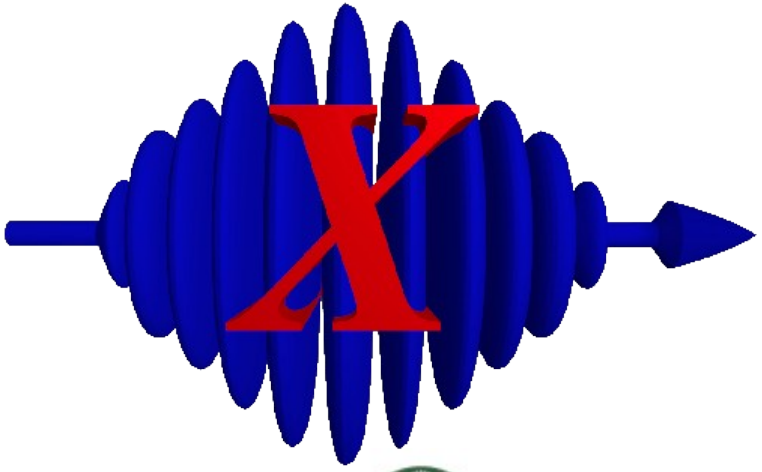
McXtrace



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)

What is it?

- 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)

H

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

Project mailing list at
mcxtrace-users@mcxtrace.org

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

- GPL-license
- DSL / Compiler Technology.

Using Lex & Yacc

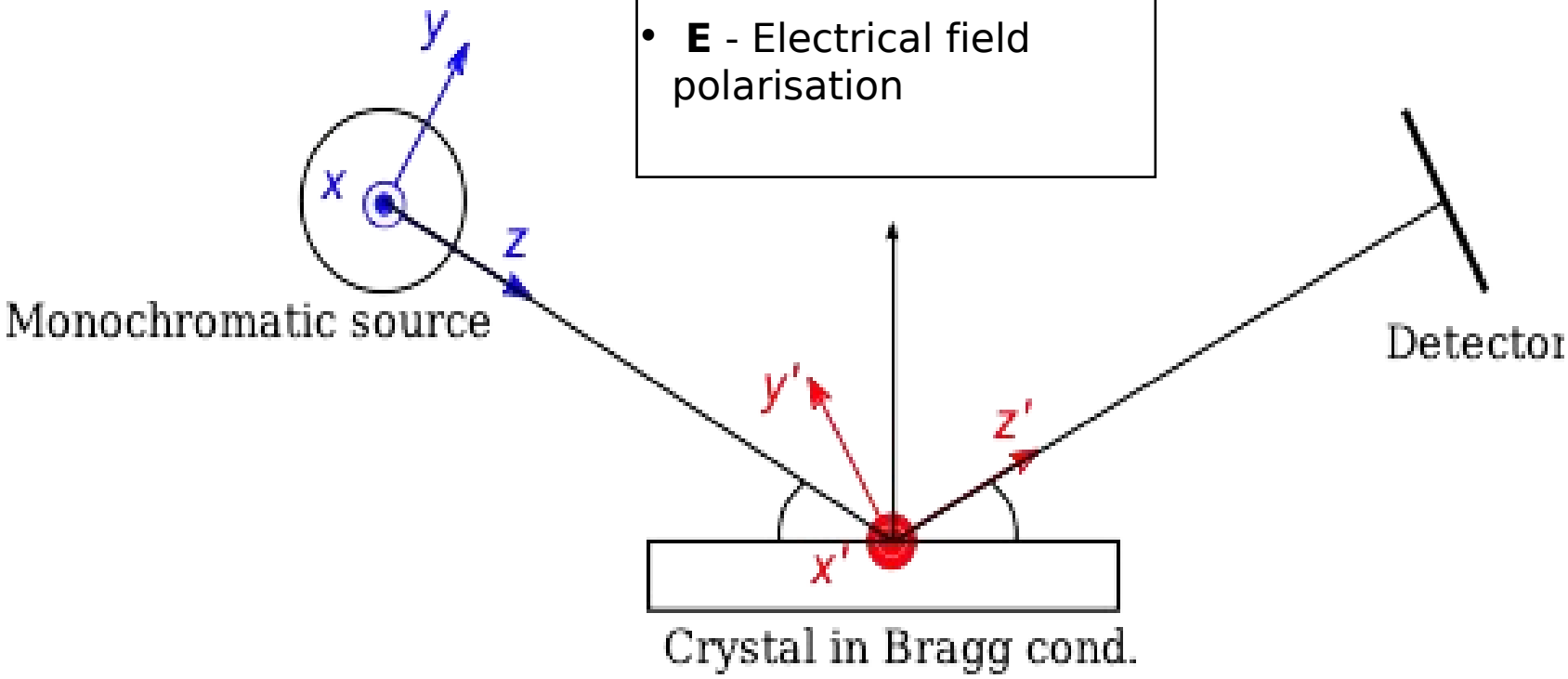
- Modular Open Structure.

