CORSIKA and sim_telarray – Simulation of the imaging atmospheric Cherenkov technique

Konrad Bernlöhr

May 30, 2001
Last revised: December 21, 2023

Contents

List of Figures

Chapter 1

Introduction

Energetic cosmic-ray particles and γ-rays entering the atmosphere initiate particle cascades which are called *extensive air showers* (at least in that energy domain where shower particles reach ground level). Shower particles having a velocity faster than the speed of light in the medium air emit Cherenkov light. The imaging atmospheric Cherenkov technique uses large optical telescopes and fast imaging devices (at present: cameras of many photomultipliers) to detect the showers by their Cherenkov light. If you are not familiar with the measurement technique, read introductory material like [1] before continuing with this manual.

For the simulation of the Cherenkov light emission several program packages are available to simulate air showers. This document describes only one of them, the CORSIKA[1] program [2]. Due to its public availability, due to its rich set of interaction model options and due to the fact that it is used and tested by a large number of physicists, CORSIKA was chosen as the basis of the air shower simulation. Current simulations are using CORSIKA version 6.0 and up (frequently used in productions: 6.990, lastest supported version so far: 7.7400) which includes a number of features added by the author of this note to CORSIKA versions 5.7 through 5.945.

The original CORSIKA Cherenkov option – developed mainly at the University of Madrid – was aimed at the simulation of the AIROBICC array of non-imaging detectors and requires a rectangular grid of rectangular detectors in a horizontal plane. The Cherenkov part in CORSIKA was completely revised and a new output interface was developed, which is activated with the IACT option.

With the IACT option[7], the detector configuration is a 3-dimensional collection of spheres. For efficiency reasons, an approximation is used: only Cherenkov light within a given angle (more than 10°) from the shower axis would be guaranteed to hit detectors out of the 'detection level' (as used in CORSIKA). For the anticipated purpose this should not be a problem at all. Photon bunches are now recorded if they pass within a specified radius from a detector (telescope) position. The scheme is illustrated in Fig. 1.1. Since the time required for this part of the simulation is negligible compared to CORSIKA itself, a given array of telescopes is not only simulated once but many times with (horizontal) random

[1] CORSIKA is provided by D. Heck and T. Pierog at Karlsruhe Institute of Technology (KIT).

displacements of each array. In contrast to the AIROBICC-style Cherenkov option where a regular grid of detectors could be displaced with fixed (kind of quasi-random) offsets, the IACT Cherenkov option generates different pseudo-random offsets for each shower. This way, showers with cores quite far from the telescope array can be easily taken into account.

For recording the photon bunches and other information not the old-fashioned (operating system, CPU type, and compiler dependent) CORSIKA data format is used but the `eventio` [4] format already used for all CRT data and the raw and DST data of the HEGRA telescope system. Item types 1200 to 1210 are defined for that purpose. The CORSIKA run and event header and trailer blocks are also included. These give all the necessary details about the simulated showers. Instead of writing all photon bunches as they emerge from CORSIKA, all bunches are sorted by telescope and array. They are written to the output file only after the simulation of a complete shower. This way, the detector simulation needs to treat only one telescope at a time.

The second step, the simulation of the detector response, is done by a second program called `sim_telarray` [5] [2] (for historical reasons also called `sim_hessarray`). This program was originally developed for simulating the HEGRA telescope system and then adapted to H.E.S.S. by making it much more configurable (mainly at run-time but some limits related to the amount of memory used need to be configured at compile-time). Where the term `sim_hessarray` is used in this note, it denotes the H.E.S.S.-specific variant of the program. Apart from improved flexibility by moving hard-coded things into more and more configuration data files, the conversion to output data had to be written from scratch. The output, apart from additional Monte Carlo data, includes (in a different format) all kinds of data which can come from the real H.E.S.S. array after merging data blocks from different computers. Depending on configuration options the output may also include types of data not produced by H.E.S.S. telescopes, including things anticipated with future instruments like CTA, for which many extensions have been added.

[2] `Sim_telarray` is available from this web page and with additional material from restricted areas for H.E.S.S. or CTA

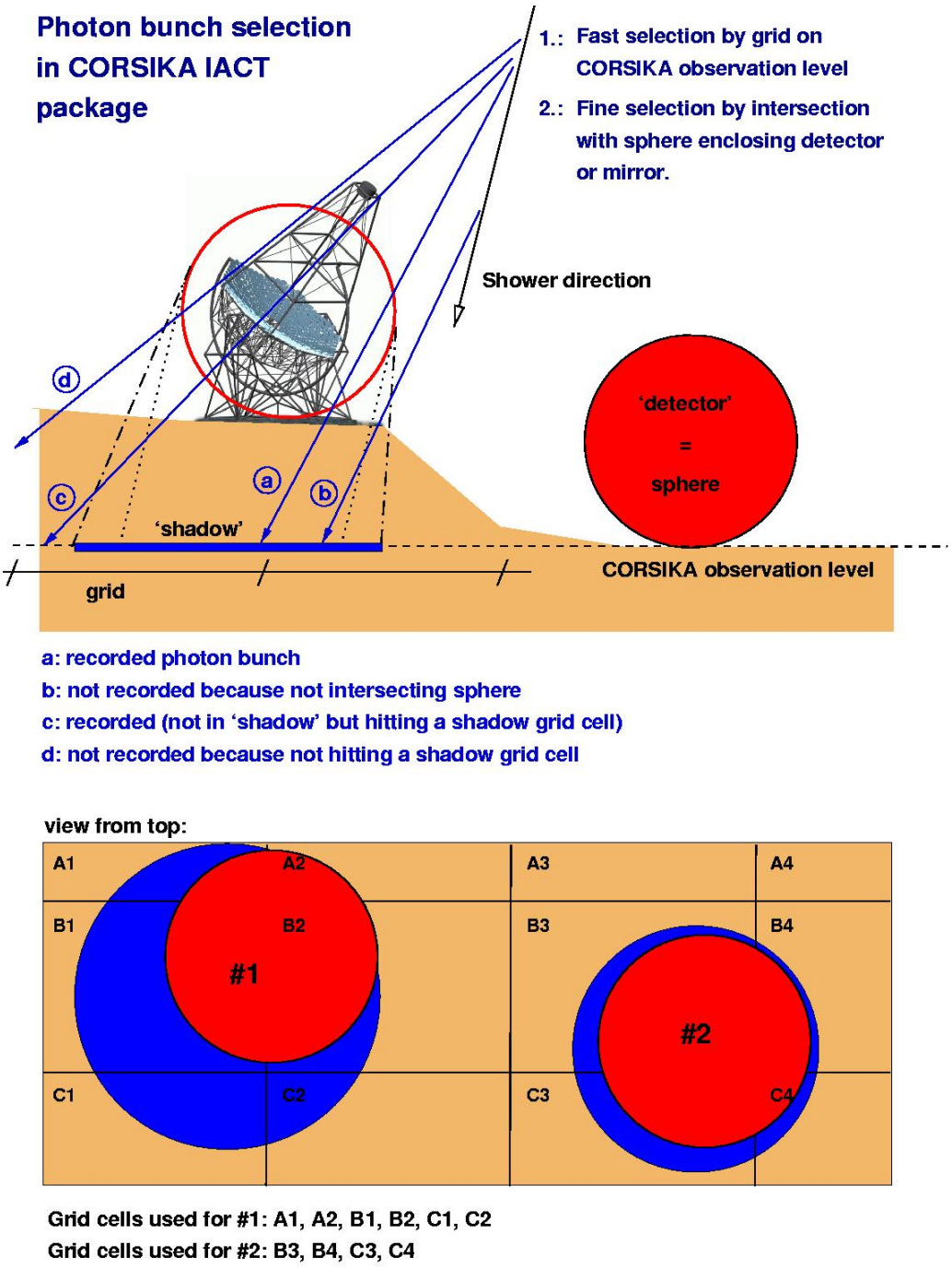

Figure 1.1: Scheme of selection of photon bunches for output in the CORSIKA IACT package.

Chapter 2

Usage of CORSIKA 6 and 7 with IACT option

2.1 Compiling CORSIKA versions 6.0 to 6.2xx

First of all, you need to process the modified CORSIKA source code through CMZ[1] to extract the required code version for simulation of Cherenkov telescope arrays. For this purpose both the **CERENKOV** and **IACT** options have to be selected. Other options depend on the wanted interaction options and the machine architecture. The **VIEWCONE** option is often very useful for simulation of extended or diffuse emission. The **CEFFIC** option is not intended to be used together with later `sim_telarray` simulations although its output is compatible with the IACT option. Use the **cmz_extract** shell script for extraction, unless you are familiar with CMZ and the CORSIKA options. The compilation is best done with the supplied **Makefile**. The syntax is

```
make target
```

or, for the VENUS interaction model, simply

```
make
```

and should work on a range of systems, including Linux (x86), DEC Unix and HP-UX. Support for these old CORSIKA versions was dropped after IACT/ATMO package version 1.47. In addition, support for CMZ has been dropped by CERN many years ago (although it might still work if you happen to have a copy). The rest of this document does not refer to it anymore.

2.2 Compiling CORSIKA versions 6.5xx to 6.7xxx

After version 6.2xx CORSIKA is made available as a tar file, with build procedures added on top of GNU autoconf etc.

[1] The CMZ source code management system is no longer supported.

In contrast to even earlier versions (see above), the CORSIKA versions 6.500 to 6.7xx include the IACT/ATMO package. Version 1.34 of the latter, bernlohr-1.34.tar.gz, as included in CORSIKA 6.500, however has compatibility problems in the core offset correction. Make sure to use at least version 1.35. Assuming you want to install CORSIKA version 6.735 with the latest IACT/ATMO package (version 1.36 or newer recommended instead of 1.35; newer versions having additional features), you need

- corsika-6735.tar.gz (needs a patch to compile) and qgsdat-II-03.gz (for QGSJET-II only, has to be gunzipped)
- bernlohr-1.44.tar.gz (or any later version of the IACT/ATMO package)

Unpack them like:

```
tar zxvf corsika-6.735.tar.gz
cd corsika-6735/bernlohr
tar zxvf ../../bernlohr-1.44.tar.gz
cd ..
```

Use the corsika-install script for an interactive setup of CORSIKA options to be compiled in, or type the following to compile the default IACT/ATMEXT/VIEWCONE version:

```
bernlohr/gen_config
./corsika-install < /dev/null
```

For production use make sure to compile with optimization '-O2 -fomit-frame-pointer -march=generic' (or better '-march=native' if to be used on the same CPU type) or such rather than the '-O0 -fbounds-check -g' on which the corsika-install script tries to insist. Example (assuming bash as your shell, machine-specific options and available FORTRAN compiler may vary):

```
if [ "`uname -m`" = x86_64 ]; then \
 if [ `cat /proc/cpuinfo | grep Xeon | wc -l` = 0 ]; then \
 a=k8; else a=nocona; fi; \
 else a=i686; fi
opt="-O2 -fomit-frame-pointer -march=$a" # -m32 # on x86 add: -malign-
wrn="-Wall -Wunused -Wuninitialized"
export CFLAGS="$opt $wrn"
export CXXFLAGS="$CFLAGS"
export FFLAGS="$wrn $opt -fno-automatic" # no '-finit-local-zero' with
export CC=gcc; export CXX=g++; export CPP="gcc -E"
export F77="gfortran -std=legacy -frecord-marker=4" # or: export F77=g
CORSIKA_USER_COMP=1 ./corsika-install clean < /dev/null
CORSIKA_USER_COMP=1 ./corsika-install < /dev/null
```

For some interaction models (FLUKA, DPMJET, QGSJET-II, ???) the '-m32' flag was needed on 64-bit machines, at least when compiling with g77. Sometimes the reason for

that lies in record markers of binary data files which gfortran is able to handle also in 64-bit mode. To get rid of the -g option, you have to modify the configure script. Look for lines like

```
   CFLAGS="$CFLAGS -g"
   CXXFLAGS="$CXXFLAGS -g"
and
   FFLAGS="$FFLAGS -g"
```

But this has little impact on performance and is not normally done. You will find the CORSIKA binary in the run sub-directory (name depending on interaction model etc.)

For H.E.S.S. simulations you will also need the atmospheric profiles:

- atmprof10.dat, atmprof11.dat, and atmprof12.dat as well as
- the example INPUTS file INPUTS_example_for_HESS.

In the relevant source web pages for H.E.S.S. and CTA you may also find a full package with CORSIKA, recent IACT/ATMO package, sim_hessarray/sim_telarray, and the hessio library, complete and without needing any modules from the HESS source tree: corsika_simtelarray.tar.gz (well, depending on the interaction model you use, you may need the qgsdat-II-03.gz file as well).

2.3 Compiling CORSIKA versions 6.9xx to 7.7xxx

The CORSIKA versions 6.9xx and newer (at this time at least up to 7.7400) come with a configuration tool called 'coconut'. You can use that manually to compile specialized versions. Much easier and with the required compiler directives for IACT simulations should be the corsika_build_script script. Try:

```
tar zxvf corsika-6.990.tar.gz
cd corsika-6990
../corsika_build_script
```

For CTA, a source code package is available with additional build tools, where the complete installation of CORSIKA and `sim_telarray` can be as simple as

```
tar zxvf corsika_simtelarray.tar.gz
# Get or link the qgsdat-II-03 data file here
./build_all prod5 qgs2
```

The alternate package for the CORSIKA version 7 currently includes the qgsdat-II-04 file in its corsika-7.xxxx.tar.gz package. For automated build of CORSIKA 7.6xxx you need at least IACT/ATMO package version 1.52, for CORSIKA 7.7xxx you need at least IACT/ATMO package version 1.60 (or an adapted variant of 1.59).

2.4 Running CORSIKA

To run CORSIKA you will need a number of data files available, with the exact files depending on your interaction models. Some of the necessary files would be rebuild by CORSIKA if not present – but at a very high computational cost. In the absence of other files, CORSIKA would immediately exit. So you better check that you have all necessary files available, either from the CORSIKA tar package or as additional files from the CORSIKA download site.

If an atmospheric model other than the built-in model is used, you also need the corresponding one of the files

```
atmprof1.dat
atmprof2.dat
atmprof3.dat
atmprof4.dat
atmprof5.dat
atmprof6.dat
atmprof9.dat
atmprof10.dat
atmprof11.dat
atmprof12.dat
```

which hold the atmospheric profiles of MODTRAN [10] atmospheric models 1–6, the Antarctic model (9), and the Windhoek average (10) and seasonal extremes (11 and 12). If you created your own profile, give it a number above 20. For H.E.S.S., the models 10, 11, and 12 were built, based on balloon soundings from Windhoek. Different experimental groups (CTA, MAGIC, ...) have additional profiles for their sites, derived either from radiosonde measurements (plus extrapolation for higher altitudes) or from models.

For running you should first prepare an input file (in the following called `INPUTS`). The following gives an example for generating 10 vertical gammas between 100 GeV and 10 TeV:

```
RUNNR   1001                            number of run
EVTNR   1                               number of first shower event
NSHOW   10                              number of showers to generate
DATBAS  yes                             write a file with parameters used
*
* [ Random number generator: 4 sequences used in IACT mode ]
*
SEED    58485   430   0                 seed for 1st random number sequence
SEED    63432   435   0                 seed for 2nd random number sequence
SEED    20103   481   0                 seed for 3rd random number sequence
SEED    59475   490   0                 seed for 4th random number sequence
*
* [ Primary particle options ]
*
PRMPAR  1                               particle type of prim. particle
```

```
ESLOPE  -2.0                            slope of primary energy spectrum
ERANGE  100 10E3                        energy range of primary particle (in GeV)
THETAP  30.  30.                        range of zenith angle (degree)
PHIP    -14. -14.                       range of azimuth angle (degree):
*                                          from South if PHIP=ARRANG
VIEWCONE 0. 0.                          can be a cone around fixed THETAP/PHIP
*
* [ Site specific options ]
*
OBSLEV  1800.E2                         observation level (in cm)
ATMOSPHERE 10 N                         table of atmospheric profile
*                                         (10 N: Windhoek average, no refr.)
* ATMOSPHERE 1 N                         table of atmospheric profile
*                                         (1 N: tropical, no refr.)
MAGNET  12.5  -25.9                     magnetic field Gamsberg [H, Z] (muT)
ARRANG  -14.                            rotation of array to north [D] (degree)
*
* [ Cherenkov emission parameters ]
*
CERSIZ  10.                             bunch size Cherenkov photons
CERFIL  F                               Cherenkov output to extra file
CWAVLG  200.  700.                      Cherenkov wavelength band
*
* [ H.E.S.S. telescopes ] (x -> North, y -> West)
*
*            X     Y    Z    R          (all in cm)
TELESCOPE     0 -8485    0  750         Tel. 1
TELESCOPE  8485     0    0  750         Tel. 2
TELESCOPE     0  8485    0  750         Tel. 3
TELESCOPE -8485     0    0  750         Tel. 4
*
CSCAT   10  500e2  0.                   use shower several times
*
* [Interaction flags]
*
FIXHEI  0.  0                           first interaction height & target
FIXCHI  0.                              starting altitude (g/cm**2)
ELMFLG  T   T                           em. interaction flags (NKG,EGS)
RADNKG  200.E2                          outer radius for NKG lat.dens.determ.
HADFLG  0  0  0  0  0  0                flags for hadr. interaction
ECUTS   0.3  0.1  0.020  0.020          energy cuts for particles
MUADDI  F                               additional info for muons
MUMULT  T                               muon multiple scattering angle
LONGI   F  20.  F  F                    longit.distr. & step size & fit
MAXPRT  0                               max. number of printed events
ECTMAP  1.E6                            cut on gamma factor for printout
STEPFC  1.0                             mult. scattering step length fact.
*
* [ Debugging and output options ]
*
DEBUG   F  6  F  1000000                debug flag and logical unit for output
DIRECT  /dev/null                       /dev/null means no normal CORSIKA data
```

```
TELFIL iact.dat:100:100:1              Telescope photon bunch output
*                                         (eventio format)
*
* [ This is the end, my friend ]
*
EXIT                                   terminates input
```

The CORSIKA keywords are described in detail in the CORSIKA User's Guide[3]. Unused keywords can be commented out with a 'C' or '*' as the first and a blank as the second character:

```
C DIRECT  /dev/null
```

The interpretation of the 'DIRECT' parameter, by the way, has changed with CORSIKA 7.5000 and a line like

```
DIRECT  /dev/null
```

to indicate that none of the normal CORSIKA output file (particle and Cherenkov output) should actually be written to disk, will no longer work. The same functionality is now achieved with

```
DIRECT ./
PAROUT F F
```

The `corsika_autoinputs` tool does the replacement automatically when it encounters an inputs file with the old style but detects CORSIKA 7.5000 or up. Another change coming with CORSIKA 7.570 is related to the CERFIL keyword which is no longer true or false but takes numeric values.

Now check that you have enough disk space and memory and you should be ready to start CORSIKA with, depending on your shell, either:

```
./corsika <INPUTS >& something.log
```

(C shell and derivatives) or

```
./corsika <INPUTS > something.log 2>&1
```

(with Bourne shell derivatives). This can as well be run in the background. Often though CORSIKA is not run directly but rather through additional scripts which set up a number of environment variables, create run-specific data directories, link required data files etc. They may also involve preprocessing the CORSIKA inputs file (for different random number seeds, run numbers, and so on). With the CTA simulation source code package installed, it may be as simple as

```
NSHOW=1000 ./prod5_baseline_run Paranal gamma South 20
```

Keep in mind that each simulation run must have new and independent SEED parameters to avoid having the simulations done again and again. There are different tools available to generate the SEEDs in an automatic way, e.g. with `corsika_autoinputs` (part of `sim_telarray`), also incrementing run numbers. As a recommended practice, you should create a new working directory for each CORSIKA run, with symbolic links to all the necessary data files and the pre-processed INPUTS file and run CORSIKA in this working directory to avoid any possible conflicts between temporary or permanent output files produced by different CORSIKA processes running in parallel. Do not forget to clean up your symlinks after CORSIKA has completed.

*Note that unter i*86 (32-bit) Linux, file sizes are limited to 2 Gigabytes unless special care is taken during compilation. The GNUmakefile in the IACT/ATMO package will try the* `-D_FILE_OFFSET_BITS=64 -D_LARGE_FILE_SOURCE` *compiler flags when using GNU compilers. Unless you have a very old C library, this should usually be sufficient. If data files are also written by CORSIKA proper (i.e. from Fortran code), the 2 Gigabytes limit may still apply to them. Filesystem-specific limitations on file sizes may also apply.* An unavoidable limitation on 32-bit system remains the 2 GB limit on the data block size.

Chapter 3

CORSIKA compilation options specific to Cherenkov simulation

The CORSIKA source file (`corsika.F`, to be pre-processed with the C pre-processor) contains various options which can be selected for compilation, some being compatible with each other while other options are mutually exclusive. The definitive guide to available options is the CORSIKA User's Guide[3] coming with the corresponding source package. This section only mentions a selection of those most relevant to Cherenkov light simulation. Most of these options, perhaps with exception of the really IACT-specific ones, are explained in more detail in the CORSIKA User's Guide.

ATMEXT
: selects code for treatment of the atmosphere according to MODTRAN [10] model for various atmospheres by tabulated values. ATMEXT is recommended with the CERENKOV option for careful treatment of refractive index. This option needs linking with *atmo*.c routines of the IACT/ATMO ('bernlohr') package.
Note: This option is **needed** *for using the tabulated atmospheric profiles for the H.E.S.S. site.*

CEFFIC
: selects code to respect the atmospheric absorption, mirror reflectivity, and photomultiplier quantum efficiency of Cherenkov light. This option is only available in connection with the CERENKOV option.
Note: This option was contributed by Whipple/VERITAS collaborations (J. Knapp and others). Its compatibility with `sim_telarray` *is poorly tested. The provided extinction tables, mirror transmissions and quantum efficiencies do not apply for H.E.S.S. It is, therefore,* **not recommended** *to use this option together with* `sim_hessarray`*.*

CERENKOV
: selects code for additional generation of Cherenkov light. It needs the simulation with EGS4.

Note: This option is **absolutely unavoidable** *if you want to get any Cherenkov light out of CORSIKA. Please note the spelling.*

CERWLEN

is a non-standard option, i.e. not enabled by default (available as a patch with CORSIKA 6.2, included with CORSIKA proper in 6.5). The index of refraction is made wavelength dependent. As a consequence, photon bunches will carry a specific wavelength. Photons of shorter wavelengths (with larger index of refraction) will result in larger Cherenkov cone opening angles and larger bunch sizes. For very fast particles this will generally have a small effect (less than 0.03 deg in the opening angle, for example) but near the Cherenkov threshold the effect can be larger. Boundary cases are particles only emitting at the shortest wavelength.

This option may also require to use a smaller maximum bunch size (see CERSIZ keyword) since all photons in a bunch are of the same wavelength and, therefore, the peak quantum efficiency rather than the average quantum efficiency determines the maximum acceptable bunch size. (In combination with the CEFFIC option, you should use a maximum bunch size of 1, as usual.)

CURVED

selects special code to treat showers with large zenith angles $\theta < 90^\circ$. This option replaces the outdated HORIZONT option and enables also simulations with EGS4 for Cherenkov light.

Note: This options makes a bit of a difference for 60° to 70° zenith angle and a substantial difference for zenith angles well exceeding 70°. For anything below 60° zenith angle, this option is **not needed**.

IACT

selects code for simulation of **I**maging **A**tmospheric **C**herenkov **T**elescope arrays. This option is only available in connection with the CERENKOV option and needs linking with *iact*.c routines of the IACT/ATMO ('bernlohr') package.

Note: For follow-up simulations with `sim_telarray` *this option is* **absolutely unavoidable**.

IACTEXT

is a non-standard option (like ATMEXT, initially only available as a patch to CORSIKA and later with CORSIKA itself). The interface to the `TELOUT` function is extended by parameters describing the emitting particle. This extended information can be stored as an additional 'photon bunch' (after the normal one) with mass, charge, energy, and emission time replacing the `cx`, `cy`, `photons`, and `zem` fields, respectively, and are identified by a wavelength of 9999. The compact output format is disabled for making that possible.

In addition, all particles arriving at the CORSIKA observation level can be included in the eventio format output file (since IACT/ATMO package 1.66 even without IACTEXT option), in a photon-bunch like block identified by array and detector numbers 999.

The `x`, `y`, `cx`, `cy`, and `ctime` fields keep the normal sense, with coordinates, directions and time counted in the CORSIKA detection level reference frame (once per shower, without random displacements of detector arrays). The particle momentum is filled into the `zem` field (negative for upward-moving particles) and the particle ID is filled into the `lambda` field. If thinning is used, the particle weight is in the `photons` field.

VIEWCONE
selects the primary to come from a cone around a fixed zenith and azimuth angle.
Note: If you want to do simulations of extended or diffuse emission, this option is **highly recommended**.

CEROPT
is needed for compiling with the SSE/AVX vector-optimized variant of the Cherenkov emission. It also requires a patch to CORSIKA and additional code, like a C implementation of the CERENK subroutine as well as vectorized versions of mathematical functions. It is not yet covered by the usual build procedure but **needs a custom compilation**. It assumes the ATMEXT, CERENKOV, and IACT options. It is not compatible with any of the CEFFIC, CERWLEN, CURVED, IACTEXT, THIN, UPWARD, and AUGCERLONG options.

The CORSIKA User's Guide explains how to select specific options interactively with the `coconut` tool, which that then marks for subsequent compilation. Most of the CORSIKA options, in particular those relevant with Cherenkov light, can also be activated with the `gen_config` script (included with the IACT/ATMO package) which is used by the `build_all` and `corsika_build_script` tools on top of `coconut` for automated builds with pre-selected set of options.

Chapter 4

CORSIKA keywords specific to Cherenkov simulation

This section explains a number of keywords in the CORSIKA inputs file and their parameters. This is limited to those keywords directly dealing with Cherenkov light, or which are of particular relevance for Cherenkov simulations, or where complications arise or common misunderstandings are known to happen. Part of the explanations are drawn from the CORSIKA User's Guide (which may contain more concise descriptions in the meantime), supplemented by notes more specific to Cherenkov simulations, to the IACT/ATMO package, or immediately to `sim_telarray`.

RUNNR : Run number.

```
RUNNR    NRRUN
```

Format = (A5, I), Default = 1
NRRUN : Run number of this simulation. This number is used to form part of the name of the various output files.
Limit is: $0 \leq$ NRRUN ≤ 999999 (rsp. 0 *le* NRRUN ≤ 999999999 in case of the NRREXT option.

PRMPAR : Primary particle ID.

```
PRMPAR    PRMPAR(0)
```

Format = (A6, I), Default = 14
PRMPAR(0) : Particle type of the primary particle. See the CORSIKA User's Guide for particle codes and limits. Not all particle types are suitable as primary particles (may depend on interaction model). For more details see the CORSIKA User's Guide. This keyword is not available in the STACKIN option and may be ignored with the `IACT EXTPRIM` input card.

ERANGE : Energy range of primary particle.

```
ERANGE    LLIMIT    ULIMIT
```

Format = (A6, 2F), Defaults = 1.E4, 1.E4 LLIMIT : Lower limit and ULIMIT : Upper limit of the primary particle energy range (in GeV). The primary energy is selected at random out of this interval. If LLIMIT = ULIMIT, the primary energy is fixed at this value.
The energies are total energies and include the particle rest mass. For more details see the CORSIKA User's Guide. This keyword is not available in the STACKIN option and may be ignored with the `IACT EXTPRIM` input card.

ESLOPE : Slope of energy spectrum.

```
ESLOPE    PSLOPE
```

Format = (A6, F), Default = 0.
PSLOPE : Exponent γ of differential primary energy spectrum. The primary energy is taken at random from an exponential energy spectrum of the form $dN/dE_0 \propto E_0^{\gamma}$. PSLOPE has no meaning in case of fixed primary energy. The energies are total energies and include the particle rest mass. This keyword is not available in the STACKIN option and may be ignored with the `IACT EXTPRIM` input card.

ATMOSPHERE : Tabulated atmospheric profiles.

```
ATMOSPHERE     IATMOX    FREFRX
```

Format = (A10, I, L), Defaults = 0, F
IATMOX : Use MODTRAN [10] atmospheric model `IATMOX = i` (in terms of density and refractive index) instead of CORSIKA built-in model. This requires a file named *atmprof*`i`.dat. Supplied MODTRAN model atmospheres include tropical (`i` = 1), mid-latitude summer (2), mid-latitude winter (3), sub-arctic summer (4), sub-arctic winter (5), and U.S. standard atmosphere 1976 (6). User supplied models are possible (`i`$\geq$ 7).
FREFRX : If true, the atmospheric refraction for Cherenkov photons is taken into account (for plane-parallel atmosphere); if false, refraction is ignored. The value of this second argument is ignored if the CERENKOV option is not selected.
This keyword is only available in the ATMEXT option.
Note: This parameter is the recommended way of using the atmospheric profile matching the H.E.S.S. site. Either use the all-year average profile number 10 or the seasonal extremes (number 11 and 12, respectively). CORSIKA has several other ways to change the atmospheric profile, but no H.E.S.S.-specific parametrization is available for the other ways. The atmospheric refraction has a significant impact on the performance. If refraction is considered unimportant for the purpose of your simulations, set `FREFRX` to `false`.
See also `IACT ATMOFILE` as an alternative.

ECUTS : Energy cut-offs.

```
ECUTS      ELCUT(i), i=1... 4
```

Format = (A5, 4F), Defaults = 0.3, 0.3, 0.003, 0.003
ELCUT(i) : The low energy cut-off (in *GeV*) of the particle kinetic energy may be selected differently for hadrons ($i = 1$), muons ($i = 2$), electrons ($i = 3$), and photons ($i = 4$). For nuclei ELCUT(1) is applied to the energy per nucleon.
Limits are: ELCUT(1) $\geq$ 0.05 ; ELCUT(2) $\geq$ 0.05; ELCUT(3), ELCUT(4) $\geq$ 0.00005
Note: In contrast to normal CORSIKA recommendations, Cherenkov light simulations require only electrons above the Cherenkov threshold (about 20 MeV). For gammas you can use the same threshold. For muons, the Cherenkov threshold is not a good guess because decaying muons may produce local electromagnetic subshowers. Therefore, muons (and also nuclei) should be left at a low cut-off, e.g. 200–300 MeV kinetic energy. The latter is not an efficiency problem.

CWAVLG : Cherenkov wavelength band

```
CWAVLG     WAVLGL    WAVLGU
```

Format = (A6, 2F), Defaults = 300., 450.
WAVLGL : Lower limit (in *nm*) of the wavelength band for Cherenkov radiation production.
WAVLGU : Upper limit (in *nm*) of the wavelength band for Cherenkov radiation production.
Limits are: 100. $\leq$ WAVLGL $<$ WAVLGU $\leq$ 2000 (180 nm to 700 nm without IACT option).

There is actually no good reason why it should be limited to this wavelength range unless the CEFFIC option is used with the default PMT efficiency (which is cut off below 700 nm). Well, a good atmospheric Cherenkov detector should better cut off below 550 nm, in order to improve signal/sqrt(night sky background), but not every detector does that. Where a detector has significant sensitivity beyond the limit imposed by CORSIKA, this limit can later only be circumvented by the telescope simulation program if the wavelength does not get thrown inside CORSIKA (neither CEFFIC nor CERWLEN options used for compiling CORSIKA). Such a circumvention would imply scaling every bunch size by a factor

$$\frac{1/\lambda_{\mathrm{l,new}}^2 - 1/\lambda_{\mathrm{u,new}}^2}{1/\lambda_{\mathrm{l}}^2 - 1/\lambda_{\mathrm{u}}^2},$$

where λ_{l} and λ_{u} stand for the limits imposed in CORSIKA (WAVLGL and WAVLGU, respectively). Increasing a 250 to 700 nm range to a 250 to 1000 nm range implies a scale factor of 1.075.

CERSIZ : Cherenkov bunch size definition

```
CERSIZ    CERSIZ
```

Format = (A6, F), Default = 0.
CERSIZ : Defines the maximal bunch size of Cherenkov photons that are treated together. If set to 0., by the subroutine *getbus* (with CERENKOV) option) the program calculates a bunch size which was appropriate for the non-imaging HEGRA AIROBICC array.
Limit is: CERSIZ ≥ 0.
Note: You want to keep the chances of more than one photo-electron from a single photon bunch to stay rather low. You should typically use a value of 5 with normal PMTs in your camera and when the `CEFFIC` *option is disabled. With high-efficiency PMTs a value of 3 to 4 would be more conservative. When the* `CEFFIC` *option is enabled, a value of 1 seems appropriate. When* `CERWLEN` *is enabled, a value of more than 2 cannot be recommended. Although values below 1.0 are possible, they make no sense.*

CERFIL : Cherenkov output steering
In versions prior to 7.570:

```
CERFIL    LCERFI
```

Format = (A6, L), Default = T
Note: This file is not needed with the `IACT` *option. Ignore it or set CERFIL to false.*
Since version 7.570, it is an integer, with the value 1 indicating that Cherenkov photons should be written to a dedicated file (as before) while a value 0 means that they are to be written into the particle output file as well (all FORTRAN unformatted sequential format). Turning off the CORSIKA particle output with the newer versions is using the `PAROUT` data card. In older CORSIKA inputs file you will often see the cards

```
DIRECT /dev/null
CERFIL F
```

to turn off either CORSIKA output. The `corsika_autoinputs` tool will translate this into its newer equivalent

```
DIRECT ./
CERFIL 0
PAROUT F F
```

Check with the CORSIKA User's Guide of the proper version for details.

CSCAT : Multiple use of Cherenkov events

```
CSCAT     ICERML    XSCATT    YSCATT
```

Format = (A5, I, 2F), Defaults = 1, 0., 0.
ICERML : Number of uses of each event.
XSCATT : Max. scattering of core location in $\pm X$ direction (in *cm*).
YSCATT : Max. scattering of core location in $\pm Y$ direction (in *cm*).
Limits are: $0 \leq$ ICERML ≤ 20 ; XSCATT, YSCATT ≥ 0.
In case of `IACT` option (Cherenkov telescopes) ICERML telescope arrays are simulated randomly in the specified area which is a circle of radius XSCATT, if YSCATT $= 0.$, or within a rectangle of area 2 XSCATT $\cdot$ 2 YSCATT.
Instead of the flat distribution of impact positions (with VOLUMEDET option in a plane perpendicular to the direction of the incoming primary particle), a non-uniform distribution can be enabled with the `IACT TELSAMPLE` card.

CERQEF : Cherenkov quantum efficiency, atmospheric transmission and mirror reflectivities.
Note: Only available with the `CEFFIC` option. See the CORSIKA User's Guide. There are no instrument-specific data files available for this option – and the files provided with `sim_telarray` will, in general, not match the required format.

OBSLEV : The altitude of the observation level(s) above 'mean sea level' or the geoid. Although CORSIKA can handle more than one level, the Cherenkov simulations in general and the IACT option in particular only use the lowest level, so no point in specifying more than one level. Units are centimeters while typical notations try to be more readable by mimiking meters or kilometers like

```
OBSLEV 1835E2
OBSLEV 2.15e5
```

for levels of 1835 m and 2150 m a.s.l., respectively. Since the format allows for more than one level, be careful with comments behind it which might get interpreted as additional levels. Particles are terminated once they reach the (lowest) observation level and no Cherenkov light gets emitted below it.

MAGNET Earth's magnetic field

```
MAGNET     BX    BZ
```

Format = (A6, 2F), Defaults = 20.40, 43.23
BX : Is the horizontal component of the Earth's magnetic field (in μT) to the x-direction of the coordinate system (North) (see Fig. 1 of the CORSIKA User's Guide) BZ : Is the vertical component of the Earth2019s magnetic field (in μT) downwards. The default values represent the magnetic field for the Karlsruhe location. The values of other locations may be obtained from the program Geomag

which is available on-line in the world wide web [1]. The value H of Geomag corresponds with our BX, the value Z with our BZ. For the orientation of the CORSIKA coordinate system see also Fig. 1 of the CORSIKA User's Guide.
Limits are: BX, BZ $\neq 0$.

In addition to the above text from the CORSIKA program, it seems important to note that Geomag output is in nanoTeslas rather than microTeslas. It also seems prudent to point out the unusual axes orientations for the field componenets: X is towards geographic North, Y towards East, and Z is downward! A positive field inclination means that the field points below the horizon. A positive declination (positive Y or East component until any future field reversal) means that a compass would point East of the intended North direction. For example,
`geomag70 IGRF13.COF 2020.00 D K1.835 -23.271778 16.500222`
would result in the following output for the H.E.S.S. site in 2020:

```
Geomag v7.0 - Jan 25, 2010


 Model: IGRF2020
 Latitude: -23.27 deg
 Longitude: 16.50 deg
 Altitude: 1.83 km
 Date of Interest:  2020.00

-------------------------------------------------------------------------------
  Date          D           I           H        X        Y        Z        F
  (yr)      (deg min)   (deg min)     (nT)     (nT)     (nT)     (nT)     (nT)
 2020.00   -12d  53m   -64d  30m   11997.8  11695.9  -2674.3 -25147.9  27863.3
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  Date         dD          dI          dH       dX       dY       dZ       dF
  (yr)      (min/yr)    (min/yr)    (nT/yr)  (nT/yr)  (nT/yr)  (nT/yr)  (nT/yr)
 2020.00       3.2         3.7        -1.3      1.3     11.3     72.4     -65.9
-------------------------------------------------------------------------------
```

That would correspond to the following inputs cards:

```
MAGNET  11.9978 -25.1479
ARRANG  -12.883
```

ARRANG : Array rotation (geographic to magnetic North)

```
ARRANG    ARRANG
```

Format = (A6, F), Default = 0.
ARRANG : Defines a rotation angle (in °) between the detector array x-direction and magnetic north direction; positive if detector array x-direction points to the West.
Limits are: -180. $\leq$ ARRANG $\leq$ 180.

Note: This parameter (or the above description taken from the CORSIKA User's Guide) is somewhat complicated (and actually was stating just the opposite in earlier versions). Normally, the geomagnetic 'declination' is counted positive if magnetic

[1] The Geomag program is is available in C and FORTRAN versions and as a Python package. See here for more information.

North is east of geographic North. In that case, the 'array x-direction' (which is geographic North for us) is westwards of the magnetic North direction and, thus, ARRANG positive. **Simple rule: use the magnetic declination.** *Since the PHIP direction of the primary is the direction of flight and not the arrival direction, the normal azimuth value (counted eastwards from geographic North) is (modulo 360 degrees)*

$$Az = 180^\circ + \texttt{ARRANG} - \texttt{PHIP}.$$

Example: For the H.E.S.S. site in Namibia the geomagnetic declination is about -14° (geomagnetic North is 14° west of geographic North), implying ARRANG$=-14$. For PHIP=180 (a particle flying towards magnetic South, i.e. coming from magnetic North) we get $Az = -14^\circ$, for PHIP=76 we get $Az = 90^\circ$ (which means coming from geographic East).

CERARY : Cherenkov Detector Array Definition

This parameter is not available when the IACT option is enabled. It is specific to non-IACT Cherenkov light simulations with a regular and flat detector layout.

TELESCOPE : Cherenkov telescope definition

```
TELESCOPE     X     Y     Z     R
```

Format = (A5, 4F)
X, Y, Z : Co-ordinates of Cherenkov telescope (in *cm*). This keyword adds a new telescope at position X, Y, Z with radius R, within which the telescope is fully contained. At least one telescope has to be specified. Limits are: $0 < R$; $1 \leq$ number of telescopes < 1000.
This keyword is only available in the CERENKOV option together with the IACT option for Cherenkov telescopes.

Note: In contrast to what the full entry from the CORSIKA User's Guide seems to imply, X is counted towards geographic North, Y towards geographic West, and Z is counted upwards from the observation level (see OBSLEV) keyword. Otherwise the changing geomagnetic field would require new telescope coordinates all the time.

If any detector sphere extends to below the observation level ($R > Z$), the whole detector array is lifted up by $R - Z$, unless the IACT interface is compiled with IACT_NO_GRID defined (which is typical for the LightEmission package but unusual and inefficient for normal IACT array simulations). Instead of defining detectors with fiducial sphere extending to below the observation level, reduce the OBSLEV altitude and increase all detector Z values to compensate for that.

TELFIL : Cherenkov telescope data file name

```
TELFIL     TELFNM
```

Format = (A5, A100)
TELFNM : The telescope-specific data are to be written to a file named TELFNM in `eventio` format. Lower case characters of TELFNM are not converted to capitals. If this file exists and is write-enabled, new data is appended. After ending the run the file will be set read-only to avoid accidental overwriting. If the file name itself ends in `.gz` or `.bz2` then the output will be compressed on the fly with the `gzip` or `bzip2` commands, respectively. If starting with a `|` (after an optional + or −) it indicates that output should be piped into standard input of another program. The file name `/dev/null` suppresses the output file.
This keyword is only available in the `CERENKOV` option together with the `IACT` option for Cherenkov telescopes.

Note: Actually the 'name' contains a few tricks, which are explained in full detail in the IACT reference manual under function `telfil_`. To summarise the most important here: A leading '+' means that the non-compact format should be used instead of the default compact format. A leading '−' means that the compact format should be used (since IACT/ATMO package version 1.66). The compact format has a number of limitations (not too large telescope radius times secant of zenith angle, and so on), explained in detail in the introduction of the IACT reference manual. If the compact format is specified but its limitations are obviously violated, the non-compact format is used.

At the beginning or after the '+', a '`|`' character indicates that output is not written to a file of the name following but is piped into the telescope simulation program started by a script of the name following. A script, because no blanks can be embedded in the file name supplied and, therefore, no command-line options/parameters for the telescope simulation program can be included here. A colons as separators (keep in mind: no whitespace allowed), the number and frequency of events to be logged can be specified (see IACT reference manual for details). See also under `IACT TELFIL`, `IACT TELOPT`, and `IACT PRINT_EVENTS` parameters.

PHIP : Azimuth angle definition

```
PHIP     PHIPR(1)     PHIPR(2)
```

Format = (A4, 2F), Defaults = 0., 0.
PHIPR(1) : Low edge of azimuth angle range of primary particle (in degrees).
PHIPR(2) : High edge of azimuth angle range of primary particle (in degrees).
The azimuth angle is selected at random out of this interval. If PHIPR(1) = PHIPR(2), the azimuth angle is fixed at this value. For *phi* = 0 degrees the shower axis points to magnetic North, for ϕ = 90 degrees it points to West, see Fig. 1 in CORSIKA User's Guide).
Limits are: -360. $\leq$ PHIPR(i) $\leq$ 360.

Note that the azimuth angles in CORSIKA differ from how they are handled in astronomy (and in `sim_telarray`), by both the zero point and the direction into

which it gets increased. See also the ARRANG parameter for transforming between CORSIKA and astronomical azimuth angles.

Note also that, when combined with the VIEWCONE input card, the actual cone is centered corresponding to the mean of the two values.

THETAP : Zenith angle definition

```
THETAP     THETAPR(1)     THETAPR(2)
```

Format = (A4, 2F), Defaults = 0., 0.
THETPR(1) : Low edge of zenith angle range of primary particle (in degrees).
THETPR(2) : High edge of zenith angle range of primary particle (in degrees). The zenith angle is selected at random out of this interval in a manner which respects equal particle fluxes from all solid angle elements of the sky and a registration by a horizontal flat detector arrangement. THETPR is the angle of incidence at a horizontal detector. THETPR(i) = 0. is vertical. If THETPR(1) = THETPR(2), the zenith angle is fixed at this value. Limits are: 0. $\leq$ THETPR(i) $\leq$ 70.

Note that, when combined with the VIEWCONE input card, the actual cone is centered at a zenith angle corresponding to the mean of the two values.

VIEWCONE : Viewing cone specifications

```
VIEWCONE     VUECON(1)     VUECON(2)
```

Format = (A5, 2F), Defaults = 0., 0.
VUECON(1) : Inner limiting angle of viewing cone (in $^\circ$).
VUECON(2) : Outer limiting angle of viewing cone (in $^\circ$).
The VIEWCONE option selects the direction of primaries in a circular cone around the fixed primary direction THETPR(1) and PHIPR(1) (see THETAP and PHIP parameters in the CORSIKA User's Guide) with the inner opening VUECON(1) and the outer opening VUECON(2). The zenith angular dependence of the selected detector geometry is maintained for flat horizontal or spherical detectors (without or with VOLUMEDET option).
Limits: 0. $\leq$ VUECON(1) $\leq$ VUECON(2) $<$ 90. The generation of showers with angles beyond the range of the program validity is skipped.
This keyword is only available in the VIEWCONE option.

DATBAS : Write data base file

```
DATBAS     FDBASE
```

Format = (A6, L), Default = F
FDBASE : If true, the essential run parameters are written to file 'DAT*nnnnnn*.dbase'

onto the output directory DSN (see DIRECT). This file may be used to build a data base for examining the content of an air shower library.
Note: The file written is actually an ASCII text file but tools exist for conversion into an SQL database. If you plan to use such tools, you should set 'DATBAS true' *(that is independent of Cherenkov light).*

DIRECT : Output directory

```
DIRECT    DSN
```

Format = (A6, A64), Defaults = 'anynameupto64characters'
DSN : May be used to define a name of an output directory. Lower case characters of DSN are not converted to capitals. To suppress the output you might give /dev/null or use the keyword PAROUT.
Since you don't need the normal CORSIKA output with the IACT option you better set 'DIRECT /dev/null'*, but only with CORSIKA version before 7.5000.* Starting with version 7.5000 the only accepted way is with

```
PAROUT  F F
DIRECT  ./
```

IACT : Pass-through of parameters to IACT package

```
IACT iact-keyword iact-parameter
```

No specific format, depends on actual keyword passed through. Some of the keywords are also recognized by the CORSIKA inputs handling, e.g. TELFIL, others are not directly known to CORSIKA code. Examples:

```
* [ IACT tuning parameters ]
*
* Split data with more than 15 million bunches:
IACT SPLIT_AUTO 15M
* At 32 bytes per bunch this could be up to 500 MB:
IACT IO_BUFFER 1000MB     // support up to 1000 MB
* Let photon bunch thinning set in earlier:
IACT MAX_BUNCHES 1000000  // <= a million bunches per telescope
* Importance sampling correction (with extra code):
* IACT TELSAMPLE sampling-config-file
* Disables correction of bending in B field:
* IACT IMPACT_CORRECTION off  // not recommended
* TELFIL is possible with and without IACT prefix:
* IACT TELFIL filename.corsika.gz
* TELFIL filename.corsika.gz
```

The following list include IACT keywords as known with IACT/ATMO package version 1.66 (can be UPPER/lower/MiXed case):

IACT TELFIL filename: Passed through by CORSIKA `TELFIL` parameter. Filename may be prefixed by '+' to enforce long bunch format, even if automatic detection does not find it necessary, and by '|' to indicate a script to be executed (no blanks and no shell meta-characters allowed, length restricted by CORSIKA input card handling). It may also optionally be followed (without blanks) by several colon-separated numbers (see IACT/ATMO package README file for details). It is recommended to use the `IACT TELOPT`, `IACT PRINT_EVENTS`, `IACT INTERNAL_BUNCHES`, and `IACT IO_BUFFER` settings rather than appending these numbers in the old style to the file name.

IACT TELOPT text: Command line options to be appended to an external program/script in TELFIL to receive IACT data, as in:

```
TELFIL +|my_simulation_script.sh
IACT TELOPT -c my_configuration.cfg
```

It can include blanks but no shell meta-characters.

