Adventures in Reciprocal Space

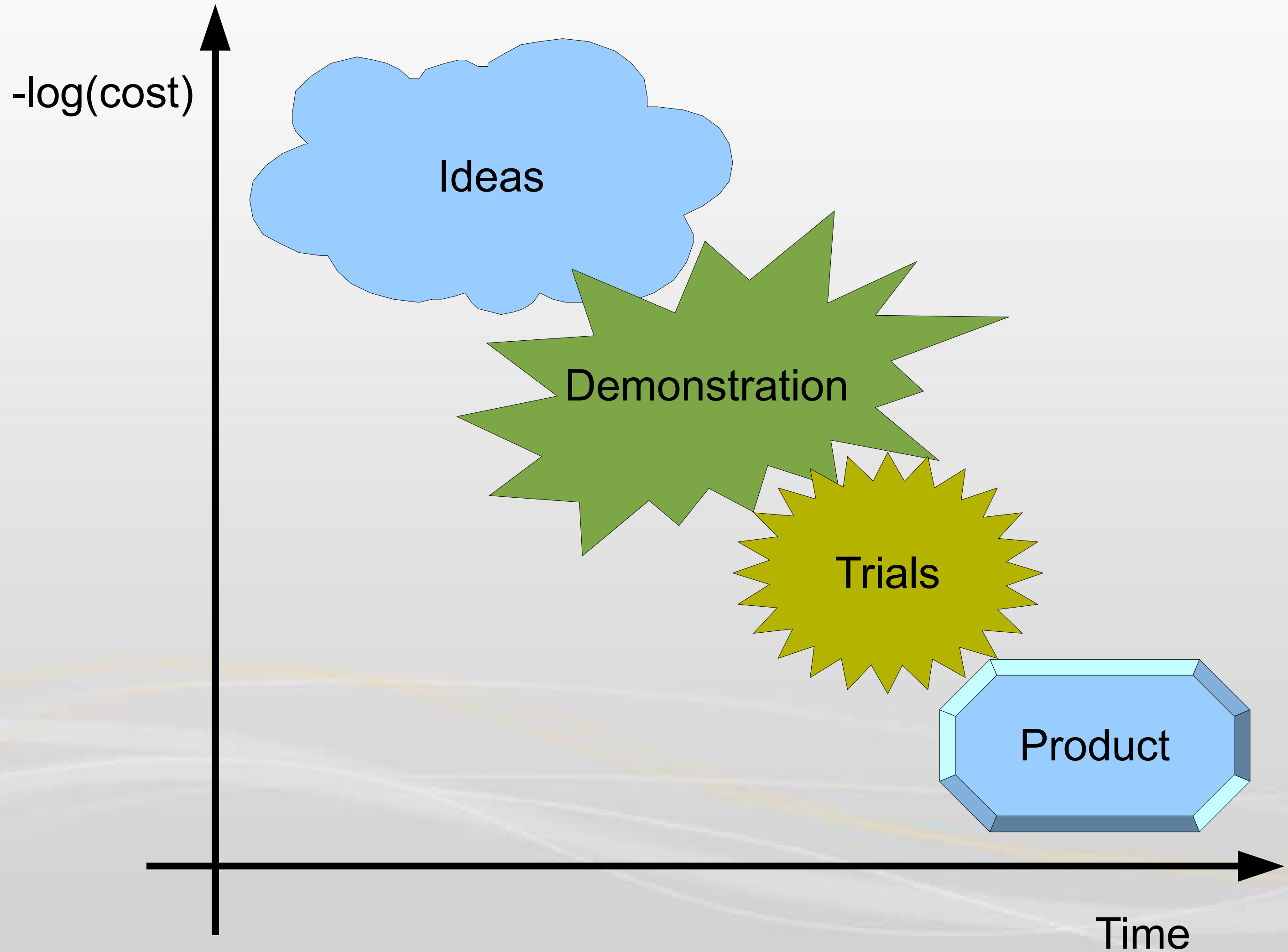
Jon Wright

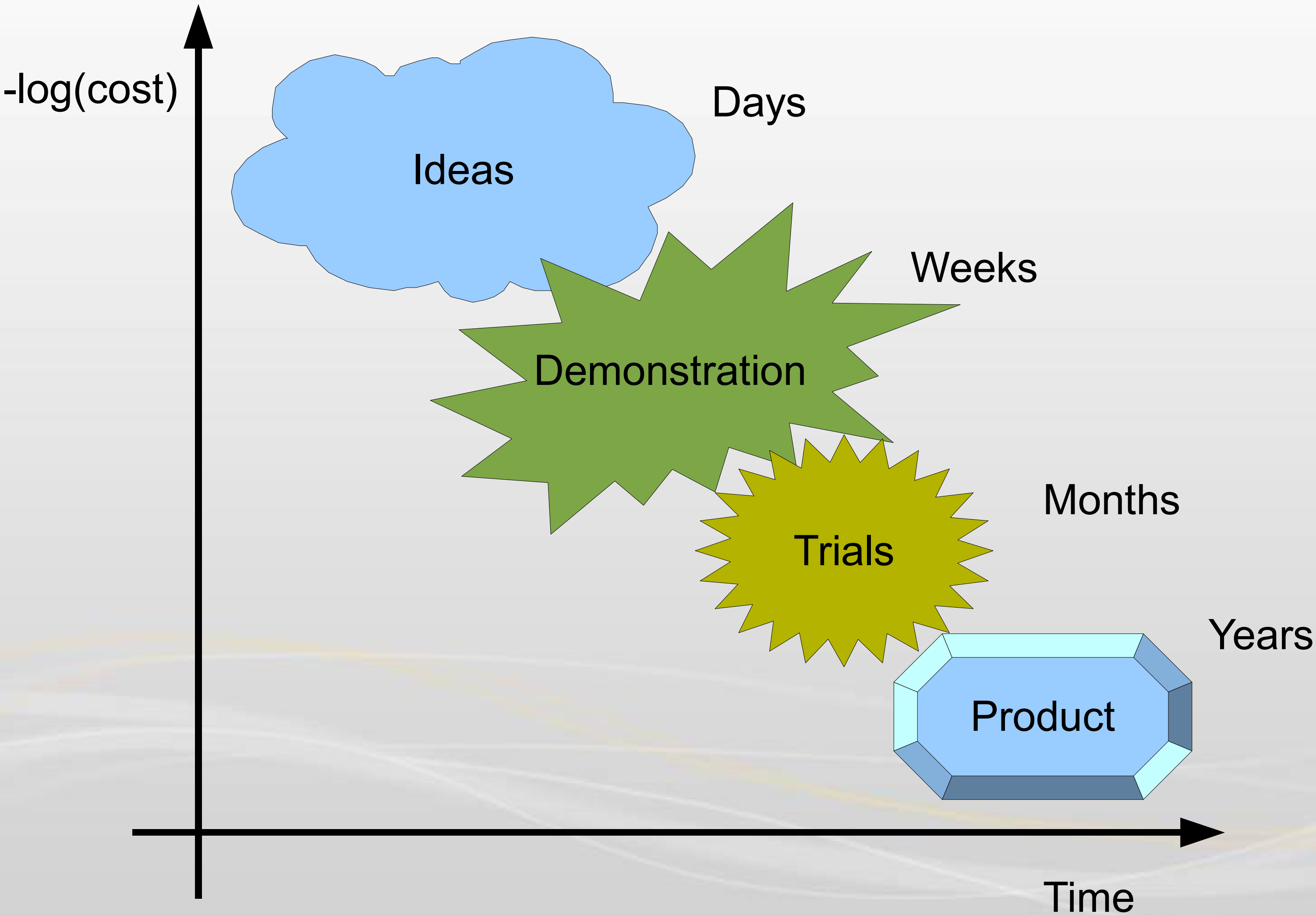
European Synchrotron Radiation Facility

ImageD11 priorities

- Process 2D Image data in real time at beamline ID11
- Enable decision making for experimental strategy
- Some hardware can be aligned easily in software (tilts)
- "Freedom of Speech" in data processing:
  - see what & how the program actually computes
- access to test new ideas (learning curve remains)

- Also runs in office (win32), when travelling (netbook)


-log(cost)
Ideas
Demonstration
Trials
Product
Time


-log(cost)
Ideas
Days
Demonstration
Weeks
Trials
Months
Product
Years
Time

What made it to Products?

Peaksearch

- Finds the connected blobs in images above simple threshold
- Can do all ID11 corrections
- Efficient/parallel

Transformation

- Converts blob center of mass position in image to reciprocal space
- Helps with ID11 calibrations
- See Practical Session

What is still only a "Trial" ?

- Indexing - many known unit cells, few unknown
  - old gui based code
  - newer index_unknown program
- Background estimation
- 3D Refinement of grain positions & unit cells

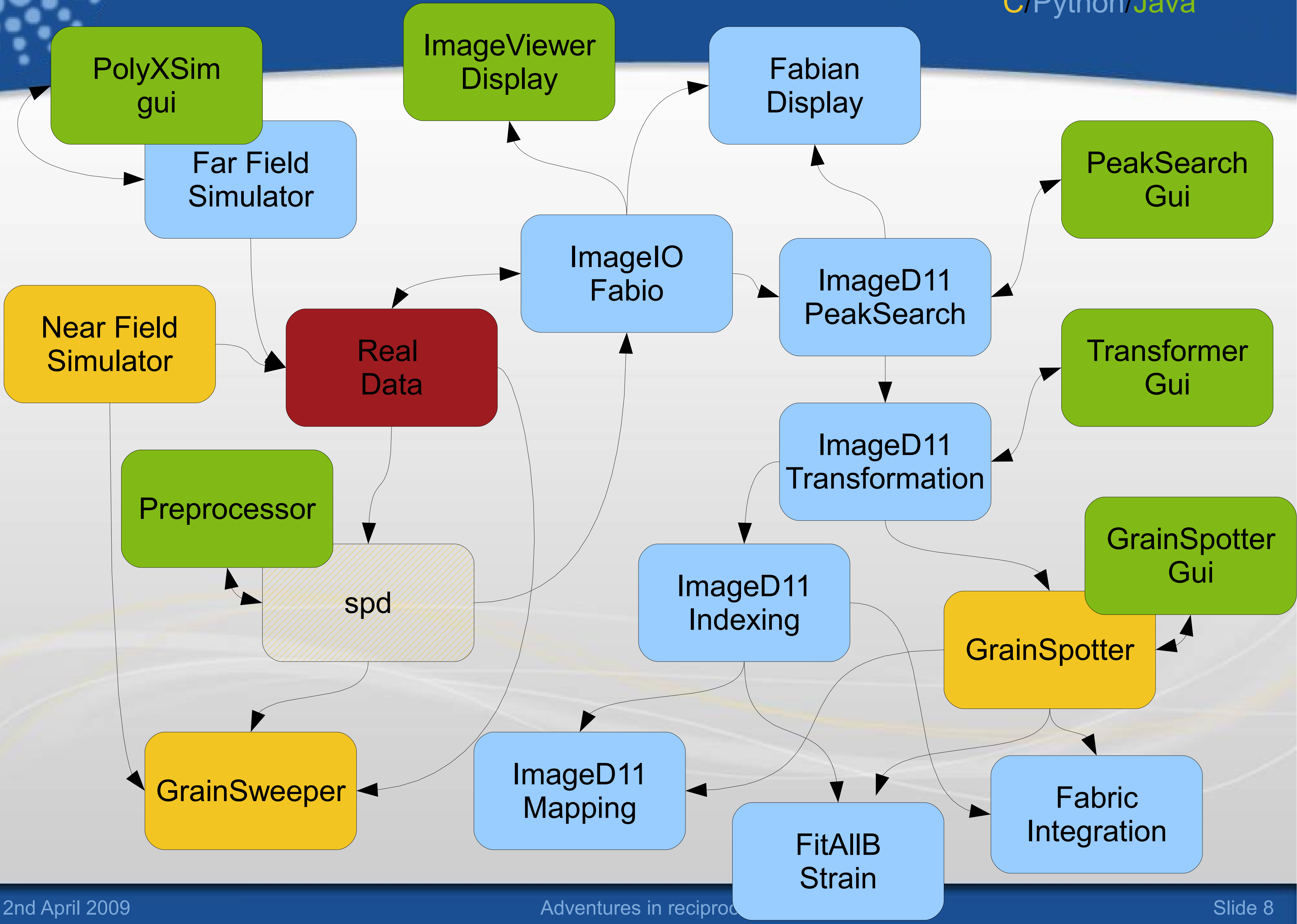
- Trial == poor documentation
- relatively untested

Which are the "demonstrations"?

