NAME

trace-shell - ray trace a shell, hey!

SYNOPSIS

trace-shell *options*

ARGUMENTS

trace-shell uses an IRAF-compatible parameter interface. The available parameters are:

tag

A prefix to be used on all intermediate files created. There are lots of intermediate files; see the section on *Intermediate Files*.

src

The location of a **raygen** compatible source script. If it is the string `default`, the value of the `source_spec` keyword in the **trace-shell** configuration file is used.

srcpars

Extra parameters to be passed to the source script. If it ends in `.lua` it is interpreted as being the name of a Lua script. See the documentation for the source script for information on which parameters are available.

output

The output stream to which to write the rays. It may be a filename, or the string `stdout`, in which case rays will be written to the standard output stream. If it is the string `default`, a file name will be created by appending the **output_fmt** to the **tag** (with an intervening period).

output_fmt

The output format of the rays. May be one of `fr`, `bpipe`, `rdb`, or a `fits` variant. See *Output Formats* for more information.

output_coord

The output coordinate system of the rays. May be one of `osac`, `hrma`, `xrcf`.

output_fields

Which data fields to output for each ray. The value may be one of

`all`

A rather large amount of information.

<field names>

A comma delimited list of field names to output. Field names may be prefixed with `-`, indicating that they are to be *removed* from the list of output fields. If the only fields specified are those to be removed, the initial output list contains all of the fields in the data.

The field name `min` is an alias for specifying the following fields:

```
position direction weight energy time
```

The order of additive and subtractive fields is unimportant; all additive fields are inserted into the list before the subtractive fields are removed.

shell

The shell to raytrace.

seed1

The first seed for the random number generator. It must be in the range [1,2147483562].

seed2

The second seed for the random number generator. It must be in the range [1,214748339]

block

The random number block at which to start. It must be in the range [0,1048575].

block_inc

The spacing between random number blocks for each random process. 100 is a good number.

tstart

The start time of the observation in seconds. If less than zero and jitter is turned on, the start of the valid jitter time range is used.

limit

The quantity of whatever limit_type specifies that **raygen** must generate. If limit_type is a unit of time, this is added to the start time (see tstart) to determine the stop time of the simulation. If jitter is on and this is set to 0, then the stop time is set equal to the end of the valid jitter time range.

limit_type

The units of the limit at which to stop generating rays.

ksec

kiloseconds of observation time

sec

seconds of observation time

Mrays

millions of rays at the entrance aperture

krays

thousands of rays at the entrance aperture

rays

rays at the entrance aperture

r/mm2

a ray density at the entrance aperture in rays / mm^2

r/cm2

a ray density at the entrance aperture in rays / cm^2

focus

A boolean parameter indicating that the focus of the system is to be determined. See the

Focus section for more details.

z

The position along the *Z* (optical) axis at which to leave the rays.

tally

If non-zero, a tally of photons will be written to the standard error stream every `tally` rays. This is useful if you're wondering why it's taking so long to run the raytrace. This tallies the number of rays which make it out of the shell, after all of the post-optic apertures.

config_dir

trace-shell will change into this directory before reading the configuration file. This allows relative includes within standard configuration files to work.

config_db

The name of the configuration file which provides the details of the mirror configuration. If this begins with `.` or `/`, the configuration file in the specified directory will be used. Note that **trace-shell** will still change directory to **config_dir** before reading the configuration file.

version

Print out the version information and exit.

help

Print out this message and exit.

debug

A comma separated list of debugging options. See *Debugging* for more information.

DESCRIPTION

trace-shell raytraces a single Wolter type I X-ray telescope shell with various apertures and baffles. It was designed around the AXAF HRMA, but may be used for other systems. In order to isolate the source specification from the specification of the optics, it uses a separate optics configuration file (see *Configuration File*).

trace-shell uses a variety of programs to accomplish the raytrace. To see the actual raytrace command pipeline, use the **debug** `pcomm` option.

Configuration File

The **trace-shell** configuration file (specified by the **config_dir** and **config_db** parameters) describes the telescope configuration. See the **ts_config** documentation for more information.

Intermediate Files

trace-shell produces a few intermediate files, prefixed by the value of the **tag** parameter:

`tag.gi`

This is a rather arcanely formatted file required by **SAOdrat**. It's not of much general interest.

`tag.totwt-in`

This file contains the number and weight of the rays at the entrance aperture. It is produced by **tot_wt**.

`tag.totwt-out`

>> This file contains the number and weight of the rays which have made it through the entire configuration. It is produced by **tot_wt**.

`tag.focus.lis`

>> This is an arcanely formatted file created during a focus run by **saofocus**.

`tag.summary.rdb`

>> This summarizes the `tag.totwt-in`, `tag.totwt-out`, and `tag.focus.lis` files.

Output Formats

trace-shell produces output in one of the following formats, specified by the **output_fmt** parameter:

`fr`

>> The `fr` format has no header. Each ray is in a `fullray` structure. See */proj/axaf/simul/include/fullray.h* for the formats of the ray structure.

`bpipe`

>> The rays are in `bpipe` format. See the **bpipe** documentation for more information on this.

`rdb`

>> The rays are written as an RDB table.

a `fits` variant

>> Various FITS formatted outputs may be specified. In all cases the output must be to a file.

>> `fits-axaf`

>>> The rays are written according to the AXAF FITS Photon standard.

>> `fits-events`

>>> The rays are written in the common astronomical X-ray "events" format. Most X-ray Astronomy software uses this convention.

