Journal of
Synchrotron
Radiation

ISSN 0909-0495

Received 14 January 2011
Accepted 2 July 2011

SHADOW3: a new version of the synchrotron X-ray optics modelling package

Manuel Sanchez del Rio,[a]* Niccolo Canestrari,[b,a] Fan Jiang[c] and Franco Cerrina[c]†

[a]European Synchrotron Radiation Facility, 6 Jules Horowitz, 38000 Grenoble, France, [b]Institut Louis Néel, CNRS, Grenoble, France, and [c]Electrical and Computer Engineering, Boston University, 8 St Mary's Street, Boston, MA 02215, USA. E-mail: srio@esrf.eu

A new version of the popular X-ray tracing code *SHADOW* is presented. An important step has been made in restructuring the code following new computer engineering standards, ending with a modular Fortran 2003 structure and an application programming interface (API). The new code has been designed to be compatible with the original file-oriented *SHADOW* philosophy, but simplifying the compilation, installation and use. In addition, users can now become programmers using the newly designed *SHADOW3* API for creating scripts, macros and programs; being able to deal with optical system optimization, image simulation, and also low transmission calculations requiring a large number of rays ($>10^6$). Plans for future development and questions on how to accomplish them are also discussed.

Keywords: *SHADOW*; ray-tracing; X-ray optics.

1. Introduction

One step before the construction of any X-ray instrument, such as a synchrotron beamline, is the accurate conceptual design of the optics. The beam should be propagated to a given image plane (usually the sample position) and its characteristics should be adapted to the experimental requirements in terms of flux, monochromatization, focal dimensions, *etc*. The designer's goal is not only to verify compliance to a minimum set of requirements, but also to find an optimum matching between the source and the optics phase space to obtain the best possible performances.

SHADOW is a widely used program for the simulation of optical systems, specifically geared to the synchrotron radiation domain. It is based on a geometrical ray-tracing approach, but also traces field amplitude with phase difference, and therefore wave features beyond the validity domain of geometric optics. This is called the phase ray-tracing method (Lee & Zhang, 2007). While there are many optical ray-tracing programs, *SHADOW* is unique because of its special focus on synchrotron radiation. This is evident in the interface, but is also present in its core or engine where models are built to specifically address problems of synchrotron radiation beamlines such as crystal diffraction, glancing optics and photon energies in the X-ray range. In fact, the code *SHADOW* has become the *de facto* standard for synchrotron radiation ray-tracing calculations because it is flexible and capable of adapting to any beamline configuration. It has also demonstrated its reliability during more than 25 years of use, as shown in hundreds of publications. Ergo, it is relatively simple to use and well documented, and is freely available (open source).

Indeed, almost all of the synchrotron beamlines today in existence have in some way benefited from the help of *SHADOW*. In most facilities a large number of beamline optics have been designed and verified using the program, including applications in: mirror optics, from microscopes to X-ray lithography beamlines; grating monochromators, fixed- and variable-line spacing; capillary optics, crystal optics, both in reflection and transmission, and polarized sources and polarization transfer.

2. History of *SHADOW* and the birth of *SHADOW3*

The birth of *SHADOW* is linked to the first dedicated use of synchrotron radiation at the University of Wisconsin during 1965–1967. A team led by particle physicist Ednor Rowe built Tantalus. He quickly adapted the machine to make synchrotron radiation and soon the facility was crowded with experimentalists from all over the world. In 1977, Aladdin, a new and larger synchrotron radiation source, began construction. *SHADOW* was born during the Tantalus–Aladdin transition with a clear scientific motivation: Monte Carlo simulation of X-ray optical systems, in particular grating monochromator design (toroidal grating monochromator and the extended range grasshopper) and reflection by toroidal and spherical mirrors. The requirements were ambitious

† Deceased.

enough to promote *SHADOW* to a universal level independently of Aladdin's beamlines. Among these requirements one can mention: accuracy and reliability, ease of use, flexibility, economy of computer resources, VAX-11 computers, efficient Monte Carlo approach, use of reduced number of rays, exact simulation of synchrotron radiation sources, use of vector calculus for directions and operations rather than angles and trigonometry, implementation of a structured code, easy-to-use user-interface, and to be available to users. It took about two years of development to finish the first *SHADOW1* version, written in Fortran 77 with deep usage of the VAX/VMS extensions. The program was introduced in 1982, and has continued growing since. A first publication (Cerrina, 1984) explained the main philosophy of the new ray-tracing code and its application to grating monochromators. Between 1984 and 1990, *SHADOW* was exclusively used in the VAX-VMS mainframes, and the code was distributed in magnetic tapes sent by post. C. Welnak supported the increasing users community, by writing documentation and debugging, *etc.* *SHADOW* was upgraded to include new models with the help of F. Cerrina's students: B. Lai (Lai *et al.*, 1988, 1989), K. Chapman (Chapman *et al.*, 1989), G. J. Chen (Chen, Cerrina *et al.*, 1994; Chen, Guo *et al.*, 1994), S. Singh (Singh *et al.*, 1996), *etc*. Some papers on *SHADOW* and its upgrades appeared (Lai *et al.*, 1988, 1989; Welnak *et al.*, 1992, 1994). Other scientists collaborated with F. Cerrina to develop models for new optical elements, like J. Underwood for multilayers or M. Sanchez del Rio for several types of crystals (Sanchez del Rio *et al.*, 1992, 1994; Sanchez del Rio & Cerrina, 1992).