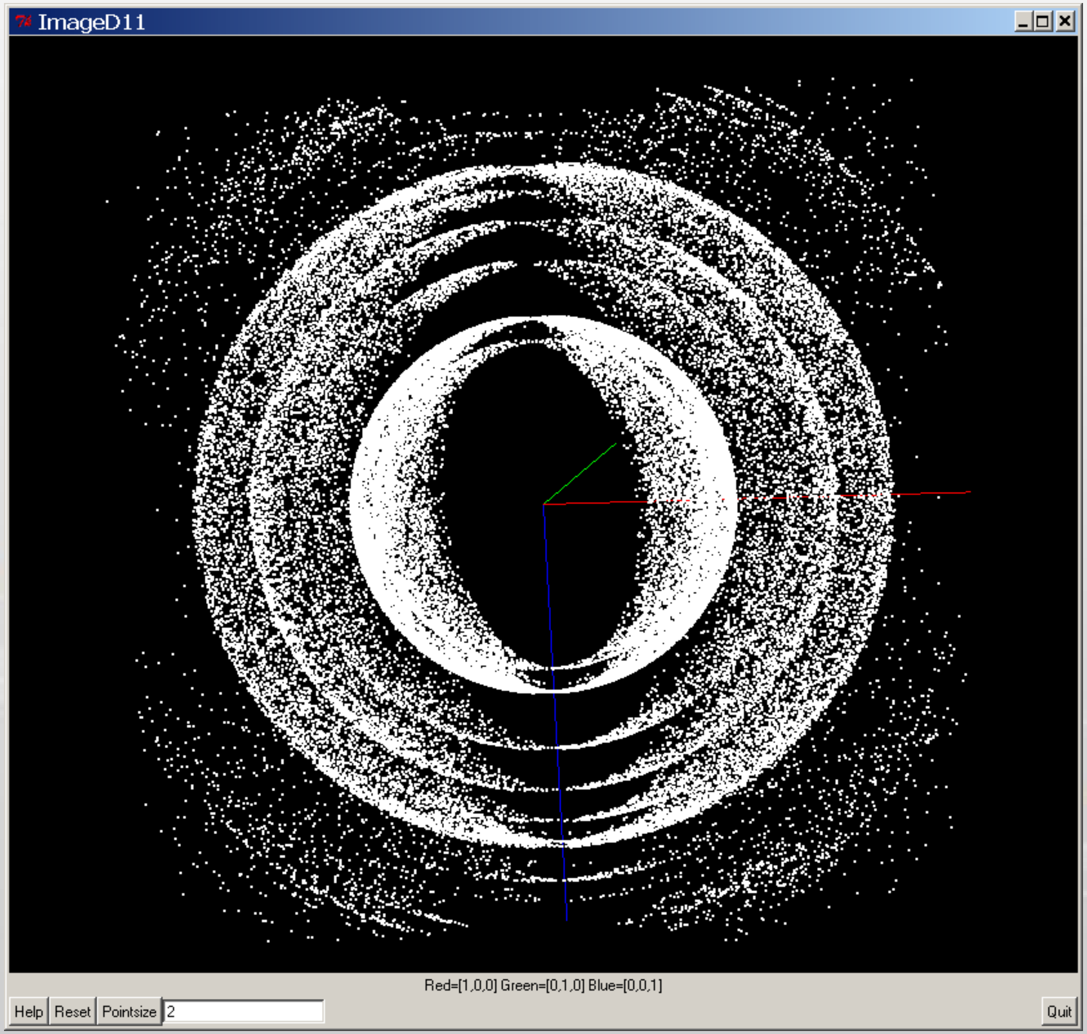
- Reciprocal space volume reconstruction
  - Maps series of images into 3D volume
- Grain mapping module
  - computes grain shapes
- Image deconvolution function

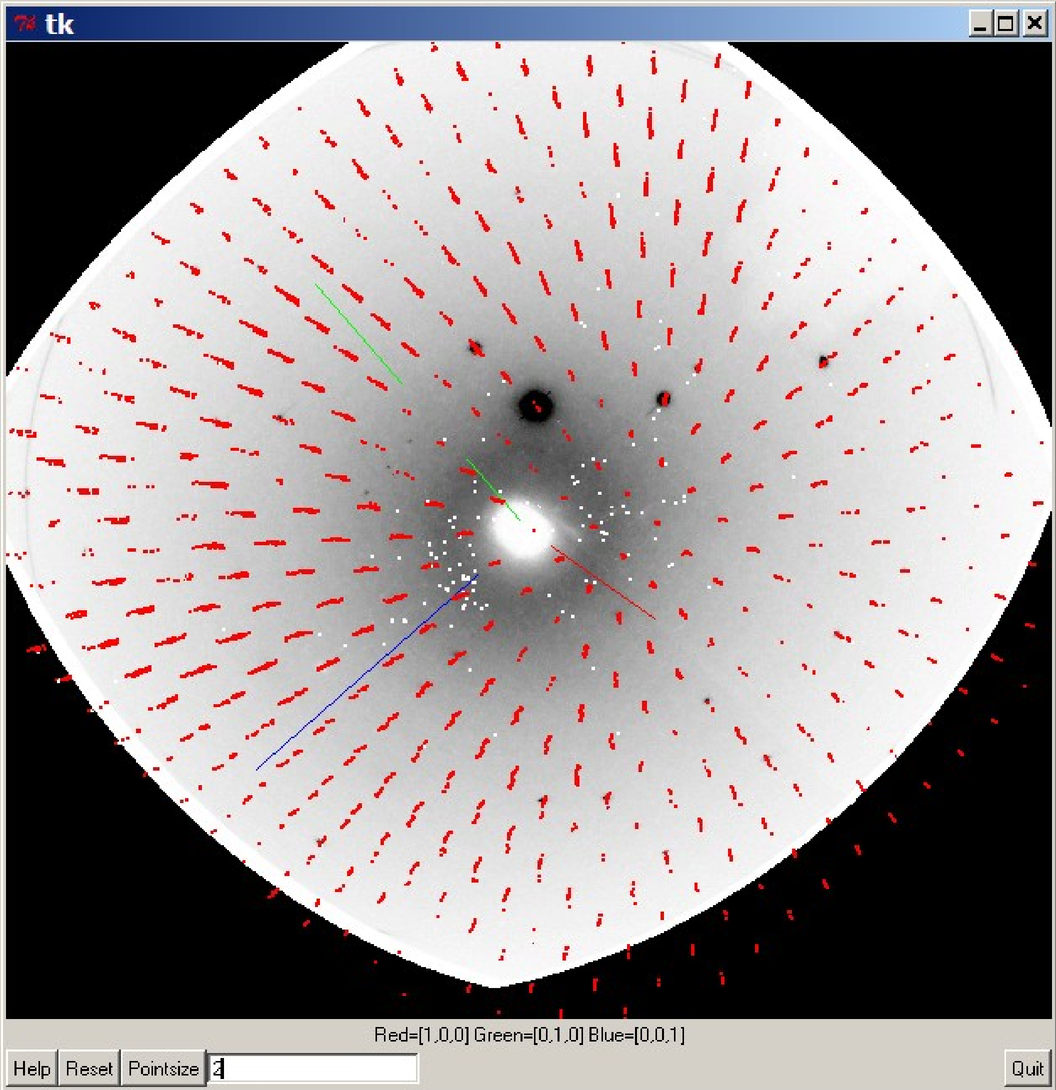
- Little documentation outside of full source code

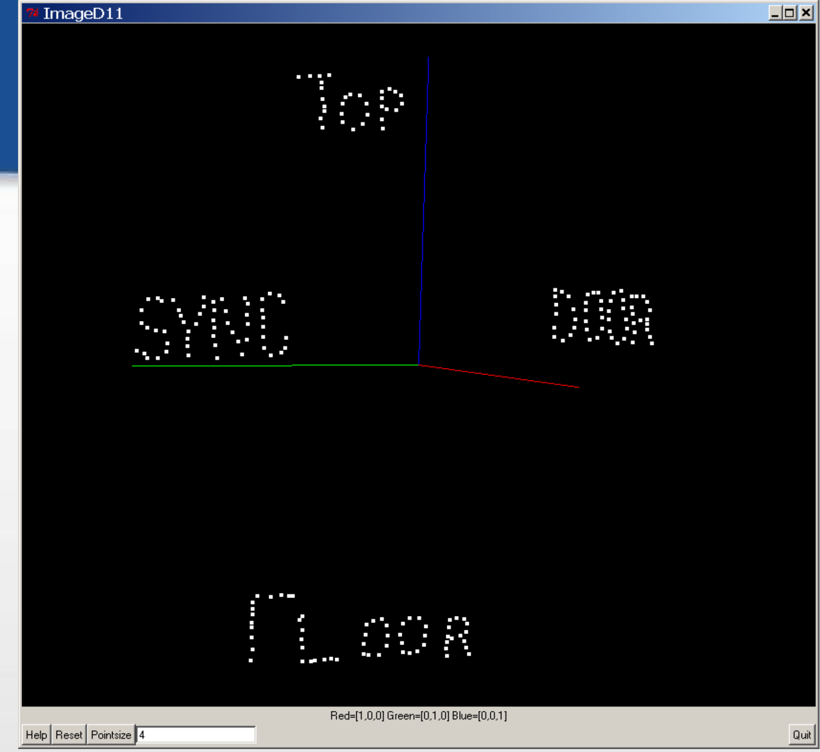
C/Python/Java

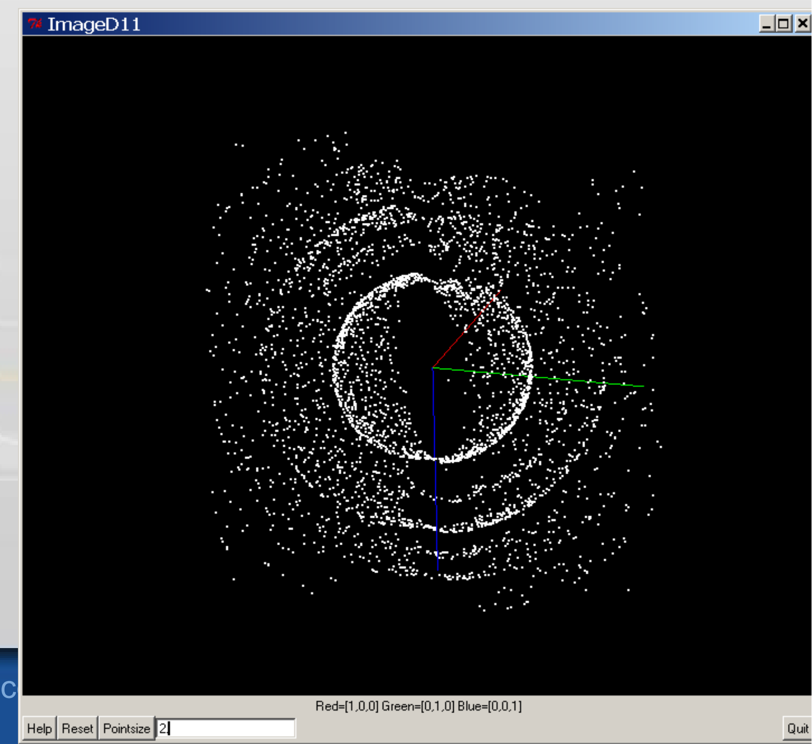


Powder ring gives 3D sphere





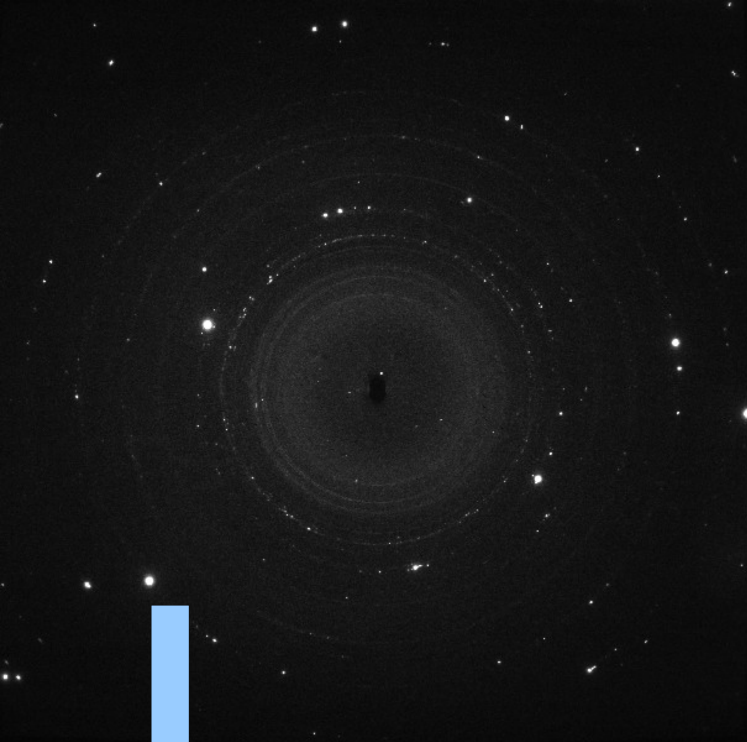




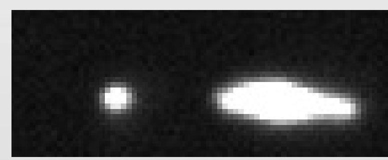
2D image is projected onto the surface of a sphere in 3D reciprocal space.

