LightEmission

2023-01-31

Generated by Doxygen 1.9.1

Chapter 1

IactLightEmission - simulate artificial light sources

1.1 Introduction

The data format used with the IACT package for CORSIKA can also be used to simulate 'artifical' light sources rather than Cherenkov light emitted by extensive air showers. Like with CORSIKA, you set up detector fiducial spheres in some observation level plane (usually just one, and due to the '-DIACT_NO_GRID' definition in rather than above that plane), start a simulation 'run', with a single or many events in it. Instead of the air showers, light is produced by user-defined light sources, with given positions, alignment, angular and spectral emission characteristics, and emitting a requested number of photons in bunches of a given number of photons per bunch.

Many parameters in CORSIKA run header and events headers make little sense with these simulations, although those used to control subsequent telescope or camera simulations work just like for air showers. Thus output from these simulations of artificial light sources can be processed by sim_telarray just like CORSIKA air shower simulations. Light sources may illuminate an entire telescope, or an array of telescopes, as with shower simulations. Or they may illuminate a camera directly or, in a dual-mirror telescope, after only reflection on the secondary mirror (see BYPASS_OPTICS parameter for sim_telarray).

The typical program needs a

- Run class constructor and output file assignment.
- Setting up one or more light sources.
- A loop over a given number of events, each with a NextEvent() call for the run object, in order to write out any photon bunches from the preceding event and initialize the next on, followed by an Emit() call for each of the defined light sources (or a subset of those light sources if they alternate from event to event).
- Output of photon bunches from the final event and clean-up will be performed by the Run class destructor.

A very simple program might look like:

```
#include "IactLightEmission.hh"
int main()
{
   int iev, nev=10;
   Run run(1);
   SimpleLightSource ls;
   for ( iev=0; iev<nev; iev++ )
   {
      run.NextEvent();
      ls.Emit(run,1000.,1.,0.);
   }
   return 0;
}
```

1.2 Light sources types

There are a number of types of light sources avaible, from simple, point-like sources with isotropic emission, to extended and/or moving sources with user-defined emission characteristics. These include in particular:

- SimpleLightSource : Isotropic emission at fixed wavelength and very simple time profile (top-hat or Gaussian), characterized by its Full-Width-at-Half-of-Maximum (FWHM) pulse duration.
- BackgroundLightSource : A diffuse light source emitting for a given time period, isotropic within a given angular range, and with a user-defined spectral characteristic which defaults to the known night-sky background spectrum.
- LightSource : A more generic kind of light source with user-defined spectral, angular, and temporal characteristics (which can be a keyword like 'Gauss' followed by a colon and a value, or a table from which the corresponding distribution gets loaded) which can have a linear extension along its primary direction and can move along that with some finite speed. This may emulate, for example, an LED flasher, a muon emitting Cherenkov light, or a laser beam scattered in the atmosphere.

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 BackgroundLightSource Class Reference

A light source class with diffuse emission characteristics.

`#include <IactLightEmission.hh>`

Public Member Functions

- **BackgroundLightSource** (const char ∗spec_fname=0, double pwid=0., double pdel=0., double vc=0., const DirectionVect &direct=DirectionVect(0., 0.,-1.))
- void **SetMaxViewcone** (double mvc)
- void **SetSpectrum** (const char ∗fname)
- void **SetEmissionPulseWidth** (double pw)
- void **SetDelay** (double t)
- int Emit (const Run &run, double photons, double bsize, double when)

  Emit the requested number of photons in bunches of the requested size at the requested time. Depending on the distance between light source and detector fiducial sphere(s) and the solid angle actually subtended, the number of bunches that need to be written might be lower than expected from looking at photons and bsize alone.

4.1.1 Detailed Description

A light source class with diffuse emission characteristics.

The background (mainly nightsky background) light source is not a localized emitter but corresponds to a uniform flux of isotropic background, actually generated at the fiducial sphere. If there is more than one fiducial sphere the requested number of photons are split up to be distributed according to sphere cross section areas among the sphere. For efficiency reasons the emission is not across the whole hemisphere but within a 'viewing cone' up to a given limiting angle from a principal direction (which gets recorded as direction of the 'primary'). The spectrum can be specified as for the generic LightSource class but the default spectrum is the Benn&Ellison NSB spectrum. The light pulse time distribution can only be a flat-top distribution, the default being instantaneous for compatibility with other light source classes.

4.1.2 Member Function Documentation

4.1.2.1 Emit()

```
int BackgroundLightSource::Emit (
            const Run & run,
            double photons,
            double bsize,
            double when )
```

Emit the requested number of photons in bunches of the requested size at the requested time. Depending on the distance between light source and detector fiducial sphere(s) and the solid angle actually subtended, the number of bunches that need to be written might be lower than expected from looking at photons and bsize alone.

Parameters

run	Run class
photons	Number of photons that would be emitted into 4 pi steradian (isotropic).
bsize	Intended bunch size. A conservative approach would use a value of 1.0. Any value below one will be replaced by 1.0.
when	Nominal emission time, with individual photon bunches obtaining a random time value to reproduce the configured pulse shape, on average. Unit: nanoseconds.

Returns

0 (OK)

The documentation for this class was generated from the following files:

- IactLightEmission.hh
- IactLightEmission.cc

4.2 DirectionVect Class Reference

A normalized direction vector.

`#include <IactLightEmission.hh>`

Public Member Functions

- DirectionVect ()

  Note that length must be normalized, cannot be (0,0,0).
- **DirectionVect** (double cxp, double cyp, double czp)
- **DirectionVect** (double alt, double az=0.)
- **DirectionVect** (const SpaceVect &sv)
- SpaceVect operator∗ (double d) const

  A given distance along the direction defines a space vector.
- SpaceVect SV () const

  Reverse conversion without multiplication.
- double Altitude () const

  Altitude (or Elevation) angle above x/y plane, in radians.

- double Polar () const

  Angle from pole (or zenith), in radians.
- double Azimuth () const

  The azimuthal angle in the x/y plane, counted in the mathematical (counter-clockwise!) sense, in radians.
- double AltitudeDeg () const

  The Altitude angle converted into degrees.
- double PolarDeg () const

  The polar (or zenith) angle converted into degrees.
- double AzimuthDeg () const

  The azimuthal angle converted into degrees.
- double AngleTo (const DirectionVect &d2) const

  Angle between two direction vectors (or via typecase between a direction vector and a space vector), in radians.
- double AngleToDeg (const DirectionVect &d2) const

  Angle between to vectors converted into degrees.
- double SProd (const DirectionVect &dv2) const

  Scalar (inner) product of two direction vectors.
- double **operator**∗ (const DirectionVect &dv2) const
- double SProd (const SpaceVect &sv2) const

  Scalar (inner) product with a space vector, without typecast.
- SpaceVect VProd (const SpaceVect &sv2) const

  Vector (outer) product with a space vector is a space vector (not normalized).
- SpaceVect VProd (const DirectionVect &dv2) const

  Vector (outer) product with a direction vector is still not normalized.
- DirectionVect NVProd (const DirectionVect &dv2) const

  Normalized vector (outer) product of two direction vectors.
- bool IsParallelTo (const DirectionVect &dv2) const

  Better check if two vectors are (anti-) parallel before asking for the normalized vector product.
- bool AlmostParallelTo (const DirectionVect &dv2) const

  Perhaps two direction vectors are (anti-) parallel within possible rounding errors.
- DirectionVect & Rotate_z (double phi)

  Rotate counter-clockwise around given axis (active rotation)
- DirectionVect & **Rotate_x** (double phi)
- DirectionVect & **Rotate_y** (double phi)
- DirectionVect & Rotate (double alpha, double beta, double gamma)

  Active rotation with Euler angles in standard x convention (sequence: z,x',z'')
- DirectionVect & Scatter (double dtheta, double dphi)
- ostream & Print (ostream &os) const

  Show human-readable version of the class.