IACT PRINT_EVENTS numbers: Up to five numbers indicating how after and at which initial offset to print event info into the log output (see IACT/ATMO package README file for details).

IACT INTERNAL_BUNCHES nbunch: Number of bunches to be held in memory per telescope before starting to push them to a (per-telescope) temporary file.

IACT MAX_BUNCHES nbunch: Maximum number of bunches that can be kept per telescope, including in temporary files. Telescopes for which this limit is exceeded will have the number of bunches reduced in increasing powers of two (by discarding every second bunch and adding its bunch size to the bunch kept).

IACT IO_BUFFER size: Maximum size of data blocks written by the IACT module. Size should be directly followed by a unit as in '800MB' or '8GiB'.

IACT SPLIT_AUTO nbunch: If the full data block for the entire array would exceed the given number of bunches, it gets split up into separate blocks per individual telescope, in order to avoid exceeding the hard I/O buffer size limit. Example solution:

```
IACT SPLIT_AUTO 15M      // Split data with more than 15 million bunches
IACT IO_BUFFER 1000MB    // At 32 bytes per bunch this could be up to 500
IACT MAX_BUNCHES 1000000 // Let photon bunch thinning set in earlier.
```

IACT SPLIT-ALWAYS : Always split IACT output data to data blocks for individual telescopes.

IACT SETENV variable value: Set an environment variable to the given value. This can later be expanded internally, e.g. in TELFIL, or externally in the script passing the output data along.

IACT ATMOSPHERE natmo: Also passed through by CORSIKA `ATMOSPHERE` parameter. Tabulated atmosphere number to be loaded from a file `atmprof<natmo>.dat` interpolated and fit.

IACT ATMOFILE filename: Use a specific file name with the tabulated atmosphere rather than being limited by a small range (1-98) of possible `ATMOSPHERE` numbers. The atmosphere number will be recorded as '99' in the output, indicating the custom filename.

IACT EXTPRIM filename: Instead of using the primary particle energies and directions randomly generated in CORSIKA internally, they will be loaded from an external file. Also allows for a mix of different primary particle types in the same simulation run. See IACT/ATMO package README file for details. Beware of re-using the same file for multiple simulation runs – like you should beware of re-using the same SEED values again.

IACT TELSAMPLE details: Instead of a uniform sampling of the CSCAT random telescope array offsets within the given range, this allows for a non-uniform ("importance") sampling which may be dependent on energy alone or energy and zenith angle. Possible variants:

```
IACT TELSAMPLE fixed:k,r1,r2
IACT TELSAMPLE 1d:fname-with-E-k-r1-r2
IACT TELSAMPLE 2d:fname-with-E-theta-k-r1-r2
```

where the first variant works without any extra file while the 1-D and 2-D external file variants with specific formats from which the actual sampling parameters for a given primary particle of energy E and zenith angle theta are interpolated (linearly in log(E) although the file lists E and not log(E), and linearly in theta). The r1 and r2 values are fractions of the CSCAT radius R, with flat distribution inside $r_1 R$ and outside $r_2 R$, and a power-law drop with $k < 0$ inbetween. See Figure 4.1 for example distributions. The corresponding 1-d TELSAMPLE file reads:

```
# E    k   r1   r2
1.0   -5   0.1  0.3
10.   -5   0.15 0.3
100.  -5   0.2  0.4
1e3   -4   0.3  0.6
1e4   -3   0.5  1.0
1e5   -2   1.0  1.0
```

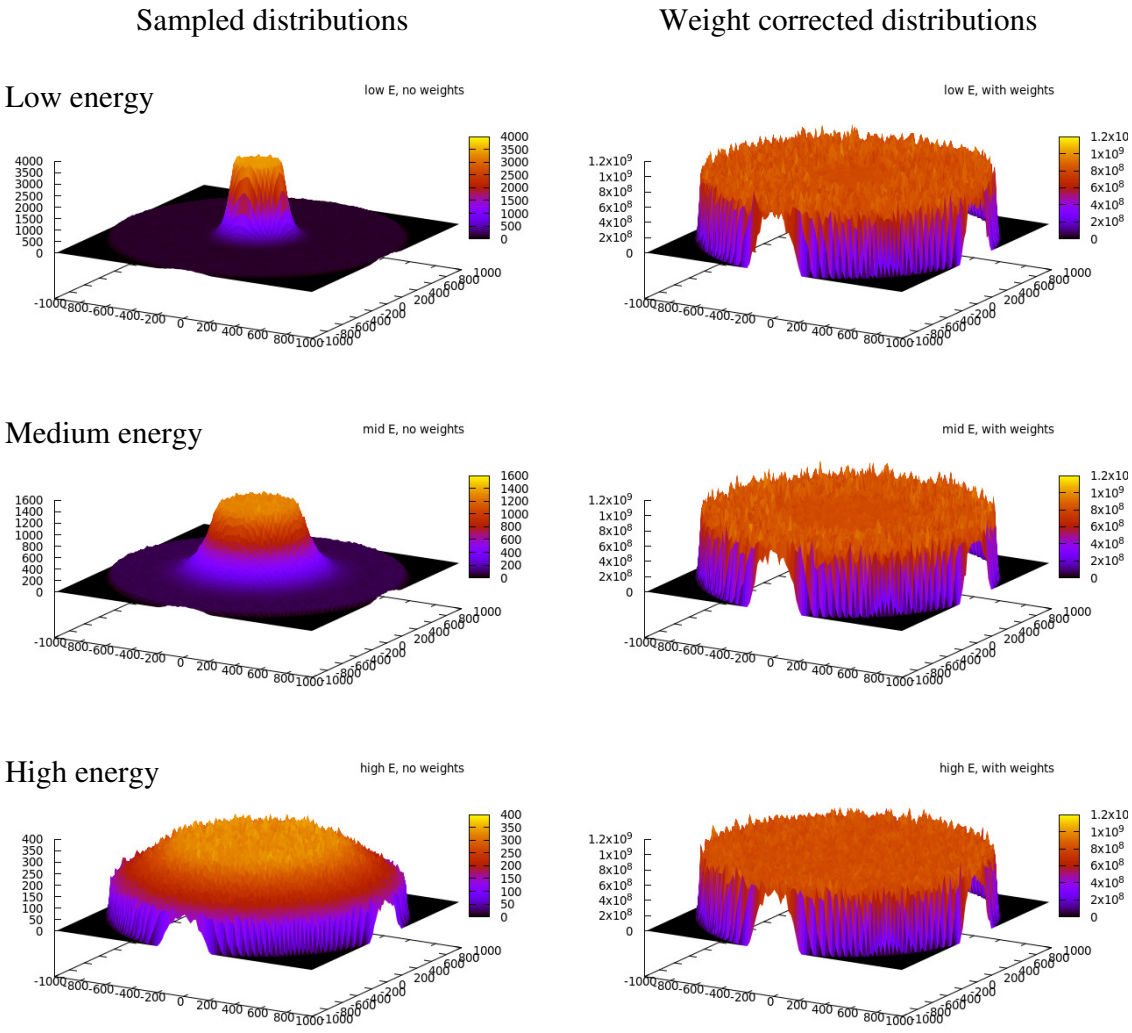

Figure 4.1: An example of energy-dependent non-uniform core-position sampling via `IACT TELSAMPLE`, with the distributions generated core positions on the left side and the corrected distributions (each event with its 'area weight' as recorded in the data) on the right side. While the low-energy showers are highly concentrated at small core distances, the high-energies are much more uniformly distributed. Such a set-up should be particularly useful for an installation with a few large telescope (for low energy showers) in the center and many smaller telescopes (not sensitive to low energy showers) distributed over a larger area.

IACT STORE-PARTICLES bool

The IACT interface can store the particles reaching observation level just as well as the CORSIKA native FORTRAN output. It used to be available only with the (compile-time) IACTEXT option and only if the PAROUT input card enabled particle output. Since the FORTRAN output cannot be disabled anymore with the DIRECT input card (since version 7.5), this means that the particle output would be written to two files or none. With IACT/ATMO package version 1.66, this behaviour was changed (also requiring a patch to CORSIKA, for the time being). There is no dependency on the IACTEXT option anymore and no dependency on the PAROUT card. Instead, the new input card (default value: false) can be used to enable particle output in the IACT output only. It also works with the SSE/AVX vector-optimized CORSIKA variant (CEROPT option).

IACT STORE-EMITTER bool

When compiled with the IACTEXT option, the IACT interface also obtains information about the particles emitting Cherenkov photons. Writing this information used to be enabled by compiling the IACT interface with `STORE_EMITTER` defined. With IACT/ATMO package version 1.66, this feature can be enabled at run time if built with IACTEXT option (as otherweise the information is not there). The default value is true only if compiled with `STORE_EMITTER` defined. Otherwise, no extra information about the emitting particles gets written unless explicitly enabled. Since the CEROPT variant is not compatible with IACTEXT, no extra information can be recorded with it. Trying to enable it without IACTEXT will result in an error message but will otherwise be ignored.

Chapter 5

The `pfp` pre-processor for CORSIKA inputs and `corsika_autoinputs`

5.1 pfp

The `sim_telarray` program by defaults pre-processes its configuration with a custom pre-processor termed ‘ `pfp` ’ (orignally: Portable Fortran Preprocessor). This tool has a syntax very similar to the C/C++ compiler pre-processor, although with some noteable differences. Among these differences is the (so far) case-insensitivity of its keywords, the macro substitution using a more explicit syntax, and the evaluation of (floating-point) numerical expressions. The same pre-processor is also routinely used for CORSIKA input files, using one template for multiple purposes, like different primary types and zenith angles, with dependent settings of energy ranges, CSCAT radius etc. The following shows a few example features frequently used from (selected parts of) such templates:

```
RUNNR 12345
*
#ifdef PRIMARY_SILICON
PRMPAR  2814          // Particle type of prim. particle (2814: silicon)
ERANGE  0.05E3 5000E3 // Energy range of primary (in GeV): silicon
NSHOW   2500          // Number of showers to generate
IACT setenv PRMNAME silicon
#define DIFFUSE 1
#endif
*
#if defined(EMIN) && defined(EMAX)
ERANGE   $(EMIN)E3 $(EMAX)E3 // Requires EMIN and EMAX in units of TeV.
#endif
*
#if defined(VIEWCONE)
VIEWCONE 0. $(VIEWCONE) // Custom viewcone setting
#elif defined(DIFFUSE)
VIEWCONE 0. 10.  // Diffuse components (gammas, electrons, protons & nuclei)
#else
VIEWCONE 0. 0.   // Fixed THETAP/PHIP (gamma point source)
```

```
#endif
*
#if defined(HAVE_MATHEVAL) && defined(YEAR)
* Correct B field evaluated for $(YEAR):
MAGNET $[11.930-0.0336*($(YEAR)-2020)] $[-25.310+0.0358*($(YEAR)-2020)]
# define BFDECL $[-12.750+0.102*($(YEAR)-2020)]
ARRANG $(BFDECL)
# define PHIP_SOUTH $(BFDECL)
# define PHIP_EAST  $[$(BFDECL)+90]
# define PHIP_NORTH $[$(BFDECL)+180]
# define PHIP_WEST  $[$(BFDECL)+270]
#else
* Without pre-processor math expression evaluation use B field for 2020.0:
MAGNET  11.930 -25.310   // Magnetic field (H, Zdown) at H.E.S.S. site
ARRANG  -12.750          // B field declination = array rotation
# define PHIP_SOUTH -12.750
# define PHIP_EAST 77.250
# define PHIP_NORTH 167.250
# define PHIP_WEST 257.250
#endif
*
#if defined(FROM_SOUTH)
PHIP $(PHIP_SOUTH) $(PHIP_SOUTH)  // CORSIKA azimuth for primaries coming from South.
IACT setenv AZM 180               // Corresponding astronomical azimuth.
#elif defined(FROM_EAST)
PHIP $(PHIP_EAST) $(PHIP_EAST)    // CORSIKA azimuth, from East.
IACT setenv AZM 90                // Corresponding astronomical azimuth.
#elif defined(FROM_WEST)
PHIP $(PHIP_WEST) $(PHIP_WEST)    // CORSIKA azimuth angles, from West.
IACT setenv AZM 270               // Corresponding astronomical azimuth.
#else
PHIP $(PHIP_NORTH) $(PHIP_NORTH)  // CORSIKA azimuth angles, from North.
IACT setenv AZM 0                 // Corresponding astronomical azimuth.
#endif
*
#ifndef ZA
# define ZA 20
#endif
IACT setenv ZA $(ZA)
THETAP $(ZA) $(ZA)
*
TELFIL run${RUNNR}_${PRMNAME}_za${ZA}deg_azm${AZM}deg.corsika.zst
```

That shows only one example of a primary particle type covered by the template. Note the different type of parentheses/brackets/braces following the dollar signs. The parenthesis `$(...)` is used for the expansion of pre-processor macros (`#define` in the file or the pre-processor started with a corresponding `-D...` option). The brace `${...}` expands an environment variable, either set externally beforehand, through the `IACT setenv` inputs line, or automatically set by the IACT module processing of some CORSIKA input parameters, like RUNNR as an example. Finally, the bracket `$[...]` denotes a numerical (and/or logical) expression evaluation, which allows for macro expansion inside.

(Environment variable expansion is tricky in these expressions, as interpretation of the `IACT setenv` would only by done when CORSIKA is running and, thus, too late for the pre-processor which has to be run before CORSIKA.) Since older versions of `pfp` may come without that feature, and also current versions if compiled by older C compilers (using pre-C99 language standards), the availability of the feature can be tested by the HAVE_MATHEVAL macro. The example includes a fall-back.

5.2 `Corsika_autoinputs`

The `corsika_autoinputs` both finishes the set-up of the CORSIKA inputs with a unique (usually: incremented) run number and SEEDS input cards with randomly generated values.

Depending on how simulation jobs are organized, the run number can be auto-incremented (all jobs having access to the same `CORSIKA_DATA` directory and file-locking will work) or defined ahead of submitting a simulation job. With `corsika_autoinputs` a pre-defined run number would be set with the `--run-number` or `-R` option (*not the* `--run` *option*, which tells which program to run as CORSIKA. Try with the `--help` option for more details).

CORSIKA uses several independent sequences of pseudo-random numbers, from a generator based on an older CERN Library function. For the same seed values, as given with four 'SEED' lines in the input file, the same sequences will result. Running CORSIKA with the same seeds for those would result in the same events in every run – a mistake all too easy to make. Therefore, `corsika_autoinputs` re-generates these seeds, based on its own, not reproducible random numbers. In order to force a reproducible CORSIKA simulation, `corsika_autoinputs` can be told, with the `--keep-seeds` option, to leave the 'SEED' lines in the original input file intact.

A few version-specific problems are also spotted and fixed. `Corsika_autoinputs`also creates the working directory for CORSIKA under `CORSIKA_DATA` and populates it with symbolic links to the needed data files from `CORSIKA_PATH`.

Chapter 6

CORSIKA IACT files without CORSIKA – the LightEmission package

Either as part of `sim_telarray` or as a stand-alone package, there is a library and a number of useful (or example) applications available which emulate the IACT output from CORSIKA and can be used as input to `sim_telarray` or other Cherenkov light detector simulation codes. It does not include or depend on any part of CORSIKA itself but only the IACT/ATMO interface.

The main part of the code is a C++ library that defines and implements a single run per file, like CORSIKA, and one or multiple ficudial spheres. These fiducial spheres may correspond to telescopes, to a mirror, or to a camera. In the latter cases, the 'observation level' corresponds to a plane behind or in front of the mirror or the camera front, and the detector simulation would skip part of the ray-tracing of incoming photons (see `BYPASS_OPTICS` with `sim_telarray`). A variety of pulsed light sources can be set up with parameters describing the temporal, spectral, and angular distributions by hardcoded values or read them from a data file. Since almost everything is covered by the library, the applications can be very short. A typical application would include:

- Setting up the run, with site altitude, atmospheric profile etc. Example:

```
Run run(nrun, alt_m*100., atmosphere, cam_fid_cm);
run.SetOutput(filename);
```

- Setting up one or multiple light sources. Example:

```
LightSource flasher(
   SpaceVect(ff_xy_cm[0],ff_xy_cm[1],ff_dist_cm),
   spectrum, pulse, angular);
```

- Loop over events with each light source emitting photon bunches according to its set-up. The first event takes care of writing run header etc. Example:

```
for ( int iev=0; iev<events; iev++ )
{
   // Takes care of writing out photon bunches
   // of previous event, if any:
   run.NextEvent();

   // Emit light for this event (per light source):
   flasher.Emit(run, nphot_per_pos, bunch_size, 0.);
}
```

- Write the last event, run-end etc. and clean up as part of the desctructor of the Run class:

```
 return 0;
```

Some of the provided applications are rather useful while others are more a showcase of other possible uses:

ff-1m Direct flat-field illumination of a Cherenkov camera.

ff-gct Flat-field illumination via reflection on secondary mirror.

pixled Light source in front of individual pixels for trigger logic tests.

xyzls Light sources at arbitrary positions. An example application would be an 'Illuminator' sending a light flash from a few hundred meters away towards a Cherenkov telescope pointed to the Illuminator, resulting in images spread over a large number of pixels.

fake-muon Cherenkov light by fake muon (or other charged particle). In contrast to CORSIKA, we start with the end point and track backwards, including multiple scattering.

octo A light source (on an assumed octocopter) moving unrealistically fast in the sky (in an '8' pattern). Just for demo.

ls-beam A pulsed laser beam as the primary light source with scattered light emitted along the beam path.

See the LightEmission package documentation for more details.

Chapter 7

Compilation options for `sim_telarray`

The `sim_telarray` program has a few built-in limits that have been set up originally for a 16 telescope HESS array. However, the `hessio` library needed by `sim_telarray` had default limits set up for a 4 telescope HESS array, for optimum performance in a memory constrained environment. Now, simulations are under way with more than 4 telescopes, with larger telescopes (more mirrors, more pixels, other trigger logic). And memory constraints are not so severe as they were a couple of years ago. Default settings now compile both `hessio` and `sim_telarray` for up to 16 telescopes with up to 4096 pixels. Old settings are used when you build libraries/binaries with

```
( cd $HESSROOT/hessio && make EXTRA_OPTIONS='-DHESS_PHASE_1' )
( cd $HESSROOT/sim_telarray && make EXTRA_OPTIONS='-DHESS_PHASE_1' )
```

If different settings are needed (e.g. more than 16 telescopes), see definitions near the beginning of the header files `hessio/hess/io_hess.h` (or `hessioxxx/include/io_hess.h`, depending on the source code distribution you use) and `sim_telarray/common/mc_aux.h`. Some of them can be changed via the `EXTRA_OPTIONS` parameter with `make` while others, less likely to be modified, may need changes to the mentioned header files.
Example:

```
( cd $HESSROOT/hessio && make EXTRA_OPTIONS='-DH_MAX_TEL=64' )
( cd $HESSROOT/sim_telarray && make \
    EXTRA_OPTIONS='-DMAX_TEL=64 -DH_MAX_TEL=64' )
```

Note that `sim_telarray` must always be built with the same settings of the `H_MAX_...` definitions as used for compiling `hessio`! Since these definitions depend on the `HESS_PHASE_`, `[NO_]LARGE_TELESCOPE`, and `[NO_]SMARTPIXEL` definitions, as well as the various `CTA*` definition used over time, those must match too.

Most of the changes to `mc_aux.h` won't require recompilation of `hessio` but a few are only useful if corresponding limits from `io_hess.h` are overridden to at least the

same size as in `mc_aux.h`. A few definitions in `mc_aux.h` are explained in the following (and corresponding limits in `io_hess.h` mentioned, if any):

`MAX_ARRAY` The maximum number of re-using a shower (as defined in the CORSIKA input parameter CSCAT) that can be handled by `sim_telarray`. This limit should only be changed by compiling with a suitable definition of the MAXIMUM_ARRAYS parameter.
Default: 100

`MAX_TEL` The maximum number of telescopes in the CORSIKA output that can be handled. *The* `H_MAX_TEL` *definition in* `io_hess.h` *should also have a sufficient value and the* `MAXIMUM_TELESCOPES` *run-time parameter may need modification.*
Default: 16

`MAX_IGNORE` The maximum number of telescopes in the CORSIKA output which are ignored in the telescope simulation.
Default: 15

`MAX_MIRRORS` The maximum number of mirror tiles per telescope.
Default: 1500 with `LARGE_TELESCOPE` option (500 otherwise).

`MAX_PIXELS` The maximum number of pixels per camera. *The* `H_MAX_PIX` *definition in* `io_hess.h` *should also have a sufficient value.*
Default: 4096 with `LARGE_TELESCOPE` option (960 otherwise).

`MAX_FADC_SIGNAL` The largest value that can be registered by the (F)ADC in one time slice.
Default: 2048

`MAX_FADC_BINS` The maximum number of time slices for which the digitzed signal can be simulated.
Default: 128

`MAX_TRIG_BINS` The maximum number of time slices for which the trigger logic can be simulated. Note that internally these time slices are sub-divided into 2^n sub-steps, depending on the `DISC_BITS_...+` definition ($n = 0...5$).
Default: 128

`DISC_BITS_...` Normally the logic (comparator or discriminator, telescope trigger) signals are simulated in 0.5 ns steps, compared to 1 ns for the digitisation. Define one of `DISC_BITS_`, `DISC_BITS_` `DISC_BITS_`, `DISC_BITS_`, `DISC_BITS_`, or `DISC_BITS_` to simulate logic signals in 1/1, 1/2, 1/4, etc. to 1/32 of the digitisation time step (which is defined by the run-time parameter `FADC_MHZ`).
Default: `DISC_BITS_2` is defined.

`MAX_PER_CHANNELS` The number of digitizers used in round-robin fashion to digitise signals of one readout channel.
Default: 1 (and conversion to H.E.S.S. output will not work with any other value).

WITH_LOW_GAIN_CHANNEL If defined, then each PMT is read out into two channels, a high-gain and a low-gain channel.
Default: defined (and required for H.E.S.S. output).

WITH_BYPASS_OPTICS Enable the experimental code for bypassing the optics ray-tracing (activated then at run-time with BYPASS_OPTICS=1 or 2).

RAYTRACING_INTERSECT_RODS If enabled, an addition data file MASTS_FILE can include obscuring elements, typically support structure elements, of filled cylindrical geometry.

ADDITIONAL_AFTERPULSING If active, allows using an alternate implementation of PMT afterpulsing signals at run-time, throwing prompt NSB signals like Cherenkov signals plus addition signals following an exponential distribution.

EXTRA_CLOUD In addition to the continuous extinction by Rayleigh scattering, Mie scattering and absorption as tabulated, a single geometrically thin absorbing layer of wavelength-independent (gray) extinction can be added.

STORE_PIX_PHOTONS This option disables the usual optimization short-cuts in the simulation. All photons get traced down to the pixels, rather than getting rid of most of them before the ray-tracing. Thus significant impact on CPU time. Only recommended for special simulations if you want to record the number of photons hitting each pixel (either all photons or only those in the 330-550 nm range, see parameter SAVE_PHOTONS).

For reasons of convenience a number of preprocessor flags are available to combine setting the above limits as appropriate for simulations in a certain context. You may want to set just one of:

HESS_PHASE_ Four telescopes with up to 960 pixels.

HESS_PHASE_ Add support for a fifth telescope with more pixels.

HESS_PHASE_D **or** HESS_PHASE_=2 HESS CT5 with FlashCam, enabled for 16-bit read-out (instead of non-linear response).

CTA Up to 100, possibly large telescopes.

CTA_PROD1 **or** CTA_ULTRA3 Up to 275 telescopes with other limits set to match actual limits in the CTA Prod-1 MC production (with "ULTRA3" configuration files), in order to conserve memory.

CTA_PROD2 **or** CTA_ULTRA5 Up to 200 telescopes for CTA Prod-2 MC production without SCTs ("ULTRA5" configuration files). CTA_MINI2 is a variant for fewer telescopes.

`CTA_PROD2_SC` Up to 229 telescopes for CTA Prod-2 MC production with SCTs (also "ULTRA5" configuration files).

`CTA_PROD3`**,** `CTA_MINI3`**,** `CTA_PROD3_SC` are similar definitions used the CTA Prod-3 MC production ("ULTRA6" configuration files).

`CTA_PROD3_DEMO` is suitable for any specific CTA layout, with or without SCTs (up to 126 telescopes) but not the super-layouts used the Prod-1/2/3/4.

`CTA_PROD4` is matched to CTA South baseline layout simulations. Similar for `CTA_PROD5` and `CTA_PROD6`, except for different limits on the number of telescopes. Plus mini-array or SCT-enabled variants of those similar to Prod-3.

`CTA_MAX` **and** `CTA_MAX_SC` with limits big enough to include all other simulations without or with SCTs, respectively, but at a high cost in terms of main memory usage.

See the mc_aux.h and io_hess.h files for other possibilities. Preprocessor definitions like `MAXIMUM_TELESCOPES`, `MAXIMUM_PIXELS`, and `MAXIMUM_SLICES` can be used to tailor the memory usage for specific applications without touching the code. All of these settings have consistent meanings for the `sim_telarray` and `hessio` compilation. `Sim_telarray` will fail if inconsistent definitions are used.

*Note: On i*86 Linux, the Makefile here, as for CORSIKA, takes care of compiling with support for files exceeding 2 Gigabytes. Actually, this is taken care of by the* `hessio` *library. Since* `-D_FILE_OFFSET_BITS=64` *conflicts at present with ROOT, it uses* `-D_LARGEFILE64_SOURCE` *and uses explicit* `fopen64` *etc. calls. If compiled without either of these flags, you will not be able to read large CORSIKA files on 32-bit systems.*

Chapter 8

Building `sim_telarray`

The first step again is compiling. This should be easy if you have all necessary source code available. With the stand-alone distribution consisting of `hessioxxx.tar.gz` and `sim_telarray.tar.gz`, you unpack the tar files side-by-side, first go into the hessioxxx directory, `make` (for possible options see below), then go into the sim_telarray directory and do the same there. In the H.E.S.S. framework, you should have the `hessio` code installed from CVS in `hessio`. The `sim_hessarray.tar.gz` (combining files otherwise found in `sim_telarray.tar.gz` and `sim_telarray_config.tar.gz`) is best installed in the directory referred to by the environment variable `HESSROOT`, creating a sub-directory sim_telarray, as for the stand-alone distribution. Apart from that little difference,

```
make
```

should be enough to compile and link `sim_telarray`. The `Makefile` is made to work under Linux (and in the old days also DEC Unix) with GNU `make`, reported to be working also under Mac OSX but not tested by its author.

There are more bells and whistles for compiling. You can modify the Makefile if you want other options for diagnostic output, e.g. `-DDEBUG_TEST_ALL`. But that is the point where you should have a look into the source code. Generally such extra definitions can be passed to `make` without modifying any Makefiles as in

```
make CDEBUGFLAGS="-g -O0" EXTRA_DEFINES="-DCTA -DDEBUG_TEST_ALL"
```

Keep in mind that any definitions that change the size of data structures (see compilation options) must be used identical for building hessio(xxx) and `sim_telarray`. When changing them, it is recommended to completely rebuild everything in both directories. You should also be aware that with very big data structures there may be special compiler flags necessary (e.g. for the memory model, `-mcmodel=large`) or there may be preparations necessary at the OS level before running it, like

```
ulimit -S -s 12000
```

if the default stack size limit is smaller (typically 8192).

Chapter 9

Usage of `sim_telarray`

For running `sim_telarray` like illustrated in [5] you should have the following files available – assuming you don't change the name of the configuration file on the command line or the other names either on the command line or in the configuration file:

`hess.cfg`	General configuration file for H.E.S.S. phase 1.
`hess_bestguess.cfg`	Detailed parameters for all H.E.S.S. phase 1 stages.
`atm_trans_1800_1_4_0_0.dat`	Atmospheric transmission model.
`hess_mirrors.dat`	Mirror positions, sizes, and types.
`hess_camera.dat`	The camera definition (pixels, trigger).
`hess_funnels.dat`	The angular efficiency of funnels.
`hess_qe2.dat`	The quantum efficiency as measured in Heidelberg (for $\langle S_{CB} \rangle = 11.6\,\mu$A/Lm).
`hess_reflect.dat`	The mirror reflectivity.
`hess_spe2.dat`	The single photo-electron response.
`hess_disc_shape.dat`	Pulse shape at the pixel comparators.
`hess_fadc_shape.dat`	Pulse shape at the ADCs.
`atmprof*.dat`	As for the CORSIKA simulations.

For more information on these files see section 11. For most of these files there are alternatives available to check possible systematics. There are further data files which are used only with special-purpose compile-time configurations, like the optional (but very time-consuming) shadowing by masts, spanning rods, and camera lid, reading from `masts.dat`.

A large number of additional files were prepared for (and with) the HESS and CTA communities and distributions of these additional files may be more limited.

For a complete list of possible parameters in the configuration files see section 12. In addition you might have a file with star positions, perhaps called `stars.dat` (by default no such file is read, the name needs to be configured). Such a file would consist of one line per star, giving the azimuth, altitude and 'flux' of the star (the 'flux' being roughly

in units of photoelectrons per nanosecond in a 'typical' PM). For details see the STARS configuration keyword.

Now having all the necessary files starting `sim_telarray` can be as simple as

```
sim_telarray
```

or somewhat more complicated

```
sim_telarray -C telescope_azimuth=180 \
  -C nightsky_background=all:0.17 \
  -h test.hdata iact*.dat
```

The following command line options are recognised:

-c fname Use configuration from file `fname`.

-h fname Write histograms (in `eventio` format) to file `fname`. Conversion of these histogram files to HBOOK format is done with the `hdata2hbook` program, to the ROOT format with `hdata2root`. Contents can also be extracted as text with the `list_histograms` program. The contents of many histogram files (for all matching histograms in them) can be added up with the `add_histograms` program.

-i fname Use input file `fname`.

-p fname Write a couple of ASCII tables for plots to `fname`.

-o fname Write the output telescope raw data to file `fname`. This is in some `eventio` format, depending on compilation flags and configuration options. The same result is achieved through `-C output_file=fname`. **Note that by default no output is written.**

-r fname Similar to `-o` but existing output will be replaced. Effectively the same as `-o fname -n`.

-n Output to a new file rather than appending to existing file.

-l alpha Set power law index of spectrum to `alpha`. By appropriate event weights the generated spectrum is corrected to the desired spectrum as far as the histograms are concerned.

-C something Interpret `something` as a configuration statement (like it is interpreted in the configuration file; it supercedes all corresponding statements in the configuration file).

-W something This is similar to the '-C' option but gets interpreted before the configuration file(s). If the same parameter gets set by the file(s) that will supercede the '-W' value. For that reason, it is called a *weak* configuration setting.

-Dflag Define a flag or variable for the configuration pre-processor `pfp` which processes the main configuration file (example: `-DCTA=1`).

-Ipath Include files in the main configuration file are searched in `path` (example: `-Icfg/CTA`).

-V or –version Report the version of the code used. This will normally include the date and time of the last source code change and by whom and where it was made and built.

For each configuration parameter there are effectively four levels of priority: 1) the hard-coded defaults as the lowest priority because they get processed first; 2) the weak command-line settings ('-W' option) which get processed after the hard-coded defaults but before any configuration file; 3) the configuration file after it gets pre-processed with the chosen defines and the chosen pre-processor; 4) the (hard) command-line settings ('-C' option) being processed after all others and superceding any prior values. Note that functions assigned to configuration parameters are called whenever they appear in the corresponding stage. Take as an example the 'SHOW=all' as a call to the internal 'SHOW' function. It does not have a hard-coded default, so never gets called in stage 1. A '-W show=all' would show the parameters before processing the configuration file(s). Only '-C show=all' will show that status of the configuration after processing the configuration file(s) and after processing all earlier '-C' options.

Any remaining command line arguments are interpreted as file names of input files containing photon bunches and other data in `eventio` format.

Note that all files with names ending in `.gz`*,* `.bz2`*,* `.lzo`*,* `.lz4`*,* `.lzma`*,* `.xz`*, or* `.zst` *are assumed to be compressed and reading/writing will be through on-the-fly decompression/compression with the* `gzip`*,* `bzip2`*,* `lzop`*,* `lz4`*,* `lzma`*,* `xz`*, or* `zstd` *commands, respectively. This is the case for all input, output, configuration files, histogram files etc. The necessary programs are assumed to be available on the system.*

Chapter 10

Coordinates and coordinate systems in the simulations

10.1 Overview

For the most part, the simulation system does not actually deal with geographic or celestial coordinates but assumes a locally flat 'observation level' at a given altitude above sea level. Both CORSIKA and `sim_telarray` follow a *nwu* (North/West/Up) coordinate convention at this point but also differ in their convention and in particular in the usage of angles. CORSIKA (for efficiency reasons) handles all shower simulations in a system where the geomagnetic field is in the *x*-*z* plane, i.e. the *x* axis points to geomagnetic North, with directions *towards* which the particles (the primary particle, for example) propagate and the horizontal angle ϕ (as in input parameter PHIP) counted counter-clockwise from geomagnetic North. `Sim_telarray`, on the other hand, has its coordinate system rotated by the geomagnetic declination angle, with the *x* axis pointing to geographic North. The azimuth angle *A* *from* where a particle is coming, is counted clockwise from geographic North, following astronomical conventions. Already at this point some level of detail is needed for the proper definition of coordinate systems.

Given a geomagnetic field (B_x, B_y, B_z – note that by convention B_z, for example with the International Geomagnetic Reference Field IGRF, is given positive when the field points downward, as typical so far in the Northern hemisphere, *x* towards North, *y* towards West, thus effectively defined in a left-handed system), with a geomagnetic declination angle Δ counted positive if geomagnetic North is west of geographic North (programmed as: `Delta = atan2(By,Bx)`). The CORSIKA ϕ angle for a particle coming from azimuth angle *A* is

$$\phi \;=\; \Delta + 180 - A.$$

For building an observatory of IACTs or any other telescopes (from the infrastructure point of view) and for operating it (mainly from the point of view of the necessary software for operating the telescopes, like pointing them in the right direction, and from preparing

the data for science results) a number of coordinate system get involved. In a very simplified world, these would be celestial coordinates, Right Ascension (RA, α) and Declination (Dec, δ), local sky coordinates, Azimuth (Az, A) and Altitude (Alt, a), space coordinates in the telescopes and around the observatory (x, y, z), as well as the geographical location of the observatory, (geodetic) Longitude (λ, by international convention counted eastward from the "Greenwhich" mercator) and Latitude (φ).

For a number of reasons, like the motion of the Earth, together with the Moon, around the Sun, the rotation of the Earth around an axis changing its orientation with respect to celestial coordinates and also (but far less) its location and orientation with respect to the Earth's body, the Earth deforming due to tidal forces from the Moon and the Sun, movement of continental plates, density inhomogeneities in the Earth's crust and mantle, curvature of the Earth's surface at the observatory, or bending of telescope structural elements by gravity, wind loads, and thermal stress the reality gets way more complex than the set of five coordinate systems introduced in the first paragraph. We will start from the simplified world and add more details as we go. As a starting point we put an infrastructure person (engineer, architect, or whatever) in the so-defined "central" location of an observatory, extending from there.

10.2 Site location and telescope positions

The location of the "central" point of an observatory can be given in different coordinate system, with respect to some geoid model and geodetic system (or "datum", as it also gets called). The most frequent coordinate systems are geodetic longitude and latitude w.r.t. to some ellipsoid as a geoid model plus height above the geoid model "sea level" or the Universal Transverse Mercator (UTM) set of coordinates. The most frequently used geoid and datum is perhaps the World Geodetic System 1984 (WGS84), also used by the Global Positioning System (GPS). Regional definitions for official maps may differ, for example by some 200 meters in Australia. Differing definitions, e.g. in what "sea level" means have caused problems in building bridges between two countries.

Thus coordinates in any system must identify the assumed geoid model and datum. We suggest sticking to GPS and WGS84 where possible (maps needed for building permits may differ due to national or state codes requiring a different datum) and never dropping the assumed geoid/datum. The WGS84 is actually defined by the locations of the GPS tracking and control stations and regularly updated to be aligned with the International Terrestrial Reference Frame (ITRF) within 0.1 meters. The actual version of WGS84 used is indicated by a GPS week number, e.g. "WGS84 (G1150)". The ITRF itself, based on GPS, Satellite Laser Ranging (SLR), Very-Long-Baseline Interferometry (VLBI), and other measurement methods, and balancing the individual continental drift vectors, also sees occasional revisions, the latest being ITRF2014 although that is not much in use yet. Where a 0.1 meters accuracy is good enough (which corresponds to a mere 0.02 arc seconds on an angular scale) the up-to-date representation of WGS84 coordinates with the GPS is a good enough representation of the ITRF of whatever revision as well. The Galileo navigation system will also use ITRF (ITRF2005 currently). Site coordinates may need

occasional updates for continental drift although that is well below a meter per decade everywhere (most are below half a meter per decade). GPS/Galileo measurements with suitable devices can be as accurate as about a centimeter although it takes a while.

As far as the UTM system – based on a system developed by the German Airforce in WW II and further developed by U.S. forces – is concerned, one should be aware of artefacts coming with the way it tries to map places on an assumed ellipsoid onto flat coordinates. It devides the Earth into 60 longitude zones, each covering 6 degrees in longitude and then basically wraps a flat grid (think of paper) around the ellipsoid, pole-to-pole along the mercator for the reference longitude of the zone. As this would overestimate all distances, it is actually not around the ellipsoid but intersects with it, around 0.9996 times the geoid size. Depending on where in an UTM zone you are it may then under- or overestimate distances. Distances should always be calculated directly in the ellipsoid system. Also note any altitude values coming with UTM are always above the geoid and not above the wrapped-around coordinate sheet and that UTM is not suitable around the poles. Coordinate values given with UTM are the "Northing" (N), counted in meters from the Equator for the northern hemisphere and from the South Pole for the southern hemishere, and the "Easting" (E), with a value of 500 000 m for the reference longitude and increasing eastwards. To be unique, the Northing and Easting values must be complemented by the zone identifier (incremented eastwards from longitude 180, i.e. the first being between longitude 180 and 174 West). CTA-South, at longitude -70.3° (70.3°W), latitude -24.7°(24.7°S) for example is in longitude zone 19 (from 72°W to 66°W), with suffix letters up to M indicating the Southern Hemisphere (here: 19J), suffix letters N and beyond indicating the Northern Hemisphere. CTA-North at 28.8°N, 17.9°W is in zone 28R. UTM coordinates are usually assuming the older GRS80 ellipsoid and not WGS84 but differences between these two are negligible for the level of accuracy needed with IACTs.

Convenient transformation of coordinates between longitude/latitude and UTM is, for example, possible with the cs2cs (coordinate-system-to-coordinate-system) program from the "Proj" software package[1] (here assuming WGS84 and CTA-South which is in UTM zone "19J"):

```
cs2cs +proj=longlat +datum=WGS84 +to +proj=utm +zone=19 +south
```

and reverse with

```
cs2cs -f %.7f +proj=utm +zone=19 +south +to +proj=longlat +datum=WGS84
```

Note that in Python or C/C++ code using the 'proj' package, the UTM system for the Southern site (zone 19J) is also known by the name 'EPSG:32719'[2], that for the Northern site (zone 28R) by the name 'EPSG:32628', while the WGS84 longitude latitude system is 'EPSG:4326'. The ITRF 2014 geocentric x/y/z coordinates are known as 'EPSG:7789'. When using the corresponding transformations in your own code, keep in mind that 32-bit floating point numbers are not precise enough. You will need 64-bit values.

[1] https://github.com/OSGeo/proj.4.git

[2] See EPSG web site at https://epsg.org/home.html.

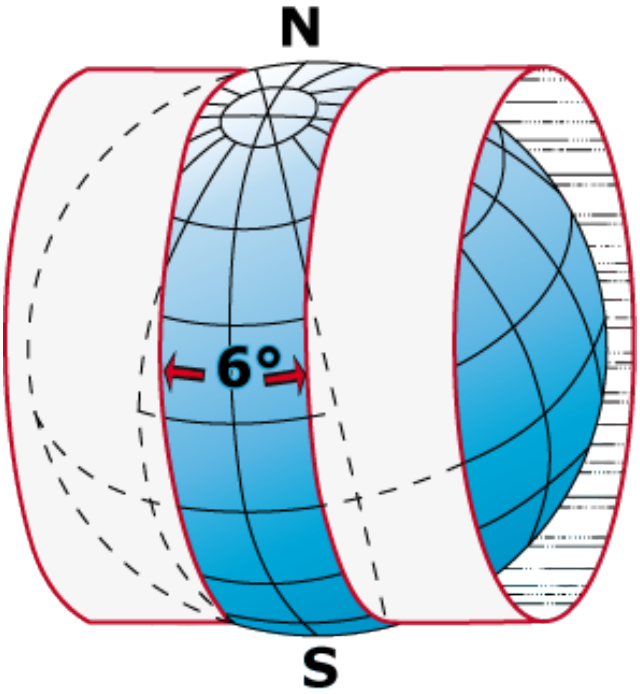

Figure 10.1: Universal Transverse Mercator (UTM) grid system for one longitude zone.

The locations of telescope positions on the observatory site can be in the same coordinate system as the "central" position, longitude/latitude or UTM. They can also be in a more local system w.r.t. to the "central" position. Whether this is a Local Transverse Mercator system (wrapped around the geoid through the "central" place) or a tangential plane has negligible impact on X and Y coordinates over the kilometer-scale (or less) extension of IACT systems. The height h should, for consistency, still be the one above the geoid. Note that the difference to an Cartesian Z coordinate is about eight centimeters at a kilometer distance from the "central" place – not considered a problem in simulations which usually assume a flat Earth and atmosphere model. Where higher precision is required the difference between curved and flat Earth should not be neglected though (like in telescope pointing).

For consistency with existing simulation programs (like CORSIKA and `sim_telarray`) as well as existing analysis packages, we define the site-local Cartesian coordinate system in the "nwu" (North/West/Up) convention:

X axis points North, $X_{\text{central}} = 0$
Y axis points West, $Y_{\text{central}} = 0$
Z axis points Up, $Z_{\text{central}} = h_{\text{central}}$ (height above the geoid = "sea level")

Example transformations between longitude/latitude and Local Transverse Mercator X and Y, for an assumed central position are obtained with

```
cs2cs +proj=longlat +datum=WGS84 +to +proj=tmerc +ellps=WGS84 +datum=WGS84 +lon_=-70.3163449936 +lat_=-24.6834291530 +axis=nwu +units=m +k_=1.000339223402
```

for longitude/latitude to X/Y (at CTA-South) and reverse with

```
cs2cs -f %.7f +proj=tmerc +ellps=WGS84 +datum=WGS84 +lon_=-70.3163449936 +lat_=-24.6834291530 +axis=nwu +units=m +k_=1.000339223402 +to +proj=longlat +datum=WGS84
```

The `k_0` assignment takes care of the site altitude (think of the transverse Mercator being a roll of paper wrapped, across the poles, around the ellipsoid; we need an ellipsoid enlarged here such that it touches the site altitude; `k_0=1.0` is sea level). Instead of Transverse Mercator ('tmerc' for cs2cs) the Swiss Oblique Mercator ('somerc') projection could be used as an alternative, with no noticeable difference at the scale of any IACT system.[3] The most exact transformation would be into the ITRF geocentric x/y/z frame, subtracting the corresponding reference point x/y/z, followed by rotations according to geographic longitude and latitude. That would even take the curvature of the Earth into account.

At this point we still neglect that the Earth does not exactly have the shape of an ellipsoid (the Earth Gravitational Model, EGM2008, for example, has 2159 spherical harmonics) and that the actual "Up" as defined by the inverse of the direction of local gravity at the "central" place may differ a bit from the normal to the ellipsoid (by definition given by λ and φ). It can be expected that any such deviation would be corrected for by the telescope pointing models.

In summary, accurate GPS measurements of a site's "central" location and perhaps a few other points on the site, in the WGS84 datum used by GPS are good enough representations of their positions in the current ITRF, which is the basis of transformations between terrestrial and celestial reference frames. Coordinates given in the UTM system are **not** suitable for evaluating distances due to distortions inherent to the system. Transformations between UTM and longitude/latitude and, for a given "central" location, also site-local Cartesian coordinates (in "nwu" convention, X towards North, Y towards West, and Z is Up) are readily available. For convenience, telescope positions should be listed in all three system. Note that Earth curvature is not entirely negligible across the site.

[3] If the Proj package has a projection into a tangential plane, I have not found it yet - but as the comparison between Transverse Mercator and Swiss Oblique Mercator shows any differences are way below millimeter level. In the case of a sphere that is easy to calculate. Say at 1 km distance along the surface on a, say, $r = 6371$ km sphere its parallel projection onto a tangential plane is 6371 km $\times$ sin(1/6371) and its central projection 6371 km $\times$ tan(1/6371), both within 10 micrometers from 1 km. Since the Earth is close enough to a sphere, this must also hold for the geoid.

10.3 Coordinate systems on the telescopes

To avoid unnecessary transformations, the definition of Cartesian coordinates in the absence of any deformations and camera rotation as well as Earth curvature is very simple:

```
All coordinate systems are aligned when a telescope
points to zenith at azimuth zero (x=North, y=West, z=Up,
telescope points to the North for positive zenith angles).
```

In particular, it links the ground Cartesian system (X, Y, Z) with any system on the telescope (x, y, z). This convention is used by all current simulation and analysis packages. In the absence of Geomagnetic Field (GF) declination it also matches the system used inside the CORSIKA shower simulation package, with output through the IACT/ATMO package corrected for GF declination being always in the defined coordinate system. Where the z coordinate of a component is implied by other parameters (e.g. pixel z positions are implied by the camera curvature), it is sufficient to provide x and y coordinates (note that these are projections and not measured along a curved surface).

Note that the above definition implies that in a telescope rotated in altitude or zenith angle to point to the horizon at azimuth zero the x axis now points downwards, the y axis still points to the West and the z axis now points North.

If pixels in a camera are to be represented in a system where (in the absence of focal surface curvature) the square or hexagonal pixel shapes are aligned with the system, then an additional rotation angle between the pixel and camera coordinate system is needed, with the latter following the general definition. For ray-tracing in simulations, where pixel shapes are important, this is necessary. For other purposes, like image reconstruction, it may be optional, i.e. de-rotated pixel positions could be used, but for consistency reasons the same distinction between pixel and camera coordinate systems is recommended.

Transformations between two directly linked coordinate systems can generally be represented by the following sequence: a displacement, a rotation around an axis, an optional intermediate displacement, a rotation around another axis, and a final displacement. The intermediate displacement is necessary where the two axes do not intersect, e.g. in Alt/Az telecope mounts where the altitude axis is displaced with respect to the azimuth axis. Otherwise the two rotations can be combined into a single rotation matrix.

While the Earth's curvature is not very relevant for the telescope Z coordinates in a site-local Cartesian system, it is relevant for the reconstruction of source positions in the sky. A kilometer distance on the ground corresponds to about half an arc minute difference of the Up direction. There are thus two issues involved and decisions to be made:

- Should telescopes be built with each 'azimuth' axis according to local gravity or should all azimuth axes be aligned with the central telescope?

- Independent of that (and keeping in mind that alignment will never be perfect) the pointing corrections for each telescope could correct to a common frame, aligned with the site-local Cartesian system, or to individual frames for each telescope, according to their location.

The answer to the first question should probably be 'yes' but may require an additional effort in the construction phase, and may not have been considered at all by the infrastructure team. It will reduce the necessary corrections somewhat – but if corrections, for bending, initial alignment, sagging of foundations etc. are expected to be of that order anyway the effort during construction may not be worthwhile.