Focus

If you wish to determine where the focal point for a given configuration is, set the **focus** parameter to `yes`. Because of bad interactions between the focus algorithm and wildly scattered rays, micro-roughness induced ray scattering and ghost-ray tracking is turned off when focussing. You should nominally only focus with a point source. If the **src** parameter is set to `default`, the default focus source (as specified in the configuration file) will be used. You may need to specify arguments for the focus source via the **srcpars** parameter. The focus procedure is carred out by **saofocus** which leaves its results in files called `tag.focus.lis` and `tag.focus.rdb` (where you've specified **tag**). The first file's format is pretty arcane; generally to extract the focus from there, run the script *getfocus* on it:

```
getfocus tag.focus.lis
```

which will write out the focal position (in OSAC coordinates) to the standard output stream. The second file (`tag.focus.rdb`) contains the three-dimensional position of the Global Optimal Focus.

Debugging

The **debug** options that are available are:

`pcomm`

Print out the raytrace command before executing it. This gives you some idea of which programs are running and what their inputs are.

noexec

Generate the raytrace command and any required intermediate files, but do not execute it. Most useful with the `pcomm` debug option.

reuse

Reuse the raytrace output from a previous **identical** run to regenerate the summary information. `noexec` must *not* be specified simultaneously. The raytrace parameters should be identical except for the addition of this flag.

normalize_limit

If specified and the `limit_type` parameter is density related, the output ray weights will be scaled so that the input photon density is 1. This allows specifying different photon densities for each shell to provide a uniform statistical errors while retaining the ability to easily coadd the raytraces for different shells.

noproject

Do not project the rays to the value specified by the `z` parameter. This is a temporary kludge, and will probably not survive into the next version of **trace-shell**.

save-rays:*location*

save-history:*location*

save-rays and save-history are complementary means of getting a look at the rays as they pass through the raytrace.

- save-rays will create a copy of the current state of the rays on disk. Save intermediate rays. Rays are saved in `bpipe` format to the file *${tag}-where.bp*.

- save-history will store a copy of the current state of the rays in the ray stream. It does this by changing every data packet field into an array, and using it as a stack; newer data is at index 0; the oldest is at the end of the array. To limit the number of fields which have history, use the `save-history-fields` debug option.

There are a number of pre-defined locations in the raytrace at which ray history may be saved. Multiple locations may be specified. Use the format

```
save-rays:location
save-history:location
```

location is one of

input

Rays coming out of the ray generator

h-pre-intercept

p-pre-intercept

Rays before they are intercepted with the optic.

h-pre-reflect

p-pre-reflect

Rays before they are reflected at the optic.

h-pre-scatter

p-pre-scatter

> Rays before they are scattered off of the optic

h-post-scatter

p-post-scatter

> Rays after they are scattered off of the optic

`save-history-fields=`*colon separated list of fields*

> By default, saving history (see the `save-history` debug option) saves *all* fields. This can be expensive. To limit the number of fields saved, set this option to a *colon* separated list of fields:
>
> ```
> save-history-fields=position:direction:id
> ```

`input-tap=`*command*

> The rays exiting the ray generator (before they hit the shells) will be copied to the standard input stream of the specified command (thus the name `input-tap`). The command may refer to any of the parameters given to **trace-shell** using the syntax $parameter or ${parameter}. For example,
>
> ```
> debug=input-tap='frobnicator input=stdin output=$tag.frob'
> ```
>
> The rays are in **bpipe** format.

`input-filter=`*command*

> The rays exiting the ray generator are passed through the provided command before being sent to the shells. The command must read the rays from its standard input and write the modified rays to its standard output. The command may refer to any of the parameters given to **trace-shell** using the syntax $parameter or ${parameter}. For example,
>
> ```
> debug=input-filter='snackmaster input=stdin output=stdout'
> ```
>
> The rays are in **bpipe** format.

`noghosts`

> Ghost rays will not be propagated through the system.

`output-tap=`*command*

> The rays exiting the optics (after projecting to the final requested position, but before any coordinate conversions) will be copied to the standard input stream of the specified command (thus the name `output-tap`). The command may refer to any of the parameters given to **trace-shell** using the syntax $parameter or ${parameter}. For example,
>
> ```
> debug=output-tap='frobnicator input=stdin output=$tag.frob'
> ```
>
> The rays are in **bpipe** format.

`output-filter=`*command*

> The rays exiting the optics (after projecting to the final requested position, but before any coordinate conversions) are passed through the provided command before being tallied and finally written to the requested destination. The command must read the rays from its

standard input and write the modified rays to its standard output. The command may refer to any of the parameters given to **trace-shell** using the syntax $parameter or ${parameter}. For example,

```
debug=output-filter='snackmaster input=stdin output=stdout'
```

The rays are in **bpipe** format.

scat_min_prob=*fractional probability*

set the minimum scattering probability for both optics

scat_max_prob=*fractional probability*

set the maximum scattering probability for both optics

scat_p_min_prob=*fractional probability*

set the minimum scattering probability for the paraboloid

scat_p_max_prob=*fractional probability*

set the maximum scattering probability for the paraboloid

scat_h_min_prob=*fractional probability*

set the minimum scattering probability for the hyperboloid

scat_h_max_prob=*fractional probability*

set the maximum scattering probability for the hyperboloid

SEE ALSO

trace-nest3, **ts_config**

COPYRIGHT AND LICENSE

Copyright 2006 The Smithsonian Astrophysical Observatory

This software is released under the GNU General Public License. You may find a copy at: *http://www.fsf.org/copyleft/gpl.html*

AUTHOR

Diab Jerius <djerius@cfa.harvard.edu>