During the early 1990s the use of VAX-VMS mainframes started to decline, with the incipient use of UNIX workstations for scientific computing. *SHADOW* required a significant remodelling and restructuring to run with the new machines. M. Khan performed this conversion and *SHADOW2* was born. This version was completed by a new user interface and new graphic tools based on the *PLPLOT* libraries, replacing the TopDrawer library in VAXes. During this period *SHADOW* was essential for the development of the third-generation X-ray sources, in particular at the ESRF. A new complete and independent graphical user interface (GUI) called *ShadowVUI* was developed at the ESRF, and it was soon made available to the user community through the X-ray optics toolbox *XOP* (Sanchez del Rio & Dejus, 2004). During this time some tools for mirror roughness (Singh *et al.*, 1996) or a pre-processor for slope errors (Sanchez del Rio & Marcelli, 1992) were also developed.

At the turn of the millenium, *SHADOW* continued to give services using the *SHADOW2* kernel. Much of the graphical tools and post-processors were replaced by the ones built with the *ShadowVUI* interface. The fact that it is still widely used today is a testament to the original architecture of the program, created with extensibility and accuracy as the main goals. As the field of synchrotron radiation instrumentation advances, the requirement on the optics become more stringent. Today, third- and fourth-generation sources often use diffraction-limited optics in a broad domain of wavelengths, from the infrared to hard X-rays. There is a clear need for an optical modelling tool capable of simulating not only the behaviour of an ideal optical system but also the effect of imperfections, misalignments and other issues found in real beamlines.

Today's computers are very powerful tools, and easily surpass the old mainframes. Thus, it is important to take advantage of new possibilities for performing studies and analysis that would have been impossible on machines with lower speed and smaller memory. It was becoming more and more urgent to update *SHADOW* because: the binaries available presented problems in new machines using new libraries; the current version became heavy and hard to recompile (*e.g.* the g77 compiler used to build *SHADOW2* is no longer supported); some limitations that were inherited from the old computers (like the limitation in number of rays) became important drawbacks; and it became difficult to add new features owing to an old structure that became complex after the many upgrades over more than 20 years.

The path towards *SHADOW3* was discussed on several occasions, but no commitment was made because of the lack of manpower and funds. It is notable that *SHADOW*, together with its *ShadowVUI* interface available today, was created and developed without its own budget. It used resources for the beamline design and construction of Aladdin and other synchrotrons including the time and dedication of some of the authors through their institutions (like the ESRF). The birth of *SHADOW3* was fuelled by the ESRF Upgrade Programme 2008–2017. The design and construction of the new beamlines require a powerful ray-tracing tool, and *SHADOW* still was the most advanced X-ray tracing code for that. M. Sanchez del Rio discussed this issue in 2008 with F. Cerrina, defining an upgrade planning with clear requirements, specifications and time scheduling, that ends now with *SHADOW 3.0* .

3. Specifications and implementation

From the user point of view, the *SHADOW* package can be divided into several parts:

(i) A kernel consisting of two main programs, one for calculating the sources (`gen_source`) and a second one for tracing this source though the beamline (`trace`).

(ii) A set of pre-processors linked to an optical library (X-ray cross sections) for dealing with reflectivity and transmission of mirrors (`prerefl`), multilayers (`pre_mlayer`) and crystals (`bragg`). Another useful pre-processor is used for preparing mesh surfaces (`presurface`).

(iii) A set of utilities to visualize and analyze results files (`star.xx`, `mirr.xx`, `screen.xx` files), such as `histo1`, `plotxy`, `focnew`, *etc.*

(iv) Other utilities (graphic libraries, menu, *etc.*).

(v) Graphical user interface.

The key part of *SHADOW* is its kernel, consisting of two main programs: `gen_source` and `trace`. A minimum set of utilities were also upgraded, and others that could be replaced by external user interfaces were suppressed. The GUI, the graphics based on *PLPLOT*, and the terminal *MENU* are

no longer supported (these are very complex and system-dependent). The *SHADOW* primer, a useful manual for getting started with *SHADOW*, has been updated for *SHADOW3*.

The update of *SHADOW* was evaluated facing the possibility of rewriting everything from scratch, perhaps using a new language. However, this option was discarded because of the inability to meet the resources needed; for example, the lack of specialized man-power to write, review and test the new version.

It was decided to rebuild *SHADOW* using the new Fortran standards, to match important requirements:

(i) Back-compatibility, meaning that *SHADOW*'s users will feel 'comfortable' with the new version, and files created by the old version are accepted by the new one.

(ii) Solve important limitations of the old versions, such as the limitation in number of rays.

(iii) Flexibility: adding and modifying features must be easier, helped by a simple compilation mechanism.

(iv) Interoperability: *SHADOW* should be callable from an API. The advanced user or programmer can easily modify the main code and create *ad hoc* codes. The API also will allow the possible use of different GUIs, implementation of new ones and also the integration of *SHADOW* into other packages.

The structured code of *SHADOW2*, split into many Fortran functions and subroutines, allows the re-use and reorganization of most routines and code parts. In addition, the original architecture of *SHADOW* is well suited to the new modular programming typical of modern languages. The Fortran 77 common blocks used massively in *SHADOW2* have been completely removed, and replaced by global variables within Fortran 90 modules, in accordance with modern programming recommendations. Changes and improvements are still needed on the computational side, like the encapsulation of variables in Fortran types (*i.e.* structures or packs of variables) or to make standard the use of `implicit none`, a good programming practice, which was not followed in *SHADOW2*. The new changes will be implemented gradually, maintaining a balance between user support and new development.

4. The new *SHADOW3* code structure

4.1. Modules and variables

The new *SHADOW* source uses the modular approach made available by Fortran 90. This was a major Fortran upgrade from Fortran 77. Since then, Fortran 95 was a minor release, and Fortran 2003 (and its Fortran 2008 update) introduced new concepts in object-oriented programming not yet exploited in *SHADOW*. The *SHADOW* functions and subroutines have been distributed in a few modules. The *SHADOW* kernel consists of:

(i) `shadow_globaldefinitions`: basic definitions used everywhere, like variable kind, *etc*.