The answer to the second question is almost certainly 'yes'. Otherwise combining the data from individual telescopes into a common reference frame may be too inconvenient, error prone or CPU intense. But the alternatives need to be studied. Since simulations are generally for a flat Earth, the CORSIKA 'CURVED' option being of no help at the typical telescope separations, the simulated telescope axes are implicitely aligned in a common reference frame.

10.4 Telescope pointing considerations

10.4.1 Pointing concepts and corrections

In contrast to the ideal world in which the coordinate systems have been defined, and where the z axis in the reflector and camera coordinates systems define an optical axis, the real telescopes will have a deformed reflector, each mirror segment misaligned differently, the camera support structure bent, resulting in a camera displacement and maybe also a camera rotation. Even the telescope structure may be bent, not even leaving a fixed orientation of the azimuth axis. The whole foundation might be sagging on one side over time. As pointed out previously, the telescope will also not be built with perfect orientation, the real Earth is not flat (while in simulations it generally is) and local gravity is not perfectly aligned with the assumed geoid (ellipsoid).

What is important is that the telescopes can be positioned/aligned well enough to a target in the sky (moving due to Earth rotation) to allow data taking without loss of efficiency. Since point-spread function, trigger efficiency, etc. change slowly over the field-of-view, requirements to the accuracy for the nominal positioning are rather loose. For the shower reconstruction (as part of data reduction), the actual alignment/pointing must be known well enough that the angular resolution and gamma/hadron discrimination will not suffer, for each individual measured event. The strictest accuracy is needed for the position of an identified gamma-ray source in the sky - which will average over many measured events and will generally combine measurements taken in different observation periods, perhaps years apart.

The positioning/alignment will need some previously determined coefficients of a pointing model to achieve a sufficient accuracy, although feedback by optical sensors (e.g. CCD camera) may be used in addition. The event-wise information will generally need some hardware support, like shaft encoder, counting motor steps, or whatever, with readout at several Hertz but due to telescope inertia not necessary at the actual event rate, using some interpolation, and it will apply pointing model coefficients updated after the observations are taken. The source location accuracy is expected to be achieved after systematic studies with a sample of source known to be point-like and with accurately known po-

sitions, e.g. from radio VLBI. This accuracy is thus expected to be improving as more data is accumulated and more systematic effects get understood - as long as the event-wise information is good enough.

In any case, the task is to establish a mapping between pixel or camera coordinates and Alt/Az coordinates in the site-level Cartesian ground-based system. That includes not only the sky position imaged to the camera center (the optical axis for an idealized telescope), but also the alignment (camera rotation), the image scale (effective focal length) and non-linear distortions in the imaging. Whether a common type of pointing model (set of mapping functions) is applicable for all telescopes – although with type- and telescope-specific coefficients – or if telescope-type-specific models are needed (e.g. because the physical model for telescope bending depends on the type of telescope mount) is at this point not clear.

The image scale and non-linear distortions can be rather tricky in the presence of point-spread functions which are not radially symmetric. Any image cleaning may cut off more of the tails of an image on one side than on the other side. The image scale can then depend on the cleaning, and for shower images (in the Cherenkov camera) may differ from optical measurements (in CCD or CMOS cameras). For shower images it may even depend on the image amplitude (more intense images having a smaller fraction getting cut-off by the cleaning).

An observation made with ray-tracing simulations of the different telescope types in an IACT system is that using pixel positions using x/y projections result in a much more linear imaging for non-planar camers (in dual-mirror telescopes) than trying to define pixel positions along the curved surface. The common alignment of coordinates in the telescope also has the added benefit that no distinction between single-mirror telescopes (camera looking towards the primary/only mirror) and dual mirror telescopes (camera looking away from the primary mirror) is needed.

For the actual image reconstruction there are different ways to apply the necessary transformations and corrections. Shower images (Hillas parameters etc.) can be reconstructed in pixel or camera coordinates and the image parameters transformed into a common system, or every pixel position can be transformed into its corresponding Alt/Az coordinate in the common system. For shower model/template fits the transformation can be the other way around, starting from the shower model to the corresponding intensity expected in each pixel. Pointing models must thus work in both directions, with equivalent results. And they may be updated at a later time, thus the coefficients used so far must be identifiable. How pointing corrections are applied is not a topic of this paper.

At the end of the event reconstruction, a shower axis (direction and impact position) in the site-local Cartesian system and the Alt/Az coordinates derived from it are expected. What is unclear at this moment is where along this path the following corrections get applied:

– The atmospheric refraction of the shower Cherenkov light is only part of the refraction of star light as it originates from inside the atmosphere. While the refraction of star light only depends on ground-level air density (index of refraction, to be precise) and zenith angle, except near the horizon, the fraction applicable for shower images

also depends on the depth of shower maximum (and thus in a systematic way on energy).[13]

– Due to bending of electrons and positrons by the geomagnetic fields, with slightly more electrons than positrons in the shower, shower images are not only spread out in the bending plane of the GF but also distorted (major axis not pointing back to origin).[14, 15, 16, 17]

10.4.2 Parallel, convergent, and divergent pointing

By default, `sim_telarray` will assume that all telescopes point to the same direction in the sky (*parallel pointing*). For small to intermediate telescope arrays, pointing to the same place in the atmosphere, at a typical height of shower maxima, can improve the telescope multiplicity in most events, at the expense of losing events with large impact distance entirely. That is called *convergent pointing*. On the other hand, the pointing directions of telescopes can be spread out, increasing with distance from the center of the array, to increase the overall field-of-view, at the expense of the telescope multiplicity in individual events. This would be *divergent pointing*.

The convergent pointing is achieved by pointing all telescopes to a common point on an axis starting from a reference position (`CONVERGENT_POSition`) towards the nominal array viewing direction (`TELESCOPE_THETA`, `TELESCOPE_PHI` at global level), see Figure 10.2. As an extension of that, divergent pointing is achieved by the opposite of the telescope direction pointing to a common point on that axis, below the observation level.

In `sim_telarray` there are three possible ways to specifying convergent pointing in a very convenient way: a) a distance of this common point from the reference position along the given axis (`CONVERGENT_DISTance`), b) a height above the observation level (`CONVERGENT_Height`) where this common point should be, or c) an atmospheric depth along an inclined shower axis (`CONVERGENT_DEPTH`), for a shower coming from the array viewing direction. Only one of them is supposed to be activated - the latter choices taking priority if more than one is non-zero. Negative values activate divergent mode. While simple convergent or divergent point would have the same parameter for all telescopes, they can be set separately for each telescope (for example depending on telescope type).

Alternatively, all telescope pointing directions could be set individually in the configuration file to pre-calculated values, depending on the telescope ID (note: in that case neither `TELESCOPE_THETA` nor `TELESCOPE_PHI` must be set on the `sim_telarray` command line by the '-C' option since that would override the telescope-specific settings).

A question for the practical operation of a telescope array is which way the convergent/divergent pointing can be specified for continuous updating while tracking a source. For the `sim_telarray` configuration, basically representing a snapshot in telescope pointing for each simulation run, that is just a matter of how convenient it is to set up.

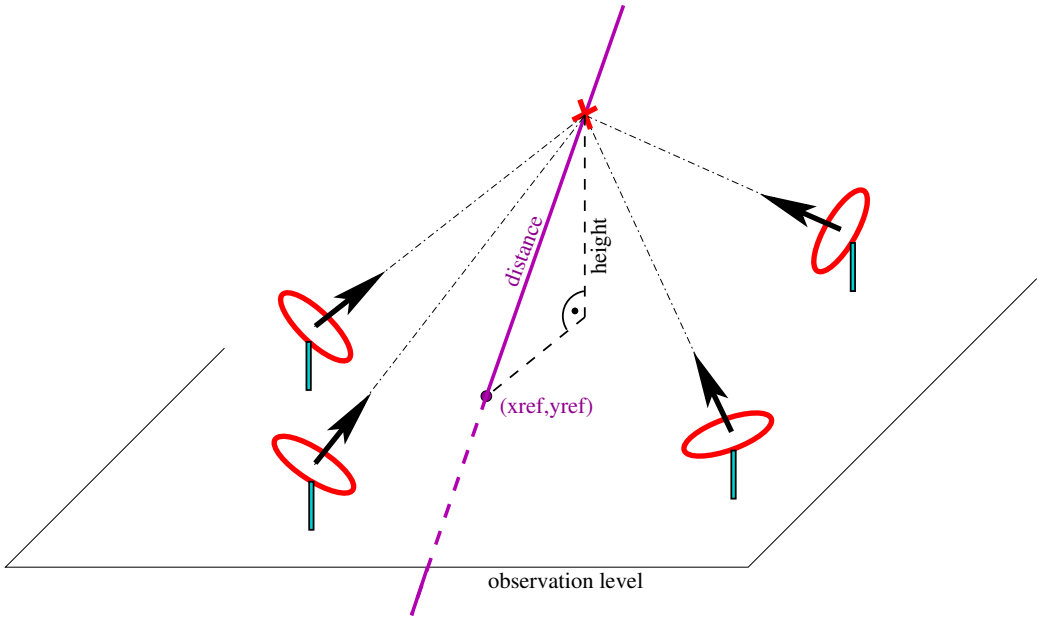

Figure 10.2: Convergent pointing by having all telescopes looking towards the same point on an axis through a reference position (usually the array center). Either the distance along this axis, its height above the observation level plane, or the atmospheric depth along the 'array viewing' incoming direction can be specified.

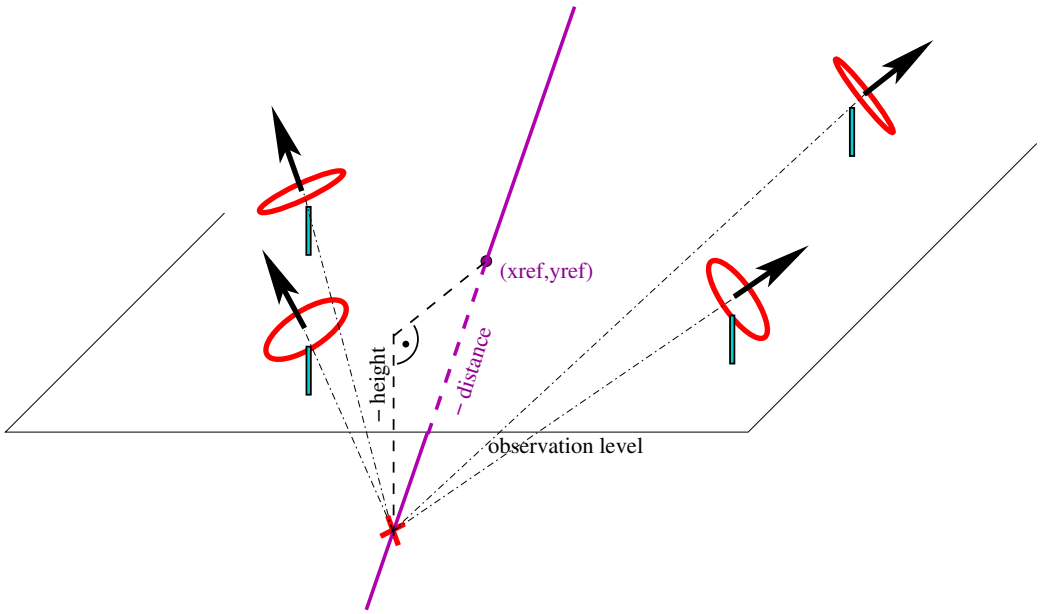

Figure 10.3: Divergent pointing by having all telescopes looking away from the same point on an axis through a reference position (usually the array center). Either the distance along this axis (negative), its height below the observation level plane (negative), or an atmospheric depth along the 'array viewing' incoming direction (negative of the value corresponding to a positive height) can be specified.

10.5 Celestial coordinates and transformation to/from the site-local Alt/Az system

The current system for celestial coordinates is Right Ascension α and Declination δ in the Interational Celestial Reference System (ICRS), with only small deviations from the system provided with the Fifth Fundamentalkatalog (FK5). It is worth noting that the FK5 predecessor, the FK4, suffered from large-scale measurement biases and transformation between FK4 and FK5 needs to compensate for the resulting distortions. This is notable because the system of Galactic coordinates was defined w.r.t. FK4 and the IAU never re-defined it w.r.t. ICRS. As a result, it generally is ambiguous if compensation for the FK4/FK5 distortion needs to be taken into account or not. With some astrometric software a set of transformations like Galactic $\rightarrow$ FK4 $\rightarrow$ FK5 $\rightarrow$ Galactic does not end up where it started. Use of Galactic longitude/latitude as primary coordinates is therefore discouraged and Galactic coordinates should only be used for illustrative purposis.

With the exception of solar system bodies, all targets can be represented by α and δ in the ICRS system for the epoch of J2000, by their parallax in arc seconds (or its inverse, the distance in parsec), and by their apparent proper motion in the ICRS J2000 system. For candidate targets of IACT observations, being at least hundreds of parsecs away the parallax and proper motion are generally negligible while for some stars used in the determination of pointing model corrections they may be significant. Thus transformations need to take them into account.

The transformation between instanteneous apparent Alt/Az coordinates in the site-local Cartesian ground system (linked to the ITRF through GPS/Galileo) and (apart from proper motion) fixed coordinates in the ICRS follows a procedure defined jointly by the International Astronomical Union (IAU) and the International Union of Geodesy and Geophysics (IUGG) which in 1987 set up the International Earth Rotation and Reference Sytems Service (IERS). The IAU-approved library for this task is called SOFA[4] and it comes with a license prohibiting any modification, not even for getting it compiled on a system not supported. Without the IAU branding, the same code is the basis for the ERFA library,[5] which is often preferred to SOFA in scientific applications. Other software, if verified against SOFA, may be suitable as well.

[4] `https://www.iausofa.org/`

[5] `https://github.com/liberfa/erfa`

Chapter 11

Data files and important parameters for sim_telarray

11.1 Configuration files and pre-processing

If the general configuration file (by default `hess.cfg` or `cta.cfg`, depending on compile-time definitions) is not present, the compiled values are used by default. For telescopes, where no telescope-specific configuration is found, the values from the general configuration file or the compiled values apply. Command-line configuration values specified by the '`-C`' (configure) option override both compiled values and values from the general configuration file and any configuration files included from there. Note that the command line options cannot address individual telescopes. Configuration values from the '`-W`' (weak configure) command-line option still override compiled values but not values from the configuration file(s).

Note that all of these files are searched in several directories. Unless the '`-I`' command line option is used when starting `sim_telarray`, these include the current working directory and the directories `$(PREFIX)/cfg/common/` and `$(PREFIX)/cfg/hess/` where `$(PREFIX)` normally is the directory where `sim_telarray` was compiled. Other directories like `$(PREFIX)/cfg/CTA` may also be included, depending on compile-time options.

The general (or main) configuration file has a special syntax and is pre-processed through the `pfp` program, somewhat similar to the C or C++ pre-processor but with more limited functionality and, in particular, more restrictive macro substitution. Configuration parameter assignments resulting after the pre-processing are described in section 12. Note that the configuration file has the percent character (`%`) for starting comments because the hash character (`#`), used for comments in configuration data files, is used by the pre-processor. For the available types of parameters and the syntax to assign actual values, see Section 12.1.

Some of the configuration data files also have a dedicated syntax, either with special keywords, like the camera definition file, see section 11.11, or have multiple values on a line, some of them required and some optional, like the mirror definition file, see section

11.7. Others are of a simple syntax with two values (x, y) separated by white-space (blank or tab characters) per non-comment line, suitable for 1-D interpolation. Keep in mind that some of them are clipped at the defined range (e.g. pulse shapes) while others are extrapolated with the value at the corresponding edge of the defined range (e.g. reflectivity).

The `pfp` configuration pre-processor can be exchanged against different programs with the `SIMTEL_CONFIG_PREPROCESSOR` environment variable (default setting is "`pfp -v -I.`"). A simplistic alternative, only picking the lines for the requested telescope, is `sspp` - with a different configuration file syntax, like (illustrating a few possible ways):

```
(@cfg) global<tab>ARRAY_CONFIG_NAME<tab>Paranal-baseline-prod5
0<tab>ARRAY_WINDOW = 1000
(@cfg) CT17<tab>QE_VARIATION<tab>0.03
17<tab>CAMERA_BODY_SHAPE 2
```

where `<tab>` stands for the ASCII character 0x09, which is required between the identification field (e.g. `global`) and the actual configuration line. A conforming configuration file for `sspp` use can be generated with `sim_telarray` itself, using the command line option

```
-C list=no-internal
```

and selecting, from its standard output, all lines starting with `(@cfg)` (a prefix which may or may not be stripped off).

Any alternate preprocessors have to accept the same syntax like the `pfp` (or the C pre-processor `cpp`) which includes any number of the "`-D`" (Define), "`-U`" (Un-define), and "`-I`" (Include-path) options, followed by one or two filename parameters (input and optional output, with "-" standing for standard input/output). Of the "`-D`" options it must respect at least the "`-DTELESCOPE=...`" definitions.

11.2 Data table 1-D interpolation with the rpolator code

The data tables used by `sim_telarray` all, except for atmospheric transmission, used to be read separately for each configured telescope, always one-dimensional (although some files like single-p.e. response or pulse shapes can have multiple y columns in parallel) and were always interpolated linearly. This has changed with introduction of the `rpolator.c` interpolation code. The rpolator

- loads files only once (unless different options are used),
- offers different interpolation schemes (see Figure 11.1),
- auto-detects if supporting points are at equidistant positions (resulting in faster lookup),
- offers mapping between linear and log scales, etc.,

- can be driven by a special (comment-style) control line in the data file.

The special control line is optional; if present it has to be the first line in the data file, with the following syntax:

```
#@RPOL@[ymarkup] ndim options
```

For the `ymarkup` and `ndim` parts see Section 11.3. For one-dimensional tables ndim should be 1.

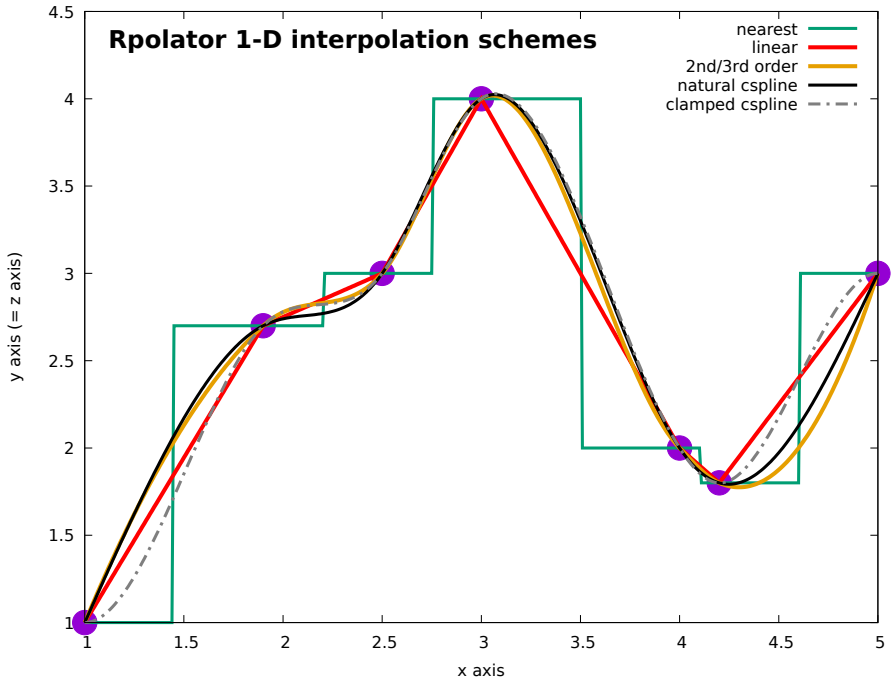

Figure 11.1: Interpolation schemes available with rpolator.c for 1-D tables.

The `#@RPOL@` header line may also include a number of options, after the string "`OPTIONS:´´` (all uppercase), if any. The function call to load the data table may extend on these options or override them. Known options (case ignored) include

CLIP or CLIP=1 or CLIP=on or CLIP=yes to enable clipping. Interpolation returns zero for lookups outside of the range provided in the table.

NOCLIP or CLIP=0 or CLIP=off or CLIP=no to disable clipping, using the value on the corresponding boundary to extrapolate with a constant.

ZXMAX or ZXMIN to indicate that for 2-D tables also the upper and lower bounds of the projection onto each x value are made available.

XCOL= or YCOL= or ZCOL= the index (starting at 1) of the column with the x or y or z values for format 3 or format 1 (only XCOL and YCOL) if the corresponding data is not in the natural columns 1, 2, and 3. You could re-use the same multi-column file for multiple 1-D interpolation purposes, e.g. for high-gain channel pulse shape using columns 1 and 2 and low-gain channel pulse shape using columns 1 and 3.

COLS= or COLUMNS= to give the expected number of columns for format 2, which can avoid having to re-allocate memory for the data, as long as the given number is accurate. Otherwise not needed.

SCHEME= to assign the interpolation scheme to use. A value of 0 indicates to use the nearest neighbour, 1 to use linear interpolation, 2 to use piece-wise polynomial interpolation of 2nd to 3rd order (only available for 1-D). Cubic splines (1-D) are available in two variants: natural csplines (second derivative goes to zero at the boundaries) or clamped csplines (first derivative goes to zero at the boundaries). Where the requested scheme is not available, it falls back to linear interpolation - which is also the default. For tables on a 2-D grid (see next section) it is always bi-linear interpolation.

XLOG or YLOG or ZLOG to internally convert values in the given coordinate to logarithmic units. Where enabled, any user-provided x or y values are first converted to log units before the interpolation and z values resulting from the interpolation are converted from log to linear before returning it. This may be useful if the underlying relation is more linear in the chosen log units but the calling function is not aware of it.

XSCALE= or YSCALE= or ZSCALE= to scale the chosen values by a given factor right after reading the table. Interpolation has to use the scaled values. A special scaling factor is `deg2rad` for $pi/180$ to show human-readable values in degrees in the table but internally use radians to avoid extra multiplications (`rad2deg` scales the other way round).

VERBOSE= to set the level of verbosity from rpolator functions.

ROWS= and COLS= (or COLUMNS=) may be useful as hints on how many rows and columns of data to expect which can avoid or reduce memory re-allocations while reading the table but is not really necessary.

Typically, the appropriate options are provided by the calling function to load a table. Note that a given table name with given caller options is only loaded once, sharing the pointer for all telescope types using the same table and options (lacking smart pointers in C no complete removal from memory is supported). In addition to a `#@RPOL@` line in the file (first processed, has lowest priority) and options provided by the calling function, the user can also include options by attaching them to the real file name after `#rpol:` (in a shell environment typically making it necessary to enclose the file name in quotes; no quotes needed in `sim_telarray` configuration files). Example file name with options:

`/tmp/c3.lis#rpol:scheme=3,clip`. This method is interpreted as the last one and thus would override conflicting options provided in the file itself or by the calling function.

The `rpolist` application is available to show information on what format a table is found, to list the values found in it (note: log settings get reverted for that) in a uniform two or three column format suitable for easy plotting, and two test-evaluate interpolation from the given table. This application can also do the interpolation. The curves in Figure 11.1 were, for example, obtained, from $x = 1$ to 5 in Δx steps of 0.01 with

```
rpolist -s <n> --ex 1,5,0.01 /tmp/c3.lis
```

where for `<n>` the values from 0, ..., 4 were used to illustrate the different schemes. A more explicit way to set the interpolation scheme would be

```
rpolist -o scheme=<n> --ex 1,5,0.01 /tmp/c3.lis
```

and like any other option that can also be attached to the file name:

```
rpolist --ex 1,5,0.01 "/tmp/c3.lis#rpol:scheme=<n>"
```

11.3 Switching from 1-D to 2-D table interpolation

Some of these traditionally simple data files are in the process of being upgraded to optional 2-D tables in the sense that existing 1-D tables are interpreted as they used to be while actual 2-D tables (different formats possible, with three values x, y, z being one possibility) enable an additional dimension for interpolation (angle of incidence for reflectvity and filter transmission, signal intensity for pulse shapes, ...). The optional data-file-driven switch from 1-D to 2-D is handled by the `rpolator` code component, given 1-D as the default format unless the first line of the data file starts with

```
#@RPOL@[ymarkup] ndim options
```

with `ndim` being either 1, 2, or 3. A value of 1 makes it an explicit 1-D table, a value of 2 a 2-D table with y values in a special line (which may be prefixed by the optional markup text indicated above as `ymarkup`), or a values of 3 to make it a 2-D table where the two lookup coordinates (x, y) and the resulting value (z) are explicit in each line. Since the 2-D tables can only be above rectangular grids (same intervals for x at any given y and vice-versa) the third format has to repeat the same x values for every y value and the same y values for every x value, in a systematic order, or the table is considered invalid. Possible table formats for a very short 2-D table with just two x values (1.1, 1.2), three y values (2.1, 2.2, 2.3) and six z values (3.1 to 3.6), without options, are:

```
#@RPOL@ 2
2.1 2.2 2.3
1.1 3.1 3.2 3.3
1.2 3.4 3.5 3.6
```

or

```
#@RPOL@[#YVALUES=] 2
#YVALUES= 2.1 2.2 2.3
1.1 3.1 3.2 3.3
1.2 3.4 3.5 3.6
```

or

```
#@RPOL@ 3
1.1 2.1 3.1
1.1 2.2 3.2
1.1 2.3 3.3
1.2 2.1 3.4
1.2 2.2 3.5
1.2 2.3 3.6
```

or

```
#@RPOL@ 3
1.1 2.1 3.1
1.2 2.1 3.4
1.1 2.2 3.2
1.2 2.2 3.5
1.1 2.3 3.3
1.2 2.3 3.6
```

Note that older, not rpolator-enabled, versions of `sim_telarray` may misinterpret such tables rather than failing to use them.

11.4 Atmospheric transmission

The table, from which atmospheric transmission values are obtained by interpolation, is configured through the `ATMOSPHERIC_TRANSMISSION` parameter of `sim_telarray`. There are a range of such tables available, all created with the MODTRAN[10] program. The default table `hess_atmo_trans.dat` is normally set up as a symbolic link to the transmission table for maritime haze and sea-level as the bottom of the boundary layer. Figure 11.2 shows a comparison of transmission from 10 km to 1.8 km for a vertical line-of-sight with different tables available for the H.E.S.S. site.

All transmission tables are in steps of 1 nm in wavelength and for a range of arbitrary altitudes, giving the optical depth for vertical transmission to the observation level. Which file is actually used, is controlled by the `ATMOSPHERIC_TRANSMISSION` parameter. Note that, while these files are handled by dedicated code, they are usable with the rpolator

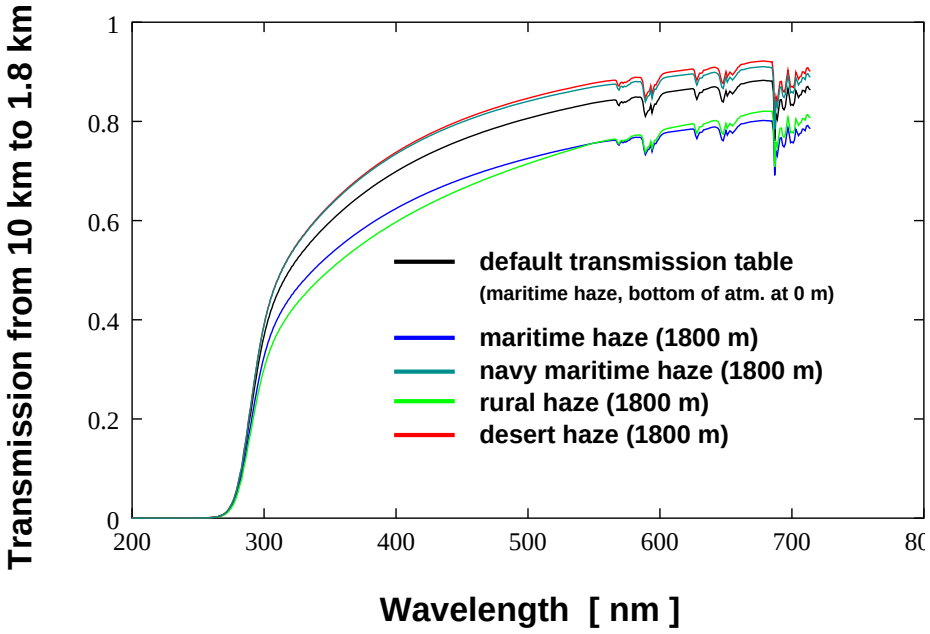

Figure 11.2: Transmission tables for the H.E.S.S. site.

code (and thus the rpolist application) with ndim=2 and a y-axis marker string of '`# H2=`' (rpolist command line: `-n 2 -m '# H2='`).

The H2 value of the transmission table should not be above the CORSIKA observation level – or you will get a warning that you seem to have an underground installation. For best results, it should also not be far below the observation level.

11.5 Atmospheric density profile

The tables with the atmospheric density profile are an important ingredient for the CORSIKA simulation (see the `ATMOSPHERE` / `IACT ATMOSPHERE` / `IACT ATMOFILE` parameters there), but may play a role as well in the telescope simulation. If the convergent tracking is specified and the level, where the lines-of-sight of the individual telescopes should intersect, is given in terms of the atmospheric depth (`CONVERGENT_Depth` parameter, in g/cm^2 vertically), the same table applies for finding the corresponding height. The same procedure would apply in the analysis of simulated data if CORSIKA was using the `FIXCHI` parameter to start the primary particle inside the atmosphere, passed along as `start_depth` (hessio) or `start_grammage` (pyeventio) variable. A similar lookup, although along the inclined arrival direction, applies for the data variables describing the shower maximum (Xmax for particles, Cmax for Cherenkov light).

Which file got used by CORSIKA used to be encoded (only) in the CORSIKA run header data block, in case of a `ATMOSPHERE` line in the CORSIKA inputs. That would indicate an atmospheric profile *number*, for a file `atmprof%d.dat` located in the include paths, with `%d` replaced by the profile number as used in CORSIKA. With the newer `IACT`

ATMOFILE as a more flexible alternative to ATMOSPHERE / IACT ATMOSPHERE, this would not work anymore (indicated as atmosphere number 99). Neither would it work if a parametrized rather than a tabulated density profile was specified in CORSIKA (indicated as atmosphere number zero). A special case is the CORSIKA default profile, corresponding to the U.S. Standard Atmosphere, which is represented by `atmprof6.dat`.

The actually used atmospheric profile is passed now from the CORSIKA IACT interface (since version 1.60) in the data stream down to `sim_telarray`, even if no density table but a parametrized 5-layer description was used. Where available, it includes both the table data and the 5-layer parameters. It also gets passed from `sim_telarray` into its output file. (Where older data or older software versions do not provide that data block, `sim_telarray` may be looking for the file again. Fortunately, all older `sim_telarray` productions used the ATMOSPHERE line in the CORSIKA input and the corresponding `atmprof%d.dat` gets loaded automatically.)

Note that interpolation of tabulated profile needs to take advantage of the approximately exponential density profile, also for the tabulated atmospheric depth (grammage), by interpolation height versus log(density), or log(grammage). At the top of the atmosphere, this breaks down as the assumed grammage goes gradually to zero while the density jumps to zero. There are specialized interpolation functions available where the generic `rpolator` would fail. Without those functions, using linear interpolation in height versus density (or grammage) in the first interval below the top of the atmosphere and linear interpolation in height versus log(density) (or log(grammage)) in the second interval should work. Below that, a natural cubic spline in height versus log(density) (or log(grammage)), set up without the two upper height levels, would be the best solution.

11.6 Assumptions and parameters defining the telescope geometry in `sim_telarray`

The geometry of the telescope components in `sim_telarray` is simplified on purpose. Most components that are not active in the optical path, like camera support etc., get ignored in normal operation, for efficiency reasons. The ratio between the effective mirror area in accurate ray-tracing versus the simplified-shadowing ray-tracing can be applied as a radially symmetric function of incidence angle (see TELESCOPE_TRANSMISSION) but could also be represented as a 2-D tabulated map of degraded efficiency at the primary or secondary mirror, or at the camera level.

The only element always casting a shadow is the camera box, which may come in circular, hexagonal, or square shape and with either a finite depth or just as a single layer. In dual-mirror telescopes, the shadowing by the outline of the secondary mirror would also be included but not the details of its segmentation (for example not rays coming through the gaps between segments). The shadowing (circular) area corresponding to the secondary mirror can be enlarged or reduced w.r.t. the actual secondary, and displaced along the optical axis, for a better match with shadowing by the actual telescope structure. Another component only available with dual-mirror telescopes is a cylindrical or conical baffle around

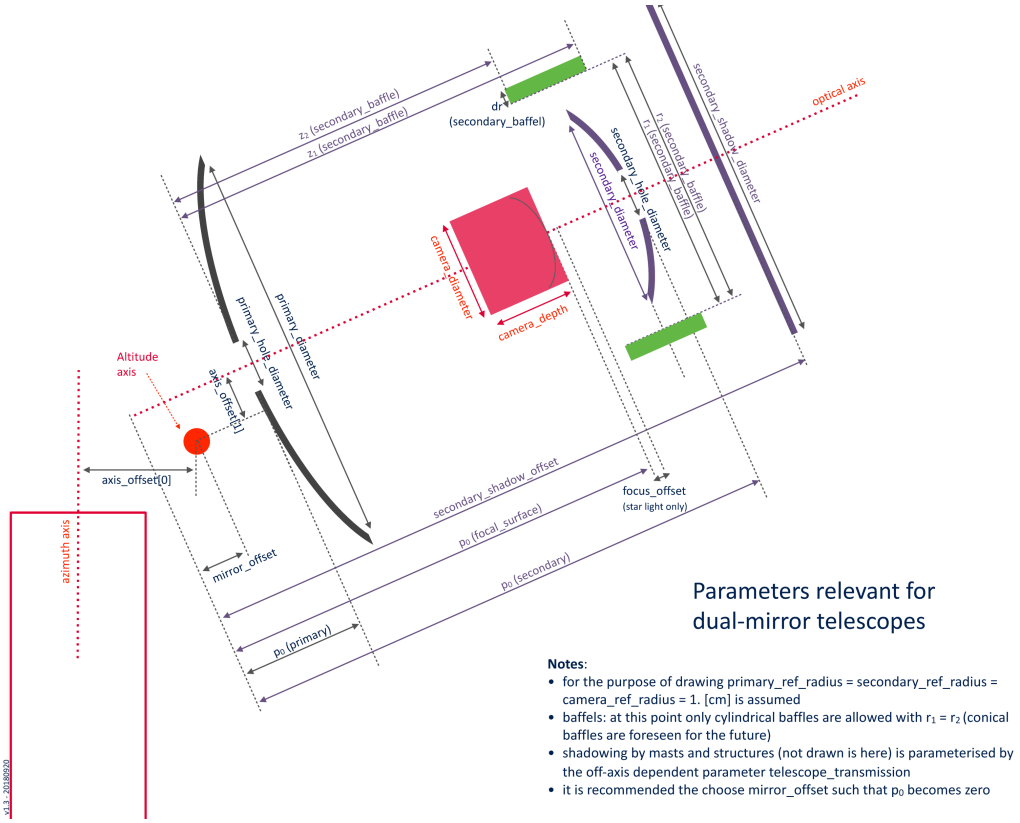

Figure 11.3: Relevant geometrical parameters for dual-mirror telescopes in `sim_telarray` (graphics courtesy G. Maier, DESY). Note that the z axis as used in the simulation is along the optical axis (here towards the upper right), the x axis is in the plane of the telescope tilting (as used in this illustration), perpendicular to the optical axis (here in the plane of the illustration towards the lower right) while the y axis, also perpendicular to the optical axis, can be thought as going into the plane of this illustration.

the secondary, typically used for reducing nightsky and stray light falling into the camera. A possible baffle around the primary, which may keep sunlight out in park position during daytime, is not relevant for the infalling Cherenkov light and not implemented.

The positions of mirror segments and camera pixels are listed in separate data files. Still, the number of parameters describing the basic telescope geometry is quite large. Which parameters are actually active for a given telescope type depends on the `MIRROR_CLASS` assignment. This includes segmented single reflectors with spherical surface segments, single parabolic reflectors, Schwarzschild-Couder type dual-mirror telescopes (segmented or non-segmented), as well as Fresnel lens optics. For all of them, the optical axis and/or the altitude axis can be displaced with respect to the azimuth axis (only Alt-Az mounts implemented, no equatorial mounts).

The segmented single reflectors, where the `MIRROR_LIST` table includes every segment position, size, and shape, the segment heights may follow a spherical or a parabolic shape, including intermediate radii of curvature. In that case, the parameters

PARABOLIC_DISH, DISH_SHAPE_LENGTH, and FOCAL_LENGTH complete the shape of the optics, with a fixed MIRROR_FOCAL_LENGTH for all segments or one calculated from its distance to the camera center. Or the segments can be completely specified by the corresponding table, and that may even include separate focal lengths for each mirror segment. Segments may have a circular, hexagonal, or square shape. On top of that, segments may come with random displacement, random mis-alignment (zenith-angle dependent), and random errors in their focal length.

For dual-mirror telescopes, the optical shapes are described by even-order polynomials. Which can come in a monolithic shape or segmented, with more options for segment shapes than available with segmented single reflectors. Without segmentation, both the primary and secondary mirror are assumed to have a circular shape, with a possible hole (thought as covered in black) in the middle. An example of relevant parameters for dual-mirror telescopes is shown in Figure 11.3, courtesy of G. Maier.

Deformations of the mirrors (as a whole or individually) and the telescope structure by gravity and wind load is not included, other than through the zenith-angle-dependent effective mirror alignment error. Resulting systematic pointing errors from such deformations are assumed to be corrected in the analysis by pointing corrections for the actual instrument, while no corrections need to be applied to simulated data. Random pointing errors come in *known* (measured and can be corrected) and *unknown* (cannot be corrected) varieties.

11.7 Mirror positions and sizes

While early versions of `sim_telarray` (for the HEGRA system) just filled in all possible mirror positions in a given radial range, all current simulations with `sim_hessarray` take positions of each mirror segment individually from a table. This file is only redundant for dual-mirror telescopes where we have a different way of handling segments; see below. The mirror definition file like `hess_mirrors.dat` as well as the corresponding figure 11.4 can be generated with the program `plot_mirrors` (or the plot alone generated with `draw_mirrors`). To account for the positions of the sky CCD camera etc., two mirrors in this specific example were commented out by hand before drawing. The old-style simple mirror configuration is no longer available, and specifying a mirror configuration file is always necessary.

While most mirror configurations consist only of one type of mirrors the configuration file can handle mirrors of circular, square, or hexagonal type, with a possible grading of focal lengths etc. Even with multiple shapes and sizes on the same telescope. Which file to use is controlled through the MIRROR_LIST parameter. The maximum number of mirrors is defined via MAX_MIRRORS (see [6]).

Each non-empty and not commented-out line the mirror list contains four to six values, with missing defaults set to column-specific defaults. Some configuration files include mirror IDs after a comment separator; these IDs are not used by `sim_telarray` itself.

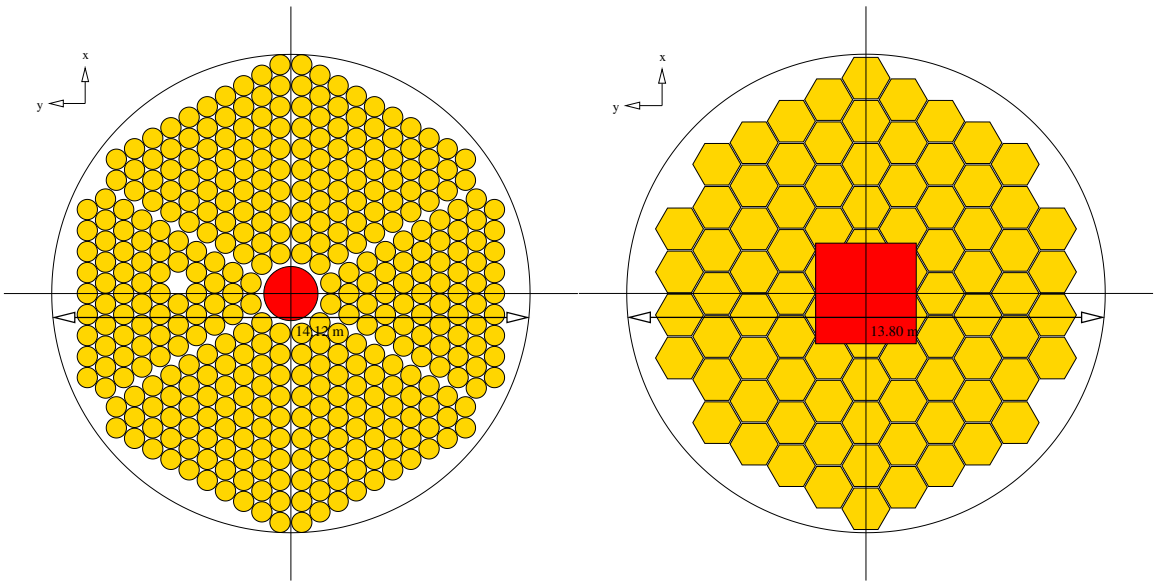

Figure 11.4: Layout of the mirror segments on H.E.S.S. CT1 to CT4 (left) and on a CTA MST (right). This is a *bird's eye view* of a telescope pointing upwards and not a view of a telescope pointed to the horizon. North (x) is up, West (y) is left. When turning the telescope to point to the horizon, the x axis would be downward, y to the right when looking from the camera side onto the mirror. The red circle and square represent the assumed camera bodies, as far as shadowing is concerned.

1. Mirror center x position w.r.t. optical axis; units: cm. For a telescope pointing to the horizon, x is downward (for coordinate system definitions see Section 10.3).
2. Mirror center y position w.r.t. optical axis; units: cm. For a telescope pointing to the horizon, and looking from the camera onto the mirror, y is to the right.
3. Flat-to-flat (or circle) diameter; units: cm.
4. Mirror focal length; units: cm. A value of zero here is interpreted as either a) using the common `MIRROR_FOCAL_LENGTH` value, if that is non-zero, or b) automatically adapting the mirror focal length to the distance of the segment center from the camera center, before applying random offsets and randomizing the focal length. That mentioned distance is determined by the dish shape, depending on the dish shape type (`PARABOLIC_DISH`), the nominal focal length (`FOCAL_Length`) and the dish curvature (`DISH_SHAPE_Length`, equal to the radius of curvature for a spherical dish shape and equal to the paraboloid 'focal length' for a parabolic dish). Note: For positive values, no `RANDOM_FOCAL_LENGTH` based errors are added in `sim_telarray`. For negative values, the sign gets inverted and random errors will be added.
5. Shape type. 0=circular, 1=hex. with flat side parallel to y, 2=square, 3=other hex. (default: 0).
6. z position (height above dish backplane); units: cm. Typically omitted (or zero) to adapt to the dish shape settings.

7. Optional comment separator (“#”, “%”, or “#%”), protecting against possible future extensions with more parameters used by `sim_telarray`.
8. Optional assignment of ID or other values not used by `sim_telarray` itself.

In configurations with secondary mirror optics, both the primary and the secondary mirror are treated, at least in a first step, as monolithic, of circular shape, with a circular non-transparent, non-reflecting (i.e. black) hole. The mirror configuration file defines just the outer diameters of them, not the diameter of the hole. This file can actually be bypassed with `MIRROR_LIST=none`. In that case all relevant parameters are set entirely in the main configuration file, including the `PRIMARY_DIAMETER` and `SECONDARY_DIAMETER` values – which otherwise would be taken from the diameter of the first and second mirror listed (enforcing zero x and y values). Segmented primary and/or secondary mirrors can be specified through optional data files (`PRIMARY_SEGMENTATION`, `SECONDARY_SEGMENTATION`).

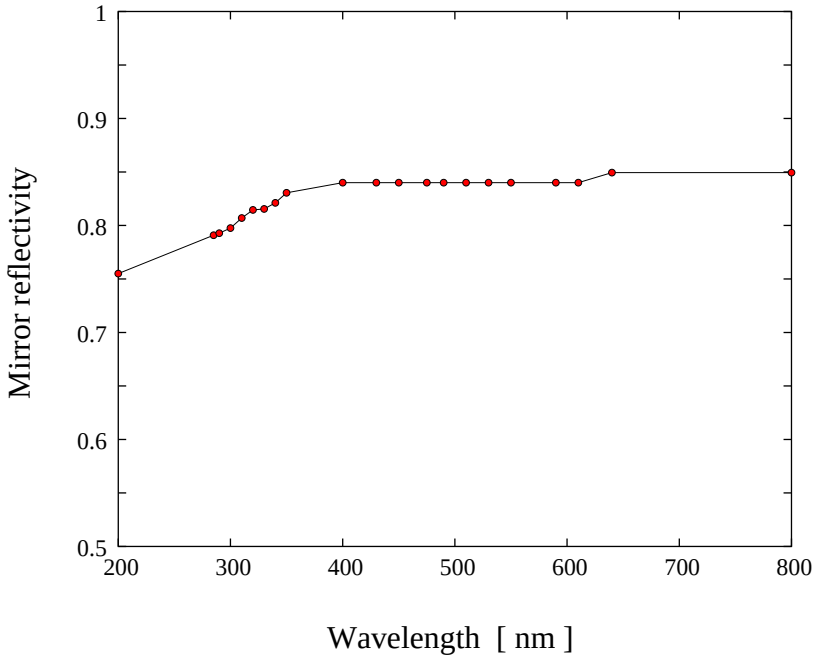

Figure 11.5: Reflectivity of H.E.S.S. mirrors.

11.8 Mirror reflectivity curve

The mirror reflectivity, taken from `hess_reflect.dat` by default, accounts for the wavelength dependence of typical H.E.S.S. as taken from the database of mirror measurements, extended with literature data for aluminised mirrors. Steps in that table can be arbitrary. Linear interpolation is used. Which file to use is controlled through the `MIRROR_REFLECTIVITY` parameter. The default reflectivity curve is illustrated in figure 11.5. With secondary mirror optics, the secondary by default is assumed to have the

same reflectivity as the primary but that can be changed by specifying the corresponding data file through the MIRROR_SECONDARY_REFLECTIVITY parameter. Through the MIRROR_DEGRADED_REFLECTION, MIRROR2_DEGRADED_REFLECTION, and/or CAMERA_DEGRADED_EFFICIENCY parameters one can adapt to poorer actual reflectivities (actually optical efficiency, including dust in/on a camera), even on a telescope-by-telescope basis (through '#if TELESCOPE==*n*' in the config file). These parameters also change NSB pixel rates and currents in a consistent way. On top of that an additional position-dependent degradation can be configured for the (primary) mirror as PRIMARY_DEGRADED_MAP – in case of dual-mirror optics separately also for the secondary as SECONDARY_DEGRADED_MAP. On top of that a position-dependent degrading (for example due to dust not settled uniformly or imhomogeneous camera window) can be set at the camera entry as CAMERA_DEGRADED_MAP. Note that these maps may slow down the ray-tracing, in particular if non-equidistant tables are used, and its impact on NSB pixel rates and currents is not taken care of; it needs to be calculated by actual ray-tracing and then configured with the NSB pixel rate settings.