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

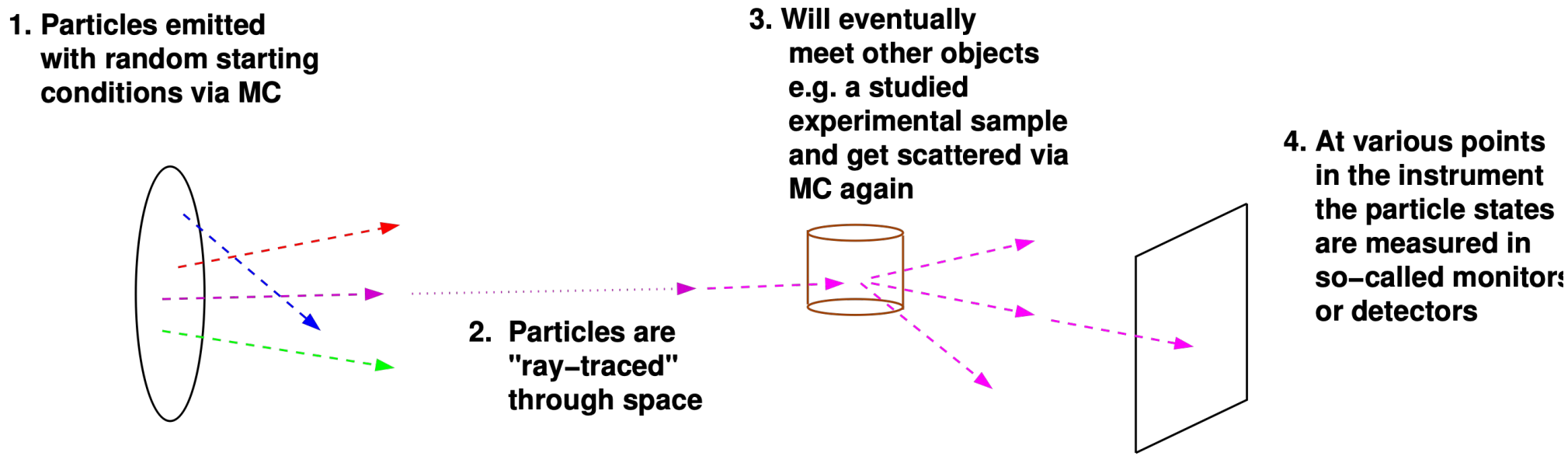
- Dependencies: c-compiler (python3/qt5 or perl/tk for gui).
- Permanent staff at DTU Physics and SOLEIL maintaining the code

How?

- Photon ray/package:
- (**r**,**k**,φ,t,p,**E**)
- **r** - spatial coordinates
- **k** - wave vector
- φ - phase
- t - time
- p – photon weight
- **E** - Electrical field polarisation



Overall picture: Monte Carlo Ray-Tracing



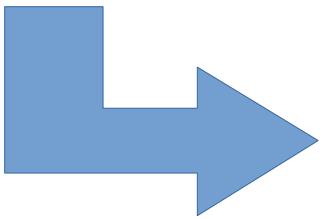
Important efficiency mechanisms:

"Focusing" - e.g. lab source to first slit (4π vs. limited solid angle only)

Rather vs. single particle description, absorption handled though statistics and downscaling the ray weight

