XRMC (X-ray Monte Carlo) version 6.4.1

TUTORIAL

Bruno Golosio, Tom Schoonjans, Antonio Brunetti, Giovanni Luca Masala, Piernicola Oliva

Università degli Studi di Sassari

Copyright & Disclaimer

Table of Contents

1 Installation

1.1 Requirements

The following requirements exist for building XRMC:
ANSI-C Compiler and Build System
Make sure you have an ANSI-C compiler installed. The GNU C compiler (GCC) from the Free Software Foundation (FSF) is recommended. If you don't have GCC then at least make sure your vendor's compiler is ANSI compliant. In addition, your PATH must contain basic build tools such as make.
xraylib (version >= 2.15.0)
To compile XRMC you need the xraylib shared library version >= 2.15.0 .
xraylib is a library of x-ray interaction parameters, freely available for several platforms at the following URL:

http://github.com/tschoonj/xraylib

Note that only the xraylib shared library is needed for XRMC installation. The command line tool and the bindings are not necessary, although they can be valuable tools for researchers working with x-rays.
This version of XRMC was tested with xraylib v2.16.0

1.2 Extracting the source

Extract the source using any program able to extract/uncompress tar.gz archives.
For instance, in Linux, Unix and Unix-like systems you can extract the source using the gzip utility, by typing the command:

```
gunzip -xvzf xrmc-x.x.x.tar.gz
```

where x.x.x refers to the version of the program that you are installing. This will create a new directory xrmc-x.x.x under the current directory containing the source code for the distribution.

1.3 Building and installing the program

There are two main methods of installing XRMC from source: system wide installation and personal installation. The former requires root privileges on most systems. Both ways are described below.

System wide installation
After unpacking the distribution, go to the directory "xrmc-x.x.x" (where x.x.x refers to the version of the program that you are installing) and type the command:

```
./configure
```

You can then build the program by typing,

```
make
```

To verify that the build went ok, use,

```
make check
```

The installation can be performed with the command

```
make install
```

The default installation directory prefix is /usr/local.
The installation directory can be changed with the --prefix option to configure. Consult the "Further information" section below for instructions on installing the library in another location or changing other default compilation options.
On some systems you might need to update the ldconfig cache by doing:

```
ldconfig
```

Try running XRMC:

```
xrmc
```

Personal installation
After unpacking the distribution, go to the directory "xrmc-x.x.x" (where x.x.x refers to the version of the program that you are installing) and type the command:

```
./configure --prefix=DIRNAME
```

where *DIRNAME* is the name of the directory where you want to install the binary executable and the libraries.
You can then build the program by typing,

```
make
```

To verify that the build went ok, use,

```
make check
```

The installation can be performed with the command

```
make install
```

This command will install the executable in the subdirectory bin/ of the installation directory, and the libraries in the subdirectory lib/
You should add the subdirectories bin/ and /lib to the PATH and to the LD_LIBRARY_PATH environmental variables, respectively.
For instance, if you use the bash shell you can add the following lines to the file .bashrc (or

bashrc_private if you use it) in your home directory:

```
export PATH=${PATH}:DIRNAME/bin:
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:DIRNAME/lib:
```

If you use the tcsh shell you can add the following lines to the file .cshrc
(or cshrc_private if you use it) in your home directory:

```
setenv PATH ${PATH}:DIRNAME/bin:
setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:DIRNAME/lib:
```

Try running XRMC:

```
xrmc
```

1.4 Platforms known to compile and run XRMC

Linux (32-bit/64-bit)
Mac OS X (32-bit/64-bit)
Windows (native and Cygwin)

In principle, it should be possible to compile and run XRMC in any platform that supports the xraylib library. See the section "Platforms known to compile and run xraylib" in the INSTALL documentation file of xraylib for further information.

2 Support

Sorry, we don't give any kind of support to XRMC users besides this tutorial, mainly because we lack human resources for doing it. You can use the program freely if you find it useful, however please don't write us emails asking for help with your simulations, or asking for explanations if they do not give the results that you expect.

3 Bugs

For sure there are bugs in XRMC. In every new version of the code we try fix bugs from previous versions. If you find a bug, your feedback will be greatly appreciated. However, bugs should be reported in a proper way. If you believe that there is a bug in the program, please try to insulate it, reducing your setup to the minimum one that presents such problem, and send your bug report together with the configuration files to golosio@uniss.it .

4 Getting started

This section contains some basic advises for new XRMC users.

- It can be useful to start working with existing examples, which are described in section 6. You can choose the example that is more similar to the experimental conditions that you want to simulate. First, run the example as it is and display the results of the simulation. Then you can try to modify the example to make it closer to the setup that you want to simulate. It is better that you modify one device file at a time, so that in case something does not work as you want you know where is the problem.

- The current version of XRMC does not include tools to visualize the experimental setup. However, you can use the program itself to visualize radiographic projections of the sample. Even if you want to run a simulation with a single element detector (for instance if you are simulating a spectroscopy measurement) it can be useful to define also a second source and an array detector, as those used in the imaging examples, loaded by a second input file, used only for visualization purpose. In this way you can check if the sample geometry looks correct.
- If the sample geometry does not look as you expect, try to visualize only one three-dimensional object at a time. You can do this easily by commenting all the lines corresponding to the other objects in the *geom3d* file, as described in section 5.8.
- When defining the geometric description of the sample, pay particular attention to XRMC limitations in three-dimensional objects definition, as outlined in section 5.8.
- At the beginning, run your simulations with a very small number of events, just to check that everything is working fine. Increase the number of events gradually to check that the output is reasonable before running very long simulations.