Public Attributes

- double **cx**
- double **cy**
- double cz

  Leave the direction cosines public for efficient access.

4.2.1 Detailed Description

A normalized direction vector.

The DirectionVect class is, like SpaceVect, just a 3-tuple of x/y/z coordinates but, in contrast to SpaceVect, meant as a normalized(!) direction vector and not a position or difference of positions (which is again a SpaceVect). Scaling a DirectionVect with a length gives a SpaceVect.

4.2.2 Member Function Documentation

4.2.2.1 Scatter()

```
DirectionVect& DirectionVect::Scatter (
            double dtheta,
            double dphi )  [inline]
```

New direction after scatter by angle dtheta between 0 and pi and (typically randomly chosen) angle dphi around the initial direction.

The documentation for this class was generated from the following file:

- IactLightEmission.hh

4.3 LightSource Class Reference

A much more generic type of light source than the SimpleLightSource.

```
#include <IactLightEmission.hh>
```

Public Member Functions

- LightSource ()
  Default light source constructor.
- LightSource (const SpaceVect &ls_pos, const char *spec_fname=0, const char *pulse_fname=0, const char *angdist_fname=0, const DirectionVect &direct=DirectionVect(0., 0.,-1.))
- LightSource (const SpaceVect &ls_pos, const string &spec_fname, const string &pulse_fname="Simple:0", const string &angdist_fname="isotropic", const DirectionVect &direct=DirectionVect(0., 0.,-1.))
- LightSource (const SpaceVect &ls_pos, const LightSource &ls1)
  Light source constructor copying an existing source into another place.
- const SpaceVect & GetPos () const
  Get the (starting) position of a light source.
- double **GetX** () const
- double **GetY** () const
- double **GetZ** () const
- void SetSpectrum (const char *fname)
  Set the spectrum of a light source.
- void **SetSpectrum** (const string &fname)
- void SetEmissionPulse (const char *fname)
  Set the pulse shape parameters for a light source.
- void **SetEmissionPulse** (const string &fname)
- void SetAngularDistribution (const char *fname)
  Set the angular distribution parameters for a light source.
- void **SetAngularDistribution** (const string &fname)
- void SetPosition (const SpaceVect &npos)
  Set the position of a light source.

- void SetDirection (const DirectionVect &dir)

  Set the orientation of a light source.
- void SetDirectionTo (const SpaceVect &topos)

  Make a light source point from its current location towards another location.
- void SetLength (double l)

  Set the length of a light source.
- void SetSpeed (double b)

  Set the propagation speed along the length of a light source, in units of c.
- void SetDelay (double t)

  Set the time delay after which the light source should fire.
- void SetName (const string &n)

  Set the name of a light source for later identification.
- void **SetName** (int n)
- double **GetLength** () const
- double **GetSpeed** () const
- double **GetDelay** () const
- const string & **GetName** () const
- int Emit (const Run &run, double photons, double bsize, double when)

  Initiate actual emission of light pulse from this one light source.

4.3.1 Detailed Description

A much more generic type of light source than the SimpleLightSource.

A fairly generic light source with a tabulated spectrum, a tabulated pulse shape, tabulated emission angle profile symmetric around a given pointing direction. If the emission angle distribution is missing, it is reduced to an isotropically emitting light source. Predefined angular distribution names not requiring to read any table are "↩ Isotropic", "Gauss:<rms>", "Rayleigh", "Cone:<angle>", and "FilledCone:<angle>" as well as "Parallel", with all angles given in degrees, all with respect to the given direction vector (vertically downwards if missing). If the light source has a non-zero length and velocity (in units of the vaccum speed of light), it is handled as a moving source, in the given direction.

If the pulse shape is missing, it is reduced to instantenous emission. Predefined pulse shapes not requiring to read any table are "Simple:<width>[,<mean>]" (for top-hat shape of given width [ns], like for a SimpleLightSource) or "Gauss:<rms>" (sigma=r.m.s. value in [ns]) and others. Unfortunately, with versions prior to 2020-12-01 the "Gauss:..." value was interpreted as a FWHM (as in SimpleLightSource), in contrast to documentation. Therefore, more explicit variants have been added: "Gauss-rms:<rms>" (r.m.s. value in [ns]) and "Gauss-fwhm:<fwhm>" (giving the FWHM duration of a Gaussian pulse profile [ns], corresponding to a SimpleLightSource with the 'gs' (Gauss) flag set to true).

Optional <mean> time of emission (default 0) also in [ns].

If the spectrum is missing, a wavelength of 400 nm is assumed. If the spectrum string represents a number it is interpreted as a fixed wavelength in [nm]. Another predefined distribution is "Cherenkov:<from_wl>:<to_wl>", resulting in a 1/lambda^2 distribution from from_wl to to_wl (both in [nm]). In all other cases files are opened to read in the corresponding distributions.

Remaining restrictions are that the light source is still point-like or on a line, with a possible propagation of the point of emission along that line. There is no correlation of angular characteristics or spectrum with time of emission.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 LightSource() [1/2]

```
LightSource::LightSource (
              const SpaceVect & ls_pos,
              const char * spec_fname = 0,
              const char * pulse_fname = 0,
              const char * angdist_fname = 0,
              const DirectionVect & direct = DirectionVect(0.,0.,-1.)  )  [inline]
```

Light source constructor that can pass most, but not all, parameters along, with text parameters as C strings. Some less frequently used parameters may have to be set after creation.

4.3.2.2 LightSource() [2/2]

```
LightSource::LightSource (
              const SpaceVect & ls_pos,
              const string & spec_fname,
              const string & pulse_fname = "Simple:0",
              const string & angdist_fname = "isotropic",
              const DirectionVect & direct = DirectionVect(0.,0.,-1.)  )  [inline]
```

Light source constructor that can pass most, but not all, parameters along, with text parameters as C++ strings. Some less frequently used parameters may have to be set after creation.

4.3.3 Member Function Documentation

4.3.3.1 Emit()

```
int LightSource::Emit (
              const Run & run,
              double photons,
              double bsize,
              double when )
```

Initiate actual emission of light pulse from this one light source.

Emit the requested number of photons in bunches of the requested size at the requested time. Depending on the distance between light source and detector fiducial sphere(s) and the solid angle actually subtended, as well as the angular distribution chosen, the number of bunches that need to be written might be lower than expected from looking at photons and bsize alone.

This more generic form of a light source allows for arbitrary angular emission shapes. Instead of being a point source, it can have a length (from the starting point towards a given direction). The actual emission point can move along that length with a given speed - which can cover scattered light from a narrow laser beam as well as Cherenkov or fluorescence emission from a charged particle.

