CWI

Universiteit Antwerpen

# The ASTRA Tomography Toolbox

5 April 2016

## Advanced topics

CWI

Universiteit Antwerpen

# Today

- Introduction to ASTRA
- Exercises
- More on ASTRA usage
- Exercises
- **Extra topics**
- Hands-on, questions, discussion

CWI

Universiteit Antwerpen

# GPU Usage

CWI

Universiteit Antwerpen

# GPU Usage

- Besides the 2D CPU code you've used, ASTRA has accelerated GPU code for 2D and 3D tomography for NVIDIA GPUs.
- Usage of the 2D GPU code is very similar to what you've seen. Look at the GPU samples for more information.
- One difference is that there is currently no choice of discretization, and the backprojection is **not** the exact mathematical transpose of the forward projection for GPU code.
- 3D in ASTRA is currently GPU-only.

CWI

Universiteit Antwerpen

# 3D Geometries

CWI

Universiteit Antwerpen

# Flexible geometries

- ASTRA supports:
  - 2D: Parallel beam, fan beam
  - 3D: Parallel beam, cone beam

- All with fully flexible source/detector positioning

# 3D Geometries – basic

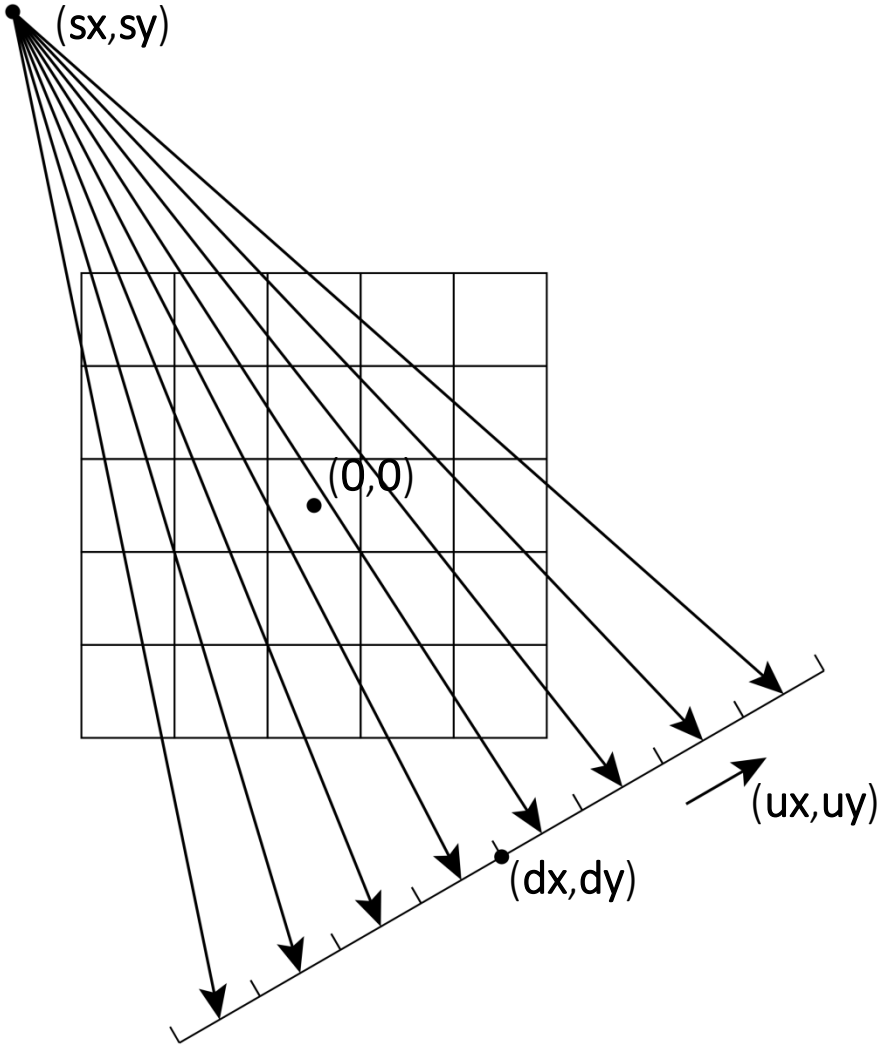
```
angles = linspace2(0, 2*pi, 180);

% Parallel
proj_geom = astra_create_proj_geom('parallel3d', 1.0, 1.0, ...
    256, 256, angles);


% Cone
proj_geom = astra_create_proj_geom('cone', 1.0, 1.0, ...
    256, 256, angles, 2000, 0);
```

# Fan beam – vector form

Three 2D parameters per projection:

**s, d, u**

These form a 6 element row vector.