5 Using the program

Before running XRMC, you must prepare the input files that describe the experimental setup that you want to simulate. Those files include a main input file, a parameter file and some *device* files. The term *device* refers to a C++ object that is created to be used by the simulation and whose parameters are loaded from the corresponding file. It is not necessarily a physical device. For instance, the phase (material) array and the sample are considered to be devices. Each device file consists of a header that specifies the type and name of the device, a list of commands or variable names, and finally the "End" command. The order of commands/variable names is generally not important, unless they are part of the same group (for instance, the atomic number and weight percent of the elements in a material), as described in the following paragraphs. Each command/variable name can be followed by one or more arguments (real values, integer values or strings, depending on the command). Command/variable names need not to be in the same line. Comments can be inserted in all input files by preceding them by any of the following characters: ';', '!' or '#'. XRMC is case sensitive. Most commands and variable names in the input files follow the CamelCase covention, i.e. the first letter of each concatenated word is capitalized. The following paragraphs describe the commands that can be used in all device files used by the program. Although all variables have a default value, and therefore most commands are not mandatory, it is strongly recommended to explicitly assign a value to all variables in the input files.

5.1 The main input file

The main input file specifies the commands for loading all device files, the command for running the simulation and the command for saving the output in a file.
A device is created, and its parameters are loaded from a file, through the command:

Load *filename*

where *filename* is the name of the device file.
The typical setup for x-ray imaging/spectroscopy simulations include the following devices:

- a source (*source* device);
- a spectrum (*spectrum* device);
- a detector (*detectorarray* device);
- a sample (*sample* device);

- a phase (material) composition array (*composition* device);
- a quadric array (*quadricarray* device);
- a three-dimensional object geometric description (*geom3d* device).

The associated input files are described in the following sections.
The order of the "Load" commands is not relevant.
After such commands, the simulation can be started by the command

Run *devicename*

where *devicename* is the name of the device that controls the acquisition of the simulation results, i.e. in x-ray imaging/spectroscopy simulations the name of the *detectorarray* device, as specified in the header of the corresponding file.
Finally, the result of the simulation is saved by the command

Save *devicename dataname filename*

where *devicename* has the same meaning as described for the previous command, *dataname* is a name associated to the data that should be saved, and *filename* is the name of the file used to save the results. The allowed entries for *dataname* depend on the device; for the detectorarray device the only allowed entry for *dataname* is *Image.*

5.2 The spectrum file

The source spectrum is modeled as the sum of two components: a set of discrete lines and a continuous component. The radiation can be unpolarized, partially polarized or totally polarized. The lines can have a Gaussian or a Dirac δ distribution (the latter one being simply a particular case of Gaussian distribution with $\sigma = 0$). Each line is specified by its mean energy E_l, by its intensity I_l and by σ_l.
In case of (partially or totally) polarized radiation, the intensity of the two components polarized along the local x and y directions are specified separately for each line.
The continuous component is defined by N samples at arbitrary energies E_1, …., E_N, by specifying for each sample the corresponding height of the spectral distribution I_1, …., I_N. The height of the distribution in the interval between two consecutive energies of the sample E_i, E_{i+1} is approximated by a linear function of E that goes from I_i to I_{i+1}, therefore the spectrum in each interval between two consecutive samples has a trapezoidal shape. The area of the trapezium $(E_{i+1} - E_i)\cdot(I_i + I_{i+1})/2$ represents the intensity of the interval. In case of (partially or totally) polarized radiation, the height of the x and y components are specified separately for each sample of the continuous component.
There are two possible ways of extracting the initial energy of x-ray photons produced by the source:

1) extract random energies on the whole spectrum: the photon initial energy is extracted using the whole spectrum itself as a probability distribution;
2) loop on all lines and on all intervals of the spectrum: a loop is made on all lines and on all intervals of the spectrum; the initial energy of the photon is extracted according to the probability distribution limited to the single line or to the single interval; the event is assigned a weight proportional to the line/interval intensity.

The first is the traditional Monte Carlo approach. Lines or regions of the spectrum with lower intensity
are less represented in the generated statistic, no matter how important is the contribution that they give to the detected signal. There can be some drawback in this approach. For instance, if the spectrum has a relative low intensity at higher energies and if the sample is a strongly absorbing object, the statistic of events with higher energies will be low even though they

give the most important contribution to the detected signal.
The second approach is closer to deterministic integration methods and should normally be preferred to the first one. All lines and all interval are equally represented in the generated statistic, and their relative probability is corrected by using the method of weighting the event.
If the second method is chosen, than for each interval of the continuous component there are two possible ways of extracting the photon energy:

1) extract the energy randomly according to the probability distribution inside the interval itself (which is modeled by a linear function, as discussed previously);
2) force the photon energy to be equal to the central energy of the interval.

The second method is the *pure deterministic* approach, and normally it should not be used.
The program offers the opportunity to resample the continuous component after its definition. In this case, the user must specify the starting energy, the energy step and the number of points used for the resampling. The intensities I_i are then recalculated for the new values of E_i. Normally this option will not be used, however it may be useful for variance reduction if the space between sampling energies in the continuous component definition is too large and a finer separation is desired or, in the opposite case, if the energy step in the initial definition of the spectrum is unnecessarily too small.

spectrum device file header:

```
Newdevice spectrum      ; Device type
name (string)           ; Device name
```

Commands:

PolarizedFlag *val*
specifies if the beam is polarized:
val = 0: unpolarized beam;
val = 1: polarized beam;

LoopFlag *val*
energy extraction method:
val = 0: extract random energies on the whole spectrum;
val = 1: loop on all lines and sampling points;

ContinuousPhotonNum *val*
val (integer): multiplicity of events for each interval in spectrum;

LinePhotonNum *val*
val (integer): multiplicity of events for each line in spectrum;

RandomEneFlag *val*
enable/disable random energy in each interval of the continuous spectrum;
val = 0: random energy disabled;
val = 1: random energy enabled (recommended);

Lines

specifies the discrete energy lines of the spectrum;
energies and σ are expressed in keV:

N_l (integer): Number of lines in the spectrum

E_l σ_l I_l (real values): energy, width (rms) and intensity of the 1st line

…..