(ii) `stringio`: some string manipulation tools.

(iii) `gfile`: a new Fortran module to manipulate files with list of input/output variables (called *g-files* in *SHADOW*).

(iv) `shadow_beamio`: routines to access to binary beam files (`start.xx`, `mirr.xx`, `screen.xxyy`).

(v) `shadow_math`: mathematical tools.

(vi) `shadow_variables`: definition of variables and Fortran types used by the kernel, plus the routines to manipulate them.

(vii) `shadow_kernel`: contains the global variables (ex-common blocks) and the routines in the *SHADOW* kernel.

In addition to the kernel, *SHADOW3* contains the following:

(i) `shadow_synchrotron`: synchrotron sources (bending magnets, wiggler and undulators).

(ii) `shadow_preprocessors`: pre-processors, like `prefefl` or `bragg`.

(iii) `shadow_pre_sync`: pre-processors for `shadow_synchrotron`.

(iv) `shadow_postprocessors`: post-processors, like `histo1` or `ffresnel`.

(v) Main programs: all *SHADOW3* is included into a single command line executable: `shadow3`. All pre- and post-processors are included in this executable. In addition, `gen_source` and `trace` are also created for being 100% compatible with the old versions.

The modules map is schematized in Fig. 1. In the old *SHADOW*, the variables were imported from the `start.xx` files. For that, *SHADOW1* used `NAMELIST`, a non-standard Fortran 77 extension, whereas *SHADOW2* used a set of C routines linked to Fortran. Once imported, the variables were sent directly into common blocks. In *SHADOW3*, the variables defined in the `start.xx` files are now put into three levels: (i) from the files, they are read into a `gfile` Fortran type which is just a table of variable names and values, (ii) these values are then copied to two Fortran types, called `pool` (`poolSource` and `poolOE`), which are binded to C, and (iii) the pool variables are copied to global variables in `shadow_kernel`. This three-level scheme may appear complex, but it was considered necessary to gain in modularity and portability, and to be able to define an API. In the future, the copy of the pool variables into global variables in `shadow_kernel` will be removed, thus directly using the pool variables in the computation.

In the old version, *SHADOW* sources (`gen_source`) dealt with all kinds of sources (geometrical, bending magnet, wiggler and undulator). The new version contain three separated identities in the code:

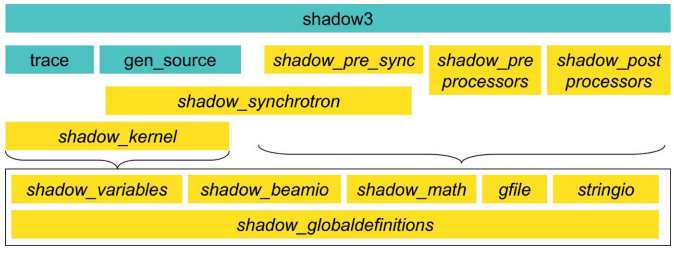

Figure 1
Module dependencies in *SHADOW3*. Modules are in light grey boxes (yellow online) and main programs in dark grey boxes (green online).

(i) `sourceGeom`: a routine in the `shadow_kernel` module used to generate geometrical sources.

(ii) `shadow_synchrotron`: an independent Fortran module for bending-magnet and insertion devices sources. It has been separated from the kernel. The synchrotron pre-processors are in a new module `shadow_pre_sync`. The old `make_id` script is now fully implemented in the `shadow3` executable.

(iii) `source_cdf`: a new generic source. It reads the characteristics from external files, *e.g.* produced by external programs [*SRW* (Chubar & Elleaume, 1998), *SPECTRA* (Tanaka & Kitamura, 2001), *XOP* (Sanchez del Rio & Dejus, 2004), *etc.*] and sample rays using these distributions. This is still under development and will be available in *SHADOW3.1*.

SHADOW has an optical library populated with information on cross sections and scattering factors which is used to obtain the material properties, refraction index and crystal structure factors. The *SHADOW* kernel does not directly call the optical library, but only reads material data files filled with physical constants that were created using pre-processors linked to the optical library. Thus, the optical library is external to the kernel, simplifying the scheme. The old optical library is still used, but new libraries with updated values are also provided. In addition, the pre-processors can also be built using other well established optical libraries, such as `xraylib` (Brunetti *et al.*, 2004). The two principal graphic post-processors, `plotxy` and `histo1`, have been modified to produce gnuplot graphics (see http://www.gnuplot.info/), a powerful free and multiplatform graphics package.

4.2. The *SHADOW3* API

Ray-tracing calculations can sometimes be tedious owing to the high number of beamline parameters and combinations to analyze. The optimization of a single parameter can be done manually or automatically by sampling the parameter and performing a loop of *SHADOW* runs. However, to simultaneously optimize several parameters, it is obvious that an exhaustive search of the best value is impractical so the only solution, at present, is the heuristic intuition of the developer. An automation of such a global search of optimal parameters can be done using genetic algorithms, simulating annealing or any other global optimization technique. An API is needed for such techniques, and for allowing the programmer to create loops, macros and scripts with *SHADOW*.

