Astronomical Data Analysis Software and Systems VI
ASP Conference Series, Vol. 125, 1997
Gareth Hunt and H. E. Payne, eds.

Modeling AXAF Obstructions with the Generalized Aperture Program.

D. Nguyen, T. Gaetz, D. Jerius, and I. Stern

Smithsonian Astrophysical Observatory, Cambridge, MA 02138

Abstract. The generalized aperture program is designed to simulate the effects on the incident photon stream of physical obstructions, such as thermal baffles and pre- and post-collimators. It can handle a wide variety of aperture shapes, and has provisions to allow alterations of the photons by the apertures. The philosophy behind the aperture program is that a geometrically complicated aperture may be modeled by a combination of geometrically simpler apertures. This is done by incorporating a language, *lua*, to lay out the apertures. User provided call-back functions enable the modeling of the interactions of the incident photon with the apertures.

This approach allows for maximum flexibility, since the geometry and interactions of obstructions can be specified by the user at run time.

1. Introduction

The Advanced X-ray Astrophysics Facility (AXAF), due to be launched in 1998, is the third of NASA's four Great Space Observatories. AXAF will provide unprecedented capabilities of high resolution imaging and spectroscopy over the X-ray range of 0.1–10 keV.

As part of our efforts to support the AXAF program, the SAO AXAF Mission Support Team has developed a software suite to simulate AXAF images generated by the flight mirror assembly. One of the tasks of this system is to simulate the physical obstructions in front of and behind the AXAF mirrors.

The generalized *aperture* program is designed to simulate the effects on the incident photon stream of apertures in physical obstructions, such as the X-ray and thermal baffles. It can handle a wide variety of aperture shapes, and has provisions to allow alteration of the photons by the apertures. Apertures can simply pass, block, redirect, modify, or generate new (e.g., fluorescent) photons.

2. The Generalized Aperture Description

The conventional approach to simulating physical obstructions is to assign absolute positions and orientations to each aperture. This approach would be tedious, error prone, and inflexible to changes in design later on.

The modeling of apertures as combinations of sub-apertures, and recognizing that the bookkeeping involved in tracking the photons through the apertures is independent of any particular type of aperture, led structuring the program into three components: the *front end*, *central engine*, and the *back end*.

The *front end*, which parses the description of the openings and generates lists of apertures against which to check the photons; the *central engine*, which reads the photons and checks them against the apertures in the lists, and the *back end*, composed of modules which model each of the apertures, are called upon by the front end and central engine.

The three sections of the program are encapsulated so that a minimum of information is exchanged between them, permitting integration of new modules into the back end without affecting the rest of the program.

2.1. The Front End

The user interface The *aperture* program's user interface is a programming language, *lua*. Embedding an interpreted language such as *lua* in the front end of the *aperture* program enables the application program *aperture* to be programmable at run time. The *lua* language is a small language with flow control (*if..then, while..do..end*, etc.), functions, floating point arithmetic, multi-dimensional arrays and structures, and implicit dynamical memory allocation.

Aperture Positions and Orientations The *aperture* program uses the concept of a local coordinate system in which each aperture is placed. It keeps track of the transformations required to map between the external coordinate system in which the input photon positions and directions are specified, and the local coordinate systems of the apertures. It provides for the hierarchical layering of coordinate systems:

- A *global* coordinate system, relative to which assemblies are specified.
- An *assembly* coordinate systems, relative to which either sub-assemblies or apertures are specified.
- A *sub-assembly* coordinates systems, relative to which either nested sub-assemblies or apertures are specified.

At each level, changes to the current coordinate system do not affect the higher level coordinate systems.

2.2. The Central Engine

The generalized aperture program takes a list of assemblies and apertures created by the front end, and compares each input photon to the apertures, in turn. It calls functions provided by the back end modules, which do the actual checking of the photons. It provides the logic to move the photon to the next assembly, should an aperture accept a photon and pass it along.

2.3. The Back End

Each back end module consists of two components. The first is called by the user's aperture definition script, and creates an instance of an aperture with a given set of aperture specific parameters, attaching it to the list of apertures for the current assembly. The second component contains the logic necessary to determine if a photon falls within it and is affected by it. Apertures can simply pass, block, redirect, modify, or generate new photons (e.g., fluorescent).

3. Detailed Descriptions of the Aperture Program

The *lua* program creates assemblies and sub-assemblies, the assemblies are represented internally as a list of *Assembly* structures. The sub-assemblies are temporary constructs which exist only to allow stacking of the current transformation matrix.

The back end modules called from the *lua* program create instances of apertures by creating aperture-specific data objects which contain the information necessary to fully describe the aperture. For example, an annulus is described by its inner and outer radius; a rectangle by its height and width. The module itself is described by an aperture module structure, which contains pointers to standard routines provided by the module (aperture instantiation, initialization, photon processing, and cleanup). This structure and the data object are passed to the front end utility routine, which encapsulates them inside an abstract object and appends that object to the list of apertures for the current assembly.

The processing of photons begins by reading in a photon. This is usually done from the specified input stream, but may be done from a special internal stack if any photons have been generated by an aperture. It then determines the first valid assembly with which the photon will interact. Usually this is the first assembly, but if the photon is generated by an aperture, the first valid assembly is determined by the assembly to which the generating aperture belongs.

Beginning with this first valid assembly, the central engine simulates the interactions of the photon with the apertures by an in-order traversal of the assembly's aperture list. At each visit of an aperture, the module associated with the aperture is passed the data object specific to the aperture and a copy of the photon data packet. It uses these data to determine whether it can process the photon, and returns a code to the central engine indicating whether or not it has accepted the photon for processing and the state of the photon after processing. Normally, the module does not alter the contents of the photon data packet; passing it a copy is a safety measure. In the event that it has done so purposefully, it signals the central engine (via the return code) to transfer the contents of the copied photon data packet to the original, so that the remaining apertures interact with the modified photon.

Depending upon the returned code, the central engine may discard the photon and read a new one, begin processing the next assembly in line, or restart the traversal of the current assembly. The last action is taken if the option *loop-mode* is enabled and photons have been redirected or generated. It is the only means by which a photon can successfully interact with more than one aperture in an assembly.

References

Du Bois, P. F. 1994, Computers in Physics, 8,

Ierusalimschy, R., de Figueiredo, L. H., & Filho, W. C. 1995, Reference Manual of the Programming Language Lua 2.1[1]

[1] ftp://ftp.inf.puc-rio.br/pub/docs/techreports/95_12_ierusalimschy.ps.gz