The McXtrace way

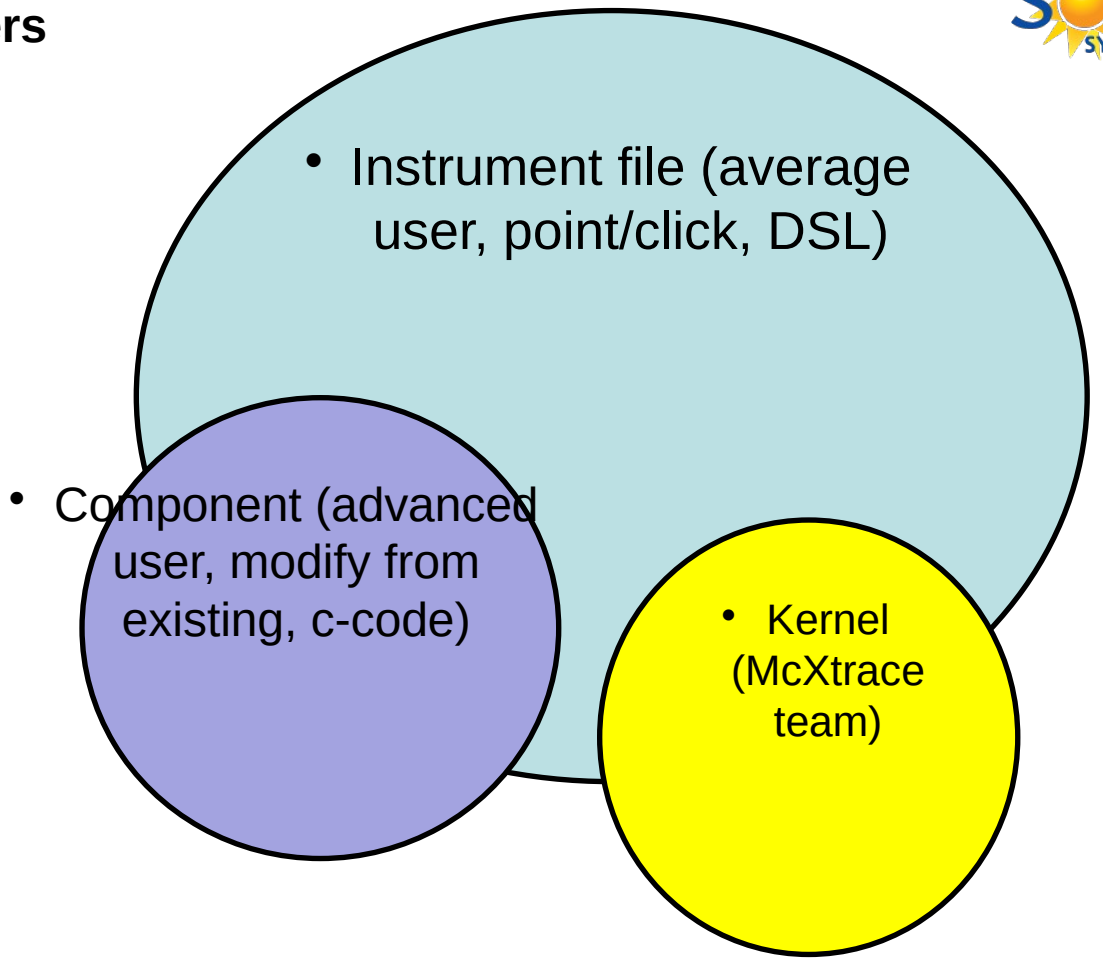
1. Describe your beam-line in the McXtrace language (In a text file).
2. Automatically convert beam-line into ANSI c
3. Compile
4. Run



