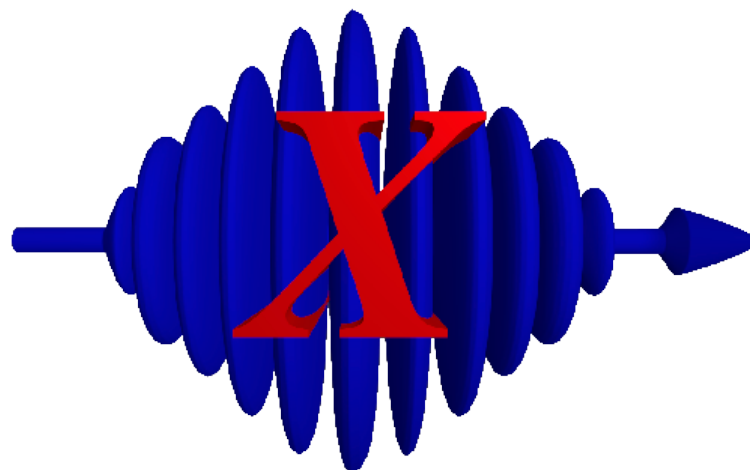


McXtrace



Powered by McStas technology

"McXtrace: a Monte Carlo software package for simulating X-ray optics,
beamlines and experiments",

Journal of Applied Crystallography, vol. 46, part 3, June 2013

Peter Willendrup <pkwi@fysik.dtu.dk>
Emmanuel Farhi <emmanuel.farhi@synchrotron-soleil.fr>
(and Erik Bergbäck Knudsen)

- Built on proven base of McStas for neutron ray tracing
K. Lefmann and K. Nielsen, Neutron News 10, 20, (1999).
- This is Monte-Carlo: we use random numbers to describe particle distributions
- Releases 1.7 and 3.0 (GPU support)
- Portable code (Unix/Linux/Mac/Windows, 32 and 64 bit support)
- Has run on all from iPhone to FERMI (world no. 7)



Project website at
<http://www.mcxtrace.org>

Project mailing list at
mcxtrace-users@mcxtrace.org

Source code at
<https://github.com/McStasMcXtrace/McCode>

- NVIDIA GPU support from v. 3.0 

- GPL-license
- DSL / Compiler Technology.

Using Lex & Yacc

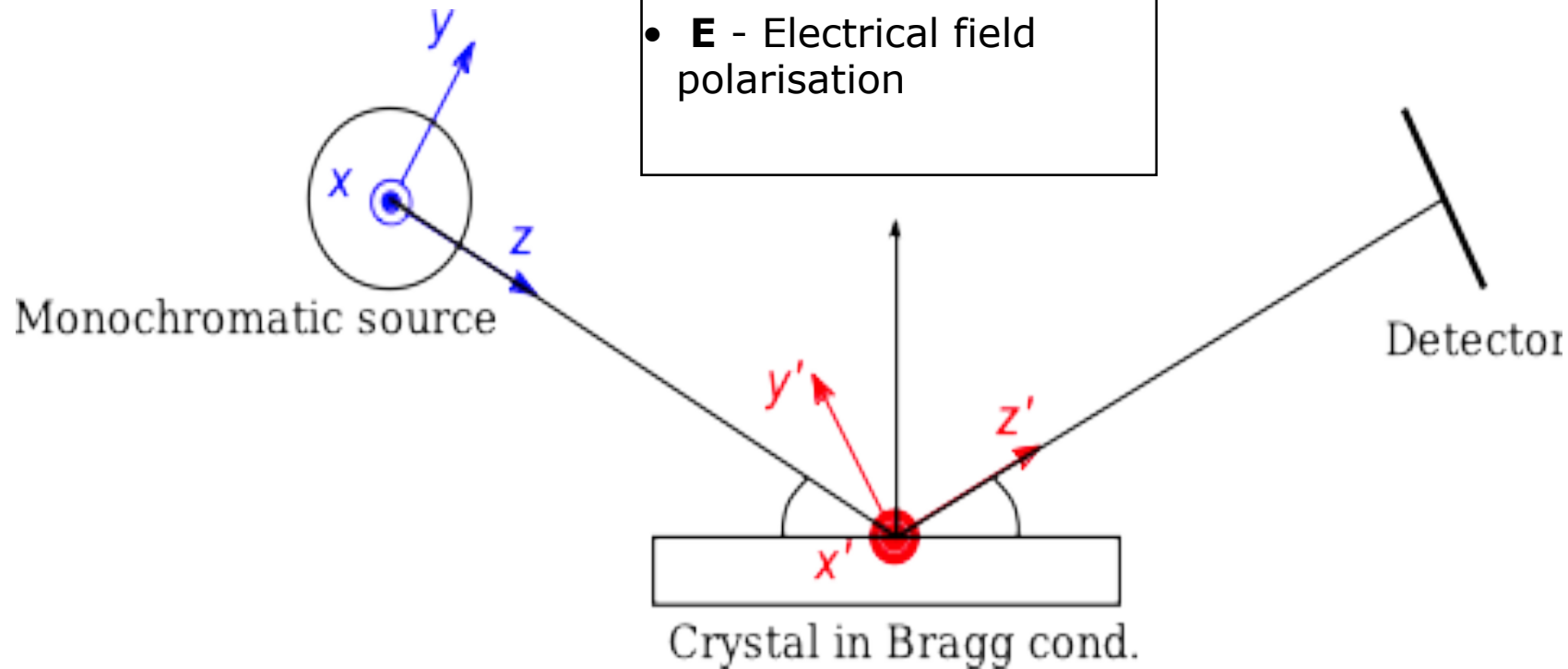
- Modular Open Structure.