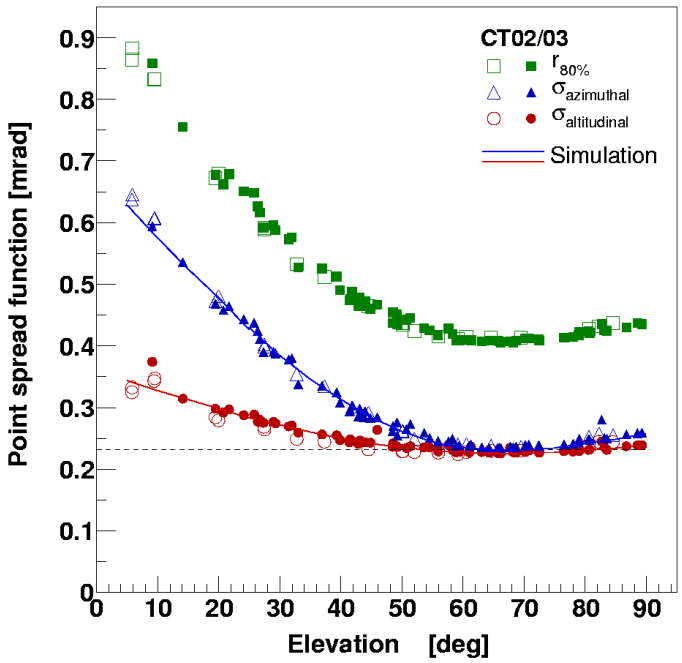

Figure 11.6: Point spread function on-axis as a function of zenith angle. Plot with measurements from R. Cornils. The dashed line corresponds to the fixed p.s.f. produced by older versions of this program, the solid lines to the separate components produced by versions since July 2003.

If the specified reflectivity table is recognized as a 2-D table, the angle of incidence (in degrees w.r.t. normal incidence) is used as a second dimension in the interpolation, in addition to the wavelength dependence always used. Note that for 1-D tables the reflectivity is completely absorbed into the wavelength-dependent efficiency applied, for computing efficiency reasons, before the start of the ray-tracing instead of during the ray-tracing. For

2-D tables, the upper envelope in its projection onto the wavelength axis is applied in this initial step while the ratio of angle-dependent to upper-envelope reflectivity is applied during the ray-tracing procedure (separate for primary and secondary mirror in case of dual-mirror telescopes).

11.9 Point spread function

Through the ray-tracing method, the actual point spread function of the telescope can be reproduced simultaneously on the front of the lid (as used for star ligh measurements with the lid CCD camera) and on the front of the PMT funnels. The dependence on the angle with respect to the telescope axis is automatically reproduced. A defocusing effect on large zenith angle showers on the pixels is also automatically included (note that pre-defined 2.8 cm difference between lid front and funnel front is optimized for low zenith angle H.E.S.S.-1 observations). In versions since July 2003, the zenith angle dependence of the point spread function for stars on the lid (see Figure 11.6 is included as well. In theses newer versions, this is achieved through two sets of parameters in `MIRROR_ALIGN_RANDOM_HORIzontal` and `MIRROR_ALIGN_RANDOM_VERTical` separately for y and x components in the dish coordinate frame, while in earlier version there was only one corresponding constant parameter in `MIRROR_ALIGN_RANDOM_ANGLE`.

Other parameters with an impact on the point spread function are the spread in focal lengths of the mirror tiles, `RANDOM_FOCAL_Length`, its random displacement `MIRROR_ALIGN_RANDOM_DISTance` from the specified dish shape, and in particular the surface quality described by `MIRROR_REFLECTION_RANDOM_Angle`.

11.10 Camera masts and other shadowing elements

Although too inefficient for normal use, the ray-tracing method in `sim_telarray` allows for explicit simulation of the shadowing effects of camera masts and other such elements, the camera body and the camera lid. Figure 11.7 shows a resulting shadowing pattern for light 2° off-axis. A `-DRAYTRACING_INTERSECT_RODS` compiler flag is required to active this. In that case, geometry definitions of the shadowing elements (except the camera body) are taken from a file, by default from `hess_masts.dat`. This is controlled through the `MASTS_File` parameter. With normal compilation no such file is used and only shadowing by the camera is handled explicitly, while shadowing by other elements is treated by the `TELESCOPE_TRANSmission` parameter.

11.11 Camera and trigger definition

A camera and trigger configuration (or definition) file sets the pixel type properties, the camera rotation, the pixel positions prior to this rotation, as well as the trigger topology. A

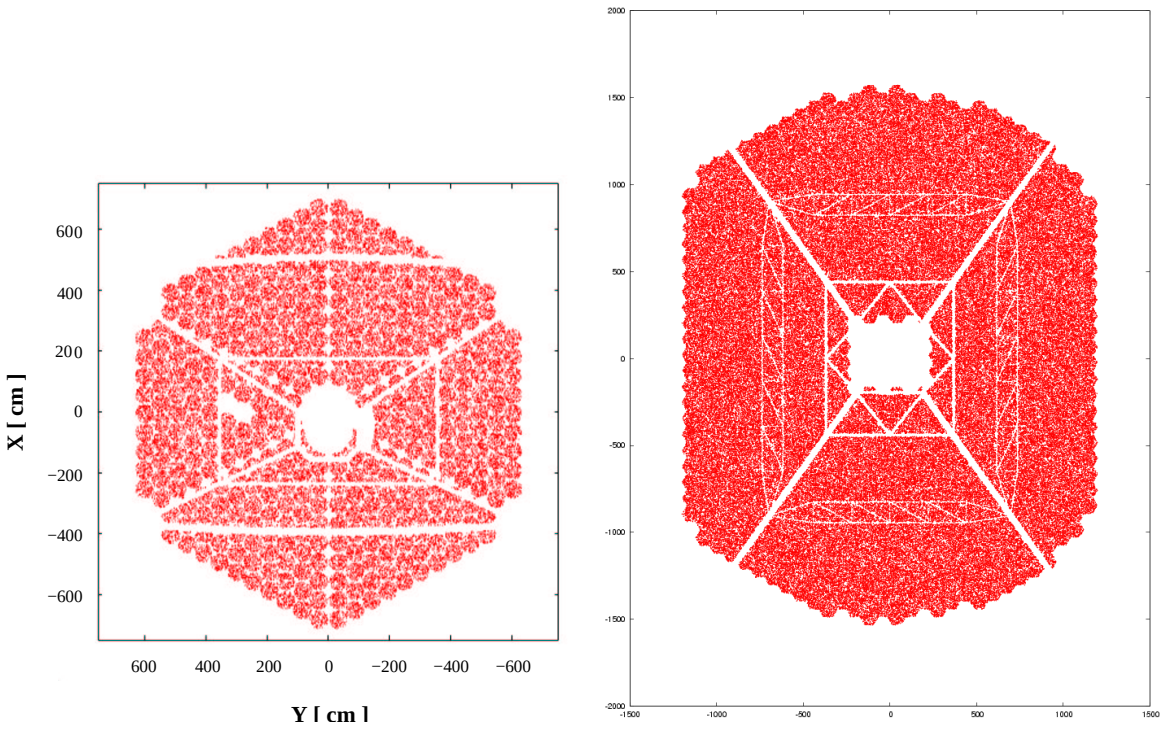

Figure 11.7: Shadowing by masts, camera lid etc. in a small H.E.S.S. telescope (left) and in the big H.E.S.S. telescope CT5 (right). Bird's eye view from above of telescopes pointing towards zenith.

camera entrance window or the camera body are not part of it. The file actually used for that can be changed through the CAMERA_CONFIG_FILE parameter.

The camera and trigger definition file is made up by four types of definitions (counting all variants of trigger definitions as one type):

Rotate Defines a rotation angle (in degrees) of the whole set of pixels inside the camera, affecting not just pixel positions but also the edges.

PixType Defines a pixel type, including the geometrical shape (circular, square, hexagonal) of the pixel border and of the visible cathode, the size, the depth of light collecting elements (funnels) and their reflectivity (as a number) or angular acceptance (as a file name).
Example:

```
PixType 1    0   0 2.100    1 4.150 5.100    "hess_funnels.dat"
```

Values following the 'PixType' keyword denote the pixel type ID, the PMT type (0 for now), the geometry type of the open photocathode area (not used with funnel efficiency tables), the size of the open photocathode (flat-to-flat for hexagonal and square geometry types, not used with funnel efficiency tables), the geometry type of the funnel outer edge, the size of the funnel (flat-to-flat for hexagonal and square

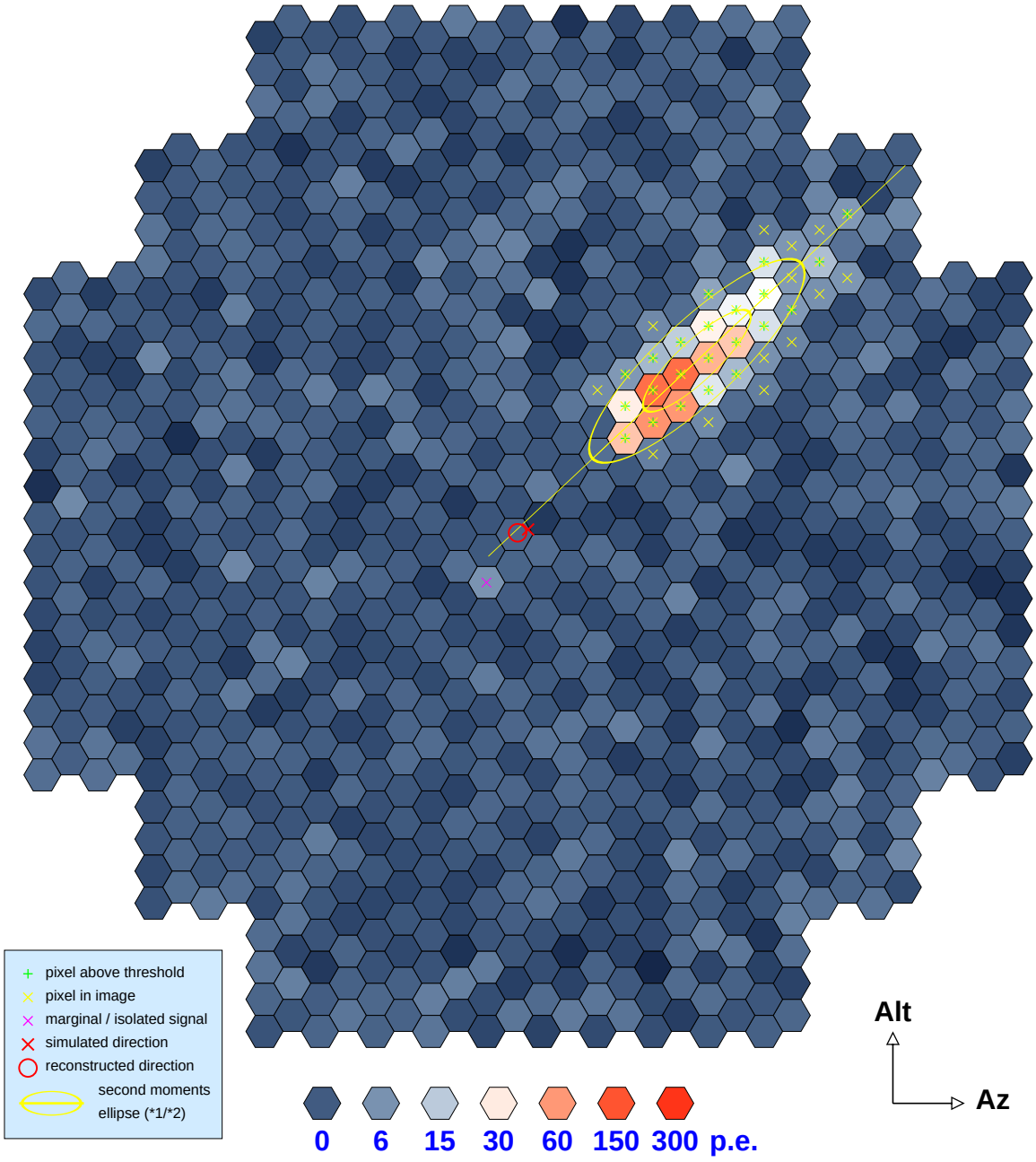

Figure 11.8: The H.E.S.S. camera in the simulation

geometry types) and the depth of the funnel (not used with funnel efficiency tables). If the following parameter is enclosed in double quotes it is assumed to be the name of file containing funnel efficiency tabulated as a function of the angle to the axis, averaged around the axis. Otherwise two more values follow: the transparency of a pixel and the reflectivity of its walls for typical angles of incidence.

Pixel Defines mainly of which pixel type each pixel is and its position in the camera. Also included is the assignment to electronics channels.
The complete list of the 15 required and optional parameters to the Pixel line include

PixelID A unique integer between zero and npix-1 (see below).

PixelType Corresponds to the `PixType` line above; must be 1.

Xpix Projected pixel *x* coordinate (in cm) before camera rotation.

Ypix Projected pixel *y* coordinate (in cm) before camera rotation.

Module (Optional; default = 0). Module number to which the pixel belongs, used e.g. for pixel alignment with `PIXELS_PARALLEL`=2.

Board (Optional; default = 0). Board number.

Channel (Optional; default = 0). Input channel on electronic board; may be used in future version for electronic cross-talk.

ModuleID (Optional; default = 0). If a module is swapped into a different place in the camera, its module number (defined by geometry) would change, along with all pixel coordinates, but the module ID would remain. Not currently used. Should be hexadecimal, starting with 0x.

OnFlag (Optional; default = 1). A value of zero indicates a turned-off or missing pixel.

QeRel (Optional; default = 1.0). Quantum efficiency (or photon detection efficiency) of this pixel over the nominal value, before any random fluctuations get applied.

GainRel (Optional; default = 1.0). Signal gain or amplification in pixel relative to average before any random fluctuations get applied.

Dz (Optional; default = 0.0). Shift of pixel height w.r.t. nominal focal surface shape (or module) in centimeters, positive towards the infalling light. No shift is needed where pixels are aligned module-wise.

PixRot (Optional; default = 0.0). Pixel rotation angle (deg.).

Nx (Optional; default = 0.0). X component of non-standard pointing direction of pixel axis is indicated by vector (Nx, Ny, 1.0).

Ny (Optional; default = 0.0). Y component of (Nx, Ny, 1.0).

Where an optional value gets used, all preceding values must be present.
Example:

```
Pixel 959 1 -65.363 -34.143 59  1  7 0x76 1
```

The values following the 'Pixel' keyword are the pixel ID, the pixel type, and the x and y coordinates. Further values indicate the pixel module ('drawer'), the board number within the module, the channel number on the board, and a group ID in hexadecimal notation. The group ID (e.g. drawer ID) is not currently used in the simulation. It may be followed by a flag indicating if the pixel is enabled (1) or disabled (0). Disabled means that high voltage is off and the pixel not considered in the trigger. If that flag is missing, the pixel is enabled by default. The drawer, board and channel numbers are only used to indicate the configuration used in the output but are not used internally (e.g. for crosstalks etc.) The actual number of pixels defined in the camera configuration file is expected to match the CAMERA_PIXELs (npix) value in the main configuration and must not exceed the MAX_PIXELS compile-time limit.

Trigger This keyword used to have the meaning now used with 'MajorityTrigger'. Since different trigger types are now supported with sim_telarray, the meaning of the 'Trigger' keyword can be assigned. The default meaning is assigned with the DEFAULT_TRIGger configuration parameter (telescope-specific, typically in the main configuration file or one of the files included by it). It can be re-assigned in the camera configuration file discussed here by the DefaultTrigger keyword.

MajorityTrigger Special case for the 'Trigger' keyword which applies only to the majority trigger logic with comparators or discriminators at the signal line from each photo-sensor. By default, the 'Trigger' keyword is set to mean the 'MajorityTrigger' case. Determines how many of which pixels have to exceed a threshold before a camera triggers. There can be any number of such trigger statements. A '*' for the number of pixels required for a trigger means that it will be replaced by the per-telescope default configured with the TRIGGER_PIXELs parameter. Any non-zero number in the 'Trigger' definition overrides the default value.
Example:

```
Trigger * of  156 157 183 184 185 212 213
```

The values following the word 'of' are pixel ID numbers. They can be preceded by a '+' to indicate that this pixel must have fired, in addition to the requested multiplicity (e.g. central pixel in the SmartPixel design).
For the majority trigger, a pixel fires when the analog channel signal exceeds DISCRIMINATOR_THRESHOLD for at least a time DISCRIMINATOR_TIME_OVER_THRESHOLD (with pixel-to-pixel variation DISCRIMINATOR_VAR_TIME_OVER_THRESHOLD) and an integrated charge-over-threshold of at least DISCRIMINATOR_SIGSUM_OVER_THRESHOLD (with pixel-to-pixel variation DISCRIMINATOR_VAR_SIGSUM_OVER_THRESHOLD). The resulting output signal of amplitude DISCRIMINATOR_OUTPUT_AMPlitude (with pixel-to-pixel variation DISCRIMINATOR_OUTPUT_VAR_PERCENT, in percent) has

a (linear) rise time of DISCRIMINATOR_RISE_TIME and a fall time of DISCRIMINATOR_FALL_TIME, with either a fixed length DISCRIMINATOR_GATE_LENGTH (discriminator style) or until the analog channel signal falls to a value DISCRIMINATOR_HYSTERESIS below the threshold value but given minimum length (comparator style) but minimum length as before.
For a resulting camera trigger, the sum of the discriminator/comparator output signals of the listed pixels has to exceed

$$(\texttt{TRIGGER\_PIXELS} + \texttt{MULTIPLICITY\_OFFSET}) * \texttt{DISCRIMINATOR\_OUTPUT\_AMPlitude}$$

for at least a time TELTRIG_MIN_TIME and at least a charge-over-threshold integral of TELTRIG_MIN_SIGSUM. All majority (and analog sum) trigger calculations are done in a factor four finer time intervals than the FADC intervals (depending on preprocessor definitions during compilation).

AnalogSumTrigger Another special case of the 'Trigger' keyword, with identical syntax but trigger conditions determined by other configuration parameters. Basically, it works by adding up the same signals otherwise used as discriminator input, optionally after clipping the individual signals, and applying a threshold after summation. There are also optional image shaping options available after signal clipping. See the ASUM_SHAPING_FILE, ASUM_CLIPping, ASUM_THRESHold and ASUM_OFFSET configuration parameters. Basically, the shaped signal (can thus differ from the relevant pulse shape of a competing majority trigger, may be shifted by the given time offset) and clipped at a maximum amplitude; then the resulting signals of the listed pixels are added up and compared to the threshold. The only parameter shared between analog sum and majority trigger types is the average amplitude of the signal per photoelectron, before any shaping and clipping, from DISCRIMINATOR_AMPLITUDE.

DigitalSumTrigger The third special case of the 'Trigger' keyword. In this case the discriminator input signal is not used at all but only the digitized signal. The operations applied to that signal are basically equivalent to that applied on the analog signal for the analog sum trigger, plus some extra features not possible in analog. Corresponding configuration parameters include DSUM_PEDSUB (subtract pedestal?), DSUM_OFFSET (time offset for shaping to apply), DSUM_SHAPING_FILE (shaping kernel from file), DSUM_SHAPING_RENORMalize (auto-normalize kernel), DSUM_CLIPping (after shaping), DSUM_PRE_CLIPping (before shaping), DSUM_IGNORE_BELOW, DSUM_ZERO_CLIP, DSUM_PRESCALE, DSUM_PRESUM_MAX, DSUM_PRESUM_SHIFT, and DSUM_THRESHold. The relevant amplitude parameter here is FADC_AMPLITUDE since the trigger operates on the normal digitized signal. The digital sum trigger does not use the analog channel signals and ignores all DISCRIMINATOR_... settings.

DefaultTrigger This sets the meaning of the 'Trigger' keyword to either 'Majority', 'AnalogSum' or 'DigitalSum'. Example:

```
DefaultTrigger AnalogSum
Trigger * of 1 2 3 4 5 6 7
```

is equivalent to

```
AnalogSumTrigger * of 1 2 3 4 5 6 7
```

The majority trigger definition can include a (analog channel) pre-summation of individual pixels into trigger-level super pixels, like the digital sum trigger can include a (digital channel) pre-summation. Example:

```
MajorityTrigger * of 1[2,3,4] 5[6,7,8] 9[10,11,12]
DigitalSumTrigger * of 224[226] 64[65,66] 223[225,227]
```

Majority super pixels fired will be registered only under the pixel ID of the 'master' pixel in front of the (usually comma-separated) list of 'slave' pixels inside the square brackets although all pixels contribute in a symmetric way towards the actual trigger. Super pixels can also be prefixed by a '+' to indicate that the particular super pixel is required to have fired for a camera trigger coming from this trigger group.

The total number of trigger group definitions should not exceed the `H_MAX_SECTORS` definition in the compilation of the hessio library to avoid loss of trigger information in the resulting data file. If more than one type of trigger is used for the same camera, the `TRIGGER_DELAY_compensation` parameter can be used to adjust systematic differences in delays between the trigger types (three values in the order: majority, analog sum, digital sum), as there is only one readout window defined by the earliest trigger (and starting `FADC_SUM_OFFSET` intervals before that trigger time).

As already mentioned, the maximum number of pixels is defined via `MAX_PIXELS` (see [6]). A picture of the H.E.S.S.-I camera in a simulation is shown in figure 11.8. The default file is `hess_camera.dat`. An alternative with an angle-independent funnel efficiency of 73% is found in `hess_camera_73.dat` An alternate definition for a Smart-Pixel-like camera with a comparable number of pixels is found in `smartpixel_camera.dat`. A collection of camera definitions used for CTA simulations is illustrated in figure 11.9.

11.12 Camera window as a filter

The camera can have a window to hold off dust and humidity from the sensory and electronics. This window also acts as a filter, for example to hold off infrared photons from SiPM sensors otherwise sensitive in the IR region. If any such filter is used its transmission table is specified by the `CAMERA_FILTER` parameter, by default a 1-D table (transmission as a function of wavelength) and optionally a 2-D table (function of wavelength and angle of incidence, w.r.t. the pixel orientation). On top of that, a wavelength- and angle-independent transmission factor can be specified by the `CAMERA_TRANSmission` parameter.

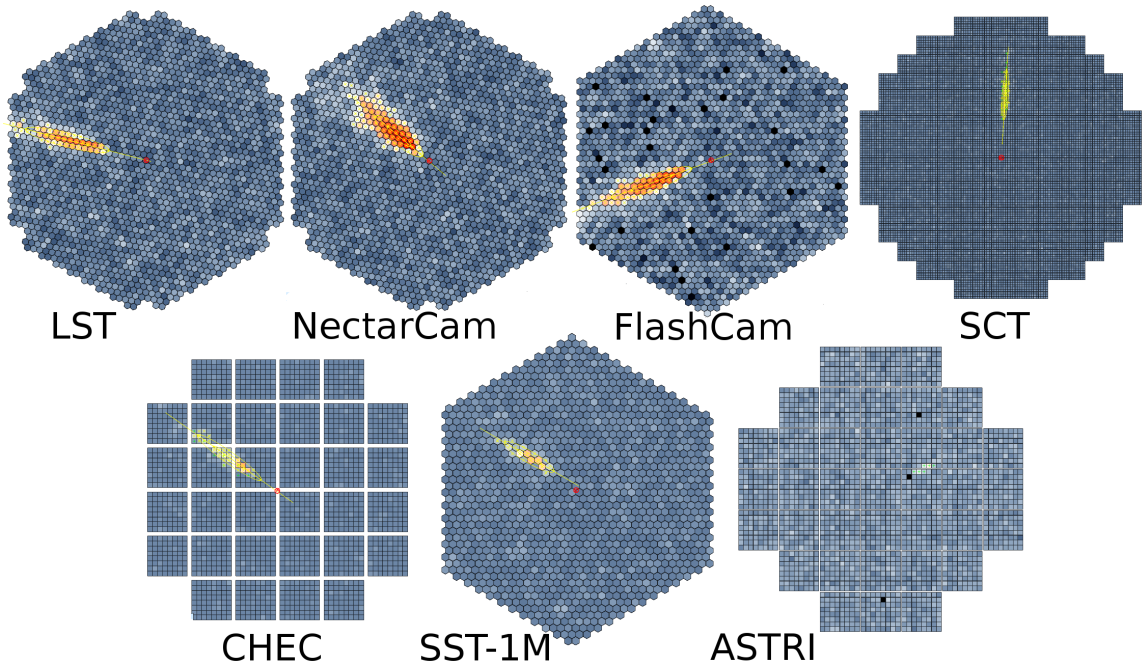

Figure 11.9: Different types of cameras in CTA telescopes (not to the same scale).

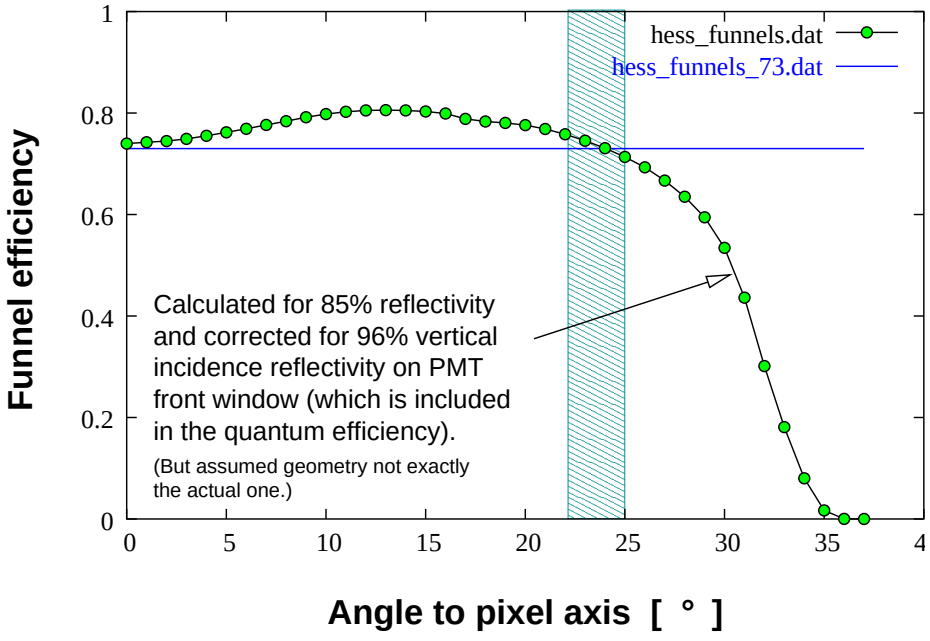

Figure 11.10: H.E.S.S. funnel efficiencies

The impact of the camera window on the photon path due to refraction still needs to be implemented. The optical properties of the camera window will be defined by the parameters CAMERA_WINDOW_HEIGHT, CAMERA_WINDOW_THICKness, CAMERA_WINDOW_RADIUS, and CAMERA_WINDOW_REFIDX, with the geometrical parameters optionally modified by the CAMERA_SCALE_FACTOR parameter. Since Fresnel reflection is supposed to be accounted for with CAMERA_TRANSmission and/or CAMERA_FILTER, no rays reflected on the window surfaces are considered in the optical ray-tracing.

11.13 Funnel angular response

Light collection in the 'Winston cone' funnels is implemented with two different algorithms. The older, used when the pixel type definition (`PixType` line in the camera configuration file) includes a number for the funnel reflectivity, assumes an efficiency of 100% if the photon hits the cathode directly. Otherwise it assumes that the photon is reflected exactly once with the given reflectivity and then hits the cathode. The newer algorithm takes a data table with efficiencies as a function of the angle of incidence but does not distinguish between direct cathode hits and single/multiple reflections. That version is used when the pixel type definition includes a file name (enclosed in double quotes) for the funnel reflectivity/efficiency. Neither the depth of the light cone nor the diameter of the open cathode nor the funnel wall reflectivity are used in this case.

Figure 11.10 shows the efficiencies of two available data files, one with efficiencies calculated by ray-tracing with 85% reflectivity for the funnel surfaces, taking the reflection on the glass surface on top of the flat cathode into account, and afterwards correcting for vertical incidence reflection on the glass surface (which already enters into quantum efficiency measurements).

Note: The funnel response used in the default camera configuration file was changed in June 2003 to use `hess_funnels_r78.dat` instead of `hess_funnels.dat`. The newer file was calculated with the actual funnel geometry and 78% reflectivity of funnel walls.

The `PixType` also allows for an additional wavelength dependence of the funnel efficiency (second file name in quotes). This is not interchangeable with the window filter transmission (see section 11.12). The extra funnel efficiency table is interpreted such that efficiency factors are not double-counted on top of that in the angle-dependence table (normalized to yield no extra correction for 400 nm wavelength at the expected average incidence angle). The window filter transmission, on the other hand, is always applied as an extra factor.

11.14 Quantum efficiency curve

Quantum efficiency curves (Q.E. as a function of wavelength) from different sources have been found to differ by a substantial amount and, thus, are one of the main uncertainty fac-

tors in absolute sensitivity predictions (after trigger electronics response). The Q.E. curve to use is also related with the single photo-electron (s.p.e.) response (see next section), because a fraction of the photo-electrons either doesn't hit the first dynode at all or is just reflected inelastically off the first dynode (without any amplification). These effects make up the collection efficiency.

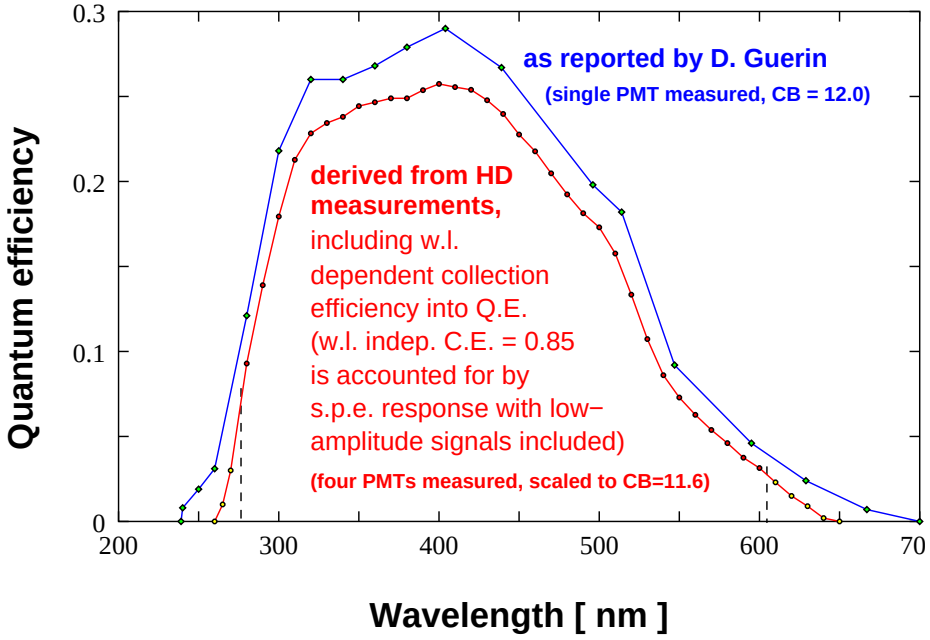

Figure 11.11: H.E.S.S. photomultiplier tube quantum efficiency curve

In the `sim_telarray` simulation the Q.E. curve is meant to account only for the quantum efficiency itself (i.e. as obtained by measuring cathode currents), while the collection efficiency is dealt separately (see next section). Figure 11.11 demonstrates that the Q.E. measurements at Heidelberg [11] are consistent with the latest Q.E. curve from Photonis, if the different absolute sensitivities (as expressed by the *Corning Blue (CB)* value) are taken into account. The default Q.E. curve for `sim_telarray` simulations is taken from `hess_qe2.dat` and corresponds to a CB value of 11.6, which is the average from the PMT database.

The file actually used can be changed through the `QUANTUM_EFFiciency` parameter.

11.15 Single photo-electron response

The amplitude measurable at the anode varies from photo-electron to photo-electron. This statistical fluctuation is accounted for through the single photo-electron (s.p.e.) response table. For the actual amplitudes for each photo-electron random numbers are use which, on average, represent the specified s.p.e. table. The table used in `sim_telarray` simulations, as contained in `hess_spe2.dat`, is based on an electron multiplication Monte

Carlo program with results adapted to measurements in Heidelberg. The problem with taking the measurements directly is that some noise is unavoidable and that high-amplitude signals are, to a large extend, due to multiple photo-electron events because otherwise the counting rate would be too low. Therefore, the PMT simulation was adapted to the rates in measurements and then re-ran for single photo-electrons (see figure 11.12). The simulation also reproduces low-amplitude events by inelastic scattering of photo-electrons on the first dynode and, thus, accounts for most of the effects of the so-called collection efficiency. The remaining collection efficiency factor (`PM_COLLECTION_EFFiciency` parameter) to be used in conjunction with this s.p.e. response table is therefore set to 1.

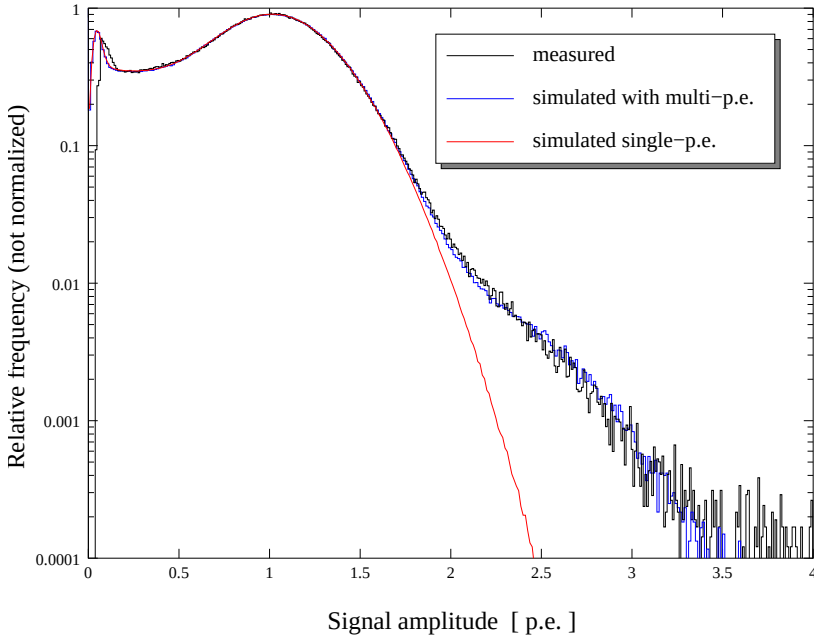

Figure 11.12: Single photo-electron response (measured and simulated).

The s.p.e. response table has to be scaled such that its mean value divided by the remaining collection efficiency parameter is 1 and, therefore, the average signal of 100 photons converted to photo-electrons in the cathode is 100 *p.e.* In contrast, the experimentalist determines the s.p.e. response not by its mean value but by the peak. The peak in `hess_spe2.dat` is at 1.17, translating the experimental gain value of $2 \cdot 10^5$ (number of electrons collected at the readout for the most probable case) to a MC gain of $1.72 \cdot 10^5$ (defined as the average number of electrons collected).

The s.p.e. response tables also accounts for afterpulses. Since afterpulses from the Cherenkov light itself arrives much too late to be registered with the shower, afterpulses are only taken into account for night-sky background light, where it doesn't matter if the original photo-electron was a few hundred nanoseconds earlier.

The file actually used can be changed through the `PM_PHOTOELECTRON_SPECTRUM` parameter.

11.16 Single photo-electron pulse shapes

The pulse shape of the signal at the input of the comparator or discriminator logic is an important ingredient in simulations but difficult to measure. The length of the pulse shape of the PMT output itself can be taken as a lower limit to the pulse shape of interest here. Due to preamplifier band width, the pulse shape to be used here might be somewhat longer. Measurements in Paris and their impact on the effective area of the H.E.S.S. system are discussed in an internal note[12]. The pulse shapes there are significantly smaller than reported by Photonis and as implemented in the old default data file `hess_disc_shape_slow.dat` (originally named `hess_disc_shape.dat`). The discrepancy still needs to be resolved. As a rule of thumb: shorter pulses correspond to higher amplitudes required to trigger the camera, in a similar but not exactly the same way for night-sky background and for shower photons. The actual file to be used is controlled through the `DISCRIMINATOR_PULSE_SHAPE` parameter. The pulse shape from the mentioned H.E.S.S. note is available as `hess_disc_shape-01-10.dat` Another alternative available is `hess_disc_shape_parismc.dat` (originally named `pshape.dat`), representing the pulse shape used for intercomparison of different simulation programs. The available shapes are compared in Figure 11.13, with arbitrary offset of the time zero point and arbitrary scaling of the amplitudes, since `sim_telarray` locates the peak position and shifts the time offset and scales the amplitude accordingly.

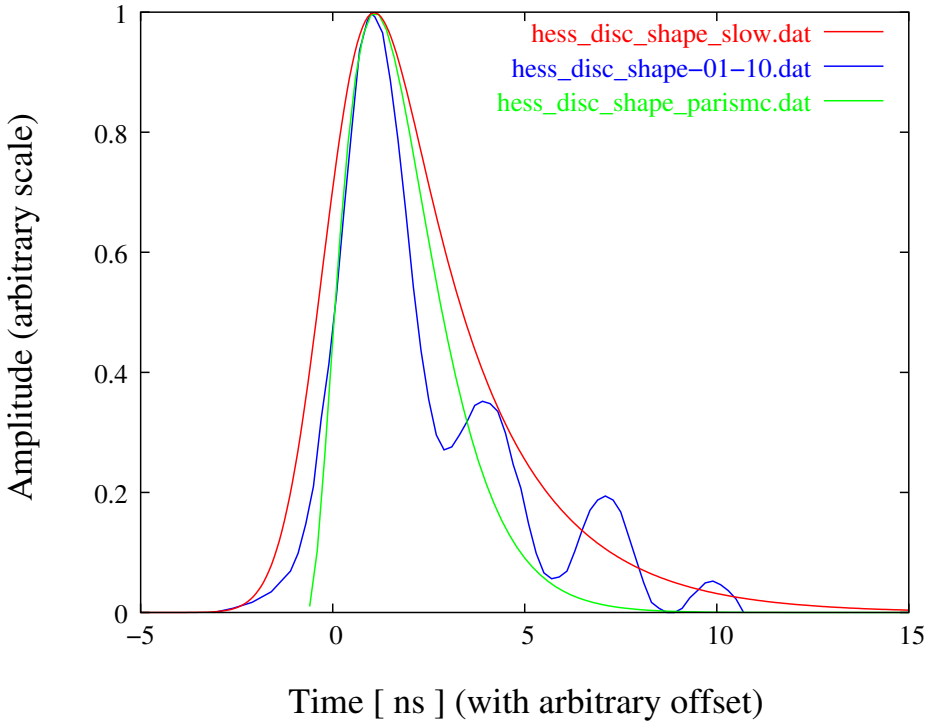

Figure 11.13: Several configurations for the pulse shape at the comparator.

A similar file is used for the pulse shapes seen in the digitized signals. This seems to be measured to a better extend and is normally taken as `hess_fadc_shape.dat`. The

actual file to be used is controlled through the FADC_PULSE_SHAPE parameter, in two format variants (implicit or explicit time steps) and for either a single or two gains. Note that any delays between the pulse positions are preserved in the resulting data.

11.17 Random number generators and seeds

In contrast to CORSIKA, sim_telarray is using a single pseudo-random number sequence, for multiple purposes. These purposes include the configuration stage with many randomly chosen parameters, like random mirror panel alignments, random discriminator output amplitudes. And they also include the event-wise (and photon-by-photon) simulation stage. For some workflows it can be useful to have reproducible telescope configurations but independent event simulation in multiple runs. To cope with such and other workflows, the random number sequence can be re-initialized after the configuration stage. By default, no such re-initialisation with a new seed value is enforced. If a second seed value is given with the RANDOM_SEED parameter (separated by a comma), the first one will take effect at the start and remain in effect only for the configuration while the second seed is used to control all random numbers generated after the configuraton stage. Each parameter can be one of

- the special word auto,
- an integer between 1 and 2147483647,
- a *method : filename* combination.

The auto choice (a single one being the default) results in a non-reproducible seed (combining high-precision time, process ID, and, in particular, random content derived from /dev/urandom).

The integer choice results in a fully reproducible set-up or entirely reproducible simulation (given identical inputs).

The third choice allows to select a number from a text file which contains multiple pre-defined seed values, one value per non-empty, non-comment line (empty lines and '#' comment lines ignored). The methods available to determine which line to use for a seed, include:

file-by-run Use the CORSIKA_RUN environment variable. Keep in mind that no input data has been read at that point and external information on the run number is needed.

file-by-time Use a combination of seconds and microseconds of current time.

file-by-random Use an auto-generated random number (involves /dev/urandom).

The method-specific number, modulo the number of available seeds, determines which of the pre-generated seeds will be picked and returned as the random seed. Given a file with 50 pre-generated seeds, the *file-by-run* method would chose the first seed for CORSIKA run number 1 (or 51, or 101, ...), the second seed for run number 2 (or 52, etc.), and so

on. The other methods are less predictable but over a large number of simulation runs, all seeds should be used about as often as all others. The gen_seedlist tool can be used to generate such seed lists in a convenient way (keep in mind that seeds should be in the range 1 to 2147483647).

Examples of random seed settings:

```
RANDOM_SEED = auto
RANDOM_SEED 12345,98765
random_seed = 12345,auto
Random_Seed file-by-run:mylist.dat,auto
```

In addition to the seeds, the random number generator may be selectable, depending on compilation options. If sim_telarray was compiled with -DWITH_GSL_RNG and linked against the GNU Scientific Library, any random number generator available from the GSL can be selected with the RANDOM_GENERATOR parameter. Otherwise only the traditional (and default) choice RanLux (in 'luxury level' 3) is available. RanLux, like most other generators, only produces 32-bit floating-point random numbers and is not the fastest generator but is considered a safe choice. Use other generators at your own risk.

11.18 Random pixel properties

Even though there is some infrastructure in place for different types of pixels in a camera, that was never used and camera configurations are currently limited to a single PixelType (see CAMERA_CONFIG_FILE). Nevertheless, pixels are not all equal; there can be a random scatter of a number of properties:

While all pixels (of the same PixelType, in case the single type restricted gets lifted) are assumed to have the same shape of the quantum efficiency (QE) curve for PMTs or photon detection efficiency (PDE) curve for SiPMs, as specified by the QUANTUM_EFFiciency file name, the overall scale can vary randomly (by fraction QE_VARIATION) when the camera gets defined initially. In a similar way the gain (PMT times electronics) (PM_AVERAGE_GAIN) can vary by a fraction GAIN_VARIATION and the high voltage with which a PMT is operated will scatter by a fraction PM_VOLTAGE_VARIATION.

However, these variations are not to be used as independent. There are two possible camera adjustments: flatfielded or gain adjusted. In a camera with flatfielding the gain is adjusted such that equal signal results from equal illumination, i.e. higher gain is applied where the sensor has lower quantum efficiency. Without flatfielding it is assumed that pixels are adjusted to equal gains, i.e. equal single-p.e. amplitudes (within errors). Any gain variation, either random or for flatfielding is assumed to come from voltage adjustments (and not in the electronics), gain g and voltage U assumed to be related by $g \propto U^p$ where p is set as PM_GAIN_INDEX. Changing the high voltage of a PMT also changes its transit time PM_TRANSIT_TIME as proportional to one over the square root of the voltage (basically assuming a simple resistor chain for the PMT voltage divider). In case of a PMT with the first dynode stabilized to a fixed voltage, the fixed part of the voltage (third and, optionally, fourth value in PM_TRANSIT_TIME) and the fixed part of the transit time

(second value in PM_TRANSIT_TIME) are excluded here. On top of that there may be an additional random difference in the transit time, as defined by TRANSIT_TIME_ERROR. If the latter is set to -1, the readout interval is assumed to be adjusted, pixel by pixel, such that resulting transit time differences are within one readout time slice. For correcting transit times by electronic means with time steps independent of the sampling frequency the TRANSIT_TIME_COMPENSATE_STEP and TRANSIT_TIME_COMPENSATE_ERROR can be used. The resulting transit time difference, after application of compensation if used, is reported in the calibration data block, but only up to a random accuracy defined by TRANSIT_TIME_CALIB_ERROR.

Although variations of the high voltage will, in reality, also result in changes in the p.e.-to-p.e. transit time jitter and in the pulse shape, that depends on the details of the voltage devider. For simplicity, they are assumed to be the same for all pixels. Also the collection efficiency – either implied by the single-p.e. response distribution (PM_PHOTOELECTRON_SPECTRUM) or explicit on top of that distribution (PM_COLLECTION_EFFiciency) – is assumed to be independent of the high voltage. Note that the simulated transit time jitter of Gaussian r.m.s. TRANSIT_TIME_JITTER, includes effects due to position on the photocathode - which we don't know since the ray-tracing ends at the entry to the light cone - and due to the p.e. initial velocity vector.

With the majority trigger logic, there are parameters controlling per-pixel differences in the required time over threshold (DISCRIMINATOR_VAR_TIME_OVER_THRESHOLD) and the required charge over threshold (DISCRIMINATOR_VAR_SIGSUM_OVER_THRESHOLD) as well as the amplitude and (minimum) width of the output signal (DISCRIMINATOR_OUTPUT_VAR_PERCENT, DISCRIMINATOR_VAR_GATE_LENGTH).

The baseline of the digitised signal varies from pixel to pixel by FADC_VAR_PEDESTAL and FADC_LG_VAR_PEDESTAL for high gain and low gain, respectively. A common offset of all pedestals in the camera can be set through FADC_SYSVAR_PEDESTAL and FADC_LG_SYSVAR_PEDESTAL. Where multiple interlaced FADCs are used for the same channel, each has a separate pedestial, differing from others in the same channel by FADC_DEV_PEDESTAL. In principle, pedestals are still assumed to be exactly known, unless FADC_ERR_PEDESTAL / FADC_LG_ERR_PEDESTAL are greater than zero. Sometimes the true spread of pedestal values can be a hindrance to effective image cleaning and a homogenisation or compensation step gets applied after readout, for a smaller spread of apparent pedestals after compensation. See parameters FADC_COMPENSATE_PEDESTAL, FADC_LG_COMPENSATE_PEDESTAL, FADC_ERR_COMPENSATE_PEDESTAL, and FADC_LG_ERR_COMPENSATE_PEDESTAL.

Reproducing the same per-pixel settings in a telescope (as well as random misalignments of mirrors etc.) is currently only possible when using a copy of the same random number status file (RANDOM_STATE) each time or the same RANDOM_SEED (the first seed, at least). The optional file defined by CHANNEL_SAVE_RESTORE to save such parameters from one run to another (and otherwise allow independent random numbers) may not include all relevant variables.