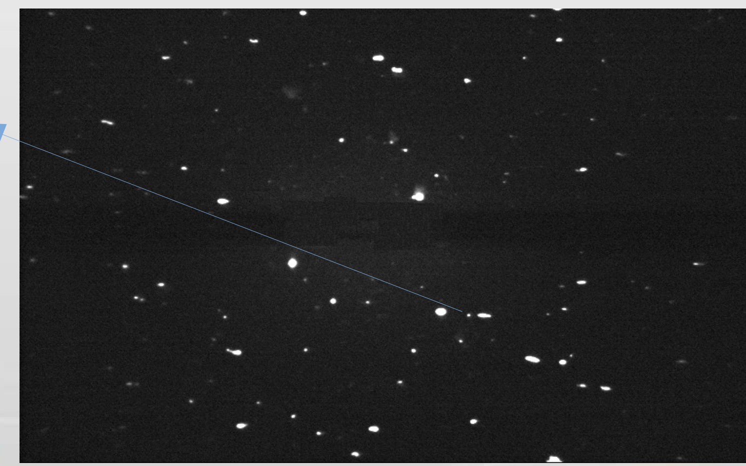
As the sample rotates, the image is rotated about the beam centre

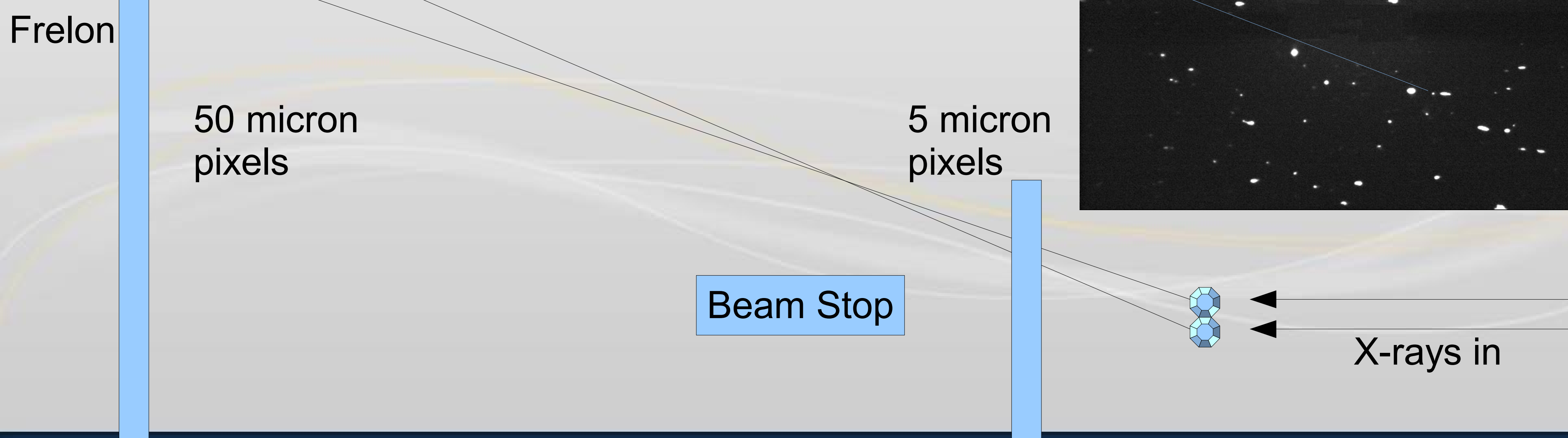
3D detector



- Close to sample: high resolution, semi transparent
- Position and angle of rays
- Projections of grain shapes

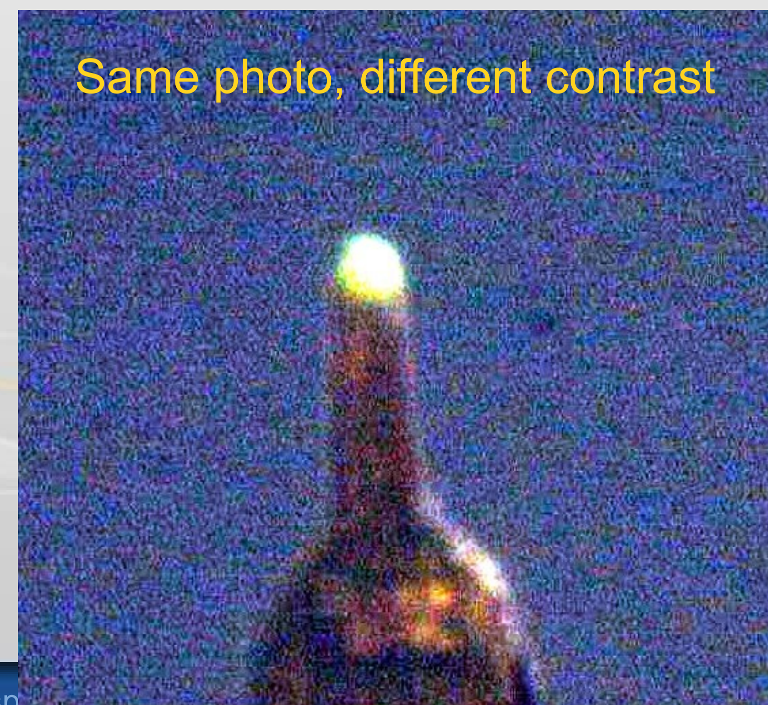




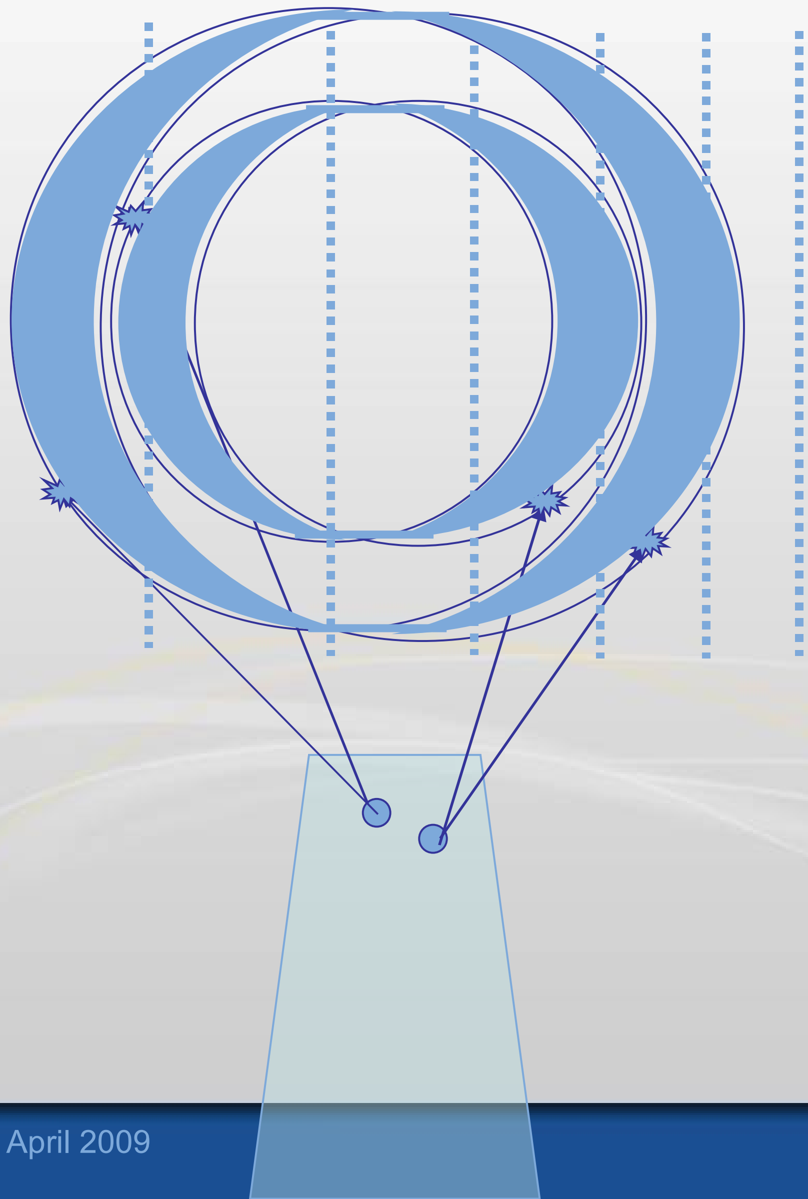


Example...

- The “3D” detector
- Calibration of first screen position
- Small crystal of “LumiLux”, florescent material for easy alignment in the 5 micron high beam