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

Code Levels

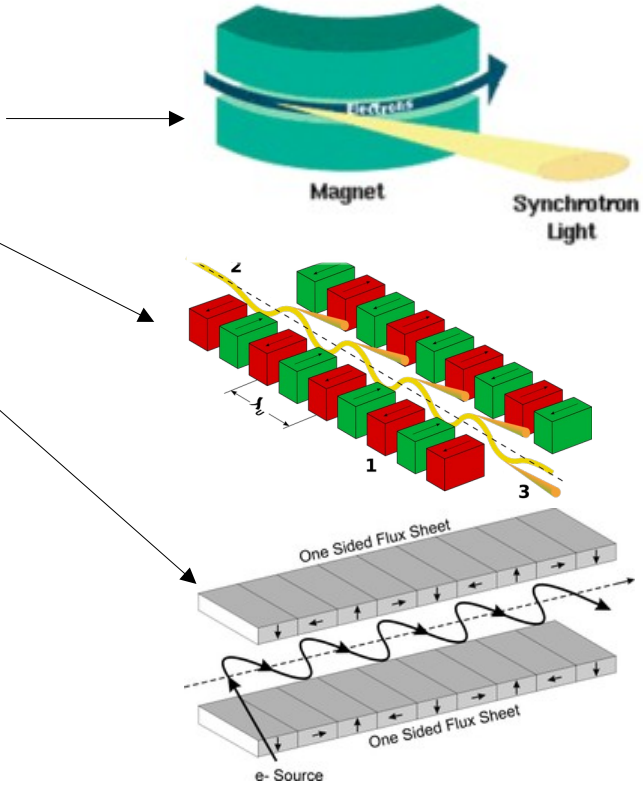
- **Instrument/beam-line file – all users**
  - **existing examples**
  - **user written – GUI assisted**
- Component files – some users
  - Short pieces of code
  - Easy to modify from existing
- Kernel code – McXtrace developers
  - Propagation routines
  - Intersections
- Generated ISO-C code – “no” users
  - Assembled by code generation
  - Very low overhead of unneeded code
  - Includes runtime libs that comps rely on (propagation etc.)



Components: Sources

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

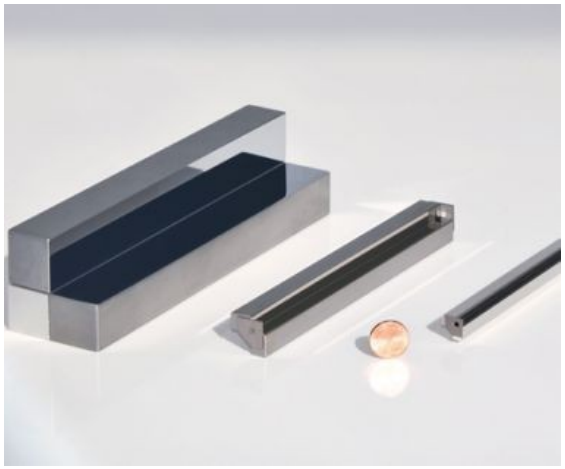
Interfaces with other software

- ***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/>

Components: Optics

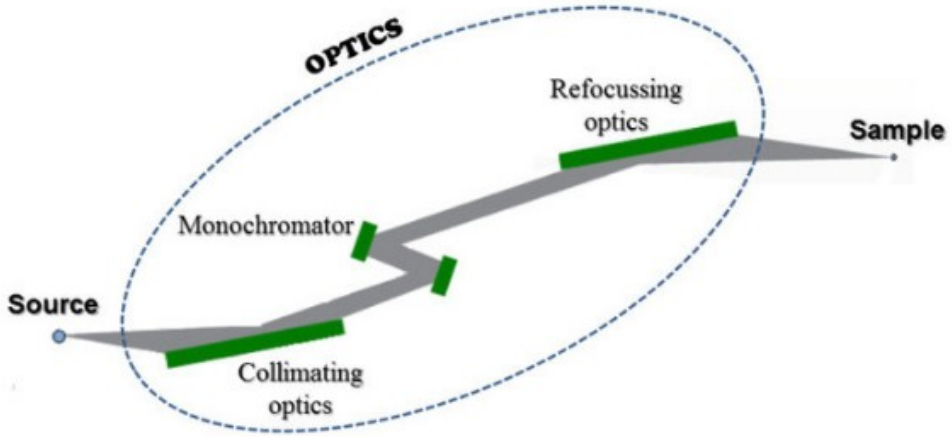
Enough to start with:

- 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, ...



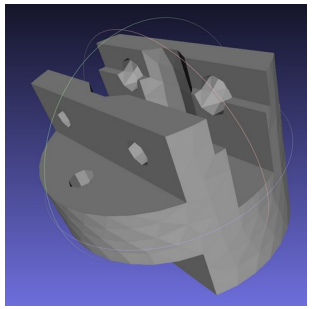
More to come (with your help ?)

