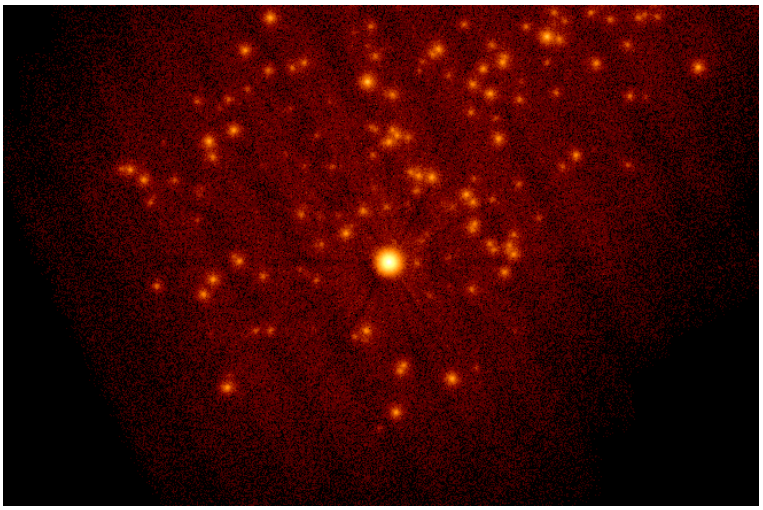
NuSim Manual



Authors
Kristin Kruse Madsen[1], Andreas Zoglauer[2], Takao Kitaguchi[1]
[1]**California Institute of Technology**
[2]**Space Science Laboratory, Berkeley**
2010

Abstract

Here go NuSim abstract ...

Preface

Here be a preface.

Acknowledgement

Here go acknowledgements.

Contents

1 Install **3**
- 1.1 NuSIM quick installation instructions . . . 3
  - 1.1.1 Installation ROOT . . . 3
  - 1.1.2 HEAsoft . . . 3
  - 1.1.3 Boost . . . 4
  - 1.1.4 NuSIM . . . 4
  - 1.1.5 Frequently asked questions . . . 5

2 NuSIM and NuSTAR science. Quick Start Guide. **9**
- 2.1 What NuSIM can do (and what not) . . . 9
  - 2.1.1 What NuSIM cannot do . . . 10
- 2.2 User science input . . . 11
- 2.3 Performing simulations . . . 12
- 2.4 NuSIM science output . . . 13
- 2.5 Example simuations . . . 13
  - 2.5.1 Simulation of extended object . . . 13
  - 2.5.2 Simulation of survey . . . 14

3 The Modules **15**
- 3.1 Introduction . . . 15
  - 3.1.1 Universal Saver and Loader: Saving and loading NuSIM simulations 16
- 3.2 Satellite super module . . . 16
  - 3.2.1 Orbit engine . . . 16
  - 3.2.2 Pointing engine . . . 16
  - 3.2.3 Orientation and alignment engine . . . 17
  - 3.2.4 Time engine . . . 18
  - 3.2.5 Geometry and detector properties . . . 18
- 3.3 Event pipeline . . . 18
  - 3.3.1 Source engine . . . 18
  - 3.3.2 Optics engine . . . 21
  - 3.3.3 Aperture engine . . . 22
  - 3.3.4 Background engine . . . 23
  - 3.3.5 Detector interactions engine . . . 24

3.3.6 Detector effects engine . . . 24
3.3.7 Trigger engine . . . 25
3.3.8 Detector data calibrator . . . 26
3.3.9 Event selector . . . 27
3.3.10 Science analyzer . . . 28
3.4 Metrology and Star tracker pipeline . . . 28
3.4.1 Metrology engine and calibrator . . . 28
3.4.2 Star tracker engine and calibrator . . . 29
3.4.3 Observatory reconstructor . . . 29

4 NuSIM Observatory Alignement Database 31
4.1 Purpose . . . 31
4.1.1 List of Coordinate System Transformations . . . 31
4.2 Format . . . 32
4.2.1 Layout . . . 32
4.2.2 Template . . . 32
4.2.3 Diagrams . . . 32
4.3 Thermal Mast Bending . . . 33
4.3.1 Thermal mast bending database . . . 33
4.4 NuSIM Database Transformer . . . 34
4.4.1 Requirements . . . 34
4.4.2 Description . . . 34
4.4.3 Use . . . 37
4.4.4 Examples . . . 38

5 The output file format keywords 43

6 The various input file formats 49
6.1 For the source engine . . . 49
6.1.1 fits-files . . . 49
6.1.2 The spe-format for input spectra . . . 49
6.1.3 The lgt-format for input light curves . . . 50
6.1.4 The 3Ddat format . . . 50
6.1.5 The src file to import sources . . . 51
6.2 For the pointing engine . . . 54
6.3 For the background engine . . . 54

7 Quaternion Rotation Convention in NuSim 57
7.1 Quaternion convention in NuSim . . . 57
7.1.1 Point and Frame Rotations . . . 58
7.1.2 Rotations in NQuaternion.cxx . . . 58
7.1.3 Rotations in NOrientation.cxx . . . 58
7.1.4 Multiple Transformations . . . 59
7.2 Space craft inertial pointing . . . 60

7.3 Star tracker pointing 60
7.4 Aspect Reconstruction 61
7.5 AspectSolve() 62

A Mast Bending report **63**
A.1 Mast bend model and database generation 63
A.1.1 Modeling the Mast as an Arc 63
A.1.2 Database creation 65
A.2 NuSIM results 66
A.3 Comparison with external code 66
A.4 Conclusion 66

B NuSim Aspect Reconstruction Verification Document **77**
B.1 Purpose 77
B.2 Test description 77
B.3 Results 78
B.3.1 Purely linear displacement 78
B.3.2 Pure tilt of the benches 78
B.3.3 Mast bending 79
B.4 Conclusion 79

C NuSim - Raytrace, MLI and Aperture stop verification **85**
C.1 Purpose 85
C.2 Test Description 85
C.3 Results 85
C.3.1 On-axis 86
C.3.2 Off-axis results 86
C.3.3 MLI results 87

Chapter 1

Install

1.1 NuSIM quick installation instructions

1.1.1 Installation ROOT

The first prerequisite is ROOT. Here are some tips for the ROOT installation

- For the current version make sure you have installed at least version 5.22, higher is probably better, but do not install a version with an uneven minor version number, e.g. 5.25, 5.27 as those are development versions

- Make sure you don't mix 32 bit and 64 bit (i.e. if you compile NuSIM as 64 bit, you also need a 64 bit ROOT).

- Do not forget to set all ROOT variables, i.e. ROOTSYS (pointing to the directory where ROOT is installed), and add ROOT to the PATH and LD_LIBRARY_PATH (or DYLD_LIBRARY_PATH an Mac) variables if you install ROOT in a place not yet included in those paths.

1.1.2 HEAsoft

In order to read and write fits files, HEAsoft is required. A standard installation should do, but take into account the following things:

- NuSIMs configuration tool verifies that HEAsoft is installed only by checking of the HEADAS environment variable is set.

- It seems that on the latest HEADAS installations the required libcfitsio.[so/dylib] (so: Linux, dylib: Mac) does not exist but only a version one, e.g. libcfitsio_34.so. Since NuSIM should be able to run with any version of libcfitsio you might have to make a link, e.g. do: ln -s libcfitsio_3.24.[so/dylib] libcfitsio.[so/dylib] in the heasoft/<system version>/lib directory.

1.1.3 Boost

Finally NuSIM requires the Boost library to work (www.boost.org). On Linux just use your favorite package manager to install the boost development tools – any version within the last 5 years should be fine. On a Mac, the tested and thus preferred installation path is to download the source code from the website, follow the setup instructions from the boost website, compile it, and install it in /usr/local. On the Mac, you should do something similar to this:

```
tar xvfz boost_1_47_0.tar.gz
cd boost_1_47_0
bootstrap.sh
sudo ./bjam install --prefix=/usr/local
```

1.1.4 NuSIM

Next, retrieve NuSIM from its repository by using subversion:

```
svn co https://www.srl.caltech.edu/svn/nusim/trunk nusim
```

Old versions before 0.9.0 can be found in:

```
svn co https://www.srl.caltech.edu/svn/nustar/trunk/nusim nusim
```

Please ask Andrew Davis (ad@srl.caltech.edu) for a svn login and password.

The above command should have generated the directory nusim with all the source code.

Then set all your paths correctly (the paths are set in a similar way to your ROOT paths - this example is using bash on Linux):

```
export NUSIM=${some directory}/nusim
export PATH=${NUSIM}/bin:$PATH
```

For Linux (attention: in some system this variable might not yet have been defined!)

```
export LD_LIBRARY_PATH=${NUSIM}/lib:${LD_LIBRARY_PATH}
```

For Mac OS X (attention: in some system this variable might not yet have been defined!):

```
export DYLD_LIBRARY_PATH=${NUSIM}/lib:${DYLD_LIBRARY_PATH}
```

To configure and compile do the following on Linux:

```
cd $NUSIM
sh configure -linux -debug -optimized
make
```

For Mac OS X do:

```
cd $NUSIM
sh configure -macosx -debug -optimized
make
```

For the time being you cannot install NuSIM into a different directory (i.e. there is no "make install").

On Mac OS X you will sometimes run into 32-bit/64-bit problems, which manifest themselves in the linker complaining about incompatible libraries. To avoid this, you can force NuSIM to compile in 32-bit mode by configuring it the following way:

```
sh configure -macosx32 -debug -optimized
```

This also requires that ROOT and HEAsoft are compiled with 32-bit. While the current version of HEAsoft on Mac OS X can only be compiled in 32-bit, you can force ROOT to compile in 32-bit using ROOT's configure flag "macosx" and not "macosx64", i.e. "./configure maxosx" vs. "./configure macosx64".

If you launch NuSIM the first time, or if you expect that the configuration file format has been changed, launch NuSIM with the default configuration:

```
cd \$NUSIM
nusim -c resources/configurations/Ideal.cfg
```

Several more NuSIM configuration files can be found in the resources/configurations directory. Now you should be able to play around with NuSIM.

Some more tips:

- Calling **make man** creates doxygen documentation. Make sure you have doxygen & graphviz installed.
- If you update NuSIM make sure to completely recompile NuSIM: **make clean** followed by **make**. Otherwise you might experience unexpected crashes or weird behaviour.

1.1.5 Frequently asked questions

Example configuration

For Linux with bash the configuration files should look similar to this. Do NOT simply copy and paste this to your .bashrc-file! You have to adapt it to your own system! This is just an example, to check if you forgot something! Attention: The sequence does matter, since e.g. HEAsoft and ROOT have libraries which are named the same way. In addition, HEAsoft provides its own version of libreadline.so, which might interfere with system maintenance as super-user root.

```
PRG=/prg

# HEASOFT
```

```
if [ "$USER" != "root" ]; then
  export HEADAS=${PRG}/headas/i686-pc-linux-gnu-libc2.5
  alias heainit=". $HEADAS/headas-init.sh"
  source $HEADAS/headas-init.sh
fi

# ROOT
export ROOTSYS=${PRG}/root
export PATH=$PATH:$ROOTSYS/bin
export LD_LIBRARY_PATH=$ROOTSYS/lib:$LD_LIBRARY_PATH

# NUSIM
export NUSIM=${SOFTWARE}/Nusim
export PATH=${NUSIM}/bin:$PATH
export LD_LIBRARY_PATH=${NUSIM}/lib:${LD_LIBRARY_PATH}
```

ROOT compilation problems

Error message similar to:

```
Compiling XrdNetDNS.cc
g++ -c -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64
-D_REENTRANT -D_GNU_SOURCE -Wall -D__macos__
-Wno-deprecated -undefined dynamic_lookup -multiply_defined suppress  -O2 -DXrdDEBUG=0
     -I. -I.. XrdNetDNS.cc -o ../../obj/XrdNetDNS.o
XrdNetDNS.cc: In static member function 'static int XrdNetDNS::getHostAddr(const char*, sockaddr*, int, char**)':
XrdNetDNS.cc:73: error: 'gethostbyname_r' was not declared in this scope
XrdNetDNS.cc:82: error: 'gethostbyaddr_r' was not declared in this scope
XrdNetDNS.cc: In static member function 'static int XrdNetDNS::getPort(const char*, const char*, char**)':
XrdNetDNS.cc:393: error: 'getservbyname_r' was not declared in this scope
make[5]: *** [../../obj/XrdNetDNS.o] Error 1
make[4]: *** [Darwinall] Error 2
make[3]: *** [all] Error 2
make[2]: *** [XrdNet] Error 2
make[1]: *** [all] Error 2
make: *** [net/xrootd/src/xrootd/lib/libXrdSec.so] Error 2
```

The Xrd component of ROOT has sophisticated dependencies, but it is not required for NuSIM. Thus simply disable it during ROOT configuration with:

```
./configure -disable-xrootd
```

NuSIM configuration/compilation problems

Error message similar to:

```
(2) ROOT

Found ROOT: /home/andreas/prg/root/bin/root
Found ROOT version: 5.26/00 (minimum: 5.22, maximum: 5.28)
[: 108: ==: unexpected operator
```

You didn't use bash to run configure or you didn't start configure with: ./configure

Error message:

```
Generating dictionary... This may take a while...
rootcint: error while loading shared libraries: libCint.so: cannot open
shared object file: No such file or directory
```

Something is wrong with your ROOT installation:

- Did you install ROOT correctly?
- Does your LD_LIBRARY path contain the correct settings for ROOT?

Error message:

```
Library not found for -lcfitsio
```

Something is wrong with your HEAsoft installation:

- Your system has only a versioned version of libcfitsio, e.g. libcfitsio_34.so but no libcfitsio.so. Since the version ID changes from HEAsoft version to HEAsoft version, HEAsoft should contain a link from the versioned to the unversioned library. but it doesn't. Therefore you have to make it yourself:
  ln -s libcfitsio_3.24.[so/dylib] libcfitsio.[so/dylib] in the heasoft/<system version>/lib directory.
- If the above is not the problem, most likely HEAsoft is not or not correctly installed.

NuSIM execution problems

The NuSIM GUI program crashes with an error message like (or you see no GUI at all or have funny fonts):

```
Attaching to program: /proc/29042/exe, process 29042 done.
[Thread debugging using libthread\_db enabled]
[New Thread 0x7f9e513606f0 (LWP 29042)]
0x00007f9e49a1ffd5 in waitpid () from /lib/libc.so.6 error detected on stdin
The program is running.  Quit anyway (and detach it)? (y or n) [answered Y; input not from terminal]
Detaching from program: /proc/29042/exe, process 29042
```

Some Xft implementations (Xft is used for font smoothing) seem to have problems how ROOT is using them. You have to reconfigure ROOT with the option –disable-xft, recompile ROOT and recompile MEGAlib. As an alternative you can also comment out the line: gEnv->SetValue("X11.UseXft", "true"); in the file $NUSIM/src/main/src/NGlobal.cxx

Error message similar to:

```
Error in <TUnixSystem::DynamicPathName>: MathMore[.so | .sl | .dl | .a | .dll]
does not exist in <long list of paths>
Error in <ROOT::Math::IntegratorOneDim::CreateIntegrator>:
Error loading one dimensional GSL integrator
```

Something is wrong with your ROOT installation. Either you did not compile ROOTs MathMore library (did you say disable-mathmore during configuring ROOT?), or the MathMore library couldnt be compiled because GSL (GNU scientific library) isnt installed on your system. Either way make sure GSL is installed on your system and you have configured ROOT to compile MathMore. Fixing this requires a recompilation of ROOT.

Chapter 2

NuSIM and NuSTAR science. Quick Start Guide.

One of the main goals of NuSIM is to predict, reproduce, and help understand NuSTAR measurements. For these tasks NuSTAR can be considered as a black box, and the user only has to think about the required input data and the output data he gets from NuSIM.

This chapter will give an overview of NuSIM's science capabilities, the required user inputs, how to perform the simulations, and the output the user can expect from NuSIM. This chapter therefore can be considered as a quick start guide to NuSIM.

If you are not actually going to be running NuSIM, but are going to provide the NuSIM team with input data, you don't have to be worrying about section 3, also parts of section 2 may not be relevant but browsing through it is still important. In particular pay attention to desired input units and formats.

2.1 What NuSIM can do (and what not)

NuSIM's strongest point is its imaging simulation capability. The NuSTAR observatory is very dynamical and a steady nominal observation is far from actually steady. The mast will go through an orbital bending cycle due to the sun/shade patterns on the mast, and this can in the worst case cause the effective area to change by as much as 10%. In addition there are other thermal perturbations which may affect the reconstruction of the source, increasing the PSF. All of these effects are included in NuSIM, and NuSIM will mimic a typical observation environment, then perform the typical aspect reconstruction on the data. The output will be an aspect reconstructed event list in sky coordinates, equivalent to a level 2 fits file.

NuSIM is very well suited to investigate tiling and pointing patterns of sources as it offers full control of how to orient and where to point the observatory. Also due to the detector gaps care should be taken to think about exact locations.

NuSIM will also correctly process the input spectrum of the source, running it through a detector engine that simulates the NuSTAR detector response. However,

if the goal of the simulation is just to get the spectrum, then it is better to use XSPEC, since the choices in input spectrum are limited and a simulation may take several hours.

NuSIM is not well suited for iterative investigations. It is therefore advised that first passes and best estimates on what spectrum and integration time to use in a simulation, should be done prior to the simulation in XSPEC or a similar tool.

NuSIM is suitable for :
Pointing and tiling pattern simulations for extended fields.
Extended objects.
Pointing environmental stability.
Sensitivity studies.
NuSIM is not suitable for:
Variability. NuSim can not simulate variable spectra.
Detailed spectral analysis. Very time consuming as any change in parameter requires a new simulation.

2.1.1 What NuSIM cannot do

Currently NuSIM outputs an event list that can be loaded into DS9. Extractions can be made using FTOOLS, however, at the time NuSIM does not provide an ARF generation routine, so it is not possible to extract precise fluxes. At the time of writing we are trying to remedy that, but it is a complicated process that may not be imminently available.

NuSIM currently can not simulate variable sources, i.e. there is no way to input a time varying spectrum. Optionally though if a source is known to be in several different states at a certain fraction of time, then NuSIM can simulate each state separately, and the simulations can be stitched together afterwards.

Similarly NuSIM can not assign several different spectral shapes to an input image. Each input image can only have one spectrum assigned to it. If several spectrum exists then it becomes necessary to make an image for each spectrum and simulate them as separate sources.

In any instance where one observation is composed of several individual simulations that must be added together in parallel (as opposed to stitching together simulations back to back), there will be an error in the absolute countrate. The reason is that the deadtime calculation depends on the total flux, but each sub simulation will have ran at a fraction of the total flux. For example an image of a source consisting of a core and a halo, that each has a different spectrum, should be broken up into two sub images. If each of the images is responsible for 50% of the combined flux, the two simulations will be run with 50% of the flux. The resulting countrate of each sub-simulation will be higher than the actual countrate had there been only one simulation with 100% flux. The error in the countrate depends on the flux level, and will be significant for fluxes above 10 milicrabs. Below that it is negligible.

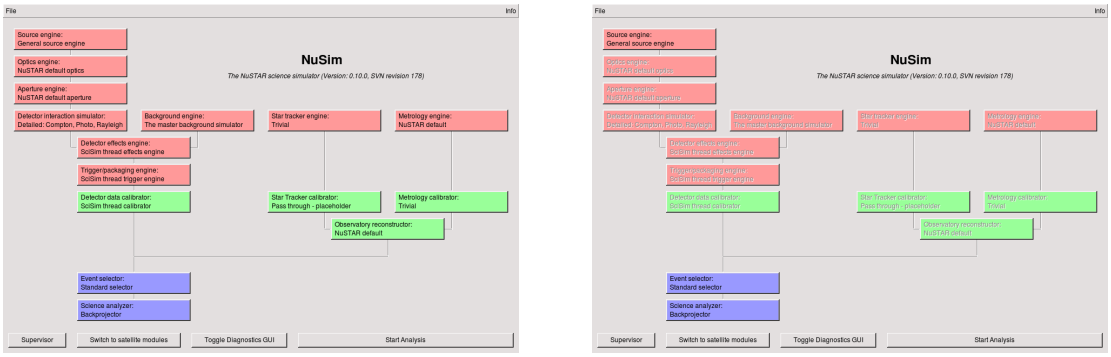


Figure 2.1: Illustration of the main NuSIM GUI window is normal (left) and astrophysics (right) mode: In the latter, only the source, event selection, and backprojection module are active.