SHADOW3 has been first interfaced to C, with not only the idea of writing new programs in C, but also as an intermediate step to using other higher-level languages, such as Python (http://www.python.org/) and IDL (http://www.ittvis.com/). The memory allocation of the beam (a collection of rays) is an important issue of the API. This must be done at the main level (Fortran, C, *etc.*). The API module imports a reduced number of procedures from Fortran, all of them considered functions in C (the *exposed* functions). Although it is technically possible to directly use the Fortran functions from C, it creates problems for a C programmer, because in Fortran all the arguments are passed by reference and the definition of strings is not compatible. Therefore, a solution has been chosen where only a set of C-functions (named with the prefix `CShadow`) and C-structures access the Fortran procedures and types. These C-functions are documented, and their names mimic the Fortran counterparts. This C-API also puts the basis for a C++ layer that utilizes the C-structures and the exposed functions to define classes. *Source*, *Optical-element* and *Beam* are classes in the C++ layer. The C-API is also the ground level for developing Python classes. Python comes with an advantage: it is a script language, thus the user has direct access to the *SHADOW* kernel permitting any batch or macro programming without any compilation. *SHADOW3* has also been interfaced to IDL, because the *ShadowVUI* GUI is written in IDL. Fig. 2 shows an example of a simple main code to run a source and a single optical element written in Fortran, C, C++, Python and IDL.

4.3. GUIs

SHADOW3 is delivered without a GUI. The old TCL/TK *SHADOW* GUI shipped with previous versions of *SHADOW* is now obsolete and has been discontinued. Work is invested in improving and updating *ShadowVUI*, an IDL-based GUI that is available free to users as part of the *XOP* package. *ShadowVUI* has been adapted to *SHADOW3* and can be configured to either select *SHADOW3* or the previous *SHADOW2*. The *ShadowVUI* also has powerful scripting capabilities, using script commands based on IDL and calling directly the IDL functions used in *ShadowVUI*. See Fig. 3 for an example. The IDL scripting will slowly migrate to the more powerful Python scripting.

4.4. *SHADOW3* distribution

Two servers have been set for *SHADOW3*. The first one, http://ftp.esrf.eu/pub/scisoft/shadow3/, is used for downloading the *SHADOW3* binaries compiled for different platforms (Windows, MacOS and Linux) and related documentation. It is intended for use by users only interested in running *SHADOW*. The *SHADOW* source code is distributed and managed using the version control system *git* (http://www.git-scm.com/). Sources can be downloaded by any user using the command: 'git clone git://git.epn-campus.eu/repositories/shadow3/'. The http://forge.epn-campus.eu/projects/shadow3/ server contains the complete history and full revision tracking of *SHADOW3* plus other services like an issue or bug tracker and a developer forum (wiki). Any user can see it but, for entering new issues and uploading new code, the developers must apply for an account. We propose that any registered developer could upload new applications, macros, post-processors, *etc.*, but modifications in the *SHADOW* kernel will be subject to the acceptance of the manager. However, *git* is flexible enough to allow any registered developer to create a new code *branch* in the server, that can be incorporated in the *master branch* by the manager.

FORTRAN

```
PROGRAM example01
use shadow_kind
use beamio
use shadow_variables
use shadow_kernel
use shadow_synchrotron

implicit none
type (poolSource)   :: src
type (poolOE)       :: oe1
real(kind=skr), allocatable, dimension(:,:) :: ray
integer(kind=ski)   :: ierr

! load variables from start.00
CALL PoolSourceLoad(src,"start.00")
print *,'Number of rays: ',src%npoint
src%npoint=100000
print *,'Number of rays (modified): ',src%npoint


! allocate ray
ALLOCATE( ray(18,src%npoint) )

! calculate source
CALL SourceSync(src,ray,src%npoint)
call beamWrite(ray,ierr,18,src%npoint,"begin.dat")

! reads start.01 into oe1
call PoolOELoad(oe1,"start.01")

! traces OE1
call TraceOE(oe1,ray,src%npoint,1)

! write file star.01
CALL beamWrite(ray,ierr,18,src%npoint,"star.01")
DEALLOCATE(ray)

END PROGRAM example01
```

C

```
#include <stdio.h>
#include <string.h>
#include <shadow_bind_c.h>

int main()
{
  poolSource src;
  poolOE     oe1;
  double    *ray=NULL;



  // load variables from start.00
  CShadowPoolSourceLoad(&src, "start.00");

  printf(" Number of rays: %d\n", src.NPOINT);
  src.NPOINT=100000;
  printf(" Number of rays (modified): %d\n", src.NPOINT);

  // allocate ray
  ray = CShadowAllocateBeamFromPool(&src,ray);

  // calculate source
  CShadowSourceSync(&src, ray);
  CShadowBeamWrite(18,src.NPOINT,ray,"begin.dat");

  // reads start.01 into oe1
  CShadowPoolOELoad(&oe1,"start.01");

  // traces OE1
  CShadowTraceOE(&oe1,ray,src.NPOINT,1);

  // write file star.01
  CShadowBeamWrite(18,src.NPOINT,ray,"star.01");
  free(ray);

  return EXIT_SUCCESS;
}
```

C++

```
#include <iostream>
#include <shadow_bind_cpp.hpp>

using namespace std;

int main()
{
  Source src;
  OE     oe1;
  Beam   ray;

  // load variables from start.00
  src.load( (char*) "start.00");

  cout << " Number of rays: "
       << src.NPOINT
       << endl;
  src.NPOINT=100000;
  cout << " Number of rays (modified): "
       << src.NPOINT
       << endl;

  // calculate source
  ray.genSource(&src);
  ray.write( (char*) "begin.dat");


  // reads start.01 into oe1
  oe1.load( (char*) "start.01");
  // traces OE1
  ray.traceOE(&oe1,1);


  // write file star.01
  ray.write( (char*) "star.01");

  return EXIT_SUCCESS;
}
```