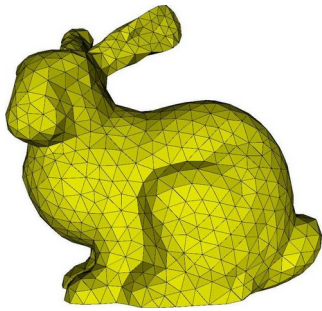
Components can be arranged in groups.



Components: Samples

- 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





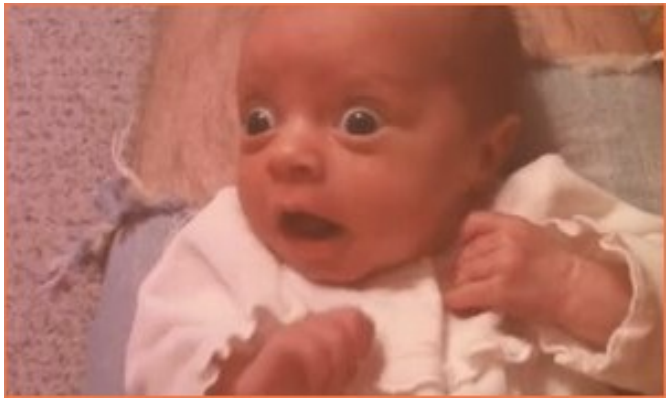
All samples can have simple geometric shapes (incl. hollow).
Powder and SX can have any shape (PLY/STL).
Powder sample supports multiple concentric geometries (*e.g.* for cryostat).
McXtrace comes with a material data base, and can use *e.g.* NIST files.

Components: Detectors

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 !

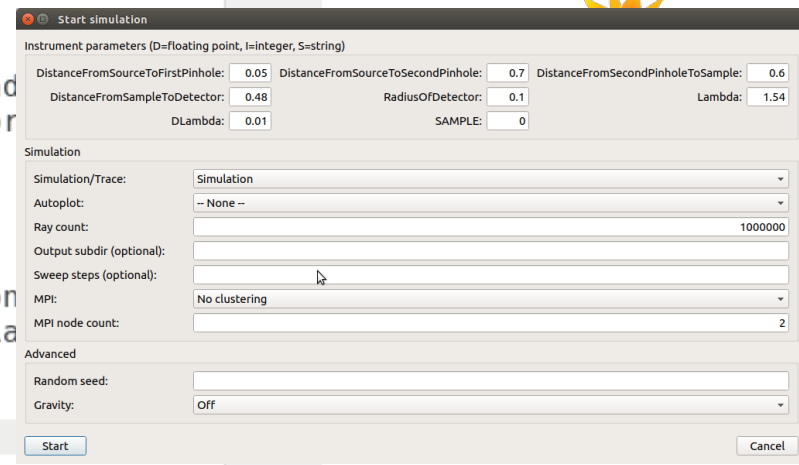


Actual detector efficiency to be added (simple models exist), e.g.

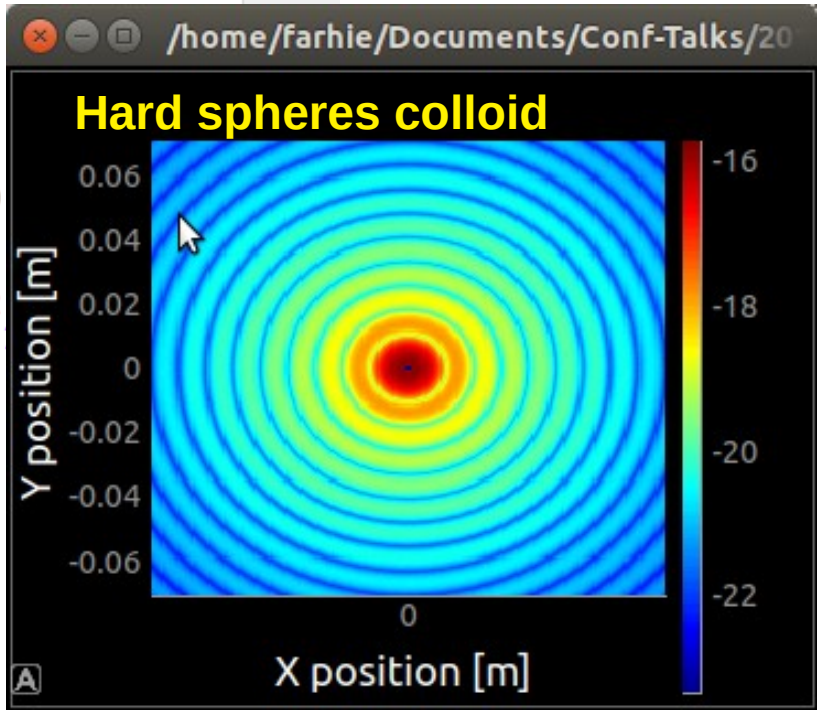
- <https://bl831.als.lbl.gov/~jamesh/mlfsom/>