2.2 User science input

NuSIM contains many more possible input options than are required for science simulations. Indeed, there are many possibilities to set wrong parameters. In order to prevent this, NuSIM has an astrophysics mode, which will allow the user to only modify modules relevant for science simulations. All other modules are set to standard values by loading them from a default astrophysics configuration file. This has the advantage that the user still can use his or her old configuration files even if the default configuration a simulation module has changed. In order to activate astrophysics mode, one has to start NuSIM using the "-p"-option: *nusim -p <name of configuration file>*.

Then only the following modules can be modified:

- The source module: It contains a list of sources which are defined by a name, a beam (e.g. far field point source, or a fits file) determining the origin directions, a spectrum, as well as a flux. The beams relevant for science simulations are the far-field point source, which require the RA and DEC of the source, and the fits-image. The later is usually an image which has been measured by e.g. Chandra. The spectrum is defined by its type, for example power law. After the type is selected, the user can set the other spectral options. In the case of a power law this encompasses the minimum and maximum energy as well as the photon index. The final parameter is the flux. In order to enable simulations of various beam and spectral parameters, it is necessary to give the energy-integrated flux of each source in ph/s/cm^2. The user has to take care of the integration. An exception is the special spectral option "Normalized function in ph/cm2/s/keV". There you give the spectrum in mathematical form, which must contain the correct normalization in a C/C++ compatible form such as "7.0e-6*pow(x/10, -1)*exp(-sqrt(x/2.2))". Pay attention to use of "x" for energy and "e" not "E" for the exponent. Finally one can give a light curve, either a flat one (which means no light curve) or a light curve from file with the additional option to loop over the light curve (e.g. for pulars). To set many sources at once, it is also possible to

read them from an ASCII file using the "Import from file" button. The file is a space-delimited csv file with different columns. If you only have point sources with simple power laws, the columns are: source name (no white spaced allowed), beam type (1 = far-field point source), RA (deg), DEC (deg), spectral type (3 = power law), minimum energy (keV), maximum energy (keV), photon index, flux (ph/s/cm^2). An example can be found in the resource/configurations directory: GalacticCenter.ImportExample.txt

- The pointing module: It is located in the "satellite options" pane. Here you give the RA and DEC of the direction where NuSTAR should point. This should obviously be the direction of your sources. If you do not click the "All times are absolute" button then you will be able to set the total observation time in the Supervisor and the times given in the pointing will be scaled. Otherwise all times are absolute. If you have a more complex pointing situation, you can import a list of pointings from file. One easy possibility is to generate a default pointing pattern file by clicking the "Pointing" button in the source module. This will generate a text file with different pointings which will cover the area of the sources given in the source generator. The file format is described in 6.2.

- The event selector: The event selector has two tasks, to store the simulated events to file and to perform event cuts. Currently only energy cuts are implemented. The events can be stored in three formats, a fits file containing an event list which can be used with the standard fits tools, a ROOT file, or an ASCII file. Name and path of the file is given in the supervisor UI. The user can also choose to store the events before or after the event cuts.

- The supervisor: In the supervisor you give the total observation time and the name and path of the files you want to save to. If you have set absolute times in the pointing module, the simulation will be stopped if the last pointing is finished, or earlier if the observation time is over.

The options of all others modules should be left at the standard settings — unless you are really sure what you are doing.

2.3 Performing simulations

Performing simulations is as simple as pressing the "Start Analysis" button. As soon as all modules have initialized the diagnostics window will come up. Every module can have a diagnostics tab attached. The default ones are associated with the source module, the detector effects engine, and the backprojector. Switching to the backprojector (the "Results" tab) will show backprojections of the simulated events as well as the simulated spectrum. The GUI is updated after as many events as given in the supervisor GUI. If this number is too low, the GUI is updated too frequently which will slow down the simulations. The simulation can be stopped at any time by pressing the "Stop Analysis" button. If a file is selected in the event selector then all events are not only displayed in

the GUI, but in parallel also written to the file. After the simulations is finished some useful summary information is printed to the screen.

2.4 NuSIM science output

NuSIM outputs the results of the simulation in a simple FITS OGIP compatible event list format, that has coordinates of the event in the standard FITS OGIP sky coordinates (X, Y), and pulse height PI in keV. Note that the (X,Y) units are not in degrees. DS9 handles the conversion itself, but if the user needs to convert into real RA,DEG this is done using the keywords provided in the header. For details on how to do this see 2002A&A...395.1077C.

Finally there is a GRADE keyword which currently specifies whether an event was:

1 A properly double bounced source photon

2 A background photon

3 A single bounce source photon (ghost ray)

This allows the user to sort the list and make different energy and grade cuts. It should be stressed, however, that all three grades are part of the actual image.

The output file will not run cleanly in XSELECT due to the fact that XSELECT does not have a NuSTAR mission parameter file. To use XSELECT with the files it is necessary to download a mission parameter file not part of the general HEASOFT distribution. This file with instructions can be found on the projectspaces site at:

`http://nustar.projectspaces.com/documents/index.php?action=detail&cid=8&id=979`

2.5 Example simuations

Shown here are two example simulations describing the input used to run them.

2.5.1 Simulation of extended object

When simulating an extended source the objective is usually looking for features such as filaments in a supernova remnant. If the source has already been observed with CHANDRA or XMM then the input to NuSIM may conveniently be a fits image from those observatories. If not then the object must be simulated with some external tool to create an intensity map of the source. This map must then be saved in a fits image file with the appropriate header keywords present. For details on how to do this see 2002A&A...395.1077C.

Once the choice of input image has been made, a spectrum for the source must be specified. The spectrum may be either a spectra from the NuSIM gui, or input as a file. When the shape of the spectrum has been chosen, a desired energy range must be specified, and the total flux to be used for the simulation, must be calculated over

that particular energy range. For example if a power-law with normalization 10 and photon index 2.1 is chosen over the energy range of 5-80 keV, the total flux is 1.47 ph/cm^2/sec. This number must be supplied with the spectrum parameters, since in NuSIM the normalization is kept separate from the spectral models and included in the total flux. In a similar way an input spectral file need not be in absolute units, but should be supplied with the total flux of the energy range over which the file is specified.

Only one spectrum can be assigned per image. If a source exhibits several different spectra the image must be decomposed into separate images, and each image assigned to a spectrum.

Finally due to the geometry of the detectors a pointing plan must be considered. Things to think about are the number of pointing's needed to cover the source, and placement with respect to detector gaps.

2.5.2 Simulation of survey

Typically for the simulation of a survey, the objects are considered point sources. In this instance a catalogue with locations of each source can be used. If in addition the source spectra can be approximated by the NuSIM supplied models, the model parameters can be loaded as list together with the source location. If the sources require a more complex spectra it is necessary to load the spectra for each source as a separate file.

When the shape of the spectrum has been specified, a desired energy range must be chosen and the total flux calculated over that particular energy range, just as it was described above for the extended object.

For surveys pointing strategies are of great importance. Things to consider when designing a pointing strategy is how much each consecutive pointing should overlap, what rotation of the observatory, the integration time, etc. There are tools in NuSIM to help facilitate the optimal pointing, so the values to supply the NuSIM team with, is the total size of the field to be surveyed, the desired exposure time per pointing, the total exposure time, and the amount of desired overlap. We will then optimize a pointing strategy based on that.

Chapter 3

The Modules

3.1 Introduction

Figure 3.1 left panel shows the NuSIM graphical interface. Left clicking on any of the coloured boxes will bring up an options menu for different modes of the module, and left clicking on the module will bring up the options menu for the chosen mode.

At the bottom of the window there are four grey boxes. The "supervisor" button brings up the main time control of the simulation. The first entry is the total time of the simulation in seconds, and the second entry is the update rate of the diagnostics plots in terms of number of events. It is advisable to set the update rate of the diagnostics window high, since the plotting slows down the simulations speed.

The second grey button "Switch to satellite modules" switches the main window to the satellite control window shown in Figure 3.1 right panel. This is the control window of the satellite and mainly deals with the loading of databases and setting the space craft pointing. The modules available through this window are described in detail in 3.2.

The third grey button toggles the displaying of the diagnostics window.

The final button starts the simulation.

3.1.1 Universal Saver and Loader: Saving and loading NuSIM simulations

There are a variety of output file formats in NuSIM. Each of these will be described under their respective module. However, at almost any point in the simulation the user can choose to save the contents of the EVENT object to an ASCII file. The format of this NuSIM file is in detail described in Chapter 5.

Under almost every module there are two common options: "Event Saver: Universal saver" and "Event loader: Universal loader". The save option will save the contents of EVENT at that point in the pipeline. The loader will load a NuSIM file and continue the simulation from that point in the pipeline. These two options are mostly used for debugging and testing of the NuSIM functionality.

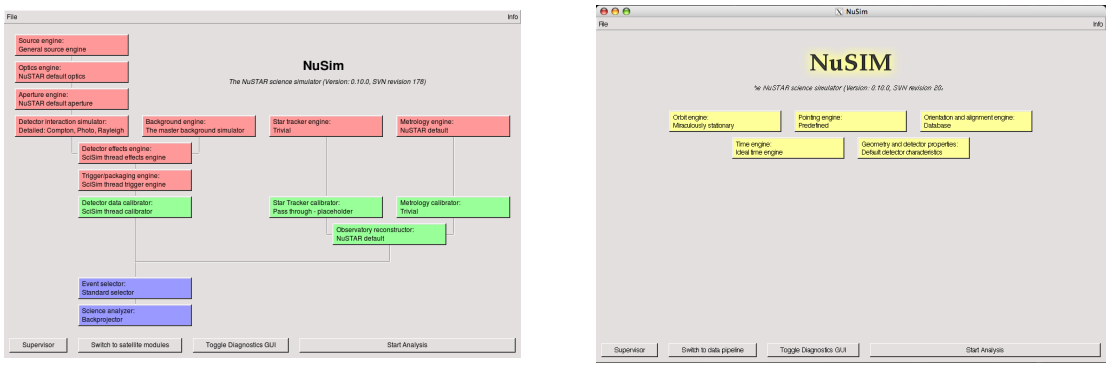


Figure 3.1: Illustration of the main NuSIM GUI window (left) and satellite module window (right).

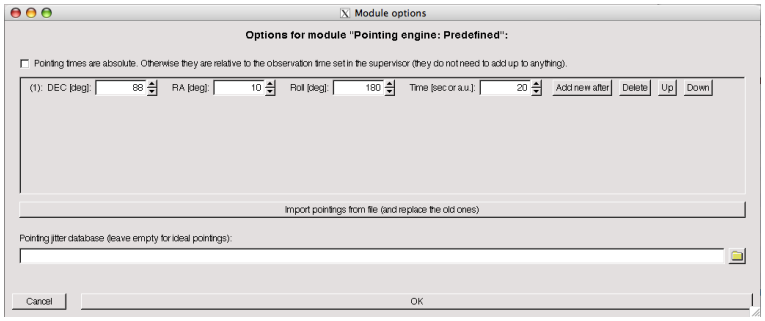


Figure 3.2: Pointing module options GUI.

3.2 Satellite super module

The satellite super-module (internally represented by the class NSatellite) provides an interface to all other satellite modules, which are required by various other simulation and analysis modules.

3.2.1 Orbit engine

The orbit engine provides information about the current orbit position of the satellite, e.g. altitude, inclination, position in TBD coordinates.

3.2.2 Pointing engine

The pointing engine provides the pointing of the focal plane module in declination and right ascension. Figure 3.2 shows the options interface GUI for the pointing module. As illustrated the pointing module allows the user to define multiple pointings. Each pointing requires a coordinate set, a space craft yaw (rotation around z-axis), and an exposure time. Checking the box in the upper left turns the exposure times into absolute

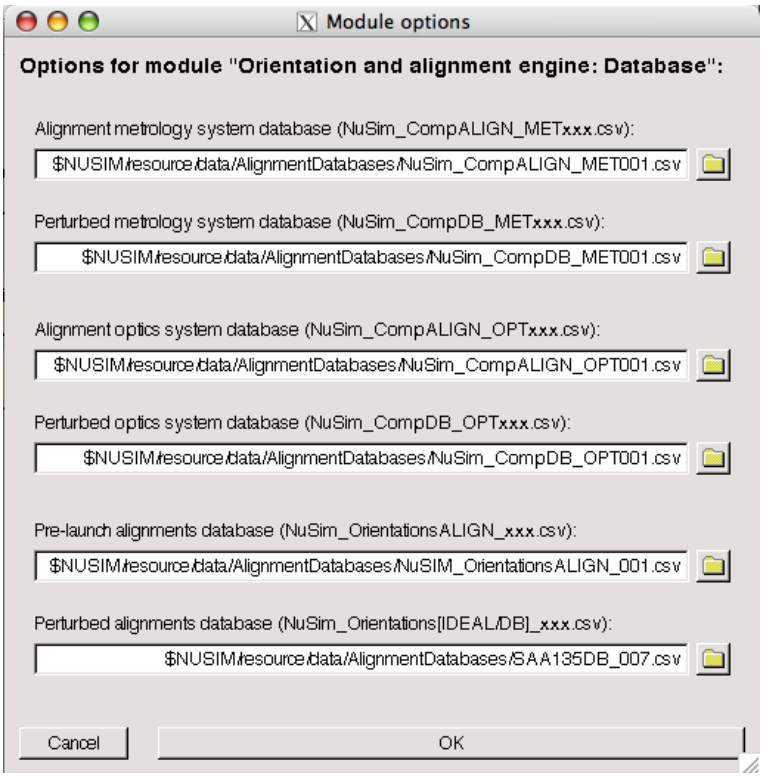


Figure 3.3: Database and alignments options GUI.

seconds, where as leaving it unchecked, the times become relative to the integration time given by the supervisor.

Alternatively the pointings can be loaded in from a file, which will replace any manually entered pointings. Such a pointing file can be for example be created by the source module, which finds an optimal pointing pattern for a given region size.

Finally the last box in the GUI allows the user to load a pointing jitter data base of the spacecraft.

3.2.3 Orientation and alignment engine

The database and alignments module controls input databases and Figure 3.3 shows the interface options GUI. Each database entry comes in two forms: the ideal alignment of the system as defined "pre-flight", and the "in-flight" alignments which will be subject to thermal perturbations. The databases are divided into 3 groups: the optics, the metrology and star tracker, and finally the rest of the spacecraft alignments. The optics, metrology and star tracker are kept separate so that they can be changed frequently without having to redo the other databases.

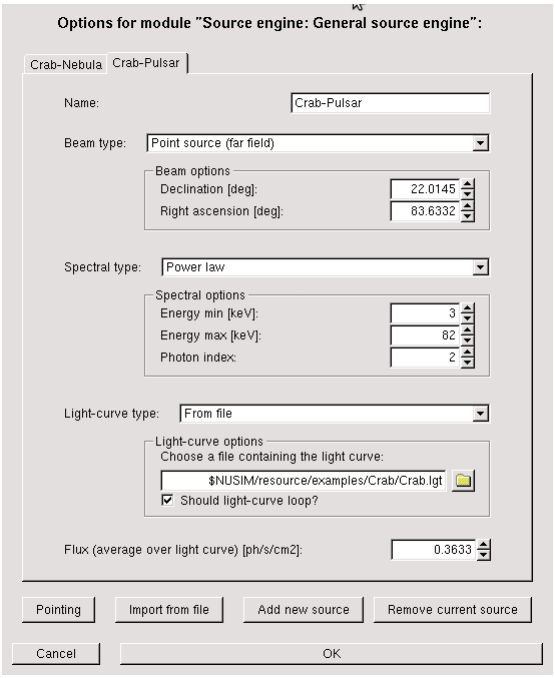


Figure 3.4: Source module options GUI.

3.2.4 Time engine

At the moment the time engine provides the absolute time of the satellite. Modules such as the detector module, the metrology and star tracker module derive their own time from this absolute satellite time and transfer it to the event, star tracker and metrology data sets.

3.2.5 Geometry and detector properties

3.3 Event pipeline

3.3.1 Source engine

The source module generates the initial spatial and spectral photon distribution for the simulator. Figure 3.4 shows the options interface GUI for the source module. Multiple sources can be generated and each source needs to have specified a beam type, spectral type and a flux. It is important to note the the flux here is in (ph/cm^2/s). In addition, the flux is always the integrated flux within the given energy bounds set with the spectral options.

You need to set four parameters for a source: its name, its beam type, its spectral type, and its flux.

The beam type defines the geometry and the position of the source. The position

is absolute and in degrees. Make sure that the source and telescope pointing matches. The following beam types are available:

- **Point source (far field)**: This is a point source at infinity. It requires RA and DEC coordinates in degree.
- **Disk source (far field)**: This creates a disc at infinity defined by its radius. It requires RA and DEC coordinates in degree as well as the radius of the disk in degree.
- **Point source (near field)**: This is used for mimicking a calibration source at a finite distance above the detector. It requires the position of the source (x, y, z in mm).
- **Pencil beam (near field)**: This beam is used to verify CZT detector calibrations. The pencil beam requires a start position as well as a direction (x, y, z in mm), as well as the radius of the beam.
- **Read from fits file**: This creates a photon field based on a fits intensity image.
- **Normalized combined energy-beam-flux file**: This is the only all-in-one option, a file which contains a normalized (ph/cm^2/s/keV/sr) energy-beam-matrix. You do not need to give an additional spectrum or the flux. The file format is described in section 6.1.4.

The spectral options are:

- **Mono-energetic**: Requires only the line energy in keV.
- **Linear**: Requires the upper and lower boarder of the energy range in keV.
- **Power-law**: Requires the upper and lower boarder of the energy range in keV as well as the photon index.
- **Broken power-law**: Requires the upper and lower boarder of the energy range in keV, the break energy in keV, as well as the lower and upper photon index.
- **Black body**: Requires the upper and lower boarder of the energy range in keV as well as the temperature in keV.
- **File with differential flux**: Read differential spectrum from an ASCII file. The spectrum does not require any special normalization besides being 1/keV, since the total flux is as always given by the flux keyword. An example file of the format can be found at:
  $NUSIM/resource/data/SourceGenerator.examplespectrum.dat
- **Normalized function in ph/cm^2/s/keV**: Requires the upper and lower boarder of the energy range in keV as well as a function in C/C++ form. You can use all C/C++ function such as exp, log, pow, sqrt, tan, cos, sin, etc. as well as all

ROOT functions in the standard ROOT form such as TMath::Gaus(x), etc. Please consult the ROOT manual for all available functions.

Example 1: 7.0e-6*pow(x/10, -1)*exp(-sqrt(x/2.2))

Example 2: TMath::Gaus(x, 67.9, 1.5)

Pay attention to use of "x" for energy and "e" not "E" for the exponent. This is the only spectral option which does not need a flux input, because it is assumed to be correctly normalized.

The light curve options are:

- **Flat - no light curve**: Do not perform any flux variations based on a light curve.
- **From file**: The light curve is given in a file (see 6.1.3). The values given in the file have arbitrary units, since the total normalization is performed by the flux. Therefore, the given flux is always an average value. The file also contains the start and the stop value, which ultimately correspond to start and stop values of the source. This allows to simulate, e.g., flares. If the option "Should light-curve loop" is selected, the the light-curve is in an infinite loop, which allows the simulation of, e.g., pulsars.

The final required parameter is the flux in $ph/cm^2/s$, which is always the average flux (i.e. also averaged over the light curve). This format instead of e.g. $ph/cm^2/s/keV/sr$ had to be chosen to allow to combine each spectrum with each beam. Therefore don't forget to integrate over keV, if you use non-mono-energetic spectra which are given in keV^{-1}. This final input is not required for the spectral option "Normalized function in $ph/cm^2/s/keV$" or the beam "FarFieldNormalizedEnergyBeamFluxFunction".

