CWI

Universiteit Antwerpen

The ASTRA tomography toolbox
3D tomography

Willem Jan Palenstijn, Tim Elberfeld

Table of Contents

About us

Tim Elberfeld

Doctoral Researcher
imec-Vision Lab
Dept. Physics, University of Antwerp

Developing reconstruction algorithms with model priors

Willem Jan Palenstijn

Scientific Software Developer
Centrum Wiskunde & Informatica (CWI), Amsterdam

Lead developer on ASTRA Toolbox

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 algorithms and geometries	**Powerful** C++ and CUDA
Easy to use MATLAB and Python interface	**Free** 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 in 2019

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

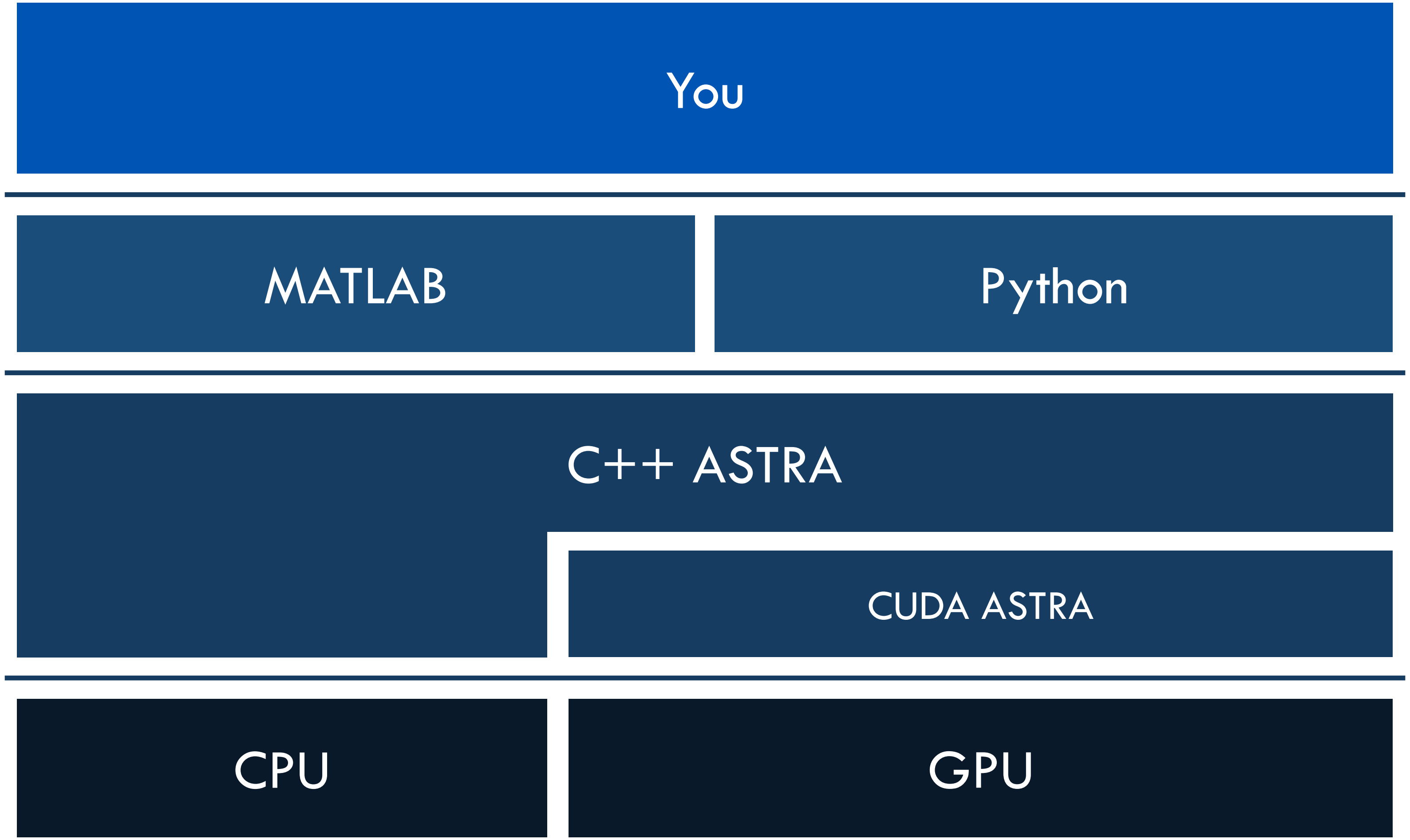
Data alignment, preprocessing

Matched GPU FP, BP operators

- CPU FP, BP are matched; GPU are not

Modern advanced reconstruction algorithms

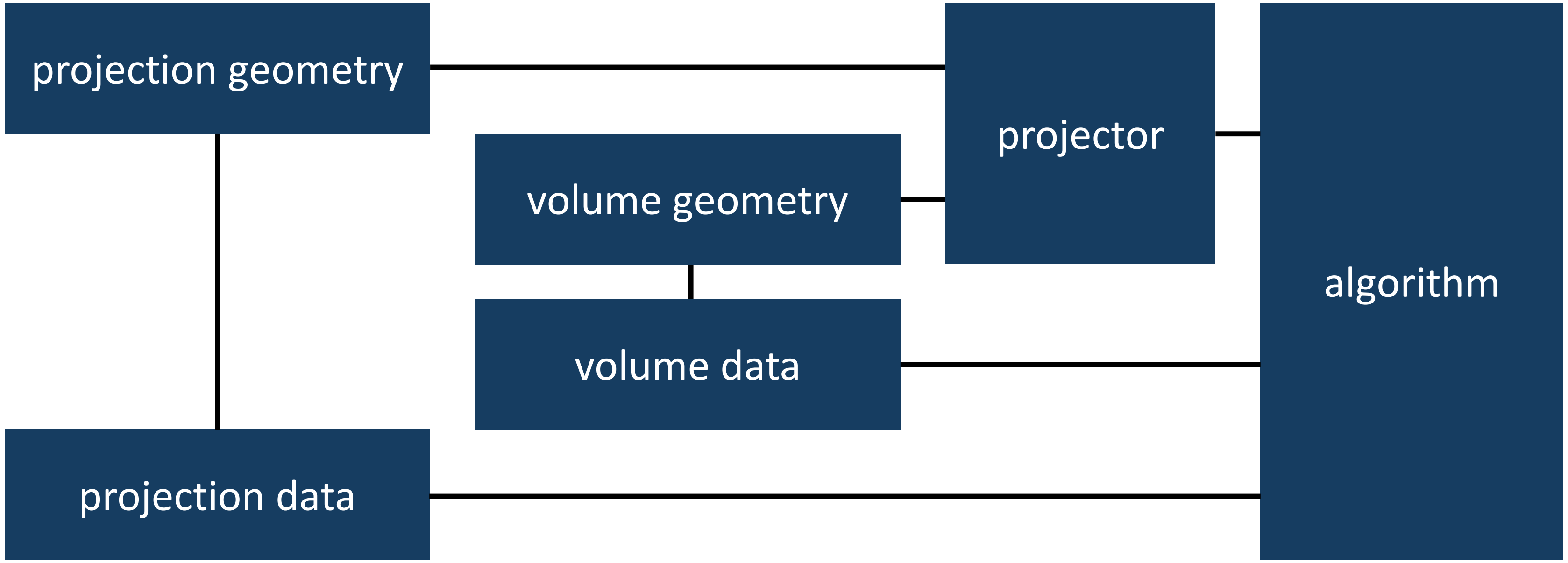
ASTRA Architecture



Geometries in ASTRA

Modeling of the x-ray scanning setup

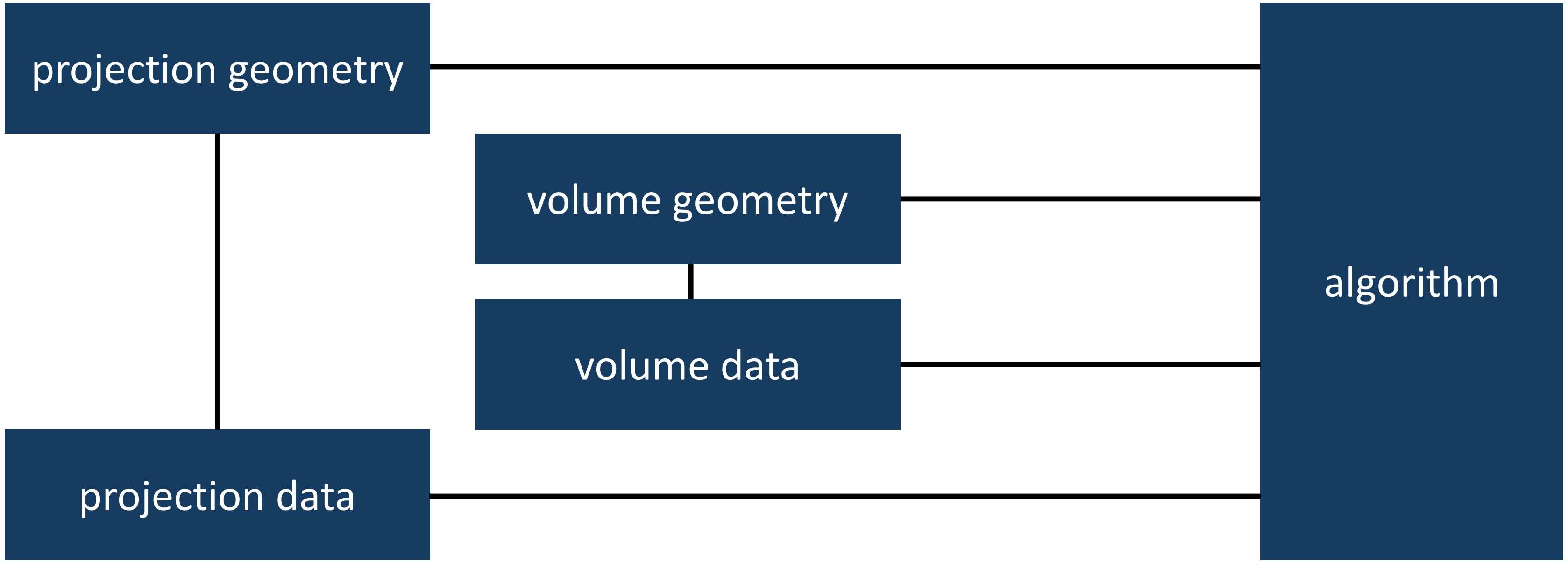
- Phantom / volume
- Source
- Detector



Geometries in ASTRA

Modeling of the x-ray scanning setup

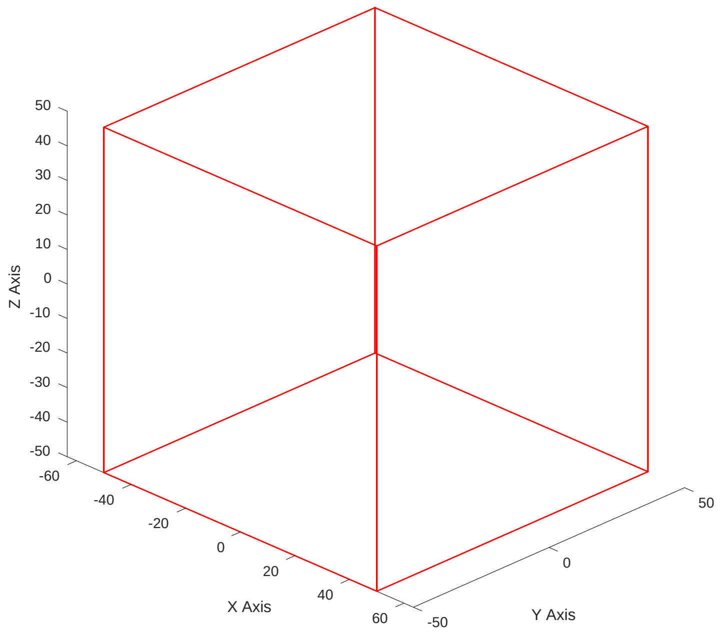
- Phantom / volume
- Source
- Detector



Geometries in ASTRA

Volume geometry and volume data

```
vol_geom = astra.create_vol_geom(ny, nx, nz)
```



limitation:

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

Geometries in ASTRA

Volume geometry and volume data

```
rec_id = astra.data3d.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, is tied to it

Geometries in ASTRA

Volume data

```
rec_id = astra.data3d.create('-vol', vol_geom)

rec_id = astra.data3d.create('-vol', vol_geom, 0)
astra.data3d.store(rec_id, 0)

rec_id = astra.data3d.create('-vol', vol_geom, V)
astra.data3d.store(rec_id, V)

V = astra.data3d.get(rec_id)