# Cone beam – vector form

```
% Cone - vector form
proj_geom = astra_create_proj_geom('cone_vec', ...
    256, 256, vectors);
```

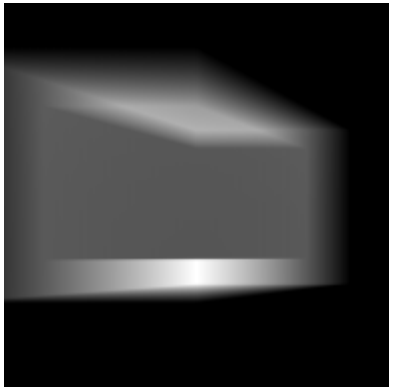
`vectors` consists of a 12 element row vector per projection.

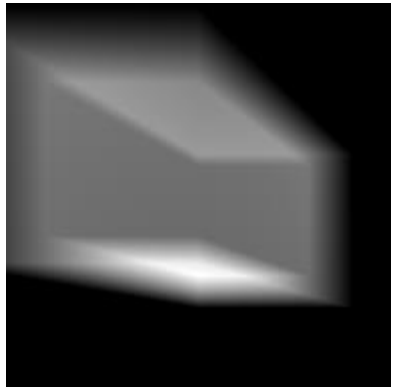
Four 3D parameters per projection:

- **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

CWI

Universiteit Antwerpen

# Large 3D data sets

CWI

Universiteit Antwerpen

# ASTRA for large data sets

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

With ASTRA 1.7, we've started removing this limit.

Current status:

- 3D Forward and backprojection transparently handle this.
- Reconstruction algorithms don't (yet)

# ASTRA for large data sets

## For FP, BP 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);

[x, projdata] = astra_create_sino3d_cuda(voldata, proj_geom, vol_geom);
[y, voldata] = astra_create_backprojection3d_cuda(projdata, proj_geom, vol_geom);
```

## Also works transparently with Spot operator

## Upcoming feature (next release, “soon”):

- Multi-GPU support for 3D FP/BP

```
astra_mex('set_gpu_index', [0 1]);
```

CWI

Universiteit Antwerpen

# Distributed ASTRA

Major new feature in ASTRA 1.7 (Dec 2015):

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.

CWI

Universiteit Antwerpen

# Distributed ASTRA

Our approach:

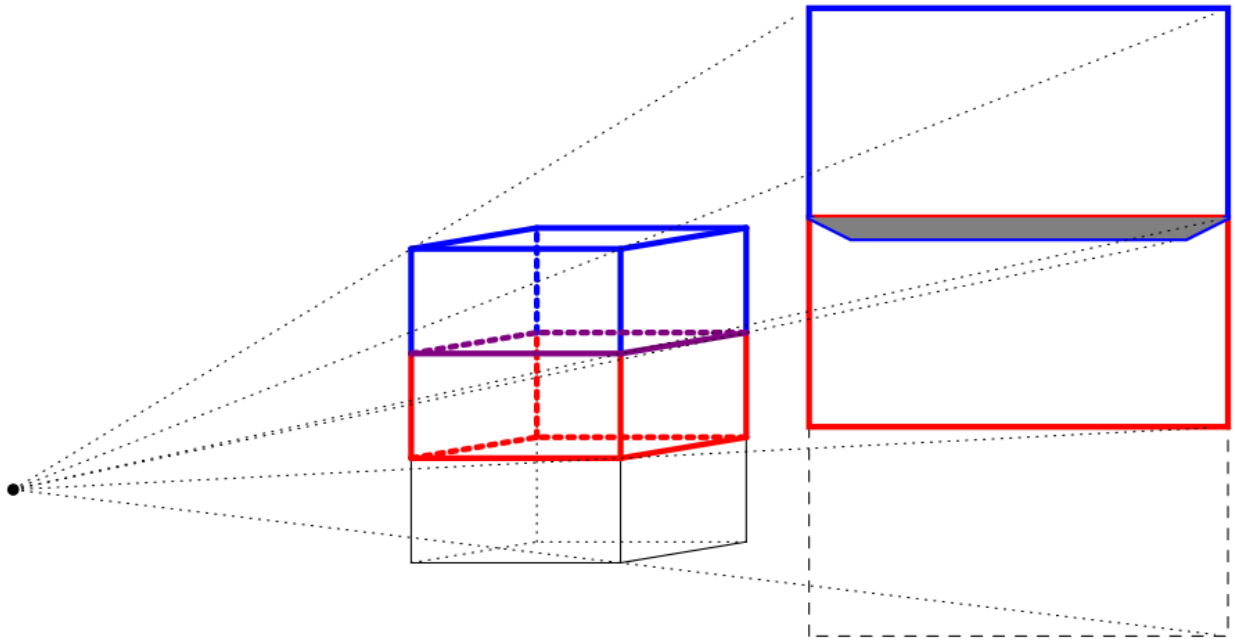
Data objects are distributed, and (some) 3D algorithms will automatically run distributed:

- SIRT
- CGLS
- Forward projection
- Backprojection

CWI

Universiteit Antwerpen

# Distributed ASTRA



CWI

Universiteit Antwerpen

# (Un)distributed ASTRA – (Python!)