Components/devices written in structured ISO-c automatically fits in the system

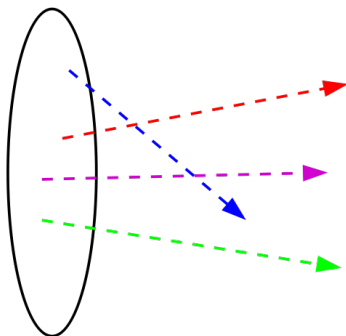
- Dependencies: c-compiler (python3/qt5 or perl/tk for gui).
- Permanent staff at DTU Physics and SOLEIL maintaining the code

How?

- Photon ray/package:
- $(\mathbf{r}, \mathbf{k}, \varphi, t, p, \mathbf{E})$
- \mathbf{r} - spatial coordinates
- \mathbf{k} - wave vector
- φ - phase
- t - time
- p - photon weight
- \mathbf{E} - Electrical field polarisation

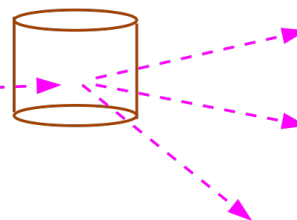


1. Particles emitted with random starting conditions via MC



2. Particles are "ray-traced" through space

3. Will eventually meet other objects e.g. a studied experimental sample and get scattered via MC again



4. At various points in the instrument the particle states are measured in so-called monitors or detectors



Important efficiency mechanisms:

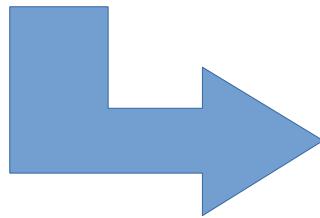
- "Focusing" - e.g. lab source to first slit (4π vs. limited solid angle only)
- Rather vs. single particle description, absorption handled through statistics and downscaling the ray weight