E_{Nl} σ_{Nl} I_{Nl} (real values): energy, width (rms) and intensity of the Nth line

for polarized beam, or

E_l σ_l I_{x1} I_{y1} (real values):

energy, width (rms) and intensities of the two polarization components of the 1st line

…..

E_{Nl} σ_{Nl} I_{xNl} I_{yNl} (real values):

energy, width (rms) and intensities of the two polarization components of the Nth line

ContinuousSpectrum

continuous component of the spectrum;

N_I (integer): Number sampling points in the continuous spectrum

filename : name of the file containing the continuous spectrum;

in the case of unpolarized beam, the file has the following format:

E_l I_l (real values): energy and intensity of the 1st sampling point

…..

E_{NI} I_{NI} (real values): energy and intensity of the Nth sampling point;

while for polarized beam it has the following format:

E_l I_{x1} I_{y1} (real values):

energy and intensities of the two polarization components of the 1st sampling point

…..

E_{NI} I_{xNI} I_{yNI} (real values):

energy and intensityies of the two polarization components of the Nth sampling point;

Resample *val*

val = 0: do not resample continuous spectrum;

val = 1: resample continuous component of the spectrum;

If *val* = 1, the following arguments must also be provided:

N_R (integer): number of resampling points;

E_{min} (real): minimum resampling energy (keV);

E_{max} (real): maximum resampling energy (keV);

End

End of file.

The total number of generated events is given by the product of the multiplicities in the spectrum, in the interactions with the sample and in the detector pixels.

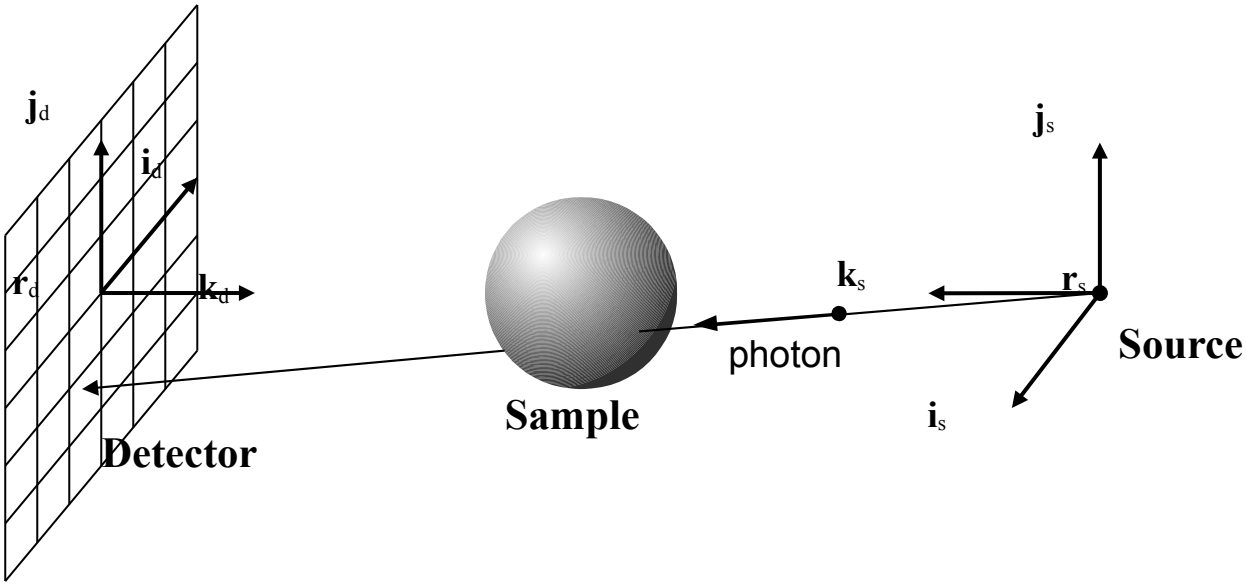

Figure 1. The standard experimental setup simulated by XRMC consists of a x-ray source, a sample and a detector (two-dimensional array or single element). A local coordinate system is attached to the source and to the detector. $\mathbf{r}_s$ and $\mathbf{r}_d$ are the position vector of the source and of the detector geometric center, respectively. $\mathbf{i}_s$, $\mathbf{j}_s$ and $\mathbf{k}_s$ are the directions of the source local *x*, *y*, and *z* axis, respectively. $\mathbf{i}_d$, $\mathbf{j}_d$ and $\mathbf{k}_d$ are the directions of the detector local *x*, *y*, and *z* axis, respectively. The detector local *z* axis is perpendicular to the detector surface, while the local *x* and *y* axes are parallel to the detector rows and columns, respectively.