astra.data3d.delete(rec_id)
```

Geometries in ASTRA

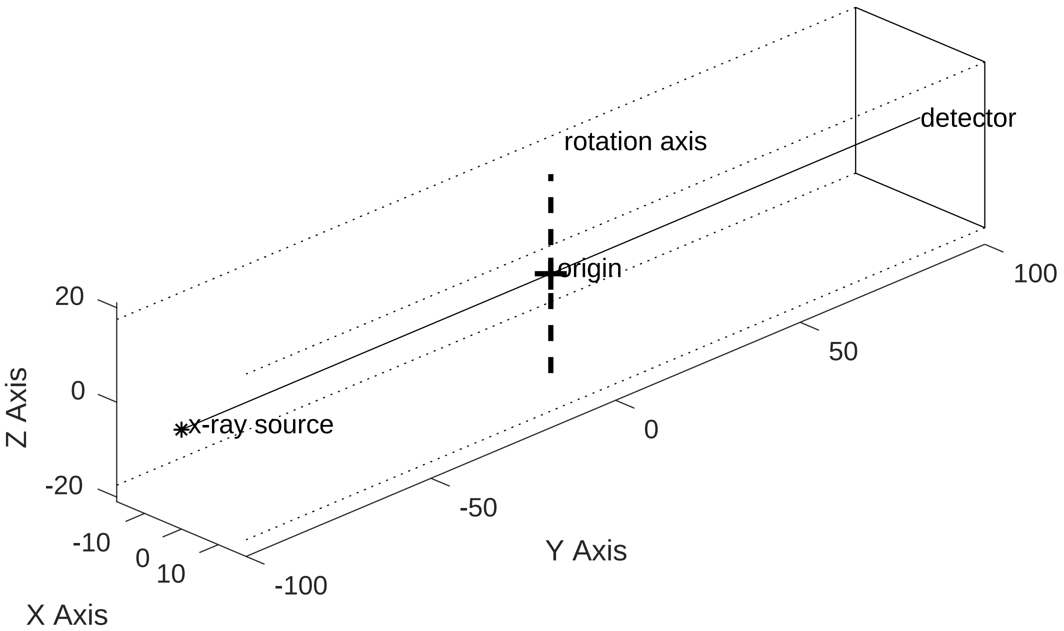
Projection geometry and data

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

1. parallel beam
2. cone beam

Geometries in ASTRA

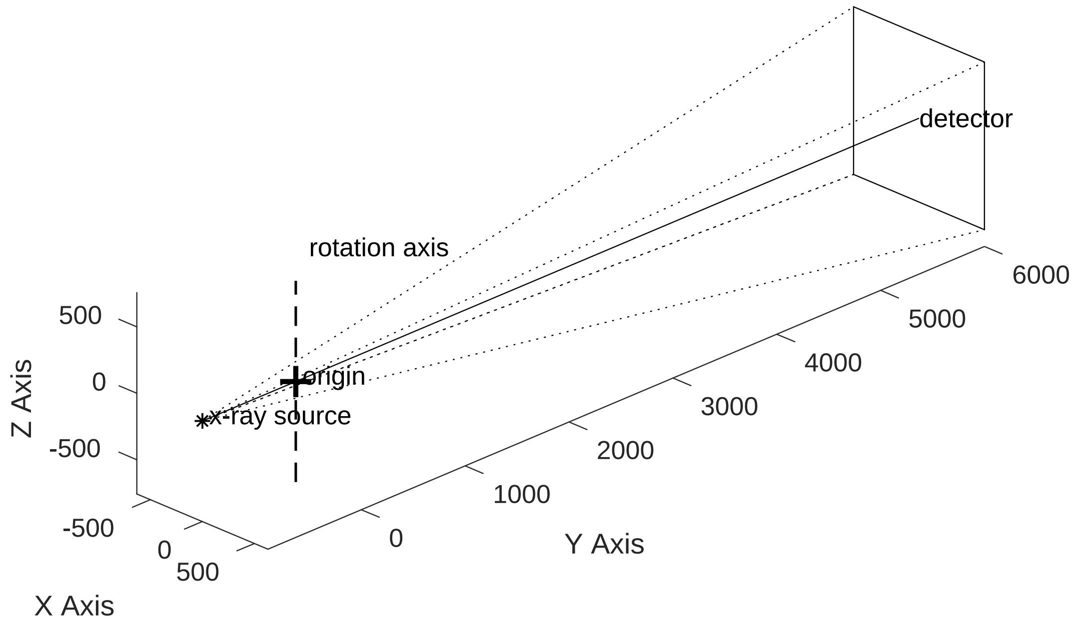
parallel projection geometry