Global experiment parameters

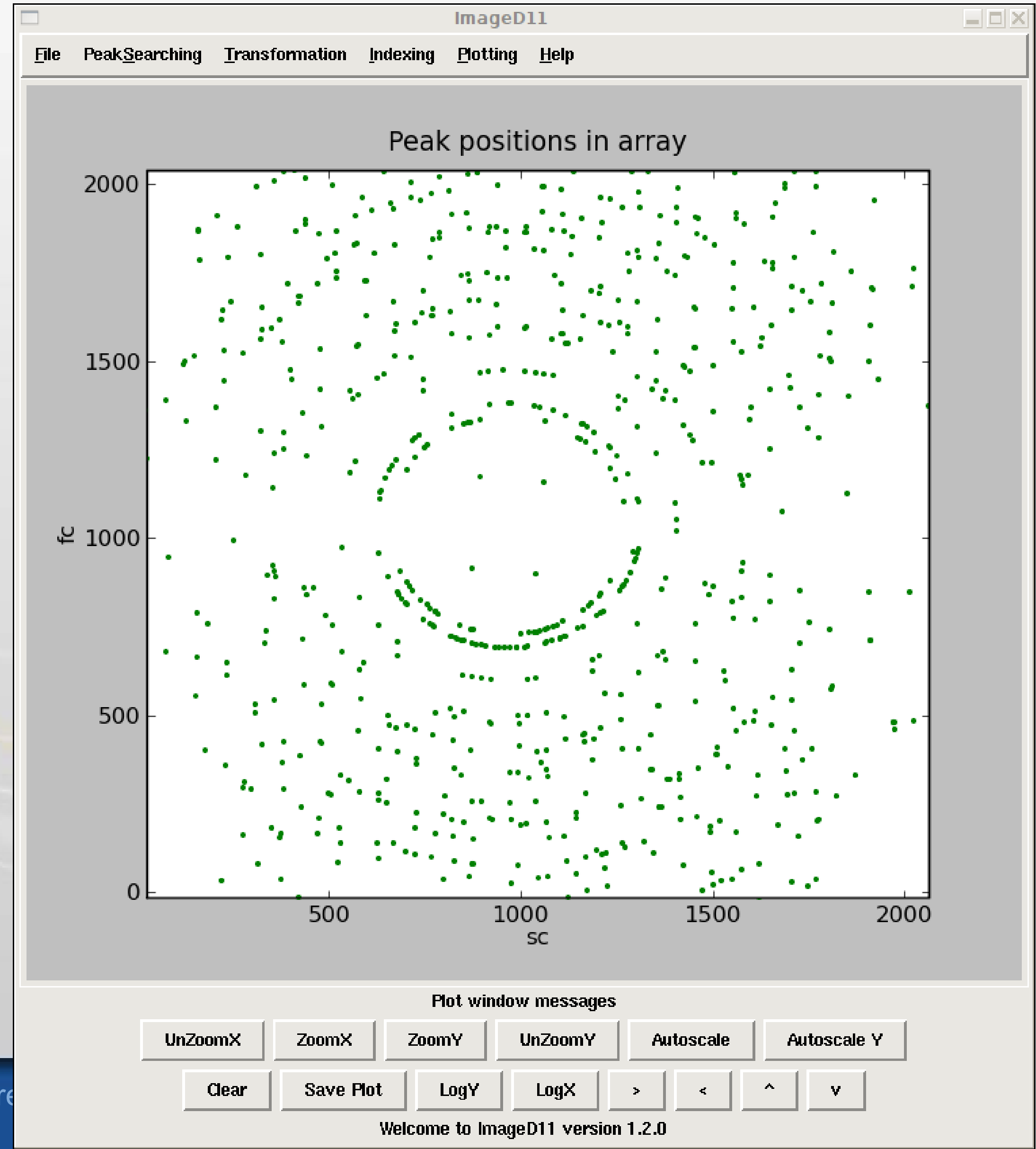


- grain mapping experiments
- "3DXRD"
- Need to find:
  - Position, tilts of detector
  - Rotation axis
  - Beam
- Index a single grain
- Refine **all parameters** together
- **Including grain position!**

Disjoint set
youtube CS61A

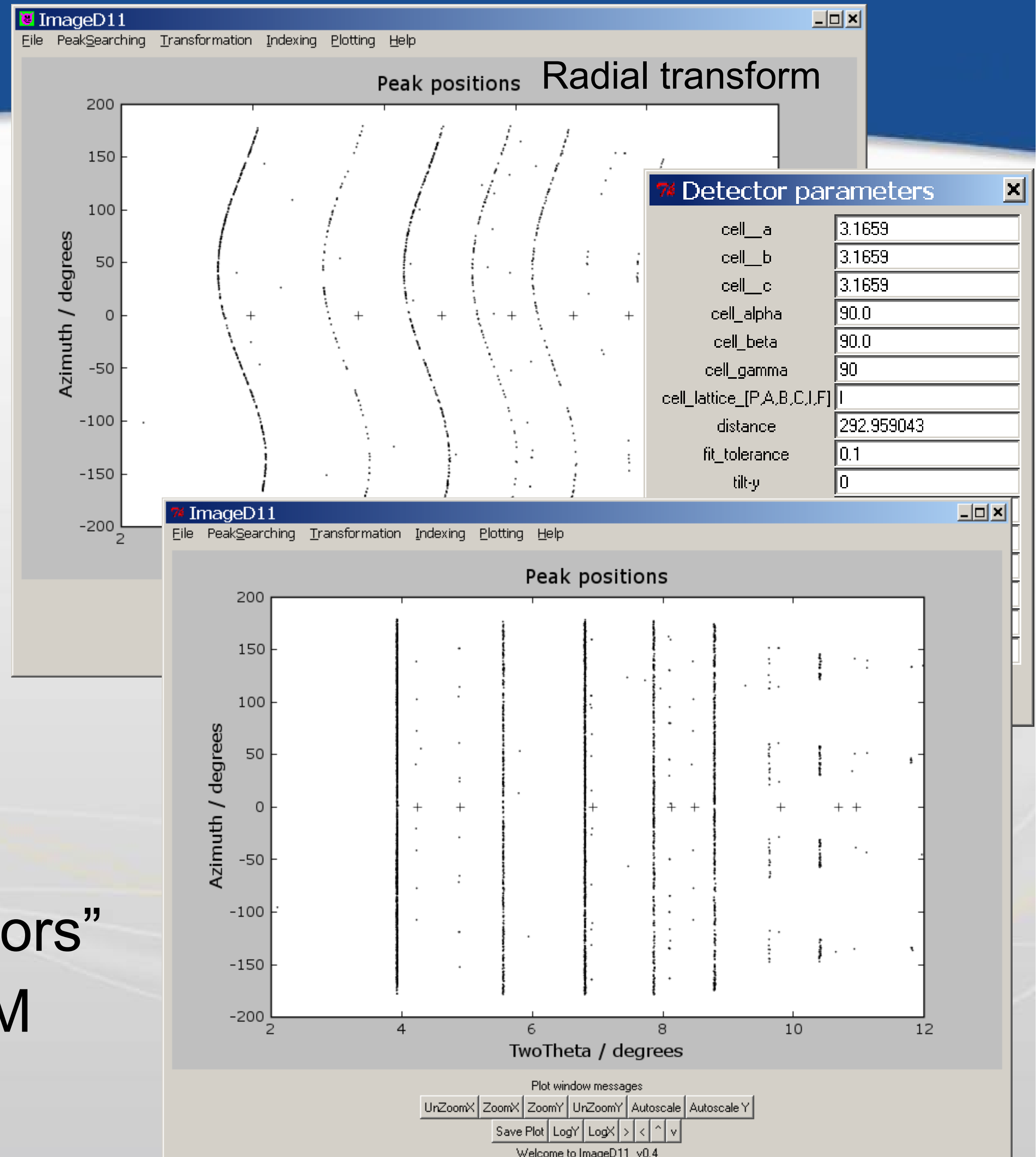
Peaksearch from frelon gives:

- Beam centre by inspection
- Sample detector distance approx. from ruler
- Unit cell was (is) unknown (almost)
- Far back detector



Detector calibration

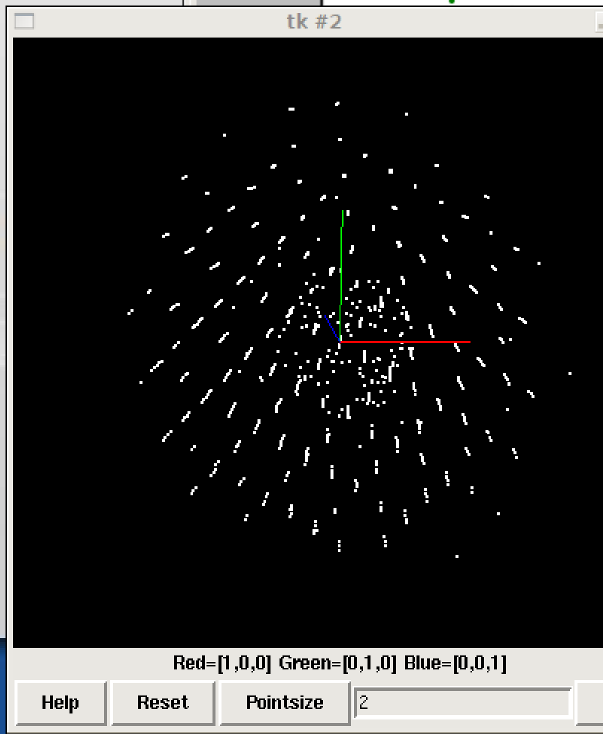
- Distance
- Centre
- Wavelength (cell)
- Tilts
- Simplex algorithm
  - Convergent
  - No derivatives!!!
- Generates "g-vectors"
- SEE TRANSFORM

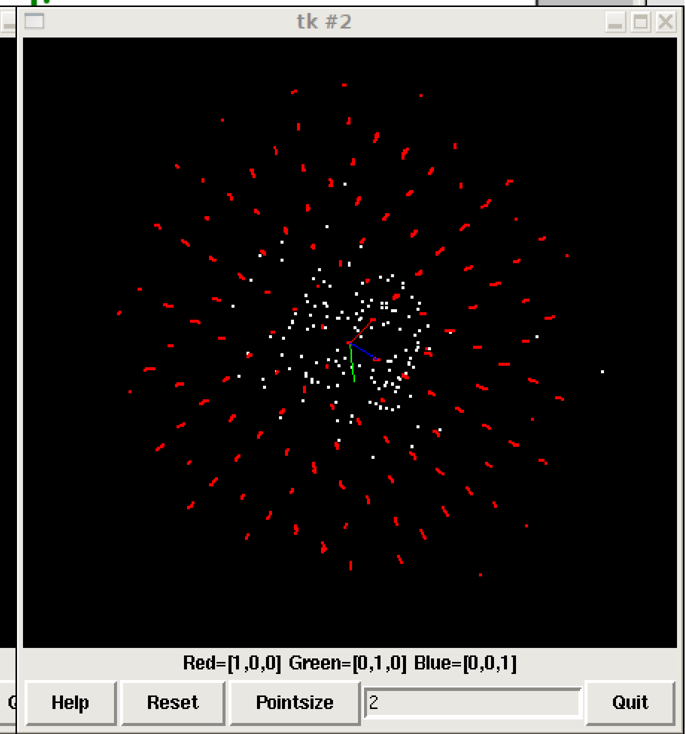


Make g-vectors

- Vectors form a single lattice in 3D
- Indexed via ImageD11:

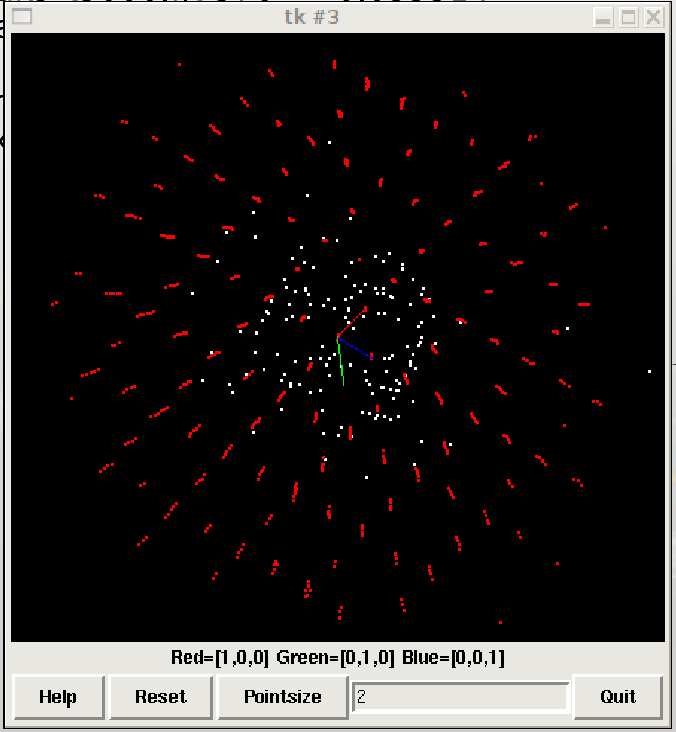
index_unknown.py
  -g test.gve
  -m 50
  -k 1
  -f 0.6
  -v 0.01