1. Describe your beam-line in the McXtrace language
(In a text file).

2. Automatically convert beam-line into ANSI c

3. Compile

4. Run



1. Optimized for your platform
2. Only includes what you use

◆ Instrument/beam-line file – all users

◆ existing examples

◆ user written – GUI assisted

◆ Component files – some users

◆ Short pieces of code

◆ Easy to modify from existing

◆ Kernel code – McXtrace developers

◆ Propagation routines

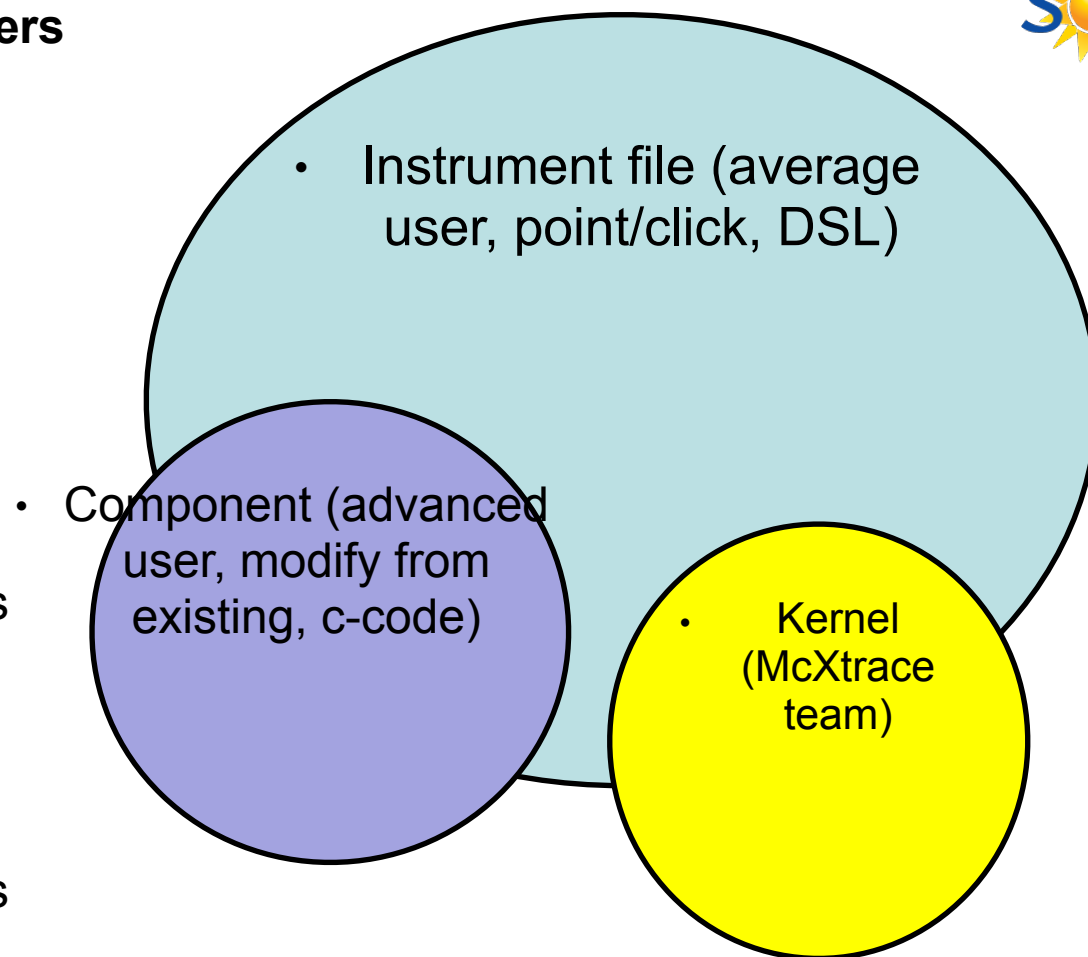
◆ Intersections

◆ Generated ISO-C code – “no” users

◆ Assembled by code generation

◆ Very low overhead of unneeded code

◆ Includes runtime libs that comps rely on (propagation etc.)



Synchrotron ID

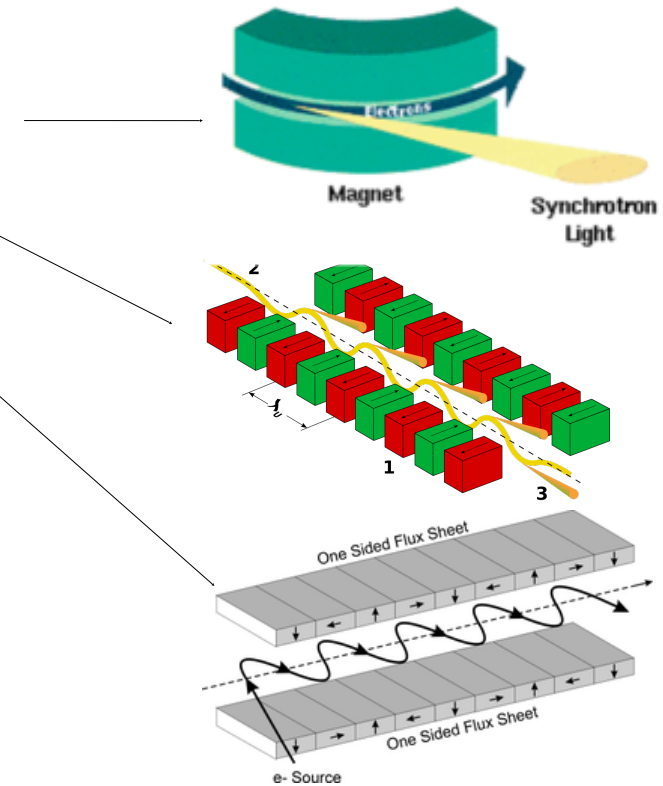
- Bending magnet [B.D. Patterson, *Am. J. Phys.* **79**, 1046 (2011)]
- Undulator [K.J. Kim, *AIP, conf. proc.*, 184, 1989]
- Wiggler [B.D. Patterson, *Am. J. Phys.* **79**, 1046 (2011)]