```
# 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)
```

CWI

Universiteit Antwerpen

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

CWI

Universiteit Antwerpen

# Distributed ASTRA

It is also possible to run your own functions on distributed data:

```
def addScaledArray(src_id, dst_id, value):
 dataS = astra.data3d.get_shared_local(src_id)
 dataD = astra.data3d.get_shared_local(dst_id)
 dataD[:] = dataD + value*dataS2


mpi.run(addScaledArray, [tmp_id, vol_id, lambda])
```

Also functions that return output are possible, such as for example taking norms or inner products over the full volume.

CWI

Universiteit Antwerpen

# Distributed ASTRA

Running your own functions on distributed data:

- Many possibilities:
  basic arithmetic, segmentation, inner products, norms, file I/O, …

- Some operations, such as edge detection, blurring, etc, look at multiple pixels at once, which can be in neighbouring volume blocks.

  We handle this by (optionally) letting the volume blocks overlap slightly.

CWI

Universiteit Antwerpen

# Distributed ASTRA – Limitations

Limitations:

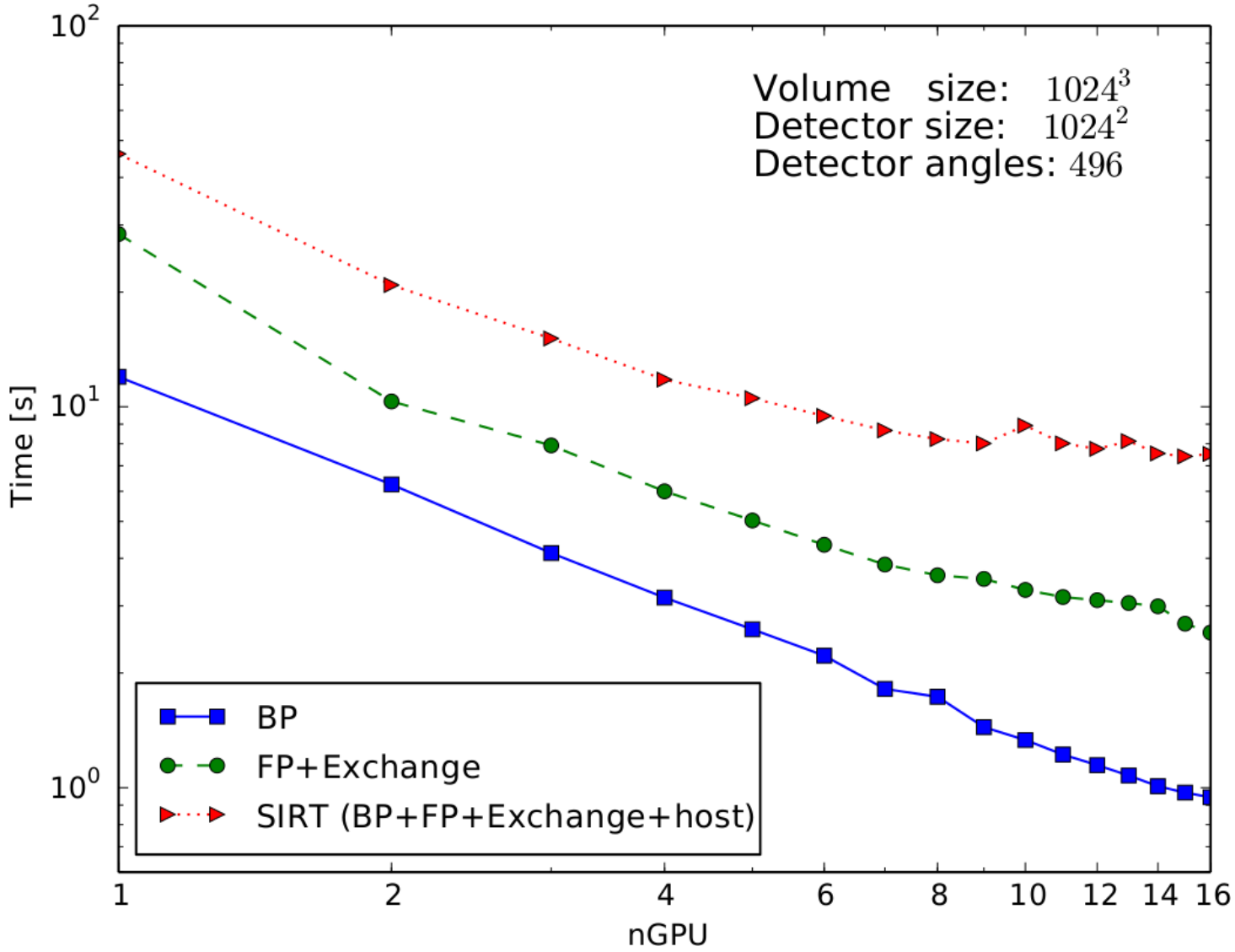
- Python only
- Linux only