```
angles = np.linspace(0, np.pi, 180, False)
proj_geom = astra.create_proj_geom('parallel3d', det_width, det_height, det_rows, det_cols, angles)
```

Geometries in ASTRA

cone beam projection geometry

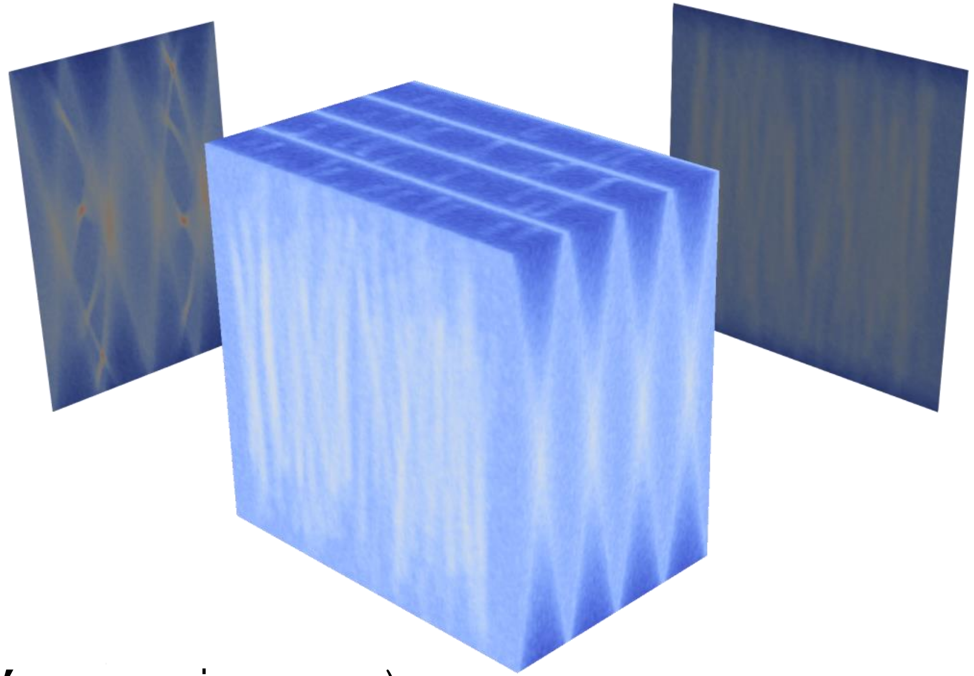



```
angles = np.linspace(0, 2*np.pi, 360, False)
proj_geom = astra.create_proj_geom('cone', det_width, det_width, det_height,
        det_rows, det_cols, angles, source_origin_dist, origin_detector_dist)
```

Geometries in ASTRA

projection data

- place to store projections
- similar to volume data
- tied to projection geometry

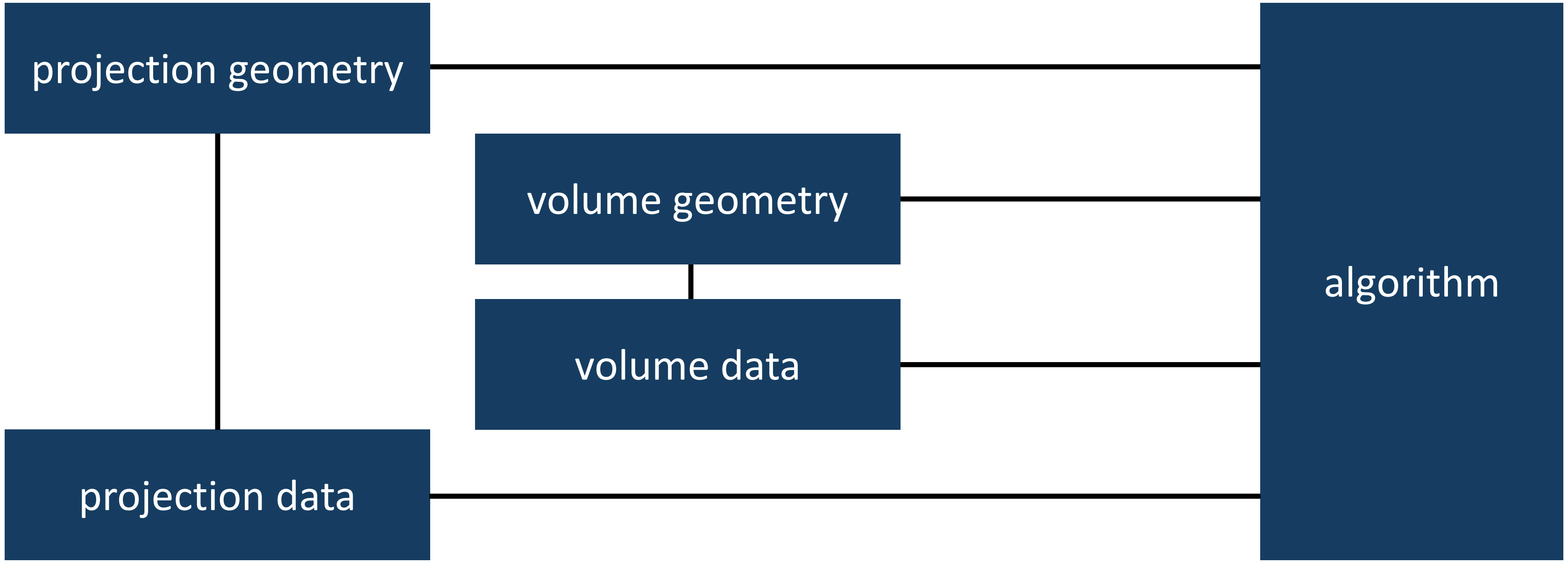