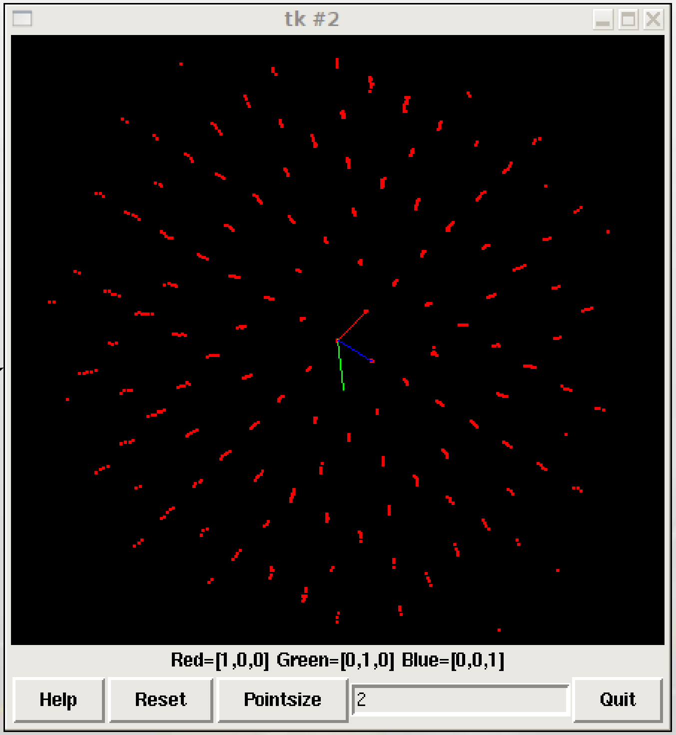


Select peaks for this grain only

```
$ filtergrain.py -p grains.par  -u grains.ubi -f peaks_t5000.flt -F grains.flt
[abridged output]
                   grainname scanname       npeak    <drlv>
INFO     : tol 0.010000        0 peaks_t5000.flt 17       0.008773
INFO     : tol 0.025000        0 peaks_t5000.flt 365      0.017960
INFO     : tol 0.050000        0 peaks_t5000.flt 562      0.023666
INFO     : tol 0.100000        0 peaks_t5000.flt 565      0.024167
INFO     : tol 0.150000        0 peaks_t5000.flt 565      0.024167
INFO     : tol 0.250000        0 peaks_t5000.flt 576      0.035314
INFO     : tol 0.500000        0 pea
Enter tolerance 0.1
INFO     : Total peaks before filter
INFO     : Peaks which were index
```

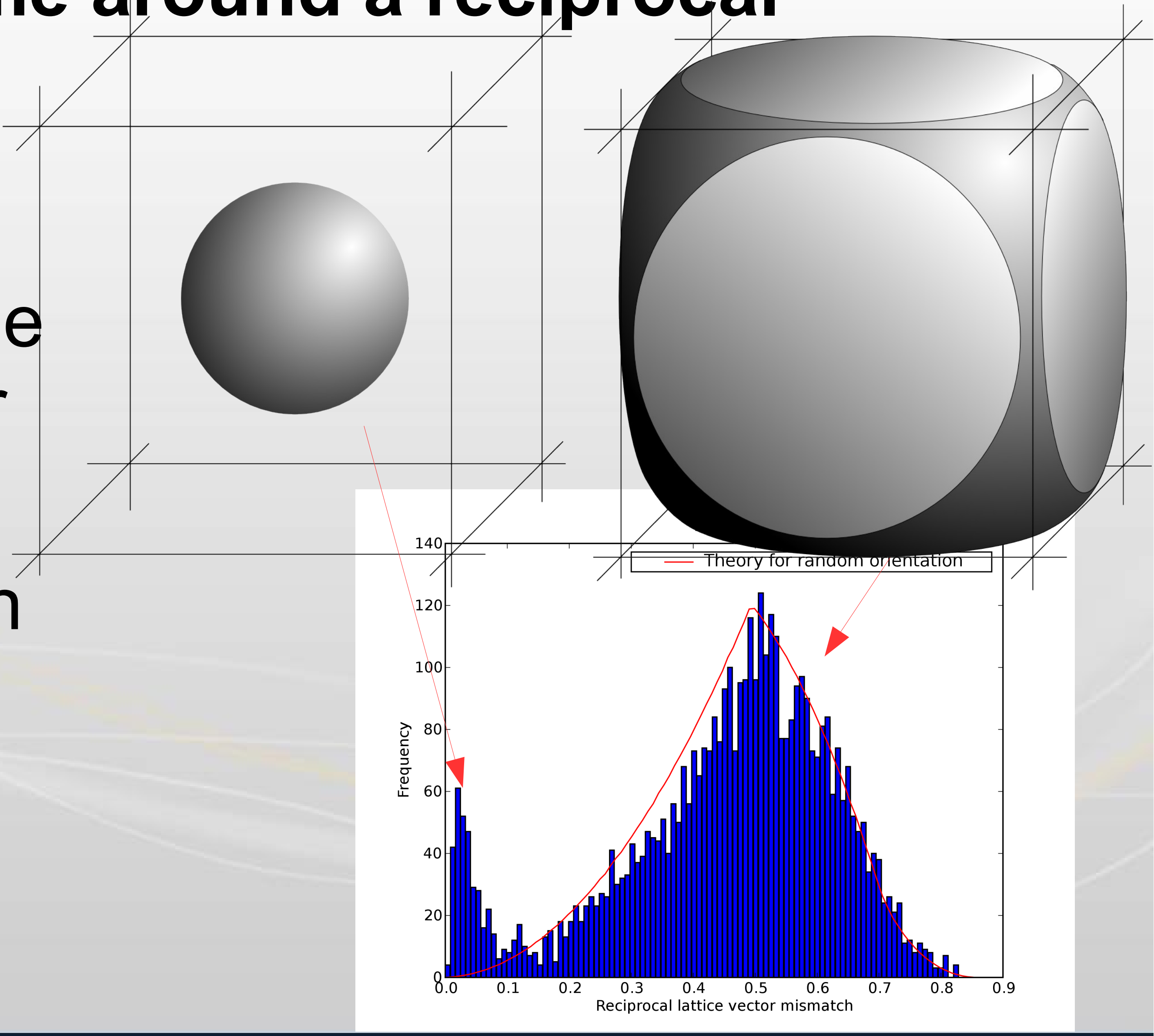


Gives a clean dataset for refinement



Imagine the volume around a reciprocal lattice point

- Plot number of peaks versus the error in g-vector position
- Peak near origin is good
- Aspirin data are shown here ->



Fit your parameters

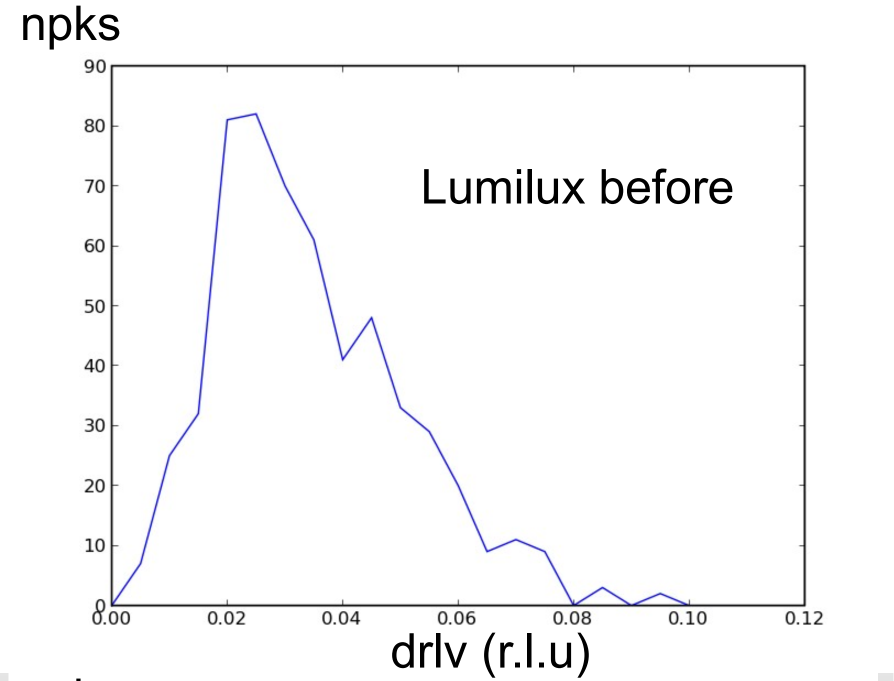
$ fitgrain.py -p grains.par -u grains.ubi -f grains.flt -P grains.par -U grains.ubi

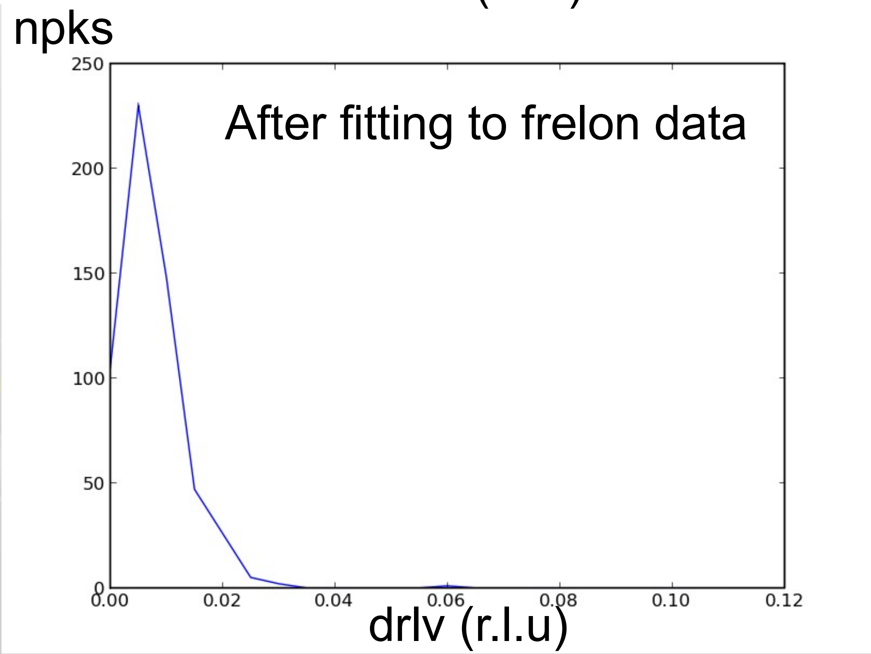
```
 grainname   scanname npeak   <drlv>
        0 grains.flt 565      0.024167
INFO     : Varying ['y_center', 'z_center', 'tilt_y', 'tilt_x', 'tilt_z', 'wedge', 't_x', 't_y', 'distance']
.....
y_center 1039.91568304
z_center 963.333198072
tilt_y -0.00108562852963
tilt_x -0.00115346144983
tilt_z -0.000910117913193
wedge 0.00221573050031
t_x 50.0051798301
t_y 11.8013210782
distance 195614.73322

 grainname   scanname npeak   <drlv>
        0 grains.flt 565      0.012281
```

$ plotgrainhist.py t5000.flt new.par fitted.ubi 0.1 20

$ ubi2cellpars fitted.ubi

3.92981 2.26833 3.92790 89.97861 89.99631 89.97653





Mapping

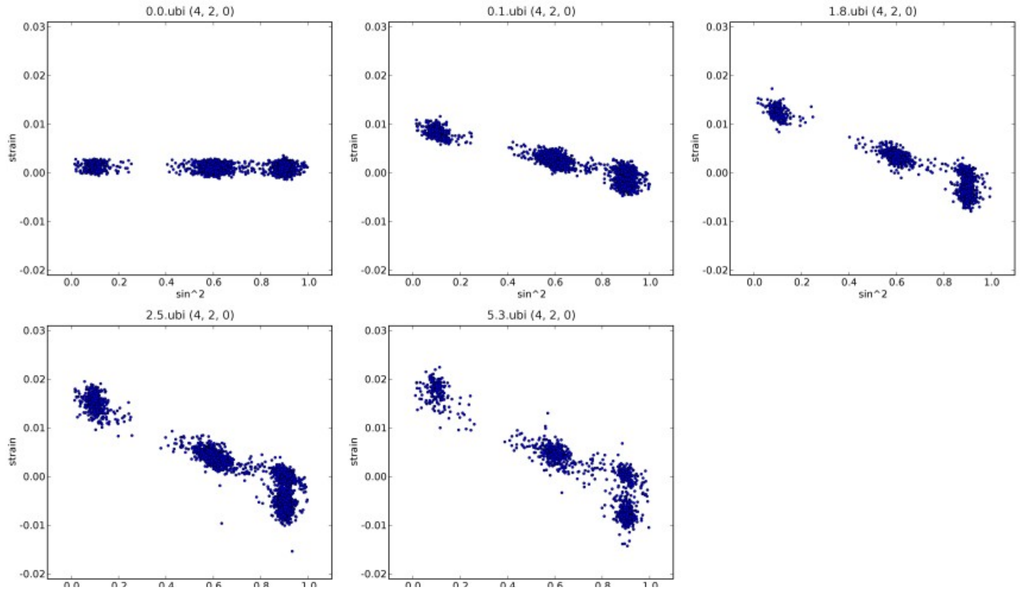
- Makemap.py script will take a list of grains and a data file and refine all grains for position, unit cell and orientation.
- Gives a “ubi” file with centre of mass as well
- See Gavin Vaughan (or fable svn) for
  - compare_ub program to match grains across scans
  - Matlab scripts to make voronoi diagrams
- Also grainspotter (Soren) and FitAllB (Jette)