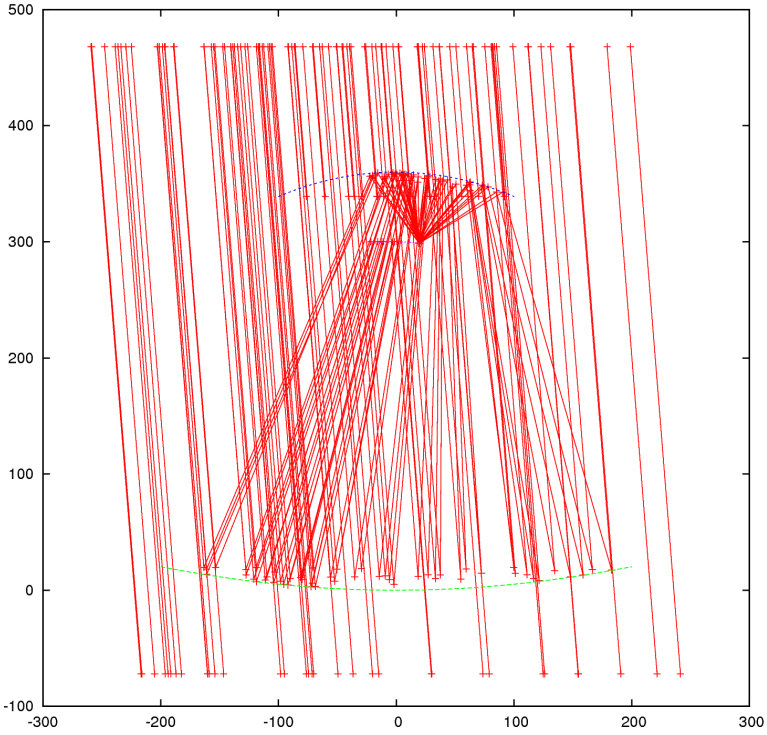

Figure 11.14: Ray-tracing with a dual-mirror telescopes.

11.19 Special settings for dual-mirror telescopes

With MIRROR_CLASS=2 a dual-reflector optics gets simulated. The two reflectors of a dual mirror telescope (see Figure 11.14) and also the focal surface are assumed to follow an even polynomial of the distance to the telescope optical axis (PRIMARY_MIRROR_PARAMeters, SECONDARY_MIRROR_PARAMeters, FOCAL_SURFACE_PARAMeters, with the distance to the axis evaluated in units of PRIMARY_REF_RADIUS, SECONDARY_REF_RADIUS, and FOCAL_SURFACE_REF_RADIUS, respectively). Apart from that the primary and secondary mirror are defined by an outer diameter (PRIMARY_DIAMETER, SECONDARY_DIAMETER, obsoleting the use of any MIRROR_LIST) and an inner diameter (PRIMARY_HOLE_DIAMETER, SECONDARY_HOLE_DIAMETER), by an optional file specifying the segmentation of the reflector (PRIMARY_SEGMENTATION, SECONDARY_SEGMENTATION, see Figure 11.15), and by extra elements always included as shadowing elements in the ray-tracing (SECONDARY_SHADOW_DIAMETER, SECONDARY_SHADOW_OFFSET, SECONDARY_BAFFLE). Pixels in the (normally curved) camera focal surface can be either parallel to the telescope optical axis (PIXELS_PARALLEL=1, that is staggered) or aligned to look to the focal surface normal at the pixel position (PIXELS_PARALLEL=0, see Figure 11.16). Mirror reflectivity curves can be provided separately for primary (MIRROR_REFLECTIVITY) and secondary (MIRROR_SECONDARY_REFLECTIVITY, the special value "same" indicates to use the same ta-

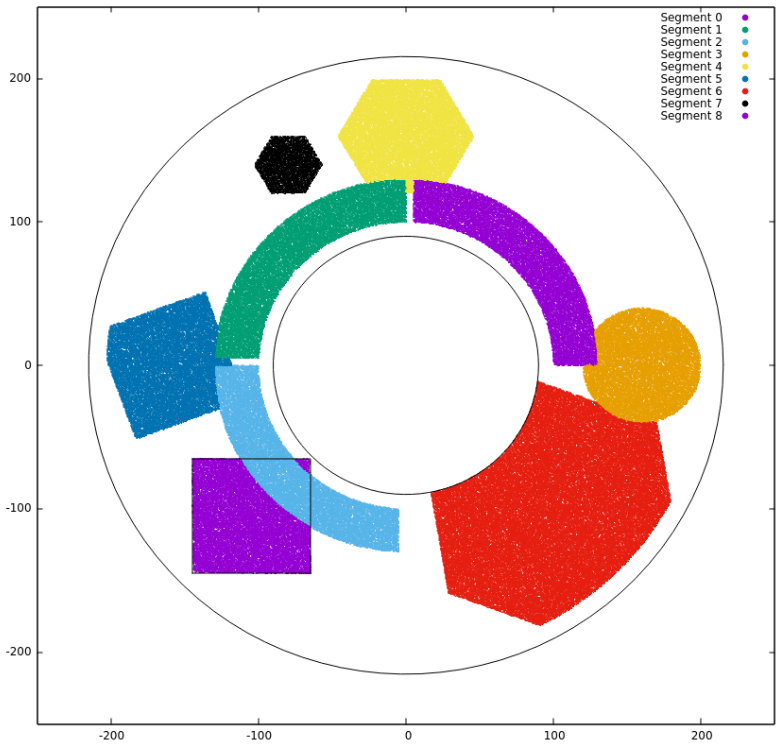

Figure 11.15: Possible but unrealistic segmentation of a dual mirror primary reflector. The inner circle indicates the hole diameter, the outer circle the mirror diameter (the effective ray-tracing mirror edge seen just inside that circle is due to the size of the secondary mirror, as only impact positions of photons traced through the complete optics are shown).

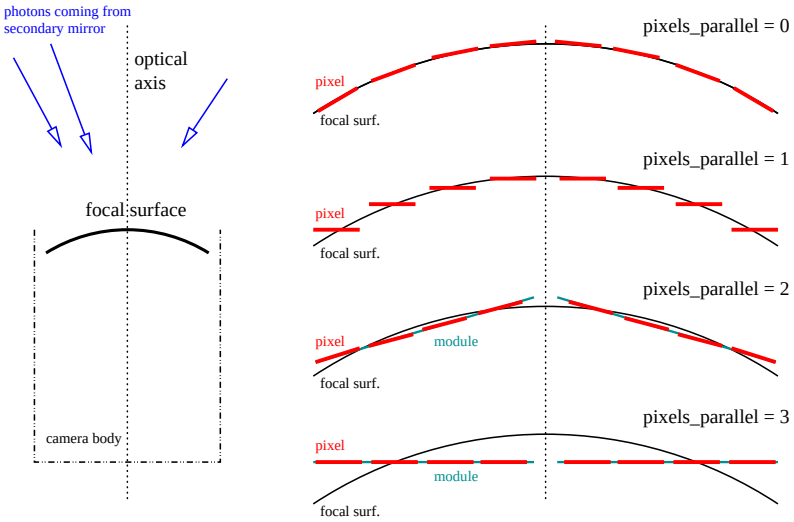

Figure 11.16: Pixel alignment types in the camera of a dual-mirror telescope with a curved focal surface. Note that pixels_parallel=1 is the default.

ble as for the primary). In addition to position dependent mirror degradation on the primary (PRIMARY_DEGRADED_MAP), similar degradation can be set for the secondary (SECONDARY_DEGRADED_MAP).

11.20 Fresnel lens telescopes

With MIRROR_CLASS=3 a simplified optical simulation of a thin Fresnel lens is applied. Instead of actually defining the steps in the optical surface, each photon sees a planar entrance side and then immediately an inclined surface on the exit side, with the inclination corresponding to a spherical lens of radius $r = f(n-1)$ for a given focal length f (FOCAL_LENGTH) and an assumed index of refraction n of the glass, so far assumed as wavelength independent and set with the LENS_REFIDX_NOMinal parameter. The photons never encounter the steps seen in a real Fresnel lens, the lens is infinitesimally thin, there are no internal reflections, except where total reflection prohibits rays leaving the lens on the exit side, and the effect of diffraction gets ignored. The size and boundary shape of the Fresnel lens is defined by a single entry in the MIRROR_LIST file. Parameters like RANDOM_FOCAL_LENGTH and MIRROR_ALIGN_RANDOM_DISTance should be used with care for the single lens and best set to zero. There are no parameters specific for the Fresnel lens optics at this point but most of the parameters for single-reflection telescopes apply by analogy. For example, the MIRROR_REFLECTIVITY table applies for the transmission of the lens. The CAMERA_BODY_DIAMETER and CAMERA_BODY_SHAPE are not relevant since no shadowing by the camera applies in this case. A rather unusual (and negative) value of MIRROR_OFFSET would be needed to place the lens (as "mirror") in front of the az/alt axes, while the usual values with other optics classes would have the (primary) mirror at or behind the axes intersection.

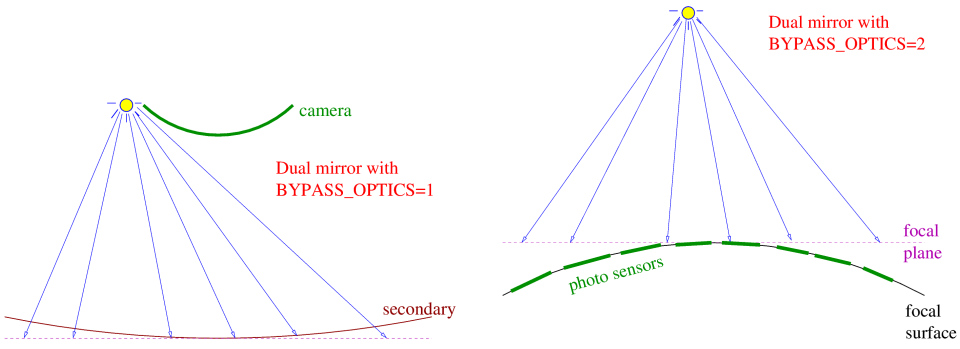

Figure 11.17: Bypassing part or all of the ray-tracing for flatfield calibration devices in a dual-mirror telescope.

11.21 Bypassing the ray-tracing

Normal `sim_telarray` simulations would work with photon bunches hitting a telescope fiducial sphere, trace them to the primary mirror, then from there to the camera (single reflection or Fresnel lens telescope) or to the secondary mirror and then on to the camera (dual-mirror telescope). For calibration devices, the light source might be inside the telescope or, with parts of a camera, in a laboratory. In such cases, parts or all of the ray-tracing can be skipped. Either the camera front plane or the plane behind the secondary mirror is treated as the detector fiducial sphere mid-plane, where the photon bunches get recorded. From there, the photons only need to be followed from the camera front plane into the pixels or from the secondary mirror back-plane to the camera and into the pixels. For a single-reflection or Fresnel lens telescope, any non-zero value of `BYPASS_OPTICS` will assume the photons are in the camera front plane. For a dual-telescope a `BYPASS_OPTICS` value of 1 will bypass the first part of the ray-tracing, assuming the photons are on the secondary-mirror back-plane, while a value of 2 assumes them on the camera front-plane ('focal plane', for a curved camera not coinciding with the actual focal surface which the pixels would, ideally, follow). Figure 11.17 illustrates the two variants for flatfield calibration devices.

Chapter 12

sim_telarray configuration parameters

12.1 Parameter definitions and parameter syntax

All sim_telarray configuration is processed with the module hconfig.c of the hessio package. Several lists of structures of type CONFIG_ITEM are declared for this purpose. Each structure holds

1. Keyword. The uppercase part of the configured name must be given full, the lowercase part can be abbreviated. These parameter names (or valid abbreviations) may then be used in upper, lower, or mixed case.
2. Type. Like 'I' or 'Int' for integer, 'Double', 'Func', ... See below.
3. Number of elements (-1 for functions).
4. Pointer to the variable holding the configuration (NULL for functions).
5. Pointer to configuration functions (NULL for variables).
6. Initial value (initial argument for functions).
7. Lower bound (if present). No boundary applies if missing or empty.
8. Upper bound (if present). No boundary applies if missing or empty.
9. Optional flag bits like, for example, CFG_HARD_BOUND to indicate that any violation of lower or upper bound (where present) should result in an error. For parameters defined without CFG_HARD_BOUND, configured values outside the given boundaries are silently limited to the corresponding boundary.

The parameter type has to match the variable (or array of variables) where it gets stored, including the size and signed/unsigned of integer types. Some freedom in this matching exists for logical (boolean) type parameters - which can either be set up as an integer of size and sign matching the unterlying variable or as a 'Bool' variant, if that is available for the corresponding size. The same abbreviation rules apply to the type as to the name, that is that the upper-case part is required and the rest is optional, all case-insensitive. For a parameter type internally declared "UInt", acceptable user-level declarations, for example as in files hess_defaults.h or cta_defaults.h, would include "UI", "uIN", "uint" etc. Exact

matches would be preferred for readability and safety in case of future changes to the list of available types. Currently, it covers:

Character	Integer value stored in a 'char' variable (beware of values below zero or above 127 for reasons of portability).
UCharacter	Unsigned integer value (0 to 255) stored in an 'unsigned char' variable.
XCharacter	Unsigned integer value of the same size and range as 'UCharacter' but with values provided as hexadecimal, for example "4a" for 74.
Short	A signed integer stored in a 'short' variable.
UShort	An unsigned integer stored in an 'unsigned short' variable.
XShort	Like 'UShort' but specified in hexadecimal, for example "ffFF" for 65535.
Integer	A signed integer stored in an 'int' variable.
UInteger	An unsigned integer stored in an 'unsigned int' variable.
XInteger	Like 'UInteger' but specified in hexadecimal.
Long	A signed integer stored in a 'long' variable.
ULong	An unsigned integer stored in an 'unsigned long' variable.
XLong	Like 'ULong' but specified in hexadecimal.
Bool	A boolean logical value (0 or 1) stored in a 'bool' variable type, if such a type is known to the compiler or in a 'char' variable, if not. In addition to the "0" and "1" values, abbreviations of "False", "True", "Yes", "No", "ON", and "OFf" are acceptable. Other integers are not accepted. These features and restrictions also apply to potential upper and lower bounds. Listed values and bounds will be shown in integer representation.
IBool	Behaves like the "Bool" type but is stored in an 'int' variable. Ideal for transition of an "Integer" parameter with range 0 to 1 to a boolean type, as the underlying variable remains the same.
UBool	Also behaves like the "Bool" type but is stored in an 'unsigned int' variable.
FLoat	A floating-point value stored in a 'float' variable.
Double	(or 'Real') A floating-point value stored in a 'double' variable.
Text	A zero-terminated character string with a maximum length (in bytes, not counting the terminating zero byte) not exceeding the given number of elements. The assigned 'char *' array needs to be large enough for that, including the additional final zero byte. As long as the length restriction is satisfied, it is insensitive to character encoding.
FUnction	Triggers the call of the user-defined or built-in function with the given character string as its argument. No variables are directly associated with the parameter. Requires a '-1' as the number of elements, a NULL pointer for variables, and a non-NULL function pointer to a matching function.

See the `hessio` source code documentation for details. Setting a configuration value in the configuration file can be done by a line like

```
telescope_altitude 100
```

or

```
Telescope_Altitude=100
```

The case of the entered keywords is ignored. The equal sign is just an optional placeholder. Setting a vector of variables may look like any of the following:

```
mirror_opt = -0.0025, -0.10, 0.0
nightsky_background all: 0.049
mirror_x (9,10,13-14): 53
mirror_opt 3*0
```

The first form specifies each individual value. Values not entered are unchanged, either at a value set earlier or - if never set - at the compile-time default value. The second form sets all elements of a vector, the third form a selected subset of a vector and the fourth a given number (3) of elements starting with the first. The different addressing schemes can be combined, even on a single line – but that can be confusing and is not recommended. Multiple lines are possible:

```
nightsky_background all: 0.049
nightsky_background (1536-2047): 0.065
nightsky_background (13): 0.040
nightsky_background (14): 0.045
nightsky_background (15): 0.053, (16): 0.052, (17): 0.49
...
```

The following (somewhat long) listing shows the variable declarations used for the configuration. See the `CONFIG_ITEM` structures for the names of all possible configuration parameters. Note that most lengths are in units of centimeters (as in CORSIKA) but there are exceptions like in `ALTITUDE`, `CONVERGENT_POSition`, `CONVERGENT_DISTance`, or `CONVERGENT_Height` where meters are used. All angles are in degrees. Also note that some default or maximum values may depend on the presence of preprocessor symbols (e.g. HESS_PHASE1 or CTA_PROD6). Filename parameters are all indicated as having a maximum length of 4095. That can be system dependent, as the maximum of `PATH_MAX`, `FILENAME_MAX` (as defined in system header files), and 1024, minus one for a trailing zero byte. The system itself may have separate limits for the pathname part and the filename part, perhaps depending on the filesystem on which the files are located.

12.2 Global configuration parameters

Global parameters can only be set before the configuration for the individual telescopes is being processed (TELESCOPE undefined or zero when pre-processing the configuration file). No telescope-specific values are possible, in contrast to parameters described in the later sections. Global parameters for meta-information only are described in Subsection 12.4.

12.2.1 Input and output files

- **INPUT_FILE**
  (type: Text, max. length: 4095, default: "iact.dat")
  List of input files, usually taken from command line, following any command line options.

- **PLOT_FILE**
  (type: Text, max. length: 4095, default: "none" /* e.g. "plot.gpl"*/)
  Name of an ASCII format file for miscellaneous information. A value of "none" here and in the following means that no such file is used.

- **HISTOGRAM_FILE**
  (type: Text, max. length: 4095, default: "ctsim.hdata")
  File with histograms (use 'hdata2hbook' to convert to hbook or 'hdata2root' to convert to ROOT-format histogram files).

- **PHOTOELECTRON_FILE**
  (type: Text, max. length: 4095, default: "none" /*"iact_pe.dat"*/)
  Obsolete, synonym for `OUTPUT_FILE`.

- **OUTPUT_FILE**
  (type: Text, max. length: 4095, default: "none" /* e.g. "ctsim.dat"*/)
  Output in HESS/HEGRA-specific data format.

- **IMAGE_FILE**
  (type: Text, max. length: 4095, default: "none" /* e.g. "image.ps"*/)
  File for Postscript camera images. These are the nice plots.

- **IMAGING_LIST**
  (type: Text, max. length: 4095, default: "none" /* e.g. "imaging.lis"*/)
  File with camera hit positions of star light photons. Relevant for point-spread function checks.

- **STARS**
  (type: Text, max. length: 4095, default: "none")
  An optional file with a list of stars shining, containing one row per star with azimuth, altitude (both in degrees) and weighting factor for the number of photons. An optional fourth value is the distance of the light source (note: in kilometers) or 0. (standing for infinity). In addition to ray-tracing evaluation the stars also contribute to the nighsky background rate in the affected pixels.

- **RANDOM_STATE**
  (type: Text, max. length: 4095, default: "none")
  The random number generator in this program can save its internal state at the beginning and/or the end of execution and read it in again when started the next time. When several runs are done in parallel they may all start with the same random

number sequence (which may or may not be what you intended). Except for special values (see below) this parameter would normally have a file name for storing/loading the random generator state. Since actual random number seeds are reported in the data file, there is little remaining use for this feature. The special value `none` does neither write nor read a file (changed 2023-12-14). The special value `init` (`none` before 2023-12-14) would just write the initial state into a file named `telarray_rand.conf.used` but not read an existing state file. The special value `auto` indicates that the initial state should be written into `telarray_rand.conf.used-`*pid* (*pid* being the `sim_telarray` process ID). The `auto` case is also supposed to remove its state file at the end of the program, unless program execution was interrupted.
Other values, if corresponding to existing files, would restore that into the random generator state, after applying the (first) seed and obsoleting that. For other names then `none`, `init`, and `auto`, the actual state is to be saved at the beginning and, again, at the end of the program. A second `RANDOM_SEED` value would still get applied.
The format of these files differs between the built-in Ranlux random generator and the generators provided through the GNU Scientific Library (GSL). Attempts to load files written for a different generator may leed to unpredictable results.

12.2.2 Global parameters for general set-up

- **MOVIE**
  (type: IBool, items: 1, default: "0", minimum: "0", maximum: "1")
  True if images for a movie to be made (one camera image per sample interval). The resulting output goes into the file for Postscript camera images (`IMAGE_FILE`).

- **POWER_LAW**
  (type: Double, items: 1, default: "2.68")
  Power law of weighted spectrum. Events will be histogrammed with a weight according to the difference between this exponent and the one used for the shower simulations.

- **ALTITUDE**
  (type: Double, items: 1, default: "1800.", units: meters)
  Altitude of observation level [m] above sea level. It is mainly used to check that the atmospheric transmission table corresponds to the proper altitude.

- **CONVERGENT_POSition**
  (type: Double, items: 3, default: "0., 0., 0.", units: meters)
  Reference *x*/*y*/*z* position [m] from where nominal viewing direction is true direction. *x* is counted towards North, *y* towards West, and *z* upwards from the altitude of the observation level, just as the telescope positions. By how much the telescope viewing direction converges towards some point on that axis (or diverges away from that axis) can be set by one of the telescope-specific parameters `CONVERGENT_`

DISTance, CONVERGENT_Height, and CONVERGENT_Depth (and depends on the telescope position).

- **SELECT_LIGHt_component**
  (type: Int, items: 1, default: "0", minumum: "-1", maximum: "3")
  Normal simulations will accept any photon bunch for processing. With non-zero values of this parameter there are two ways to switch between two classes of light sources, both aimed at different versions of CORSIKA with fluorescence light emission but other classes are possible, depending on the data. For a value of '1' only bunches with positive bunch size are accepted (normal Cherenkov bunches, with or without wavelength), for a value of '-1' only bunches with negative bunch size are accepted (meant to encode fluorescence or other alternate light source). For a value of '2' only bunches with zero wavelength (Cherenkov bunches without wavelength) are accepted and for a value of '3' only bunches with positive wavaelength (either Cherenkov light generated with CERWLEN enabled in CORSIKA or fluorescence light) are accepted.

- **STAR_PHOTONS**
  (type: Long, default: "4000")
  The number of photons, randomly thrown over 1.2 times the actual telescope diameter and ray-traced to the camera. Mirror reflectivity and quantum efficiencies are taken as 100 percent for evaluating the resulting currents and random rates per pixel.

- **SKY_IS_VARiable**
  (type: Int, items: 1, default: "0")
  If true (non-zero), then sky is recalculated after pointing changes. Note that this can be *very* CPU-intense and is also likely to cause problems with the NSB-dependent pedestal offsets in DC-coupled cameras.

- **ONLY_TRIGGERED_ARRAYS**
  (type: IBool, items: 1, default: "1")
  If true (non-zero), then only showers which triggered the array are considered. If you want to have data also for non-triggered showers, set this to zero.

- **ONLY_TRIGGERED_TELESCOPES**
  (type: IBool, items: 1, default: "1")
  If true (non-zero), then only triggered telescopes are read out. If you want to have data also for non-triggered telescopes, set this to zero.

- **ARRAY_TRIGGERs**
  (type: Text, max. length: 4095, default: "none")
  If present (value not "none"), this parameters indicates the name of a file with multiple combinations of telescopes from which a valid stereo trigger is accepted. These lines can also include specific coincidence gate widths and different telescope multiplicities for each line. Example:

```
Trigger 2 of 2, 3, 5, 6 to 59, 97 to 168 width 1000
```

- **ARRAY_WINDOW**
  (type: Double, items: 1, default: "100", units: nanoseconds)
  The length of a coincidence window of the default stereo triggers, after correction of fixed (cable length, focal length, etc.) and variable (view direction) delays. This value is overriden by the values in the file which may be specified by ARRAY_TRIGGERs.

- **ARRAY_CLOCK_WINDOW**
  (type: Double, items: 1, default: "0.", units: nanoseconds)
  The array-wide time values when the readout in each telescope was started may contain MC-true information in the light arrival zero-point. That is unphysical but necessary to place the photo-electron lists properly w.r.t. the pulse traces. Otherwise this value, if non-zero, can be used to scramble any such zero-point information as single random offset (flat between zero and the given value) will be added to all telescope readout start values.

- **SAVE_CALIBRATION_PE**
  (type: Int, items: 1, default: "0", minimum: "0", maximum: "1")
  Separate from the option to save the list of signal photo-electrons in external events read from the input (usually shower events), saving the signal photo-electrons produced by the laser or LED light in internal calibration events can be enabled with this parameter set to 1. The SAVE_PE_WITH_AMPlitude controls both for external (shower) events and internal calibration events if random amplitudes are to be reported in addition to times.

- **SAVE_PE_WITH_AMPlitude**
  (type: Int, items: 1, default: "0", minimum: "0", maximum: "3")
  True if photo-electron data is to be stored with the resulting amplitudes (in mean-p.e. units) in addition to the arrival times. Storing of photo-electron data should be activated separately (see SAVE_PHOTONS (bit 1) and STORE_PHOTOELECTRONS parameters) but non-zero values of this parameter will also activate SAVE_PHOTONS bit 1. Storing p.e. in internal calibration events is controlled separately by SAVE_CALIBRATION_PE.

- **SAVE_PHOTONS**
  (type: Int, items: 1, default: "0", minimum: "0", maximum: "3" or "7")
  True if to save photon data (from the CORSIKA photon bunches) and/or the list of Cherenkov photo-electrons registered to the output file. Set bit 0 on (value 1 or 3) to save CORSIKA photon bunches and set bit 1 on (value 2 or 3) to save photo-electrons. If `sim_telarray` got compiled with `-DSTORE_PIX_PHOTONS`, bit 2 can be used to switch between storing of the total number of photons incident onto each pixel (SAVE_PHOTONS=2) to selecting only photons in the 300-550 nm wavelength range (SAVE_PHOTONS=6). When compiling with

`-DSTORE_PIX_PHOTONS=2`, bit 2 is always on. (That compile-time option costs significant CPU time in either case, as the usual short-cuts to bypass ray-tracing for most photons cannot be applied.)

- **CHANNEL_SAVE_RESTORE**
  (type: Text, max. length: 4095, default: "none")
  File to save channel parameter data. If such a file is given and not present, the random parameters are saved. If the file is present, parameters are restored from it. This is useful if several simulations should use the same random deviations of quantum efficiencies, high voltages etc. but differ in other random properties not related to camera response.

- **MAXIMUM_EVENTS**
  (type: Int, items: 1, default: "0", minimum: "0")
  In case a simulation test is not supposed to go over the full length of the input data but finish up after a given number of events encountered on the input, set a non-zero value here.

- **MAXIMUM_TRIGGERED_EVENTS**
  (type: Int, items: 1, default: "0", minimum: "0")
  In case a simulation test is not supposed to go over the full length of the input data but finish up after a given number of events have triggered the telescope system, set a non-zero value here.

- **IGNORE_NONTRIGgered_showers**
  (type: IBool, items: 1, default: "0", minimum: "0", maximum: "1")
  If this flag is true (1), Monte Carlo data is only written for showers/events that actually triggered the telescope system. This may be useful if most simulated showers are below threshold and the amount of data should be kept as low as possible. The drawback of setting this flag is that calculations of effective areas, spectra and so on can be done only on the basis of nominal simulated shower numbers (per core distance or energy interval) and so on. Studying for example systematic differences between triggered and non-triggered showers becomes completely impossible.

- **MC_ONLY_TRIGGERED_showers**
  (type: IBool, items: 1, default: "0", minimum: "0", maximum: "1")
  An alias to the `IGNORE_NONTRIGgered_showers` keyword.

- **IGNORE_MC_DATA**
  (type: IBool, items: 1, default: "0", minimum: "0", maximum: "1")
  If this flag is true (1), Monte Carlo specific data is omitted from the output, thus trying to look pretty much like real measured data. This concerns in particular the amount of data stored.

- **IOBUF_MAXimum**
  (type: Long, items: 1, default: "200000000", minimum: "100000")

The buffers for input and output of eventio data are limited to a maximum size which can be defined by this parameter.

- **IOBUF_OUTPUT_MAXimum**
  (type: Long, items: 1, default: "20000000", minimum: "100000")
  The maximum size of the I/O buffer for output data, including raw data for all telescopes (in bytes).

- **RANDOM_GENERATOR**
  (type: Text, max. length: 59, default: "Ranlux")
  Define the random number generator to be used. Unless compiled with `-DWITH_GSL_RNG` the only valid and known generator is "Ranlux". When using the GNU Scientific Library for random number generators, this can be "Ranlux", "mt19937", "taus", "Ranlxd2", or "GSL:" plus the name of any other GSL-based generator, or "GSL_RNG_TYPE" to obtain the generator from the environment variable of that name.
  Note that, although the built-in and the GSL-based Ranlux generator produce the same sequence of 'flat distribution' random numbers (for identical seeds), the Gaussian and Poisson distribution numbers are derived in different ways from the flat distribution numbers. Simulated events will therefore differ.

- **RANDOM_SEED**
  (type: Text, max. length: ..., default: "auto")
  `Sim_telarray` normally generates an automatic random number seed from different sources. Each run will then be unique, even with different optics and electronics random settings. To avoid user mistakes resulting in running again and again with the same random number sequence this is also the recommended way. If a fixed random number seed is needed for specific comparisons, debugging, etc., it can be set as a positive integer in this parameter. As an extension, there can be a comma-separated second value which – if it is non-empty and not "0" (zero) – is used to set a new random number generator seed after the configuration stage, with "auto" or a negative number again resulting in an auto-generated seed value. One possible application could be "1234,auto" for a user-defined seed in the configuration stage (reproduceable setup, "1234" just being an example value) but independent NSB photo-electrons and independent detection efficiencies for each use of the program in the later stages.
  To allow groups of simulation runs to use the same random detector parameters, it is possible to select pre-configured seeds from a file, following one of several strategies. For each of them, the number of suitable lines in the given text file with one integer per line is counted. A chosen integer modulo the number of lines determining which line is to be used. For strategy `file-by-run:` the chosen integer is based on the `CORSIKA_RUN` environment variable. For `file-by-time:` it is based on the time (seconds and microseconds), for `file-by-random:` some data from `/dev/urandom` plus time plus process ID get used. A second seed of "auto" is

highly recommended in such a case. Example:
`random_seed=file-by-run:random_seeds.txt,auto`

- **ALWAYS_AWEIGHTs**
  (type: Int, items: 1, default: "0", minimum: "-1", maximum: "1")
  If this switch is set to 1, output data always contains the area weights for each MC event, even if core positions (or actually the array offsets) of the events were thrown in a uniform distribution over the whole area. When the input data contained no area weights they are calculated.
  If the switch is set to -1, output never contains the area weights for MC events, even if events were thrown in a non-uniform distribution.
  If the switch is set to 0, area weights are written to output if they were found in the input.

- **ALL_WL_RANDOM**
  (type: IBool, items: 1, default: "0", minimum: "0", maximum: "1")
  There is a special variant of CORSIKA to mark direct Cherenkov light by the primary particles (like iron nuclei) through a wavelength of 1 instead of the usual 0. This would normally be interpreted as a wavelength of 1 nm, resulting in undetectable photons. If this variable is set to 1, even non-zero wavelengths are randomized between the limits. Thus, showers simulated with this special variant of CORSIKA can be processed once with direct light included and once with direct light ignored.

12.2.3 Atmospheric transparency table filename

The extinction of Cherenkov light (either by absorption or by scattering) is expressed by a table of optical depths (negative log of the probability to reach ground level) as a function of wavelength and starting altitude. These are typically based on calculations with the MODTRAN or 3S tools. These tables can be tweaked for more or less aerosols, additional absorbing layers etc. without going through MODTRAN or 3S again, for example with `atmtrans_interpol`. An experimental and very simple way of including an additional absorbing layer is available if `sim_telarray` gets compiled with `-DEXTRA_CLOUD` but the recommended way would be to come up with a corresponding transmission table.

- **ATMOSPHERIC_TRANSMISSION**
  (type: Text, max. length: 4095, default: "hess_atmo_trans.dat")
  A file with optical thickness as a function of emission altitude and wavelength. There is a variety of such tables available, all calculated with the MODTRAN program. If the file contains more levels than supported (MAX_TRANS_HEIGHT), this will result in a configuration error.
  Note: In `sim_telarray` versions before 2023-08-28, any extra levels got simply ignored.

- **CLOUD_HEIGHT**
  (type: Double, items: 1, default: "-1000.", minimum: "-1000.", maximum:

"150000.", units: meters, compilation definition required: EXTRA_CLOUD)
In addition to the optical depth table in the file specified by ATMOSPHERIC_TRANSMISSION, a geometrically thin layer of wavelength-indendent (gray) extinction can be added (compilation with -DEXTRA_CLOUD required). This parameter specifies the height of this layer above sea level. Unit, like ALTITUDE, is in meters. Values below ALTITUDE indicate that no such layer is present.

- **CLOUD_TRANSMISSION**
(type: Double, items: 1, default: "1.0", minimum: "0.0", maximum: "1.0", compilation definition required: EXTRA_CLOUD)
See CLOUD_HEIGHT. This parameter specifies the fraction of light transmitted through the extra absorbing layer.

12.2.4 Source direction and reference position

- **SOURCE_AZIMUTH**
(type: Double, items: 1, default: "0.", units: degrees)
Azimuth angle of source or 0. The value '0.' has a special meaning: the telescope azimuth value is used. Azimuth is counted from North towards East. Note that this is different from the settings in the CORSIKA INPUTS file where azimuth is counted from magnetic South towards magnetic East. This parameter is only used for shower analysis, but not for the simulation itself.

- **SOURCE_ALTITUDE**
(type: Double, items: 1, default: "0.", units: degrees)
Altitude angle of source or 0. See SOURCE_AZIMUTH.

- **REFERENCE_POSITION**
(type: Double, items: 3, default: "0., 0., 750.", units: centimeters **!!**)
Reference position with respect to the observation level. This position is used in calculating the shower core distance from the 'array'. This distance is used for filling some histograms.

12.2.5 Other quasi-global parameters

- **MAXIMUM_TELESCOPES**
(type: Int, items: 1, default: "16", minimum: "1")
The maximum number of telescopes that are configured at program startup. This should be at least as large as the number of telescopes used in the CORSIKA simulation and at most as large as the hardcoded limit MAX_TEL (currently by default 16 but modified by the compile-time configuration options, see mc_aux.h). If sim_telarray was compiled with HESS_PHASE_1 defined, the default and

maximum possible is only 4. Other configurations may allow for hundreds of telescopes. For efficiency reasons, it should normally be set to the number of telescopes simulated with CORSIKA.

- **BASE_TELESCOPE_NUMBER**
  (type: Int, items: 1, default: "1")
  The number at which 'official' telescope numbers (usually counted in natural numbers, i.e. from 1 upwards) start. If a single telescope simulated should correspond to CT3, you could set this to 3.

12.3 Telescope-specific parameters

These parameters can differ from one telescope to another. Use `#if TELESCOPE==...` in the chosen configuration file if individual values are intended. Otherwise they apply to all telescopes. Note: On the command line, it is not possible to distinguish between telescopes.

12.3.1 Uncategorized so far (were incorrectly in global section)

- **MIN_PHOTONS**
  (type: Double, items: 1, default: "-1.")
  Minimum no. of photons required before doing simulation. **Note**: *This will save a lot of CPU time when many of the simulated showers are of too low energies or too far away to trigger the telescopes. In events where a telescope receives less light, no full simulation is attempted. The number corresponds to the sum over the CORSIKA photon bunches and thus is before atmospheric absorption, etc. etc. With H.E.S.S. simulations, I am usually using values between 200 and 500. Optimum values depend on the wavelength range used, on the radius of the enclosing sphere, on the trigger thresholds and other parameters. The safe (but inefficient) default is not to ignore any events. I usually check with fewer showers what the lowest number of photons is in triggered events, and then leave factor of 2 or so margin.*

- **MIN_PHOTOELECTRONS**
  (type: Int, items: 1, default: "-1")
  Minimum no. of detected photoelectrons required in a camera before running the more CPU-intense electronics simulation. **Note**: *This can save CPU time in addition to `MIN_PHOTONS`. This cut is applied after ray-tracing all photons to the pixels and deciding if photons are detected. A `MIN_PHOTONS` cut is generally much more effective since it can bypass the complete simulation. A cut in `MIN_PHOTOELECTRONS` should be used only in addition to the former. It adds particularly in efficiency when most photons miss the camera.*

- **STORE_PHOTOELECTRONS**
  (type: Int, items: 1, default: "-1")

While the SAVE_PHOTONS=2 configuration option will save photo-electron data for all events, this parameter allows to set a lower limit on the number of photo-electrons which must get registered before deciding to store them. Setting it to a value of zero or bigger (default: -1) will automatically enable the corresponding SAVE_PHOTONS bit. It should be cross-checked that the MIN_PHOTONS parameter is unlikely to result in fewer than the given number of photo-electrons, as no attempt is made to trace photons through the telescope and produce any photo-electrons if there are too few photons in the first place.

- **TAILCUT_SCALE**
  (type: Double, items: 1, default: "1.0", minimum: "0.0", maximum: "10")
  The special scheme for selecting pixels belonging to a shower image as implemented in `sim_telarray` (as opposed to the traditional two-level tailcut scheme) has multiple thresholds for the various levels of confidence that a pixel actually belongs to the shower image. Instead of making them individually configurable, a common scale factor can be applied to all of them.

12.3.2 Telescopes

- **TELESCOPE_PHI**
  (type: Double, items: 1, default: "0")
  Azimuth angle of the telescope(s). Unit: degrees from North towards East. When this is set in the global context, it defines the nominal array pointing position. Individual telescopes may differ from that, for example when setting up custom convergent/divergent patterns that cannot be represented by the CONVERGENT_DISTance, CONVERGENT_Height, and CONVERGENT_DEPTH parameters. If the parameter is set on the `sim_telarray` command line with a '-C' option, that would override any individual setting in the configuration file. Make sure to use the `-W` option (weak configuration) if you expect values in the configuration file to take priority.

- **TELESCOPE_AZIMUTH**
  (type: Double, items: 1, default: "0", units: degrees)
  This is simply a synonym to TELESCOPE_PHI.

- **TELESCOPE_THETA**
  (type: Double, items: 1, default: "0", units: degrees)
  Zenith angle of the telescope. Same rules as for TELESCOPE_PHI for configuration versus command-line priority.

- **TELESCOPE_ALTITUDE**
  (type: Func) This is an alternative to giving a zenith angle [degrees] (TELESCOPE_THETA will be set to 90 − TELESCOPE_ALTITUDE). Do not confuse this parameter with the (global) site ALTITUDE parameter.

- **TELESCOPE_RANDOM_ANGLE**
  (type: Double, items: 1, default: "0.001", units: degrees)
  Random (known) misalignment of the telescope in each axis [degrees].

- **TELESCOPE_RANDOM_ERROR**
  (type: Double, items: 1, default: "0.001", units: degrees)
  Random (unknown) alignment error of the telescope in each axis [degrees].

- **CONVERGENT_DISTance**
  (type: Double, items: 1, default: "0.", units: meters)
  Distance from array reference position [m] where viewing directions should intersect. If this distance is negative, the telescopes are aligned for divergent pointing, to directions opposite to where they would be pointing for convergence at the given height below observation level.

- **CONVERGENT_Height**
  (type: Double, items: 1, default: "0.", units: meters)
  Height above obs. level [m] where viewing directions should intersect. If this height is negative, the telescopes are aligned for divergent pointing, to directions opposite to where they would be pointing for convergence at the given height below observation level. If non-zero, the `CONVERGENT_DISTance` value gets ignored.

- **CONVERGENT_DEPTH**
  (type: Double, items: 1, default: "0.", units: g/cm^2)
  Atmospheric depth [g/cm^2] where viewing directions should intersect. It is counted from the top of the atmosphere along the nominal viewing direction of the telescope array. A negative value indicates that divergent pointing is requested. If non-zero, both `CONVERGENT_Height` and `CONVERGENT_DISTANCE` get ignored. If custom setting of the viewing direction is chosen via `TELESCOPE_THETA` and `TELESCOPE_PHI`, no convergence correction on top of that is likely needed.

- **RANDOM_VIEWING_RING**
  (type: Double, items: 2, default: "all: 0.", minimum: "0.", maximum: "10.", units: degrees)
  Minimum and maximum radius of ring [degrees] around basic phi/theta direction to which system viewing direction is randomised. Note that this does not have the same effect as the VIEWCONE option in CORSIKA, in particular near zenith. When there are stars included in the simulation and their positions on the camera should change accordingly, you also have to set the `SKY_IS_VARiable` parameter – but note that pretty much CPU time is then spent on ray-tracing star light.

- **REVERSE_MODE**
  (type: IBool, items: 1, default: "0", minimum: "0", maximum: "1")
  If 1, then reverse positioning is used (telescope altitude angle is beyond zenith position).

- **TELESCOPE_TRANSmission**
  (type: Double, items: MAX_TELTRANS, default: "0.89,0,0,0,0,0")
  Accounting for absorption/shadowing by masts and other telescope structure elements not explicitly included in the `sim_telarray` ray-tracing. Although actual absorption varies a bit over the field of view, it is taken as flat in the default value shown here. A value of 0.89 means that 11% of all light is assumed to be absorbed by masts. If compiled with the `RAYTRACING_INTERSECT_RODS` pre-processor definition, the default value is 1.00, as the absorption should get calculated explicitly.
  New: extension for angle-dependent transmission (effect of shadowing) adds more parameters. The extra parameters are: function number (0: constant, 1: variable $T(\theta)$) and the necessary parameters. Function number 1 is $T(\theta) = p_0/(1. + p_2((\sin\theta)/r_3)^{p_4})$ for $p_5 = 0$ or $T(\theta) = p_0/(1. + p_2((\sin\theta)/r_3)^{p_4})^{p_5}$ for $p_5 \neq 0$ where $r_3 = p_3\pi/180$ and p_0, p_2, p_3, p_4, p_5 are the first, and third to sixth values; with default values of $p_4 = 2$, $p_3 = 0.5d/f$ when these are missing/zero. The p_5 parameter may not be available/supported and is effectively zero/unused if not.
  This parameter or set of parameters (if angle-dependent) is typically derived by fitting the ratio of effective geometrical mirror area obtained by a detailed ray-tracing tool devided by the equivalent area obtained with the simplified ray-tracing in `sim_telarray`, as a function of the angle of incidence θ w.r.t. the optical axis. It is intended as a geometrical factor only. Other (wavelength-independent) factors decreasing the optical efficiency include `MIRROR_DEGRADED_REFLECTION`, `MIRROR2_DEGRADED_REFLECTION` (dual-mirror telescopes only), and `CAMERA_DEGRADED_EFFICIENCY`.

- **MASTS_File**
  (type: Func, default: "hess_masts.dat")
  If `sim_telarray` was compiled with the `RAYTRACING_INTERSECT_RODS` pre-processor definition, a file with the geometry of masts and additional elements is read and explicitly accounted for in the ray-tracing. Since this is very CPU-intense, it is not recommended for normal use. Note that `TELESCOPE_TRANSmission` is ignored then (set to 1.0) in order not to count the masts twice. For efficiency reasons, this parameter is normally not available, and no masts explicitly included in the raytacing.

12.3.3 Mirrors

Different categories of optical set-ups

- **MIRROR_CLASS**
  (type: Int, items: 1, default: "0", minimum: "0", maximum: "3")
  Allows to switch from the usual segmented primary mirror optics with spherical mirror tiles (0) to alternate configurations. This includes a parabolic pri-

mary (1), made of a single piece, and a secondary mirror optics (2), with primary and secondary each made of a single piece. Many of the other optics parameters do not apply when the alternate designs are chosen. For mirror class 1 only the focal length applies as well as the surface quality. For class 2 not even the focal length applies. For this type, the optical surfaces are entirely defined by polynomials, as set up with `PRIMARY_MIRROR_PARAMeters`, `SECONDARY_MIRROR_PARAMeters`, and `FOCAL_SURFACE_PARAMeters`. Segmentation of the primary and secondary reflector is optional and set up with `PRIMARY_SEGMENTATION` and `SECONDARY_SEGMENTATION`. Class 3 represents a simple Fresnel lens implementation.

- **BYPASS_OPTICS**
  (type: Int, items: 1, default: "0", minimum: "0", maximum: "2")
  Experimental, may need special compilation flag. Bypass the ray-tracing in the telescope reflectors completely or partically. For a single-reflector telescope values of 1 or 2 both correspond to complete bypassin. For dual-reflector telescopes a value of 1 bypasses only reflection on the primary mirror, a value of two both reflections. In case of complete bypassing the camera focal *plane* is assumed to be at the fiducial sphere mid-plane (camera looking up), in case of partial bypassing the center of the secondary is at the center of the fiducial sphere and the secondary is looking up. These settings only make sense with simulated artificial light sources, e.g. camera calibration or testing devices, and not with CORSIKA shower simulations.

For most or all types of optical configurations

- **FOCAL_Length**
  (type: Double, items: 1, default: "1500", minimum: "10", maximum: "10000")
  Nominal overall focal length of whole telescope. This basically defines the image scale (near the centre of the field of view). For segmented primary focus telescopes this determines the alignment of the segments and the separation from the reflector, at its center to the camera. For secondary mirror configurations this value is not actually used in the optics simulation but only reported as a nominal value, typically close to the effective focal length. Unit: cm.

- **EFFECTIVE_FOCAL_Length**
  (type: Double, items: 5, default: "0", minimum: "0", maximum: "10000")
  Due to asymmetric image abberations, in particule for single-reflector telescopes, segmented or not, the inverse image 'plate scale' or, more precisely the ratio of off-center distance in the focal plane (projection for curved focal surface) to the tangent of the off-axis angle does not match the nominal focal plane very well. This effective value here should be better suitable for shower reconstruction than the nominal focal length and is supposed to be based on ray-tracing simulations of point sources, at distances typical of showers and imaged onto the actual focal surface at the level of the pixel entrances. Non-zero values will be reported as-is in the output data and may be used for the built-in reconstruction in `sim_telarray`. If no value is given,

sim_telarray may estimate a value, based on optics type and f/D ratio, for its internal purpose but will not report such an estimate in the output data. Due to subtle effects of image cleaning (and thus image intensity) on cutting off parts of the asymmetric point-spread function, analysis programs should use this value or, if not available, an estimate of it only as a starting point and evaluate the actual analysis-specific and perhaps image intensity and NSB dependent real effective focal length by itself.
A single value is sufficient for a axisymmetric PSF distribution. For a mirror more elongated in one direction than in the other direction, or a mirror offset from the optical axis, there may be further non-zero values. The complete list:

1. Mean effective length for all directions of incidence.
2. Effective length for incidence directions in mirror/camera $x-z$ plane, if non-zero.
3. Effective length for incidence directions in mirror/camera $y-z$ plane, if non-zero.
4. Any displacement along x in the focal plane from asymmetric PSF behavior.
5. Any displacement along y in the focal plane from asymmetric PSF behavior.

Unit for all: cm.

- **EFFECTIVE_MIRROR_AREA**
(type: Double, items: 1, default: "0", minimum: "0")
Meta-parameter only, not used by sim_telarray. It may be extracted for use by testeff or other tools and used for scaling the expected amount of Cherenkov or NSB light. Supposed to correspond to the mirror area obtained with on-axis sim_telarray PSF ray-tracing, without applying TELESCOPE_TRANSMISSION and other corrections or effeciency factors. That means the only shadowing applied at this stage is usually due to camera body and/or secondardy mirror. Unit: cm^2.

- **MIRRORS**
(type: Int, items: 1, default: "1", minimum: "1", maximum: MAX_MIRRORS)
Number of mirrors on the telescope. In the typical case with a list of mirror positions etc. (see MIRROR_LIST) this is meant to match the number of mirrors specified in that file. For secondary-mirror configurations (MIRROR_CLASS=2) it is supposed to be 2 (segmentation being not counted for that). For single paraboloid mirrors or Fresnel lens optics, the intended value is 1. Actually, the default value of "1" is a safe choice, effectively replaced by the real number of mirrors. Therefore, the recommendation is to leave the default value untouched. Other values might be still useful as a (limited) safeguard against accidentally using the wrong mirror list.