Parameters

run	Run class
photons	Number of photons that would be emitted into the accessible solid angle (isotropic or in any configured angular distribution).
bsize	Intended bunch size. A conservative approach would use a value of 1.0. Any value below one will be replaced by 1.0.
when	Nominal emission time, with individual photon bunches obtaining a random time value to reproduce the configured pulse shape, on average. Unit: nanoseconds.

Returns

0 (OK)

The documentation for this class was generated from the following files:

- IactLightEmission.hh
- IactLightEmission.cc

4.4 Line Class Reference

A line is defined by a point on the line and the direction vector along the line.

`#include <IactLightEmission.hh>`

Public Member Functions

- **Line** (const SpaceVect &ssv, const DirectionVect &sdv)
- const SpaceVect & Position () const

  Retrieve configured point on line.
- const DirectionVect & **Direction** () const
- double DistanceTo (const SpaceVect &p) const

  Shortest distance of a point in space from the line.
- double **DistanceFrom** (const SpaceVect &p) const
- double **Distance** (const SpaceVect &p) const
- SpaceVect DistanceVectorFrom (const SpaceVect &p) const

  Vector representing the shortest line from a point to the given line.
- SpaceVect DistanceVectorTo (const SpaceVect &p) const
- double DistanceTo (const Line &l2) const

  Shortest distance between two lines. Equivalent to DistanceVectorTo().Length() but faster.
- double DistanceFrom (const Line &l2) const

  Same as DistanceTo() as the direction does not matter.
- double Distance (const Line &l2) const

  Same as DistanceTo() and DistanceFrom().
- ostream & Print (ostream &os) const

  Show human-readable version of the class.

Public Attributes

- SpaceVect sv

  Supporting vector on line.
- DirectionVect dv

  Direction vector along line.

4.4.1 Detailed Description

A line is defined by a point on the line and the direction vector along the line.

The line is meant as infinite in either direction, although the defined point and direction can be retrieved. Any distance or distance vector is with respect to the both-sided infinite line.

4.4.2 Member Function Documentation

4.4.2.1 DistanceVectorTo()

```
SpaceVect Line::DistanceVectorTo (
            const SpaceVect & p ) const  [inline]
```

Vector representing the shortest line (v) from the given line (starting at p-v) to a point (p). That is the opposite of DistanceVectorFrom().

The documentation for this class was generated from the following file:

- lactLightEmission.hh

4.5 photon_bunch Struct Reference

Public Attributes

- double photons

  Size of photon bunch.
- double weight

  Weight = 1.0 (no thinning)
- double **xem**
- double **yem**
- double zem

  Emission position.
- double **xol**
- double **yol**
- double tol

  Position arriving at "observation level" (below telescope)
- double **dx**
- double **dy**
- double dt

  Shift in ray propagation.
- double **u**
- double **v**
- double w

  All direction cosines (not necessarily normalized)
- double ctime

  Time of emission.
- double lambda

  Wavelength of photons in bunch.

The documentation for this struct was generated from the following file:

- lactLightEmission.cc

4.6 PixLed Struct Reference

Public Member Functions

- **PixLed** (int ipix, bool rq=false, const string &p="", const string &s="", double d=0.)
- **PixLed** (int ipix, bool rq, const string &p, const string &s, double d, const vector< int > slave_ids)

Public Attributes

- int **id**
- bool **required**
- string **pulse**
- string **spectrum**
- double **delay**
- vector< int > **slaves**

The documentation for this struct was generated from the following file:

- pixled.cc

4.7 PixPos Struct Reference

Public Member Functions

- **PixPos** (int pid, double px, double py)

Public Attributes

- int **id**
- double **x**
- double **y**

The documentation for this struct was generated from the following file:

- pixled.cc

4.8 Plane Class Reference

A plane in 3-D space and intersections.

```
#include <IactLightEmission.hh>
```

Public Member Functions

- Plane ()

  Default plane is in x/y.
- Plane (const DirectionVect &nvs, double ds)

  Plane defined by normal vector and distance to origin.
- Plane (const SpaceVect &sv, const DirectionVect &v1, const DirectionVect &v2)

  Plane defined by support point on plane and two (non-parallel) direction vectors spanning the plane.
- Plane (const SpaceVect &sv, const DirectionVect &n)

  Plane defined by support point on plane and the normal vector.
- const DirectionVect & Direction () const

  Direction of normal vector, e.g. for calculating an angle with a line or such.
- double DistanceTo (const SpaceVect &p) const

  Distance of a point in space from the plane.
- double **DistanceFrom** (const SpaceVect &p) const
- double **Distance** (const SpaceVect &p) const
- bool HasIntersection (const Line &l) const

  Line is not parallel to plane and not in plane, thus must have an intersection.
- SpaceVect Intersection (const Line &l) const

  Actual intersection, returns (0,0,0) if not possible.
- ostream & Print (ostream &os) const

  Show human-readable version of the class.

Public Attributes

- DirectionVect nv

  Normal vector to plane.
- double d

4.8.1 Detailed Description

A plane in 3-D space and intersections.

The plane is internally defined by its normal direction vector plus a constant, allowing for easy calculation of the distance of any points from the plane.

4.8.2 Member Data Documentation

4.8.2.1 d

```
double Plane::d
```

In plane definition: nv[0]∗x + nv[1]∗y + nv[2]∗z + d = 0 for any point (x,y,z) being on the plane, within numerical precision.

The documentation for this class was generated from the following file:

- IactLightEmission.hh

4.9 Run Class Reference

Run class takes care of global initialization and winding-down/cleanup as well as advancing from one event to the next.

`#include <IactLightEmission.hh>`

Public Member Functions

- Run (int rn=1, double h=2000e2, int atm=1, double r=12.5e2)

  Run class constructor.
- Run (int rn, double h, const char *atmprof, double r=12.5e2)

  Run class constructor.
- Run (int rn, double h, const string &atmprof, double r=12.5e2)

  Run class constructor.
- void SetOutput (const char *fn)

  We need a file name for the output.
- void **SetOutput** (const string &fn)
- void SetDirection (double stheta, double sphi)
- void **SetDirection** (const DirectionVect &d)
- void SetPrimary (int prmid)

  Report a primary particle type as requested (where applicable)
- void **SetPrimary** (const string &prmnam)
- void SetEnergy (double E_gev)

  For some applications we may even report a useful energy for the primary.
- void AddTelescope (const TelPos &tp)

  If the first telescope should not be at (0,0,0) or if you need more than one:
- void AddHistory (const char *txt)

  Simplified function for adding history lines:
- void **AddHistory** (const string &txt)
- void **AddCmdLineHistory** (int argc, char **argv)
- void SetCore (double xcm, double ycm)

  Report given core position in event header.
- void SetScat (int n, double r, double y)

  Set random array scattering.
- void SetRefIdx (double n)

  Set a fixed index of refraction rather than deriving it from atmospheric profile.
- void SetBunchSize (double s)

  Report maximum bunch size according to application; has no practical consequences.
- void NextEvent ()

  Start of next event in run class.
- size_t NumTel () const

  Return number of configured "telescopes".
- const TelPos & Telescope (size_t itel) const

  Return telescope position.
- double Area () const

  Sum of telescope sphere fiducial cross section areas.
- double Area (size_t itel) const

  Individual telescope cross section.
- double ObsLev () const

Return configured observation level [cm] a.s.l.