Lab/ideal stuff

- Laboratory X-ray tube (e.g. rotating anode)
- Ideal, point and gaussian

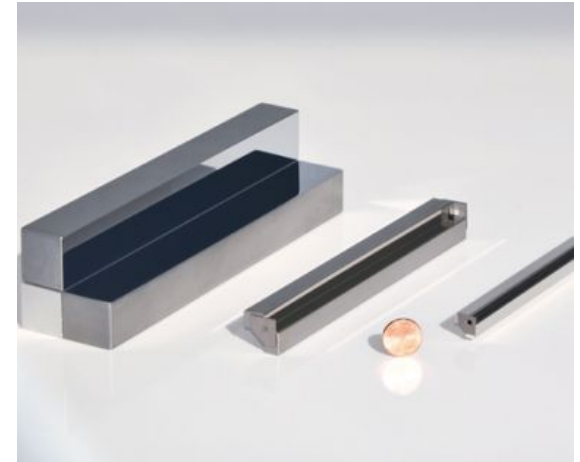
Interfaces with other software

- **Spectra** (R) <<http://spectrax.org/spectra/>>
- **Simplex** (R) <<http://spectrax.org/simplex/index.html>>
- **Genesis** (R) <<http://genesis.web.psi.ch/>>
- **Shadow** (RW) <<https://github.com/oasys-kit/shadow3>>
- **MCPL** (GEANT4, PHITS, MCNP, SRW) (RW) <<https://mctools.github.io/mcpl/>>



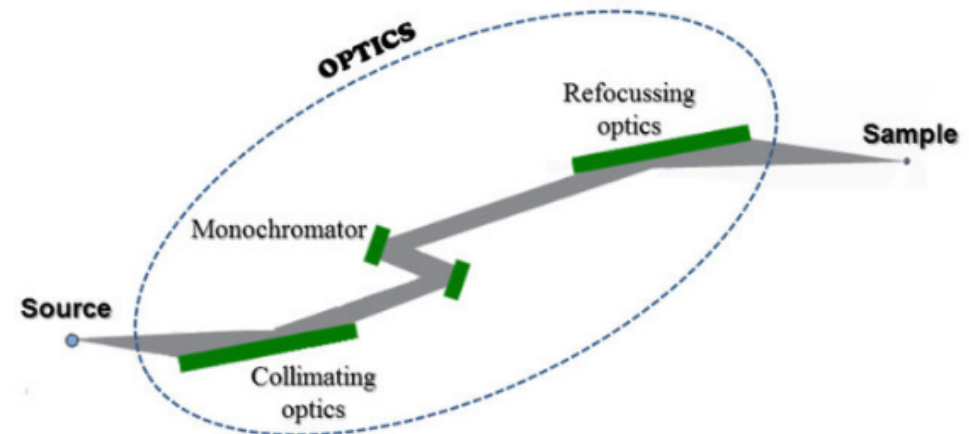
Enough to start with:

- Bragg crystal (monochromator, incl. bent)
- Capillary
- Filter (absorption and refraction)
- Lenses
- Mirrors (flat, curved, multi-layers, twin KB multi-layer)
- Zone plate
- Grating (lamellar, blazed)
- Slit, beam-stop, ...

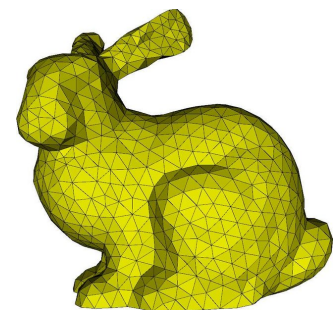
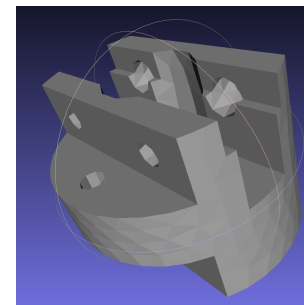


More to come (with your help ?)

Components can be arranged in groups.



- SAXS sample (60 models from SasView, PDB, Nanodiscs, Liposomes, $I(q)$, ...)
- Powder (diffraction)
- Polycrystal (diffraction)
- Pump-probe (2 states) molecule
- Single crystal (diffraction)
- Absorption (XAS)
- WIP: inelastic, fluorescence



All samples can have simple geometric shapes (incl. hollow).

Powder and SX can have any shape (PLY/STL).

Powder sample supports multiple concentric geometries (e.g. for cryostat).

McXtrace comes with a material data base, and can use e.g. NIST files.

Currently only record distributions (1D, 2D, event lists, ...) for any combination of state parameters (position, divergence, energy, power, phase, E-field, ...) → **ideal detectors**.