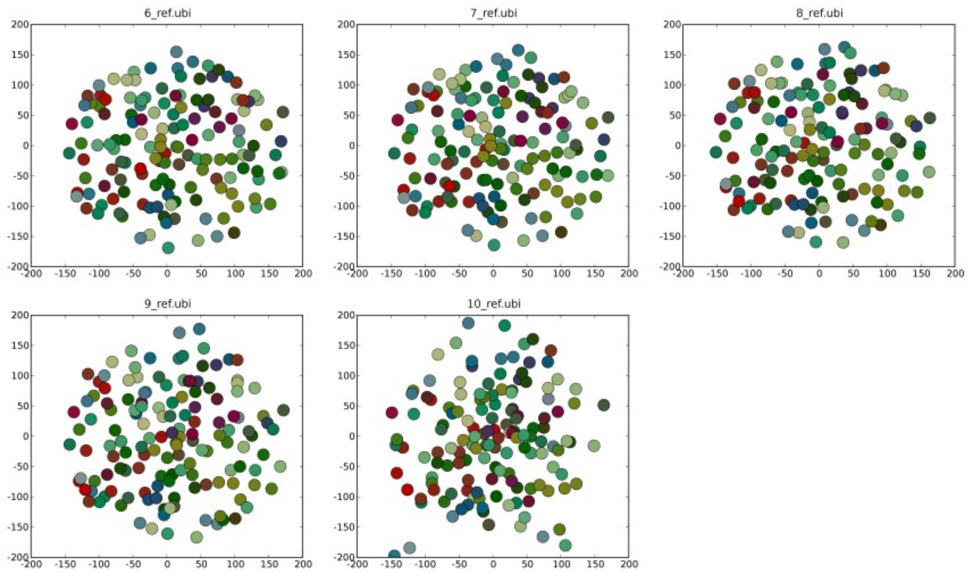
Y. Hirose *et al*. Toyota.

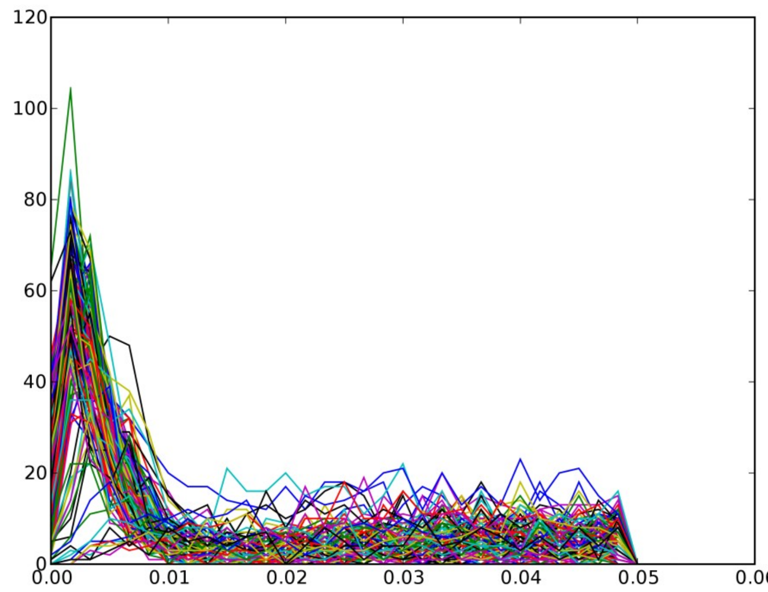
Centre of mass grain maps
sin2 psi plots from refined UBI
(circular wire)

Grain matching, validation
Automated scripts, runs on cluster

Analysis to be finished within few
days of experiment!

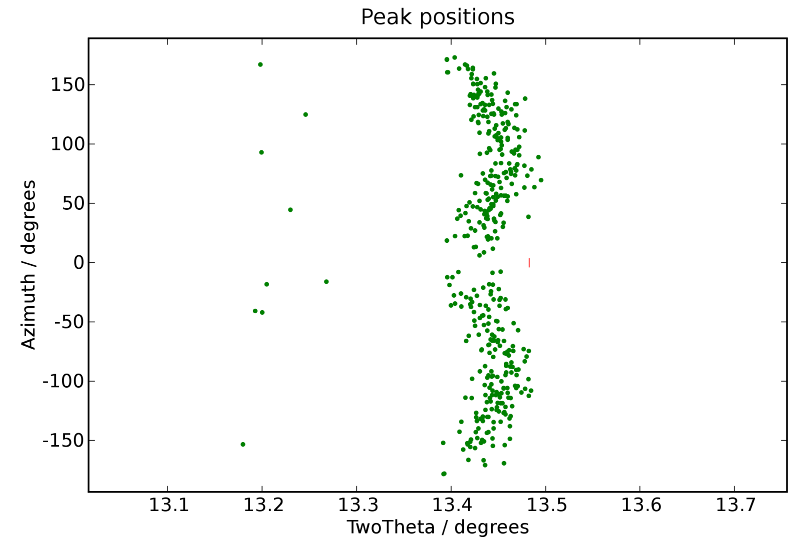


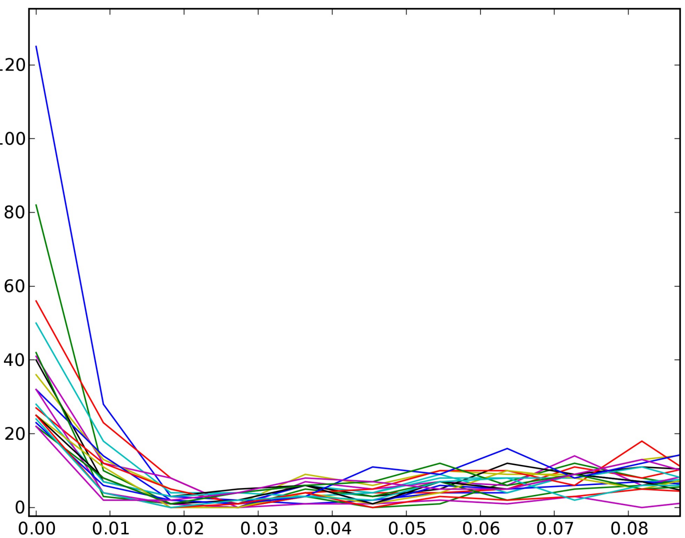


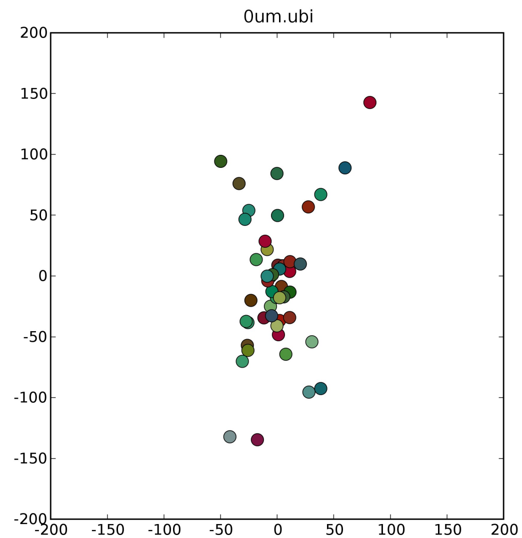


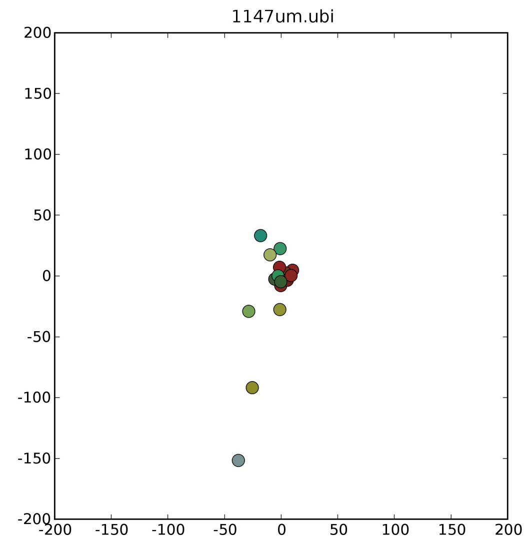
Niels + Enrique MA354

Analysis of the large (uninteresting) grains in TRIP steel







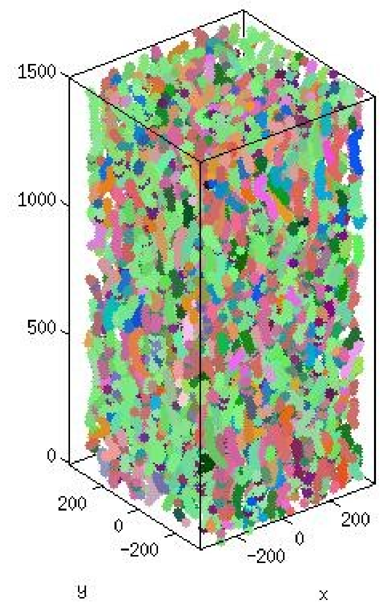


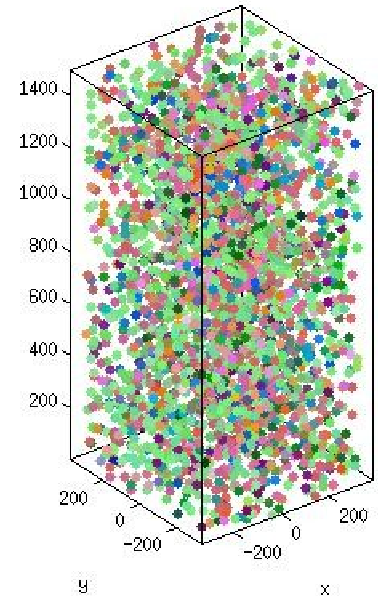
#translation: -6.176521 -24.734189 -8.963365
#name 0:../t10000.flt
#npks 152
#Rod 0.186922 -0.042649 -0.071614
#UBI:
2.502700 -1.118546 0.859265
1.288979 2.527450 -0.462600
-0.574214 0.788140 2.704304

#translation: -1.674185 -2.525916 -3.163264
#name 0:1147.flt
#npks 157
#Rod 0.186865 -0.042132 -0.072229
#UBI:
2.503833 -1.118888 0.857322
1.281995 2.534707 -0.468224
-0.573123 0.784893 2.708731