- int Atmosphere () const

  Return configured atmospheric profile number.
- int RunNumber () const

  Return configured run number.
- int EventNumber () const

  Return current event number.
- double Rmax () const

  Return radius containing all fiducial spheres.
- int **Primary** () const
- int **Energy** () const
- double RefIdx (double dz) const

  Index of refraction at given altitude difference above observation level.
- double RefIdx () const

  Index of refraction at observation level.
- double Density (double dz) const

  Density [g/cm^3] at given altitude difference above observation level.
- double Density () const

  Density [g/cm^3] at observation level.
- double AirLightSpeed () const

  Propagation speed of light in air at observation level.
- double AirLightSpeed (double dz) const

  Propagation speed of light in air above observation level.
- double MeanAirLightSpeed (double dz) const

  Approximate mean propagation speed of light in air from start to observation level.

4.9.1 Detailed Description

Run class takes care of global initialization and winding-down/cleanup as well as advancing from one event to the next.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 Run() [1/3]

```
Run::Run (
            int rn = 1,
            double h = 2000e2,
            int atm = 1,
            double r = 12.5e2 )
```

Run class constructor.

Parameters

rn	Run number.
h	Height of observation level a.s.l. [cm].
atm	Atmospheric profile table number -> atmprofd.dat.
r	Default radius of telescope fiducial sphere [cm].

4.9.2.2 Run() [2/3]

```
Run::Run (
            int rn,
            double h,
            const char * atmprof,
            double r = 12.5e2 )
```

Run class constructor.

Parameters

rn	Run number.
h	Height of observation level a.s.l. [cm].
atmprof	Atmospheric profile table given by name.
r	Default radius of telescope fiducial sphere [cm].

4.9.2.3 Run() [3/3]

```
Run::Run (
            int rn,
            double h,
            const string & atmprof,
            double r = 12.5e2 )
```

Run class constructor.

Parameters

rn	Run number.
h	Height of observation level a.s.l. [cm].
atmprof	Atmospheric profile table given by name.
r	Default radius of telescope fiducial sphere [cm].

4.9.3 Member Function Documentation

4.9.3.1 NextEvent()

```
void Run::NextEvent ( )
```

Start of next event in run class.

Starting the next event includes emitting any header material data blocks for the first event and it includes wrapping up the previous event, including writing its photon bunches, before emitting the header of the new event.

4.9.3.2 SetCore()

```
void Run::SetCore (
            double xcm,
            double ycm )
```

Report given core position in event header.

Set a 'core position' info in the event header. It does not shift any 'telescope' positions or light source positions but changes event header data only in order to enhance consistency between fake muons (or other Cherenkov emitting light sources) and actual CORSIKA muon simulations.

Parameters

xcm	Shower core or track impact position x value at observation level [cm].
ycm	Shower core or track impact position y value at observation level [cm].

4.9.3.3 SetDirection()

```
void Run::SetDirection (
            double stheta,
            double sphi )  [inline]
```

Set the 'shower direction' to report, keeping in mind that the CORSIKA azimuth reports as 0 when coming from South, pi/2 from East, ...

The documentation for this class was generated from the following files:

- IactLightEmission.hh
- IactLightEmission.cc

4.10 SimpleLightSource Class Reference

Simple light source: monochromatic, isotropic, pulse shape is Gaussian or top hat. In addition: point source, stationary.

`#include <IactLightEmission.hh>`

Public Member Functions

- **SimpleLightSource** (const SpaceVect &ls_pos, double ls_wlen=400., double ls_fwhm=0.0, bool gs=true)
- void **SetPosition** (const SpaceVect &ls_pos)
- void **SetWavelength** (double ls_wlen)
- void **SetPulseFwhm** (double ls_fwhm)
- void **SetGaussianShape** (bool ls_gs)
- const SpaceVect & **Position** () const
- double **Wavelength** () const
- double **PulseFwhm** () const
- bool **HasGaussianPulse** () const
- int Emit (const Run &run, double photons, double bsize, double when)

  Emit the requested number of photons in bunches of the requested size at the requested time. Depending on the distance between light source and detector fiducial sphere(s) and the solid angle actually subtended, the number of bunches that need to be written might be lower than expected from looking at photons and bsize alone.

4.10.1 Detailed Description

Simple light source: monochromatic, isotropic, pulse shape is Gaussian or top hat. In addition: point source, stationary.

4.10.2 Member Function Documentation

4.10.2.1 Emit()

```
int SimpleLightSource::Emit (
              const Run & run,
              double photons,
              double bsize,
              double when )
```

Emit the requested number of photons in bunches of the requested size at the requested time. Depending on the distance between light source and detector fiducial sphere(s) and the solid angle actually subtended, the number of bunches that need to be written might be lower than expected from looking at photons and bsize alone.

Parameters

run	Run class
photons	Number of photons that would be emitted into 4 pi steradian (isotropic).
bsize	Intended bunch size. A conservative approach would use a value of 1.0. Any value below one will be replaced by 1.0.
when	Nominal emission time, with individual photon bunches obtaining a random time value to reproduce the configured pulse shape, on average. Unit: nanoseconds.

Returns

0 (OK)

The documentation for this class was generated from the following files:

- IactLightEmission.hh
- IactLightEmission.cc

4.11 SpaceVect Class Reference

A 3-D spatial with some arithmetic operations.

```
#include <IactLightEmission.hh>
```

Public Member Functions

- **SpaceVect** (double xp, double yp, double zp)
- SpaceVect **operator+** (const SpaceVect &sv2) const
- SpaceVect **operator-** (const SpaceVect &sv2) const
- SpaceVect **operator**∗ (double f) const
- SpaceVect & operator+= (const SpaceVect &sv2)

  Translation by adding a second vector.
- SpaceVect & operator-= (const SpaceVect &sv2)

  Translation by subtracting a second vector.
- SpaceVect & operator∗= (double f)

  Scaling the vector with a factor.
- bool IsOrigin () const

  Is this a zero length vector (origin) ?
- double Length () const

  Length of space vector (= distance to origin)
- double LengthSqr () const

  Square of length of space vector.
- double DistanceTo (const SpaceVect &sv2) const

  Distance as a length. v1.Distance(v2) is equivalent to (v2-v1).Length() but faster.
- double DistanceFrom (const SpaceVect &p) const

  DistanceTo(), DistanceFrom(), and Distance() are the same thing.
- double **Distance** (const SpaceVect &p) const
- SpaceVect & Rotate_z (double phi)

  Rotate counter-clockwise around given axis (active rotation); modifies the vector.
- SpaceVect & **Rotate_x** (double phi)
- SpaceVect & **Rotate_y** (double phi)
- SpaceVect & Rotate (double alpha, double beta, double gamma)

  Active rotation with Euler angles in standard x convention (sequence: z,x',z'')
- double SProd (const SpaceVect &sv2) const

  Scalar (inner) product of two vectors.
- double operator∗ (const SpaceVect &sv2) const

  v1∗v2 denotes a scalar product. Use v1.SProd(v2) if this is confusing.
- SpaceVect VProd (const SpaceVect &sv2) const

  Vector (outer) product of two vectors.
- SpaceVect operator/ (const SpaceVect &sv2) const

  v1/v2 denotes a vector product. Use v1.VProd(v2) if this is confusing.
- double TProd (const SpaceVect &sv2, const SpaceVect &sv3) const

  Triple product of three vectors. Returns the volume enclosed.
- ostream & Print (ostream &os) const

  Readable output, including type and units of variable.