For the sources at infinite distance the photons are started randomly from a disk on a sphere surrounding the opening of the optics modules (the module is chosen randomly) to enable the correct simulation of the effective area as a function of incidence angle. The start time is randomly determined (Poisson distribution) according to the source flux. When the start position, start direction and photon energy are determined, the photon is handed over to the optics module for further simulation.

Known issues

- If you have too many sources then no tab is displayed in the GUI. This is a ROOT issue within the TGTab class. If you have too many sources it can on some systems take a long time to display the GUI. This is another ROOT issue within the TGTab class.
- If you change the beam or spectral type, all parameters get reset
- Stay away from having sources close to the zenith or nadir (RA, DEC is 0 or 180 degrees)

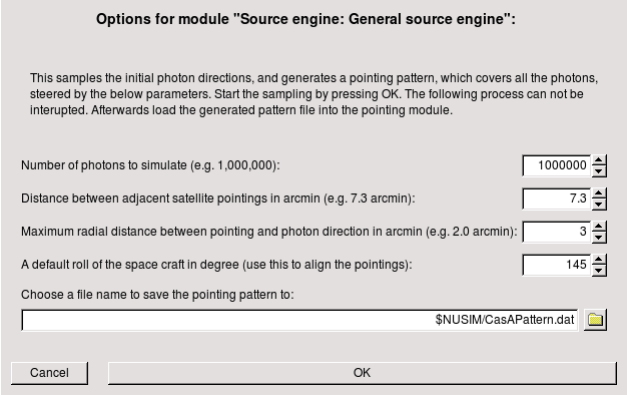


Figure 3.5: GUI for generating pointing patterns.

A special feature of the source engine is that it can also generate a simple pointing pattern. To do so press the button "Pointing" in the source engine GUI and the GUI see in Fig 3.5 should appear.

The pointing pattern is generated by first simulating a set of photons. The number of photons is given in the first entry box. It should be high enough to sample all sources, but not too high because then the simulation takes too long. 1,000,000 photons seems a reasonable compromise for most situations. For each photon the start direction in RA and DEC is stored. After the simulation, the source region is covered with a rectangular pattern of pointings. The distance between the pointings is given in the second entry field. Since the source field is not necessary rectangular many individual pointings may be not necessary. The last entry in GUI determines how close the pointing direction (i.e. the optical axis of the instrument) must be to a simulated photon in order to accept the pointing. A value of, e.g., 3 arcmin means that one photon must have been simulated within a radius of 3 arcmin of the pointing direction of the instrument. Don't make the value too small, otherwise it's unlikely you have simulated a photon within the disk, and do not make it larger than the half the field-of-view of the instrument. The next input box gives the yaw (rotation around z-axis) of the space craft, which allows to align the field-of-view (by trial and error at the moment). In the bottom entry box you give the name of the file to which the pointings are stored. You have to read this file in the pointing module in order to use it.

In order to test and verify the coverage of your generated pointing pattern, as well as the evenness of your image, you can make a full simulation with a flat input distribution, e.g. a disk source, which is large enough to cover all pointings.

3.3.2 Optics engine

The optics engine is a full-fledged ray-trace to simulate the actual optics. It incorporates the exact geometry of the optics, and uses externally generated reflectivity files to calculate the reflection of the photons off the mirrors. Additionally it has a module to

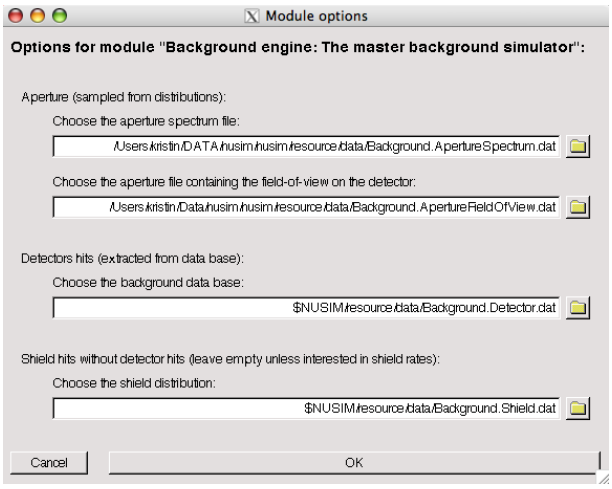


Figure 3.6: Background interface options GUI for the "master background simulator".

simulate deflection of the X-ray photons due to mirror imperfections, which will generate the mirror point spread function. The ray-trace also keeps track of single reflections, called "ghost rays", where the photon only reflects off one mirror while passing through.

The module has three options:

- **Scattering**: This options turns on the scattering of the photons due to figure errors in the mirrors. Scattering must be on for a science simulation. It is important to note that this is scattering in the reflection sense only, and not of any particle effects.

- **Perfect Optics**: This option focuses all rays to one spot. This option should not be used with science simulations and is primarily for debugging.

- **Ghost rays**: This option enables the ghost rays. As a default it should be on when running a science simulation.

The input of this module is a photon location, direction and energy handed to it by the source module, and the output is location, direction and energy after the photon has exited the optics.

3.3.3 Aperture engine

The aperture engine places an aperture in the photon path and rejects any photon hitting the aperture. The aperture stop is located 833.187 mm above the detector, and has an opening diameter of 58 mm.

3.3.4 Background engine

The background engine draws the background from an external GEANT simulation, combining it with the expected background flux from the sky. This module has three options:

- **No Module**. It is possible to run NuSIM without having a background engine.
- **Master Background Simulator**. As the name implies this is the actual background option to use when running science simulations. Figure 3.6 shows the interface options GUI and the files can be found in **$NUSIM/resource/data/**.
- **Event Loader: Universal loader**. Allows the user to load a previously saved event list.

The master background engine contains 2 different background components, the instrumental background and the aperture background.

The instrumental background has been generated using Geant simulation which comprised the complete expected background: cosmic photons, protons, electron, and positrons, Albedo photons, electrons, positrons, neutrons and protons, trapped protons from passage through the SAA as well as all activation from neutrons and protons. it is split into two parts which have to be set in the GUI: All events which cause interactions in the detector — this includes detector only and shield and detector interactions — and shield only interactions. Although 20.000 hours supercomputer time were available, only 100.000 s of background could be simulated. As consequence all 100 ks the background is reused. To minimize problems, a random start position in the background file is used and the interaction positions on the detector are modified by randomly switching the detector(s) in which the pattern(s) occurred and by rotating the pattern(s). The energy, however, is not modified.

The second component is the aperture component, i.e. cosmic diffuse photons which which have an unobscured path to the detector because the aperture is not perfect. However, Beryllium absorption, etc. is taken into account. The input is split into two files, a spectrum and a position pattern on the detectors (which is uneven due to obscuration of the optics). The original input for the data was the Gruber et al. cosmic diffuse spectrum.

The final component the hot-pixel engine and steered by a hot-pixel file described in 6.3. For all normal simulations this should be left empty. The hot-pixel engine is capable to simulate random noise with random fire frequency but also individual pixels with high fire frequency for a certain amount of time. The user has the option to choose

- the telescope, detector, and position or to choose a random telescope, detector, and position.
- a given or a random start time
- an on-duration with an Gaussian randomization

- an off-duration with an Gaussian randomization or the option that the pixel stays off

- the option to randomly select a new pixel in each fire sequence

- the fire frequency and a Gaussian randomization (in the next fire frequency)

- an energy with a Gaussian width — since the random noise usually has an energy at the lower energy threshold, the maximum is usually choosen below the trigger threshold in order to get the typical quick fall-off of the noise energy values.

3.3.5 Detector interactions engine

The task of the detector interactions engine is to simulate the photons once they enter the detector. Three simulation modules are available:

- **Only propagate to the detector plane**. As the name says, this module does not perform simulations inside the detector, but only propagates the photon to the detector plane. This propagation takes into account Beryllium window absorptions.

- **Ideal interactions**. After propagating the photon to the detector plane as in the previous module, this module performs a simple interaction simulation by only taking into account the interaction depth according to the total absorption probability in CZT along the direction of the original photon. It does not take into account photo effect, fluorescence, Rayleigh or Compton scattering.

- **Detailed: Compton, photo, Rayleigh**. This is the most frequently used module. It takes account Photo effect including fluorescence photons which can escape the detector (although only the photon with the highest energy is considered), Rayleigh scattering and Compton scattering. Comparisons with Geant4 have shown that the results are within 5% of Geant4 (version 9.2).

3.3.6 Detector effects engine

The main task of the detector effects engine is to convert energies deposited in the CdZnTe detector by various interactions to charges measured at each pixel. There is no time-dependent waveform/sampling information in the output signal for each pixel. Inputs are position in detector coordinate system and energy deposited for each interaction. The input may be a vector consisting of multiple interactions. Outputs are total charge at each pixel which has a non-zero charge deposition. This engine is followed by the trigger engine which uses the pixel-wise charge information to determine triggers, veto, which pixels to read out, etc. This module has 6 options:

- **No module**. It is possible to run NuSIM without having a detector effect engine.

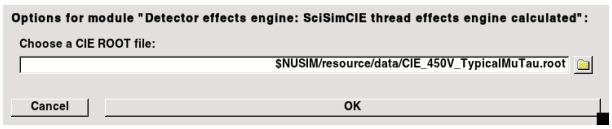


Figure 3.7: GUI for the SciSimCIE detector effect engine.

- **SciSim**. The module converts energy of each interaction to charge only in the pixel where interaction occurs with a charge loss database derived from observed data. The surrounding 8 pixels have zero energy. After that, charges in each pixel are integrated.

- **SciSimCIE**. The module converts energy to charge by a simple multiplicative factor, charge induction efficiency (CIE), which depends on interaction position. In order to shorten computation time, the CIE values are produced by an external software which simulates charge transport and induction in the CdZnTe detector based on the Shockley-Ramo theorem. These values are filled in a 3-d histogram with a volume of 3-by-3 pixels in the ROOT file (**$NUSIM/resource/data/CIE_450V_TypicalMuTau.root**).
  The ROOT file can be selected via the SciSimCIE GUI in Fig 3.7. At the moment, the detector is assumed to have uniform properties, that are temperature (5°C), electric field (500 V/2 mm), and mobility-lifetime products for electrons (1.0×10^{-2} cm^2 V^{-1}) and holes (2.0×10^{-5} cm^2 V^{-1}). The SciSimCIE module needs to be followed by the SciSim trigger engine and the SciSimCIE calibrator.

- **PixSim**. (Not yet implemented.) The main task of this module is to correctly reproduce grade for charge sharing between pixels based on a database measured with an X-ray generator. It can not generate accurate spectrum or depth plot. Monochromatic peaks are simply Gaussian with zero depth. It requires the detector interactions engine as well as incident position to the detector.

- **PHE**. This module randomly generates event information by reading a PHE FITS file.

- **Ideal**. This is an ideal detector effects engine with zero energy resolution and no charge trapping or diffusion.

3.3.7 Trigger engine

The trigger module represents the trigger and downlink decision hardware aboard the NuSTAR satellite. It shall decide if an energy deposit in the shield represents a veto or the pattern on the detector is valid for a downlink. The following 4 modules are available.

- **No module**. NuSIM can run without having a trigger engine.

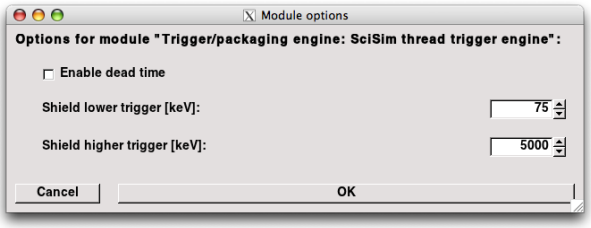


Figure 3.8: GUI for the SciSim trigger engine.

- **Ideal**. This is a simple trigger module which counts the number of trigger-activated pixels and generate a shield veto.
- **SciSim**. This module is designed to have the same function as the actual trigger system and must follow the science detector effects engine of SciSim, SciSimCIE, or PixSim. Firstly, it searches for a pixel with the highest charge and fill event information with energies in 3-by-3 pixels which are selected pixel as the center and surrounding 8 pixel. These 9 energies are used to reconstruct an initial energy in the following detector data calibrator for a science simulation and energies in the other pixels are ignored. Next, the module generates a shield veto flag if an energy deposited in the shield is above a lower threshold, which can be changed by the interface options GUI for the SciSim trigger engine in Fig 3.8. Finally, the module generates a flag to judge whether or not this event are in the dead time. This dead-time selection can be disabled by the top button in GUI.
- **PHE**. This is a specialized trigger engine for the PHE detector effects path. At the moment, it does nothing.

3.3.8 Detector data calibrator

The task of the detector data calibrator is to calibrate the data produced from a detector effects engine and the following trigger engine, and generate a level 3 data. The main calibrations are the depth cut, depth collection, and the charge share events collection. The following 5 modules are available.

- **No module**. NuSIM can run without having a detector data calibrator.
- **SciSim**. This is a detector data calibration engine which belongs to the science simulator thread. At this time, There is no depth collection or charge share collection.
- **SciSimCIE**. This module uses the 9 energies from the SciSim trigger effect engine to reconstruct an energy to estimate an incident energy. The reconstructed energy is the total energy of triggered pixels minus the average energy of the other non-triggered pixels. Then, energy scales among events with the different

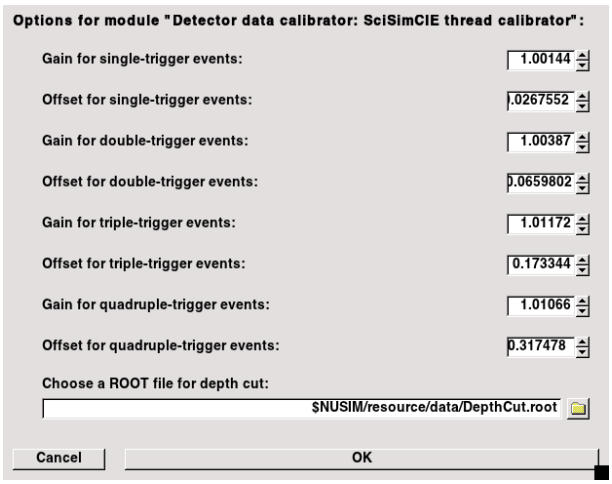


Figure 3.9: GUI for the SciSimCIE detector data calibrator.

number of triggers are calibrated with gain parameters. Although these parameters can be changed by the interface options GUI for the SciSimCIE calibrator in Fig 3.9, the default values are already optimized for the current CIE database. They may need to be updated if the CIE ROOT file changes. In addition, the module sets a flag for depth cut, which is an event selection in a depth plot to improve a S/N ratio. A boundary curve for depth cut is in a ROOT file, **$NUSIM/resource/data/DepthCut.root**. The module does not reject bad events, but just set the depth cut flag. You can select cleaned events in the following event selector module.

- **Ideal**. This module is a calibrator which goes with the ideal detector effects engine.
- **Replica for background sims**. This module is a calibrator for a background simulation.

3.3.9 Event selector

The Event Selector allows the user to filter the output and produce event-list files. The GUI for this module is shown in Figure 3.10 The first section gives you options to store the events — the file name and path are set in the supervisor GUI. Saving as FITS file will produce a simple level 2 fits file that can be loaded into DS9. Saving as ROOT will produce a ROOT file format, and saving as ASCII will produce the human readable dat file format. In addition, you can create an energy response (incident vs. measured energy) in ROOT file format.

An energy range can be entered to bound the range of the output. If there is no entry then the input range is used. The top check box allows the user to decide whether they

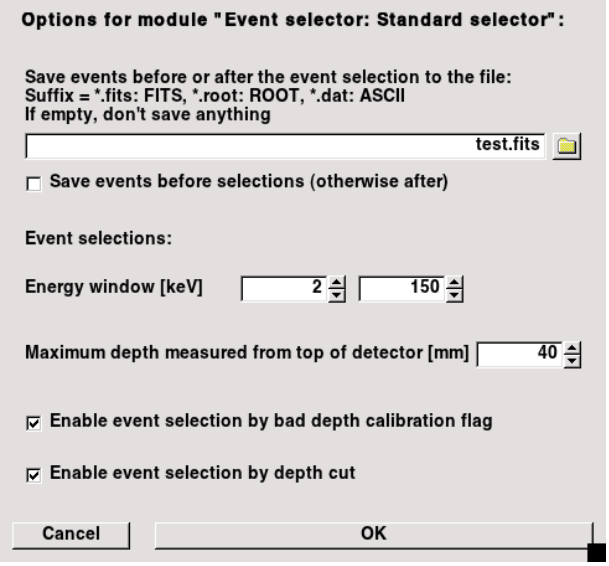


Figure 3.10: Event selector interface options GUI.

want to save the file before or after the energy filter. The reason is that there is a high energy background which otherwise would be excluded. The bottom two check boxes enable the users to decide whether they want to filter out events with depth information.

3.3.10 Science analyzer

3.4 Metrology and Star tracker pipeline

3.4.1 Metrology engine and calibrator

The purpose of the metrology engine is to generate data at a rate equivalent to the onboard metrology system. The engine takes the known perturbed aspect of the instrument system and finds the intersection of the metrology laser with the metrology detector. It will then apply noise to the data set simulating the centroiding error of the metrology detector. This is done by applying a Gaussian error to each of the measurement axis. The blurring can be toggled in the metrology GUI, but should be on by default. The 1-sigma error is reported in the database file:
resource/AlignmentDatabases/NuSim_CompDB_MET001.csv .

The metrology PSD suffers from non-linearity. In the aspect reconstruction this non-linearity is corrected out, but for testing reasons it can be overridden such that the aspect reconstruction runs without correcting for the non-linearity. The shifts can be toggled in the metrology GUI.

The engine passes the coordinates on to the Observatory Reconstructor for use in deriving the aspect reconstruction.

3.4.2 Star tracker engine and calibrator

Based on the input pointing defined by the pointing engine, the task of the Star Tracker Engine is to generate a quaternion, which defines the rotation between the Camera Head Unit, CHU, to the J2000.0 heliocentric inertial equatorial reference frame, also called VSN (Vernal, Summer, North). The origin is the intersection of the CCD plane with the optical axis of the camera. The CHU z-axis points along the boresight, and x/y axis span the CCD plane. The output of the Star Tracker is a Quaternion that defines the attitude of the CHU w.r.t. the VSN, such that

$$x_{CHU} = (Q_{CHU->VSN}) * x_{VSN}. \tag{3.1}$$

Thus if the a unit vector in the star tracker frame is transformed into the VSN frame by

$$x_{VSN} = (Q_{CHU->VSN})^T * x_{CHU}, \tag{3.2}$$

then RA=atan(y_{VSN}/x_{VSN}), DEC=asin(z_{VSN}). In the module $(Q_{CHU->VSN})^T$ is produced at a rate equivalent to the onboard Star Tracker. The module will add a Gaussian noise to the transformation to mimic solution error of the Star Camera. The 1-sigma error is reported in the database file:
resource/AlignmentDatabases/NuSim_CompDB_MET001.csv .

The transformation is passed on to the Observatory Reconstructor, which interpolates the transformation for a specific time and derives the aspect reconstruction.

3.4.3 Observatory reconstructor

The observatory reconstructor is the set of algorithms that solves the attitude and aspect problems of the NuSTAR observatory, given the metrology and star tracker data. The observatory reconstructor does not have access to the satellite module, which would contain the actual attitude and aspect of the observatory. It is responsible for recalculating that attitude and aspect, given only a certain number of inputs, which are simulated satellite data. The inputs to the observatory reconstructor are:

- The calibrated positions, in local coordinates, at which the metrology lasers impinge on their detectors. These are interpolated in time.
- The calibrated star tracker data, which is a transformation (quaternion) from local star tracker coordinates to inertial (J2000.0) coordinates. These are interpolated in time.
- The on-ground alignment data defining the ideal locations of all the instrument components. Specifically, the pointing of the optical axis in optics bench coordinates, the location and pointing direction of the metrology laser in optics bench coordinates, the location and rotation to the metrology detector from focal plane bench coordinates and to the star tracker from optics bench coordinates.