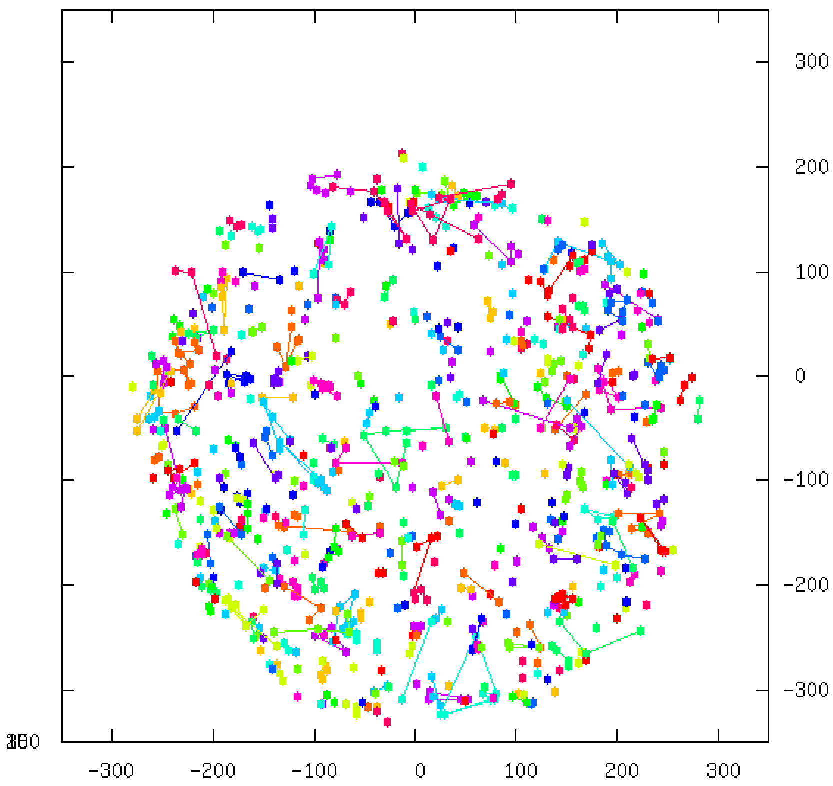
BCC steel

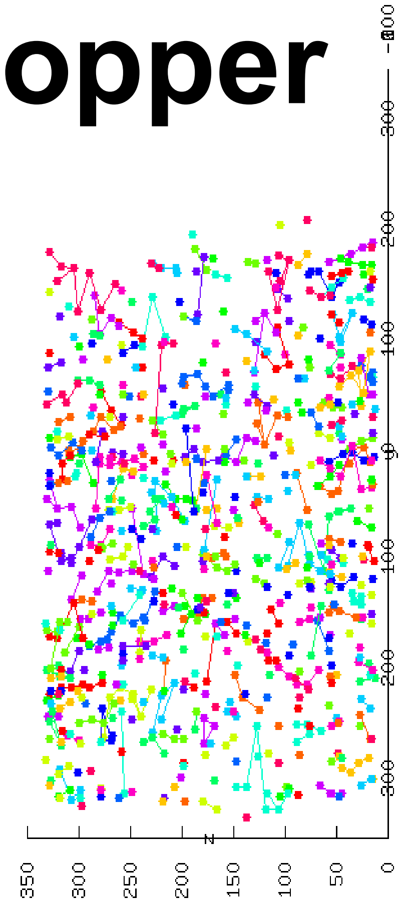
- Grain maps from ImageD11
- Top: 2D grain maps stacked up in 3D – same grain found in many layers
- Below: 3D centre of mass
- Color represents orientation
- Figures from matlab script
- Thanks to user group team: Grethe Winther (who made figures), Larry Margulies, Masakazu Kobayashi, Xiaoxu Huang, Henning Poulsen.

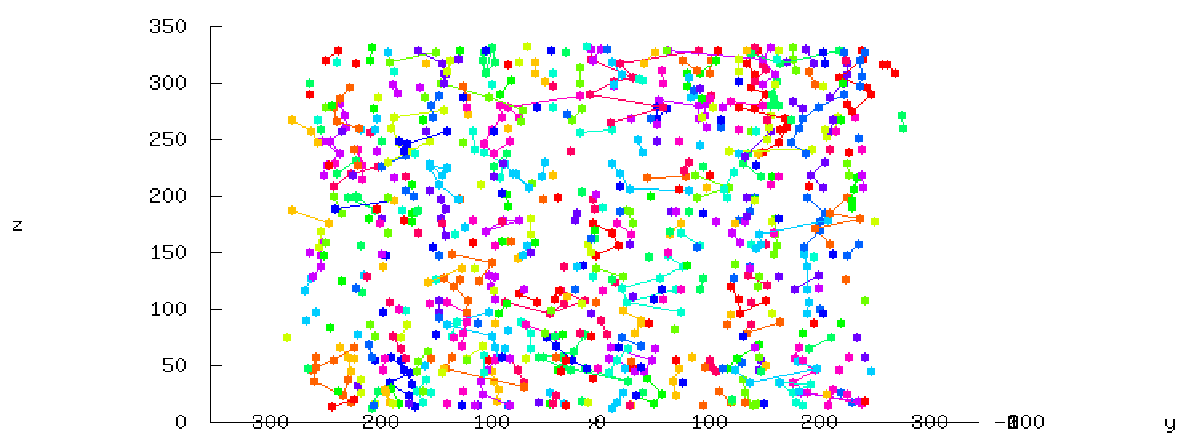




Copper



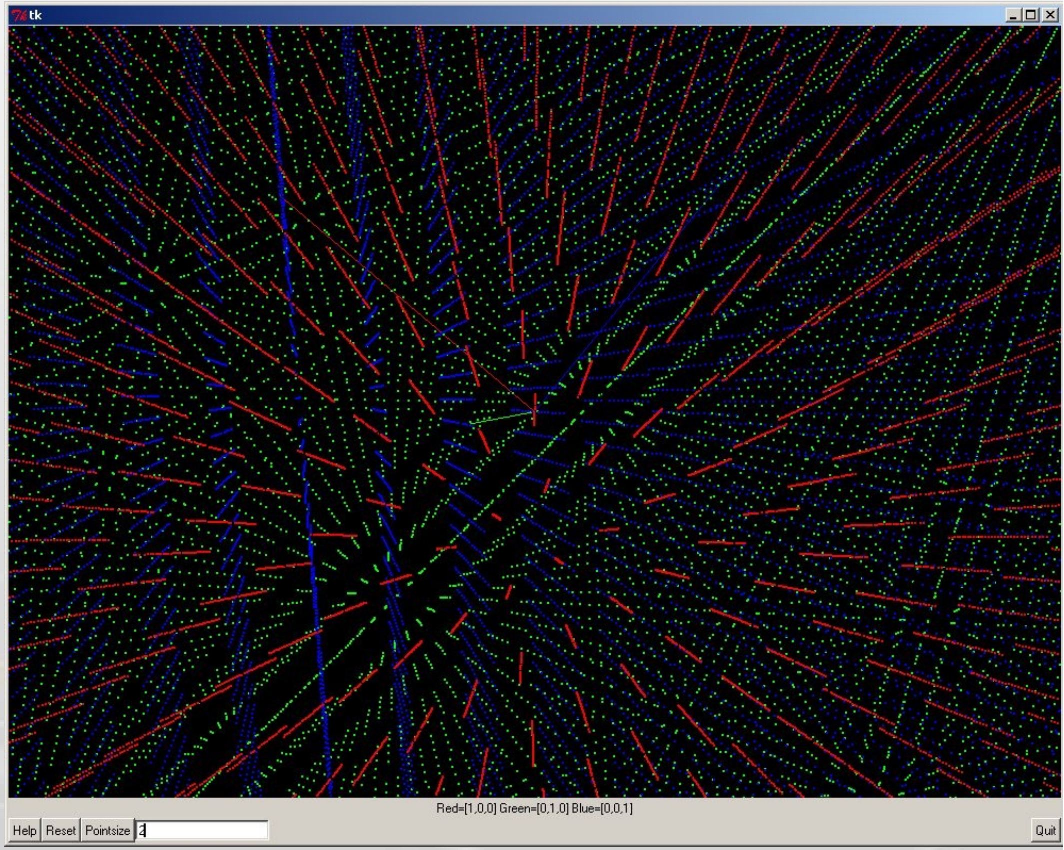




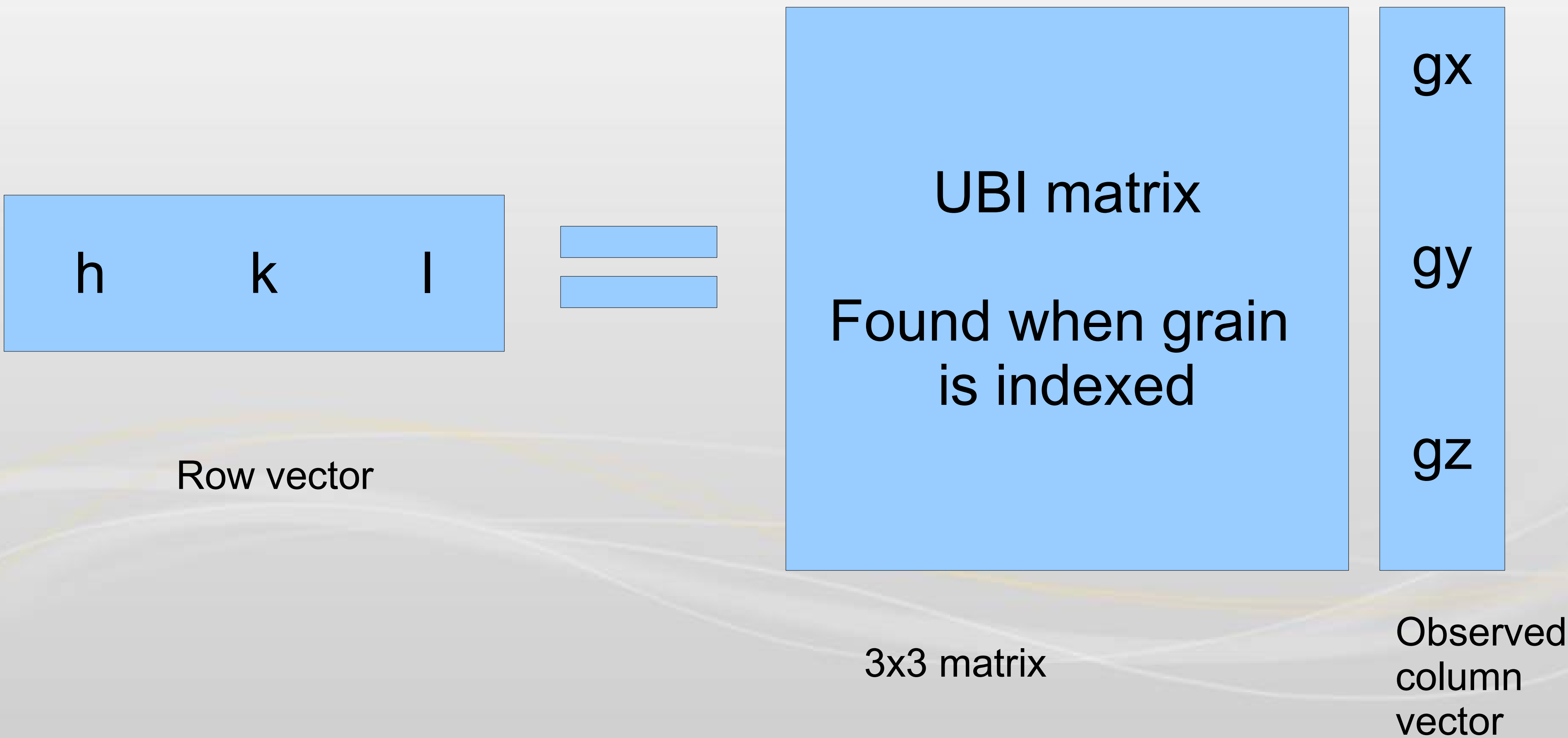
Leg of an electrical resistor
Recrystallised by heating
Grain maps for centre of mass positions from ImageD11
Grain shape mapping in progress (Carsten Gundlach).
Also Voronoi tilings (Gavin Vaughan)

Indexing with the FFT

- Intention is to untangle peaks by using real space vectors
- Code also includes reciprocal space methods but needs difference vectors to be added
- Further work:
  - “Auto guess” cryptic parameters