- **MIRROR_REFLECTION_RANDOM_Angle**
(type: Double, items: 3, default: "0.0066,0.,0.", minimum: "0.0", maximum: "2.0",

units: degrees)
Gaussian random fluctuations of reflection angles, due to small-scale surface deviations, with one or two components. The first parameter is the projected r.m.s. of the first component. If two components are used, the second parameter is the fraction following the second component and the third parameter is the projected r.m.s. of that second component. The value of 0.0066 degrees together with RANDOM_FOCAL_Length of 7.4 cm was adapted to measurements of single H.E.S.S.-I mirror spot sizes in Namibia. Unit: degrees (except for second parameter).

- **MIRROR_REFLECTION_RANDOM_Table**
(type: Text, max. length: 4095, default: "none")
Not yet implemented. If a file name for a table of random reflection angle intensities (relative intensity per solid angle versus random angle in degrees) is given, it replaces the normal distribution(s) produced with MIRROR_REFLECTION_RANDOM_Angle.

- **MIRROR_DEGRADED_REFLECTION**
(type: Double, items: 1, default: "1.0" /* 1.0 means as good as in reflectivity table */, minimum: "0.0", "1.0")
The overall optical efficiency or throughput is by the given factor worse than the efficiency from nominal tables (same factor at all wavelengths). This might include actually degraded reflectivity, dust on the mirror(s), or on/in the camera. It used to be the only degradation parameter in older sim_telarray versions and may still represent the combined effect rather than accounting for the degraded optical efficiency of the primary mirror (this parameter and also PRIMARY_DEGRADED_MAP), the optional secondary mirror (MIRROR2_DEGRADED_REFLECTION and SECONDARY_DEGRADED_MAP), and the camera (CAMERA_DEGRADED_EFFICIENCY, CAMERA_DEGRADED_MAP) separately. What matters in normal simulation is only the overall product of the efficiencies. Breaking it down into mirror(s) and/or camera degradation only matters where BYPASS_OPTICS becomes involved. Then this parameter is actually interpreted as being for reflection on the primary mirror (would not be relevant for a flat-fielding light source directly illuminating the camera or illuminating it via reflection on the secondary mirror). The nightsky background in each pixel is also scaled by the same factor (even if the actual simulation bypasses optical ray-tracing). For wavelength-dependent degradations you need adapted reflectivity, window transmission, light cone efficiency etc. tables. For a Fresnel-lens telescope, this would correspond to a degraded optical efficiency of the lens.

- **MIRROR2_DEGRADED_REFLECTION**
(type: Double, items: 1, default: "1.0" /* 1.0 means as good as in secondary mirror reflectivity table */, minimum: "0.0", "1.0")
Similar to MIRROR_DEGRADED_REFLECTION but strictly interpreted as being

for reflection on the secondary mirror of a dual-mirror telescope, and still gets applied in simulations where only the reflection on the primary mirror is bypassd (for example a flat-fielding light source illuminating the camera via reflection on the secondary). For a single-reflector telescope it is always ignored. See also SECONDARY_DEGRADED_MAP.

- **PRIMARY_DEGRADED_MAP**
  (type: Text, max. length: 4095, default: "none")
  An optional position-dependent map of degradation factors for the primary mirror only. Expected format is the rpolator x/y/z format (nd=3). Acceptable values are in the range between 0.0 and 1.0. This degradation gets applied on top of MIRROR_DEGRADED_REFLECTION. Note that the impact of it on the nightsky background is not automatically accounted for and must be evaluated separately and included in the configured NSB pixel p.e. rates. Therefore, it is recommended to use a map with an average efficiency of 1.0 and use MIRROR_DEGRADED_REFLECTION for the overall degradation.
  Tables with equidistant spacing in both x and y are highly recommended because the interpolation is faster.

- **SECONDARY_DEGRADED_MAP**
  (type: Text, max. length: 4095, default: "none")
  Same as PRIMARY_DEGRADED_MAP, applying to the secondary mirror in dual-mirror optics only. It gets applied on top of MIRROR2_DEGRADED_REFLECTION in the actual ray-tracing. Like PRIMARY_DEGRADED_MAP it is not applied to scale down the configured NSB pixel p.e. rates. See recommendations there.

- **MIRROR_REFLECTIVITY**
  (type: Text, max. length: 4095, default: "hess_reflect.dat")
  Mirror reflectivity (wavelength dependent) is loaded from the given file. Note that with secondary mirror optics this is still the reflectivity for one mirror and is thus applied twice.

- **MIRROR_SECONDARY_REFLECTIVITY**
  (type: Text, max. length: 4095, default: "same")
  The reflectivity curve of the secondary reflector in a dual-mirror configuration, by default assumed to be identical to the primary.

For all kinds of segmented primary mirror configurations

- **DISH_SHAPE_Length**
  (type: Double, items: 1, default: "0", minimum: "0", maximum: "10000", units: centimeters)
  Dish curvature length, best equal to focal length. For a Davies-Cotton dish, this is radius of the sphere on which the mirror tiles are positioned. For a parabolic dish, this is the focal length of the paraboloid on which the mirrors are placed. This is not

normally needed but only when variations to the standard shapes are tried out, e.g. intermediate shapes between parabolic and Davies-Cotton. For the solid secondary mirror optics (mirror class 2), this parameter does not apply. Unit: cm. Normally 0 to adapt it to overall focal length.

- **MIRROR_FOCAL_Length**
  (type: Double, items: 1, default: "0", minimum: "10", maximum: "10000", units: centimeters)
  Standard focal length of mirror tiles. This parameter is only used if the focal lengths of the mirror tiles in the mirror configuration file are 0. For a parabolic dish any non-zero value will then force all mirror tiles to have the same focal lengths (and which should then match the average distance of the parabolic dish from the focus, not exactly the overall focal length). For the solid secondary mirror optics (mirror class 2), this parameter does not apply. Unit: cm. Normally 0 to adapt it to overall focal length.

- **PARABOLIC_DISH**
  (type: Int, items: 1, default: "0", minimum: "0", maximum: "1")
  Allow replacing spherical dish outline (Davies-Cotton) by a parabolic dish with isochronous on-axis propagation time but slightly worse off-axis imaging quality. Mirror tiles still have spherical shape then but their focal lengths are adapted to the distance from the focus (since the geometric mean of minimum and maximum radius of curvature of the paraboloid is very close to that distance). For the solid secondary mirror optics (mirror class 2), this parameter does not apply.

- **GRADING_OF_FOCAL_Length**
  (type: Double, items: 1, default: "0.")
  Gradual change of focal length from inner to outer mirrors, starting from Davies-Cotton design. Not recommended any more. Use explicit values in the mirror list file instead.

- **RANDOM_FOCAL_Length**
  (type: Double, items: 1, default: "7.4, 0.", units: both centimeters)
  Random error in mirror facet focal lengths in a segmented single-dish reflector. Unit: cm. These parameters get only applied if the focal lengths in the mirror list file (`MIRROR_LIST`) are zero (automatic) or negative (individual with error). For positive focal lengths in that list (definite values) the random focal length value does not get applied. The first parameter sets a Gaussian distribution, the second a top-hat distribution – which can be combined. (Historical note: The default value of 7.4 cm together with `MIRROR_REFLECTION_RANDOM_Angle` of 0.0066 degrees was adapted to measurements of single HESS mirror spot sizes in Namibia.)

- **FOCUS_OFFSET**
  (type: Double, items: 4, default: "2.8, 28., 0., 0.", units: cm,degrees,cm,cm)
  Distance of the starlight focus from the camera pixels (light guides) entry [cm]. In

telescopes without dynamic focusing capability (either through mirror alignment and camera positioning) this is best set such that starlight gets focused onto the camera lid while showers are focused onto the pixel entry. Positive values indicate that the pixel entries are – for the incoming light – behind the starlight focal surface, as appropriate for focusing at a finite object distance. For MIRROR_CLASS = 1 or 2 this is further from the primary mirror, at a distance of focal length plus focus offset from the (virtual) center of the reflector. For MIRROR_CLASS = 3 it is further from the secondary mirror and for MIRROR_CLASS = 4 it is further from the Fresnel lens.
The zenith angle dependence is experimental and not used by default. It includes a reference zenith angle θ_0 at which the nominal value applies and a coefficient for the dependence on the cosine and on the sine of the zenith angle (see also MIRROR_ALIGN_RANDOM_HORIzontal but note that components are added linear for the focus offset, as in $\Delta_f = \Delta_0 + \Delta_c + \Delta_s$, for Δ_0 being the first parameter, $\Delta_c(\theta) = (cos(\theta) - \cos(\theta_0))k_c$, and $\Delta_s(\theta) = (sin(\theta) - \sin(\theta_0))k_s$ where θ_0 is the second, k_c the third, and k_s the fourth parameter.

- **MIRROR_OFFSET**
  (type: Double, items: 1, default: "130.", units: centimeters)
  Offset of mirror backplane from fixed point of telescope mount (if axes intersect) or from the altitude rotation axis, along the direction of the optical axis. Positive if fixed point (or altitude axis) is between the mirror backplane and the focus or, in other words, the center of the primary mirror is behind/below the altitude rotation axis. Unit: cm.

- **AXES_OFFSETs**
  (type: Double, items: 2, default: "0.,0.", units: centimeters)
  Geometric offsets to be used in case that the azimuth, altitude, and optical axis do not intersect at the reference point (the center of the fiducial sphere in CORSIKA simulations). The first value is a horizontal offset between the (vertical) azimuth axis and the (horizontal) altitude rotation axis. A positive value corresponds to an altitude axis towards the reflector. If the altitude axis is on the optical axis, the second parameter is zero. Otherwise it represents the displacement perpendicular to the optical axis and the altitude axis of the altitude rotation axis w.r.t to the reflector. If both values are non-zero but equal, the optical axis coincides with the azimuth axis for vertical pointing.

- **MIRROR_F_SCALE**
  (type: Double, items: MAX_MIRRORS, default: "all: 1.", minimum: "1.0", maximum: "1.0")
  List of focal length scaling factors (for each mirror individually).

- **MIRROR_ALIGN_RANDOM_ANGLE**
  (type: Double, items: 1, default: "0.0030", units: degrees)
  Obsolete parameter now superceded with MIRROR_ALIGN_RANDOM_

`HORIzontal` **and** `MIRROR_ALIGN_RANDOM_VERTical`**. It can be reproduced if the third and fourth value of those is set to zero and the first component of those is set to the same value.** Fluctuations of the mirror alignment angle with respect to nominal alignment. Unit: degrees.

- **MIRROR_ALIGN_RANDOM_HORIzontal**
  (type: Double, items: 4, default: "0.0035, 28., 0.023, 0.0", units: deg/deg/deg/deg)
  Fluctuations of the mirror alignment angle with respect to nominal alignment in the horizontal component (in azimuth or y direction in dish coordinates) are parametrized by four values. First a constant minimum value σ_0, second a zenith angle θ_0 a which the minimum is reached (that is typically where alignments were done), third a term k_c used in $\sigma_c = (\cos(\theta) - \cos(\theta_0))k_c$, and fourth a term k_s used in $\sigma_s = (\sin(\theta) - \sin(\theta_0))k_s$. The three terms are added up as independent (i.e. by squares; therefore the square of the horizontal mirror alignment error is $\sigma^2 = \sigma_0^2 + \sigma_c^2 + \sigma_s^2$). The default values are based on a fit to the measured point spread functions of H.E.S.S. CT2 and CT3 in Namibia, with the single mirror point-spread function adapted beforehand. Note that, since this affects the angles of the mirrors, the impact on the combined spot size is twice as large. Units: all in degrees.

- **MIRROR_ALIGN_RANDOM_VERTical**
  (type: Double, items: 4, default: "0.0034, 28., 0.01, 0.0", units: deg/deg/deg/deg)
  Fluctuations of the mirror alignment angle with respect to nominal alignment in the vertical component (in altitude or x direction in dish coordinates). See the `MIRROR_ALIGN_RANDOM_HORIzontal` for details. Units: all in degrees.

- **MIRROR_ALIGN_RANDOM_DISTance**
  (type: Double, items: 1, default: "2.0", units: centimeters)
  Fluctuations in the aligned distance from the focus. Unit: cm.

- **MIRROR_OPT**
  (type: Double, items: 3, default: "0,0,0")
  Optimisation parameters relative to simple Davies-Cotton design. Not used in many years since there are more explicit ways now to configure the mirror set-up. Deprecated.

- **FLIP_MIRRORS**
  (type: Int, items: 1, default: "0", minimum: "0", maximum: "1")
  If this parameter is non-zero, the mirror configuration is flipped over the $x = y$ axis, i.e. the x and y coordinates of each mirror tile are swapped and hexagonal mirrors of type 1 are changed into type 3, and vice versa. Since off-axis PSF evaluation in the y-z plane may use this parameter, it is recommended to set up the mirror list file such that it corresponds to `flip_mirrors=0`.

For secondary mirror optics only

- **PRIMARY_DIAMETER**
  (type: Double, items: 1, default: "0.", units: centimeters)
  The outer diameter of the primary reflector, only used in case no MIRROR_LIST file is used (i.e. MIRROR_LIST=none). If the primary mirror is segmented (PRIMARY_SEGMENTATION), no reflection is possible at a distance from the optical axis of more than half the diameter, effectively clipping the outer edge of the segments. Units: in cm.

- **PRIMARY_HOLE_DIAMETER**
  (type: Double, items: 1, default: "0.", units: centimeters)
  The primary mirror may have a central hole since the mirror there would be basically obstructed by a large secondary. With a segmented primary, it effectively clips the inner edge of the segments. Units: in cm.

- **PRIMARY_MIRROR_PARAMeters**
  (type: Double, items: MAX_NPAR_MIRR, default: "all: 0.", units depend on PRIMARY_REF_RADIUS)
  Defines the position of the primary mirror along the optical axis and its shape. The first parameter (p_0) is the offset of the mirror with respect to the common reference point defined by MIRROR_OFFSET, with positive values indicating that the centre of the primary (assuming it has no central hole) is above the reference point. Apart from that, a parameter p_i adds a term $p_i r^(2i)$ to the height of the mirror at radial offset r. A parabolic primary will have the second parameter (p_1) at a positive value and all further parameters at 0. (which can be omitted since this is the pre-defined default). Units: all in cm (for radius r also in cm).

- **PRIMARY_REF_RADIUS**
  (type: Double, items: 1, default: "1.0", units: centimeters)
  The length scale (in cm) to which the PRIMARY_MIRROR_PARAMeters parameters apply. Typical values could be 1.0 or the focal length of the primary. Only applicable for MIRROR_CLASS=2. Units: in cm.

- **PRIMARY_SEGMENTATION**
  (type: String, default: "none")
  The name of a file with the segments or segment groups of the primary reflector. If not active (value "none") then the primary is assumed to be of one circular piece, with an optional central hole. The outer and inner diameter still apply, even in case of segmentation.
  The segmentation files (both for primary and secondary) have lines of the following syntax:

  ```
  type nseg ...
  ```

  where 'type' is one of RINGsegments, SQUAREsegments, (Y)HEXsegments, CIRCULARsegments, or POLYgonsegments, and 'nseg' is the number of segments

(1 except for ring segments). The additional parameters after nseg depends on the segment type. For RINGsegments (shortest abbreviation: RING, case ignored) it is

```
RINGsegments nseg rmin rmax dphi [ phi0 [ gap ] ]
```

(with zero implied for missing phi0 or gap values) and for POLYgonsegments it is

```
POLYgonsegments 1 rot x1[, ]y1 x2[, ]y2 x3[, ]y3 [ ... ]
```

listing all the corners (with or without the last one being the same point as the first one) while for all other types it is

```
type 1 x y diam [ rot ]
```

For ring segments 'rmin' and 'rmax' are the minimum and maximum projected radius, 'dphi' is the azimuthal angle subtended per segment, and 'phi0' is the angle where the first segment starts. The 'gap' parameter specifies a fixed-width gap between neighbouring segments, with half the gap size being clipped off from either (azimuth) side of a segment. For the other segment types, always specified individually (nseg=1), 'x' and 'y' are the projected position of the segment center, 'diam' is the (flat-to-flat for square and hexagon) diameter, measured perpendicular to the surface normal in the segment center, and 'phi0' is the rotation angle of the segment. At angle 0 there are two sides parallel to the x-axis for both SQUAREsegments and HEXsegments (a corner of the hexagon pointing towards x), and two sides parallel to the y-axis for YHEXsegments. Any part of a segment inside the PRIMARY_HOLE_DIAMETER or outside the PRIMARY_DIAMETER is clipped off.

- **SECONDARY_DIAMETER**
  (type: Double, items: 1, default: "0.", units: centimeters)
  The outer diameter of the secondary reflector, only used in case no MIRROR_LIST file is used (i.e. MIRROR_LIST=none). With a segmented secondary, it effectively clips the outer edge of the segments. Units: in cm.

- **SECONDARY_HOLE_DIAMETER**
  (type: Double, items: 1, default: "0.", units: centimeters)
  The diameter of any non-reflective but not transparent (black) central part of the secondary mirror in a dual-mirror telescope (with MIRROR_CLASS=2). With a segmented secondary, it effectively clips the inner edge of the segments. Units: in cm.

- **SECONDARY_SHADOW_DIAMETER**
  (type: Double, items: 1, default: "-1.", units: centimeters)
  The diameter of any non-reflective but not transparent (black) central part of the secondary mirror in a dual-mirror telescope (with MIRROR_CLASS=2). A value of -1 indicates that the same value as for the reflective diameter of the secondary SECONDARY_DIAMETER is to be used. Units: in cm.

- **SECONDARY_SHADOW_OFFSET**
  (type: Double, items: 1, default: "0.", units: centimeters)
  The secondary mirror shadowing element is normally assumed at the level of the

edge of the secondary. If this parameter is non-zero, it can be set at any position above the center of the primary mirror (with MIRROR_CLASS=2). The reference point is the same as for primary, secondary, camera, or baffle (correct values should thus be around the first of the SECONDARY_MIRROR_PARAMeters times SECONDARY_REF_RADIUS, see also SECONDARY_BAFFLE). Even if placed in front of the secondary, it will not be used for photons reflected from the primary to the secondary or from the secondary to the camera, only for the photons incident onto the primary. Units: in cm.

- **SECONDARY_BAFFLE**
  (type: Double, items: 3 to 5, default: "all: 0.", units: centimeters)
  Defines a baffle around the secondary mirror (with MIRROR_CLASS=2) to keep NSB light from falling directly into the camera. The values are z_1, z_2, r_1, plus optionally dr and r_2. The first two values define the beginning and end of the baffle along the optical axis, counted from the primary center, which means a z value of SECONDARY_REF_RADIUS times the first SECONDARY_MIRROR_PARAMeters value is at the height of the center of the secondary. The radius of an open cylinder or cone shape at the corresponding z values is set by the r_1 and r_2 values. For now only cylinder accepted ($r_2 = r_1$ or $r_2 = 0$). The optional dr value specifies the wall thickness of the baffle, with r_1 and r_2 being inner radii in case of finite wall thickness. Absence of a baffle is indicated by $r_1 = 0$. Shadowing by the baffle is applied both to incoming photons and to photons reflected from the primary to the secondary mirror. Units: all in cm.

- **SECONDARY_MIRROR_PARAMeters**
  (type: Double, items: MAX_NPAR_MIRR, default: "all: 0.", units depend on SECONDARY_REF_RADIUS)
  Defines the position of the secondary mirror along the optical axis and its shape. They are defined just like PRIMARY_MIRROR_PARAMeters except that the secondary mirror looks the other way, i.e. p_0 will usually be negative since in the coordinate frame of the secondary, it is 'below' the common reference point. A concave secondary, reducing the focal length if placed between the primary and its focus, will have $p_1 > 0$. A convex secondary (e.g. for Cassegrain or Ritchey-Chrétien optics), enlarging the focal length, will have $p_1 < 0$. Units: all in cm (for radius r also in cm).

- **SECONDARY_REF_RADIUS**
  (type: Double, items: 1, default: "1.0", units: centimeters)
  The length scale (in cm) to which the SECONDARY_MIRROR_PARAMeters parameters apply. Typical values could be 1.0 or the focal length of the secondary or the primary. Only applicable for MIRROR_CLASS=2. Units: in cm.

- **SECONDARY_SEGMENTATION**
  (type: String, default: "none")

The name of a file with the segments or segment groups of the secondary reflector. If not active (value "none") then the secondary is assumed to be of one circular piece, with an optional central hole. The outer and inner diameter still apply, even in case of segmentation. See PRIMARY_SEGMENTATION for the file syntax, except that SECONDARY_DIAMETER applies instead of PRIMARY_DIAMETER and SECONDARY_HOLE_DIAMETER instead of PRIMARY_HOLE_DIAMETER.

- **FOCAL_SURFACE_PARAMeters**
  (type: Double, items: MAX_NPAR_MIRR, default: "all: 0.", units depend on FOCAL_SURFACE_REF_RADIUS)
  Defines the position of the focal surface along the optical axis and its off-axis shape. They are defined just like PRIMARY_MIRROR_PARAMeters. The FOCUS_OFFSET still applies, but with a curved focal surface only in the camera centre, such that star light would be focused on the camera lid surface but light from the typical distance of the shower maximum would be focused on the pixel entrance. Note that this offset may be impractibly small with secondary mirrors reducing the plate scale. The direction of the incoming rays is not transformed into the normal plane of the focal surface, thus corresponding to pixels shifted w.r.t. to a plane. Actual implementations, if really following a curved shape (technically difficult), may differ. Units: all in cm (for radius r also in cm).

- **FOCAL_SURFACE_REF_RADIUS**
  (type: Double, items: 1, default: "1.0", units: centimeters)
  The length scale (in cm) to which the FOCAL_SURFACE_PARAMeters parameters apply. Only applicable for MIRROR_CLASS=2. Units: in cm.

- **PIXELS_PARALLEL**
  (type: Int, items 1, default: "1")
  A value of 1 indicates, that pixels are only shifted parallel to the optical axis. Pixel surfaces remain perpendicular to the optical axis. A value of 0 indicates that the pixel is oriented such that the surface is aligned with the focal surface (pixel axis parallel to focal surface normal). A value of 2 places all pixels belonging to the same module (as configured in 'Pixels' lines in the CAMERA_CONFIG_FILE) on a common plane, with its orientation along the normal to the focal surface in the module center (c.o.g. of pixels) and with zero mean displacements of pixels along the z axis. A value of 3 (=2+1) has all pixels in a common plane per module and all of the modules (and thus all of the pixels) looking parallel to the optical axis. The 'height' (in z) of the module is again the average of where individually placed pixels would be (zero mean displacement in module). An extension to the camera definition file even allows for pixels to be configured individually and not exactly following the focal surface (in both height and orientation). Only applicable for MIRROR_CLASS=2. For an illustration see Figure 11.16.

For Fresnel-lens optics only

- **LENS_REFIDX_NOMinal**
  (type: Double, items: 1, default: "1.50", minimum: "1.001", maximum: "5.0")
  The nominal index of refraction of the material of a Fresnel lens (used for calculating the necessary radius of curvature) and its actual wavelength-independent index of refraction in the absence of an actual table of the wavelength dependence. Only applicable for MIRROR_CLASS=3. Units: in cm.

File with list of individual mirror positions

The actual number of mirrors (unless exceeding MAX_MIRRORS) is just the number of useful lines in the mirror list file.

- **MIRROR_LIST**
  (type: Text, max. length: 4095, default: "hess_mirrors.dat")
  List of mirror positions. If such a file is given, all direct mirror positions are ignored. Columns in this data file are:

  1. mirror x position [cm],
  2. mirror y position [cm],
  3. flat-to-flat diameter [cm],
  4. focal length [cm] (a value of zero will be replaced by the MIRROR_FOCAL_LENGTH parameter (plus random), positive values are used as-is, negative values will get a RANDOM_FOCAL_LENGTH added),
  5. shape type (0: circular, 1/3: hexagon, 2: square),
  6. mirror z position (optional) [cm] (a value of zero means an automatically generated position depending on dish shape type (spherical/parabolic, PARABOLIC_DISH), nominal focal length (FOCAL_Length) and dish curvature (DISH_SHAPE_Length).

  For dual-mirror optics this file can be used to define the diameters of primary and secondary (circular shape required) but can be turned off (value "none") and these diameters instead directly set with the PRIMARY_DIAMETER and SECONDARY_DIAMETER parameters, the recommended procedure. If the mirror list file is used for dual-mirror optics, the primary is the first mirror in the list, the secondary the second one, both taken on-axis. For more details and for optional further columns see Section 11.7.

Direct mirror positions (obsolete and not used in H.E.S.S./CTA simulations)

These parameters, historically used in HEGRA CT simulations, are no longer available.

- **MIRROR_X**
  (type: Double, items: MAX_MIRRORS, default: "all: 0.")
  X position of all mirrors. Unit: cm.

- **MIRROR_Y**
  (type: Double, items: MAX_MIRRORS, default: "all: 0.")
  Y position of all mirrors. Unit: cm.

- **MIRROR_Diameter**
  (type: Double, items: MAX_MIRRORS, default: "all: 0")
  Diameters of all mirrors. For `MIRROR_CLASS`=2 and `MIRROR_LIST`=none, the diameters of primary and secondary mirrors can be specified here. Unit: cm.

- **MIRROR_TYPE**
  (type: Int, items: MAX_MIRRORS, default: "all: 0", minimum: "0", maximum: "2")
  Type: 0 (circ.), 1 (hex, flat -> x), 2 (hex, flat edge -> y).

12.3.4 Camera

All camera types

- **CAMERA_TYPE**
  (type: Int, items: 1, default: "3", minimum: "3", maximum: "3")
  Obsolete and disabled types 1 (hexagonal pixels/camera) and 2 (square) were used without camera definition file. Only mentioned for historical reasons. For many years now only type 3 (flexible camera configuration) is available, with pixels etc. set up in a camera definition file.

- **CAMERA_PIXELs**
  (type: Int, items: 1, default: "960", minimum: "1", maximum: `MAX_PIXELS`)
  Number of pixels in camera (allowed values depend on compile-time definitions). Must match the number of pixels set up in the camera definition file.

- **CAMERA_BODY_SHAPE**
  (type: Int, items: 1, default: "0", minimum: "0", maximum: "3")
  Shape type of camera body (for shadowing). Values as for other shape types: 0= circular (default), 1= hexagonal with two sides parallel to y axis, 2= square, 3= hexagonal with two sides parallel to x axis.

- **CAMERA_BODY_DIAMETER**
  (type: Double, items: 1, default: "160", minimum: "0", maximum: "1000", units: centimeters)
  Diameter of camera body (for shadowing). Flat-to-flat for square and hexagonal shapes. Unit: cm.

- **CAMERA_BODY_OFFSET**
  (type: Double, items: 1, default: "0", units: centimeters)
  While for traditional IACTs, mirrors would generally be aligned such that stars get focused on the camera lid, thus the camera front by definition being in the (starlight) focal plane, this is not universally the case. Like FOCUS_OFFSET allows for for an offset of the pixel entrance, this parameter (**not yet implemented**) allows for an offset of the camera body (only used for shadowing). Positive values indicate that the camera body front is – for the incoming light – behind the starlight focal plane (MIRROR_CLASS = 1 or 2: further from the primary mirror; MIRROR_CLASS = 3: further from the secondary mirror). Unit: cm.

- **CAMERA_DEPTH**
  (type: Double, items: 1, default: "0", minimum: "0", maximum: "1000", units: centimeters)
  Depth of a camera body (for shadowing). If non-zero, shadowing is checked both at the height of the camera front and the camera depth. Unit: cm.

- **CAMERA_TRANSmission**
  (type: Double, items: 1, default: "1.00", minimum: "0.01", maximum: "1.00")
  Global and wavelength-independent transmission factor of the camera (including any plexiglass window).

- **CAMERA_FILTER**
  (type: Text, max. length: 4095, default: "none")
  Wavelength dependence of the camera transmission, independent of the pixel type and the angle of incidence. If a file name is provided, the table (wavelength and efficiency columns) will be loaded and interpolated. The corresponding efficiency factors will be applied on top of the global CAMERA_TRANSmission factor, and on top of any pixel-type dependent efficiency (both angular and by wavelength), according to the PixType configuration lines in the file specified by CAMERA_CONFIG_FILE. The file is, by default, assumed to be a 1-D table (wavelength-dependent only, in nanometers) but, with a #@RPOL@ header line (see section 11.3 can be marked and used as a 2-D table (wavelength- and angle-dependent), with the incidence angles specified in the file in units of degrees but internally converted to radians.

- **CAMERA_WINDOW_HEIGHT**
  (type: Double, items: 1, default: "0.0", minimum: "0.0")
  Height of the inner surface of the camera window above the pixel entrance (or photo-sensor, if without light cones), at the camera center. If there is a non-unity CAMERA_SCALE_FACTOR parameter, the scaling gets applied to this and the following window geometry parameters. *Not yet implemented.* Unit: cm.

- **CAMERA_WINDOW_THICKness**
  (type: Double, items: 1, default: "0.0", minimum: "0.0")

Thickness of the camera window material, above the the camera center. No impact on the photon optical path for zero thickness. *Not yet implemented.* Unit: cm.

- **CAMERA_WINDOW_RADIUS**
  (type: Double, items: 2, default: "0.0", minimum: "0.0")
  Use zero radii for flat windows. Otherwise these are the radius of curvature of the inner surface and, optionally, the outer surface. If the inner surface has a non-zero radius but the outer a zero radius, the outer radius is supposed to be the inner radius plus the `CAMERA_WINDOW_THICKness`, resulting in a curved window of constant thickness. *Not yet implemented.* Unit: cm.

- **CAMERA_WINDOW_REFIDX**
  (type: Double, items: 1, default: "1.0", minimum: "1.0")
  Index of refraction of the camera window material. No refraction considered for a value of 1.0 (the index of refraction of air being rather irrelevant for this purpose). *Not yet implemented.*

- **CAMERA_DEGRADED_EFFICIENCY**
  (type: Double, items: 1, default: "1.0", minimum: "0.0", "1.0")
  While `MIRROR_DEGRADED_REFLECTION` and `MIRROR2_DEGRADED_REFLECTION` no longer have any effect (other than for NSB calculations) if `BYPASS_OPTICS` is set to ignore the corresponding optical surface, any degradation of the camera itself will still take effect. Without any optics bypassing, only the product of camera, primary and secondary mirror degradation (for dual-mirror telescopes) or the product of camera and mirror/lens degradation (for all other telescope types) matters.

- **CAMERA_DEGRADED_MAP**
  (type: Text, max. length: 4095, default: "none")
  Like the `PRIMARY_DEGRADED_MAP` and, for dual-mirror telescopes, the `SECONDARY_DEGRADED_MAP` this is a filename for a position-dependent degradation of optical efficiency, in that case at the camera entry (focal plane for single-reflector or Fresnel lens, focal surface for dual-mirror). Expected format of the file is the rpolator x/y/z format (nd=3). Acceptable values are in the range between 0.0 and 1.0. This degradation gets applied on top of `CAMERA_DEGRADED_EFFICIENCY` and is not accounted for in the NSB pixel p.e. rate calculations (and in particular not for a localized NSB reduction). Tables with equidistant spacing in both x and y are highly recommended.

For flexible camera type only

This applies to the H.E.S.S. and CTA variant of the simulation (camera type 3). Camera type 3 is the only type supported with current versions of `sim_telarray`.

- **CAMERA_CONFIG_FILE**
  (type: Text, max. length: 4095, default: "hess_camera.dat")

Additional configuration file for more flexible camera configuration. Pixel types, positions of all pixels, and the circuitry for trigger decisions is defined there.

- **CAMERA_SCALE_FACTOR**
  (type: Double, items: 1, default: "1.0", minimum: "0.1", maximum: "10")
  Scales all pixel positions and sizes from the camera configuration file at the same time. Useful when changing the focal length and keeping the angular scale of the camera constant. The same scaling factor also applies to geometrical parameters of the camera window - but any resulting bulk absorption changes in the window material require a matching `CAMERA_FILTER` table.

For fixed hexagonal/square camera types only (obsolete)

This applies to the HEGRA variant of the simulation (camera types 1 and 2). You cannot set them in current versions of `sim_telarray`. Obsolete and removed.

- **FRONT_VIEW**
  (type: Int, items: 1, default: "0", minimum: "0", maximum: "1")
  Obsolete. Set if camera co-ordinates are as seen from front.

- **PIXEL_SIZE**
  (type: Double, items: 1, default: "3.9", minimum: "0.1", maximum: "100")
  Obsolete. Pixel size (flat to flat). U: cm.

- **PIXEL_DEPTH**
  (type: Double, items: 1, default: "4.0", minimum: "0.0", maximum: "100")
  Obsolete. Depth of PM below pixel entrance.

- **CATHODE_DIAMETER**
  (type: Double, items: 1, default: "2.1", minimum: "0.1", maximum: "100.")
  Obsolete. Photocathode diameter. Unit: cm.

- **LIGHTGUIDE_GAP**
  (type: Double, items: 1, default: "0.01", minimum: "0.", maximum: "1.")
  Obsolete. Insensitive gap between the lightguides of neighbouring pixels.

- **LIGHTGUIDE_REFlectivity**
  (type: Double, items: 1, default: "0.85", minimum: "0", "1")
  Obsolete. Reflectivity of the light guide material in front of the camera.

12.3.5 Photomultipliers or SiPM sensors

- **QUANTUM_EFFiciency**
  (type: Text, max. length: 4095, default: "hess_qe2.dat")
  File name for quantum efficiency curve. Currently only one quantum efficiency curve is used, even if several different pixel types would be defined.

- **PM_COLLECTION_EFFiciency**
  (type: Double, items: 1, default: "1.00", minimum: "0.01", maximum: "1.00")
  Photoelectron collection efficiency in the photomultiplier first stage. The default value of 1.0 is meant to be used together with a single-photo-electron response distribution which includes the low-amplitude signals.

- **PM_AVERAGE_GAIN**
  (type: Double, items: 1, default: "1.72e5" /* 2e5/1.16 */, minimum: "1e4", maximum: "3e7") Only used for DC currents from NSB pixel rates.

- **QE_VARIATION**
  (type: Double, items: 1, default: "0.04" /* From HESS PMT database (Corning blue) */, "0.0")
  Quantum efficiency variation between PMTs. Unit: fraction of average Q.E.

- **ADJUST_GAIN**
  (type: Double, items: 1, default: "1.0", minimum: "0.1", maximum: "10.")
  Common multiplicative adjustment for FADC and discriminator/comparator amplitude and PMT gain.

- **PM_GAIN_INDEX**
  (type: Double, items: 1, default: "5.5" /* From HESS PMT database */, minimum: "0.0", maximum: "15.0") Gain rises as given power of the PMT voltage. A value of zero turns off any voltage variation to adjust gains for compensating QE, thus not affecting pixel transit times. For a PMT where the first dynode is stabilized (see `PM_TRANSIT_TIME`) this applies only to the variable part of the voltage.

- **GAIN_VARIATION**
  (type: Double, items: 1, default: "0.02", minimum: "0.0")
  By how the gain may vary between PMTs after the voltage has been adjusted for approximately the same gains. Unit: fraction.

- **FLATFIELDING**
  (type: Int, items: 1, default: "1", minimum: "0", maximum: "1")
  If enabled, the gains in pixels with higher-than-average QE are decreased to achieve the same signal from the same illumination. If disabled, the gains are adjusted to have equal single-p.e. amplitudes.

- **PIXEL_CELLS**
  (type: Int, items: 1, default: "0", minimum: "0")
  If this value is non-zero, pixel saturation should correspond to the limited number of cells in a pixel (of SiPM type) which each can only fire once during the typical arrival time spread of Cherenkov photons. The presence of a non-zero value does not imply any microscopic model of the pixel structure.

- **PM_VOLTAGE_VARIATION**
  (type: Double, items: 1, default: "0.03" /* HESS PMT database: 6% */, minimum:

"0.0")
Variations of transit times are usually caused by needing different voltages to reach the same gains. Unit: fraction of the total nominal voltage including the voltage between cathode and first dynode, even if that is stabilized. Example: For PMTs with a total nominal voltage of 900 V and the first dynode fixed to 300 V, a value of 0.05 (5%) corresponds to 45 V r.m.s. voltage variation applied to the later stages of the dynode chain (and its related transit time). *Note: The fact that the default variation is much smaller than found in the H.E.S.S. PMT database ist justified if PMT are sorted by voltages into drawers.*

- **PM_TRANSIT_TIME**
  (type: Double, items: 4, default: "20.,0.,0.,0.", minimum: "0")
  Value 1: Total transit time of the PMT at the average/nominal voltage. Value 2: Fixed transit time between cathode and first dynode, in case of the first dynode being stabilized. Use zero for a passive divider. Value 3: The fixed voltage (or fraction of total nominal voltage) applied to a stabilized first dynode. Use zero for a passive divider. Value 4: Total nominal voltage. If zero (or one), value 3 is assumed to represent the fraction of the total nominal voltage. Units: nanoseconds, nanoseconds, Volts or fraction, Volts.

- **TRANSIT_TIME_JITTER**
  (type: Double, items: 1, default: ".75", minimum: "0.0")
  Jitter of individual photo-electrons in nanoseconds (r.m.s. value). Unit: nanoseconds.

- **TRANSIT_TIME_ERROR**
  (type: Double, items: 1, default: "0.0", minimum: "-1.0")
  Extra errors (r.m.s.) in transit time, not related to high voltage. A value of -1 has a special interpretation (see text). Unit: nanoseconds.

- **TRANSIT_TIME_CALIB_ERROR**
  (type: Double, items: 1, default: "0.0", minimum: "0.0")
  Accuracy with which the actual signal delay due to transit time variation and corresponding compensation can be determined. Unit: nanoseconds.

- **TRANSIT_TIME_COMPENSATE_STEP**
  (type: Double, items: 1, default: "0.0", minimum: "0.0")
  If signal delays can be compensated independent of the sampling, this is the time step in which this compensation can be done. A value of zero means no compensation is applied. The same compensation is applied to all channels. Unit: nanoseconds.

- **TRANSIT_TIME_COMPENSATE_ERROR**
  (type: Double, items: 1, default: "0.0", minimum: "0.0")
  An extra error (r.m.s.) in how well the number of steps for transit time compensation can be determined before the compensation is applied. Unit: nanoseconds.

- **DEAD_PIXELS**
  (type: Int, items: MAX_DEAD_PIXELS, default: "all: -1")
  List of broken pixels. Value of '-1' terminates the list.

- **DEAD_PIXEL_PROBABILITY**
  (type: Double, items: 1, default: "0.", minimum: "0.0", maximum: "1.0")
  In addition to listed DEAD_PIXELS each pixel has a probability of being completely dead in all of its channels.

- **DEAD_BOARD_PROBABILITY**
  (type: Double, items: 1, default: "0.", minimum: "0.0", maximum: "1.0")
  In addition to listed DEAD_PIXELS and randomly dead pixels according to DEAD_PIXEL_PROBABILITY all pixels connected to the same board (or card) in the same module (or drawer) are set as dead according to a given probability.

- **DEAD_MODULE_PROBABILITY**
  (type: Double, items: 1, default: "0.", minimum: "0.0", maximum: "1.0")
  In addition to dead pixels and dead boards/cards (see DEAD_BOARD_PROBABILITY) all pixels in the same module (or drawer) are set as dead according to a given probability.

- **PM_PHOTOELECTRON_SPECTRUM**
  (type: Text, max. length: 4095, default: "hess_spe2.dat")
  File name for single p.e. response distribution. Usually not equidistant.

- **PM_SPE_TABLE_SIZE**
  (type: Int, items: 1, default: "10000", minimum: "1000", maximum: "20000")
  Size of look-up table for creating random numbers according to the given single p.e. spectrum (only used with the VERY_FAST_SPE compile-time option).

- **PHOTON_DELAY**
  (type: Double, items: 1, default: "0.0", units nanoseconds)
  An additional delay added to the arrival times of all photons at the photosensors. Unit: ns.

12.3.6 Additional afterpulsing

The usual photo-electron amplitude spectra include the afterpulsing case folded in in its third column. In order to be more flexible there is an alternate method to take afterpulsing into account, by separately simulating an exponential amplitude distribution for NSB photo-electrons. This requires activation of the ADDITIONAL_AFTERPULSING flag at compile-time.

- **AFTERPULSE_ALTERNATE**
  (type: Int, items: 1, default: "0", minimum: "0", maximum: "1" or "0")
  Flag to activate the alternate handling of afterpulse signals. The default

method is to have a convolved distribution of prompt and afterpulse signals of NSB photo-electrons. The alternate method (requires compilation with ADDITIONAL_AFTERPULSING defined for the pre-processor) uses the prompt amplitude distribution for the prompt part and additionally throws afterpulses following an exponential amplitude distribution.

- **AFTERPULSE_RATIO**
  (type: Double, items: 1, default: "0", minimum: "0", maximum: "1")
  The fraction of NSB photo-electrons resulting in an additional afterpulse of more than four (or any other value specified by `AFTERPULSE_THRESHOLD`) times the mean amplitude of a single photo-electron. Only applicable if `AFTERPULSE_ALTERNATE` is active.

- **AFTERPULSE_THRESHOLD**
  (type: Double, items: 1, default: "4.0", minimum: "0.0")
  The amplitude level (in units of the mean amplitude of a p.e.) above which the afterpulse ratio `AFTERPULSE_RATIO` is specified. Only applicable if `AFTERPULSE_ALTERNATE` is active.

- **AFTERPULSE_SCALE**
  (type: Double, items: 1, default: "7.5", minimum: "0.0", maximum: "50")
  The intensity scale (in units of the mean amplitude of a p.e.) of the exponential distribution of afterpulse amplitudes. Only applicable if `AFTERPULSE_ALTERNATE` is active.

- **AFTERPULSE_MAX**
  (type: Double, items: 1, default: "50", minimum: "10", maximum: "200")
  The maximum intensity of afterpulse signals (in units of the mean amplitude of a p.e.). Only applicable if `AFTERPULSE_ALTERNATE` is active.

12.3.7 Trigger

- **SIMPLE_THRESHOLD**
  (type: Double, items: 1, default: "4.0", minimum: "1", maximum: "10000")
  Simple threshold. Unit: p.e.s (used only without full simulation).

- **TRIGGER_CURRENT_LIMIT**
  (type: Double, items: 1, default: "20.0", units: microAmpére)
  Pixels above this limit are excluded from the trigger. [micro-Amperes]

- **TRIGGER_PIXELs**
  (type: Int, items: 1, default: "4")
  Number of pixels required for single telescope trigger. With flexible camera definitions, this is the default number for the multiplicity required per trigger group (i.e. 'sector' in the HESS phase 1 camera design). If the camera definition file contains

a non-zero number in the definition of any trigger group, that number overrides the default.

- **MULTIPLICITY_OFFSET**
  (type: Double, items: 1, default: "-0.5", minimum: "-0.5", maximum: "0.5")
  This number tells where the actual threshold of the telescope trigger is adjusted relative to the nominal number of required pixels. The default value of -0.5 means that with `TRIGGER_PIXELs` of 4, the sum of pixel discriminator/comparator outputs must exceed 3.5 times the average output amplitude for the given minimum time and by the given minimum signal integral in order to accept any 'sector' trigger (and therefore a telescope trigger) with the majority trigger logic.

- **TRIGGER_NEIGHBOURs**
  (type: Int, items: 1, default: "1")
  Obsolete. Number of neighboured pixels required for single telescope trigger. With flexible camera definitions, this number is not used. Obsolete.

- **TELTRIG_MIN_TIME**
  (type: Double, items: 1, default: "1.5", minimum: "0.", maximum: "10.", units: nanoseconds)
  Minimum time of sector trigger over threshold (flexible camera only) [ns].

- **TELTRIG_MIN_SIGSUM**
  (type: Double, items: 1, default: "7.8", minimum: "0.", maximum: "1000.", units: mV*ns)
  Minimum comparator outputs signal sum of sector trigger over threshold (flexible camera only) [mV times ns], if `DISCRIMINATOR_OUTPUT_AMPLITUDE` is in milliVolts. Note that both the minimum time over threshold and the minimum signal sum have to be satisfied before a trigger is obtained. Normally, either of these values being non-zero should be sufficient.

- **TRIGGER_TELESCOPES**
  (type: Int, items: 1, default: "2")
  Number of telescopes required for the system trigger. If an `ARRAY_TRIGGERs` file is specified, differing multiplicities can be specified there for particular telescope combinations.

- **IGNORE_TELESCOPEs**
  (type: Int, items: MAX_IGNORE, default: "all: -1")
  List of telescopes ignored (numbers starting at 1, i.e. CT 1, ...).

- **DEFAULT_TRIGger**
  (type: Text, max. length: 31, default: "Majority")
  The default meaning of a "Trigger" line in the camera definition file. If set to "Majority", a "Trigger" line would indicate a "MajorityTrigger", for "AnalogSum" it would

indicate an "AnalogSumTrigger", and for "DigitalSum" it would indicate a "DigitalSumTrigger".

- **ASUM_SHAPING_FILE**
  (type: Text, max. length: 4095, default: "none")
  The name of an optional file which describes the shaping (convolution) of an input PMT signal to the resulting signal from which an analog-sum trigger decision may be derived.

- **ASUM_CLIPping**
  (type: Double, default: "0.", minimum: "0.")
  The amplitude level (in the same units as assumed for DISCRIMINATOR_AMPLITUDE) at which the signal from each pixel (after optional shaping) is clipped for its contribution to the analog sum trigger. A value of zero indicates no clipping is applied.

- **ASUM_THRESHold**
  (type: Double, default: "0.", minimum: "0.")
  The amplitude level (in the same units as assumed for DISCRIMINATOR_AMPLITUDE) above which an analog sum is considered to result in a telescope trigger.

- **ASUM_OFFSET**
  (type: Double, default: "0.", minimum: "0.", units: nanoseconds)
  Offset in time where shaping convolution is done.

- **ASUM_SIGSUM_OVER_THRESHOLD**
  (type: Double, default: "0.", minimum: "0.")
  Not yet implemented. This is analogues to the DISCRIMINATOR_SIGSUM_OVER_THRESHOLD parameter but applied to the analog sum.

- **ASUM_HYSTERESIS**
  (type: Double, default: "0.", minimum: "0.")
  Not yet implemented. This is analogues to the DISCRIMINATOR_HYSTERESIS parameter but applied to the analog sum. Unit: same as for DISCRIMINATOR_AMPLITUDE.

- **ASUM_NOISE**
  (type: Double, default: "0.", minimum: "0.")
  Not yet implemented. This is white (Gaussian) noise applied to the analog sum, in the (depending on compilation definitions) typically four times finer time binning than used for ADC sampling. Unit: same as for DISCRIMINATOR_AMPLITUDE.

- **ASUM_SPECTRUM_NOISE**
  (type: Text, max. length: 4095, default: "none")
  Not yet implemented. If white noise is not an adequate description of the noise

spectrum after the analog sum, a file with a prepared long noise trace can be provided from which parts at random offsets will be added as noise. If ASUM_NOISE is non-zero, white noise will still be added. Unit: same as for DISCRIMINATOR_AMPLITUDE.

- **DSUM_PEDSUB**
  (type: Int, default: "1")
  If non-zero, the expected pedestal is first subtracted before any shaping, scaling, clipping etc. operations. Without pedestal subtraction, shaping kernels with non-zero sum are not practical.

- **DSUM_PRE_CLIPping**
  (type: Int, default: "0", minimum: "0")
  The amplitude level (in ADC counts above pedestal) at which the digitized signal from each pixel (before optional shaping) is clipped for its contribution to the digital sum trigger. A value of zero indicates no clipping is applied. Any such clipping is usually not a good idea, with FADC maximum value defined by FADC_MAX_SIGNAL anyway.