The output from the observatory reconstructor is a transformation from focal plane module coordinates to celestial coordinate R_{fbin}. For extensive details on the algorithm solution, see Chapter **??**, and for the performance details Appendix B.4.

Chapter 4

NuSIM Observatory Alignement Database

4.1 Purpose

The various components of NuSIM each operate in their own coordinate systems, and so in order to simulate the entirety of the spacecraft, it is necessary to keep track of the transformations between these systems. Furthermore, for many tests it is necessary to simulate NuSIM under changing conditions, and so for each timestep a set of coordinate system transformations must be provided.

4.1.1 List of Coordinate System Transformations

Name	Transformation
Inertial-SC	Inertial Frame to Spacecraft
SC-FB	Spacecraft to Focal Bench
FB-FP0	Focal Bench to First Focal Plane Module
FB-FP1	Focal Bench to Second Focal Plane Module
FB-MD0	Focal Bench to First Metrology Detector
FB-MD1	Focal Bench to Second Metrology Detector
FB-AS0	Focal Bench to First Aperture Stop
FB-AS1	Focal Bench to Second Aperture Stop
FB-OB	Focal Bench to Optics Bench (the mast)
OB-OM0	Optics Bench to First Optics Module
OB-OM1	Optics Bench to Second Optics Module
OB-ML0	Optics Bench to First Metrology Laser
OB-ML1	Optics Bench to Second Metrology Laser
OB-ST	Optics Bench to Star Tracker

4.2 Format

In order to provide NuSIM with a set of coordinate system transformations for each time step, a comma separated values (.csv) spreadsheet is used.

4.2.1 Layout

Each time-step is a pair of sequential rows: one for the translation between the coordinate systems, presented in (x, y, z), and the other for the rotation, presented as a quaternion, with the real component last: (q1, q2, q3, q4). Thus each coordinate system transformation is a set of four adjacent columns, and on the first row of each time-step the fourth column of each coordinate system transformation is empty.

There are several rows before the time-steps begin, representing header data at the top of the database file. These will include a row in which the name of each coordinate system transformation is in the first of the four columns its values occupy.

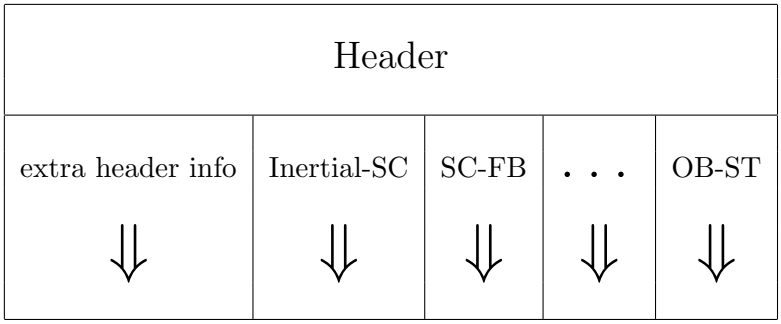
There are no rows between the time-steps, and no rows of interest after the time-steps.

4.2.2 Template

With each database format revision, a template file is produced, often in microsoft excel format. This file is simply a database with a single time-step: all the components in their ideal configuration. Historically, these file have been named along the lines of "NuSIM_OrientationsIDEAL_005.xls." Note that while Microsoft Excel's exported CSV files use carriage return ("\r") line-breaks, NuSIM requires its input databases to use newline ("\n") line-breaks.

4.2.3 Diagrams

File Layout



column layout

Section	Content			
Header	header info			
	Name			
	more header info			
Time-Step 0	x translation	y translation	z translation	
	q_1 (i coefficient)	q_2 (j coefficient)	q_3 (k coefficient)	q_4 (real)
Time-Step 1	x translation	y translation	z translation	
	q_1 (i coefficient)	q_2 (j coefficient)	q_3 (k coefficient)	q_4 (real)
Time-Step 2	x translation	y translation	z translation	
	q_1 (i coefficient)	q_2 (j coefficient)	q_3 (k coefficient)	q_4 (real)
⇓	⇓	⇓	⇓	⇓

4.3 Thermal Mast Bending

The document "Thermal_Distortion_2_for_JPL.xlsx" details the effects on the mast under the thermal effects of sunlight. In particular, it provides x and y offsets of the focal plane intersections with the beams from the optics, as well as $rotZ$, the twisting of the mast itself. These effects are most pronounced for a 170 degree angle of the mast toward the sun. To more fully model the effects of thermal mast bending on NuSTAR, a coordinate system transformation database had to be constructed for the mast in the configurations predicted for a full orbit.

4.3.1 Thermal mast bending database

The thermal mast bending databases is of the same format as the main ideal alignment database. It is produced by an IDL program which reads the simulated mast pointings and derives a transformation. Details of the mast databases and their accuracy see Appendix A.4.

4.4 NuSIM Database Transformer

NuSIM Database Transformer is a python library meant to simplify the process of generating multiple time-step databases with algorithmically generated coordinate system transformations. It takes as input the ideal template database (in csv format), and outputs sequential steps in accordance with an input function.

4.4.1 Requirements

NuSIM Database transformer is a Python script. That means it requires an installation of Python to run. Python comes standard on most modern Linux installations as well as OSX (although it may be in the developer tools, I don't recall), and is available free for most any operating system (even Windows) from python.org. To check to see if you have python, enter into a terminal:

```
python
```

If you find yourself in a python shell, you've got python. Exit with:

```
exit()
```

You do need to be able to program in Python to use NuSIM Database Transformer. Python is an interpreted language conforming to a number of common standards, which allows for mostly iterative programming with the ability to create functional (using functions themselves as variables) and object oriented programming. This readme talks about python objects, functions, lambdas, tuples, lists, and dictionaries. Tutorials and explanations are available at python.org

NuSIM Database transformer requires an ideal database in CSV format. It assumes that this database contains the entries:

```
'Inertial-SC','SC-FB','FB-FPM0','FB-FPM1','FB-MD0','FB-MD1',
'FB-AS0','FB-AS1','FB-OB','OB-OM0','OB-OM1','OB-ML0','OB-ML1',
'OB-ST'
```

All on the same row, and that the last two rows with numbers in them are the ideal translational coordinates and the ideal rotational quaternians, respectively. Furthermore, it expects each transformation name listed above to share a column with the first of the three adjacent translational coordinates and the first of the four adjacent quaternion components (which are stored constant last, by the way). This is the standard format as found in ideal databases 005 and 006, and the database can handle any other extra columns, rows, or comments inserted into the ideal database as long as these rules are adhered to.

4.4.2 Description

NuSIM Databse Transformer is simply a python library containing useful tools for the generation of parametrically-changing databases for NuSIM. These are:

arcsecondsToRadians(asec)

returns asec, but in radians.

eulerAnglesToQuaternion(xrot, yrot, zrot)

Converts euler angles to quaternions via the formula from http://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles according to the x-y-z convention, which is the same as the formula from "Quaternions and Rotation Sequences" by Jack B. Kuipers, page 207, for converting Aerospace angle sequences to Quaternions. This means that what we are doing here is technically rotating about Z, then Y, then X. There is a small change in convention here; because we are doing a frame rotation, and not a point rotation, we are reversing the coefficient terms (hence the (-1.0)*), and we are putting the constant term last instead of first. It returns these as a tuple with four entries.

Class Output_Writer

A simple class which will either print or write to a file depending on what you want to do. If no filename, or a filename of None is input, .write() prints out (sans newlines or space each time), but if a filename is input, then .write() writes to that filename. close does nothing if you are printing out, but it closes the file if you're writing a file.

Database (described below) writes with one of these so it is possible to run a script that writes a database and pipe it somewhere instead of writing immediately to a file.

Class Entry

Each entry in the database needs a quaternion (.quater) with 4 entries and a set of coordinates (.coords) with three entries. These can be set directly be inputting a tuple into setCoords or setQuatr, or by inputing 3 numbers into setCoords and 4 into setQuatr. The inputs to the Constructor are the same as the inputs to setCoordsAndQuatr, which is to say either all three coords followed by all four entries of the quatr, or two tupples in a row. getX, getY, getZ, and getQ1, getQ2, getQ3, getQ4 are self explanatory keep in mind that the last entry of our quaternions is the constant term, (getQ4)

Class Database

This is the REALLY USEFUL ONE. This is a class designed to facilitate the creation of database files as input for NuSIM. This class takes as constructor input a filename of an ideal file.

After construction, a Database object has the following fields:

- newline: the newline character (either \n or \r which this should use when writing a .csv
- desiredColumns: the list of all the column names to use in this database

- header: the text part of the database above the timesteps
- columns: the dictionary of all the column names as keys for their x coordinate in the .csv
- ideal: the ideal Entry objects read form the ideal database (with column names as keys)
- current: the current Entry objects read form the ideal database (with column names as keys)
- previous: the previous Entry objects read form the ideal database (with column names as keys)
- idealCoordsLine: the coordinates line (split by ",") which was read from the ideal database
- idealQuaterLine: the quaternions line (split by ",") which was read from the ideal database
- step: the current timestep (starts at 0)
- out: the Output_Writer object with which to write out the .csv

This means that you can get, for example, the current X coord of FB-OB with:

```
database.current['FB-OB'].getX()
```

Writing functions:

- open(filename): this function sets this object to write at the given filename. It does this by setting self.out to be an Output_Writer(filename), so you can fail to put in a filename and write to standard out.
- close(): call this when this Database is done writing.
- writeHeader(): writes the self.header to out. Do this first when writing a database,
- writeStep() writes the current Step, exactly as formatted in idealCoordsLine and idealQuaterLine, but with the information from current, to out.
- newStep(stepFunc): This is the really important one. This function creates a new step, sets previous step to current step, and current to new step. There are two things you can enter into newStep:
  - any function with input (this database object) that outputs the next step If you input to newStep any function that returns a dictionary with valid Entry objects keyed to every entry of desiredColumns, when given this database object as input (as in, newStep calls stepFunc(self)), newStep will happily make the new Step the one setpFunc output.

– any dictionary of functions which output entries keyed to any subset of desiredColumns for example, if I only wanted to play with FB-OB and SC-FB, I'd:

```
database.newStep({
'FB-OB':function_that_given_input_database_outputs_entry_for_FB-OB,
'SC-FB':function_that_given_input_database_outputs_entry_for_SC-FB
})
```

- writeDatabase(steps, stepFunc, output_filename): simply starts a database at output_filename, writes the header there, and writes steps Steps to The database, iterating each time with stepFunc, and then closes the database.

4.4.3 Use

To use NuSIM Database Transformer, a python script needs to import it as a library. Then it can use the tools listed above, which should be helpful in generating varying parameter databases for NuSIM.

The most useful single method of the library is: Database.writeDatabase(steps, stepFunc, output_filename).

To use this, create a database object (remember that the Database constructor takes as input the name of an ideal database CSV file), and then call the writeDatabase method from that object. The inputs should be:

- steps: the number of steps this database should have
- stepFunc: This can be either a function, which given a database object input, returns a dictionary with Entry objects keyed to each of: 'Inertial-SC','SC- FB','FB-FPM0','FB-FPM1','FB-MD0','FB-MD1','FB- AS0','FB-AS1', 'FB-OB','OB-OM0','OB-OM1','OB- ML0','OB-ML1', and 'OB-ST', or a dictionary containing a subset of the above keys, each keying to a function that takes a database object as input and returns an Entry object. This is the function with which the next step of the database will be computed at each step.
- output_filename: the name of the file you want to write this database to. This will be a csv file in the same format as the ideal file the Database object was created with. This can be NULL if you want to just print it.

Recall that python functions can be declared anonymously (inline) in the following format:

```
lambda inputs: output
```

Using this, it is possible to write a multi-step database in as little as one line of Python (although for readability this is inadvisable.

Remember that once a Database object has written, it increments its current step and resets its current and previous step entries. Therefore, writing multiple times from

one Database object produces a continuation of the database, possibly with varying stepFunc entries, or multiple files. If you want to write multiple different databases from one ideal database, it is easiest to use the copy library, and after creating a database object from an ideal database, copy.deepcopy() it each time you want a database object to write with.

To run a python script you've written using this or any other library, call from a terminal:

```
python script.py
```

4.4.4 Examples

Note that readme_nusim_database_transformer.py is a text-only copy of this section of this document written as executable python. Running it will run all these examples.

Importing

The first step to using the NuSIM Database Transformer library is to import it. Python allows for

```
import nusim_database_transformer
```

but this would mean we'd have to type

```
nusim_database_transformer.
```

before everything we used that was defined in the library. For example, we'd have:

```
db = nusim_database_transformer.Database("ideal_file.csv")
```

This is a pain. Therefore we will simply import everything in the library into the script:

```
from nusim_database_transformer import *
```

Create a Database Object

Next we have to make a database object. Here it is important to choose the input file which is of the correct format. The one we've been using recently as of the time of this writing is called "NuSIM_OrientationsIDEAL_005.csv" And so we create a database object from that ideal file:

```
db = Database("NuSIM_OrientationsIDEAL_005.csv")
```

If at some future date, we wanted database files computed from the ideal database "NuSIM_OrientationsIDEAL_006.csv", we would simply have written:

```
db = Database("NuSIM_OrientationsIDEAL_006.csv")
```

instead, and that would have been the only difference in the script.

Simple Database

Now it is time to think about what we'd like to write to our output databases.

Consider the simplest case: a database consisting of all ideal entries. This allows for a simple illustration of the writeDatabase method.

The most powerful input for newStep or writeDatabase is stepFunc: any function, which given the entire database object, can manipulate that object and return a dictionary representing the next step, computed in any possible fashion.

The simplest possible value for stepFunc here would be a function that given an input database returned exactly the ideal step every time:

```
def simpleStepFunc(database):
      return database.ideal
```

Now, remember that a single database object should not be used to write more than one database file, as it will always write a continuation of the steps it has hitherto been writing (the step number always increments), so it is convenient, rather than generating a new database object each time we want to write a database, to copy the one we have, and write from the copy.

```
import copy
db2 = copy.deepcopy(db)
```

To write a database of 100 steps to the file "100_ideal_steps_A_005.csv", we'd call:

```
db2.writeDatabase(100, simpleStepFunc, "100_ideal_steps_A_005.csv")
```

Furthermore, recall that using inline functions, this can be done in fewer lines:

```
db3 = copy.deepcopy(db)
db3.writeDatabase(
            100, lambda database: database.ideal,
            "100_ideal_steps_B_005.csv"
            )
```

In addition, remember that it is also possible to substitute for stepFunc a dictionary of Entry - returning functions for each entry you want to vary with each step. The current step is initialized to be the ideal step, and so we, in effect, simply do not wish to vary any entries. We can therefore input for stepFunc an empty dictionary:

```
db4 = copy.deepcopy(db)
db4.writeDatabase(100, {}, "100_ideal_steps_C_005.csv")
```

Constant Offset

Now consider a slightly more complicated case: We want a database of 100 entries which are all the same, but not ideal. For example, consider the case where FB-OB is shifted along x by 5 mm. Here we need simply change the current step, and then print out 100 steps which are all unchanged from the current step. The clearest (if not the absolute shortest, codewise) way to accomplish this is:

```
db5 = copy.deepcopy(db)
db5.current['FB-OB'] = Entry(
                            db5.current['FB-OB'].getX() + 5,
                            db5.current['FB-OB'].getY(),
                            db5.current['FB-OB'].getZ(),
                            db5.current['FB-OB'].getQ1(),
                            db5.current['FB-OB'].getQ2(),
                            db5.current['FB-OB'].getQ3(),
                            db5.current['FB-OB'].getQ4()
                )
db5.writeDatabase(100, {}, "100_FB-OB_x+5_A_005.csv")
```

Translations

Next, let's move on to somewhat more practical applications.

For example, consider shifting the x translation of FB-OB by 1 mm each step. You could add 1 to the current step each time:

```
def add1mmToX(database):
    return Entry(
                database.current['FB-OB'].getX() + 1.0,
                database.current['FB-OB'].getY(),
                database.current['FB-OB'].getZ(),
                database.current['FB-OB'].getQ1(),
                database.current['FB-OB'].getQ2(),
                database.current['FB-OB'].getQ3(),
                database.current['FB-OB'].getQ4()
            )

db6 = copy.deepcopy(db)
db6.writeDatabase(
                    100,
                    {
                        'FB-OB':add1mmToX
                    },
                    "100_FB-OB_x+n_A_005.csv"
                )
```

Rotations

Let's say you wanted to test how NuSIM reacts under various rotations about x, 0.1 arc seconds each step, between the optical bench and the focal bench, without any translation.

Recall than an Entry constructor can accept a pair of tuples, one for coordinates and one for quaternions.

```
def rotationsStepFunction(database):
    return Entry(
            database.current['FB-OB'].coords,
            eulerAnglesToQuaternion(
                                arcsecondsToRadians(0.1*database.step),
                                0.0,
                                0.0
                                )
             )

db7 = copy.deepcopy(db)
db7.writeDatabase(
                    100,
                    {
                        'FB-OB':rotationsStepFunction
                    },
                    "100_FB-OB_xrot+n_A_005.csv"
                )
```

Chapter 5

The output file format keywords

Key:	SE
Parameters:	None
Description:	Indicates the start of a new event

Key:	TI
Parameters:	1: Time in seconds
Description:	The event time in seconds

Key:	ID
Parameters:	1: Number
Description:	A unique ID of the event

Key:	OG (origin)
Parameters:	1: Number (1: source, 2: background)
Description:	Tells if the origin of the photon is from a source or a background engine

Key:	TE
Parameters:	1: Number
Description:	The telescope ID. Either 1 or 2.

Key:	RD
Parameters:	1: Right ascension 2: Declination
Description:	The RA and DEC of the original photon - empty of the photon was background.

Key:	OP (”original photon”)
Parameters:	1-3: Start position of the photon in the focal bench coordinate system in mm
Description:	4-6: Direction of the photon 7-9: Polarization of the photon 10: Energy of the photon in keV The initial parameters of the started photon (”original photon”)

Key:	IP (”initial photon relative to the optics module”)
Parameters:	see OP
Description:	Contains the parameters of the photon when it is INTIALLY rotated and translated ointo the optics module.

Key:	CP (”current photon”)
Parameters:	see OP
Description:	The last parameters of the photon (”Current Photon”): If the pipeline is saved before the detector (interactions) engine, then the current photon parameters, if the event is saved after the detector (interactions) engine then the last photon parameters.

Key:	IA ("interaction")
Parameters:	1: d or s: detector or shield 2: Telescope ID 3: Detector ID 4-6: Ideal position within the CZT detector in the CZT detector's coordinate system mm 7: Ideal energy in keV
Description:	Interactions as determined by the detector (interactions) engine.