Python

```
import Shadow as sp
import numpy as np

src = sp.Source()
oe1 = sp.OE()
ray = sp.Beam()


# load variables from start.00
src.load("start.00")

print " Number of rays: ", src.NPOINT
src.NPOINT=100000
print " Number of rays (modified): " , &
    src.NPOINT


# calculate source
ray.genSource(src)

ray.write("begin.dat")

# reads start.01 into oe1
oe1.load("start.01")
# traces OE1
ray.traceOE(oe1);

# write file star.01
ray.write("star.01")

del src
del oe1
del ray
```

IDL

```
; load variables from start.00
src=read_gfile("start.00")

print,'Number of rays: ',src.npoint
src.npoint=10000
print,'Number of rays (modified): ',src.n


; calculate source and writes begin.dat
write_gfile,pool00,file
xsh_run,'gen_source start.00'


; reads start.01 into oe1
oe1 = read_gfile("start.01")

; traces OE1 (via files)
xsh_run,'echo 0 | trace -m menu'



END
```

Figure 2
Example of a simple *SHADOW3* main program in Fortran, C, C++, Python and IDL.

5. Examples

The main objective of *SHADOW3* is to provide the full functionality of previous versions, so there is no new development from the point of view of models and algorithms. The installation is made much more simplified by the use of a single executable `shadow3` that allows calling the main commands (`source` and `trace`) plus the pre- and post-processors in a simple and integrated environment.

The *SHADOW* primer, available to users since 1994, explained in detail all aspects of the code that the user needed for setting the calculations. It contains the basic explanations

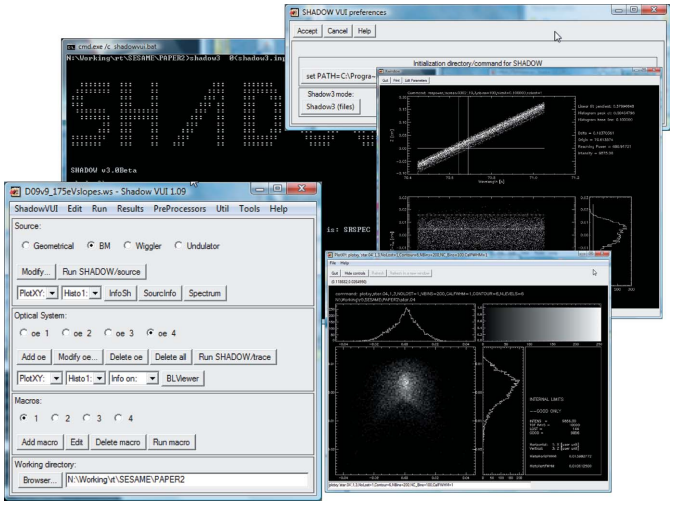

Figure 3
Window of *ShadowVUI* running *SHADOW3*.

of the *SHADOW* reference system, and explains in great detail how to run in command mode using several important examples, like the creation of geometrical and synchrotron (bending-magnet) sources, mirror focusing, and grating and crystal monochromators. We have updated the *SHADOW* primer to *SHADOW3*, including the new features, mainly the graphics output now based in `gnuplot`. Moreover, all the examples are also provided as text input files, thus it is very easy to rerun the examples using these files as standard input for *SHADOW3*. In addition, a script is available to run all the examples in the primer from a single command. The new primer and input files are available from the *SHADOW3* repository mentioned before. They also constitute a collection of tests to check back-compatibility of future upgrades.

Good practice when performing Monte Carlo simulations is to complement the calculation of a given parameter P with error estimation. Errors can be estimated in two ways. The first uses the fact that the Monte Carlo estimate is asymptotically normally distributed (approaches a Gaussian density) (James, 1980). Therefore, if one performs M *SHADOW* runs with N rays per run, one obtains M independent results for P (p_i, $i = 1, \ldots, M$) that can be used for computing the average $\bar{p}$ and its standard deviation, a good estimator of the error. It is also possible to calculate the value and the error of P from a single *SHADOW* run of N rays. Following James (1980), in order to calculate the average and the standard distribution of a scored parameter P it is necessary to accumulate: the sum of the weights w of the scored rays, $p = \sum w_i$, where the sum extends to all scored rays; the sum of the squares of the weights, $q = \sum w_i^2$; and the total number of rays, N. The final total counting for the parameter P (plus and minus one standard deviation) is

$$P = p \pm \left[q - \left(p^2/N\right)\right]^{1/2}. \quad (1)$$

Consistently, in the case that one scores non-weighted rays (*i.e.* 'reflectivity' is not set in *SHADOW*, in other words, we are 'counting rays'), every ray has the same weight equal to 1, so $q = p$ and, for $N \rightarrow \infty$, one obtains $P = p \pm p^{1/2}$, an expression typically used with counter detectors (Poisson statistics). The error estimation based on this idea has been implemented in post-processors like `histo1` and `intens`.

Both ways of computing errors are easily applied to *SHADOW3*, that can now trace any number of rays in a single run, limited only by the computer memory. In many cases it is prefereable to perform M runs with N rays and accumulate the results, rather than performing a single run with $M \times N$ rays. This is more efficient from the computer's point of view, as not all the rays are stored in the memory at the same time (and perhaps in a large file) but only a small number of rays are traced at once, then the required results are scored, before returning and processing the next run that will calculate another bunch of rays and accumulate the results. Fig. 4 shows an example where *SHADOW* runs in a loop and accumulates results in a histogram. The loop continues until a 'quality' criterium is reached. In this case, that error (standard deviation) in intensity at the centre of the histogram must be less than 2%. This program runs very fast, as no files are written. Obviously, more runs are needed if the number of bins of the histogram is increased.

