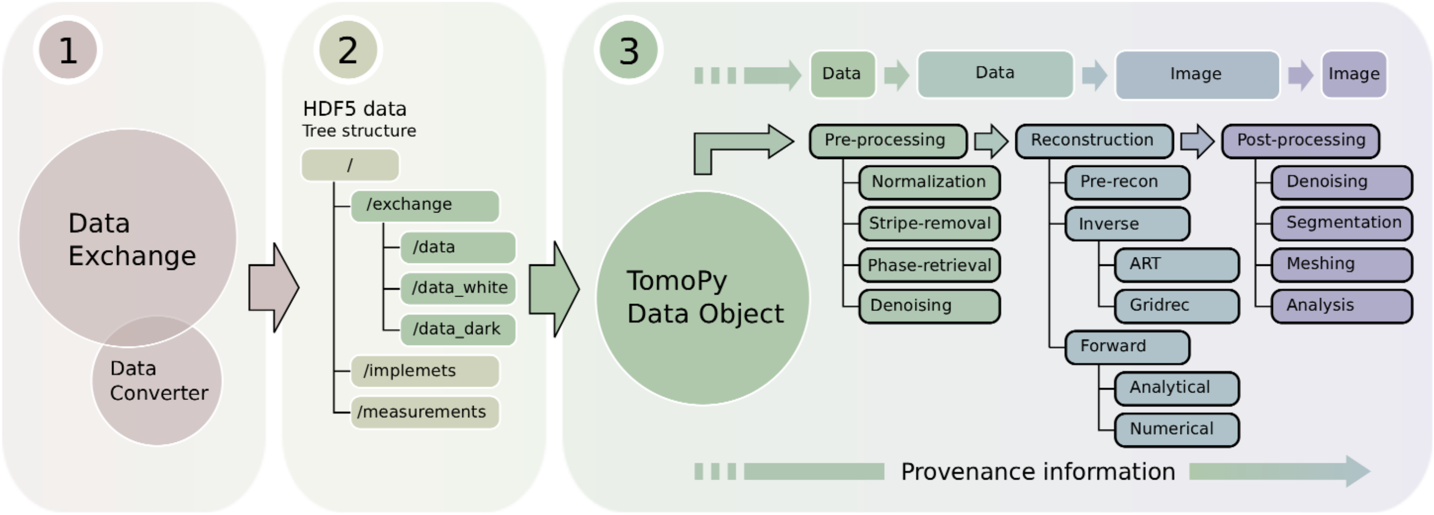
```
# Standard fan beam
proj_geom = astra.create_proj_geom('fanflat', 1.0, 256, angles, 2000, 0)

# Convert to vector form
proj_geom_vec = astra.geom_2vec(proj_geom)

# Center-of-rotation correction (by -3.5 pixels)
proj_geom_cor = astra.geom_postalignment(proj_geom, -3.5)
```

# Using ASTRA via TomoPy

TomoPy provides

*"a collaborative framework for the analysis of synchrotron tomographic data that has the goal to unify the effort of different facilities and beamlines performing similar tasks.''*

Developed by APS at Argonne National Laboratory (USA).



# Using ASTRA via ODL

> *"ODL is a python library for fast prototyping focusing on (but not restricted to) inverse problems.*
>
> *The main intent of ODL is to enable mathematicians and applied scientists to use different numerical methods on real-world problems without having to implement all necessary parts from the bottom up. ODL provides some of the most heavily used building blocks for numerical algorithms out of the box, which enables users to focus on real scientific issues."*

Developed by KTH Stockholm

# Benchmarks – NVIDIA Titan RTX

| 2D | FP per slice (ms) | Full volume FP (s) | BP per slice (ms) | Full volume BP (s) |
|---|---|---|---|---|
| 512 | 2.43 | 1.24 | 2.75 | 1.41 |
| 1024 | 6.80 | 6.96 | 7.59 | 7.72 |
| 2048 | 28.8 | 58.9 | 42.0 | 86.0 |
| 4096 | 194 | 795 | 300 | 1229 |

| 3D | | Full volume FP (s) | | Full volume BP (s) |
|---|---|---|---|---|
| 512 | | 1.00 | | 0.69 |
| 1024 | | 13.3 | | 7.75 |
| 2048 | | 179 | | 100 |
| 4096 | | 1736 | | 1460 |

NxNxN volume, N projections of NxN, parallel beam

# Useful links

- Webpage, for downloads, docs:

  https://www.astra-toolbox.com/

- Github, for source code, issue tracker:

  https://github.com/astra-toolbox/

- Email:

  astra@astra-toolbox.com

# Publications

Palenstijn et al, “*Performance improvements for iterative electron tomography reconstruction using graphics processing units (GPUs)*”, **J. of Structural Biology**, 2011

van Aarle, Palenstijn et al, “*The ASTRA Toolbox: A platform for advanced algorithm development in electron tomography*”, **Ultramicroscopy**, 2015

van Aarle, Palenstijn et al, “*Fast and flexible X-ray tomography using the ASTRA Toolbox*”, **Optics Express**, 2016

# Code example

# Now: Q&A and exercises

Exercises:

https://github.com/cicwi/xCTing_training

How does the # of iterations regularize the solution and does this differ for different reconstruction algorithms. Consider these sources of 'imperfection'

1. Additive Gaussian noise
2. Poisson noise
3. Structured noise (e.g., dead pixels)
4. Angular subsampling
5. Limited angular range