Key:	PH ("pixel hit")
Parameters:	1: Telescope ID 2: Detector ID 3: x pixel hit 4: y pixel hit 5: pre-trigger sample sum (pulse height) 6: post-trigger sample sum (pulse height) 7: ideal average depth (mm) 8: noised average depth (mm) 9: ideal energy deposit (keV) 10: noised energy deposit (keV)
Description:	A pixel hit, i.e. the detector effects engine applied to the ideal interactions

Key:	SH ("shield hit")
Parameters:	1: Telescope ID 2: Detector ID 3: ideal energy deposit (keV) 4: noised energy deposit (keV)
Description:	A shield hit, i.e. the detector effects engine applied to the ideal interactions in the shield

Key:	NH ("nine-pixel hit")
Parameters:	1: Telescope ID 2: Detector ID 3: x central pixel 4: y central pixel hit 5: pixel 1: pre-trigger sample sum (pulse height) 6: pixel 1: post-trigger sample sum (pulse height) 7: pixel 1: trigger (bool) 29: pixel 9: pre-trigger sample sum (pulse height) 30: pixel 9: post-trigger sample sum (pulse height) 31: pixel 9: trigger (bool) 32: ideal average depth (mm) 33: noised average depth (mm) 34: ideal energy deposit (keV) 35: noised energy deposit (keV)
Description:	A pixel hit, i.e. the detector effects engine applied to the ideal interactions

Key:	PE ("PHE data")
Parameters:	TBD.
Description:	Not yet used

Keyword:	HT
Parameters:	1: Telescope ID 2: Detector ID 3-5: Position in the focal plane module (detector) coordinate system in mm 6-8: Position resolution of the above position in mm 9: Energy in keV 10: Energy resolution in keV 11: Trigger grade 12: Observatory data: time in sec 13-15: Observatory data: direction of optical axis in inertial system 16-18: Observatory data: direction of event in inertial system 19-21: Observatory data: orientation of focal plane in optic bench system: translation 22-25: Observatory data: orientation of focal plane in optic bench system: quaternion 26-28: Observatory data: orientation of optic bench in inertial system: translation 29-32: Observatory data: orientation of optic bench in inertial system: quaternion
Description:	A reconstructed hit

Keyword:	OR - the keyword is optional!
Parameters:	1-7: Space craft relative to inertial 8-14: Focal plane relative to space craft 14-21: Focal plane module 1 relative to focal plane 22-28: Focal plane module 2 relative to focal plane 29-35: Metrology detector 1 relative to focal plane 36-42: Metrology detector 2 relative to focal plane 43-49: Aperture 1 relative to focal plane 50-56: Aperture 2 relative to focal plane 57-63: Optical bench relative to focal plane 64-70: Optics module 1 relative to optical bench 71-77: Optics module 2 relative to optical bench 78-84: Metrology laser 1 relative to optical bench 85-91: Metrology laser 2 relative to optical bench 92-98: Star tracker 4 relative to optical bench
Description:	All orientations at event time

Chapter 6

The various input file formats

6.1 For the source engine

6.1.1 fits-files

NuSIM is able to read standard (Chandra) fits file as input. You can test if NuSIM is able to read the file with the program ViewFits, which is located in resource/examples/AddOn. It is not compiled by default, thus you have to type

```
make PRG=ViewFits only
```

To run it type:

```
ViewFits -f <you fits file here>
```

If a canvas with your image shows up, you are good to go.

6.1.2 The spe-format for input spectra

This format represents an input spectrum. It looks like this:

```
IP LIN

DP 3 1.775
DP 3.04907 1.73616
DP 3.09894 1.69814
DP 3.14962 1.66093
DP 3.20114 1.62449
DP 3.25349 1.58882
DP 3.30671 1.55391
DP 3.36079 1.51972
DP 3.41576 1.48626
DP 3.47163 1.4535
DP 3.52841 1.42144
```

```
EN
```

The IP line give the interpolation type. Possible are LINLIN (equals LIN), LINLOG, LOGLIN, and LOGLOG (equals LOG). Choose the interpolation type which is most reasonable for you data (i.e in which your looks like a straight line). If your spectrum contains line features it is best to always use LIN.

Data point are represented by the "DP" keyword. The first number represents the energy and the second the flux in something like ph/cm^2/s/keV. The normalization does not need to be correct because it is given with the "Flux" keyword in the source input.

Make sure the last line in your file is "EN" for "The End".

Lines starting with "#" are interpreted as comments

6.1.3 The lgt-format for input light curves

This format is basically identical to the one with a spectrum, only the second column is now a time in seconds It looks like this:

```
IP LINLIN

DP 0.0 0.1
DP 0.1 0.1
DP 0.2 0.2
DP 0.3 0.4
DP 0.4 0.8
DP 0.5 0.4
DP 0.6 0.2
DP 0.7 0.1
DP 0.8 0.1
DP 0.9 0.1
DP 1.0 0.1
EN
```

The IP line give the interpolation type. Possible are LINLIN (equals LIN), LINLOG, LOGLIN, and LOGLOG (equals LOG). Choose the interpolation type which is most reasonable for you data (i.e in which your looks like a straight line). If your spectrum contains line features it is best to always use LIN.

Data point are represented by the "DP" keyword. The first number represents the time and the second the light curve value in arbitrary units. The normalization does not need to be correct because it is given with the "Flux" keyword in the source input.

Make sure the last line in your file is "EN" for "The End".

Lines starting with "#" are interpreted as comments

6.1.4 The 3Ddat format

The 3Ddat format represents a 3D data space spanned by RA, DEC, and energy. It's content is flux at the axis position in ph/cm^2/s/keV/sr.

The file look like this:

```
IP LIN

# RA axis in deg:
PA 35.1 35.2 35.3 35.4 35.5
# Dec axis in deg:
TA 5.05 5.10 5.15 5.20 5.25
# Energy axis in keV:
EA 10 15 20 25 30 35 40

AP 0 0 0  0.50
AP 0 0 1  0.25
AP 0 0 2  0.12
AP 0 0 3  0.07

# Skip the rest

EN
```

The IP line give the interpolation type. Currently only LIN, linear interpolation, is supported.

The next three lines represent the data points on the axes at which the flux is given. PA (phi-axis) represents the right ascension in degree, TA (theta-axis) represents the declination in degree, EA (energy-axis) represents the energy in keV.

The following section gives the value at the axis points for the given ID (number starting with zero!) of the data point on the three axis. For example "AP 3 1 5 1.6" represents a flux of 1.6 ph/cm^2/s/keV/sr for the 4th axis point in the RA-axis (35.4 deg), the 2nd axis point in the DEC-axis (5.10 deg), and the 6th axis point of the energy axis (35 keV).

Make sure the last line in your file is "EN" for "The End".

Lines starting with "#" are interpreted as comments

6.1.5 The src file to import sources

Finally it is possible to import whole source lists into the source engine. This is done via the src-file. the following is an example:

```
Mono          1  266.20  -29.00   1  40.0                    0.00001            1
Linear        1  266.25  -29.00   2  30.0  35.0              0.00002            1
Plaw          1  266.30  -29.00   3  3.0  80.0   0.3         0.00003            1
BrokenPlaw    1  266.40  -29.00   4  5.0  80.0  20.0 0.6 2.4  0.00004            1
FileDiffFlux  1  266.35  -29.00   5  $(NUSIM)/resource/configs/SourceGen.dat 0.4 1
BlackBody     1  266.35  -29.05   6  5.0  20.0 80.0          0.00004            1
Absorbed      1  266.30  -29.05   7  3 82 7e-6*pow(x/10,-1.1)*exp(-sqrt(x/2.2))  1
AGauss        1  266.25  -29.05   7  3 82 4.3e-6*TMath::Gaus(x,67.9,2.1)         1
```

```
MonoPoint     1  266.20  -29.15                                1  40   0.000010   1
MonoDisk      2  266.30  -29.15 0.05                           1  50   0.000015   1
MonoFile      5  $(NUSIM)/resource/examples/Tycho/Tycho.fits  1  60   0.000020   1

AllInOne      6  $(NUSIM)/resource/examples/Tycho/Tycho.3Ddat                    1

CrabPulsar    1  22.0 83.6  4  3 82 2 0.36  2 $(NUSIM)/res/examp/Crab/Crab.lgt true
```

Each line represents a source (due to tex file formating one line might look as multiple in your printout) and each line consists of up to four sections:

1. The source name
2. The beam options
3. The spectral and flux options
4. The light curve options

Source name

The first column is always the name of the source. It must not contain any spaces.

Beam

In general, different beam and spectral options result in different number of columns. Therefore the beam and spectral options section is always preceeded by a number representing the beam type:

The beam options are:

1. FarFieldPoint
2. FarFieldDisk
3. NearFieldPoint — do not use
4. NearFieldBeam — do not use
5. FarFieldFitsFile
6. FarFieldNormalizedEnergyPositionFluxFunction

Attention: Only the ones relevant for astrophysics are accessible via this input options file.

The far field point source is followed by two numbers: (1) RA in deg, (2) DEC in deg.

The far field disk source is followed by three numbers: (1) RA in deg, (2) DEC in deg, (3) The extend (radius) in deg.

The far field fits file is followed by a file name. Currently it must not contain any spaces! You can use $(NUSIM) to represent the NuSIM directory: (1) File name.

The far field normalized combined energy-spectrum-flux function is only followed by file name since the file contains already all other options: (1) File name. Currently it must not contain any spaces! You can use $(NUSIM) to represent the NuSIM directory

Spectrum and flux

The spectral options are:

1. Monoenergetic
2. Linear
3. PowerLaw
4. BrokenPowerLaw
5. FileDifferentialFlux
6. BlackBody
7. NormalizedFunctionInPhPerCm2PerSPerKeV
8. NormalizedEnergyPositionFluxFunction — not required since redundant with FarField-NormalizedEnergyPositionFluxFunction

The monoenergetic option is followed by 2 numbers: (1) Energy in keV, (2) Flux in ph/cm2/s.

The linear option is followed by 3 numbers: (1) Minimum energy in keV, (2) Maximum energy in keV, (3) Flux in ph/cm2/s within the given spectral band.

The powerlaw option is followed by 4 numbers: (1) Minimum energy in keV, (2) Maximum energy in keV, (3) Photon index, (4) Flux in ph/cm2/s within the given spectral band.

The broken powerlaw option is followed by 6 numbers: (1) Minimum energy in keV, (2) Maximum energy in keV, (3) Break energy in keV, (4) Photon index low, (5) Photon index high, (6) Flux in ph/cm2/s within the given spectral band.

The file with the differential flux is given by only one file name and one flux: (1) File name. Again it must not contain any spaces! You can use $(NUSIM) to represent the NuSIM directory. The format is described in 6.1.2. (2) Flux in ph/cm2/s within the given spectral band in the file.

The black body option is followed by 4 numbers: (1) Minimum energy in keV, (2) Maximum energy in keV, (3) Temperature energy in keV, (4) Flux in ph/cm2/s within the given spectral band.

The normalized function in ph/cm2/s/keV is given by two number and a string (no flux required!): (1) Minimum energy in keV, (2) Maximum energy in keV, (3) Function string. The string represents a function and must not contain any spaces! You can use all C/C++ (pow, exp, sqrt, cos, sin, etc.) and all ROOT functions, such as TMath:Gaus(...), etc.

Light curve

The light curve options are:

1. Flat light curve
2. Light curve from file

The flat light curve has no additional entries.

The option light curve from file has two entries: (1) File name. Again it must not contain any spaces! You can use $(NUSIM) to represent the NuSIM directory. The format is described in 6.1.3. (2) Boolean ("true" or "false") indicating if the light curve repeats (e.g. for pulsars)

6.2 For the pointing engine

A possible input for the pointing is a pointing pattern file with the suffix pat. It looks like this (from the Tycho example directory):

```
RD 6.29 64.115 0 10
RD 6.35 64.115 6 10
RD 6.41 64.115 12 10
RD 6.29 64.1375 18 10
RD 6.35 64.1375 24 10
RD 6.41 64.1375 30 10
RD 6.29 64.16 36 10
RD 6.35 64.16 42 10
RD 6.41 64.16 48 10
```

The first number represents the right ascension in degree, the second number the declination in degree, the third number the Roll of the space craft in degree and the final number the relative or absolute observation time.

6.3 For the background engine

The only user modifiable file in the background engine is the hot-pixel file. One line in the file starting with "HP" is one hot pixel entry. The columns of the line have the following meaning:

Format

1. Telescope ID: 1, 2, or "-" if random
2. Detector ID: 1, 2, 3, 4, or "-" if random
3. x-position: in mm within wafer, or "-" if random

4. y-position: in mm within wafer, or "-" if random
5. z-position: in mm within wafer, or "-" if random
6. Start time in sec in NuSIM time (not absolute); if "-" then it is randomly determined by on and off duration
7. On duration (after first event, so that always one event is created even if the duration is zero)
8. On duration uncertainty (Gaussian, 1 sigma) in sec
9. Off duration in sec, "-" means infinity, i.e. pixel is only on once
10. Off duration uncertainty (Gaussian, 1 sigma) of hotpixel in sec
11. Choose random pixel after off ("true"/"false"): if true, switch telescope, detector and position for the next firing sequence
12. Firing frequency in Hz (individual start times are Poisson distributed)
13. Firing frequency uncertainty (Gaussian, 1 sigma) in Hz
14. Mean energy in keV
15. Energy uncertainty (Gaussian, 1 sigma) in keV

Examples

Random noise: Choose a random pixel, which only fires once

```
HP - - - - - - 0 0 0.01 0.02 true 1.0 0.0 2.5 2
```

One pixel misbehaves from time-to-time for a certain amount of time

```
HP 2 3 6 -4 0.5 - 25 40 200 200.0 false 100 200.0 2.5 2
```

A random pixel behaves badly for a certain amount of time and a different one after the next firing sequence

```
HP - - - - - - 100 200 100 200 true 100 200.0 2.5 2
```

Chapter 7

Quaternion Rotation Convention in NuSim

Introduction

7.1 Quaternion convention in NuSim

The quaternion is a four vector representation of a rotation transformation. It describes a vector, $\mathbf{u}$, and the rotation angle, θ, around that vector: $Q = \cos(\theta/2) + \mathbf{u}\sin(\theta/2)$. Computationally for NuSim we have chosen the following quaternion convention

$$Q = [q_x, q_y, q_z, q_r] \tag{7.1}$$

where q_r is the real component of the quaternion. All the NuSim quaternion arithmatics are contained in NQuaternion.cxx and NOrientation.cxx.

When rotating a vector with a quaternion, the quaternion and its conjugate must be applied

$$\mathbf{w} = Q\mathbf{v}Q^* \tag{7.2}$$

where q^* is the conjugate (or inverse) of the quaternion q. The inverse operation is

$$Q^*\mathbf{w}Q = \mathbf{v} \tag{7.3}$$

Finally two coordinate systems may have offset origins, so to fully describe the transformation between two coordinate frames, a translation is required as well. The complete transformation is therefore given as a quaternion and translation vector pair, Q, $\mathbf{t}$.

In NuSIM this is stored in the NOrientation class, which contains the quaternion, the quaternion in DCM format, and the translation. In NuSIM a transformation will usually have a capital "R" as the first letter in the variable. For example the transformation from Focal plane Bench, FB, to Optical Bench is R_fbob. The pure quaternion will usually have a capital "Q" as the first letter in the variable, and a translation a capital "T".

7.1.1 Point and Frame Rotations

A transformation between two coordinate frames A and B, that takes all vectors in A and expresses them in B, is written Q_{AB}, and the inverse or conjugate of $(\mathrm{Q}_{AB})^* = \mathrm{Q}_{BA}$.

For NuSim there is an important distinction to be made between the two operations described above. If the observer is sitting in coordinate frame A and applies $Q\mathbf{v}Q^*$, then to the observer the vector $\mathbf{v}$ got rotated by θ in frame A. This is called the *point rotation*.

For the same observer applying $Q^*\mathbf{v}Q$ will move the vector by $-\theta$. However, if instead you place the observer on the vector, $\mathbf{v}$, then in this second case the *frame* moved by θ, and $\mathbf{v}$ is now expressed in coordinates of the new frame, B, offset from A by θ:

$$[\mathbf{v}]_B = Q^*[\mathbf{v}]_A Q \tag{7.4}$$

This kind of operation is therefore obviously referred to as *frame rotation*. To get $[\mathbf{v}]_B$ back into the A frame the inverse operation is applied, which is the same as the forwards point rotation.

Since in NuSim we are concerned with expressing the locations of rays of light in different frames, our transformations are in the frame rotation mindset. The photons are static things and the it is the coordinate frames that move. Therefore if a quaternion defines the alignment of the two frames A and B, Q_{AB}, then the forward transformation is:

$$[\mathbf{v}]_B = Q^*_{AB}[\mathbf{v}]_A Q_{AB} \tag{7.5}$$

and the inverse transformation

$$Q_{AB}[\mathbf{v}]_B Q^*_{AB} = [\mathbf{v}]_B \tag{7.6}$$

7.1.2 Rotations in NQuaternion.cxx

Conventionally most libraries are written for point rotations. This is also the case for the NuSIM library NQuaternion.cxx, and it thus considers $Q\mathbf{v}Q^*$ to be the forward rotation using code:

```
v_new = m_Q.Rotation(v)
```

and the inverse rotation $Q^*\mathbf{v}Q$:

```
v = m_Q.Invert.Rotation(v_new).
```

7.1.3 Rotations in NOrientation.cxx

NOrientation.cxx is the main class. It has the following functions for rotations utilizing the quaternion library NQuaternion.cxx, where:

- **Direction**: input vector direction of a photon.

- **Position**: input position of the photon.
- **m_Translation**: is the translation of the transformation.
- **m_Q**: is the quaternion transform.

TransformIn

This is the NuSim forward transformation for a quaternion, Q_{AB}, taking a fixed vector in frame A and expressing it in frame B coordinates. Because the quaternion library is written for point rotations, the inverse of the rotation function is taken:

```
Position -= m_Translation;
Position = m_Q.Invert().Rotation(Position);
Direction = m_Q.Invert().Rotation(Direction);
```

Notice the translation is taken first then rotated by q.

TransformOut

This is the NuSIm inverse transformation of a quaternion, Q_{AB}, taking a fixed vector in frame B and expressing it in frame A coordinates.

```
Direction = m_Q.Rotation(Direction);
Position = m_Q.Rotation(Position);
Position += m_Translation;
```

Notice that the translation is taken last after the position has been rotated by q.

7.1.4 Multiple Transformations

When dealing with multiple transformations, point versus frame rotation has to be carefully considered. The two quaternions, Q_{AB} and Q_{BC}, can be combined in six different ways, which can result in a bad mixture of point and frame rotations. For a frame rotation the forward transformation combining Q_{AB} and Q_{BC} into Q_{AC} is

$$Q_{AC} = Q_{AB} * Q_{BC} \tag{7.7}$$

which is the opposite of how one would combine DCMs. Reversing the multiplication order creates the inverse transform Q_{CA}. In terms of NuSIM combining two transforms, R_AB and R_BC, appears in code as:

```
R_AC = R_AB*R_BC
v_new = R_AC.TransformIn(v).
```

7.2 Space craft inertial pointing

In NuSim the target pointing is done with the spacecraft bus and the code for it is located in NPointing.cxx. The Z-axis of the bus is pointing at the target Ra, Dec and the module thus designs a quaternion, Q_{SCIN} that will take vector = (0,0,1) and turn it into (Ra,Dec). The quaternion is obtained from:

$$\begin{aligned} q_r &= \cos(Ra/2) * \cos((\pi/2 - Dec)/2) && (7.8)\\ q_x &= -\sin(Ra/2) * \sin((\pi/2 - Dec)/2) && (7.9)\\ q_y &= \cos(Ra/2) * \sin((\pi/2 - Dec)/2) && (7.10)\\ q_z &= \sin(Ra/2) * \cos((\pi/2 - Dec)/2)\ . && (7.11) \end{aligned}$$

To account for a space craft yaw, θ_z, about the space craft observatory Z-axis we design a yaw quaternion

$$Q_{yaw} = \cos(\theta_z/2) + \mathbf{z}\sin(\theta_z/2) = [0, 0, \sin(\theta_z/2), \cos(\theta_z/2)]\ . \quad (7.12)$$

An important thing to note about these two quaternions is that they are designed for point rotations, and therefore in the frame rotation notation

$$Q_{INSC} = [q_x, g_y, q_z, q_r]\ . \quad (7.13)$$

To this we apply the space craft rotation quaternion from the right, because they are point rotations and in this scenario we first rotate the space craft bus around its Z-axis then apply the (Ra, Dec) move, to obtain the combined pointing quaternion:

$$Q_{INSC} = Q_{INSC} * Q_{yaw}\ . \quad (7.14)$$

This quaternion must be inverted to obtain the transformation from space craft, SC, to inertial space, IN.

7.3 Star tracker pointing

In NuSim the target pointing is done with the spacecraft bus, but the star tracker is the ultimate reference to inertial space at reconstruction, and the code for calculating the correct quaternion can be found in NModuleStarTrackerEngineTrivial.cxx .