The power of the *SHADOW3* API also opens the door to combine *SHADOW3* with other large simulation packages. For instance, the combination of *SHADOW3* with the Bmad library (http://www.lepp.cornell.edu/~dcs/bmad) for relativistic charged-particle dynamics simulations in storage rings is being considered. The *SHADOW3* code for accurate simulation of the wiggler source has been used together with the *PENELOPE* code (Salvat *et al.*, 2008). This code performs Monte Carlo transport of electrons and photons in matter, which is of great interest to medical physics for dosimetry calculations. Experiment planning of synchrotron microbeam radiation therapy requires an accurate description of the X-ray source phase space. *SHADOW3* is used for creating an

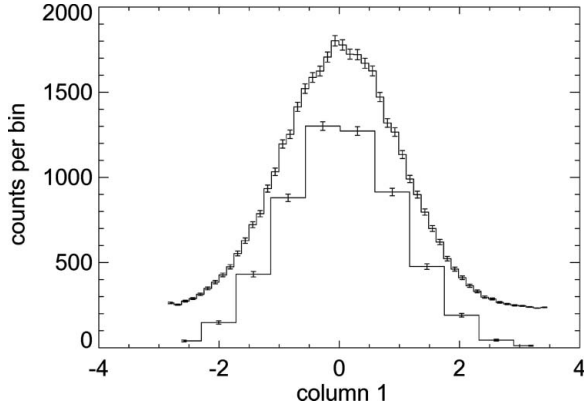

Figure 4
Example of a *SHADOW* loop to accumulate counts into a histogram (see code in Appendix *A*). The two graphs are the results for the system defined in the *SHADOW* primer (ch. 6.3) with different histogram resolution, 11 and 51 bins, that needed 10000 and 53000 rays, respectively, for reducing the standard deviation to less than 2%. The histogram with 51 bins is shifted for clarity.

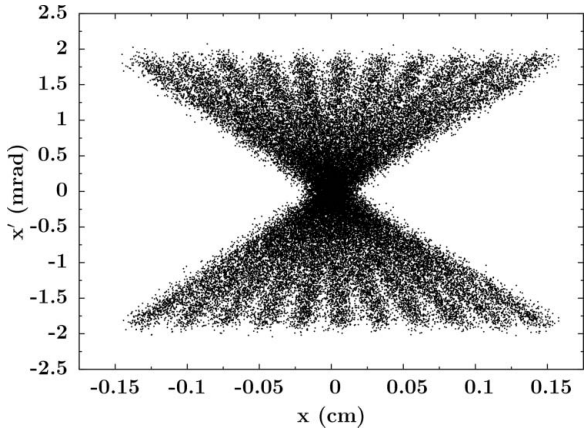

Figure 5
Plot of the horizontal phase space for a wiggler (ID17 at the ESRF) with 11 periods of 0.15 m length, K = 22.3 and electron beam energy of 6.04 GeV.

exact description of the wiggler source, which has a complex geometry (see Fig. 5) not easily implemented by geometric models. The simulated photons (*i.e.* the rays created by *SHADOW3*) are then input to *PENELOPE* for calculating the deposited energy dose in a phantom or tissue.

SHADOW can perform wave optics propagation of the beam from an image plane to a detector plane, using the `ffresnel` post-processor. It requires the propagation of every ray to each point in the detector plane by applying the Fresnel–Kirchhoff integral. In old versions, this was usually done for computing one-dimensional intensity profiles. With the new version, it is possible to create full two-dimensional interference/diffraction patterns with the new `ffresnel2d` routine. An example is shown in Fig. 6.

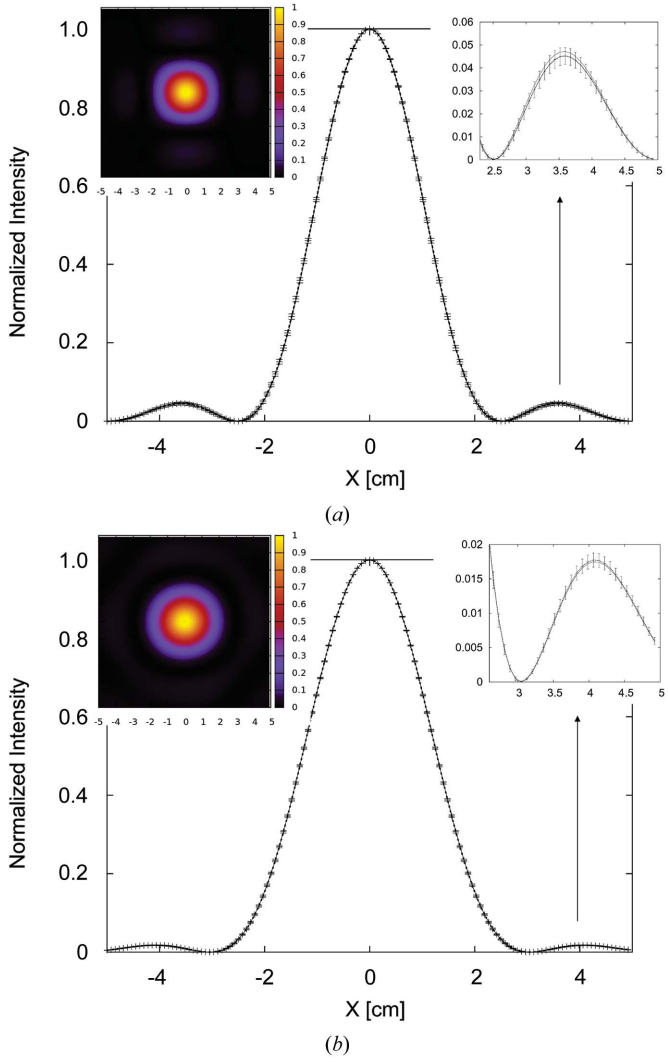

Figure 6
Fraunhoffer diffraction pattern of a square aperture (*a*) and circular aperture (*b*) as calculated by the `ffresnel2d` post-processor. A coherent Monte Carlo source of 1024 rays of 1 nm wavelength was sent to illuminate the 4 µm-wide slit. The detector is placed 100 m downstream from the slit. The image on the detector is sampled by 128 × 128 pixels. The *SHADOW* results of averaging 64 runs (dotted line), from which standard deviations are calculated, have been compared with the analytic theoretical pattern (solid line) with excellent agreement.