SAXS

```
/*******************************************************************************
*         McXtrace instrument definition URL=http://www.mcxtrace.org
*
* Instrument: TestSAXS
*******************************************************************************/

DEFINE INSTRUMENT TestSAXS(
  DistanceFromSourceToFirstPinhole = 0.05, DistanceFromSourceToSecond
 DistanceFromSecondPinholeToSample = 0.6, DistanceFromSampleToDetector
 RadiusOfDetector = 0.1, Lambda = 1.54, DLambda = 0.01, SAMPLE=0 )

TRACE
COMPONENT Source = Source_flat(
  xwidth = 0.0005, yheight = 0.0005, dist = DistanceFromSourceToSecon
 focus_yh = 0.0003, focus_xw = 0.0003, lambda0 = Lambda, dlambda = DLa
AT (0, 0, 0) ABSOLUTE

COMPONENT FirstPinhole = Slit(radius = 0.0002)
AT (0, 0, DistanceFromSourceToFirstPinhole) RELATIVE Source

COMPONENT SecondPinhole = Slit(radius = 0.00015)
AT (0, 0, DistanceFromSourceToSecondPinhole) RELATIVE Source

COMPONENT Sample0 = SAXSSpheres(
  xwidth = 0.01, yheight = 0.01, zdepth = 0.01, R = 50.0, SampleToDetectorDistance =
 DistanceFromSampleToDetector, DetectorRadius = RadiusOfDetector )
AT(0, 0, DistanceFromSampleToDetector) RELATIVE SecondPinhole

COMPONENT Beamstop = Beamstop(
  radius = 0.001)
AT (0, 0, DistanceFromSampleToDetector - 0.000001) RELATIVE Sample0

COMPONENT Monitor = PSD_monitor(
  filename = "PSDMonitor", xwidth = 2.0 * RadiusOfDetector / sqrt(2
 * RadiusOfDetector / sqrt(2.0), nx = 200, ny = 200, restore_xray =
AT (0, 0, 0.000001) RELATIVE Beamstop

COMPONENT QMonitor = SAXSQMonitor(
  RadiusDetector = RadiusOfDetector, DistanceFromSample =
 DistanceFromSampleToDetector, LambdaMin = Lambda, Lambda0 = Lambda,
 2000 )
AT (0, 0, 0.000001000000001) RELATIVE Beamstop

END
```