Public Attributes

- double **x**
- double **y**
- double z

  North, West, Altitude a.s.l. [cm], public for efficienc access.

4.11.1 Detailed Description

A 3-D spatial with some arithmetic operations.

The SpaceVect class is just a 3-tuple of x/y/z coordinates. The class also covers translations, scaling, rotation as well as scalar, vector, and triple products.

The documentation for this class was generated from the following file:

- IactLightEmission.hh

4.12 TelPos Class Reference

Telescope position as in the CORSIKA inputs card TELESCOPE, includes radius of sphere.

`#include <IactLightEmission.hh>`

Public Member Functions

- **TelPos** (double rtel)
- **TelPos** (double xtel, double ytel, double ztel, double rtel=0.)
- **TelPos** (const SpaceVect &xyz, double rtel=0.)
- SpaceVect Position () const

  Get sphere center position.
- double Radius () const

  Get sphere radius.
- double Area () const

  Get sphere cross section area.
- ostream & Print (ostream &os) const

  Show human-readable version of the class.

Public Attributes

- double x

  X coordinate of the fiducial sphere center.
- double y

  Y coordinate of the fiducial sphere center.
- double z

  Z coordinate of the fiducial sphere center.
- double r

  Radius of the fiducial sphere.
- double A

  Cross section area of sphere (pi∗r^2).

4.12.1 Detailed Description

Telescope position as in the CORSIKA inputs card TELESCOPE, includes radius of sphere.

The documentation for this class was generated from the following file:

- IactLightEmission.hh

Chapter 5

File Documentation

5.1 fake-muon.cc File Reference

Program to generate photon data similar to single muons in CORSIKA, using the CORSIKA/IACT interface.

```
#include "IactLightEmission.hh"
```

Functions

- void **syntax** (int argc, char **argv, int iarg)
- int main (int argc, char **argv)

  Main program.

5.1.1 Detailed Description

Program to generate photon data similar to single muons in CORSIKA, using the CORSIKA/IACT interface.

Compile with (will actually need paths to source files):

```
g++ -DCTA_PROD3 -DIACT_NO_GRID -Wall -Wextra  -o fake-muon \
   fake-muon.cc IactLightEmission.cc  iact.c io_simtel.c eventio.c \
   fileopen.c warning.c  straux.c atmo.c sampling.c \
   rndm2.c sim_absorb.c -lm
```

Run as:

```
./fake-muon [ options see --help ]
```

Process the resulting 'fake-muon.iact.gz' through the usual telescope simulation, e.g.

```
/bin/rm -f fake-muon.simtel.gz; zcat fake-muon.iact.gz | \
   ./sim_telarray -c cfg/CTA/CTA-ULTRA6-MST-NectarCam.cfg -DNUM_TELESCOPES=1 -Icfg/CTA \
   -C Altitude=2150 -C iobuf_maximum=1000000000 \
   -C maximum_telescopes=1 -C trigger_telescopes=1 \
   -C atmospheric_transmission=atm_trans_2150_1_10_0_0_2150.dat \
   -C telescope_theta=0 -C telescope_phi=0 -C power_law=2.68 \
   -C output_file=fake-muon.simtel.gz -
```

and plot the resulting camera signal trace images with

```
read_cta -p fake-muon.ps --plot-without-reco fake-muon.simtel.gz
```

Author

Konrad Bernloehr

5.1.2 Function Documentation

5.1.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

Main program.

< Index of refraction

5.2 ff-1m.cc File Reference

Flat-fielding light source simulation for single-reflector cameras, using the CORSIKA/IACT interface, simulating light emitted from a flasher supposed to be in the center of the dish. The resulting photon bunches can be simulated by sim_telarray with the "Bypass_Optics=2" (or 1 if truely 1M) configuration to bypass reflection on the primary mirror.

```
#include "IactLightEmission.hh"
```

Functions

- void **syntax** (int argc, char ∗∗argv, int iarg=0, const string &msg="")
- int **main** (int argc, char ∗∗argv)

5.2.1 Detailed Description

Flat-fielding light source simulation for single-reflector cameras, using the CORSIKA/IACT interface, simulating light emitted from a flasher supposed to be in the center of the dish. The resulting photon bunches can be simulated by sim_telarray with the "Bypass_Optics=2" (or 1 if truely 1M) configuration to bypass reflection on the primary mirror.

Compile with (will actually need paths to source files):

```
g++ -DCTA_PROD3 -DIACT_NO_GRID -Wall -Wextra  -o ff-1m \
   ff-1m.cc IactLightEmission.cc  iact.c io_simtel.c eventio.c \
   fileopen.c warning.c  straux.c atmo.c sampling.c \
   rndm2.c sim_absorb.c -lm
```

Run as:

```
ff-1m [ options ]
```

See "ff-1m --help" for a list of known options and default values.

Process the resulting 'ff-gct.dat.gz' through the usual telescope simulation, partially bypassing ray-tracing in the telescope, e.g.

```
/bin/rm ff-1m.simtel.gz; zcat ff-1m.dat.gz | \
   ./sim_telarray -c cfg/CTA/CTA-ULTRA6-MST-FlashCam.cfg -DNUM_TELESCOPES=1 -Icfg/CTA \
   -C Bypass_Optics=1 \
   -C Altitude=2100 -C iobuf_maximum=1000000000 -C maximum_telescopes=1 \
   -C atmospheric_transmission=atm_trans_2100_1_10_0_0_2100.dat \
   -C telescope_theta=0 -C telescope_phi=0 -C power_law=2.5 \
   -C output_file=ff-1m.simtel.gz
```

and plot the resulting camera signal trace images with

```
read_cta -p ff-1m.ps ff-1m.simtel.gz
```

or

```
read_cta -u -r 4 --tailcuts 100000,200000 -p ff-1m.ps ff-1m.simtel.gz
```

if you want to avoid the overlay of significant pixels and Hillas ellipse.

Author

Konrad Bernloehr

5.3 ff-gct.cc File Reference

Flat-fielding light source simulation for the GCT (CHEC) cameras, using the CORSIKA/IACT interface, simulating light emitted from four flashers supposed to be in the corners of the camera, here in a coordinate system relative to the center of the secondary mirror. The resulting photon bunches can be simulated by sim_telarray with the "Bypass_Optics=1" configuration to bypass reflection on the primary mirror but include reflection on the secondary mirror.

```
#include "IactLightEmission.hh"
```

Functions

- void **syntax** (int argc, char **argv, int iarg=0, const string &msg="")
- int **main** (int argc, char **argv)

5.3.1 Detailed Description

Flat-fielding light source simulation for the GCT (CHEC) cameras, using the CORSIKA/IACT interface, simulating light emitted from four flashers supposed to be in the corners of the camera, here in a coordinate system relative to the center of the secondary mirror. The resulting photon bunches can be simulated by sim_telarray with the "Bypass_Optics=1" configuration to bypass reflection on the primary mirror but include reflection on the secondary mirror.