6. The future of *SHADOW*

The *SHADOW* code has always been tied to the bright and strong personality of its main author, Professor Franco Cerrina, who passed away on 12 July 2010. The work presented here is just a first step in an ambitious plan of continued maintenance and development that has accompanied the evolution of the synchrotron radiation facilities in the last 25 years. *SHADOW3* is a first step in cleaning the kernel and preparing a programming platform that will face exciting new developments, such as: (i) propagation of coherent and partially coherent X-ray beams, *e.g.* for interference and phase contrast applications; (ii) simulation of samples for calculating instrumental functions, and to be used as sources for ray-tracing analysers; (iii) evolving from a macroscopic model for dealing with the optical elements, to a microscopic model, where a full Monte Carlo analysis is performed within the optical elements; (iv) intensive calculations with a high number of rays using variance reduction techniques and global optimization methods. There is also the urgent need for some applications; for instance, the full implementation of compound refractive lenses, ability to treat any crystal structure, introduction of roughness in multilayers, implementation of laterally graded multilayers, beam penetration simulation inside diffractive elements (multilayers and crystals), computation of the grating efficiency, *etc*. The development of *SHADOW* to match these challenging developments is uncertain, as now we lack the guidance and direction of Franco Cerrina. We firmly believe that the survival of *SHADOW* and its future development cannot proceed without the implication and engagement of the synchrotron laboratories. *SHADOW* is a unique software created and targeted to synchrotron optics, and it should be the responsibility (and the interest) of the synchrotron laboratories to guarantee a future that will certainly benefit them. In the future we will discuss and organize plans for its continuation and development, merging efforts from many synchrotron laboratories.

APPENDIX *A*

The code used for Fig. 4 is shown below in Fig. 7.

```
PROGRAM example02

! similar to example01 (i.e., reads source and oe 1 pars from
! start files) but runs into a loop and accumulates the histogram
use shadow_globaldefinitions
use shadow_beamio
use shadow_variables
use shadow_kernel
use shadow_synchrotron
use shadow_postprocessors

implicit none
type (poolSource)                          :: src,srcTmp
type (poolOE)                              :: oe1,oe1Tmp
real(kind=skr), allocatable, dimension(:,:):: ray
integer(kind=ski)                          :: ierr,i,j,nMaxRuns,nRays

! parameters for histo1
real(kind=skr), allocatable, dimension(:)  :: xArray,yArray,y2Array
real(kind=skr)                             :: center,width
integer(kind=ski)                          :: nBins, col
integer(kind=ski)                          :: ilost,inorm,irefl,iener
real(kind=skr)                             :: invn,sigma,pc


! inputs
nMaxRuns=200 ! max number of runs
nRays=1000

! inputs histo1
nBins=11
center = 0.0D0
width  = 0.0
col=1   ! col to analyze
iener=0 ! if col=11, units: 0=A, 1=eV, 2=cm-1
ilost=0 ! lost ray flag 0=Good, 1=All, 2=LostOnly
inorm=0 ! normalize to: 0=None, 1=MaxHisto, 2=Area
irefl=1 ! if 1, weigth rays with reflectivity A**2

! allocate and initialize histo1 output
ALLOCATE( xArray(nBins),yArray(nBins),y2Array(nBins) )
xArray=0.0D0
yArray=0.0D0
y2Array=0.0D0

! load variables from start.00 and start.01
CALL PoolSourceLoad(src,"start.00")
CALL PoolOELoad(oe1,"start.01")

src%npoint=nRays ! redefine the number of rays to be used
oe1%fwrite=3     ! avoid to write binary files mirr.01 star.01 and info
                 ! files effic.01 optax.01


! allocate and initialize ray
ALLOCATE( ray(18,src%npoint) )
ray = 0.0D0

! SHADOW loop
DO i=1,nMaxRuns

  print *,'>>> Running iteration ',i,' out of ',nMaxRuns

     ! calculate source
     srcTmp = src ! copy type for further use (SourceSyc modifies it!)
     CALL SourceSync(srcTmp,ray,src%npoint)
     ! traces OE1
     oe1Tmp = oe1
     call TraceOE(oe1Tmp,ray,src%npoint,1)

     !compute histogram
     !
     ! note that center and width are set to zero for i=1, so they
     ! are computed and stored for i>1
     ! Also, yarray and y2Array are zero for i=1, but for i>1 it
accumulates the counts.

     call histo1_calc_easy( &
                      ! arguments (mandatory)
                      ray,col,nBins,xarray,yarray,y2Array, &
                      ! keyword optional arguments
                      center=center,width=width,   &
                      iener=iener,ilost=ilost,inorm=inorm,irefl=irefl)
     !print           *,'AFTER      center,width,counts,      squared-counts:
',center,width,yarray(nBins/2),y2Array(nBins/2)

     ! exit if error at central bin is less that 2%
     invn = 1.0d0/(src%npoint*i)
     sigma = sqrt(y2Array(nBins/2) - (yarray(nBins/2)**2)*invn )
     pc = 100*sigma/yarray(nBins/2)
     print *,'>>> run, sigma, %error: ',i,sigma,pc
     IF (pc .LE. 2) EXIT
   END DO

   ! show results
   print *,'Maximum number of runs: ',nMaxRuns
   print *,'Number of rays per run: ',src%npoint
   print *,'Total rays: ',1.0d0/invn
   print *,'Resulting histogram:'
   print *,' j   xarray    yarray   sigma  100*sigma/yarray '
   do j=1,nBins
     sigma = sqrt(y2Array(j) - (yarray(j)**2)*invn )
     pc = 100*sigma/yarray(j)
     print *,' ',j,xarray(j),yarray(j),sigma,100*sigma/yarray(i)
   end do

   ! clean and end
   DEALLOCATE(ray,xArray,yArray,y2Array)
   STOP
   END PROGRAM example02
```

Figure 7
The code used for Fig. 4.