5.3 The source file

The current version of the code assumes that the x-ray beam is produced by a point source or by an extended source with a three-dimensional Gaussian distribution. A local coordinate system is associated to the source, specified by the vector position of the origin $\mathbf{r}_s$ and by the orthonormal unit vectors $\mathbf{i}_s$, $\mathbf{j}_s$ and $\mathbf{k}_s$ (see Fig. 1), which are the directions of the local x, y, and z axis, respectively. The local z axis represents the main source direction, while the local x and y axes are used to define the beam polarization and angular aperture. Let θ_s and ϕ_s be the polar and azimuthal angle, respectively, relative to the source coordinate system. The user can specify the angular apertures θ_x and θ_y in the x and in the y direction, respectively. In general, the angular aperture of the source is elliptical and it is defined by the inequality:

$$\theta^2 \leq \theta_x^2 \cos^2\phi_s + \theta_y^2 \sin^2\phi_s$$

The source intensity distribution is assumed to be uniform within the solid angle limited by this angular aperture.

source device file header:

Newdevice source	; Device type
name (string)	; Device name

Commands:

SpectrumName *name*

name (string): spectrum input device name;

X *val1 val2 val3*

val1, *val2*, *val3* (real values) source x, y, z coordinates

uk *val1 val2 val3*

source orientation: $\mathbf{k}_s$ components (local z axis direction, i.e. main source direction);

val1 val2 val3 (real values): k_{sx}, k_{sy}, k_{sz}

ui *val1 val2 val3*

source orientation: $\mathbf{i}_s$ components (local x axis direction);

val1 val2 val3 (real values): i_{sx}, i_{sy}, i_{sz}

Divergence *val1 val2*

beam divergence:

val1 val2 (real values): θ_x , θ_y

Size *val1 val2 val3*

Source size; the source is modeled as a three-dimensional Gaussian distribution:

val1 val2 val3 (real values) σ_x, σ_y, σ_z ;

Rotate *val1 val2 val3 val4 val5 val6 val7*

rotation of the source around the axis passing through $\mathbf{x}_0$ and with direction $\mathbf{u}$:

val1 val2 val3 (real values): x_0, y_0, z_0 ;

val4 val5 val6 (real values): u_x, u_y, u_z ;

val7 (real) rotation angle θ (degrees);

End

End of file.

If $\mathbf{i}_s$ is not perpendicular to $\mathbf{k}_s$, then the program replace it by a vector on the same common plane, but perpendicular to $\mathbf{k}_s$. If $\mathbf{k}_s$ or $\mathbf{i}_s$ are not unit vectors then they are normalized by the program. The vector $\mathbf{j}_s$ is computed by the program, so that $\mathbf{i}_s$, $\mathbf{j}_s$, $\mathbf{k}_s$ are orthonormal unit vectors.

5.4 The detector file

In general, the program can simulate two-dimensional array detectors with energy binning for each pixel. A single element detector can be simulated as a special case of array detector with only one pixel. The pixel shape can be defined as rectangular or elliptical. The latter possibility is particularly useful when a round, single element detector has to be simulated.
A local coordinate system is associated to the detector, specified by the vector position of its geometric center $\mathbf{r}_d$ and by the orthonormal unit vectors $\mathbf{i}_d$, $\mathbf{j}_d$ and $\mathbf{k}_d$ (see Fig. 1), which are the directions of the local x, y, and z axis, respectively. The local z axis is perpendicular to the

detector surface, while the local x and y axes are parallel to the detector rows and columns, respectively.
According to a variance reduction technique used in the code, each event is forced to end with a photon that reaches a pixel of the detector (and its weight is multiplied by a proper probability factor). The intersection between the last photon trajectory and the pixel can be a random position on the pixel surface, or it can be forced to be the midpoint of the pixel. The latter method is a *pure deterministic* approach, and normally it should not be used.
The user has the possibility to simulate a statistical uncertainty on pixel counts based on Poisson statistic. In the traditional Monte Carlo approach the number of photons detected by each channel of each pixel is always an integer number. Using variance reduction techniques the events are weighted, therefore the estimated number of detected photons is a real number. However it is possible to round it to the closest integer number.
The weight associated to the probability that the last photon of an event reaches a detector pixel is proportional to a geometric efficiency factor ε_{geom}, which is related to the solid angle from the interaction point to the pixel surface. If the last interaction occurs at a distance from the pixel that is comparable or smaller than the pixel size, ε_{Ω} can become very large. In order to avoid spikes in the signal on single pixels, it is useful to set a cut on ε_{Ω} . By default, this cut is set to 2π.
Two acquisition modalities are possible:

1) fluence: each channel simply counts the number of photons that it detects;
2) energy fluence: each channel sums up the energy of the photons that it detects

The energy response of the detector can eventually be taken into account by using the first modality with a sufficient number of energy bins and by a proper postprocessing of the acquisition.

detectorarray device file header:

```
Newdevice detectorarray          ; Device type
name (string)                    ; Device name
```

Commands:

SourceName *name*
name (string): source input device name;

NPixels *val1 val2*
pixel number ($N_x \times N_y$);
val1, val2 (integer values): number of columns (N_x) and rows (N_y);

PixelSize *val1 val2*
pixel size ($L_x \times L_y$, cm);
val1, val2 (real values): rectangle sides (L_x , L_y);

Shape *val*
pixel shape:
val = 0 : rectangular;
val = 1 : elliptical;

dOmegaLim *val*

cut on ε_Ω (if this entry is 0, then ε_Ω is set to the default value 2π);

val (real) ε_Ω;

X *val1 val2 val3*

val1, *val2*, *val3* (real values) detector geometric center coordinates x, y, z;

uk *val1 val2 val3*

orientation: $\mathbf{k}_d$ components (local z axis direction, i.e. normal to the detector surface);

val1 val2 val3 (real values): k_{dx}, k_{dy}, k_{dz}

ui *val1 val2 val3*

detector orientation: $\mathbf{i}_d$ components (local x axis direction);

val1 val2 val3 (real values): i_{dx}, i_{dy}, i_{dz}

ExpTime *val*

val (real value): exposure time (seconds)

PhotonNum *val*

val (integer): multiplicity of simulated events per pixel;

RandomPixelFlag *val*

val (integer): enable random point on pixels (0/1)

PoissonFlag *val*

val (integer): enable Poisson statistic on pixel counts (0/1)

RoundFlag *val*

val (integer): round pixel counts to integer (0/1)