- **DSUM_SHAPING_FILE**
  (type: Text, max. length: 4095, default: "none")
  The name of an optional file which describes the shaping (convolution) of an digitized PMT signal (time step of ADC time slices) to the resulting signal from which a digital-sum trigger decision may be derived.

- **DSUM_SHAPING_RENORMalize**
  (type: Int, default: "1")
  If non-zero, the positive part of the shaping kernel is auto-normalized to a sum of 1.0; if zero, the shaping kernel is used as-is.

- **DSUM_OFFSET**
  (type: Double, default: "0.", minimum: "0.", units: nanoseconds)
  Offset in time where digital pulse shaping is done. Time intervals at the start and end of the simulated time window that are affected by shaping of missing outside signals are not used for trigger evaluation.

- **DSUM_IGNORE_BELOW**
  (type: Int, default: "0", minimum: "0")
  FADC signals (pedestal subtracted and/or shaped) below this value, i.e. in the noise, do not contribute to the digital signal sum and are set to zero. A value of zero means that no such lower threshold gets applied.

- **DSUM_ZERO_CLIP**
  (type: Int, default: "0", minimum: "-1", maximum: "1")
  With a value of 1 any negative shaped signals are clipped at zero (which a non-zero DSUM_IGNORE_BELOW does anyway). With a value of -1 negative signals are not

clipped immediately but together with patch-wise pre-summation they are clipped at zero after pre-summation. A value of zero means that negative shaped signals are preserved for the final digital sum.

- **DSUM_PRESCALE**
  (type: Int, size: 2, default: "0,0")
  Shaped signals are scaled by first multiplying with the first value (to integer unless `sim_telarray` was compiled with `-DDSUM_DOUBLE`) and then divided by the second value (discarding remainder again usually). No such scaling is applied if first and second value are equal.

- **DSUM_CLIPping**
  (type: Int, default: "0", minimum: "0")
  The amplitude level (in ADC counts above pedestal) at which the digitized signal from each pixel (after optional shaping) is clipped for its contribution to the digital sum trigger. A value of zero indicates no clipping is applied.

- **DSUM_PRESUM_SHIFT**
  (type: Int, default: "0", minimum: "0", maximum: "4")
  After a patch-wise pre-summation, the resulting sum may be right-shifted to reduce the significant number of bits. The presence of patches is indicated in the camera configuration file in the DigitalSumTrigger lines like
  `DigitalSumTrigger * of 1[2,3] 4[5,6]`
  instead of using a plain list of pixel IDs like
  `DigitalSumTrigger * of 1 2 3 4 5 6`

- **DSUM_PRESUM_MAX**
  (type: Int, default: "0", minimum: "0")
  After bit-shifting the pre-sum, the resulting value (zero-clipped, typically) may have the given maximum value to be represented in the available number of bits. A value of zero implies no maximum to be applied.

- **DSUM_THRESHold**
  (type: Int, default: "0", minimum: "0")
  The amplitude level (in ADC counts above pedestal sum) above which a digital sum is considered to result in a telescope trigger. *Note that, like for discriminator/comparator and analog sum, the signal must* **exceed** *('>') the threshold here before we declare the telescope triggered. The assigned threshold value would have to be one count lower than in a camera-internal trigger implementation (like FlashCam) where* **reaching** *('>=') the threshold is enough.*

- **TRIGGER_DELAY_compensation**
  (type: Double, "default: "all: 0.")
  Because the majority trigger, analog sum trigger, and digital sum trigger decisions result in different times of the trigger decision, this compensation is applied for array-level coindidences of triggers from telescopes with different trigger types.

- **LONG_EVENT_THRESHold**
  (type: Int, default: "0", minimum: "0")
  Some types of telescopes have conditions where a longer readout period than in normal events may be asked for. While the actual conditions which may force the 'long event' case still need to be worked out, this threshold on the number of trigger groups fired can be used to test the subsequent handling of the long events case. In the long event case the length of the readout trace is given by `FADC_LONGSUM_BINS` instead of `FADC_SUM_BINS` and the starting point as `FADC_LONGSUM_OFFSET` instead of `FADC_SUM_OFFSET` before the trigger time.

- **MUON_MONO_THRESHolds**
  (type: Int, items: 2, default: "0,0", minimum: "0")
  If the array trigger requires triggers from more than one telescope, the events best suitable for muon-ring calibration purposes, initiated by isolated muons, would be severely suppressed. Events with triggers in an extended area of the camera are a good indicator of muon rings and can be made to pass by stereo event selection. The first parameter is a threshold on the number of trigger groups fired, the second a threshold on the number of - yet to be determined - larger regions in which any trigger groups fired.

- **RANDOM_MONO_PROBability**
  (type: Double, default: "0.", minimum: "0.", maximum: "1.")
  A certain fraction of triggered telescope events can be made to randomly pass the stereo event selection in a way very similar to muon ring candidates do.

12.3.8 Electronics

- **NUM_GAINS**
  (type: Int, items: 1, default: "1" or "2", minimum: "1", maximum: "1" or "2")
  Tells through how many different gains the input signal gets digitized (one gain without `WITH_LOW_GAIN_CHANNEL` defined, two with). Allows to compile for two gains per pixel but have some telescopes with a single gain. Support for this parameter (added 2014-10-13) can be tested by the configuration file preprocessor, checking if and how `MAX_GAINS` is defined.

- **CHANNELS_PER_CHIP**
  (type: Int, items: 1, default: "4", minimum: "0")
  Tells how many channels as specified in the camera definition file are assigned to the same readout chip. This is potentially useful for crosstalk. It is used for the camera organisation output block.

- **DISCRIMINATOR_PULSE_SHAPE**
  (type: Text, default: "hess_disc_shape.dat")
  File name for pulse shape at the discriminator/comparator of an individual pixel.
  Note: You must make up your mind which file to choose here. Since the old

default pulse shape is now known to be definitely too slow, there is no default file available. See `cfg/hess/hess_disc_shape*.dat` for available files.

- **DISCRIMINATOR_AMPLITUDE**
  (type: Double, items: 1, default: "1.0", minimum: "0.0", units: typically mV or some kind of p.e.)
  Signal amplitude after amplifier per mean p.e. at the input of the discriminators/comparators. The unit is arbitrary but the same definition has to be used for DISCRIMINATOR_THRESHOLD, DISCRIMINATOR_VAR_THRESHOLD, DISCRIMINATOR_SIGSUM_OVER_THRESHOLD, DISCRIMINATOR_VAR_SIGSUM_OVER_THRESHOLD, and DISCRIMINATOR_HYSTERESIS with the majority trigger type and for ASUM_THRESHOLD, ASUM_CLIPping, ASUM_OFFSET etc. with the analog sum trigger type. For the default value of 1.0, the unit is the average (not the most probable) amplitude of a single photo-electron, implying the same being used for thresholds and switching behaviour.

- **DISCRIMINATOR_THRESHOLD**
  (type: Double, items: 1, default: "4.", minimum: "0.0")
  Discrim./compar. threshold. Unit: same as for DISCRIMINATOR_AMPLITUDE.

- **DISCRIMINATOR_VAR_THRESHOLD**
  (type: Double, items: 1, default: "0.2", minimum: "0.0")
  Channel to channel variations. Unit: same as for DISCRIMINATOR_AMPLITUDE.

- **DISCRIMINATOR_SCALE_THRESHOLD**
  (type: Double, items: 1, default: "1.0", minimum: "0.5", maximum: "2.0")
  Scales both the discriminator or comparator threshold and its variation by the same factor.

- **DISCRIMINATOR_NOISE**
  (type: Double, default: "0.", minimum: "0.")
  Not yet implemented. This is white (Gaussian) noise applied to the signal at the discriminator or comparator input, in the (depending on compilation definitions) typically four times finer time binning than used for ADC sampling. Unit: same as for DISCRIMINATOR_AMPLITUDE.

- **DISCRIMINATOR_SPECTRUM_NOISE**
  (type: Text, default: "none")
  Not yet implemented. If white noise is not an adequate description of the noise spectrum at the discriminator/comparator input, a file with a prepared long noise trace can be provided from which parts at random offsets will be added as noise. If DISCRIMINATOR_NOISE is non-zero, white noise will still be added. Unit: same as for DISCRIMINATOR_AMPLITUDE.

- **DISCRIMINATOR_GATE_LENGTH**
  (type: Double, items: 1, default: "2.0", minimum: "0.", maximum: "100")

Effective discr. gate length [ns]. To achieve a comparator-type response this gate length must match the time over threshold below. Other positive values represent a mix between comparator and discriminator (updating discriminator). For a true discriminator response (fixed gate length independent of time over threshold), use negative values. Unit: nanoseconds.

- **DISCRIMINATOR_VAR_GATE_LENGTH**
  (type: Double, items: 1, default: "0.1", minimum: "0.", maximum: "100")
  Variation of gate length [ns]. In compator-type response, this variable is not used but only DISCRIMINATOR_VAR_TIME_OVER_THRESHOLD. Unit: nanoseconds.

- **DISCRIMINATOR_TIME_OVER_THRESHOLD**
  (type: Double, items: 1, default: "1.5", minimum: "0.0", maximum: "100")
  Time over threshold required before logic response switches to true [ns]. To achieve a comparator-type response this time must match the gate length above. Note that in addition a minimum signal integral DISCRIMINATOR_SIGSUM_OVER_THRESHOLD may be set up. If so, both time over threshold and signal integral conditions have to be met before a 'true' output signal starts. Normally, either of them being non-zero should be sufficient. Unit: nanoseconds.

- **DISCRIMINATOR_VAR_TIME_OVER_THRESHOLD**
  (type: Double, items: 1, default: "0.1", minimum: "0.", maximum: "100")
  Pixel-to-pixel variation of it. Unit: nanoseconds.

- **DISCRIMINATOR_SIGSUM_OVER_THRESHOLD**
  (type: Double, items: 1, default: "0.0", minimum: "0.")
  Integrated signal required over threshold [mV*ns].
  See also DISCRIMINATOR_TIME_OVER_THRESHOLD above for combined effects. Unit: intended as milliVolts times nanoseconds, but if unit of DISCRIMINATOR_AMPLITUDE is not milliVolts, it scales accordingly.

- **DISCRIMINATOR_VAR_SIGSUM_OVER_THRESHOLD**
  (type: Double, items: 1, default: "0.0", minimum: "0.")
  Pixel-to-pixel variation of it. Unit: as for DISCRIMINATOR_SIGSUM_OVER_THRESHOLD.

- **DISCRIMINATOR_RISE_TIME**
  (type: Double, items: 1, default: "1.0", minimum: "0.0", maximum: "100.")
  Rise time of discr./comp. output [ns]. After the discriminator/comparator logical output is set true, the output signal linearly rises from 0 to 100% within the given time period. Unit: nanoseconds.

- **DISCRIMINATOR_FALL_TIME**
  (type: Double, items: 1, default: "1.0", minimum: "0.0", maximum: "100.")
  Fall time of discr./comp. output [ns] after the logical output is reset to false. Unit: nanoseconds.

- **DISCRIMINATOR_HYSTERESIS**
  (type: Double, items: 1, default: "0.0", minimum: "0.0")
  The switching off of a comparator is normally with some hysteresis to avoid oscillating behaviour. As a consequence, the signal has to be below the threshold minus the hysteresis before it switches off. Units: as for `DISCRIMINATOR_THRESHOLD` (nominally: [mV]).

- **DISCRIMINATOR_OUTPUT_AMPlitude**
  (type: Double, items: 1, default: "42.0", minimum: "0.0")
  The nominal output amplitude of a pixel discriminator or comparator as seen at the sector (trigger group) coincidence unit. Unit: Nominally in millVolts.

- **DISCRIMINATOR_OUTPUT_VAR_PERCENT**
  (type: Double, items: 1, default: "10.0", minimum: "0.0", maximum: "50.")
  The percentage channel to channel variation of the output amplitude of a pixel discriminator or comparator. [Percent]

- **DISC_AC_COUPLED**
  (type: Int, items: 1, default: "1", minimum: "0", maximum: "1")
  If set to 1, then discriminators/comparator are AC coupled and, except for noise, independent of NSB levels. Setting this to zero is generally not a good idea.

- **DISC_BINS**
  (type: Int, items: 1, default: "20", minimum: "1", maximum: "200")
  Number of time bins used for discr./comparator simulation.

- **DISC_START**
  (type: Int, items: 1, default: "0")
  No. of bins by which discr./comp. simulation is ahead of FADC. That is mainly relevant if different time windows are simulated for comparator inputs and digitised ADC values.

- **PIXELTRG_TIME_STEP**
  (type: Double, items: 1, default: "0.0", minimum: "0.0", units: nanoseconds)
  If non-zero, the time between the telescope trigger and the time when the pixel discriminator/comparator fired is recorded under the telescope event in the given time steps (negative for pixels fired before the telescope trigger; possible delays involved in a real instrument are not accounted for).

- **FAKE_TRIGGER**
  (type: Int, items: 1, default: "0", units: photoelectrons)
  Instead of using the signal-based trigger simulation for deciding if (and when) a telescope triggers, a highly simplified (or "fake") trigger decision can be based on the time-sorted list of MC true photo-electrons in the camera. A non-zero value of this parameter activates this scheme. The absolute value of this parameter is the threshold in units of true single p.e. A positive number sets the trigger time to the

arrival time of the specified p.e. in ascending time order. In other words, a value of 10 will require at least 10 p.e. for a trigger and the trigger time will be the arrival time of the 10th p.e. plus an offset. A negative number implies that the median arrival time of all photo-electrons is used instead; a value of -10 still implying a threshold of 10 p.e.

- **FORCE_FAKE_TRIGGER**
  (type: Int, items: 1, default: "0", minimum: "0", maximum: "1")
  The FAKE_TRIGGER parameter does not activate the fake trigger mechanism with a value of zero (which is the default). Fake triggers can be enforced with FORCE_FAKE_TRIGGER set to "1", even if there is no signal photo-electron in the camera. With FORCE_FAKE_TRIGGER=1 and FAKE_TRIGGER=0, the median time is used if there are any photo-electrons while an arbitrary time gets used without signal photo-electrons (NSB p.e. do not count for the fake-trigger scheme).

- **FAKE_DELAY**
  (type: Double, items: 1, default: "0.0", units: nanoseconds)
  An additional delay applied to the time of fake triggers (see FAKE_TRIGGER), in addition to that resulting from the optical configuration and the width of the simulated time window. A positive value will result in a later trigger time, thus shifting all signals to earlier times w.r.t. the trigger. This parameter applies only if FAKE_TRIGGER is non-zero.

12.3.9 (F)ADCs

- **FADC_PULSE_SHAPE**
  (type: Text, default: "hess_fadc_shape.dat")
  File name for (F)ADC pulse shape. These files are supported with two format variants:
  a) Implicit time steps, after a line starting with "# T=" plus the given time step (in nanoseconds) per data line. The high gain pulse shape is in column 1, low gain in column 2 if the camera has low-gain channels. It is recommended to no longer use this format variant.
  b) Explicit time steps in first column of the data rows, high-gain in column 2 and low gain (if applicable) in column 3.
  Note that the pulse amplitude scale is ignored and the pulses are rescaled to peak values of FADC_AMPlitude and FADC_LG_AMPlitude, respectively, times FADC_SENSITIVITY in both cases.

- **FADC_MHZ**
  (type: Double, items: 1, default: "1000", units: MHz or MSamples/s)
  FADC frequency. A value of 250 (MHz or million samples per second) corresponds to an FADC time interval of 4 ns.

- **FADC_BINS**
  (type: Int, items: 1, default: "20", minimum: "1", maximum: "200")
  No. of FADCs bins to be filled. In the first place this means the number of bins simulated. The median of all Cherenkov light photo-electrons will be near 40% of the interval length, unless shifted with `PHOTON_DELAY`. If the `FADC_SUM_BINS` is left at 0, this will also be used as the number of bins read out.

- **FADC_PER_CHANNEL**
  (type: Int, items: 1, default: "1", minimum: "1", maximum: "2")
  How many FADCs work in parallel with corresponding delays. This is 2 for HEGRA with two interleaved 60 MHz FADCs to achieve a 120 MHz system. For H.E.S.S. and all CTA camera types this is 1.

- **FADC_NOISE**
  (type: Double, items: 1, default: "4.0", minimum: "0", maximum: "10")
  Gaussian white noise per time bin in digitisation (for high gain channel, if different gains are used). Unit: ADC counts (r.m.s.).

- **FADC_LG_NOISE**
  (type: Double, items: 1, default: "1.3", minimum: "0", maximum: "10")
  Gaussian white noise in digitisation of low-gain channel. Unit: ADC counts (r.m.s.).

- **FADC_SPECTRUM_NOISE**
  (type: Text, default: "none")
  Not yet implemented. File name with prepared long noise trace.

- **FADC_SPECTRUM_LG_NOISE**
  (type: Text, default: "none")
  Not yet implemented. File name with prepared long noise trace for low-gain channel.

- **FADC_AMPlitude**
  (type: Double, items: 1, default: "14.0", minimum: "0.0")
  Amplitude at ADC/FADC (for high gain channel, if different gains are used). Unit: arbitrary but `FADC_AMPlitude` times `FADC_SENSITIVITY` are ADC counts maximum amplitude above pedestal (per time slice) for a photo-electron with average (not most probable) signal. This is after PMT, preamplifier, cable, and shaper (and, in case of H.E.S.S., the analog ring sampler) at the input of the ADC or FADC.

- **FADC_LG_AMPlitude**
  (type: Double, items: 1, default: "1.0", minimum: "0.0")
  The same for the low-gain channel.

- **HG_LG_VARiation**
  (type: Double, items: 1, default: "0.0", minimum: "0.0", maximum: "0.25")
  The relative pixel-to-pixel variation in the ratio of high-gain to low-gain amplitudes.

- **FADC_PEDESTAL**
  (type: Double, items: 1, default: "100.0", minimum: "0.0")
  Nominal (F)ADC pedestal value (per time slice). Unit: ADC counts.

- **FADC_DEV_PEDESTAL**
  (type: Double, items: 1, default: "1.0", minimum: "0.0")
  Deviation of (F)ADCs pedestals for same channel. This is only relevant if `FADC_PER_CHANNEL` is greater than 1.

- **FADC_VAR_PEDESTAL**
  (type: Double, items: 1, default: "0.75", minimum: "0.0")
  Channel-to-channel (or pixel-to-pixel) variation of the pedestal per FADC time slice.

- **FADC_ERR_PEDESTAL**
  (type: Double, items: 1, default: "0.08" /* 0.8/sqrt(100) */, minimum: "0.0")
  Assumed error in initial calibration of pedestal (reported as monitoring data).

- **FADC_SYSVAR_PEDESTAL**
  (type: Double, items: 1, default: "0.04", minimum: "0.0")
  Systematic common (e.g. due to temperature) variation of baselines. All reported monitoring data is offset by the same (random) amount.

- **FADC_LG_PEDESTAL**
  (type: Double, items: 1, default: "-1", minimum: "-2")
  Nominal (F)ADC pedestal value (per time slice) for low-gain channels. The default value of -1 indicates that the same value as for high-gain channels should be used. Unit: ADC counts.

- **FADC_LG_DEV_PEDESTAL**
  (type: Double, items: 1, default: "-1", minimum: "-2")
  Deviation of (F)ADCs pedestals for same channel. The default value of -1 indicates that the same value as for high-gain channels should be used. This is only relevant if `FADC_PER_CHANNEL` is greater than 1.

- **FADC_LG_VAR_PEDESTAL**
  (type: Double, items: 1, default: "-1", minimum: "-2")
  Channel-to-channel variation. The default value of -1 indicates that the same value as for high-gain channels should be used.

- **FADC_LG_ERR_PEDESTAL**
  (type: Double, items: 1, default: "-1", minimum: "-2")
  Assumed error in initial calibration of pedestal (reported as monitoring data). The default value of -1 indicates that the same value as for high-gain channels should be used.

- **FADC_LG_SYSVAR_PEDESTAL**
  (type: Double, items: 1, default: "-1", minimum: "-2")

Systematic (e.g. due to temperature) variation of baselines. All reported monitoring data is offset by the same (random) amount. The default value of -1 indicates that the same value as for high-gain channels should be used.

- **FADC_COMPENSATE_PEDESTAL**
  (type: Int, items: 1, default: "-1", minimum: "-1")
  The raw pedestal values may vary quite significantly from pixel to pixel but further data processing might be easier with more homogeneous values, after actual (F)ADC read-out. In a real camera, the pedestal compensation might be done in firmware (e.g. in an FPGA) or in software (in a camera server). Here it is emulated with the constraints that the resulting values are still unsigned integers and that no rescaling takes place (pure integer pedestal offset). No compensation takes place for the default value of "-1". Values greater or equal to zero indicate the compensated pedestal value, rounded to the nearest integer. For DC-coupled sensors, the compensation includes any NSB pedestal shift. For cameras with multiple interlaced FADCs per channel and, therefore, multiple pedestal values, the same integer compensation gets applied to all pedestals of a channel. While the normal pedestal reported in the data includes the compensation, the individual compensations applied for each pixel are reported separately. If individual FADC values would fall below zero through this compensation, they are clipped at zero.

- **FADC_LG_COMPENSATE_PEDESTAL**
  (type: Int, items: 1, default: "-1", minimum: "-1")
  In analogy to `FADC_COMPENSATE_PEDESTAL`, the compensated pedestal value for low-gain channels. A value of "-1" indicates that the behavior follows that defined for the high-gain channels.

- **FADC_ERR_COMPENSATE_PEDESTAL**
  (type: Double, items: 1, default: "0.", minimum: "0.")
  The pedestal compensation is not meant to be exact, even ignoring the limitation of rounding to integers, as the camera firmware or server would have to evaluate the actual pedestal from a finite number of samples/events. If pedestal compensation gets activated through `FADC_COMPENSATE_PEDESTAL` $\geq 0.$, this value indicates an r.m.s. error in the pedestal evaluation for the compensation step, not an error in the actual pedestal value.

- **FADC_LG_ERR_COMPENSATE_PEDESTAL**
  (type: Double, items: 1, default: "-1", minimum: "-1")
  Like `FADC_ERR_COMPENSATE_PEDESTAL` but for low-gain channels. A value of "-1" indicates that the behavior follows that defined for the high-gain channels.

- **FADC_AC_COUPLED**
  (type: Int, items: 1, default: "1", minimum: "0", "1") If set to 1, then FADCs are AC coupled. A change in night sky background rate will then only change the pedestal noise but not the average pedestal.

- **FADC_SENSITIVITY**
  (type: Double, items: 1, default: "1.0", minimum: "0.0")
  FADC counts per mV voltage (or whatever unit is used for FADC_AMPlitude). The definition of '1.0' as used for H.E.S.S. means that ADC amplitudes have to be given directly in units of the average amplitude of a single photo-electron.

- **FADC_LG_SENSITIVITY**
  (type: Double, items: 1, default: "-1", minimum: "-2")
  Sets the FADC separately for the low-gain channel, if that differs from high-gain. The default value of -1 indicates that the same value as for high-gain channels should be used.

- **FADC_VAR_SENSITIVITY**
  (type: Double, items: 1, default: "0.02", minimum: "0.0")
  Relative variations in sensitivity (even for FADCs of the same channel).

- **FADC_LG_VAR_SENSITIVITY**
  (type: Double, items: 1, default: "-1", minimum: "-2")
  Relative variations in FADC sensitivity for the low-gain channel, if that differs from high-gain. The default value of -1 indicates that the same r.m.s. value as for high-gain channels should be used. If both the sensitivity and its variation are set to less than zero (like default) or both sensitivity values match and the low-gain variation is set to -2, then the low-gain channel is forced to use the same sensitivity as the corresponding high-gain channel, to avoid increasing the high-gain/low-gain variation even further w.r.t. HG_LG_VARiation.

- **FADC_SUM_BINS**
  (type: Int, items: 1, default: "0", minimum: "0")
  Number of bins summed up in ADC sum data or read out in sampled data. This number corresponds to the experimental length of the readout window. The default value of 0 means that the full simulated interval of FADC_BINS bins is read out. Otherwise the start of the readout window starts FADC_SUM_OFFSET bins before the calculated time of the trigger, as long as the readout window fits fully in the simulated window.

- **FADC_LONGSUM_BINS**
  (type: Int, items: 1, default: "0", minimum: "0")
  Equivalent to FADC_SUM_BINS in case the event is selected to be a 'long event' for the readout of this camera. A value of "0" indicates that there should be no special handling of long events (both FADC_SUM_BINS and FADC_SUM_OFFSET continue to be used, even if some long event conditions are met).

- **FADC_SUM_OFFSET**
  (type: Int, items: 1, default: "0", minimum: "0")
  Number of bins before telescope trigger where summing starts. With peak sensing readout, the same interval is used for searching the peak signal.

- **FADC_LONGSUM_OFFSET**
  (type: Int, items: 1, default: "0", minimum: "0")
  Equivalent to `FADC_SUM_OFFSET` in case the event is selected to be a 'long event' for the readout of this camera. A `FADC_LONGSUM_BINS` value of "0" indicates that there should be no special handling of long events (both `FADC_SUM_BINS` and `FADC_SUM_OFFSET` continue to be used, even if some long event conditions are met).

- **FADC_MAX_SIGNAL**
  (type: UInt, items: 1, default: "0", minimum: "0", maximum: MAX_FADC_SIGNAL)
  The maximum value of the digitized signal per sample. For a typical 12-bit ADC this would be 4095.

- **FADC_MAX_SUM**
  (type: UInt, items: 1, default: "0", minimum: "0")
  The maximum value of a pulse sum produced by hardware pulse summation, in sum mode rather than recording pulse samples. Typical limitations are 15 or 16 bits, i.e. 32767 or 65535.

- **FADC_LG_MAX_SIGNAL**
  (type: Int, items: 1, default: "-1", minimum: "-2", maximum: MAX_FADC_SIGNAL)
  Allows to define a different maximum signal for a low-gain channel, if used. The default value of -1 indicates that the same maximum value as for the high-gain channel should be applied.

- **FADC_LG_MAX_SUM**
  (type: Int, items: 1, default: "-1", minimum: "-2")
  Allows to define a different maximum pulse sum for a low-gain channel, if used. The default value of -1 indicates that the same maximum value as for the high-gain channel should be applied.

12.3.10 Night-sky background

- **NIGHTSKY_BACKGROUND**
  (type: Double, items: MAX_PIXELS, default: "all: 0.100", minimum: "0.0", units: GHz, for photo-electrons per nanosecond)
  Number of photo-electrons per nanosecond due to nightsky background. There is one value per pixel, with optional range addressing prefixes ("all:" for all pixels, numeric addressing as lists or ranges in parentheses, pixel numbers starting at zero).
  Example for numerical addressing: `(0-1236): 0.2, (1237-1854):0.15`
  In case of overlapping ranges or if there are multiple lines addressing separate ranges, they are processed in the order given.
  More complex example in a single line: `nightsky_background =`

`all: 0.2, (1237-1854): 0.15, (870,951, 1380-1385): 0.35`
Corresponding multi-line example:

```
nightsky_background = all: 0.2
nightsky_background = (1237-1854): 0.15
nightsky_background = (870,951, 1380-1385): 0.35
```

Note that the numbers here include *all* photo-electrons, including also those not properly amplified or lost at the first dynode. Measured nightsky background rates may count, depending on the actual procedure, only those suriving the collection efficiency (i.e. some 15% less). The default value of 100 MHz pixel noise rate for H.E.S.S.-I is according to S. Preuss, away from Galactic plane and zodiacal light, for the design mirror reflectivity.

- **NSB_SKY_MAP**
  (type: Text, max. length: 4095, default: "none")
  An optional sky map (Az/Alt) of NSB enhancement factors, counted on top of the configured pixel NSB p.e. rates and other scaling factors mentioned in this section, but not any starlight which might get added through the file specified via the `STARS` parameter. Recommended format is explicit 3-D rpolator notation, with one set of Azimuth, Altitude, and NSB factor per line, forming a regular grid in Azimuth and Altitude. It gets evaluated for each pixel center pointed back into the sky, using the effective focal length (no explicit ray-tracing), thus is only suitable for changes in sky brightness exceeding the pixel sizes.

- **NSB_SCALING_FACTOR**
  (type: Double, items: 1, default: "1.0", minimum: "0.0", maximum: "1000")
  While the `NIGHTSKY_BACKGROUND` configuration can be used to set the NSB individually for each pixel of each telescope, it cannot be used easily for a common scaling of the NSB in all pixels of all telescopes against some reference setting. This can be achieved with the this parameter (added 2014-03-20).

- **NSB_AUTOSCALE_AIRMASS**
  (type: Double, items: 2, default: "0.0,0.15", minimum: "0.0", maximum: "1.0")
  Instead of the manual `NSB_SCALING_FACTOR` scaling factor, the NSB can also be scaled automatically depending on airmass and thus zenith angle. The first of the two parameters is the zenith-level fraction of NSB resulting from airglow, the second is the effective extinction coefficient applicable for NSB light (due to scattering it should be smaller than extinction coefficient for line-of-sight propagation). Realistic values are expected to be on the order of 0.7 and 0.15, respectively. This only takes effect if `NSB_SCALING_FACTOR` is exactly 1.0 (added 2018-01-08).

- **NSB_OFFAXIS**
  (type: Double, items: 5, default: "all: 0.0")
  The effective optical area can change across the field-of-view in a way partly simulated explicitely and partly covered by the function parameters in `TELESCOPE_TRANSmission`. The first parameter here is the function number

(like the second parameter in TELESCOPE_TRANSmission). Function zero is a constant (all following parameters ignored), function one is again similar to that in TELESCOPE_TRANSmission: $f = 1/(1 + p_1 * (r/p_2)^{p_3})$ if $p_4 = 0$ or missing and $f = 1/(1 + p_1 * (r/p_2)^{p_3})^{p_4}$ if $p_4 \neq 0$ where r is the (x/y projected) pixel off-axis radius in units of centimeters and p_1 to p_4 are the second to fifth parameter, with p_2 understood as a camera reference radius for the purpose of this function, again in units of centimeters, while the other parameters are dimensionless. Since the parameters are usually highly correlated when fitting them against actual shadowing (from detailed ray-tracing, effective mirror area as a function of off-axis radius over on-axis value), it seems good practice to start the fit with p_2 fixed to the actual camera radius, and only later try if a free p_2 improves the fit.

- **NSB_GAIN_DROP_SCALE**
  (type: Double, items: 1, default: "0.", minimum: "0.", units: GHz)
  This parameter, if non-zero, indicates the pixel p.e. rate at which the gain drops to half the nominal value due to bias resistance and cell capacitance (applicable as such for SiPM cameras). Units: GHz (p.e. per ns), same as for NIGHTSKY_BACKGROUND. Example: 1./(2.4 kOhm * 85 fF)/1e9 ns = 4.90 GHz. (Added 2017-08-30).

12.3.11 Calibration-specific

- **LASER_PHOTONS**
  (type: Double, items: MAX_LASER_LEVELS, default: "500", minimum: "1")
  Number of laser photons at each PM.

- **LASER_VAR_PHOTONS**
  (type: Double, items: MAX_LASER_LEVELS, default: "0.05", minimum: "0", maximum: "1")
  Relative variation of laser shots from shot to shot, indepenent for each telescope.

- **LASER_EVENTS**
  (type: Int, items: MAX_LASER_LEVELS, default: "0", minimum: "0")
  Laser (or LED or other pulsed light source) events at start of run, before the first shower event. The assumed light source would typically be in the center of the dish and is assumed to illuminate all pixels uniformly. The camera lid is assumed to be open, i.e. events will also be subject to NSB. A value of zero means that the data will not contain any such events. This type of calibration events can be simulated for multiple light levels, in combinations of the corresponding values of LASER_PHOTONS, LASER_VAR_PHOTONS and LASER_EVENTS.

- **LASER_PULSE_OFFSET**
  (type: Double, items: 1, default: "0.", units: nanoseconds)
  The flat-fielding and single-p.e. calibration units are assumed to operate with an

external trigger. Thus the position of the signal in the read-out window can be shifted (negative number: earlier, positive numbers: later). Unit: nanoseconds.

- **LASER_PULSE_EXPTime**
  (type: Double, items: 1, default: "0.", minimum: "0.", units: nanoseconds)
  To be used (non-zero) if the pulse shape of the light emitted from the flat-fielding or single-p.e. calibration unit follows an exponential, e.g. as used in early HESS calibration units with a UV laser illuminating a scintillator and exciting it to emit light with a decay time of 2.5 ns. Unit: nanoseconds.

- **LASER_PULSE_SIGTime**
  (type: Double, items: 1, default: "0.", minimum: "0.", units: nanoseconds)
  To be used (non-zero) if the pulse shape of the light emitted from the flat-fielding or single-p.e. calibration unit follows a Gaussian, with the given number standing for the sigma of the Gaussian. Unit: nanoseconds.

- **LASER_PULSE_TWIDth**
  (type: Double, items: 1, default: "0.", minimum: "0.", units: nanoseconds)
  To be used (non-zero) if the pulse shape of the light emitted from the flat-fielding or single-p.e. calibration unit follows a top-hat (Heavyside) function, with the given number standing for the full width. Note: You can chose from the available functions and you can even combine them if the values of more than one are non-zero. Unit: nanoseconds.

- **LASER_WAVELENGTH**
  (type: Double, items: 1, default: "400.", minimum: "200.", units: nanometer)
  The wavelength of the light emitted from the flat-fielding or single-p.e. calibration unit. Applies to both 'laser' and 'LED' type events. Unit: nanometer.

- **LASER_EXTERNAL_TRIGGER**
  (type: Int, items: 1, default: "0", minimum: "0", maximum: "1")
  A laser/LED flatfielding unit would typically operate at a sufficient illumination that the camera can trigger by itself. If operated at low illumination (e.g. a single-p.e. calibration unit outside of the camera), then an external trigger is needed.

- **LED_PHOTONS**
  (type: Double, items: 1, default: "4.0", minimum: "0.05")
  The intensity of LEDS assumed to sit in the camera lid in front of each pixel (like originally HESS CT3) when simulating extra calibration-type events (with `LED_EVENTS` greater than zero). Typically this would be single-p.e. calibration events without NSB background.

- **LED_VAR_PHOTONS**
  (type: Double, items: 1, default: "0.0", minimum: "0", maximum "1")
  Variation of the average amplitude of the LEDs in front of each pixel, from pixel to pixel (not including the statistical event-to-event fluctuation).

- **LED_EVENTS**
  (type: Int, items: 1, default: "0", minimum: "0")
  If greater than zero than additional calibration-type events are simulated before the first shower events, assuming an independent LED in front of each pixel, taking this type of data with closed lid, i.e. without NSB.

- **LED_PULSE_OFFSET**
  (type: Double, items: 1, default: "0.", units: nanoseconds)
  As for `LASER_PULSE_OFFSET` but for in-lid LED devices.

- **LED_PULSE_SIGTime**
  (type: Double, items: 1, default: "0.", minimum: "0.", units: nanoseconds)
  As for `LASER_PULSE_SIGTime` but for in-lid LED devices.

- **DARK_EVENTS**
  (type: Int, items: 1, default: "0", minimum: "0")
  Pedestal events at start of run with camera lid closed (completely dark, no NSB).

- **PEDESTAL_EVENTS**
  (type: Int, items: 1, default: "0", minimum: "0")
  Pedestal events at start of run with camera lid open (same NSB as for normal events).

12.3.12 Output data options

Configuration parameters in this section used to be global parameters but, since they apply to each telescope individually, have been changed to telescope-specific parameters.

- **OUTPUT_FORMAT**
  (type: Int, items: 1, default: "0")
  Select a specific output format (if several available). A value of '0' means to write ADC sums, a value of '1' to write ADC samples.

- **SAMPLED_OUTPUT**
  (type: Int, items: 1, default: "0")
  This is a synonym to the `OUTPUT_FORMAT` parameter, that should be easier to find when you are looking how to write sample-mode (FADC-like) output data.

- **PEAK_SENSING**
  (type: Int, items: 1, default: "0")
  Instead of sampling the pulses and optionally integrating signals over a given number of samples, the peak sensing option only reports the sample with the largest signal in the range otherwise used for ADC sums. The output is reported to be an ADC sum over one sample; also pedestals are for one sample only. This option is incompatible with sample-mode output.

- **ZERO_SUPpression**
  (type: Int, items: 1, default: "0", minimum: "0", maximum: "2")
  Zero-suppression mode. 0: no suppression, 1: bit maps mark significant pixels, 2: a list of significant pixels is included in the data.

- **DATA_REDuction**
  (type: Int, items: 1, default: "0", minimum: "0", maximum: "2")
  Data reduction mode. Only available in 'sum' format. 0: full raw data for every (significant) pixel, 1: low-gain channel suppressed if little signal in high-gain, 2: small high-gain values squeezed into 8 bits.

- **AUX_TRACES**
  (type: int, items: 8, default: "0,0,0,0,0,0,0,0", minimum: "0", maximum: "1")
  This parameter is dedicated to debugging and illustration of the camera trigger logic. It is only available if `sim_telarray` got compiled with `WITH_AUX_TRACES` defined. This compilation option comes, at run-time, with a significant impact on memory usage and output data size, and, to a lesser extend, also on CPU usage. Each integer value disables (0) or enables (1) a particular trace of raw (analog, float value) or digitized signals internal to the simulation. Digitized signals cover the time range of the FADC simulation, analog signals the time range of the discriminator/comparator simulation at the higher time-resolution used for that part of the simulation (typically four times as many samples per trace as with digitized signals). The first four switches represent digital traces, of which only the first is implemented. Availability of the four 'analog' traces depends on the trigger type(s) in a camera. The individual switches are for enabling/disabling:

  1 Digitized traces of simulated ADC values over the full `FADC_BINS` range, as scaled, shaped, clipped, added-up in each digital-sum trigger group, before the final decision. Limited to cameras with a digital-sum trigger.

  2-4 Reserved, not used.

  5 The raw signal at the input of each comparator/discriminator or the input stage of an analog sum. Limited to cameras with majority or analog-sum trigger type(s).

  6 The added-up signal for the analog-sum trigger groups, after shaping and clipping. Limited to cameras with analog-sum trigger.

  7 The discriminator or comparator output signal in percent of the nominal output. Limited to cameras with majority trigger.

  8 The majority signal in units of pixels. With logic-type majority (no rise and fall times), this will be integer values (represented as floats) while for a fully analog majority electronics, this will be the added-up outputs of the discriminator/comparator outputs in each trigger group. Limited to cameras with majority trigger.

12.3.13 Level of simulation detail

- **FULL_SIMULATION**
  (type: Int, items: 1, default: "2", minimum: "0", maximum: "2")
  Full simulation of FADC signals if > 0, of discriminators if > 1.

12.3.14 Pixel peak detection and timing

- **PULSE_ANALYSIS**
  (type: Int, items: 1, default: "0")
  Non-zero values enable the pixel pulse timing analysis, with a range of peak, rise, and fall times evaluated for significant pixels, plus pulse sums from SUM_BEFORE_PEAK bins before the peak bin to SUM_AFTER_PEAK bins after it. Low-gain channels are not analyzed separately, with pulse sums integrated over the same period as in the corresponding high-gain channel. The local peak pulse sums are always just available for significant pixels. Global peak sums, evaluated around the peak in the co-added pulse of all significant pixels, can be either provided for all pixels (value greater than zero) or only for significant pixels (value less than zero).

- **SUM_BEFORE_PEAK**
  (type: Int, items: 1, default: "-1", minimum: "-1")
  The number of time slices before the global (local) peak position where summation of the alternate charge sums start for significant pixels. A value of -1 indicates that no such sums are recorded.

- **SUM_AFTER_PEAK**
  (type: Int, items: 1, default: "-1", minimum: "-1")
  The number of time slices after the global (local) peak position where summation of the alternate charge sums ends for significant pixels. A value of -1 indicates that no such sums are recorded.

12.3.15 Appearance of image plots

- **IMAGE_PE_RANGE**
  (type: Double, items: 1, default: "30", minimum: "10", maximum: "1000")
  Range scale for image colours.

- **IMAGE_GAMMA_COEFFicient**
  (type: Double, items: 1, default: "0.85", minimum: "-5.", maximum: "5.")
  Contrast parameter.

12.4 For (meta-) information only

Although all relevant parameters describing a telescope and needed for the analysis of the simulated data is available in the data stream, a short-hand description of the configuration used is often useful. There are three areas where such descriptions can be provided: array level, optics, and camera. In each of those areas there can be a configuration name, a configuration version, and a configuration variant - for example activated by some run-time definitions. All of these fields are by default empty text strings. Array-level information will only be used from the global configuration, optics and camera information are telescope-specific. None of these parameters has any effect on the simulation itself. They are all pre-selected as meta-parameters, at the relevant level. Other parameters may be asigned as meta-parameters with the `METAPARam` configuration function. This covers both actual configuration parameters (values will be filled in as effective in the end) or made-up parameters (values must be assigned directly).

- **CONFIG_RELEASE**
  (type: Text, max. length: 99, default: "")
  If the configuration (data) files were released as part of a package, this parameter (with no effect otherwise) may indicate the release date/time/version/comments etc. Unless the corresponding global configuration file got included, this parameter is empty by default.

- **CONFIG_VERSION**
  (type: Text, max. length: 99, default: "")
  Usually an automatically generated text when the configuration was last modified.

- **ARRAY_CONFIG_NAME**
  (type: Text, max. length: 99, default: "")
  The basic name of the array configuration like “CTA PROD4 LaPalma baseline”.

- **ARRAY_CONFIG_VERSION**
  (type: Text, max. length: 99, default: "")
  If multiple versions of the array configuration of a given name have been in use this may indicate a specific date or version text to identify this specific version.

- **ARRAY_CONFIG_VARIANT**
  (type: Text, max. length: 99, default: "")
  If the same configuration file (of a specific version or in general) contains multiple array variants activated by definitions at run-time, this may identify the variant in use, e.g. “MSTs as NectarCam”.

- **OPTICS_CONFIG_NAME**
  (type: Text, max. length: 99, default: "")
  A name (text string) identifying the basic optics configuration of a specific telescope, e.g. “MST”.

- **OPTICS_CONFIG_VERSION**
  (type: Text, max. length: 99, default: "")
  If multiple versions of the optics configuration of a given name have been in use this may indicate a specific date or version text to identify this specific version.

- **OPTICS_CONFIG_VARIANT**
  (type: Text, max. length: 99, default: "")
  If the same configuration file (of a specific version or in general) contains multiple optics variants activated by definitions at run-time, this may identify the variant in use, e.g. "MST-86" for a MST with 86 mirror tiles (although that could also be derived from the mirror area reported in the data).

- **CAMERA_CONFIG_NAME**
  (type: Text, max. length: 99, default: "")
  A name (text string) identifying the basic camera configuration of a specific telescope, e.g. "FlashCam".

- **CAMERA_CONFIG_VERSION**
  (type: Text, max. length: 99, default: "")
  If multiple versions of the camera configuration of a given name have been in use this may indicate a specific date or version text to identify this specific version.

- **CAMERA_CONFIG_VARIANT**
  (type: Text, max. length: 99, default: "")
  If the same configuration file (of a specific version or in general) contains multiple camera variants activated by definitions at run-time, this may identify the variant in use, e.g. "Analogsum14" for a camera with analog sum trigger including 14 pixels (two modules of seven pixels).

- **METAPARam**
  (type: Function)
  Syntax:
  ```
  METAPARam [ scope ] action
  ```
  or more specifically, with supported scopes and actions:
  ```
  METAPARam [ GLOBAL | TELESCOPE | ANY ]
            { ADD list |
              SET name=value |
              REMOVE list |
              CLEAR }
  ```
  The scope selection 'GLOBAL' (for global parameters), 'TELESCOPE' (for telescope-specific parameters), or 'ANY' (or 'ALL', for either of the two) can be omitted, and defaults to 'TELESCOPE'. The 'ADD' action adds names of configured parameters, optionally with a comma-separated list of names. Any acceptable abbreviation of a parameter name will be accepted - and it will be recorded under its pre-configured name. Function-type parameters cannot be selected for that as

they have no assigned values. The 'ADD' action can also use a '*' wildcard for any known, non-function parameter or 'sectionname:*' for all parameters configured under the given name of a configuration section (for example: 'main', 'transmission', 'telescope'). The 'SET' action assigns manual meta-parameters with a value. To avoid confusion with configured parameters, they will be prefixed with a $\star$ and converted to uppercase. The 'REMOVE' action can remove meta-parameters of either type, of course without affecting the actual configuration. The 'CLEAR' action removes all meta-parameters (in the chosen scope).
Global and telescope-specific meta-parameters will be recorded as lists of name/value pairs in the output file, with a dedicated data block type (IO_TYPE_METAPARAM, 75).

12.5 Hconfig built-in functions

The configuration system has a number of built-in functions, accessible like ordinary configuration parameters, most of which can be used to obtain information about configured parameters. The optional values passed to these functions can be either a configuration parameter name or one of the special names 'all' (same as '*'), 'no-internal' (similar to 'all' but excluding these built-in functions), 'modified' (parameters that have been modified from the compiled-in defaults), 'locked' (parameters that can currently not be modified), and 'unlocked' (parameters free to be modified). Other built-in functions are useful to produce informative or diagnostic messages.

Some aspects of the output produced with these functions can be further controlled through some environment variables.

- **SHOW**
  (type: Function, for configuration parameter or special names)
  Show most of the user-accessible aspects of a configuration parameter, including name, type, length, limits, default and current value(s).

- **LIMITS**
  (type: Function, for configuration parameter or special names)
  Extract lower and upper limits for numeric parameter types.

- **INITLIST**
  (type: Function, for configuration parameter or special names)
  Extract initial (default) parameter values.

- **TYPELIST**
  (type: Function, for configuration parameter or special names)
  Extract type of parameter, number of values, and options. For each matching parameter it will show its configured type string, the number of values expected or allowed (-1 for function type, the number of characters without terminating NUL character for strings), and a string of optional characters which may include

  - R: Item is rejecting any attempts to set or use it.
  - H or S: Item has hard or strict boundaries. Any attempt to violate them will result in a configuration error.
  - l or L: There is a lower boundary value configured.
  - u or U: There is an upper boundary value configured.

- **LIST**
  (type: Function, for configuration parameter or special names)
  Extract current parameter values.

- **STATUS**
  (type: Function, for configuration parameter or special names)
  Shows locked/unlocked status.

- **LOCK**
  (type: Function, for configuration parameter or special names)
  Lock given parameters (making them immutable until a corresponding UNLOCK).

- **UNLOCK**
  (type: Function, for configuration parameter or special names)
  If a parameter was previously locked, it can be unlocked again in order to be able to change it. All parameters are initially unlocked.

- **ECHO**
  (type: Function, for arbitrary text) Informative text output (usually to stdout) from a configuration file.

- **WARNING**
  (type: Function, for arbitrary text) Diagnostic text output (usually to stderr) from a configuration file.

- **ERROR**
  (type: Function, for arbitrary text) Diagnostic text output (usually to stderr) from a configuration file. This also sets the configuration error condition which, for `sim_telarray`, means that it will abort after the scanning through the configuration.

The `LIMITS`, `INITLIST`, `TYPELIST`, and `LIST` functions also check a number of environment variables (some of which get set by `sim_telarray` itself if not defined beforehand):

- **HCONFIG_LIST_STYLE** controls delimiter and end-of-line for the following styles: `latex` (ready to be pasted into a LaTeX table, `plain` (showing '=' sign), `tsv` (for tab-separated-values, default).