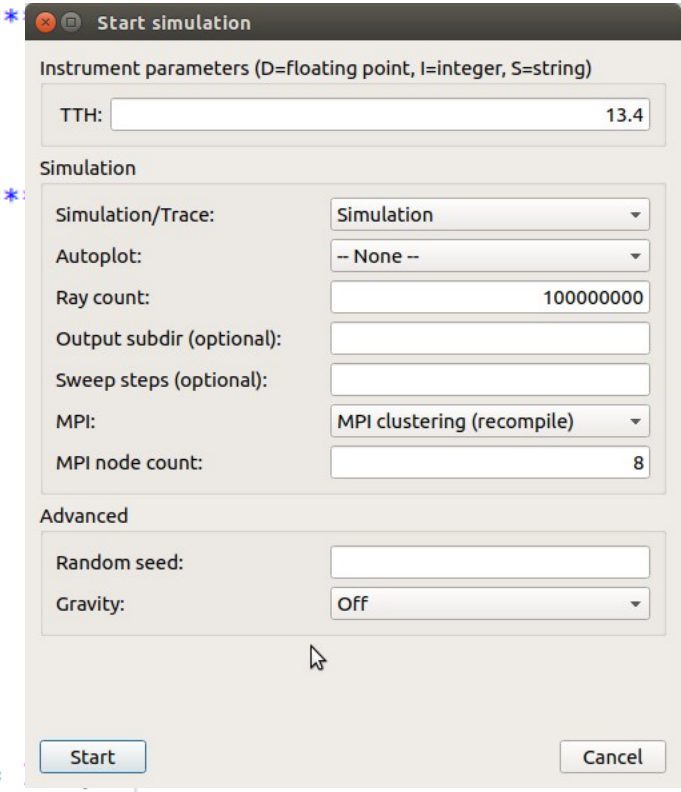


11 s

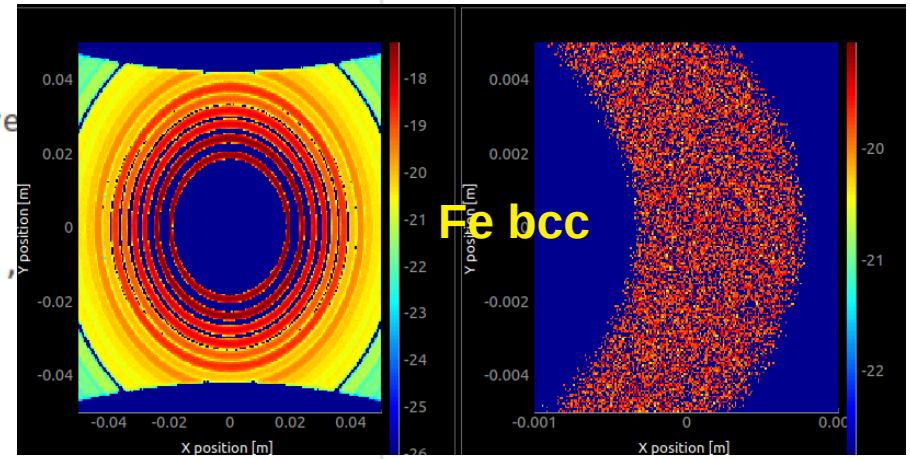


Powder Diff.