Because the pointing is done by the space craft, during a bending of the mast the star tracker head will be pointed at a slightly different celestial coordinate. Since we are dealing with the orientation only the translation part of the coordinate systems must be ignored. To calculate the instantaneous orientation of the star tracker with respect to the inertial space we have the following values:

- Space craft bus orientation in inertial space. Since the spacecraft bus is rigidly tied to the focal plane such that SC=FB, we will call it Q_{INFB}.
- Orientation of optical bench with respect to focal plane bench, Q_{FBOB}

- Orientation of the star tracker with respect to the optical bench, Q_{OBST}

First we need to get the orientation of the star tracker with respect to the focal plane bench:

$$Q_{FBST} = Q_{FBOB} * Q_{OBST} \ . \tag{7.15}$$

With this we can find Q_{STIN} by combining the transformations since the space craft, SC=FB:

$$Q_{STIN} = (Q_{FBST})^* * (Q_{INFB})^* \ . \tag{7.16}$$

7.4 Aspect Reconstruction

The aspect reconstruction code can be found in NObservatoryReconstructor.cxx.

In NuSIM the reconstruction code has no knowledge of actual real instantaneous positions of coordinate systems. It draws its information from "calibrated" values and sensors to reconstruct the event. These are the calibrated variables the reconstructor uses:

- Metrology Laser position and direction, $\mathrm{R}_{ml1/2}$.
- Metrology PSD position, $\mathrm{R}_{md1/2}$.
- Star tracker orientation and position on the OB, R_{obst}.
- Optics orientation and position on the OB, $\mathrm{R}_{obom1/2}$
- Focal Plane detector orientation and position on the FB, $\mathrm{R}_{fb1/2}$
- Direction of the Optical Axis, OA, in OB.

The sensor information that the reconstructor uses are the following:

- Interpolated metrology PSD coordinates, *md1/2*.
- Interpolated star tracker quaternion, R_{stin}.
- Event data. All the information pertaining to an event is contained in the *event* class defined in NEvent.cxx.

The steps for reconstructing the aspect and the event are:

1. Time interpolate the metrology PSD coordinates and the star tracker quaternion.

```
MVector md1 = MetrologyInterp.GetMetrologyDataSet1().GetCalibratedLaserHit();
MVector md2 = MetrologyInterp.GetMetrologyDataSet2().GetCalibratedLaserHit();
NQuaternion Robst = m_CalibratedOrientationStarTrackerRelOpticalBench.GetRotationQuaternion();
```

2. Solve the transformation between Focal Plane Bench and Optics Bench, R_{fbob}

```
NOrientation Rfbob = AspectSolve(md1, md2);
```

3. Find the star tracker quaternion from OB to Inertial space.

```
Robin.SetRotation(Robst*Rstin);
```

4. Transform event into OB coordinates using R_{fbob}.

```
FP.TransformOut(event);
Rfbob.TransformIn(event);
```

5. Find the optical axis, OA, direction vector in OB.

```
MVector OA = FindOpticalAxisInSky(Robin,module);
```

6. Construct the unit vector that points from the location of the event in OB to the center of the optics

```
MVector OMF = FindOmegaF(Rfbob,module);
MVector pF = OMF-event;
pF.Unitize();
```

7. Transform the OA and the event direction vector into a vector in inertial space.

```
Robin.TransformIn(pF);
Robin.TransformIn(OA);
```

These steps completes the vectorial aspect and event reconstruction which delivers a pair of vectors, one for the event and one for the direction of the optical axis at the same moment in time, given in celestial coordinates.

7.5 AspectSolve()

The most important step in solving the aspect reconstruction is deriving the transformation between FB and OB.

Appendix A

Mast Bending report

The bending of the mast due to the sun/shade is a very important effect which needs to be implemented in NuSIM. Simulations of the mast bend are available for a handful of solar angles. These are given as deviations in the location of the optical axis on the focal plane, and assuming that the mast will arc as it bends, we decompose the transformation between the two benched due to the mast, as a rotation and translation. We then generate a database from these values and input them into NuSIM.

A.1 Mast bend model and database generation

A.1.1 Modeling the Mast as an Arc

The document "Thermal_Distortion_2_for_JPL.xlsx" details the effects on the mast under the thermal effects of sunlight. In particular, it provides x and y offsets of the optical axis impingement on the focal plane, as well as θ_z, the twisting of the mast itself around the z-axis.

A few assumptions had to be made in order to decompose the mast bend into a rotation quaternion, Q_{fbob} and a translation, T_{fbob} which together create R_{fbob}:

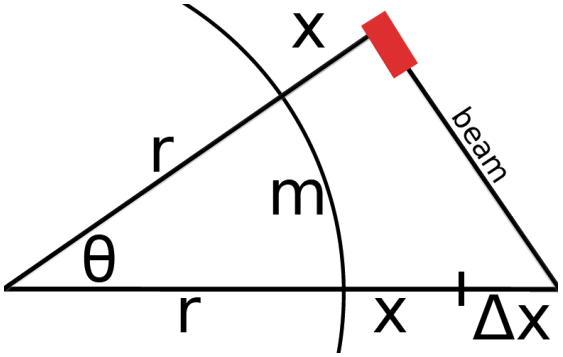


Figure A.1: Mast model.

- The twist in the mast is small enough that its effect on the bend of the mast is not worth calculating, which is to say it can be interpreted as one of the benches at either end having been rotated about the mast.
- The mast bend in x and y dimensions can be interpreted as separate arcs
- The optical axis from the optics modules are exactly parallel to the mast when there is no bend.
- Over these small angles, second order approximations of cosine are acceptable.

Figure A.1 illustrates the mast model geometry. The coordinate system in this image and for the remainder of this document is aligned such that z points up, and y points into the paper. The mast is represented by the arc labelled 'm' , which has length $m = \theta r$. The red square represents the optic and the vertices labelled 'beam' the optical axis of the optic. In the case of no bend, where $\theta = 0$, the optical axis would intersect at r+x. Because of the mast bending, the axis now intersects at r+x+Δx. In addition, the mast also rotates around its own axis by θ_z not illustrated in the figure.

First, to align the coordinate of the mast with the focal plane properly, the distortion of Δx and Δy must be modified by θ_z:

$$r = \sqrt{x^2 + y^2} \tag{A.1}$$

$$\psi = \arctan\left(\frac{\Delta x}{\Delta y}\right) \tag{A.2}$$

$$\Delta Y = r\sin(\psi - \theta_z) \tag{A.3}$$

$$\Delta X = r\cos(\psi - \theta_z) \tag{A.4}$$

$$\tag{A.5}$$

Let m be the length of the mast itself. The arc angle, θ, and the arc radius, r, can be determined as follows:

The mast is essentially an arc (of radius r) with the optics module extending out by a distance x. The angle, θ, of the arc can be found from:

$$\cos(\theta) = \frac{r + x}{r + x + \Delta x}$$

A general rule of arcs is:

$$m = \theta r$$

So:

$$\cos(\theta) = \frac{\frac{m}{\theta} + x}{\frac{m}{\theta} + x + \Delta x}$$

This is not solvable analytically, so here we shift to a second order approximation of cos, which should be accurate for these small angles:

$$1 - \frac{\theta^2}{2} \approx \frac{\frac{m}{\theta} + x}{\frac{m}{\theta} + x + \Delta x}$$

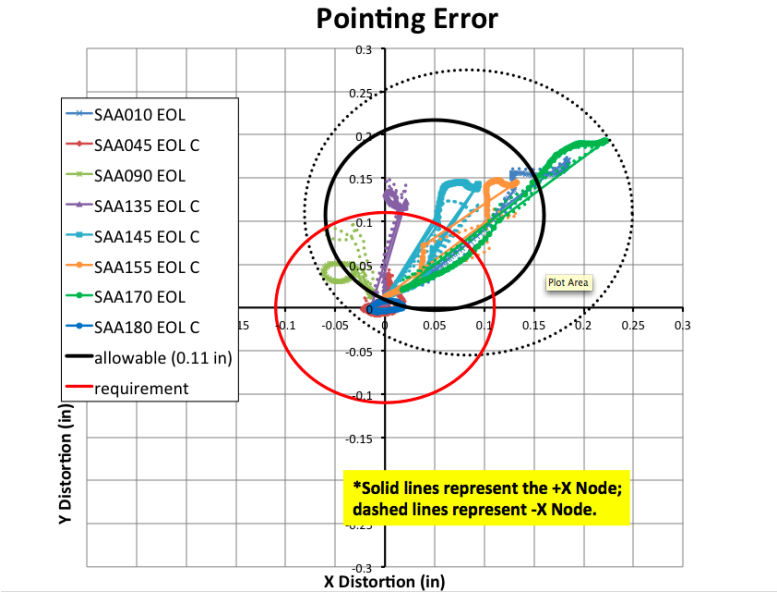


Figure A.2: Simulated Thermal scenarios.

For which the solutions for θ are:

$$\theta = \frac{-m \pm \sqrt{m^2 + 8x\Delta x + 8\Delta x^2}}{2\left(x + \Delta x\right)}$$

Given that when $\Delta x = 0$, $\theta = 0$, the correct solution is:

$$\theta = \frac{-m + \sqrt{m^2 + 8x\Delta x + 8\Delta x^2}}{2\left(x + \Delta x\right)}$$

Which does indeed grow as expected as Δx grows, so it seems a reasonable solution.

Given θ, and $r = \frac{m}{\theta}$, It should be clear that the optics bench will move by $r\left(1 - \cos\left(\theta\right)\right)$ along the x axis and $r\sin\left(\theta\right) - m$ along the z axis.

The calculated bend of the mast by a displacement Δx, is a rotation around the y-axis and vice versa.

A.1.2 Database creation

We have converted three thermal scenarios into databases: SAA90, SAA135 and SAA170. Figure A.2 shows the mast distortions of several solar angles, and SAA90 is considered a 'good case', SAA135 a "conservative case', and SAA170 a 'very bad case'. SAA90 and SAA135 have distortions within the allowed range, while SAA170 is slightly larger than the allowable distortion.

Figures A.3, A.4 and A.5 show the footprint of the optical axis on the focal plane for the three thermal scenarios. The red diamonds are the simulated thermal distortions

$(\Delta x, \Delta y)$ which are used in the formulas presented above to derive the transformation between the benches, R_{fbob}. The black stars are the result of ray-tracing the optical axis using the obtained transformation R_{fbob}. The agreement is good and tolerable considering the approximations that went into the derivation, and we consider this a good enough representation of the mast bend for use in NuSIM.

A.2 NuSIM results

The challenge for NuSIM is to reconstruct the pointing when using the mast bending databases as input. The NuSIM aspect reconstruction is insensitive to relative bend of the two benches, and it is therefore important to discover how well a pointing can be reconstructed.

Figures A.6, A.7 and A.8 show the reconstructed Ra and Dec of the pointing. In running this test we used perfect optics without scattering included or any error terms in the sensors. The error, as can be seen, is very small, and in all cases less than the width of a pixel (12").

A.3 Comparison with external code

Even though the error is small it is important to understand its source and so we compared the reconstructed transformation, R_{fbob}, which lacks information about rotations around x- and y-axis, with the original, R_{true}. Figures A.9, A.10 and A.11 show the optical axis ray-traced with the true, R_{true}, in red diamonds and reconstructed, R_{recon}, in black dots. The error here is due to the fact that the reconstructed transformation only has a rotation around the z-axis, and has replaced the rotations around x-, y-axis with translations. The error in the pointing is thus due to the slight distortion in the footprint which can now be seen to be less than a pixel width (0.6mm).

A.4 Conclusion

The NuSIM aspect reconstruction can not perfectly reconstruct the mast pointing due to the fact that it cannot tell a rotation of the plane from a translation of the plane. However, the errors are very small and within the budget. The mast bend databases are therefore considered valid and the aspect reconstruction of the mast movement understood, acceptable and thus verified.

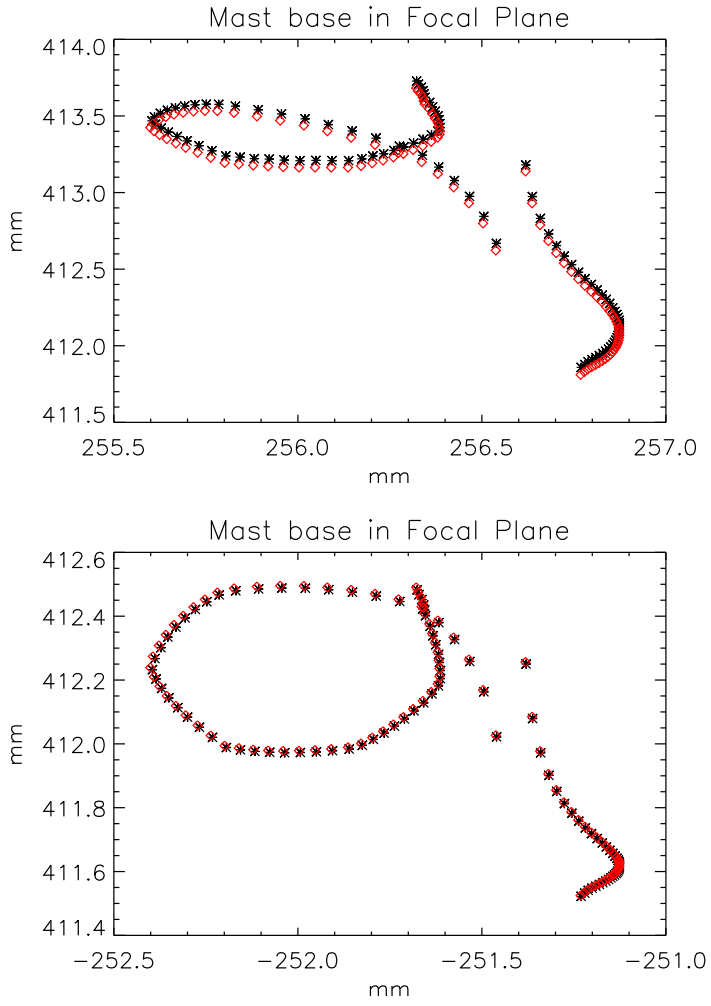


Figure A.3: SAA90 thermal mast bend distortions. Coordinates are in focal plane. Top: distortion footprint of module 1. Bottom: distortion footprint of module 2. Black stars are the ray-traced intersections of the optical axis using the transformation of the benches obtained from the thermal mast bend database which are plotted in red diamonds.

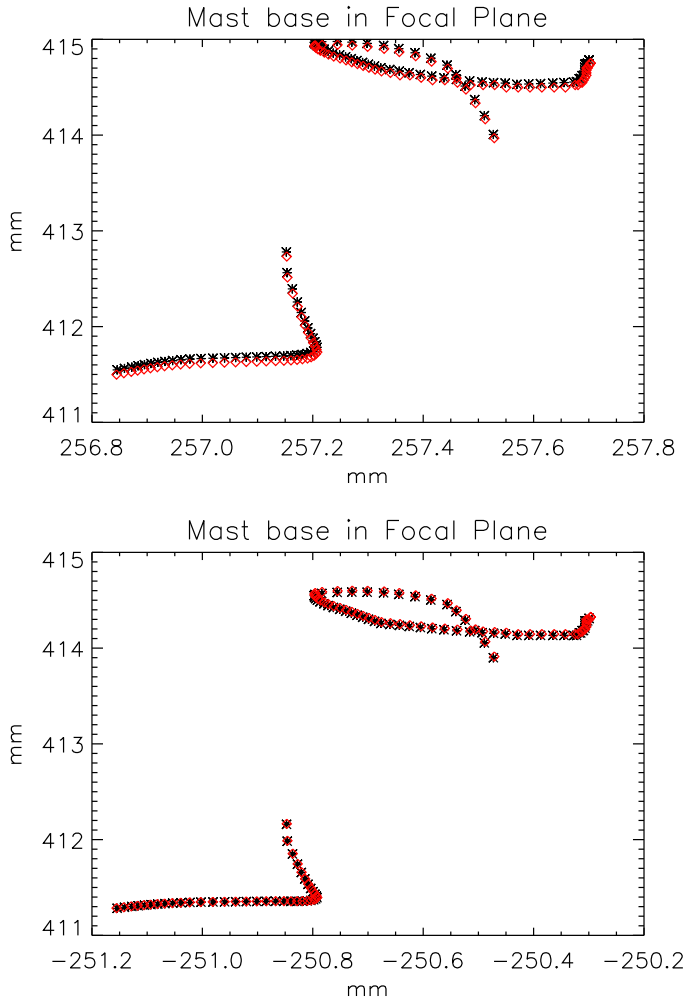


Figure A.4: SAA135 thermal mast bend distortions. Coordinates are in focal plane. Top: distortion footprint of module 1. Bottom: distortion footprint of module 2. Black stars are the ray-traced intersections of the optical axis using the transformation of the benches obtained from the thermal mast bend database which are plotted in red diamonds.

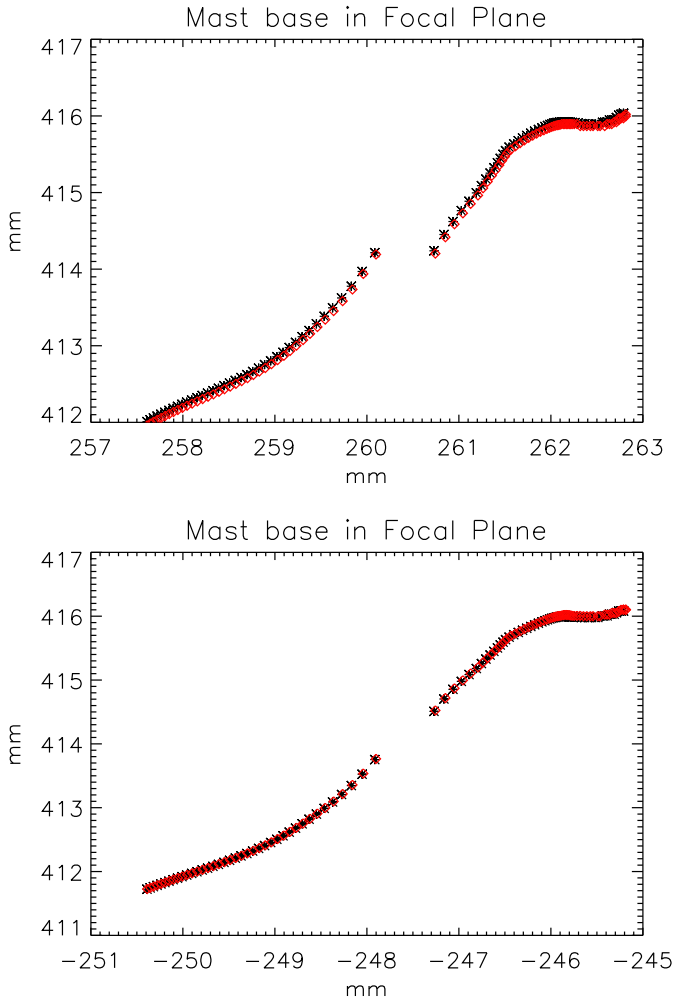


Figure A.5: SAA170 thermal mast bend distortions. Coordinates are in focal plane. Top: distortion footprint of module 1. Bottom: distortion footprint of module 2. Black stars are the ray-traced intersections of the optical axis using the transformation of the benches obtained from the thermal mast bend database which are plotted in red diamonds.