Can be adapted to include some more realistic efficiency.

Not satisfactory !



Actual detector efficiency to be added (simple models exist), e.g.

- <<https://bl831.als.lbl.gov/~jamesh/mlfsom/>>

SAXS

```

1 /*****
2 *      McXtrace instrument definition URL=http://www.mcxtrace.org
3 *
4 * Instrument: TestSAXS
5 *****/
6
7 DEFINE INSTRUMENT TestSAXS(
8   DistanceFromSourceToFirstPinhole = 0.05, DistanceFromSourceToSecond
   DistanceFromSecondPinholeToSample = 0.6, DistanceFromSampleToDetector
   RadiusOfDetector = 0.1, Lambda = 1.54, DLambda = 0.01, SAMPLE=0 )
9
10 TRACE
11 COMPONENT Source = Source_flat(
12   xwidth = 0.0005, yheight = 0.0005, dist = DistanceFromSourceToSecond
   focus_yh = 0.0003, focus_xw = 0.0003, lambda0 = Lambda, dlambd = DLambda
13 AT (0, 0, 0) ABSOLUTE
14
15 COMPONENT FirstPinhole = Slit(radius = 0.0002)
16 AT (0, 0, DistanceFromSourceToFirstPinhole) RELATIVE Source
17
18 COMPONENT SecondPinhole = Slit(radius = 0.00015)
19 AT (0, 0, DistanceFromSourceToSecondPinhole) RELATIVE Source
20
21 COMPONENT Sample0 = SAXSSpheres(
22   xwidth = 0.01, yheight = 0.01, zdepth = 0.01, R = 50.0, SampleToDetectorDistance =
   DistanceFromSampleToDetector, DetectorRadius = RadiusOfDetector )
23 AT(0, 0, DistanceFromSampleToDetector) RELATIVE SecondPinhole
24
25 COMPONENT Beamstop = Beamstop(
26   radius = 0.001)
27 AT (0, 0, DistanceFromSampleToDetector - 0.000001) RELATIVE Sample0
28
29 COMPONENT Monitor = PSD_monitor(
30   filename = "PSDMonitor", xwidth = 2.0 * RadiusOfDetector / sqrt(2)
   * RadiusOfDetector / sqrt(2.0), nx = 200, ny = 200, restore_xray =
31 AT (0, 0, 0.000001) RELATIVE Beamstop
32
33 COMPONENT QMonitor = SAXSQMonitor(
34   RadiusDetector = RadiusOfDetector, DistanceFromSample =
   DistanceFromSampleToDetector, LambdaMin = Lambda, Lambda0 = Lambda,
   2000 )
35 AT (0, 0, 0.0000010000000001) RELATIVE Beamstop
36
37 END
38

```

Start simulation

Instrument parameters (D=floating point, I=integer, S=string)

DistanceFromSourceToFirstPinhole:	0.05	DistanceFromSourceToSecondPinhole:	0.7	DistanceFromSecondPinholeToSample:	0.6
DistanceFromSampleToDetector:	0.48	RadiusOfDetector:	0.1	Lambda:	1.54
DLambda:	0.01	SAMPLE:	0		

Simulation

Simulation/Trace: Simulation

Autoplot: -- None --

Ray count: 1000000

Output subdir (optional):

Sweep steps (optional):