```
proj_id = astra.data3d.create('-proj3d', proj_geom)
proj_id = astra.data3d.create('-proj3d', proj_geom, 0)
proj_id = astra.data3d.create('-proj3d', proj_geom, V)
```

Geometries in ASTRA

Modeling of the x-ray scanning setup

- Phantom / volume
- Source
- Detector



Reconstruction Algorithms

Faster

FBP, FDK

Iterative least-squares solvers

Variational methods, statistical methods

"Better"

Reconstruction Algorithms

ASTRA provides

3D GPU

- FP3D_CUDA, BP3D_CUDA
- FDK_CUDA
- SIRT3D_CUDA, CGLS3D_CUDA

Reconstruction – SIRT

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


# Data objects for input, output
proj_id = astra.data3d.create('-proj3d', proj_geom, P)
rec_id = astra.data3d.create('-vol', vol_geom)


# Configure algorithm
cfg = astra.astra_dict('SIRT3D_CUDA')
cfg['ReconstructionDataId'] = rec_id
cfg['ProjectionDataId'] = proj_id
alg_id = astra.algorithm.create(cfg)


# Run
astra.algorithm.run(alg_id, 100)
rec = astra.data3d.get(rec_id)
```

SciPy linear operator

```
vol_geom = astra.create_vol_geom(256, 256, 256)
proj_geom = astra.create_proj_geom('parallel3d', 1.0, 1.0, 256, 256,
                                   np.linspace2(0, np.pi, 180, False))
proj_id = astra.create_projector('cuda3d', 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))
```

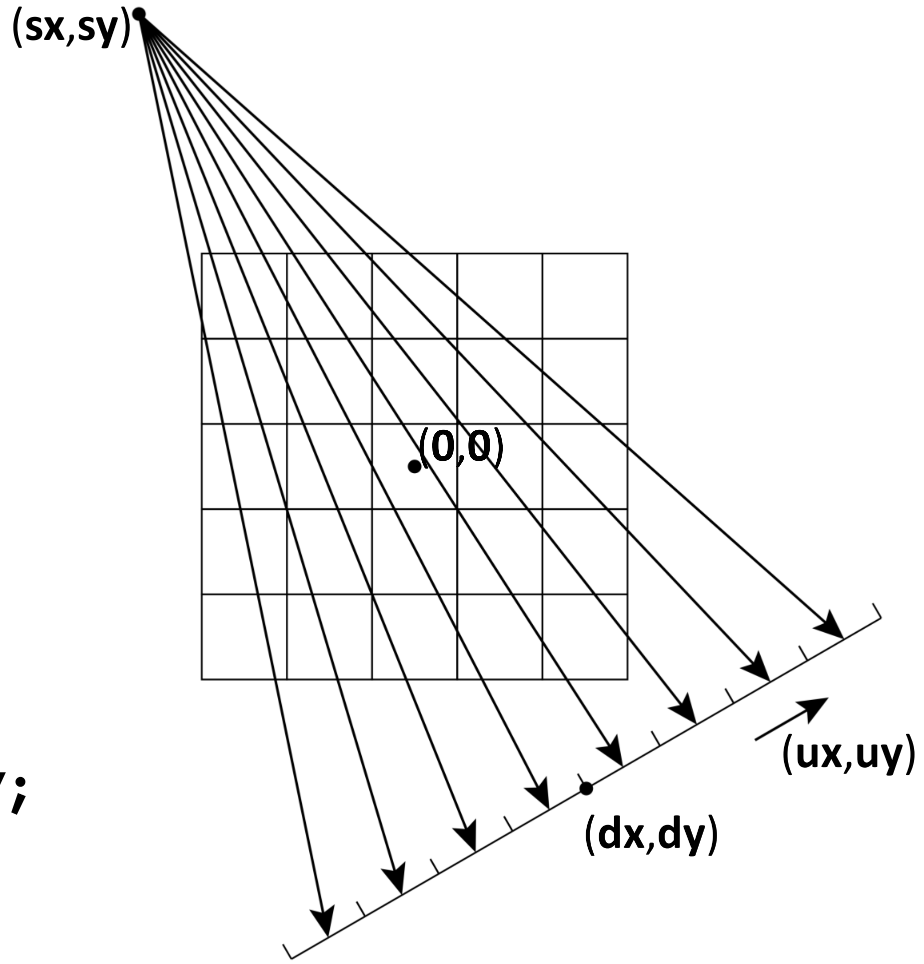
Fan beam – vector form

Three 2D parameters per projection:

s, d, u

These form a 6 element row vector.