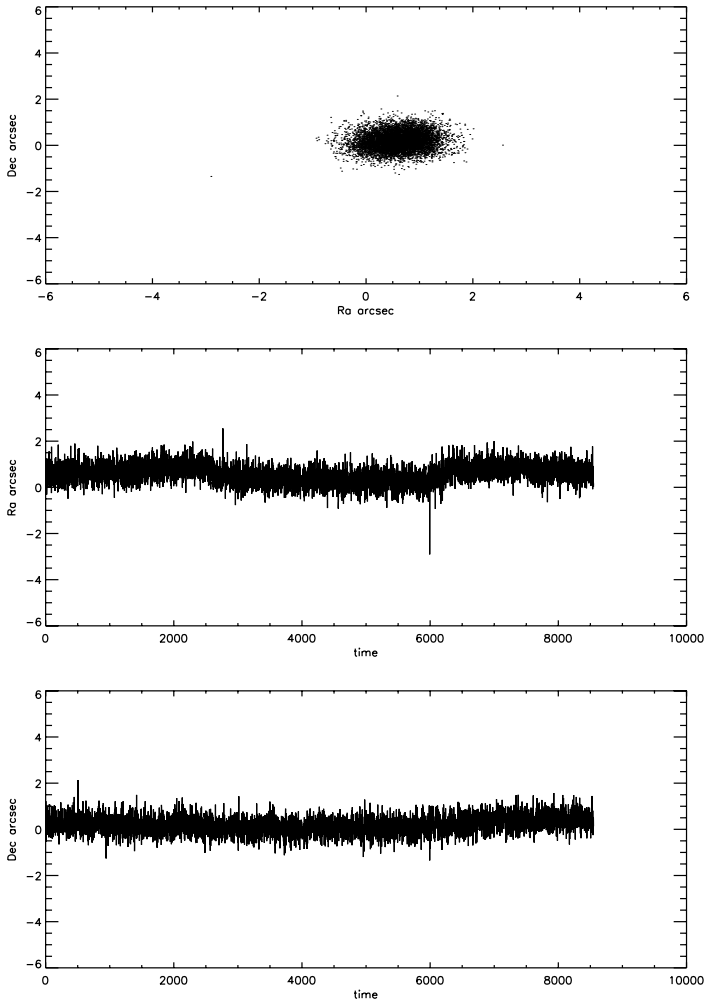


Figure A.6: Aspect reconstruction of SAA90 thermal mast bend. Top: Ra and Dec error. Source pointing is at RA,DEC=0,0. Middle: Ra error as a function of time. Bottom: Declination error as a function of time.

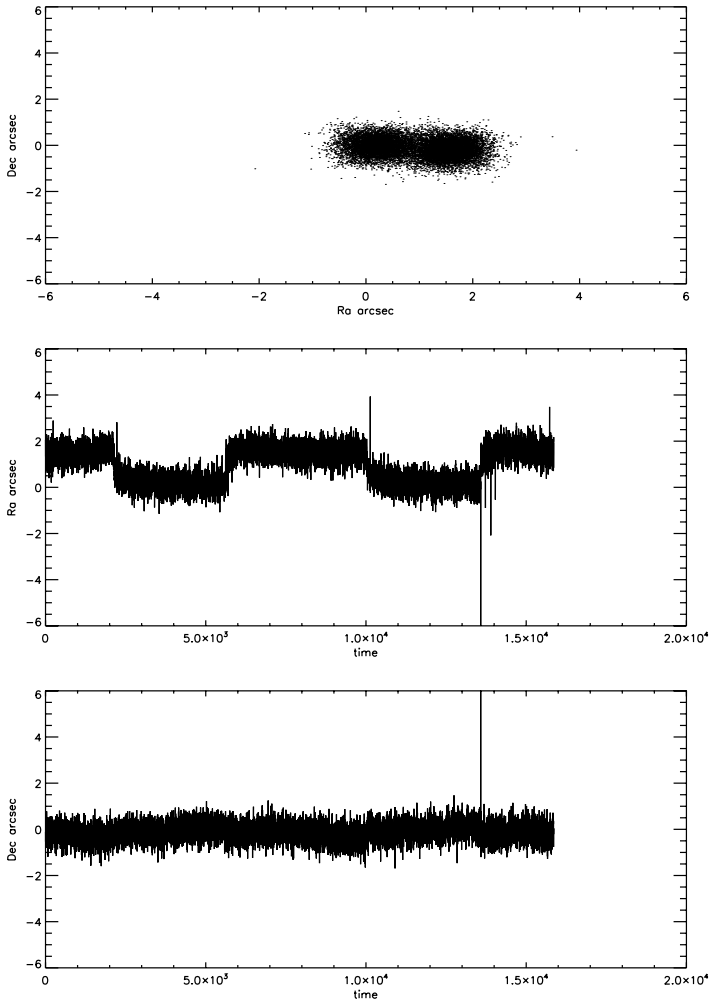


Figure A.7: Aspect reconstruction of SAA135 thermal mast bend. Top: Ra and Dec error. Source pointing is at RA,DEC=0,0. Middle: Ra error as a function of time. Bottom: Declination error as a function of time.

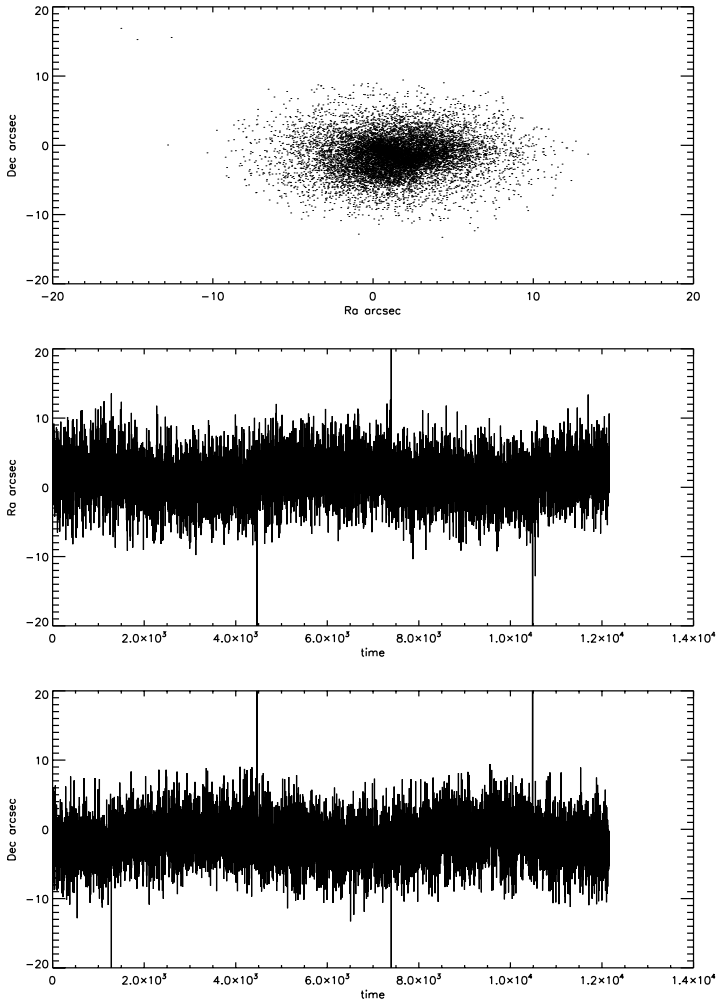


Figure A.8: Aspect reconstruction of SAA170 thermal mast bend. Top: Ra and Dec error. Source pointing is at RA,DEC=0,0. Middle: Ra error as a function of time. Bottom: Declination error as a function of time.

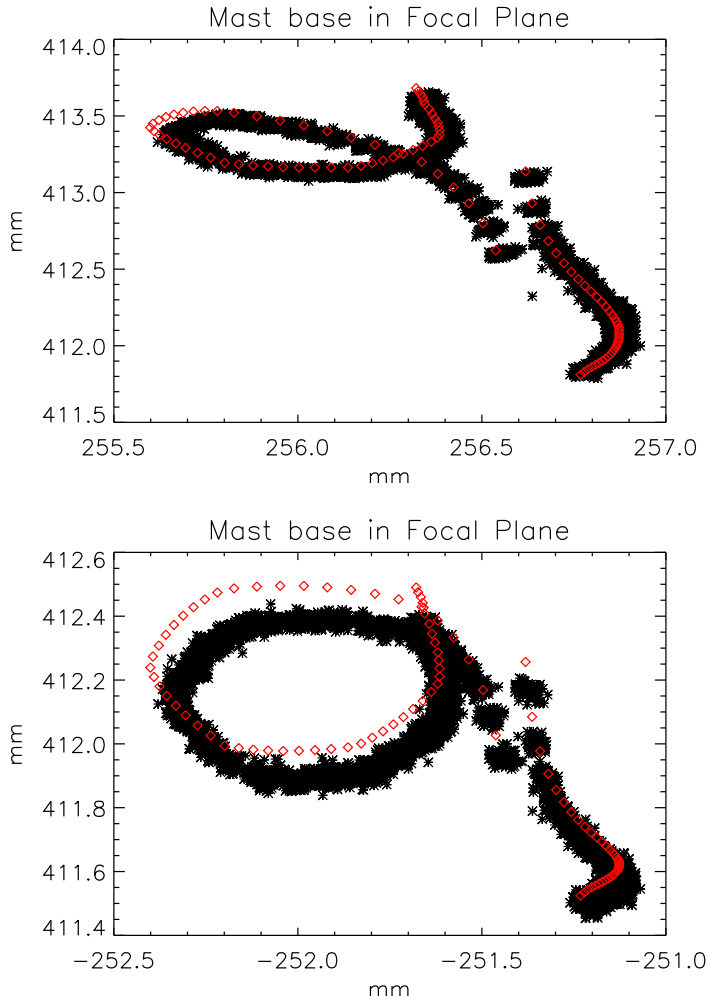


Figure A.9: SAA90 thermal mast bend distortions. Coordinates are in focal plane. Top: distortion footprint of module 1. Bottom: distortion footprint of module 2. Black stars are the ray-traced intersections of the optical axis using the reconstructed transformation of the benches obtained from the NuSIM. The original transformation is plotted in red diamonds.

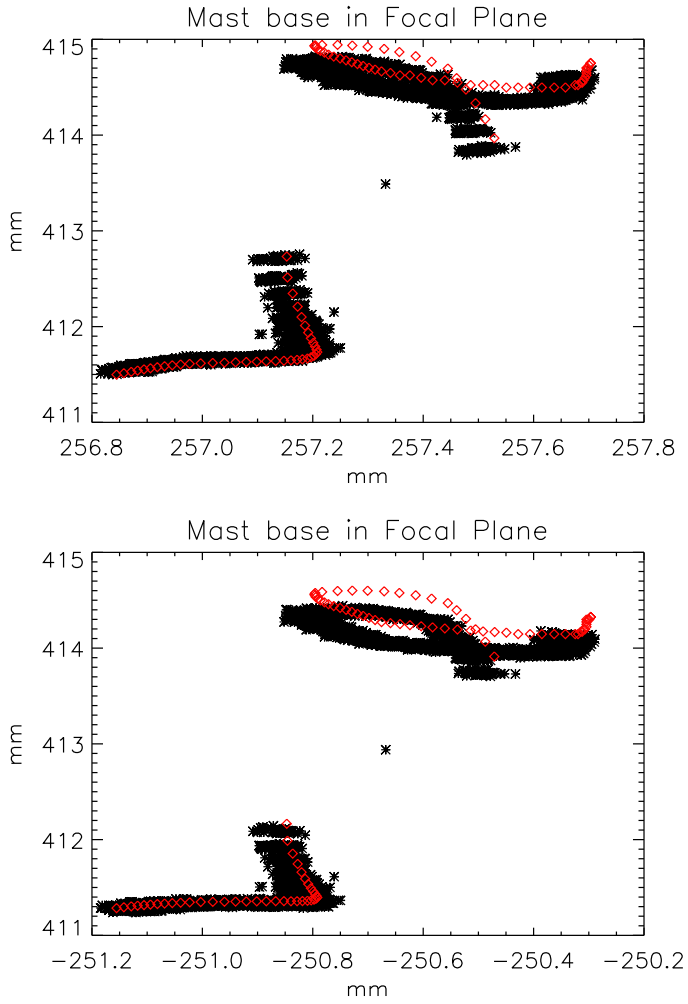


Figure A.10: SAA135 thermal mast bend distortions. Coordinates are in focal plane. Top: distortion footprint of module 1. Bottom: distortion footprint of module 2. Black stars are the ray-traced intersections of the optical axis using the reconstructed transformation of the benches obtained from the NuSIM. The original transformation is plotted in red diamonds.

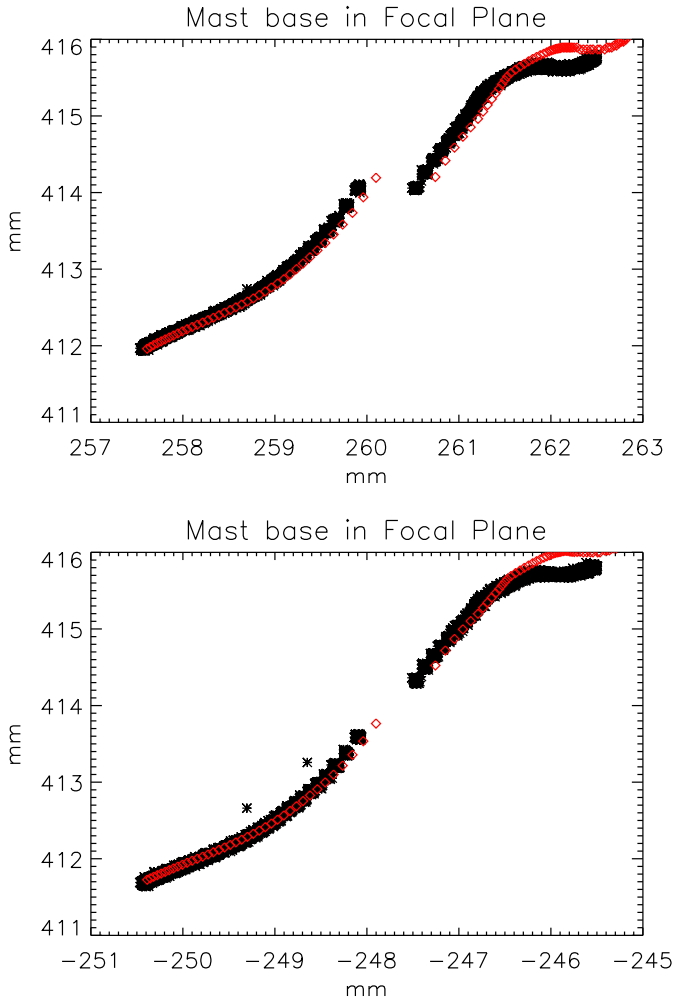


Figure A.11: SAA170 thermal mast bend distortions. Coordinates are in focal plane. Top: distortion footprint of module 1. Bottom: distortion footprint of module 2. Black stars are the ray-traced intersections of the optical axis using the reconstructed transformation of the benches obtained from the NuSIM. The original transformation is plotted in red diamonds.

Appendix B

NuSim Aspect Reconstruction Verification Document

B.1 Purpose

The purpose of this memo is to document the stability and error of the aspect reconstruction algorithm used in NuSim. Details on the mathematics of the algorithm can be found in the NuSim manual.

The aspect reconstruction algorithm relies on the metrology system and the star tracker to solve for the motion between the Focal Plane Bench and the Optics Bench. Any motion which keeps the planes of the two benches parallel can be fully solved. Tilting of the benches with respect to each other is, however, interpreted as a linear displacement. This causes an error in the solution and it is this error we will quantify here.

The results shown here were made using NuSim revision 174.

B.2 Test description

In order to probe the stability of the aspect reconstruction several tests were run using five different alignment databases. The databases are all based on the version NuSIM_OrientationsIDEAL_007. The databases used were:

- NuSIM_OrientationsDB_007PL: Purely linear displacement of the benches.
- NuSIM_OrientationsDB_007Rot: Pure tilt of the benches.
- SAA90DB_007: Solar Azimuth Angle 90 degree mast bend.
- SAA135DB_007: Solar Azimuth Angle 135 degree mast bend.
- SAA170DB_007: Solar Azimuth Angle 170 degree mast bend.

In revision 174 NuSim has an error when the source is at (Ra,Dec)=(0,0) degrees, therefore tests were run using a point source placed on-axis at (Ra,Dec)=(0.1,0.1) degrees. At this elevation the cosine scaling effect of the azimuth is essentially = 1. The detectors were shifted linearly so the spot did not fall on one of the detector gaps.

In the OpticsEngine scattering was turned off and ghostrays turned off, and the perfect optics option were used instead. The perfect optic option modifies the path of the rays such that the on-axis spot hitting the focal plane at exactly the focal length is a perfect spot. This is in reality an unphysical situation since the NuSTAR optics are conic approximations to a Wolter I and perfect focus is not possible, but in order to eliminate any blurring due to the optics we use this setting. The measurement error terms of the metrology and star tracker engines are disabled as well.

B.3 Results

The results from each database run are shown in plots of four. The top panel of the plots shows the residual between the aspect reconstructed source and the actual source input position in arcseconds as a function of time. As explained above we are using a perfect optic, but as will be shown in the linear case the residuals have a 'natural' width. The reason for the width is due to the fact that an event is not deposited at the top of the detector, but typically gets deposited somewhere inside the detector strata. This depth effect causes the spot to diverge, since it was designed to focus at the top of the detector. The natural width of the spot is therefore 0.1 arcseconds.

The two bottom plots show the difference between the location of the optical axis on the sky and the input source position. Whenever there are tilts involved, the optical axis will be offset from the source. These plots trace the amount of tilting in the system.

B.3.1 Purely linear displacement

As stated above, a pure linear displacement of the benches can be exactly solved. This is shown in Figure B.1 which just displays the natural widening of the spot due to the depth effect.

B.3.2 Pure tilt of the benches

The "tilted" database has a rate of 1"/second around the X-axis. Figure B.2 bottom panels shows the tilt of the benches with respect to the celestial coordinates, and had the source been located at exactly Ra,Dec = (0,0) the motion would have been exclusively sensed in Ra. Since the source is at (0.1,0.1) there is a slight mixing of angles, and some of the motion is therefore sensed in the declination.

The top panels show the error due to the tilting. For Ra it can be seen that approximately 100 arcseconds tilt gives rise to a 1 arcsecond error in the aspect reconstruction. As a reference the average angular size of the pixel at this elevation is 12 arcseconds. The declination plot shows the effect of the bench moving away from the focus plane

due to the tilt. Away from the plane of focus the spot of the source becomes a doughnut and that is the reason for the branching.

B.3.3 Mast bending

Figure B.3, B.4 and B.5 show the results of using the mast bending databases. Just as for the situation with a pure tilt, the bottom plots track the magnitude of the tilt, and that tilt is mirrored by roughly a 1/100 residual error in the top plots.

B.4 Conclusion

The residuals of the tilting of the benches is shown to be approximately 1" per 100" tilt. In principle the difference between the optical axis and the source can be used to correct the error, however, this requires precise knowledge of the spacecraft bus movement, since any rotation of the entire ridged space craft structure will add to the optical axis path and must be backed out before applying the correction.

The mast database test shows that the residuals even for SAA170, are only 2 arc-seconds, which is less than 1/5th of a pixel.

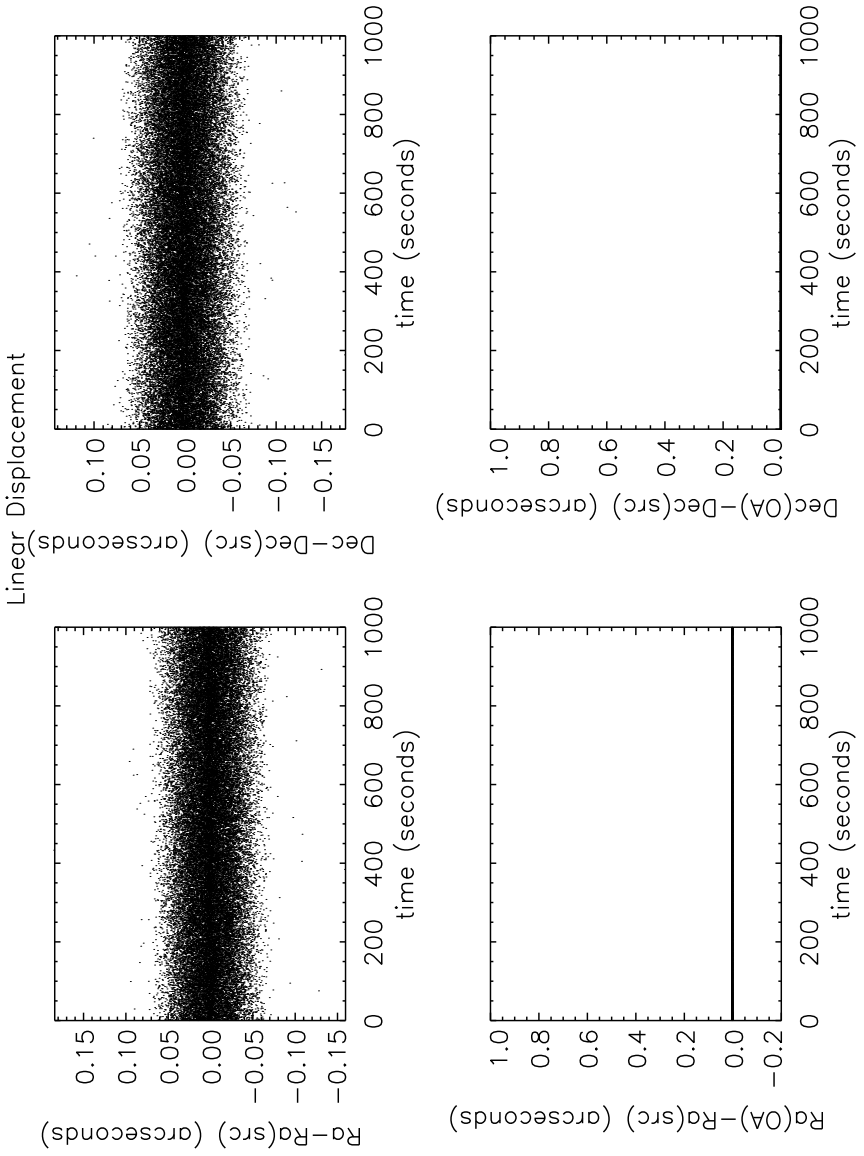


Figure B.1: Aspect reconstruction of a linear displacement of the benches. Top panel shows the residual of the reconstruction source position and the actual source position. Bottom panels shows the offset between the source position and the optical axis.