MPI: No clustering

MPI node count: 2


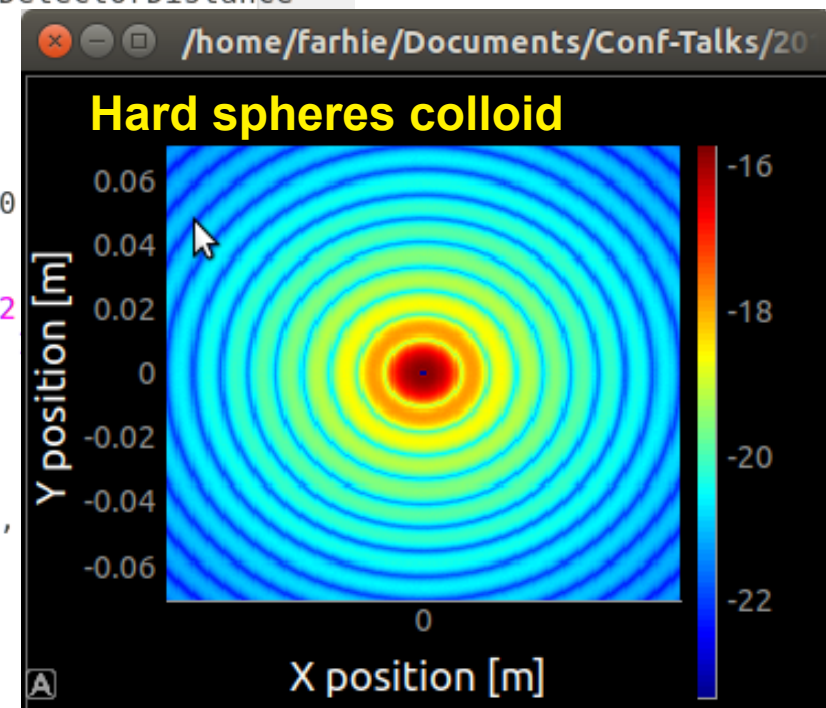
Advanced

Random seed:

Gravity: Off

Start Cancel

11 s

Powder Diff.

```

1 /*****
2 * Instrument: Test_PowderN
3 * %Parameters
4 * TTH: [deg.] Two theta rotation for detector arm
5 * %End
6 *****/
7
8 DEFINE INSTRUMENT Test_PowderN(TTH=13.4)
9
10 TRACE
11 COMPONENT src = Source_flat(
12     yheight = 1e-3, xwidth = 1e-3, dist = 10, focus_xw = 1e-3,
13     focus_yh = 1e-3, E0 = 45, dE = 1)
14 AT (0, 0, 0) ABSOLUTE
15
16 COMPONENT psd0 = PSD_monitor(
17     nx = 100, ny = 100, filename = "psd0", xwidth = 2e-3, yheight = 2e-3)
18 AT (0, 0, 1e-9) RELATIVE src
19
20 COMPONENT sample = PowderN(
21     reflections = "Fe_bcc229_lt13deg.dat", format=Crystallographica,
22     material = "Fe.txt", radius = .5e-4, yheight = 1e-3, pack = 0.5, Vc =
23     p_inc = 0, p_transmit = 0.1, DW = 0, d_phi = 45)
24 AT (0, 0, 10) RELATIVE PREVIOUS
25 EXTEND %{
26     if(!SCATTERED) ABSORB;
27 %}
28
29 COMPONENT psd4pi = PSD_monitor_4PI(
30     nx = 180, ny = 180, filename = "psd4pi", radius = 0.1, re
31 AT (0, 0, 0) RELATIVE sample
32
33 COMPONENT detector = PSD_monitor(
34     nx=200, ny=200, xwidth=0.1, yheight=0.1, filename="psd1",
35 AT(0,0,0.1) RELATIVE sample
36
37 COMPONENT ttharm= Arm()
38 AT(0,0,0) RELATIVE sample
39 ROTATED (0,TTH,0) RELATIVE sample
40
41 COMPONENT detector2 = PSD_monitor(
42     nx=200, ny=200, xwidth=2e-3, yheight=1e-2, filename="psd2",restore_xray=1)
43 AT(0,0,0.1) RELATIVE ttharm
44 END

```

Start simulation

Instrument parameters (D=floating point, I=integer, S=string)

TTH:

Simulation

Simulation/Trace:

Autoplot:

Ray count:

Output subdir (optional):

Sweep steps (optional):

MPI:

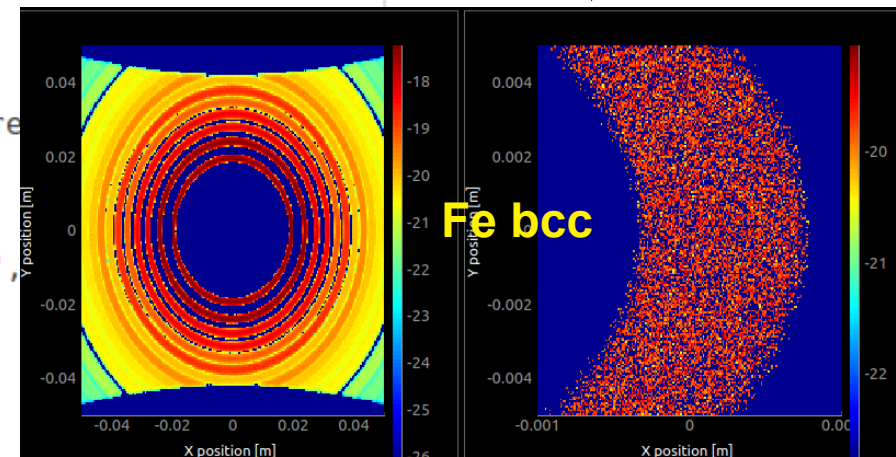
MPI node count:

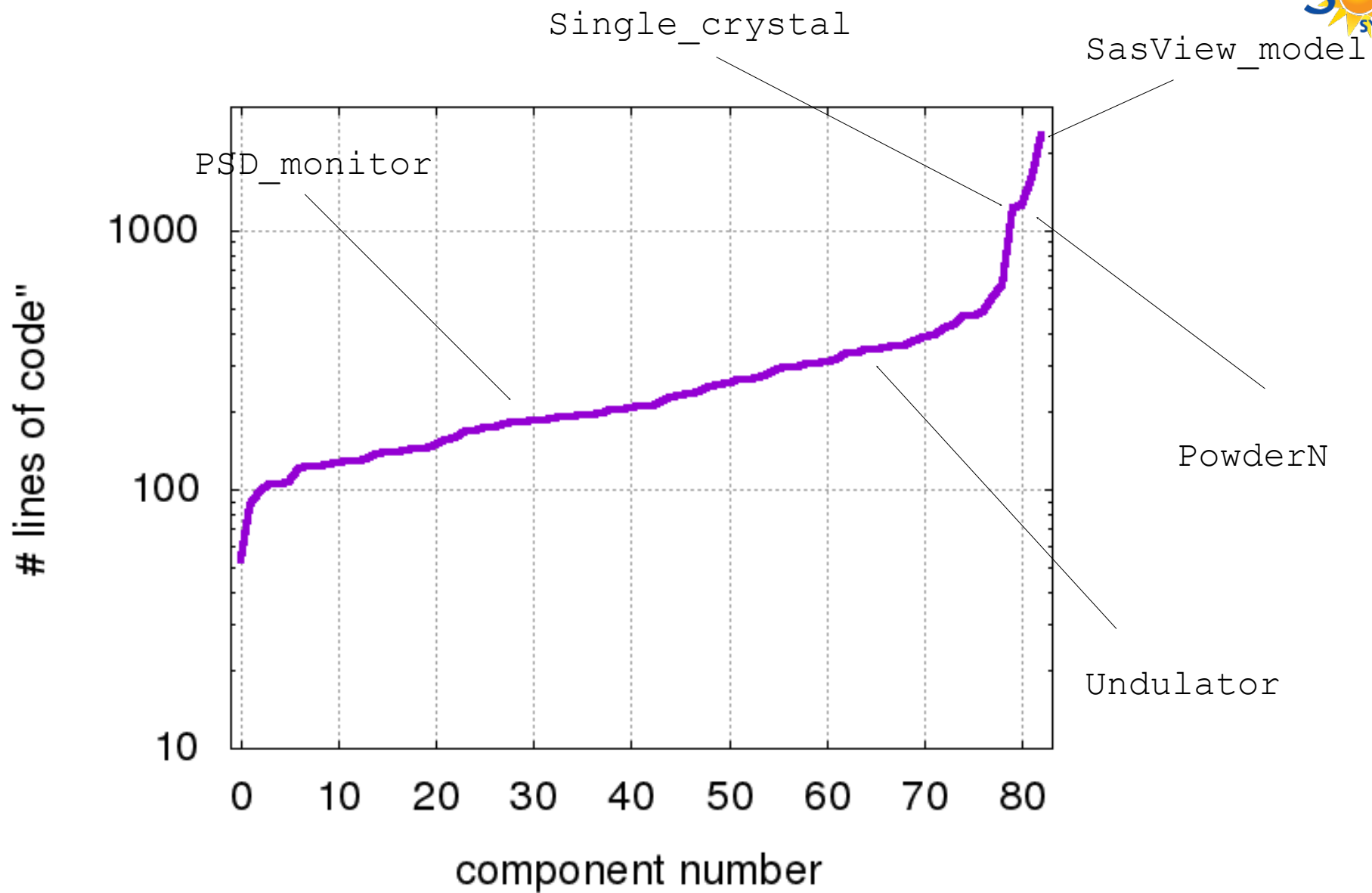
Advanced

Random seed:

Gravity:

6 s





Component file (source, optics, sample, det., ...)

```

PSD_monitor.comp (~/Repositories/McCode/mcxttrace-comps/monitors) - GVIM10
File Edit Tools Syntax Buffers Window Help

DEFINE COMPONENT PSD_monitor
DEFINITION PARAMETERS (nx=90, ny=90, nr=0, string filename=0, restore_xray=0)
SETTING PARAMETERS (xmin=-0.05, xmax=0.05, ymin=-0.05, ymax=0.05, xwidth=0, yheight=0, radius=0)
OUTPUT PARAMETERS (PSD_N, PSD_p, PSD_p2)
/* X-ray parameters: (X,y,z,kx,ky,kz,phi,t,Ex,Ey,Ez,p) */

DECLARE
%{
double **PSD_N;
double **PSD_p;
double **PSD_p2;
}%

INITIALIZE
%{
int i,j;
double *p1,*p2,*p3;

if (xwidth > 0) { xmax = xwidth/2; xmin = -xmax; }
if (yheight > 0) { ymax = yheight/2; ymin = -ymax; }

if ( ((xmin >= xmax) || (ymin >= ymax)) && !radius ) {
printf("PSD_monitor: %s: Null detection area !\n",
"ERROR (xwidth,yheight,xmin,xmax,ymin,ymax,radius). Exiting",
NAME_CURRENT_COMP);
exit(0);
}

if(!radius){
p1=calloc(nx*ny,sizeof(double));
p2=calloc(nx*ny,sizeof(double));
p3=calloc(nx*ny,sizeof(double));

PSD_N=calloc(nx,sizeof(double *));
PSD_p=calloc(nx,sizeof(double *));
PSD_p2=calloc(nx,sizeof(double *));

for (i=0; i<nx; i++){
PSD_N[i]=&p1[i*ny]; //calloc(ny,sizeof(double));
PSD_p[i]=&p2[i*ny]; //calloc(ny,sizeof(double));
PSD_p2[i]=&p3[i*ny]; //calloc(ny,sizeof(double));
}
}else{
PSD_N=calloc(1,sizeof(double *));
PSD_p=calloc(1,sizeof(double *));
PSD_p2=calloc(1,sizeof(double *));
*PSD_N=calloc(nr,sizeof(double));
*PSD_p=calloc(nr,sizeof(double));
*PSD_p2=calloc(nr,sizeof(double));
}

}%

TRACE
%{
int i,j;
PROP_ZO;
if (!radius){
if (x>xmin && x<xmax && y>ymin && y<ymax)
{
i = floor((x - xmin)*nx/(xmax - xmin));
j = floor((y - ymin)*ny/(ymax - ymin));
PSD_N[i][j]++;
PSD_p[i][j] += p;
PSD_p2[i][j] += p*p;
SCATTER;
}
}else{
double r=sqrt(x*x+y*y);
if (r<radius){
i = floor(r*nr/radius);
PSD_N[0][i]++;
PSD_p[0][i] += p;
SCATTER;
}
}

j = floor((y - ymin)*ny/(ymax - ymin));
PSD_N[i][j]++;
PSD_p[i][j] += p;
PSD_p2[i][j] += p*p;
SCATTER;
}
}else{
double r=sqrt(x*x+y*y);
if (r<radius){
i = floor(r*nr/radius);
PSD_N[0][i]++;
PSD_p[0][i] += p;
SCATTER;
}
}

if (restore_xray) {
RESTORE_XRAY(INDEX_CURRENT_COMP, x, y, z, kx, ky, kz, phi, t, Ex, Ey, Ez, p);
}

}%

SAVE
%{
if(!radius){
DETECTOR_OUT_2D(
"PSD_monitor",
"X position [m]",
"Y position [m]",
xmin, xmax, ymin, ymax,
nx, ny,
*PSD_N,*PSD_p,*PSD_p2,
filename);
}else{
DETECTOR_OUT_1D(
"PSD_monitor","Radial Position[m]", "Intensity", "R",
0,radius,nr,&PSD_N[0][0],&PSD_p[0][0],&PSD_p2[0][0],filename);
}

}%

%}

FINALLY
%{
free(PSD_N[0]);
free(PSD_N);
free(PSD_p[0]);
free(PSD_p);
free(PSD_p2[0]);
free(PSD_p2);
}%

MCDISPLAY
%{
magnify("xy");
multiline(5, (double)xmin, (double)ymin, 0.0,
(double)xmax, (double)ymin, 0.0,
(double)xmax, (double)ymax, 0.0,
(double)xmin, (double)ymax, 0.0,
(double)xmin, (double)ymin, 0.0);
}%

END

```

Users don't always have access to synchrotrons

- Prospective users may try out experiments.
- Teach users how to do experiments.
- Understand what is happening along the beam-line.

- Community driven code
- Fairly easy to extend – write a new component and it automatically fits into the framework.
- Portable (just need a c-compiler).
- Several included standard sample models.
- Plenty of examples.
- Once written, a simulation/beam-line is a real program in itself.

...but...

- Use whatever you prefer! XRT, SHADOW, SRW, Ray,...<insert package here>...