HeaderFlag *val*

val (integer): use header in output file (0/1)

RunningFasterFlag *val*

val (integer): columns(0) or rows(1) running faster in the output file

Rotate *val1 val2 val3 val4 val5 val6 val7*

rotation of the detector around the axis passing through $\mathbf{x}_0$ and with direction $\mathbf{u}$:

val1 val2 val3 (real values): x_0, y_0, z_0 ;

val4 val5 val6 (real values): u_x, u_y, u_z ;

val7 (real) rotation angle θ (degrees);

PixelType *val*

Pixel content type:

val (integer): 0, 1, 2 or 3;
0: fluence;
1: energy fluence;
2: fluence with energy binning;
3: energy fluence with energy binning;

Only if energy binning is used, i.e. if pixel content type is 2 or 3:

Emin *val*
val (real): minimum binning energy;

Emax *val*
val (real): maximum binning energy;

NBins *val*
val (integer): number of energy bins;

SaturateEmin *val*
val (integer): saturate energies lower than Emin (0/1)

SaturateEmax *val*
val (integer): saturate energies greater than Emax (0/1)

End
End of file.

If $\mathbf{i}_d$ is not perpendicular to $\mathbf{k}_d$, then the program replace it by a vector on the same common plane perpendicular to $\mathbf{k}_d$; if $\mathbf{k}_d$ or $\mathbf{i}_d$ are not unit vectors then they are normalized by the program; the vector $\mathbf{j}_d$ is computed by the program, so that $\mathbf{i}_d$, $\mathbf{j}_d$, $\mathbf{k}_d$ are orthonormal unit vectors;

5.5 The composition file

The materials that compose the sample are called *phases*. A composition file is used to list all the phases and to characterize them by their mass density and by their elemental composition, i.e. the atomic numbers and weight fractions of the atomic species that compose them. Each phase is assumed to be homogeneous. Phases are referred to by their user-defined names. There is a predefined phase named "Vacuum" with mass density equal to zero and no elements. This is the phase that fill the "universe". If the user wants to simulate an experiment in a different medium (e.g. in air) he should first define it. However, only a finite region of space can be filled by a phase different from vacuum.

composition file header:

Newdevice composition ; Device type
name (string) ; Device name

Commands:

Phase *name*

name (string): material name; define a new material;

NElem *val*

val (integer): number of atomic species N_e in the phase;

Z_1 w_1 (integer, real): atomic number and weight percent of 1st element;

…..

Z_{Ne} w_{Ne} (integer, real): atomic number and weight percent of N^{th} element;

Rho *val*

val (real): mass density of the phase (g/cm^3);

End

End of file.

5.6 The quadric array file

The sample geometry is described through a set of quadrics, which are used to define the surfaces of solid objects. A quadric is a surface in the three-dimensional space defined as the locus of zeros of a quadratic polynomial. The general quadric is defined by the algebraic equation:

$$\sum_{i,\,j=1}^{3} x_i Q_{ij} x_j + \sum_{i=1}^{3} P_i x_i + R = 0 \quad .$$

If we define $x_0 = 1$, then the general quadric may be compactly written in vector and matrix notation as

$$x A x^T = 0$$

where $x = (x_{0,} x_{1,} x_{2,} x_3)$ is a row vector, x^T is the transpose of x (a column vector) and A is a 4×4 matrix with $A_{ij} = Q_{ij}$ for $i, j = 1,\ldots,3$, $A_{0i} = A_{i0} = P_i$ and $A_{00} = R$.
The matrix A must be symmetric, thus $A_{ij} = A_{ji} \quad \forall\, i, j = 0,\ldots,3$.
A quadric divides the space in two regions, one with $x A x^T > 0$, the other with $x A x^T < 0$.
We will call those two regions space *outside* the quadric (or *external* space) and space *inside* the quadric (or *internal* space), respectively, no matter whether the quadric is closed or not.
Whenever a unit vector normal to the quadric surface has to be defined, by default we will assume that is is oriented toward the external space. Examples of quadrics are planes, ellipsoids, cylinders.

quadricarray file header:

Newdevice quadricarray	; Device type
name (string)	; Device name

Commands:

- Quadric $A_{00}\ A_{01}\ A_{02}\ A_{03}\ A_{11}\ A_{12}\ A_{13}\ A_{22}\ A_{23}\ A_{33}$
  Generic quadric defined by its elements' contents. Since the matrix has to be symmetric, only 10 elements are used.

- Plane $x_0\ y_0\ z_0\ u_x\ u_y\ u_z$
  Plane containing the point (x_0 , y_0 , z_0) with normal vector (u_x , u_y , u_z).

- Ellipsoid $x_0\ y_0\ z_0\ R_x\ R_y\ R_z$
  Ellipsoid with principal axis parallel to the main axis, centered in (x_0 , y_0 , z_0) and with semi-axes R_x, R_y, R_z .

- CylinderX $y\ z\ R_y\ R_z$
  Cylinder having the main axis parallel to the x axis with coordinates y, z on the yz plane, and having elliptical section with semi-axes R_y, R_z .

- CylinderY $x\ z\ R_x\ R_z$
  Cylinder having the main axis parallel to the y axis with coordinates x, z on the xz plane, and having elliptical section with semi-axes R_x, R_z .

- CylinderZ $x\ y\ R_x\ R_y$
  Cylinder with the main axis parallel to the z axis with coordinates x, y on the xy plane, and having elliptical section with semi-axes R_x, R_z .

- Translate $\Delta x\ \Delta y\ \Delta z$
  Translate the position of the last defined quadric by (Δx, Δy, Δz).