Compile with (will actually need paths to source files):

```
g++ -DCTA_PROD3 -DIACT_NO_GRID -Wall -Wextra  -o ff-gct \
   ff-gct.cc IactLightEmission.cc  iact.c io_simtel.c eventio.c \
   fileopen.c warning.c  straux.c atmo.c sampling.c \
   rndm2.c sim_absorb.c -lm
```

Run as:

```
ff-gct [ options ]
```

See "ff-gct --help" for a list of known options and default values.

Process the resulting 'ff-gct.dat.gz' through the usual telescope simulation, partially bypassing ray-tracing in the telescope, e.g.

```
/bin/rm ff-gct.simtel.gz; zcat ff-gct.dat.gz | \
   ./sim_telarray -c cfg/CTA/CTA-ULTRA6-SST-GCT-S.cfg -DNUM_TELESCOPES=1 -Icfg/CTA \
   -C Bypass_Optics=1 \
   -C Altitude=2100 -C iobuf_maximum=1000000000 -C maximum_telescopes=1 \
   -C atmospheric_transmission=atm_trans_2100_1_10_0_0_2100.dat \
   -C telescope_theta=0 -C telescope_phi=0 -C power_law=2.5 \
   -C FADC_BINS=128 -C DISC_BINS=128
   -C output_file=ff-gct.simtel.gz
```

and plot the resulting camera signal trace images with

```
read_cta -p ff-gct.ps ff-gct.simtel.gz
```

or

```
read_cta -u -r 4 --tailcuts 100000,200000 -p ff-gct.ps ff-gct.simtel.gz
```

if you want to avoid the overlay of significant pixels and Hillas ellipse.

Author

Konrad Bernloehr

5.4 fpls.cc File Reference

Focal plane lightsource example program for artificial light sources using the CORSIKA/IACT interface.

```
#include "IactLightEmission.hh"
```

Functions

- int **main** ()

5.4.1 Detailed Description

Focal plane lightsource example program for artificial light sources using the CORSIKA/IACT interface.

Compile with (will actually need paths to source files):

```
g++ -DCTA_PROD3 -DIACT_NO_GRID -Wall -Wextra  -o fpls \
   fpls.cc IactLightEmission.cc  iact.c io_simtel.c eventio.c \
   fileopen.c warning.c  straux.c atmo.c sampling.c \
   rndm2.c sim_absorb.c -lm
```

Run as:

```
./fpls
```

Process the resulting 'fpls.dat.gz' through the usual telescope simulation, bypassing ray-tracing in the telescope, e.g.

```
/bin/rm fpls.simtel.gz; zcat fpls.dat.gz | \
   ./sim_telarray -c cfg/CTA/CTA-ULTRA6-MST-NectarCam.cfg -DNUM_TELESCOPES=1 -Icfg/CTA \
   -C Bypass_Optics=2 \
   -C Altitude=2100 -C iobuf_maximum=1000000000 -C maximum_telescopes=1 \
   -C atmospheric_transmission=atm_trans_2100_1_10_0_0_2100.dat \
   -C telescope_theta=0 -C telescope_phi=0 -C power_law=2.68 \
   -C FADC_BINS=128 -C DISC_BINS=128
   -C output_file=fpls.simtel.gz
```

and plot the resulting camera signal trace images with

```
read_cta -p fpls.ps fpls.simtel.gz
```

The enlarged FADC_BINS and DISC_BINS is only necessary where the rotation period is basically as large as the readout period. For single-reflector telescopes it makes no difference of Bypass_Optics=2 or Bypass_Optics=1.

Author

Konrad Bernloehr

5.5 IactLightEmission.cc File Reference

Simulate an artifical light source instead of running CORSIKA and write the resulting photon bunches in the same format as with shower simulation going through the IACT/atmo interface.

```
#include "IactLightEmission.hh"
#include "rpolator.h"
#include <time.h>
#include <sys/types.h>
#include <sys/time.h>
#include <unistd.h>
#include <ctype.h>
#include <stdexcept>
#include "fileopen.h"
#include "rndm2.h"
```

Classes

- struct photon_bunch

Macros

- #define **UNUSED**(x) UNUSED_ ## x

Typedefs

- typedef struct photon_bunch **Bunch**

Functions

- void **rmmard_** (double *a, int *b, int *UNUSED(c))
- double **rhof_** (double *height)
- double **thick_** (double *height)
- double **heigh_** (double *thick)
- void random_initialize ()
- void **strip_comments** (char *line)
- int **read_table_v** (const char *fname, vector< double > &col1, vector< double > &col2)
- int **read_table_v** (const string &fname, vector< double > &col1, vector< double > &col2)
- double rpol_v (const vector< double > &x, const vector< double > &y, double xp)
- int make_random_ipol_table_v (const vector< double > &x, const vector< double > &y, vector< double > &xr, vector< double > &yr, double xlow_user=0., double xhigh_user=0.)

  Make fast lookup table for rather any number distribution.
- int ls_verbose ()
- int ls_set_verbose (int v)
- double random_from_ipol_table_v (const vector< double > &xr, const vector< double > &yr, double randnr)

  Fast random number table lookup.
- void **SaveCommandLine** (int argc, char **argv)
- vector< string > **GetCommandLine** ()
- string **GetCommandLineString** ()
- DirectionVect **IsotropicRandomEmission** ()
- DirectionVect **IsotropicRandomEmissionUp** ()
- DirectionVect **IsotropicRandomEmissionDown** ()
- DirectionVect **IsotropicRandomEmission** (double maxang, const DirectionVect &central_dir)
- DirectionVect **RandomEmission** (double angle, const DirectionVect &central_dir, double phi_from=0., double phi_to=2.*M_PI)
- DirectionVect **InvertForRandomEmission** (const DirectionVect &central_dir, const DirectionVect &emit_dir)
- double **refidx** (double alt)
- double strtof (const string &str)

  STL string equivalent to C library atof.
- string **strip_whitespace** (const string &text)
- vector< double > dbl_explode (const string &text, const string &delim)

  Given a text string and an optional list of separators, explode it into a vector of doubles.

5.5.1 Detailed Description

Simulate an artifical light source instead of running CORSIKA and write the resulting photon bunches in the same format as with shower simulation going through the IACT/atmo interface.

The resulting data file will be in the same format as CORSIKA IACT output and can thus be processed with sim_↩ telarray or any other program handling the same type of data.

Author

Konrad Bernloehr

5.5.2 Function Documentation

5.5.2.1 dbl_explode()

```
vector<double> dbl_explode (
            const string & text,
            const string & delim )
```

Given a text string and an optional list of separators, explode it into a vector of doubles.

Simplified replacement for stdtools function.

5.5.2.2 ls_set_verbose()

```
int ls_set_verbose (
            int v )
```

Set verbose flag from code rather than environment variable.

5.5.2.3 ls_verbose()

```
int ls_verbose ( )
```

Verbose output from LightSource code or not.

5.5.2.4 make_random_ipol_table_v()

```
int make_random_ipol_table_v (
            const vector< double > & x,
            const vector< double > & y,
            vector< double > & xr,
            vector< double > & yr,
            double xlow_user = 0.,
            double xhigh_user = 0.  )
```

Make fast lookup table for rather any number distribution.