This project has been partially funded by the European Union FP7-Infrastructures LABSYNC (grant number 213126). Thanks to Imma Martinez-Rovira for the work interfacing *SHADOW3* to *PENELOPE* and preparing Fig. 5. We fully acknowledge the continuous support of many people who encouraged us to complete this work and supported *SHADOW* after Franco's passing. This work is dedicated to Franco's memory. One photograph of him is shown in Fig. 8.

References

Brunetti, A., Sanchez del Rio, M., Golosio, B., Simionovici, A. & Somogyi, A. (2004). *Spectrochim. Acta B*, **59**, 1725–1731.
Cerrina, F. (1984). *Proc. SPIE*, **503**, 68–77.
Chapman, K., Lai, B., Cerrina, F. & Viccaro, J. (1989). *Nucl. Instrum. Methods Phys. Res. A*, **283**, 88–99.

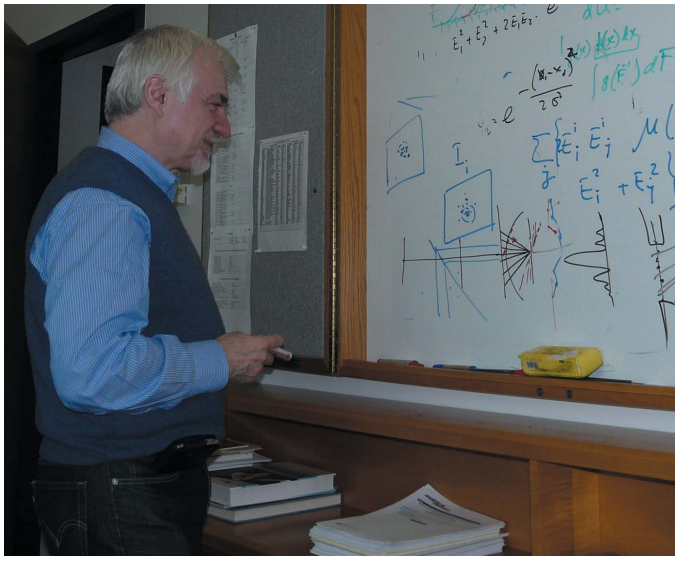

Figure 8
Franco Cerrina looking at new algorithms to be implemented in *SHADOW*. Photograph taken by M. Sanchez del Rio on 17 March 2009.

Chen, G. J., Cerrina, F., Voss, K. F., Kim, K. H. & Brown, F. C. (1994). *Nucl. Instrum. Methods Phys. Res. A*, **347**, 407–411.

Chen, G. J., Guo, J. Z. Y. & Cerrina, F. (1994). *Nucl. Instrum. Methods Phys. Res. A*, **347**, 238–243.

Chubar, O. & Elleaume, P. (1998). *Proceedings of the Sixth European Particle Accelerator Conference (EPAC98)*, Stockholm, Sweden, 22–26 June 1998, pp. 1177–1179.

James, F. (1980). *Rep. Prog. Phys.* **43**, 1145.

Lai, B., Chapman, K. & Cerrina, F. (1988). *Nucl. Instrum. Methods Phys. Res. A*, **266**, 544–549.

Lai, B., Chapman, K., Runkle, P. & Cerrina, F. (1989). *Rev. Sci. Instrum.* **60**, 2127.

Lee, H. J. & Zhang, Z. (2007). *J. Thermophys. Heat Transfer*, **21**, 330–336.

Salvat, F., Fernandez-Varea, J. M. & Sempau, J. (2008). *PENELOPE-2008. A Code System for Monte Carlo Simulation of Electron and Photon Transport.* Organisation for Economic Co-operation and Development NEA#06416 (http://www.oecd-nea.org/science/pubs/2009/nea6416-penelope.pdf).

Sanchez del Rio, M., Bernstorff, S., Savoia, A. & Cerrina, F. (1992). *Rev. Sci. Instrum.* **63**, 932–935.

Sanchez del Rio, M. & Cerrina, F. (1992). *Rev. Sci. Instrum.* **63**, 936–940.

Sanchez del Rio, M. & Dejus, R. J. (2004). *AIP Conf. Proc.* **705**, 784.

Sanchez del Rio, M., Ferrero, C., Chen, G. J. & Cerrina, F. (1994). *Nucl. Instrum. Methods Phys. Res. A*, **347**, 338–343.

Sanchez del Rio, M. & Marcelli, A. (1992). *Nucl. Instrum. Methods Phys. Res. A*, **319**, 170–177.

Singh, S., Solak, H. & Cerrina, F. (1996). *Rev. Sci. Instrum.* **67**, 3355.

Tanaka, T. & Kitamura, H. (2001). *J. Synchrotron Rad.* **8**, 1221–1228.

Welnak, C., Anderson, P., Khan, M., Singh, S. & Cerrina, F. (1992). *Rev. Sci. Instrum.* **63**, 865–868.

Welnak, C., Chen, G. J. & Cerrina, F. (1994). *Nucl. Instrum. Methods Phys. Res. A*, **347**, 344–347.