- Rotate $x_0\ y_0\ z_0\ u_x\ u_y\ u_z\ \theta$
  Rotate the last defined quadric around the axis passing through the point (x_0 , y_0 , z_0) and directed as (u_x , u_y , u_z) by an angle θ (expressed in degrees).

- TranslateAll $\Delta x\ \Delta y\ \Delta z$
  Translate the position of all previously defined quadrics by (Δx, Δy, Δz).

- RotateAll $x_0\ y_0\ z_0\ u_x\ u_y\ u_z\ \theta$
  Rotate all previously defined quadrics around the axis passing through the point (x_0 , y_0 , z_0) and directed as (u_x , u_y , u_z) by an angle θ (expressed in degrees).

- End
  End of file.

5.7 The three-dimensional object geometric description file

A solid *object* is defined as a solid shape delimited by a set of quadric surfaces that separates the space inside from the space outside it. The quadrics delimiting an object must be properly oriented, in such a way that their normal vectors are directed outward with respect to the object itself.

- **Only convex objects are allowed in XRMC.** Non-convex shapes can be built by combining convex objects. The demo files show some examples of how to build non-convex shapes.
- **The surfaces delimiting different objects can not be in contact with each other.** The users have to pay particular attention to this point, because the code does not make any check on it. If the user wants two objects to be in contact with each other (for instance when building non-convex shapes by joining convex objects) a workaround is to separate them using two different quadrics, very close to each other but separated by a small gap, in such a way that the effect of the intermediate space on the radiation is negligible. The demo files show some examples of this trick.
- An object may contain other objects, or it may be contained in another object, as far as their delimiting surfaces are not in contact.

All objects must be contained in a finite region of space, called *sample region*.

geom3d device file header:

Newdevice geom3d ; Device type
name (string) ; Device name

Commands:

QArrName *name*
name (string): quadricarray input device name;

CompName *name*
name (string): composition input device name;

Object *name*
name (string): 3d object name; defines a new object;
phase-in-name
name of the phase (material) inside the object;
phase-out-name
name of the phase (material) surrounding the object;
N_q
number of quadrics that define the object surface;
quadric-name-1 quadric-name-2 ..... quadric-name-N_q
names of the quadrics that define the object surface;

End
End of file.

5.8 The sample file

When a photon exits from the source or when it is produced by a scattering/fluorescence emission process, its trajectory is characterized by its position vector $\mathbf{r}_{ph}$ and by its direction vector $\mathbf{u}_{ph}$. The program evaluates the intersection of this trajectory with the quadric surfaces delimiting the objects, and divide it in N_s steps with uniform phases. Each step is a segment of the trajectory delimited by its intersection with different objects. There are two possible

modalities of extracting the next interaction position:

1) extract the interaction position according to the interaction probability distribution along the trajectory, which is evaluated from the linear absorption coefficient of the phases and from the steplengths of the paths;
2) extract a step at random using a random integer number $0 \leq m < N_s$, Extract a random position on step m using a uniform probability distribution, and multiply the weight associated to the event by a proper factor;

The first modality is that of the traditional Monte Carlo approach. The second one is sometimes used for variance reduction. It can be useful, for instance, when an object that emits a relevant fluorescence signal is surrounded by a strongly absorbing material, so that the probability that the incident radiation reaches such object is relatively low.

sample device file header:

```
Newdevice sample                ; Device type
name (string)                   ; Device name
```

Commands:

SourceName *name*
name (string): source input device name;

Geom3DName *name*
name (string): geom3d input device name;

CompName *name*
name (string): composition input device name;

WeightedStepLength *val*
modality that will be used for the extraction of the next interaction position:
val = 0: method 1) described above;
val = 1: method 2) described above;

FluorFlag *val*
activate Fluorescence (0/1); it can be useful to deactivate it in imaging experiments where fluorescent emission is not relevant, to save computational time;

ScattOrderNum N_I
N_I (integer): maximum scattering order (0: transmission, 1: first order scattering or fluorescence emission, 2: second order scattering or fluorescence emission, …)
M_1
multiplicity of simulated events for order 0;
…..
M_{NI}
multiplicity of simulated events for order N_I ;

End

End of file.

As discussed previously, the total number of generated events is given by the product of the multiplicities in the spectrum, in the interactions with the sample and in the detector pixels.

5.9 The output file

The output file is saved in raw binary format
The detector bin contents are written in C *double* format (64 bit real). The total number of entries is:

N. of scattering orders × N. of columns × N. of rows × N. of energy bins

with the energy bins *running faster*.
If the *header flag* is set to 1 in the detector file, then the bin contents are preceded by a 60-bytes-long header, also in binary format, containing the following information:

- N. of scattering orders (C int format, 32 bit integer)
- N. of columns (C int format, 32 bit integer)
- N. of rows (C int format, 32 bit integer)
- Pixel size Sx (C double format, 64 bit real)
- Pixel size Sy (C double format, 64 bit real)
- Exposure time in sec. (C double format, 64 bit real)
- Pixel content type (C int format, 32 bit integer)
- N. of energy bins (C int format, 32 bit integer)
- Minimum bin energy (C double format, 64 bit real)
- Maximum bin energy (C double format, 64 bit real)

6 Examples

The examples can be found in the subdirectories of xrmc/examples. The examples are described by comments in the corresponding input files. To run an example, go to the corresponding subdirectory and type the command:

xrmc input.dat

At the end of the simulation, the output will be stored in the file *output.dat* .
The output of the examples 1-6 are images in raw binary format, which can be opened with any image visualization program able to import such format. For instance, the figures in this document have been produced using ImageJ, which is a public domain image processing program, freely available for several platforms. In case you want to open the output images of the examples using this program, select from the menu:

file → import → raw...

chose the image and use the following settings in the import form:

- Image type: 64-bit real;

- Width: N. of columns;
- Height: N. of rows;
- Offset to first image: 60 if the image contains the header, 0 otherwise;
- Number of images: N. of scattering orders (1 if only transmission was simulated);
- Little endian byte order: depends on the architecture of your system;

Example 7 represents the simulation of a x-ray fluorescence spectroscopy experiment. A simple program is provided with this example, in the same directory, to simulate a Silicon Drift detector response.