- Currently, we always only split volumes in slabs along the z-axis. For this to work, the projection geometry must (approximately) rotate around the z-axis.

CWI

Universiteit Antwerpen

# Distributed benchmarks



CWI

Universiteit Antwerpen

# TomoPy

CWI

Universiteit Antwerpen

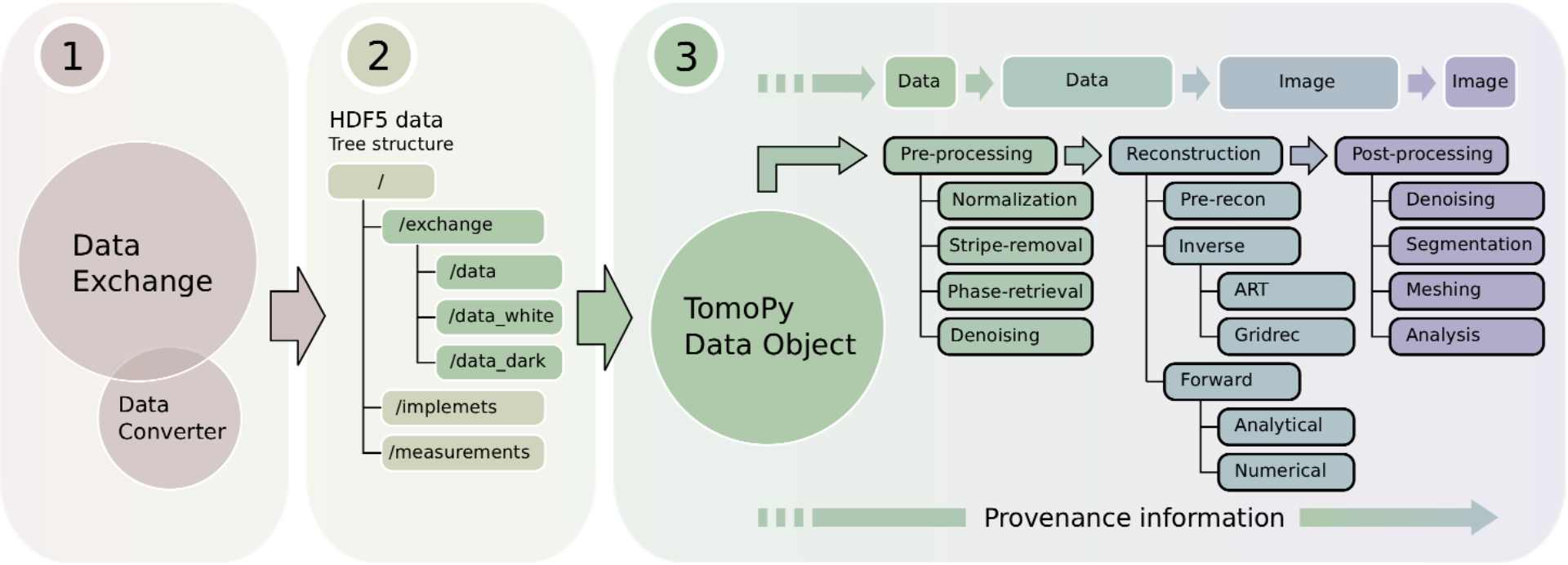
# TomoPy

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).

# TomoPy


1
Data Exchange
Data Converter
2
HDF5 data
Tree structure
/
/exchange
/data
/data_white
/data_dark
/implemets
/measurements
3
TomoPy Data Object
Data
Data
Image
Image
Pre-processing
Normalization
Stripe-removal
Phase-retrieval
Denoising
Reconstruction
Pre-recon
Inverse
ART
Gridrec
Forward
Analytical
Numerical
Post-processing
Denoising
Segmentation
Meshing
Analysis
Provenance information

CWI

Universiteit Antwerpen

# TomoPy + ASTRA

With APS, Daniel Pelt (CWI) has integrated TomoPy and ASTRA.

This provides, from TomoPy:

- Data import/export
- Pre-/post-processing
  - Finding center of rotation, ring artefact correction, …
- Parallelization of large data sets in parallel slices

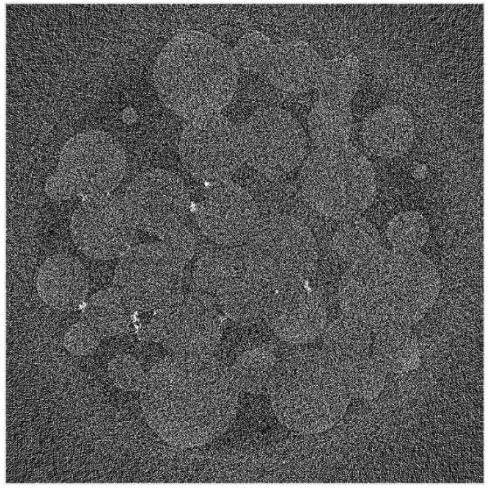
And from ASTRA:

- GPU acceleration
- Building blocks for your own reconstruction algorithms
- …