Recall: sample is stationary; source and detector move



Cone beam – vector form

```
# Cone - vector form
proj_geom = astra.create_proj_geom('cone_vec', 256, 256, vectors)
```

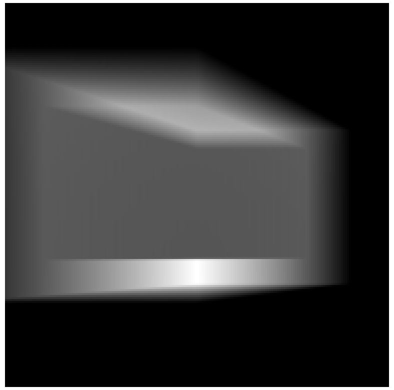
`vectors` consists of a 12 element row vector per projection.

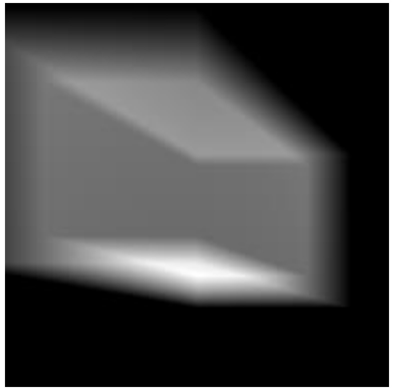
Four 3D parameters per projection:

- **s**: source location
- **d**: detector center
- **u**: horizontal detector pixel basis vector
- **v**: vertical detector pixel basis vector

Cone beam – vector form

```
vectors = [[ 192, 30, 192, 0, 0, 0, 1, 0, 0, 0, 1, 0 ],
            [ 192, 90, 192, 0, 0, 0, 1, 0, 0, 0, 1, 0 ]]
```





Parallel beam – vector form

```
# Parallel beam - vector form
proj_geom = astra.create_proj_geom('parallel3d_vec', 256, 256, vectors)
```

`vectors` consists of a 12 element row vector per projection.

Four 3D parameters per projection:

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

Conversion to vector form

Using utility function:

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

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

# Center-of-rotation correction (by -3.5 pixels horizontally)
proj_geom_cor = astra.geom_postalignment(proj_geom, [-3.5, 0])
```

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

ASTRA for large data sets

Historically, a major limitation of ASTRA has been the requirement that data fits in GPU memory.

With ASTRA 1.8, we've started removing this limit.

Current status:

- 3D FP, BP and FDK transparently handle this; limited by host memory.
- Other reconstruction algorithms don't directly

ASTRA for large data sets

For FP, BP, FDK this works transparently:

```
proj_geom = astra.create_proj_geom('parallel3d', 1.0, 1.0, 1024, 1024, angles)
vol_geom = astra.create_vol_geom(1024, 1024, 1024)
id, projdata = astra.create_sino3d_cuda(voldata, proj_geom, vol_geom)
id, voldata = astra.create_backprojection3d_cuda(projdata, proj_geom, vol_geom)
```

Also works transparently with OpTomo operator

To help with performance, you can also use multiple GPUs:

```
astra.set_gpu_index([0, 1])
```

CWI

Universiteit Antwerpen

ASTRA for large data sets – 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

ASTRA for large data sets

Reducing memory usage:

- Use `dtype=np.float32` to use single-precision floats instead of double
- Share memory between numpy and ASTRA:
  - `astra.data3d.get_shared(id)`
  - `astra.data3d.link` (instead of `astra.data3d.create`)

ASTRA for large data sets

SIRT3D_CUDA and CGLS3D_CUDA require all data (plus temporaries) to fit in GPU memory.

As a workaround use (provided) SIRT, CGLS algorithms that operate in host memory:

```
astra.plugin.register(astra.plugins.SIRTPlugin)
astra.plugin.register(astra.plugins.CGLSPlugin)
```

Then use algorithms SIRT-PLUGIN, CGLS-PLUGIN

Also see sample script s018_plugin.py

Distributed ASTRA

Major new feature in ASTRA 1.7 (Dec 2015), in separate mpi branch:

Distributed ASTRA: run ASTRA on a (GPU) cluster

- Process larger data sets faster
- A toolbox of building blocks for your (and our) own algorithms
- (Most of) the flexibility of ASTRA
- Python interface (only)

Developed by Jeroen Bédorf at CWI.

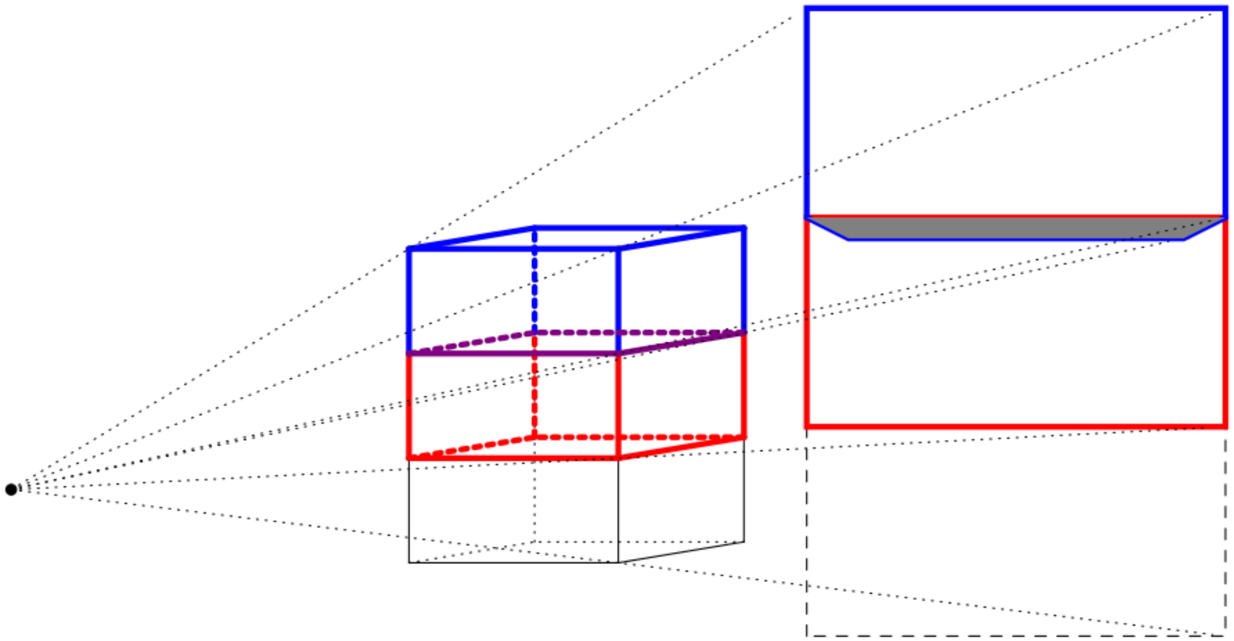
Distributed ASTRA

Our approach:

Data objects are distributed, and (some) 3D algorithms will automatically run distributed:

- SIRT
- CGLS
- Forward projection
- Backprojection

Distributed ASTRA



(Un)distributed ASTRA

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


# Data objects for input, output
proj_id = astra.data3d.create('-proj3d', proj_geom, P)
rec_id = astra.data3d.create('-vol', vol_geom)

# Configure algorithm
cfg = astra.astra_dict('SIRT3D_CUDA')
cfg['ReconstructionDataId'] = rec_id
cfg['ProjectionDataId'] = proj_id
alg_id = astra.algorithm.create(cfg)

# Run
astra.algorithm.run(alg_id, 100)
rec = astra.data3d.get(rec_id)
```

Distributed ASTRA

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


# Set up the distribution of data objects
proj_geom, vol_geom = mpi.create(proj_geom, vol_geom)


# Data objects for input, output
proj_id = astra.data3d.create('-proj3d', proj_geom, P)
rec_id = astra.data3d.create('-vol', vol_geom)

# Configure algorithm
cfg = astra.astra_dict('SIRT3D_CUDA')
cfg['ReconstructionDataId'] = rec_id
cfg['ProjectionDataId'] = proj_id
alg_id = astra.algorithm.create(cfg)

# Run
astra.algorithm.run(alg_id, 100)
rec = astra.data3d.get(rec_id)
```

Distributed ASTRA

The mpi branch of 1.7 is very experimental and a bit out of date.

Currently developing new distributed version, with more efficient partitioning/distribution of data, with Jan-Willem Buurlage (CWI)

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
  willem.jan.palenstijn@cwi.nl
  tim.elberfeld@uantwerpen.be

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

Palenstijn, Bédorf et al, "*A distributed ASTRA Toolbox*", **Adv. Struct. & Chem. Imag. 2016**

Code example

Now: Q&A and exercises

Installation instructions and exercises:

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

Also, feel free to ask us anything