The following procedures can be used for relatively fast production of random numbers according to a given table of (not to be normalized) probability values. Tails of a distribution should be sufficiently well reproduced within the limits imposed a) by the chosen generator of flat random numbers (really double, i.e. 48-bit abscissa, or effectively float, i.e. 24-bit abscissa) and b) by the assumption that the probability is constant within each bin of the input table.

The bin limits are calculated with the assumption of equidistant bins where the abscissa values of the input table give the middle of each bin. This must hold at least for the two lowest and the two highest bins.

Parameters

x	X coordinates of tabulated 'curve'
y	Y coordinates of tabulated 'curve'
xr	Vector of returned X values
yr	Vector of returned Y values (normalized to [0:1])

Returns

0 (o.k.), -1 (error)

5.5.2.5 random_from_ipol_table_v()

```
double random_from_ipol_table_v (
            const vector< double > & xr,
            const vector< double > & yr,
            double randnr )
```

Fast random number table lookup.

Get a random number from a lookup table as created by make_random_ipol_table() for a tabulated probability distribution.

Parameters

xr	Table as returned from make_random_ipol_table()
yr	Table as returned from make_random_ipol_table()
randnr	A uniform pseudorandom number in [0:1]

Returns

random number according to table

5.5.2.6 random_initialize()

```
void random_initialize ( )
```

Initialize our random number generator using system-provided sources of randomness.

5.5.2.7 rpol_v()

```
double rpol_v (
            const vector< double > & x,
            const vector< double > & y,
            double xp )
```

Interface from a pair of C++ vectors to the old C-based interpolation.

5.6 lactLightEmission.hh File Reference

Class definitions and common inline code for the lactLightEmission package.

```
#include "initial.h"
#include "io_basic.h"
#include "mc_tel.h"
#include "iact.h"
#include "atmo.h"
#include <vector>
#include <string>
#include "rndm2.h"
#include <iostream>
#include "fileopen.h"
```

Classes

- class SpaceVect

  A 3-D spatial with some arithmetic operations.
- class DirectionVect

  A normalized direction vector.
- class Line

  A line is defined by a point on the line and the direction vector along the line.
- class Plane

  A plane in 3-D space and intersections.
- class TelPos

  Telescope position as in the CORSIKA inputs card TELESCOPE, includes radius of sphere.
- class Run

  Run class takes care of global initialization and winding-down/cleanup as well as advancing from one event to the next.
- class SimpleLightSource

  Simple light source: monochromatic, isotropic, pulse shape is Gaussian or top hat. In addition: point source, stationary.
- class BackgroundLightSource

  A light source class with diffuse emission characteristics.
- class LightSource

  A much more generic type of light source than the SimpleLightSource.

Functions

- ostream & operator<< (ostream &os, const SpaceVect &t)

  Non-member output operator for SpaceVect.
- double SProd (const SpaceVect &sv1, const SpaceVect &sv2)

  Non-member scalar product of two SpaceVect.
- SpaceVect VProd (const SpaceVect &sv1, const SpaceVect &sv2)

  Non-member vector product of two SpaceVect.
- double TProd (const SpaceVect &sv1, const SpaceVect &sv2, const SpaceVect &sv3)

  Non-member triple product of three SpaceVect.
- SpaceVect operator* (double d, const SpaceVect &sv)

  Non-member scaling of a SpaceVect.

- ostream & operator<< (ostream &os, const DirectionVect &t)

  Non-member print function.
- SpaceVect operator* (double d, const DirectionVect &dv)

  Multiplication the other way round is mapped to member function.
- double SProd (const DirectionVect &dv1, const DirectionVect &dv2)

  Non-member functions for scalar product of two direction vectors, basically the cosine of the angle between them.
- double SProd (const SpaceVect &sv1, const DirectionVect &dv2)

  The scalar product between a SpaceVect and a DirectionVect includes the SpaceVect, in addition to the angle cosine.
- double **SProd** (const DirectionVect &dv1, const SpaceVect &sv2)
- SpaceVect VProd (const DirectionVect &dv1, const DirectionVect &dv2)
- SpaceVect VProd (const SpaceVect &sv1, const DirectionVect &dv2)

  Non-member function for the vector product between a SpaceVect and a DirectionVect.
- SpaceVect **VProd** (const DirectionVect &dv1, const SpaceVect &sv2)
- ostream & operator<< (ostream &os, const Line &l)

  Non-member print function.
- ostream & operator<< (ostream &os, const Plane &pl)

  Non-member print function.
- double Distance (const SpaceVect &sv1, const SpaceVect &sv2)

  Distance between to points (non-member function).
- double Distance (const SpaceVect &sv, const Line &l)

  Shortest distance between point and line (non-member function).
- double Distance (const Line &l, const SpaceVect &sv)

  Shortest distance between point and line, alternate order (non-member function).
- double Distance (const SpaceVect &sv, const Plane &a)

  Shortest distance between point and plane (non-member function).
- double Distance (const Plane &a, const SpaceVect &sv)

  Shortest distance between point and plane, alternate order (non-member function).
- double Distance (const Line &l1, const Line &l2)

  Shortest distance between two lines (non-member function).
- ostream & operator<< (ostream &os, const TelPos &t)

  Non-member print function.
- int ls_verbose ()
- int ls_set_verbose (int v)
- void **SaveCommandLine** (int argc, char **argv)
- vector< string > **GetCommandLine** ()
- string **GetCommandLineString** ()
- DirectionVect **IsotropicRandomEmission** ()
- DirectionVect **IsotropicRandomEmissionUp** ()
- DirectionVect **IsotropicRandomEmissionDown** ()
- DirectionVect **IsotropicRandomEmission** (double maxang, const DirectionVect &central_dir)
- DirectionVect **RandomEmission** (double angle, const DirectionVect &central_dir)
- vector< double > dbl_explode (const string &text, const string &delim=",")

  Simplified replacement for stdtools function.
- ostream & **operator**<< (ostream &os, const vector< double > &t)

5.6.1 Detailed Description

Class definitions and common inline code for the lactLightEmission package.

Author

Konrad Bernloehr

5.6.2 Function Documentation

5.6.2.1 dbl_explode()

```
vector<double> dbl_explode (
            const string & text,
            const string & delim )
```

Simplified replacement for stdtools function.

Simplified replacement for stdtools function.

5.6.2.2 ls_set_verbose()

```
int ls_set_verbose (
            int v )
```

Set verbose flag from code rather than environment variable.

5.6.2.3 ls_verbose()

```
int ls_verbose ( )
```

Verbose output from LightSource code or not.

5.6.2.4 VProd()

```
SpaceVect VProd (
            const DirectionVect & dv1,
            const DirectionVect & dv2 )  [inline]
```

Non-member function for the vector product between two DirectionVect. Since the result is not normalized, it is returned as a SpaceVect.

5.7 ile_version.hh File Reference

This is the lactLightEmission package version 2023-01-31.

Macros

- #define **ILE_VERSION** "2023-01-31"

5.7.1 Detailed Description

This is the IactLightEmission package version 2023-01-31.