- **HCONFIG_LIST_PREFIX** for a recognizable prefix to the produced output (to distinguish it from other `sim_telarray` output, among other things). `Sim_telarray` uses "(@cfg) " by default.
- **HCONFIG_INITLIST_PREFIX** for an additional prefix to identify output lines with default values. If none is given, `sim_telarray` will show `defaults`.
- **HCONFIG_TYPELIST_PREFIX** for an additional prefix to identify output lines with type listing. If none is given, `sim_telarray` will show `type`.
- **HCONFIG_LIMITS_PREFIX** for an additional prefix to identify output lines with numeric limits. If none is given, `sim_telarray` will show `limits`.
- **HCONFIG_SECTION_PREFIX** if set to any non-empty string not equal to "0" has the effect that the name of the section, in which the parameter was defined, will be added in square brackets.
- **HCONFIG_PREFIX_NUM** can be used to identify for which telescope the parameter applies with the `LIST` function. By default global parameters are identified as 'global' while parameters for individual telescopes are identified as 'CT' plus the telescope ID. Set it to "#" or "%d" or "num" for an entirely numeric representation of the telescope ID (with '0' instead of 'global'). `Sim_telarray` actually updates this environment variable with the corresponding string for each telescope.

Chapter 13

Miscellaneous environment variables

13.1 Pathnames for programs and data

- **HESSROOT** stands for the top-level directory of the source code tree for H.E.S.S. (not just for simulations) This is where `sim_hessarray` and `hessio` are located in that environment.
- **CTA_PATH** stands for the top-level directory of the source code tree in CORSIKA/`sim_telarray` installations for CTA, typically where the `build_all` is applied to the contents of the combined source code package (`corsika7.7_simtelarray.tar.gz` or such).
- **CORSIKA_PATH** stands for the location of the directory containing the CORSIKA executable(s) and their data files (typically a `run` sub-directory of the CORSIKA installation directory and referred to by a `corsika-run` symbolic link under `CTA_PATH`).
- **CORSIKA_WORKDIR** is the location of the CORSIKA working directory, usually set up by `corsika_autoinputs` and filled with symbolic links to files in `CORSIKA_PATH` before starting CORSIKA.
- **SIM_TELARRAY_PATH** stands for the location of the `sim_telarray` source code, typically a `sim_telarray` sub-directory of `CTA_PATH` (in the H.E.S.S. world a `sim_hessarray` sub-directory of `HESSROOT`).
- **SIM_TELARRAY_RUN_PATH** may replace the run-time location of `sim_telarray` and its tools – although not all components use it. Falls back to `SIM_TELARRAY_PATH`.
- **SIM_TELARRAY_WORKDIR** would be the location where `sim_telarray` is actually run (from `run_multipipe`, via `multipipe_corsika` and then `generic_run`, `custom_run`, or an equivalent script, typically). Falls back to `SIM_TELARRAY_RUN_PATH` and `SIM_TELARRAY_PATH`. If the variable is defined but the directory does not exist, it gets created and filled with symbolic links

for barebone operation. Sim_telarray and its launching scripts will not look at the variable since the working directory is already chosen at that point.

- **HESSIO_PATH** stands for the location of the hessio source code, typically a hessioxxx sub-directory of CTA_PATH (in the H.E.S.S. world a hessio sub-directory of HESSROOT).

- **STDTOOLS_PATH** stands for the location of the stdtools source code, typically a stdtools sub-directory of CTA_PATH or HESSROOT.

- **IACTIO_PATH** stands for the location of the iactio source code, typically a iactio sub-directory of CTA_PATH or HESSROOT.

- **MCDATA_PATH** is where the CORSIKA and sim_telarray programs are supposed to write their output data, typically a Data sub-directory or symbolic link under CTA_PATH, with corsika and sim_telarray sub-directories.

- **CTA_DATA** is an alternative variable for indicating the location of the output data, in case MCDATA_PATH is not set. If neither is set, the Data sub-directory or symlink under CTA_PATH would be chosen.

- **HESSMCDATA** is an alternative variable for indicating the location of the output data, in case MCDATA_PATH is not set (in the H.E.S.S. world).

- **CORSIKA_DATA** is the location for CORSIKA output data, typically the corsika sub-directory of MCDATA_PATH, under which the run-specific working directories are created in a run%06d format. Before starting CORSIKA, the working directory is typically populated with symbolic links to the CORSIKA program and its data files (which normally live under CORSIKA_PATH. The pre-processed and corsika_autoinputs updated inputs file (with unique run number and random seeds) as well as the CORSIKA log file can be found in the working directory. Both CORSIKA itself and the IACT interface may create temporary data files in the working directory.

- **SIM_TELARRAY_DATA** is where sim_telarray data files are created, typically a sim_telarray sub-directory of MCDATA_PATH. The actual data files may be found under off-axis angle specific sub-sub-directories in a configuration-specific sub-directory.

- **SIM_TELARRAY_OUTPUT_PATH** can be used for control over the exact location of the sim_telarray output, rather than the configuration and offset dependent path.

- **SIMTEL_CONFIG_PREPROCESSOR** is an optional variable which can be set to replace the pfp pre-processor by another program.

- **SIMTEL_CONFIG_PATH** indicates extra location(s) of configuration files (colon-separated if multiple directories) which are searched before the paths compiled into `sim_telarray`.

- **SIM_TELARRAY_CONFIG_PATH** indicates the location(s) of configuration files (colon-separated if multiple directories) which are to replace the paths compiled into `sim_telarray`.

13.2 Variables used for configuration control

These variables are used in different parts of the simulation pipeline to adjust configuration files to be used etc.

- **extra_config** and command-line parameters to the `generic_run.sh` or `custom_run.sh` script are the last way to override parameters and definitions, superceding anything set in `SIM_TELARRAY_DEFINES`, `SIM_TELARRAY_INCLUDES`, `simtel_defs`, or `extra_defs`. In particular, it can still set the configuration file to be used.

- **extra_defs** is for further `sim_telarray` options, processed after those from `SIM_TELARRAY_DEFINES` and `SIM_TELARRAY_INCLUDES`, with the `generic_run.sh` or `custom_run.sh` script.

- **simtel_defs** Like `extra_defs` and a bit clearer in its name but so far rarely used. The `generic_run.sh` and `custom_run.sh` will try to use any of them (or all, if more than one is set), in the order `SIM_TELARRAY_DEFINES`, `SIM_TELARRAY_INCLUDES`, `simtel_defs`, `extra_defs` (the latter being able to override settings from the earlier ones).

- **extra_suffix** will get attached to the `sim_telarray` output file names by `generic_run.sh` or `custom_run.sh`; usually set in the `multipipe_corsika` configuration, with a different suffix for each of multiple pipes.

- **extra_suffix2** will also get attached to the `sim_telarray` output file names. This part would usually be set in the script starting CORSIKA, if different sets of simulations should be easily distinguishable by file name.

- **CORSIKA_MULTIPLEX_SEQUENTIAL** is set if the 'sequential' method of scheduling the `multipipe_corsika` pipes is to be activated. Equivalent to using the `sequential` option for `multipipe_corsika`.

- **CORSIKA_IO_BUFFER** sets an upper limit to the I/O buffer with the IACT eventIO output file or pipe. Sizes may have "MB" or "GB" suffixes etc.

- **CORSIKA_MAX_BUNCHES** can be set to adapt the limit on the number of photon bunches in a single telescope.

- **INCLUDES_PATH** sets search paths for included files, for example in the LightEmission applications.

- **SIM_TELARRAY_DEFINES** can be used to indicate extra `sim_telarray` options, intended for extra `-D` or `-U` pre-processor definitions.

- **SIM_TELARRAY_INCLUDES** can be used to indicate extra `sim_telarray` options, intended for extra `-I` pre-processor include paths.

- **SIMTEL_MULTI_CFG_SUFFIX** indicates an optional modifier to the `multipipe_corsika` configuration name, appended after the normal name.

13.3 Variables used by EventIO

Environment variables controlling and limiting the size of the buffers used by the EventIO code are either specified directly in bytes or come in units of powers of 1000 (k/M/G/T or kB/MB/GB/TB suffix) or powers of 1024 (kiB/MiB/GiB/TiB suffix). The suffices are case-insensitive ('Kb' or 'mIb' will work just as well - there are no 'milli' bytes). Floating-point numbers will usually work as well.

- **HDATA_IO_BUFFER**: Where reading or writing histograms creates its own temporary I/O buffer, this variable can be used to set the initial size of the I/O buffer, avoiding later re-allocations, for exaxmple. Not applying where histogram reading or writing is given a pre-allocated I/O buffer.

- **MAX_IO_BUFFER**: The buffer used for EventIO data blocks can grow up to a given limit. This variable cannot only control this limit from user-level scripts but it can also get CORSIKA, `multipipe_corsika`, and `sim_telarray` to use the same limits, for example with a `IACT MAX_IO_BUFFER` line in the CORSIKA inputs file.

- **MIN_IO_BUFFER**: The initial size of any I/O buffer specified as less than that will be increased accordingly. A hard-coded minimum always applies, independent of this variable (but is quite small and the buffer would need a number of re-allocations to grow to the size needed by the application).

- **NEW_IO_BUFFER**: If an application asked for an I/O buffer of unspecified initial size (size zero), either this variable applies or, in its absence, a hard-coded default. Most applications ask for a size more appropriate for the typical use in that application, and this variable rarely applies.

13.4 Variables overriding `sim_telarray` limits

- **MAX_BUNCHES**: Photon bunches for a single telescope are retrieved from the data block and buffered in an internal array, after deciding that ray-tracing is worth the effort and before going over the actual ray-tracing.
- **MAX_PHOTOELECTRONS**: After ray-tracing, all detected photo-electrons are kept in memory while waiting for the decision if simulation of the signals is worth the effort.
- **MAX_PIXEL_PHOTOELECTRONS**: If there are too many photo-electrons to a pixel, and signals are totally saturated, processing even more of them may not be worth the effort.
- **MAX_PRINT_ARRAY**: That is actually not just a limit for `sim_telarray` but for many programs sharing library when 'printing' arrays of data. After reaching the limit in members, more data is just implied by an ellipses (...).

13.5 Variables for configuring `multipipe_corsika`

- **MULTIPIPE_SEQUENTIAL** is set if the 'sequential' method is effective.
- **MULTIPIPE_DEBUGCFG** can be set to debug the configuration processed by `multipipe_corsika`, rather than processing it.
- **SIMTEL_MULTI_CFG** is an alternative to the `-c` option for setting the configuration name to be used with `multipipe_corsika`.
- **SIMTEL_MULTI_CFG_PATH** is where `multipipe_corsika` config files are supposed to be located. Falls back to the `multi` sub-directory in `SIM_TELARRAY_PATH` or `SIM_TELARRAY_RUN_PATH`.
- **SIMTEL_MULTI_CFG_FILE** sets a specific file to used as the configuration file for `multipipe_corsika`.
- **SIMTEL_MULTI_CFG_SUFFIX** gets appended to the `multipipe_corsika` configuration name, for example to modify the resulting configuration filename without having to modify the CORSIKA inputs file. This gets ignored if `SIMTEL_MULTI_CFG_FILE` is set.
- **SIMTEL_MULTI_CFG_PREPROC** is an alternate way to enable pre-processing the `multipipe_corsika` configuration file (or having it processed by bash/python/...). If `run_multipipe` was called with `--pfp`, `-p`, or `--preproc` options, the variable is superceded accordingly.
- **SIMTEL_MULTI_OPTION** adds further options on the `multipipe_corsika` command line (typically `-D` pre-processor definitions).

13.6 Variables set up by `multipipe_corsika`

These are variables get set up by `multipipe_corsika` based on the run header, the list of telescope positions, and first event header in the data stream. They are then available to `generic_run.sh` or `custom_run.sh` (or any other script used to steer the telescope simulation as the next step in the pipeline). The same variables may also be obtained with the `extract_corsika_tel` and `extract_corsika_multi` programs (`--header-only` or `--header-only-export` option) from an existing CORSIKA/IACT data file. Note that the number of telescopes and radii reported with these tools may depend on the selection of telescopes, and additional variables separately reporting these numbers before and after selection may be created.

- **CORSIKA_ARRAYS**: Number of uses of each CORSIKA by randomly offset telescope/detector arrays (from `CSCAT` input line).
- **CORSIKA_ATMOFILE**: Named atmospheric density table, if `IACT ATMOFILE` control card was used in CORSIKA inputs.
- **CORSIKA_ATMOSPHERE**: Atmospheric density table number, if if `IACT ATMOSPHERE` or simply `ATMOSPHERE` was used in CORSIKA inputs (99 if `IACT ATMOFILE` was used instead; 0 for all CORSIKA 5-layer variants either built-in or set-up by various CORSIKA parameters).
- **CORSIKA_CONE**: `VIEWCONE` maximum code angle in degrees.
- **CORSIKA_EMAX**: Maximum energy for primary particles (unit: TeV). See the `ERANGE` input line but note the different units used.
- **CORSIKA_EMIN**: Minimum energy for primary particles (unit: TeV). See the `ERANGE` input line but note the different units used.
- **CORSIKA_ESLOPE**: Spectral index of the generated power law spectrum. See the `ESLOPE` input line.
- **CORSIKA_NUMRAD**: The number of distinct radii of telescope spheres, which may or may not indicate the number of telescope types.
- **CORSIKA_COUNTRAD**: A comma-separate count of how often each distinct radius was used.
- **CORSIKA_LISTRAD**: A comma-separate list of the distinct radii (unit: cm).
- **CORSIKA_OBSLEV**: Altitude of the (lowest) observation level (unit: meters a.s.l.) See the `OBSLEV` input line (unit: centimeters).
- **CORSIKA_PHI**: Azimuth angle as counted by astronomers (geographic North towards East for the arrival direction; unit: degrees). This differs from the `PHIP` input in the origin, the orientation in which it is counted (and the number of values).

- **CORSIKA_PRIMARY**: Primary particle ID as used by CORSIKA through the `PRMPAR` input line.
- **CORSIKA_RUN**: Run number, as in `RUNNR` input card
- **CORSIKA_SHOWERS**: Number of showers to be generated, as in the `NSHOW` input card.
- **CORSIKA_TELESCOPES**: Number of telescope positions set up for each telescope array, with a separate `TELESCOPE` input card for each of them
- **CORSIKA_THETA**: Zenith angle of primary particle arrival direction (matching mean `THETAP`, unit: degrees).
- **CORSIKA_VERSION**: CORSIKA version used to generate the data.

These variables can also be retrieved from a disk file with the `--header-only` or `--header-only-export` options of `extract_corsika_tel`. Example usage:

```
extract_corsika_tel --header-only-export \
   fname.corsika.zst > corsenv.sh
source corsenv.sh
```

In addition, `multipipe_corsika` uses the variables **CORSIKA_PID**, **MULTIPIPE_PID**, **MULTIPIPE_SIGNAL**, and **MULTIPIPE_SIGNAL_INIT** for communication between `multipipe_corsika`, CORSIKA, and `sim_telarray` in sequential mode (**MULTIPIPE_SEQUENTIAL**=1). Those should never be set by the user.

Chapter 14

Source code documentation

This part is available as a separate file [6] and is produced automatically with the `doxygen` utility. The `hessio` package, which is an important part of the CORSIKA IACT option and the `sim_telarray` program, has a similarly generated documentation [8] of its own. A similarly generated file describes the IACT/ATMO package [7].

Chapter 15

Data format of output from the CORSIKA IACT interface

15.1 A machine-independent hierarchical data format

The data from the CORSIKA program with activated IACT interface is written with the `eventio` package [4] which was already used for the CRT detector and HEGRA telescope system data. For a general description of this package see also the CRT documentation. Among the features of the `eventio` package (in complete typically found under the name `hessioxxx.tar.gz` but most relevant parts being included with the IACT/ATMO package) are

machine independence Both big-endian and little-endian byte order is supported. Floating point representation is IEEE and conversion of VAX and general internal floating point representations to and from IEEE is included. At present, the package should work on any system with an C compiler and run-time library compatible with ISO C (1998 or newer), like gcc or clang. Although its main use these days is mainly under Linux, it is known to work under MacOS. The package got tested under Ultrix (MIPS CPU), DEC Unix (cc and gcc), OS-9 (68040 CPU), Lynx OS (68040 and PowerPC) and also (although in older implementations) even on VAX/VMS, AIX, and MS-DOS – in the days when these operating system were still around. Since both byte-orders are equally supported, there is no need for forward and backward byte order reversal when the writing and reading is usually done on machines of the same byte order. The byte order for writing can be selected at run-time (and could, in principle at least, be changed from one block to the next).

hierarchical structure A hierarchical data structure is supported where one *item* may contain either several *sub-items* or *atomic data* (bytes, integers, floating point numbers). As long as sub-items are not mixed with atomic data at the same level, the structure of each data block can be listed without knowing anything about the data format (except that it is `eventio` format). For such listings the *listio* program is available from the CRT source code collection.

transmission failure recovery Even in the case that errant data is transmitted or recorded, the beginning of the next data block is found by looking for a four-byte marker sequence (which also serves to tell the byte order of the recorded data).

file compression Data (to be) stored in files can be compressed / decompressed automatically with a wide range of standard compression tools like gzip, bzip2, xz, zstd, etc., with the tool selection just based on the file name extension.

15.2 Tools for CORSIKA eventio data

15.2.1 General eventio data tools

Independent of the contents of `eventio` data, the hierarchical structure is accessible with tools like `listio` and `statio`.

The `listio` program displays the type number, ident number, and length of each top-level data block and, if run with the '-s' option, also the full hierarchy of sub-blocks. If a file with a text description of the different data types exists as either `EventioRegisteredNames.dat` in the current directory or as `.EventioRegisteredNames` in the HOME directory, with lines like

```
100:Histogram:One or many histograms (1-D or 2-D)
```

the option '-n' will also show the short type text for registered types, `[Histograms]` for type 100 as an example, while the option '-d' will show both the short text and the longer description.

The `statio` programs counts all the different types of top-level data blocks encountered and displays the statistics of block types after encountering end-of-file, including the registered type names. For a short example CORSIKA output it may look like

```
Type    Blocks  Bytes       Version(s)  Name
1200    1       1112            0       [CORSIKA run header]
1201    1       100             0       [CORSIKA telescope positions]
1202    100     111200          0       [CORSIKA event header]
1203    100     10400           0       [CORSIKA telescope offsets]
1204    1000    70340880        0       [CORSIKA telescope array]
1209    100     111200          0       [CORSIKA event end]
1210    1       32              0       [CORSIKA run end]
1211    100     158000          0       [CORSIKA shower profile]
1212    1       5172            0       [CORSIKA inputs]
```

The `filterio` program is a tool to select or discard specific data block types when copying data from input to output, for example extracting only the CORSIKA inputs block from a data file.

15.2.2 The `multipipe_corsika` tool for CORSIKA IACT data

Although IACT output from CORSIKA can be recorded as plain files and processsed later-on, a frequent scenario is to pipe the IACT into `sim_telarray` without bothering with disk I/O. With the `multipipe_corsika` tool this can be done in a convenient and efficient way, for single use or multiple use of the CORSIKA IACT data. `Multipipe_corsika` reads CORSIKA IACT data from standard input. It is part of the `sim_telarray` package. From the first few data block it will then be used set up a number of environment variable that are useful for further processing without having to look into the data again. A configuration file tells what to do with the data, and each non-empty, non-comment line in the configuration is used to open an output pipe to which the input data will be copied. Traditionally, these were single physical lines but `multipipe_corsika` has been enhanced to continue with the next line, as long as the line ends in either backslash-backslash (`\\`, TEX style) or blank-backslash (`\`, shell/Makefile style), not counting any trailing whitespace. Example:

```
line1
line2a\\
line2b \
line2c \\
line2d
line3
```

would be equivalent to

```
line1
line2aline2bline2c line2d
line3
```

Note the blank before the single backslash being removed while the blank before the double backslash is not.

If some of the output pipes finish (or fail) early they are dropped from the list but output will continue to the still active ones, as long as at least one output pipe remains active. This tool can reduce the I/O needed for simulations by a large factor as the big amount of CORSIKA data may never be written to disk but immediately enters into one or several telescope simulations. The output pipe processes would normally all run in parallel, after having read a block of data, but `multipipe_corsika` can be set to have CORSIKA, `multipipe_corsika` and any `sim_telarray` in an output pipe to process data in sequential order, i.e. apart from passing the data from one program to the other only one program will be active at a time, avoiding the sum of CPU loads significantly exceeding 100% at any time.

The `multipipe_corsika` tool generally needs an instruction on the configuration file to use. It would normally obtain data from standard input (which may be an IACT output pipe from CORSIKA, via `TELFIL |cmd`, or any other pipe providing uncompressed IACT module data in eventIO format) or open a file by name (with automatic handling of uncompressing the data if the filename extension indicates that):

```
Syntax: multipipe_corsika [ options ] [ input-file ]
Options:
   -c cfgfile       (Use the given configuration file.)
   --preproc cmd    (Use given command as pre-processor or interpreter for the
                     cfgfile, with its standard output used as configuration.)
   --pfp            (Use 'pfp' as a pre-processor for the config file.
                     Note the use of '*' as control character instead of '#'.)
   -I* / -D* / -U*  (Pass include/define/undefine options to pre-processor.)
   --postproc cmd   (Add a command to be run after output pipes are all done.
                     Can be used multiple times, in addition to 'postproc:'
                     lines in the configuration file.)
   --sequential     (Run output pipes and input pipe in sequential order,
                     as far as possible. Only effective with sim_telarray
                     or other co-operating programs in output pipe.
                     Non-co-operating programs are unaffected and remain
                     running in parallel with any other programs.)
```

The configuration file may contain any number of commands that will be opened as output pipes. Usually the immediate command will be 'env' to allow to set various environment variables, in addition to those set by multipipe_corsika itself, before starting the 'real' command (typically the script 'generic_run.sh' for setting up the sim_telarray command line from all the relevant environment variables and parameters). Example (abbreviated with '...' ellipsis for omitted parts):

```
env offset="0.0" cfg=... ... ./generic_run.sh
```

Output commands where a premature ending does not necessarily indicate an error, should better be tagged as optional:

```
optional: env offset="1.0" cfg=... ... ./generic_run.sh
```

In addition to output pipes, the configuration file may also specify post-processing commands by the 'postproc:' tag, to be executed after all output pipes are closed and after any post-processing commands requested on the multipipe_corsika command-line. The post-processing commands will not get any data on their standard input but would need to know where to get their data. They are executed in the order specified. An example application would be writing temporary compressed IACT output data and running the telescope simulation(s) only after CORSIKA is done processing:

```
zstd > run${CORSIKA_RUN}.corsika.zst
postproc: env offset="0.0" cfg=... ... ./generic_run.sh run${CORSIKA_RUN}.corsik
postproc: env offset="1.0" cfg=... ... ./generic_run.sh \\
                                run${CORSIKA_RUN}.corsika.zst
postproc: rm run${CORSIKA_RUN}.corsika.zst
```

Instead of using a configuration file directly for setting up `multipipe_corsika`, it can also be pre-processed or interpreted by a chosen command, including the `pfp` preprocessor or any shell or programming language interpreter, including 'bash' or 'python'. With the added level of complexity it may become rather useful to check the resulting configuration before actually processing it – which can be achieved by setting the `MULTIPIPE_DEBUGCFG` environment variable.

First debugging example: plain config file with post-processing both in the configuration file and as `multipipe_corsika` options:

```
echo Hello1
optional: echo Hello2
postproc: echo Hello4
```

and tested with the command

```
MULTIPIPE_DEBUGCFG=1 bin/multipipe_corsika \
   --postproc 'echo Hello3a' --postproc 'echo Hello3b' \
   -c test1.cfg dummy1.corsika.gz
```

shows the following debugging output:

```
Adding post-processing command: echo Hello3a
Adding post-processing command: echo Hello3b
Input is read from 'dummy1.corsika.gz'
CORSIKA_VERSION=6.032
...
config line = echo Hello1
config line = optional: echo Hello2
Adding post-processing command: echo Hello4
```

Note that the post-processing commands configureed via `--postproc'` options are run before any configured via the `postproc:` tag in the configuration. Running it, without setting `MULTIPIPE_DEBUGCFG`=1, shows the actual processing order (with the first two run in parallel, so order of output may vary):

```
...
Hello2
Hello1
...
Output pipes finished after 0.018 seconds run-time.
Starting post-processing now ...
Hello3a
Hello3b
Hello4
Multipipe_corsika finished after 0.022 seconds run-time.
```

As a second example, we use the file

```
*ifdef ABC
echo "Hello1 (alternate)"
optional: echo "Hello2 (alternate)"
*else
echo "Hello1 (default)"
optional: echo "Hello2 (default)"
*endif
```

and pre-process it with `pfp`, using the (debugging-only) command

```
MULTIPIPE_DEBUGCFG=1 bin/multipipe_corsika --pfp -DABC \
   -c test2.cfg dummy1.corsika.gz
```

to select the alternate of the two configuration variants. While `pfp` usually has (like the C/C++ pre-processor) the '`#`' (hash) character as the control character (starting a condition/define etc.), this character is already used as a comment character. Assigning the '`*`' (asterisk) character to function as the control character here, keeps existing configuration files in working order when interpreted first through `pfp` rather than read directly. Actually `pfp` is called as `pfp -v -C* -I. -Imulti` (plus all the `-I`, `-D`, and `-U` options from the `multipipe_corsika` command line). That might be useful to know for testing a configuration file.

A perhaps more practical example uses the shell interpreter on the file

```
#!/bin/bash

offlist="0.0,0.25,0.5,0.7,1.0"

common='cfg=phase2d  extra_config="-c hess-phase2d.cfg" \
  extra_suffix="-nsb1.00" \
  extra_defs="${extra_defs} -C nsb_scaling_factor=1.00" \
  extension="zst" ./generic_run.sh'

for o in $(echo $offlist | sed 's/,/ /g'); do
   if [ "$o" = "0" -o "$o" = "0." -o "$o" = "0.0" ]; then
      echo "env offset=\"0.0\" $common"
   else
      echo "optional: env offset=\"$o\" $common"
   fi
done
```

with the (debugging-only) command

```
MULTIPIPE_DEBUGCFG=1 bin/multipipe_corsika --preproc /bin/bash \
   -c test3.cfg dummy1.corsika.gz
```

15.2.3 Other specialized tools for CORSIKA IACT output data files

The extract_corsika_tel and extract_corsika_multi programs allow to extract data for subsets of telescopes for further processing. They are part of the hessio package. While the former only has one input and one output, the latter can have multiple output files, with different subsets of data going to each output. The extract_corsika_tel allows for more control over which events are selected while extract_corsika_multi will process all events. Both have a --header-only option to show what environment variables would be set up by multipipe_corsika to help with further processing, finishing input after the header data blocks and not writing any output data then. The very similar --header-only-export option puts an 'export ' in front of each line, for easier use with the 'bash':

```
Syntax: extract_corsika_tel [ options ] [ input ]
Options:
   --telescopes ...       (List of telescopes, e.g. 1,2,5-7,9)
                          Same: --telescope, --only-telescope, --only-telescope
   -o fname               (Output to file rather than stdout)
                          Same: --output-file
   --header-only          (Only process run and first event header)
   --header-only-export   (Same but print 'export ' in front of lines)
   --only-shower n        (Pick out data for a single shower)
   --only-array n         (Pick out photon data for a single array)

Syntax: extract_corsika_multi [ options ] [ input ]
Options:
   --array ...            (Array number: 1 to 20, to following options apply)
   --telescopes ...       (List of telescopes, e.g. 1,2,5-7,9)
                          Same: --telescope, --only-telescope, --only-telescope
   -o fname               (Output to file rather than stdout)
                          Same: --output-file
   --header-only          (Only process run and first event header)
   --header-only-export   (Same but print 'export ' in front of lines)
```

15.2.4 Other tools

A tool mainly intended for looking into and analysing sim_telarray output but also capable of showing a text-mode representation of CORSIKA output is read_hess (also known as read_cta or read_simtel), using the '-S' (show full data) option.

A much simpler program only for showing the contents of the data written by the CORSIKA IACT interface (with CORSIKA as well as with LightEmission applications) is read_iact comes included with the IACT/ATMO package since version 1.57 and also the general eventio package (hessioxxx.tar.gz).

Photon bunches coming from CORSIKA simulations with information enabled about emitting particles can be filtered with `select_iact` depending on the particle type and/or its momentum etc.

15.3 Object types in the data file

Type	Description
1200	CORSIKA run header
1201	Positions and sizes of telescopes within telescope array
1202	CORSIKA event header
1203	Offsets of multiple telescope arrays for the present event
1204	Top level item for data from one array in one event
1205	Photons hitting one telescope or particles at observation level
1206	(not used)
1207	(not implemented)
1208	Photo-electrons after ray-tracing and detection
1209	CORSIKA event end
1210	CORSIKA run end
1211	Longitudinal (vertical) profiles of particles and light
1212	A copy of the inputs file used to steer CORSIKA
1213	Marker of the beginning of per-telescope split-up data
1214	Marker of the end of per-telescope split-up data
1215	Optional extra shower data
1216	Atmospheric profile data (table and/or fit)

15.4 Object formats

The following detailed list of the object formats start with the object type number (as before), the version described, and the meaning of the ident number in the block. Variables of type *Long* are 4-byte integers, variables of type *Real* are 4-byte floating point numbers. *Short* variables are 2-byte integers.

15.4.1 CORSIKA run header

This block (like the other CORSIKA blocks) contains just the same data as in the original CORSIKA data files (but in a machine-independent format, of course). For descriptions of their contents consult the CORSIKA user's guide (for the CORSIKA version in use).

Object type: 1200 (`IO_TYPE_MC_RUNH`)
Version: 0

Identifier: run number

Variable	Type	Number	Description
len	Long	1	Combined count of data words following (usually 273)
runh	Long	1	FORTRAN string RUNH (may be reversed)
data	Real	272	see CORSIKA user's guide

15.4.2 Positions and sizes of telescopes

Object type: 1201 (IO_TYPE_MC_TELPOS)
Version: 0
Identifier: 0

Variable	Type	Number	Description
ntel	Long	1	Number of telescopes in an array
x	Real	ntel	x pos. (measured towards north, unit: cm)
y	Real	ntel	y pos. (measured towards west, unit: cm)
z	Real	ntel	z pos. (from detection level, unit: cm)
r	Real	ntel	Radii of spheres around tel. (unit: cm)

15.4.3 CORSIKA event header

Object type: 1202 (IO_TYPE_MC_EVTH)
Version: 0
Identifier: event number

Variable	Type	Number	Description
len	Long	1	Combined count of data words following (usually 273)
runh	Long	1	FORTRAN string EVTH (may be reversed)
data	Real	272	see CORSIKA user's guide

15.4.4 Offsets of telescopes

With usually multiple instances of an array of telescopes being simulated, the arrays are randomly offset in each event.

Object type: 1203 (IO_TYPE_MC_TELOFF)
Version: 0
Identifier: 0

Variable	Type	Number	Description
narray	Long	1	Number of arrays
toff	Real	1	Time delay since first interaction (already subtracted, unit: ns))
xoff	Real	narray	Offsets of arrays in x (unit: cm)
yoff	Real	narray	Offsets of arrays in y (unit: cm)

15.4.5 Data top-level block for one array

The data for one array are stored in one top-level block. Within this top-level block either the photons arriving at the telescope detection plane or the photo-electrons registered in the PM camera are contained. Note that since the data for one array is all within one top-level block, sufficient memory must be available to buffer this block. Data from the top-level block is extracted until the end of the block is reached.

Object type: 1204 (IO_TYPE_MC_TELARRAY)
Version: 0
Identifier: Array number

Variable	Type	Number	Description
either photons	1205	(any)	Photons arriving at one telescope
or photo_electrons	1208	(any)	Detected photo-electrons in one camera

15.4.6 Photon bunches arriving at the telescope places

The CORSIKA photon bunches are sorted by telescope and array and are written after a full shower has been simulated. Photon bunches are actually recorded in two different formats: a long format with 32 bits per value and a short format with just 16 bits per value. Format

conversion is done by the software automatically and is transparent at the user level – only the numerical accuracy is different and the amount of storage needed. *This block is only present in data read directly from the shower simulation.* It is always a sub-block of type 1204.

Long format:

Object type: 1205 (IO_TYPE_MC_PHOTONS)
Version: 0
Identifier: 1000 × array number + telescope number

Variable	Type	Number	Description
array	Short	1	Array number of telescope
tel	Short	1	Telescope number
Photons	Real	1	Sum of photons in all bunches
bunches	Long	1	Number of photon bunches following
per bunch:	(bunches)		
x	Real	1	x pos. relative to telescope (unit: cm)
y	Real	1	y pos. relative to telescope (unit: cm)
cx	Real	1	x direction cosine
cy	Real	1	y direction cosine
time	Real	1	arrival time relative to time when the primary travelling at $v = c$ would arrive at the core in the CORSIKA detection plane (unit: ns)
zem	Real	1	Altitude of emission (unit: cm a.s.l.)
photons	Real	1	Photons in this bunch
lambda	Real	1	Zero = unspec. (see note below) (unit: nm)

Short format:

Object type: 1205 (IO_TYPE_MC_PHOTONS)
Version: 1000
Identifier: 1000 × array number + telescope number

Variable	Type	Number	Description
array	Short	1	Array number of telescope
tel	Short	1	Telescope number
Photons	Real	1	Sum of photons in all bunches
bunches	Long	1	Number of photon bunches following

per bunch:	(bunches)		
x	Short	1	x position (unit: 0.1 cm)
y	Short	1	y position (unit: 0.1 cm)
cx	Short	1	x direction cosine (unit: 1/30000)
cy	Short	1	y direction cosine (unit: 1/30000)
time	Short	1	arrival time (unit: 0.1 ns)
log_zem	Short	1	$1000 \times \log_{10}(z_{\mathrm{em}}[cm])$
photons	Short	1	Photons in this bunch (unit: 0.01)
lambda	Short	1	Zero = unspec. (see note below) (unit: nm)

The lambda wavelength field in both the long and short formats is usually zero to indicate an unspecified wavelength, from a $1/\lambda^2$ distribution in the wavelength range given in the CORSIKA inputs and reported in the event header block. Specific wavelengths above 200 nm, either with the CORSIKA CERWLEN option or the fluorescence emission add-on are indicated by positive values (but note that values 1, 2, and 3 can appear in special configurations to indicate photons of unspecified wavelength from a primary particle or a particle of an energy close to that of the primary particle). Negative values (usually -1) indicate photo-electrons (atmospheric transmission, mirror reflectivity and quantum efficiency applied with the CEFFIC option). Values of and above 9000 (usually: 9999) indicate properties of the particles emitting the Cherenkov light (cx: particle mass in GeV/c^2, cy: charge number, photons: the particle energy in GeV, zem the emission time in seconds, either since the primary entering the atmosphere or the first interaction). For the latter, the CORSIKA and IACT interface have to be compiled with the IACTEXT option and the IACT interface also with STORE_EMITTER defined.

Data blocks with array=999 and tel=999 indicate particles hitting the CORSIKA observation level(s) (x and y are w.r.t. the core position, not any detector fiducial sphere), with the particle ID i, the generation number g, and the observation level number l encoded in the lambda field like in the first word of the CORSIKA particle-data sub-block: $i \times 1000 + g \times 10 + l$ and the particle momentum in GeV/c contained in the zem field. The photons field contains the thinning weight or 1.0. (Particle IDs 75 and 76 indicate additional information about muons, with ctime indicating the muon production height in cm a.s.l., and zem repeating its momentum in the zem field.) These particle data blocks can only be produced if CORSIKA and the IACT interface got compiled with the IACTEXT option.

15.4.7 Photo-electrons after ray-tracing and detection

The sim_telarray program writes (if wanted) the detected photo-electrons to an output file. This output file contains all blocks from the input file – but with photo-electrons instead of photons – plus the assumed camera layout. The photons are sorted by arrival time at the PM.

This block is not present in data read directly from the shower simulation. It is also not present in the normal `sim_telarray` *output.* It is a subblock of type 1204.

Object type: 1208 (`IO_TYPE_MC_PE`)
Version: 0
Identifier: 1000 × array number + telescope number

Variable	Type	Number	Description
`pixels`	Long	1	Number of pixels in camera
per non-empty pixel:			
`ipix`	Short	1	Pixel number
`npe`	Short	1	Number of photo-electrons
`time`	Real	`npe`	Times of photo-electrons (unit: ns)

15.4.8 CORSIKA event end block

Object type: 1209 (`IO_TYPE_MC_EVTE`)
Version: 0
Identifier: Event number

Variable	Type	Number	Description
`len`	Long	1	Combined count of data words following (usually 273)
`evte`	Long	1	FORTRAN string "EVTE" (may be reversed)
`data`	Real	272	see CORSIKA user's guide

15.4.9 CORSIKA run end block

Object type: 1210 (`IO_TYPE_MC_RUNE`)
Version: 0
Identifier: Run number

Variable	Type	Number	Description
`len`	Long	1	Combined count of data words following (usually 3)
`rune`	Long	1	FORTRAN string "RUNE" (may be reversed)
`data`	Real	2	see CORSIKA user's guide

15.4.10 CORSIKA shower longitudinal distributions

Longitudinal distributions are either in vertical grammage units (without SLANT option when CORSIKA is compiled) or in grammage along the incidence direction of the primary (with SLANT option, see run header block in CORSIKA user's guide). There are three possible types of distributions: type 1 = particle numbers, 2 = energy, 3 = energy deposits. Only data of type 1 distributions gets normally recorded.

Object type: 1211 (IO_TYPE_MC_LONGI)
Version: 0
Identifier: 10x event number + type (1/2/3)

Variable	Type	Number	Description
event	Long	1	Event number (is modulo 100 000 000 also in ID).
type	Long	1	1 = particle numbers, 2 = energy, 3 = energy deposits.
np	Short	1	Number of distributions following.
nthick	Short	1	Number of thickness (grammage) steps in distributions.
thickstep	Real	1	Step thickness (grammage) in g/cm^2.
data	Real	np	Distribution data of nthick values for each.

The normal type 1 distribution block includes nine distributions: gammas, positrons, electrons, mu+, mu-, hadrons, charged particles of any type, nuclei, Cherenkov photons.

15.4.11 CORSIKA input 'cards'

The CORSIKA control input (in traditional FORTRAN speak also referred to as 'cards' for each line) gets completely recorded. Together with CORSIKA version and compile-time activated options it should allow for reproducing all generated showers.

Object type: 1212 (IO_TYPE_MC_INPUTCFG)
Version: 0
Identifier: 0

Variable	Type	Number	Description
nlines	Long	1	Number of input lines ('cards') following.
cards	String	nlines	Original CORSIKA input

Chapter 16

Output from sim_telarray

The data format of the normal output from sim_telarray is based on [9] and implemented with the hessio package (using eventio). For details see the hessio package. Note that no such output file is written by default. Its name has to be given through the OUTPUT_FILE configuration parameter or the -o command line option.

The hessio package provides several tools to inspect the contents of the main output file specified that way. Tools generic to all 'eventio' format files, as already mentioned, are listio (showing the sequence and hierarchical structure of data blocks) and statio (showing the number of blocks of each type, the total length of data in them, as well as the version number(s) used), filterio for selecting or deselecting specific data block types from the data stream.

There are several tools specifically for the output from sim_telarray, with the most notable being read_simtel (also known as read_hess or read_cta) which cannot only show the contents of sim_telarray output but can also process them, including data reduction, shower reconstruction etc. For the full reconstruction power, including gamma-hadron discrimination and energy reconstruction, the gen_lookup program processes the generated histograms and produces look-up tables for the next processing iteration (three read_simtel passes might be needed, the first two with the --auto-lookup option).

The extract_simtel utility is intended for extracting data for a subset of simulated telescopes into a new data file while merge_simtel would be used to merge events for the same showers (as coming from CORSIKA) but passed through separate telescope simulations into a new output file. There are two input files per stage but multiple use allows for merging data processed in parallel through more than two telescope simulations. The demands on main memory can be challenging for that, as the the usual data structures are needed three times (two inputs, one output), all configured for the maximum number of telescopes and pixels that might be encountered on either input or output.

The extract_calibevent tool allows to extract data from internal dark/pedestal/LED/laser events into a separate file which can be processed like regular data. The split_hessio program can split up to single data stream into separate files for each telescope, as to be expected from the data acquisition of a telescope array before merging them into a common file (which sim_telarray provides directly).

In addition, plain photo-electrons can be written to a separate output file following the data format in section 15.

The sim_telarray program also writes a histogram file, including one- and two-dimensional histograms. The format of this file is also built on top of eventio. It is called 'ctsim.hdata' by default – but usually set by the HISTOGRAM_FILE parameter to match the OUTPUT_FILE name – and it can be converted to HBOOK format (of old CERNLIB fame) with the hdata2hbook program (available with the hessio package, called cvt2 in earlier releases):

```
hdata2hbook ctsim.hdata ctsim.hbook
```

The .hdata file can also be converted to a ROOT file through hdata2root (cvt3 in earlier releases):

```
hdata2root ctsim.hdata ctsim.root
```

Note that both of these programs can also be used to add up corresponding histograms in separate .hdata files before writing the output:

```
hdata2root -a run*.hdata -o allruns.root
```

A small subset of these histograms are included at the end of the main output data file (e.g. ctsim.simhess), sufficient to calculate effective areas versus energy etc. Running hdata2hbook or hdata2root on such data files will also work to convert this subset of histograms. The programs will usually also find the histograms if you just throw the end of the data file at it:

```
tail -200000c ctsim.simhess | hdata2root - ctsim.root
```

but,unfortunately, this will not work with compressed files.

The hessio package provides several tools specific to its own histogram format (for either the separate histogram files or the histograms embedded with the normal data): list_histograms shows the list of available histograms or the contents of a specific histogram (by ID), with optional projection of 2-D histograms or calculating the ratio between contents of two (matching) histograms. The output from list_histograms can be plotted with gnuplot or other plotting tools. The add_histograms tool can add up all histograms per ID (if matching in number of bins and ranges) from multiple files. Applying that to the separate histogram files can quickly provide the full statistics of a complete MC production with thousands of simulation runs.

Further optional output includes a PostScript format file as activated with the IMAGE_FILE configuration parameter. This includes one page for each telescope and event (or two in case of zero-suppression: one before and one after suppression). Whether non-triggered telescopes are included, is determined with the configuration parameter ONLY_TRIGGERED_TELESCOPES. While pages in the 'image' file usually only show the sum of the signals in the readout window, the MOVIE configuration parameter, if non-zero, changes this to one page for each time slice, allowing to create movie sequences with

shower images. As long as the sim_telarray main output data is for sample mode, read_simtel with the '-p' option can provide similar per-sample Postscript output (use gv or a similar tool to extract the range of pages for a specific event/telescope before converting that into animated GIF or an actual video format).

Through the PLOT_FILE parameter an additional output file can be activated which contains (depending on various compilation-time flags) miscellaneous ASCII-format information suitable, after selecting relevant lines, for plotting with gnuplot or other tools.

16.1 Object types in the data file

Type	Description
2000	Run header: Sim_telarray global run header
2001	MC run header: Sim_telarray MC-specific run header
2002	Camera settings
2002	Pixel settings
2003	Camera organisation
2004	Pixel settings
2005	Disabled pixels
2006	Camera soft settings
2007	Pointing correction
2008	Tracking setup
2009	Central trigger
2010	Event: Sim_telarray event data for the full array
2011	Telescope event header
2012	ADC sums
2013	ADC samples
2014	Image parameters
2015	Reconstructed shower
2016	Pixel timing
2017	Calibrated pixel intensities
2020	MC shower
2021	MC event
2022	Telescope monitoring
2023	Laser calibration
2024	Run statistics
2025	MC run statistics
2026	Photo-electron sums
2027	Pixel list
2028	Calibration event
2029	Digital auxiliary traces
2030	Analog auxiliary traces

2032	Pixel trigger w.r.t telescope trigger time
2033	MC pixel monitor
2034	Calibration event p.e. list
2090	Trigger-type bitmasks
21xx	Tracking data for tel. ID xx
22xx	Telescope data for tel. ID xx
31xx	Tracking data for tel. ID 1xx
32xx	Telescope data for tel. ID 1xx

Also present: types 1204 (containing 1208), and 1205, the former with the list of detected signal photo-electrons, the latter with a copy of the CORSIKA data about particles reaching ground level (CORSIKA data pass-through).

Bibliography

[1] H.J. Völk and K. Bernlöhr, *Imaging very high energy gamma-ray telescopes*, Exp. Astron. 25 (2009) 173–191. (WWW link here and preprint here). 1

[2] D. Heck et al., *CORSIKA: A Monte Carlo code to simulate extensive air showers*, Technical Report FZKA 6019, Forschungszentrum Karlsruhe, 1998. (WWW link here) 1

[3] D. Heck and J. Knapp, *Extensive Air Shower Simulation with CORSIKA: a User's Guide*, Forschungszentrum Karlsruhe, 2002. (WWW link here) 2.4, 3

[4] K. Bernlöhr, *'Entwurf eines Datenformats für CRT'*, internal report, 1994, and *'*`eventio` *– ein maschinen-unabhängiges hierarchisches Datenformat und seine Software-Schnittstelle'*, internal report, 1998 (updated 2005) (WWW link here)
`eventio` *– a machine-independent hierarchical data format and its programming interface*, internal report, 2000 (updated 2014). (WWW link here) 1, 15.1

[5] K. Bernlöhr, *Simulation of imaging atmospheric Cherenkov telescopes with CORSIKA and sim_telarray*, Astroparticle Physics 30 (2008) 149–158 (WWW link here and here and preprint here). 1, 9

[6] K. Bernlöhr, `sim_telarray` *Reference Manual*, generated by doxygen. (WWW link here) 11.7, 11.11, 14

[7] K. Bernlöhr, *CORSIKA add-on package IACT/ATMO: Reference Manual*, generated by doxygen. (WWW link here and here) 1, 14

[8] K. Bernlöhr, `hessio` *– an eventio-based I/O library for H.E.S.S. – Reference Manual*, generated by doxygen. (WWW link here) 14

[9] K. Bernlöhr and G. Hermann, *A preliminary H.E.S.S. data model*, internal report, 2000. (WWW link here) 16

[10] F. X. Kneizys et al. (1996), *The MODTRAN 2/3 report and LOWTRAN 7 model*, Phillips Laboratory, Hanscom AFB, MA 01731, U.S.A. 2.4, 3, 4, 11.4

[11] A. Koch and A. Kohnle (March 2001), *Quantum and Collection Efficiency Measurements of the Photonis XP2960 Photomultipliers*, H.E.S.S. internal note 01/07. (WWW link here) 11.14

[12] J. Guy, F. Toussenel, P. Vincent (December 2001), *Photoelectron pulse shape measurement and the impact on the simulation* , H.E.S.S. internal note 01/10. (WWW link here) 11.16

[13] K. Bernlöhr, Astroparticle Phys. 12, 255 (2000). (WWW link here) and (preprint here) 10.4.1

[14] C.C.G. Bowden et al., J.Phys. G18, L55 (1992). (WWW link here) 10.4.1

[15] P.M. Chadwick et al., J.Phys. G25, 1223 (1999). (preprint here) 10.4.1

[16] P.M. Chadwick et al., J.Phys. G26, L5 (2000). (WWW link here) 10.4.1

[17] K. Bernlöhr, *Geomagnetic effects on showers relevant for the H.E.S.S. experiment - a study with simulated showers*, H.E.S.S. internal note 05/02 (July 2005). (WWW link here) 10.4.1

Index

Miscellaneous

Telescope simulation parameters