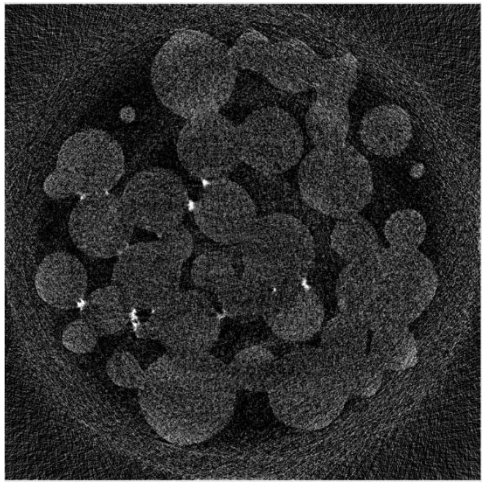
# TomoPy + ASTRA

This interface is currently available for testing.

See the TomoPy documentation on how to get and install the necessary modules. (Search for TomoPy astra)



TomoPy gridrec



TomoPy preproc +
ASTRA GPU SIRT

CWI

Universiteit Antwerpen

# The Future

CWI

Universiteit Antwerpen

# Plans

- More support for larger data sets
  - Including for reconstruction algorithms
  - Improved performance, smarter automatic subdivision of data
- Support for more projection kernels / discretizations on GPU
- Better support for block algorithms on top of ASTRA
- More flexible volume geometries (in progress)
  - Not necessarily centered around the origin
  - Not necessarily square/cubic voxels

CWI

Universiteit Antwerpen

# Useful links

- Webpage, for downloads, docs:

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

- Github, for source code, issue tracker:

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

- Email:

  astra@uantwerpen.be

  willem.jan.palenstijn@cwi.nl

CWI

Universiteit Antwerpen

# Today

- Introduction to ASTRA
- Exercises
- More on ASTRA usage
- Exercises
- Extra topics
- **Hands-on, questions, discussion**