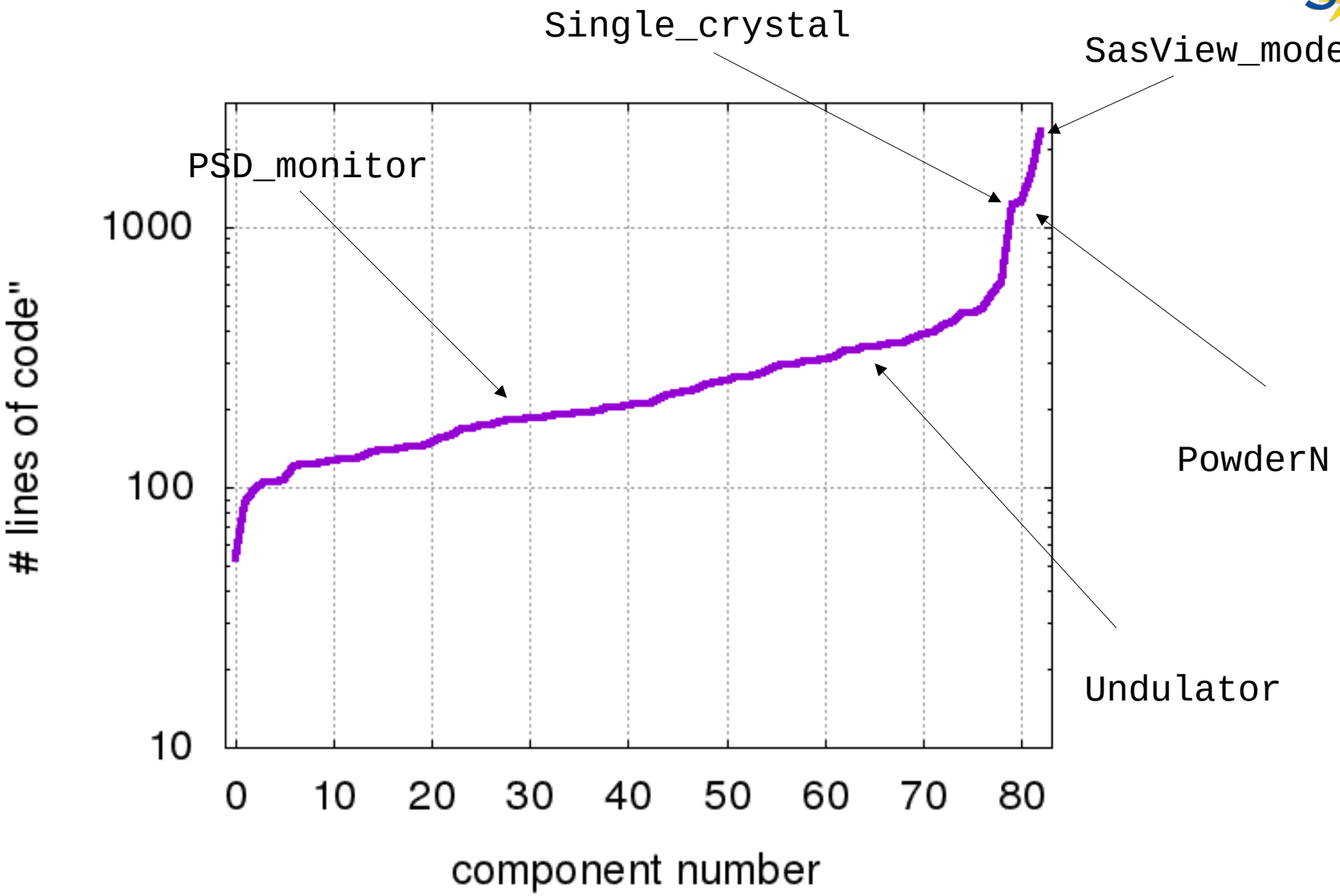
```
/*******************************************************************************
* Instrument: Test_PowderN
* %Parameters
* TTH: [deg.] Two theta rotation for detector arm
* %End
*******************************************************************************/

DEFINE INSTRUMENT Test_PowderN(TTH=13.4)

TRACE
COMPONENT src = Source_flat(
    yheight = 1e-3, xwidth = 1e-3, dist = 10, focus_xw = 1e-3,
    focus_yh = 1e-3, E0 = 45, dE = 1)
  AT (0, 0, 0) ABSOLUTE

COMPONENT psd0 = PSD_monitor(
    nx = 100, ny = 100, filename = "psd0", xwidth = 2e-3, yheight = 2e-3)
  AT (0, 0, 1e-9) RELATIVE src

COMPONENT sample = PowderN(
    reflections = "Fe_bcc229_lt13deg.dat", format=Crystallographica,
    material = "Fe.txt", radius = .5e-4, yheight = 1e-3, pack = 0.5, Vc =
    p_inc = 0, p_transmit = 0.1, DW = 0, d_phi = 45)
  AT (0, 0, 10) RELATIVE PREVIOUS
EXTEND %{
  if(!SCATTERED) ABSORB;
%}

COMPONENT psd4pi = PSD_monitor_4PI(
    nx = 180, ny = 180, filename = "psd4pi", radius = 0.1, re
  AT (0, 0, 0) RELATIVE sample

COMPONENT detector = PSD_monitor(
    nx=200, ny=200, xwidth=0.1, yheight=0.1, filename="psd1",
  AT(0,0,0.1) RELATIVE sample

COMPONENT ttharm= Arm()
  AT(0,0,0) RELATIVE sample
  ROTATED (0,TTH,0) RELATIVE sample

COMPONENT detector2 = PSD_monitor(
    nx=200, ny=200, xwidth=2e-3, yheight=1e-2, filename="psd2",restore_xray=1)
  AT(0,0,0.1) RELATIVE ttharm
END
```



6 s



Component size: typically 100-200 LOC



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

PSD_monitor.comp (~/Repositories/McCode/mcxtrace-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_Z0;
    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;
```

../McCode/mcxtrace-comps/monitors/PSD_monitor.comp 75,61 52%

```
        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;
        PSD_p2[0][i] += p*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
```

../McCode/mcxtrace-comps/monitors/PSD_monitor.comp 131,5 Bot

Why simulate a full beam-line ?

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.

Why McXtrace ?

- 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>…