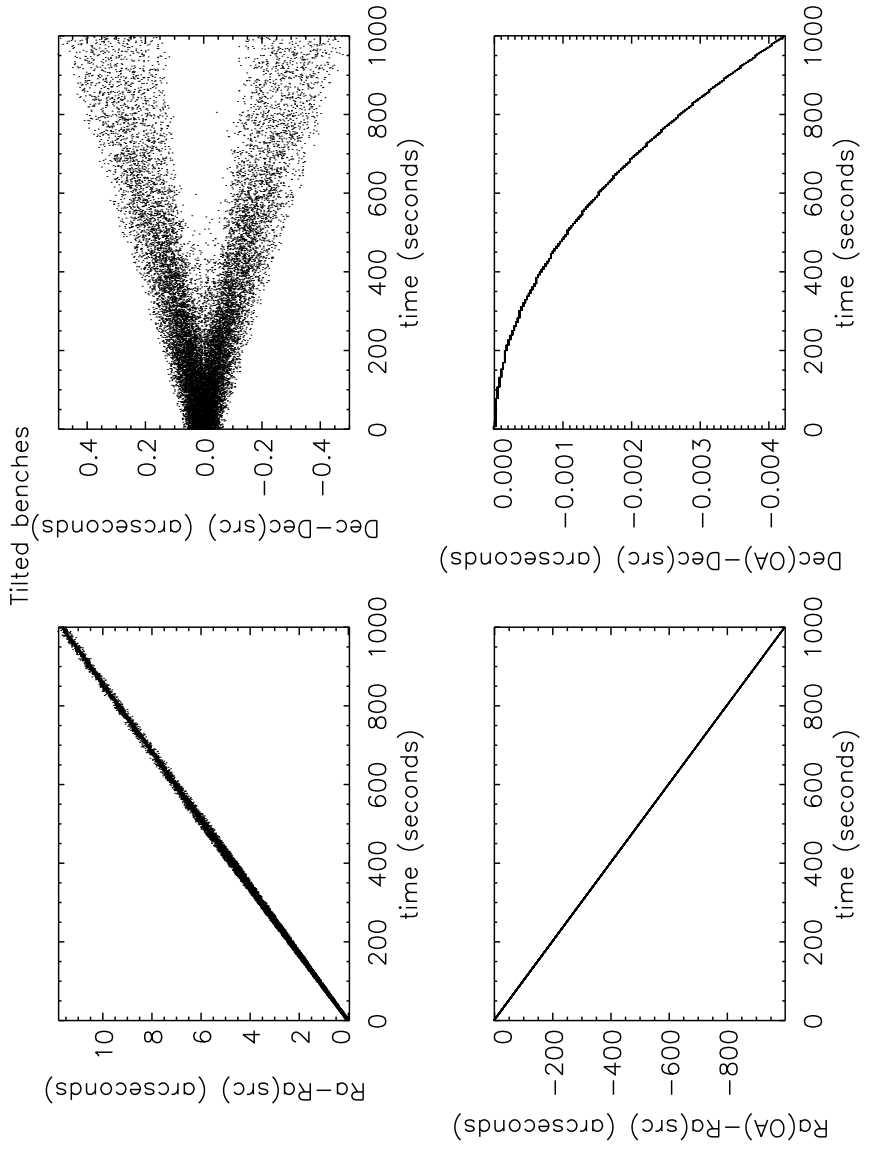


Figure B.2: Aspect reconstruction of a pure tilt between the two benches. The rate of tilt is 1"/second. Top panel shows the residual of the reconstruction source position and the actual source position. Bottom panels shows the offset between the source position and the optical axis.

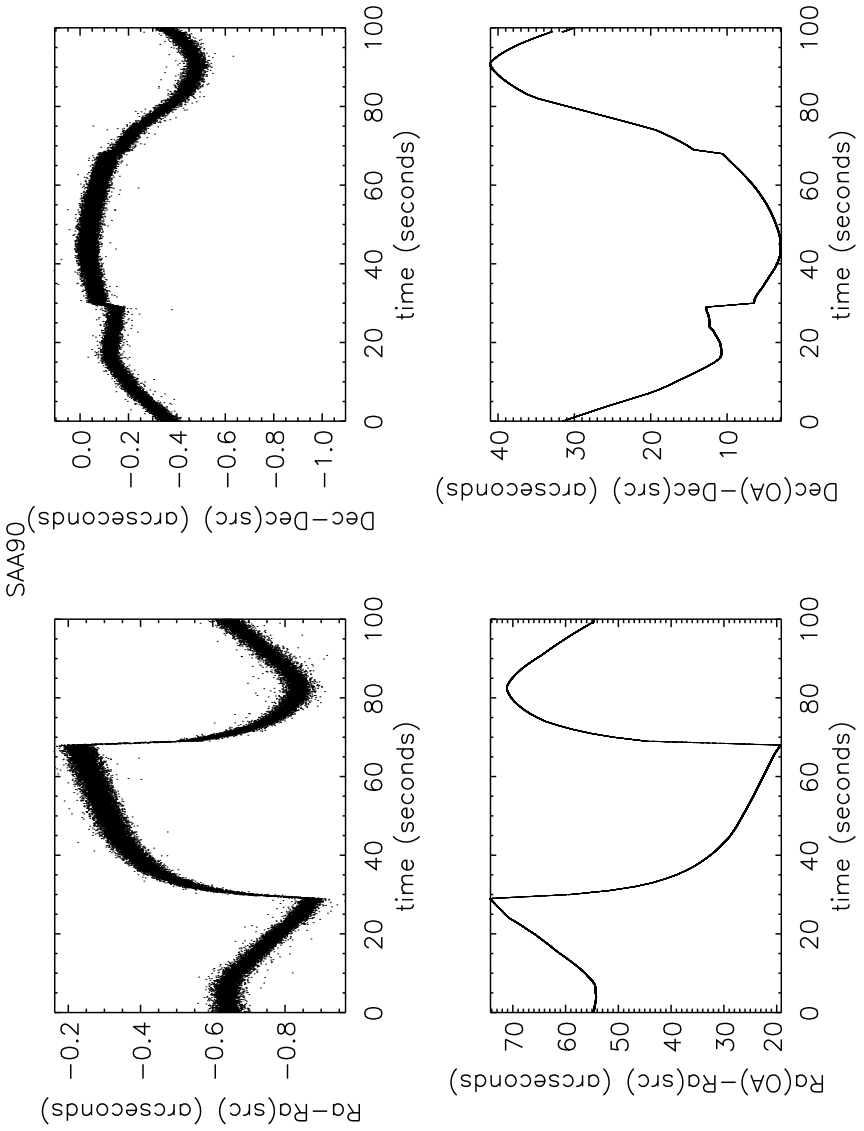


Figure B.3: Aspect reconstruction of mast bend database: mastDB90. Top panel shows the residual of the reconstruction source position and the actual source position. Bottom panels shows the offset between the source position and the optical axis.

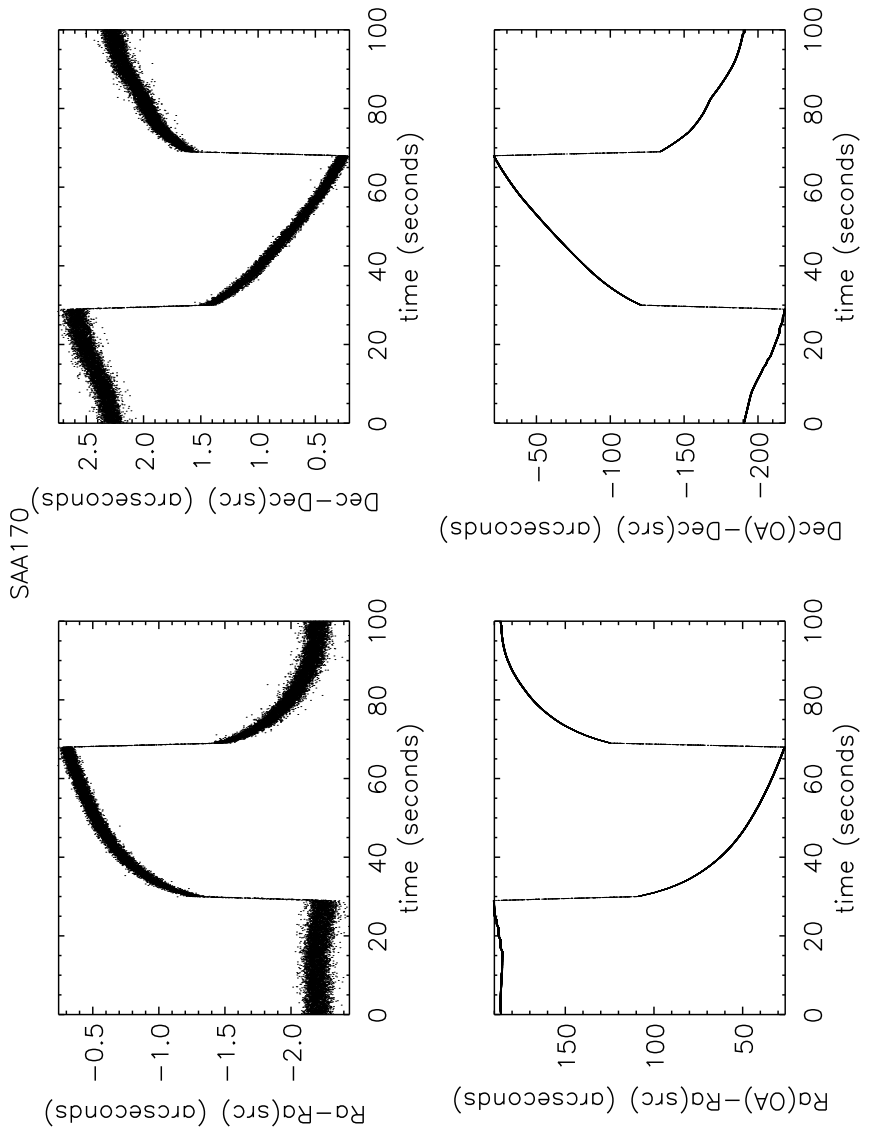


Figure B.4: Aspect reconstruction of mast bend database: mastDB170.Top panel shows the residual of the reconstruction source position and the actual source position. Bottom panels shows the offset between the source position and the optical axis.

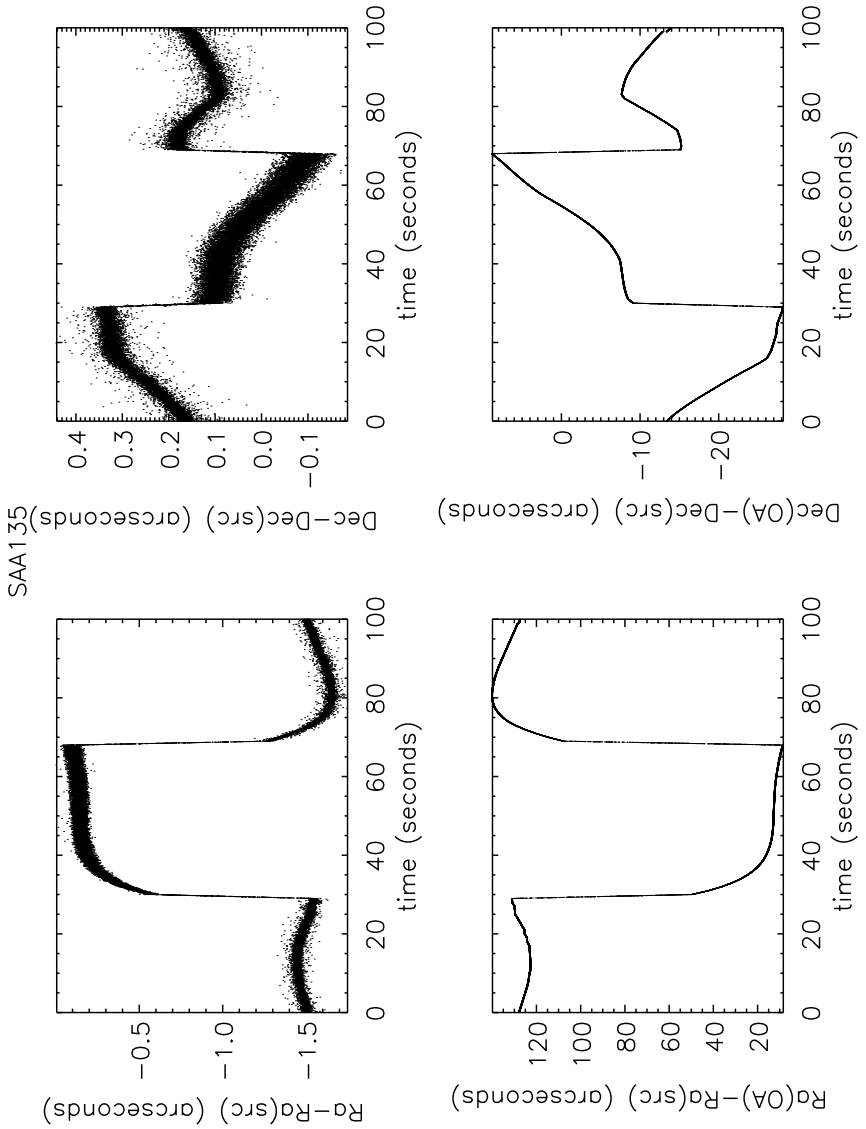


Figure B.5: Aspect reconstruction of mast bend database: mastDB90. Top panel shows the residual of the reconstruction source position and the actual source position. Bottom panels shows the offset between the source position and the optical axis.

Appendix C

NuSim - Raytrace, MLI and Aperture stop verification

C.1 Purpose

This document presents the cross verification of the Raytrace module in NuSim contained in NModuleOpticsEngine.cxx and NModuleApertureStopTrivial.cxx . The cross verification consists of checking number counts and effective area against an external raytrace called "ctrace" from which the NuSIM raytrace originated. These tests were run on NuSim revision 171.

C.2 Test Description

NuSim was run with a point source at various monochromatic energies and angular positions. The optics engine module had scattering enabled with a value of 6e-5, and ghostrays enabled. The remaining settings in NuSim have no influence on the obtained results since the results are extracted before they enter into the detector modules. The exact same geometric and reflectivity files were used as input into NuSim and ctrace.

The test was run for 3 different angular positions: on-axis, 3 arcmin off-axis and 12 arcmin off-axis. For the on-axis test energies [5,10,20,30,50,70,75] keV were used, and for off-axis tests only [10,30,70] keV.

C.3 Results

There are three different effective areas quoted. The effective area without aperture stop is the effective right after the photons leave the optics. No MLI is included. The effective area with aperture is the area after the aperture has clipped the photons. The effective are with MLI is the area after the photons exit the optic, but with the incoming photons attenuated by the MLI.

Each test was run for a variable length of time and therefore the results are presented with the total number of input counts used for the simulation. For example when quoting

the number of photons rejected by the aperture stop the first number is the photons rejected and the second the total number of incoming photons after the thermal cover has attenuated the incoming photon flux: (photons rejected/total incoming photons).

The number of ghost rays from the upper mirror and lower mirror are quoted in the same way as the aperture clipping.

The area and numbers the the following tables have been double checked against the external raytrace "ctrace". These numbers can be used to check the state of the code.

C.3.1 On-axis

Tables C.1 and C.2 show the on-axis results.

Table C.1: On-axis Effective Area

Energy (keV)	Effective Area (cm^2) w/o App	Effective Area (cm^2) w App	Effective Area (cm^2) w MLI
5	459.56± 0.97	466.89± 0.95	390.13± 0.82
10	452.45± 0.74	440.74± 0.74	435.77±0.72
20	236.06± 0.53	227.08± 0.52	230.67± 0.52
30	158.41± 0.32	151.08± 0.32	155.11± 0.32
50	82.64± 0.38	77.47± 0.37	80.99± 0.37
70	38.82± 0.29	36.12± 0.28	38.04± 0.28
75	33.97± 0.17	31.54± 0.16	33.30± 0.17

Table C.2: On-axis Raytrace statistics

Energy (keV)	# Photons App Clipped	# Photons Upper Ghost	# Photons Lower Ghost
5	6183/519070	6254/519070	0/519070
10	9902/864100	10010/864100	0/864100
20	7436/880917	7551/880917	0/880917
30	10814/1566742	11011/1566742	0/1566742
50	2869/591172	2941/591172	0/591172
70	1241/489545	1298/489545	0/489545
75	2668/1167497	2788/1167497	0/1167497

C.3.2 Off-axis results

Tables C.3 and C.4 show the 3 arcmin off-axis results. Tables C.5 and C.6 show the 3 arcmin off-axis results.

Table C.3: 3 arcmin off-axis Effective Area

Energy (keV)	Effective Area (cm^2) w/o App	Effective Area (cm^2) w App	Effective Area (cm^2) w MLI
10	435.45± 1.80	357.86± 1.63	419.26± 1.73
30	147.33± 0.56	111.51± 0.48	144.26± 0.55
70	28.72± 0.29	21.26± 0.25	28.14± 0.28

Table C.4: 3 arcmin off-axis Raytrace statistics

Energy (keV)	# Photons App Clipped	# Photons Upper Ghost	# Photons Lower Ghost
10	10365/142083	11010/142083	0/142083
30	16702/496019	18567/496019	0/496019
70	2491/355154	3240/355154	0/355154

C.3.3 MLI results

Table C.7 shows the results for the thermal cover transmission. The second column are numbers taken from the input file, and the third the NuSim calculated MLI transmission.

Table C.5: 12 arcmin off-axis Effective Area

Energy (keV)	Effective Area (cm^2) w/o App	Effective Area (cm^2) w App	Effective Area (cm^2) w MLI
10	241.17± 0.75	40.83± 0.31	232.01± 0.72
30	61.29± 0.25	12.23± 0.11	59.99± 0.24
70	9.10± 0.13	2.65± 0.07	8.91± 0.13

Table C.6: 12 arcmin off-axis Raytrace statistics

Energy (keV)	# Photons App Clipped	# Photons Upper Ghost	# Photons Lower Ghost
10	84729/449866	44481/449866	17763/449866
30	46890/1016683	23631/1016683	11888/1016683
70	2995/494528	944/494528	1276/494528

Table C.7: MLI

Energy (keV)	MLI transmission input	MLI transmission NuSIM
5	0.849	0.847
10	0.962	0.962
20	0.977	0.977
30	0.979	0.979
50	0.980	0.980
70	0.980	0.980
75	0.980	0.980