6.1 Cylindrical shell

Example 1: simulating a cylindrical shell using convex objects, directory *cylind_shell*.
To simulate a cylindrical shell using only convex objects, two objects have been used:

- An external full cylinder, delimited by two planes;
- An internal empty cylinder (the phase inside it is vacuum, phase index=0), delimited by other two planes very close to those of the external cylinder but separated from them by a small gap

Figure 2 shows the output image. Image size: 400 × 400

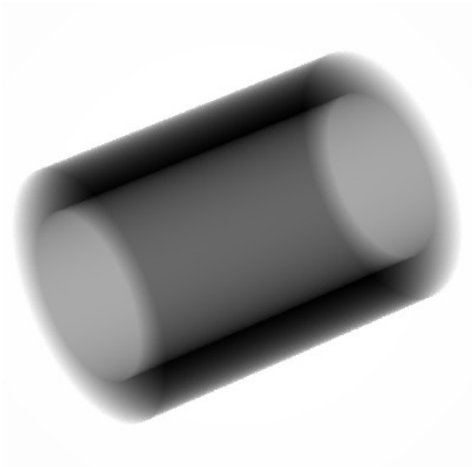

Figure 2 Example of simulation of radiographic images of a cylindrical shell This example is described by comments in the corresponding input files.

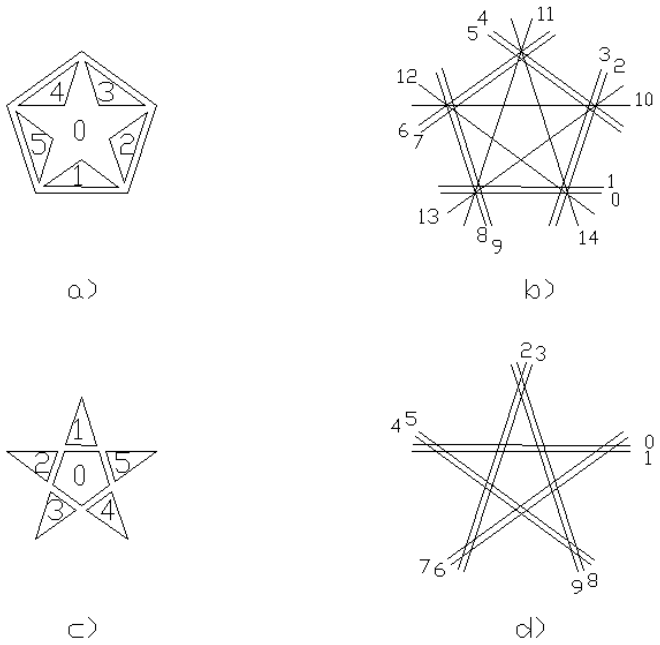

Figure 3 Schematic representations of two ways of building a star shape by combining convex objects. In the quadric definition files, the planes between adjacent objects are separated by a very small gap, however in this figure their separation is increased for clarity. **(a)** Geometrical objects used to compose the star shape described in example 2. **(b)** Planes used to define the geometrical object surfaces described in example 2. **(c)** Geometrical objects used to compose the star shape described in example 3. **(d)** Planes used to define the geometrical object surfaces described in example 3. All objects are delimited by two planes parallel to the figure plane. Those examples are described by comments in the corresponding input files.

6.2 Star shape 1

Example 2: one way of building a star shape by combining convex objects, directory *star1*.
Figure 3**(a)** shows the geometrical objects used to compose the star shape. The five triangles are separated from the inner pentagon by small gaps. The phase inside the triangles and that inside pentagon are the same. Figure 3**(b)** shows the planes used to define the geometrical object surfaces. All objects are delimited by two planes parallel to the figure plane.
Figure 4**(a)** shows the output image. Image size: 400 × 400

6.3 Star shape 2

Example 3: another way of building a star shape by combining convex objects, directory *star2*.
Figure 3**(c)** shows the geometrical objects used to compose the star shape. The five triangles are empty (i.e. the phase inside them is vacuum, phase index=0), and they are separated from the external pentagon by small gaps. Figure 3**(d)** shows the planes used to define the geometrical object surfaces. All objects are delimited by two planes parallel to the figure plane.

6.4 Different type of quadrics

Example 4: an image with different types of quadrics, directory *quadrics*.

Figure 4**(b)** shows the output image. Image size: 400 × 400

6.5 A wheel shape

Example 5: a wheel, directory *wheel*.
A wheel is built by combining different cylinders and using the rotation commands.
Figure 4**(c)** shows the output image. Image size: 400 × 400

6.6 Objects with different compositions

Example 6: cylinders with different compositions, directory *materials*.
Four cylinders having different compositions (polymethyl methacrylate, adipose tissue, water, bone-equivalent plastic) and placed inside a polymethyl-methacrylate frame are simulated.
Figure 4**(d)** shows the output image. Image size: 400 × 400