FFT indexing



FFT indexing

1	0	0
0	1	0
0	0	1

=

UBI matrix

Found when grain is indexed $(UB)^{-1}$ or inverse(UB)

gx	gx	gx
gy	gy	gy
gz	gz	gz

Row vectors

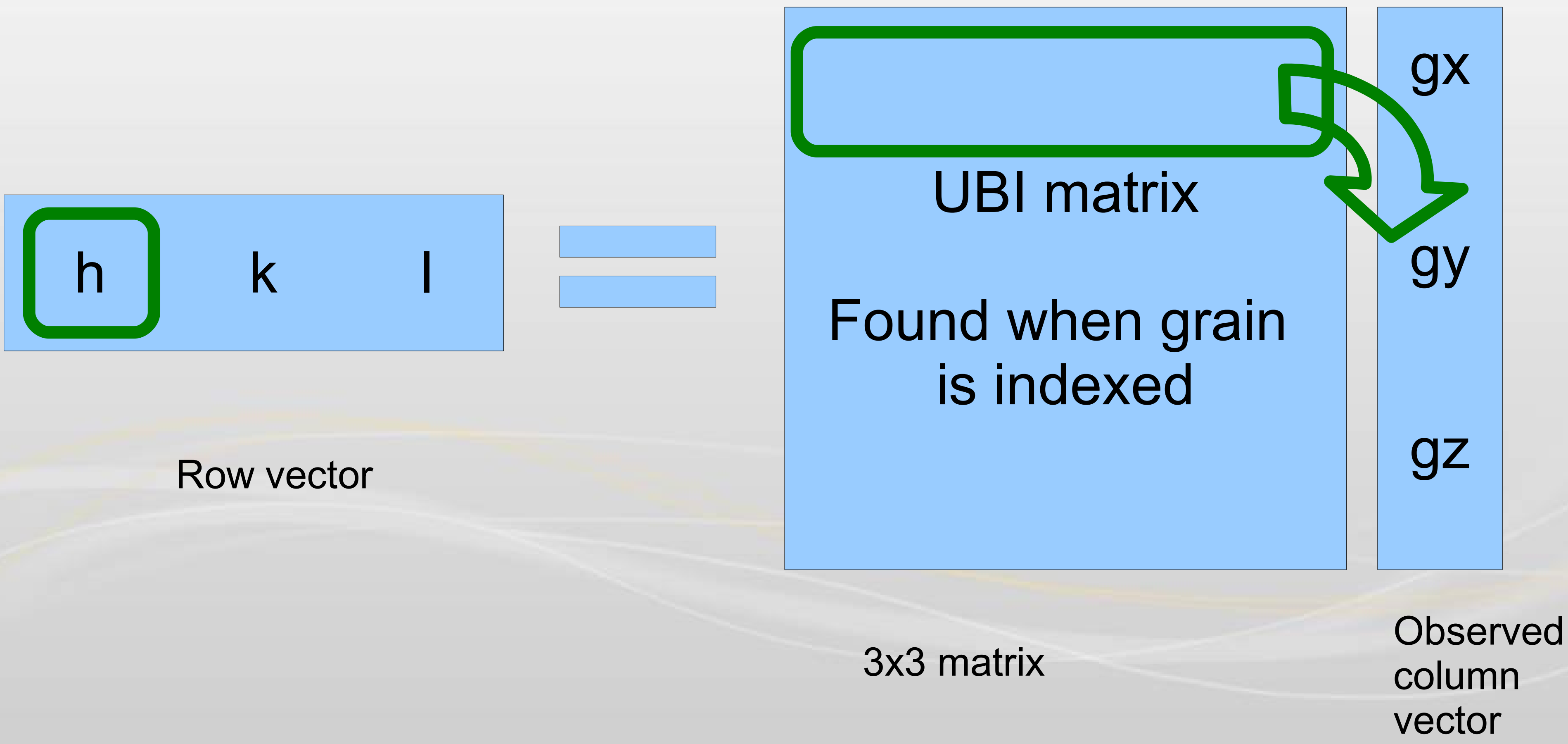
3x3 matrix

3 primitive lattice vectors in reciprocal space

FFT indexing

- UB matrix **<u>columns</u>** are just the 3 reciprocal lattice vectors
  - a*, b*, c*
- By symmetry (please don't ask me)...
- UBI matrix **<u>rows</u>** are real space lattice vectors
- Fourier transform **all** peaks to give real space lattices superimposed
  - Pick peaks to get rows of ubi
  - Rows can be tested independently

FFT indexing



FFT indexing

- Need one real space lattice vector
- All peaks from same grain have integer h
- Group peaks into grains if they have integer h
- fft_index code

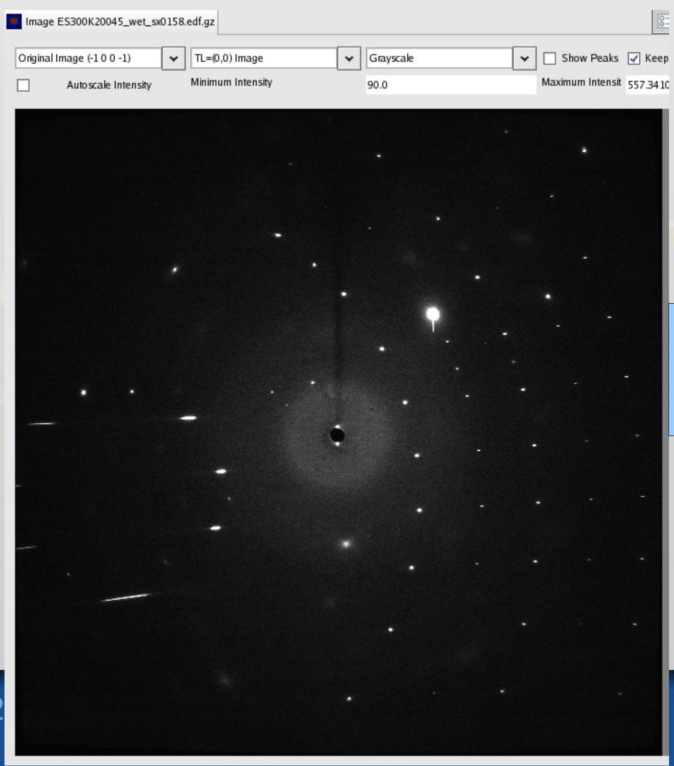
- Concluded:
- Seems easier to just pick 3 vectors from list to make full lattice
- Aimed at case of large, unknown, unit cells (no powder rings)

Reciprocal Space Mapping

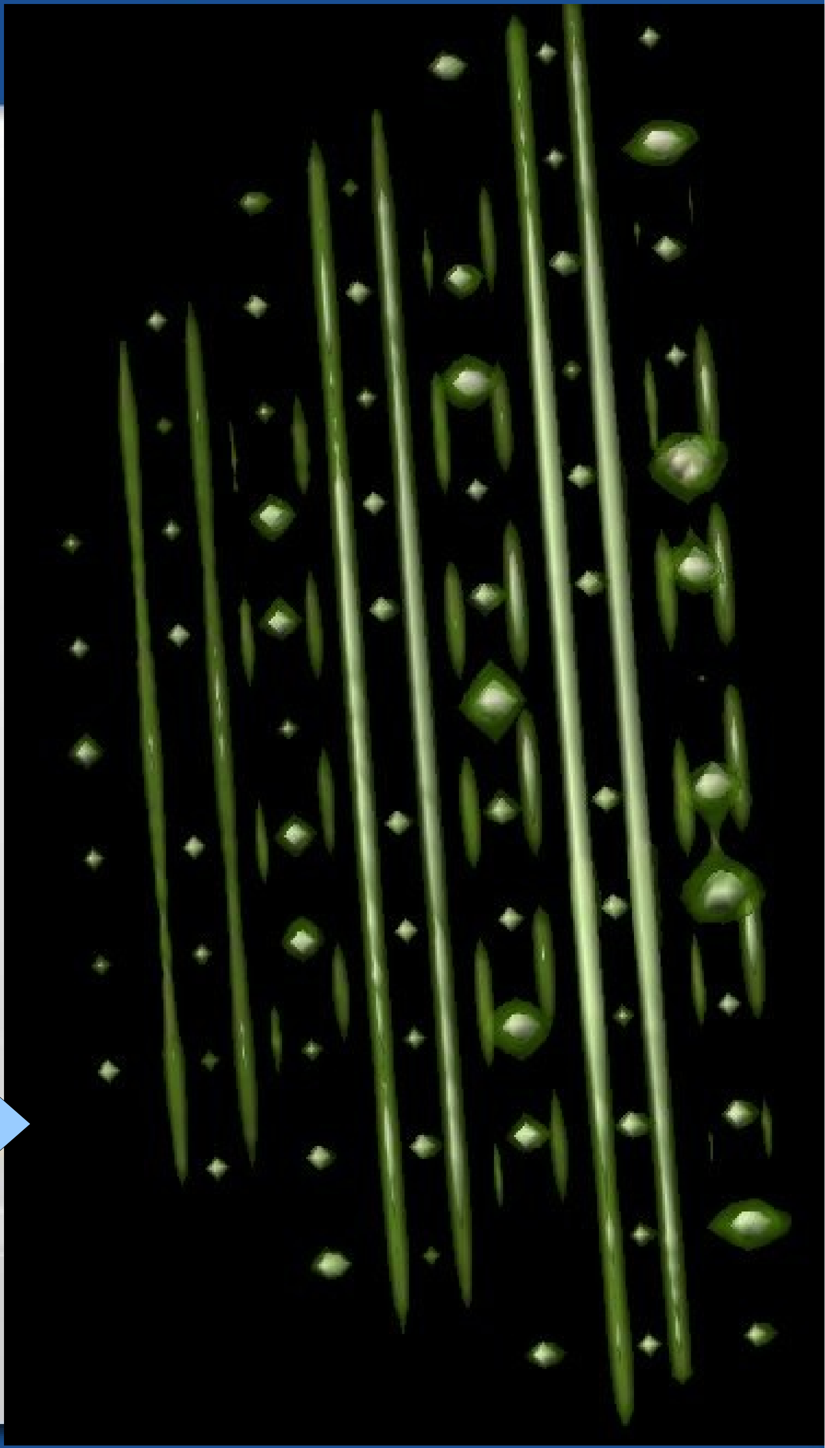
- Idea:
  - Pixel in image is found in memory at array[i,j]
  - Laboratory scattering vectors kx, ky, kz if the detector is fixed
  - Sample vectors given by **g** = **R.k**
  - 3D array indices from **hkl = n . (UBI) . g**
- Compute 3D array indices in a single matrix-vector product for cached vectors
- Increment volume data at these pixel co-ordinates

Demonstration...

- Sample is from Sai Vaidya & Sanjay Rastogi
- Rods of diffuse scattering
- Rendering by USCF Chimera



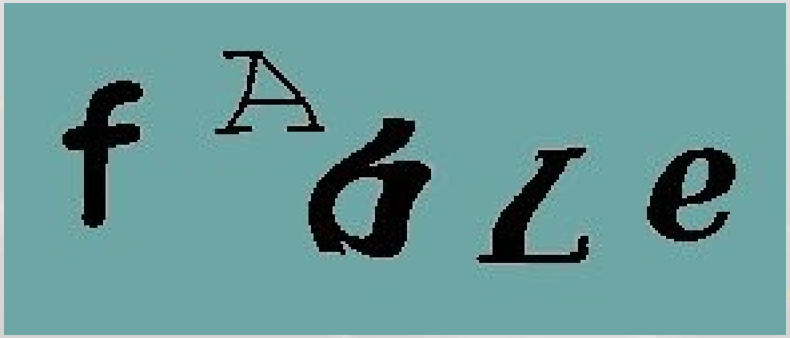
rsv_mapper.py



Finally...

- Human computing
- Converting algorithms to CAPTCHA'S

Completely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part



Thanks!

- ESRF

Gavin Vaughan
Gaelle Suchet
Carsten Gundlach
Alex Bytchkov
Caroline Curfs
Andy Goetz
Rainer Wilke
Irene Margiolaki

- Goettingen

Simone Techert
Jav Davaasambuu

- APS

Joel Bernier
Ken Evans

- Riso

Henning Poulsen
Soren Schmidt
Henning Sorensen
Erik Knudsen
Soren Faester
Xiaxu Huang
Larry Margulies
Jette Oddershede
Stine West
Grethe Winther

- Oxford

Elspeth Garman
Karthik Paithankar

- Delft

Neils Van Dijk
Enrique Melero

- Copenhagen (Novo Nordisk & Lundbeck)

Anette Frost Jensen
Anders Svensson

- Prague

Michal Dusak
Vaclav Petricek

- Toyota

Yoshiharu Hirose

- Jepp

Mike Johnson

Eclipse foundation
Python & Numpy communities

The End

Thankyou for listening