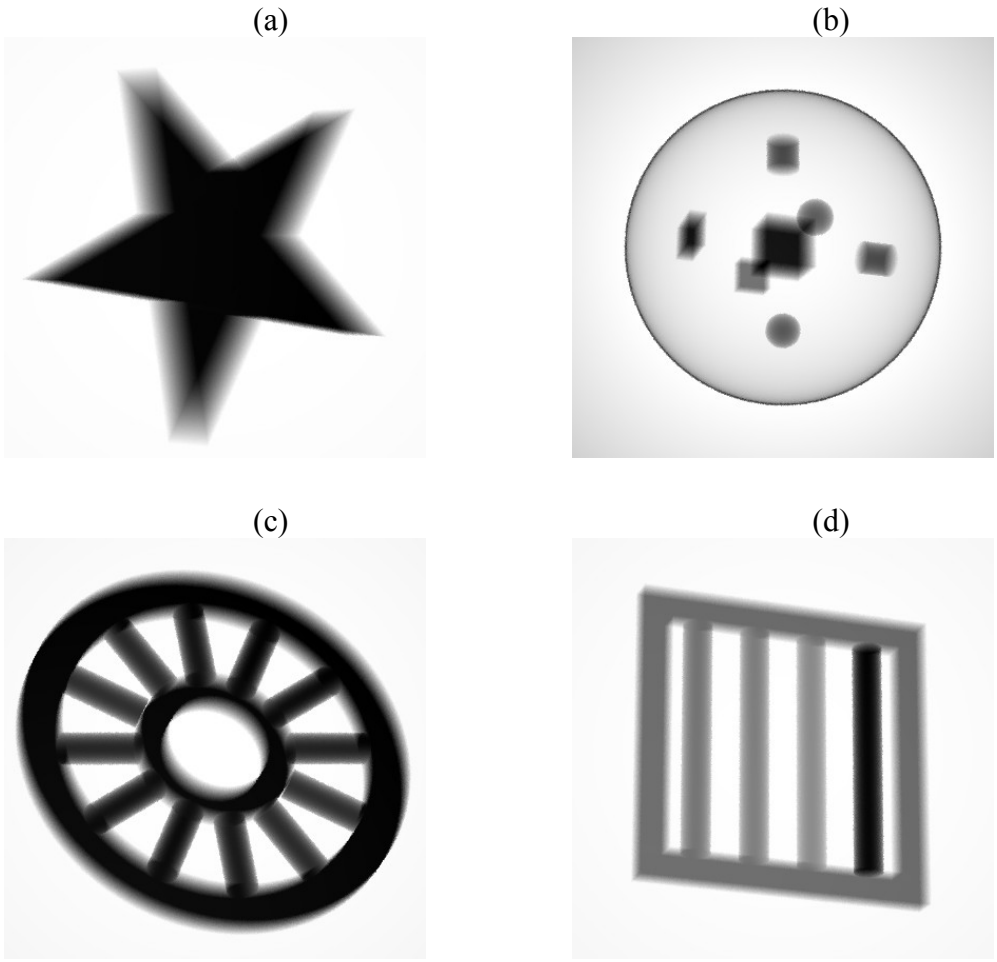

Figure 4 Example of simulation of radiographic images of different shapes: a star shape **(a)**, a thin spherical shell containing different types of quadrics **(b)**, a wheel **(c)** and four cylinders having different compositions and placed inside a frame **(d)**. All the examples are described by comments in the corresponding input files.

6.7 X-ray fluorescence spectroscopy example

Example 7: x-ray fluorescence spectroscopy example, directory *fluor_layers*. Fig. 5 represents schematically the experimental setup. The source is an Ag-anode x-ray tube set to 40 kVp and 10 μA current. The source spectrum is polychromatic and it was measured experimentally. The beam was collimated through a 1 mm wide cylindrical collimator. The detector is a 25 mm^2 diameter Silicon drift detector (X-123SDD by Amptek) 500 mm thick. The sample is a parallelepiped made of the following three layers (ordered starting from the one closest to the source):

- a 200 μm thick mixture layer incrustation simulation;
- a 300 μm thick copper layer;
- a 2.5 mm thick mixture layer, similar to the first one.

The current version of XRMC does not include the simulation of the response of real detectors. The users must take care of this point by post-processing of the simulation output. For the example just described, a separate program is provided to simulate the response of a Silicon drift detector. This program can be found in the directory *example/fluor_layers/src*, A README file explains how to compile and run it. Figure 6 shows a comparison between the measured signal and the simulated one.

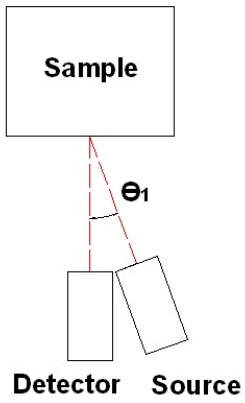

Figure 5 Schematic representation of the experimental setup for the fluorescence spectroscopy example.

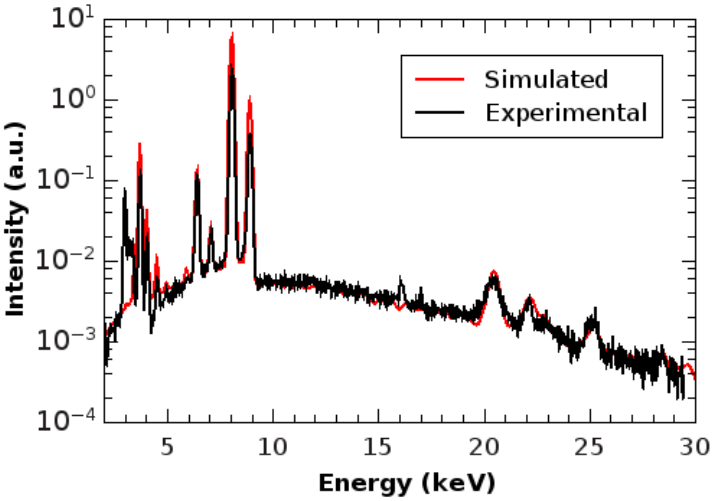

Figure 6 Comparison between simulated and measured signal for the fluorescence example.