Version

2023-01-31

5.8 ls-beam.cc File Reference

Simulate light from a laser beam crossing the field of view of a telescope, some of the laser light getting scattered along the beam. The modelling of the laser beam and it attenuation along the path is very simplified, and for performance reasons an isotropic emission angle distribution is to be preferred for now. The resulting photon bunches can be simulated by sim_telarray with the (default) "Bypass_Optics=0" configuration, not bypassing any optics raytracing.

```
#include "IactLightEmission.hh"
#include "fileopen.h"
```

Functions

- void **syntax** (int argc, char **argv, int iarg=0, const string &msg="")
- int solve_quadratic_equation (double a, double b, double c, double *x1, double *x2)
- int **main** (int argc, char **argv)

5.8.1 Detailed Description

Simulate light from a laser beam crossing the field of view of a telescope, some of the laser light getting scattered along the beam. The modelling of the laser beam and it attenuation along the path is very simplified, and for performance reasons an isotropic emission angle distribution is to be preferred for now. The resulting photon bunches can be simulated by sim_telarray with the (default) "Bypass_Optics=0" configuration, not bypassing any optics raytracing.

Compile with (will actually need paths to source files):

```
g++ -DCTA_PROD3 -DIACT_NO_GRID -Wall -Wextra  -o ls-beam \
   ls-beam.cc IactLightEmission.cc  iact.c io_simtel.c eventio.c \
   fileopen.c warning.c  straux.c atmo.c sampling.c \
   rndm2.c sim_absorb.c -lm
```

Run as:

```
ls-beam [ options ]
```

See "ls-beam --help" for a list of known options and default values.

Example use:

```
ls-beam --laser-position 1000,0,0 --laser-theta 60 --laser-phi 180 \
    --telescope-theta 30 --telescope-phi 0 --fov 15
```

Process the resulting 'ls-beam.dat.gz' through the usual telescope simulation, partially bypassing ray-tracing in the telescope, e.g.

```
/bin/rm ls-beam.simtel.gz; zcat ls-beam.iact.gz | \
   ./sim_telarray -c cfg/CTA/CTA-ULTRA6-MST-FlashCam.cfg -DNUM_TELESCOPES=1 -Icfg/CTA \
   -C Altitude=2150 -C iobuf_maximum=1000000000 -C maximum_telescopes=1 \
   -C atmospheric_transmission=atm_trans_2150_1_10_0_0_2150.dat \
   -C telescope_theta=30 -C telescope_phi=0 -C power_law=2.5 \
   -C output_file=ls-beam.simtel.gz
```

and plot the resulting camera signal trace images with

```
read_cta -p ls-beam.ps ls-beam.simtel.gz
```

or

```
read_cta -u -r 4 --tailcuts 100000,200000 -p ls-beam.ps ls-beam.simtel.gz
```

if you want to avoid the overlay of significant pixels and Hillas ellipse.

Author

Konrad Bernloehr

5.8.2 Function Documentation

5.8.2.1 solve_quadratic_equation()

```
int solve_quadratic_equation (
              double a,
              double b,
              double c,
              double * x1,
              double * x2 )
```

Solve a quadratic equation a∗x∗∗2 + b∗x + c = 0

Returns

0: OK, we have solutions, -1: no solution

5.9 ls-geo-test.cc File Reference

A few tests for the LightEmission geometry classes.

```
#include "IactLightEmission.hh"
```

Functions

- int **main** ()

5.9.1 Detailed Description

A few tests for the LightEmission geometry classes.

Compile with (will actually need paths to source files):

```
g++ -DCTA_PROD3 -DIACT_NO_GRID -Wall -Wextra  -o ls-geo-test \
   ls-geo-test.cc IactLightEmission.cc  iact.c io_simtel.c eventio.c \
   fileopen.c warning.c  straux.c atmo.c sampling.c \
   rndm2.c sim_absorb.c -lm
```

Run as:

```
./ls-geo-test
```

Author

Konrad Bernloehr

5.10 ls-test.cc File Reference

Test program for artificial light sources using the CORSIKA/IACT interface.

```
#include "IactLightEmission.hh"
```

Functions

- int **main** ()

5.10.1 Detailed Description

Test program for artificial light sources using the CORSIKA/IACT interface.

Compile with (will actually need paths to source files):

```
g++ -DCTA_PROD3 -DIACT_NO_GRID -Wall -Wextra  -o ls-test \
   ls-test.cc IactLightEmission.cc  iact.c io_simtel.c eventio.c \
   fileopen.c warning.c  straux.c atmo.c sampling.c \
   rndm2.c sim_absorb.c -lm
```

Run as:

```
./ls-test
```

Author

Konrad Bernloehr

5.11 ls-test2.cc File Reference

Test program for artificial light sources using the CORSIKA/IACT interface.

```
#include "IactLightEmission.hh"
```

Functions

- int **main** ()

5.11.1 Detailed Description

Test program for artificial light sources using the CORSIKA/IACT interface.

Compile with (will actually need paths to source files):

```
g++ -DCTA_PROD3 -DIACT_NO_GRID -Wall -Wextra  -o ls-test2 \
   ls-test2.cc IactLightEmission.cc  iact.c io_simtel.c eventio.c \
   fileopen.c warning.c  straux.c atmo.c sampling.c \
   rndm2.c sim_absorb.c -lm
```

Run as:

```
./ls-test2
```

Author

Konrad Bernloehr

5.12 nsbls.cc File Reference

Nightsky background lightsource example program for artificial light sources using the CORSIKA/IACT interface.

```
#include "IactLightEmission.hh"
```

Functions

- void **syntax** (int argc, char **argv, int iarg)
- int **main** (int argc, char **argv)

5.12.1 Detailed Description

Nightsky background lightsource example program for artificial light sources using the CORSIKA/IACT interface.

Compile with (will actually need paths to source files):

```
g++ -DCTA_PROD3 -DIACT_NO_GRID -Wall -Wextra  -o nsbls \
   nsbls.cc IactLightEmission.cc  iact.c io_simtel.c eventio.c \
   fileopen.c warning.c  straux.c atmo.c sampling.c \
   rndm2.c sim_absorb.c -lm
```

Run as:

```
    ./nsbls
Process the resulting 'nsbls.dat.gz' through the usual telescope
simulation, bypassing ray-tracing in the telescope, e.g.
@code{.sh}
   /bin/rm nsbls.simtel.gz; zcat nsbls.iact.gz | \
      ./sim_telarray -c cfg/CTA/CTA-ULTRA6-MST-NectarCam.cfg -DNUM_TELESCOPES=1 -Icfg/CTA \
      -C Altitude=2150 -C iobuf_maximum=1000000000 -C maximum_telescopes=1 \
      -C atmospheric_transmission=atm_trans_2150_1_10_0_0_2150.dat \
      -C telescope_theta=0 -C telescope_phi=0 \
      -C output_file=nsbls.simtel.gz
```

and plot the resulting map of registered true photo-electrons with

```
read_cta -p nsbls.ps \
         --plot-with-true-pe \
         --plot-with-pixel-pe \
         --plot-with-sum-only \
         --plot-without-reco \
     nsbls.simtel.gz
```

Author

Konrad Bernloehr

5.13 octo.cc File Reference

Superluminal octocopter example program for artificial light sources using the CORSIKA/IACT interface.

```
#include "IactLightEmission.hh"
```

Functions

- int **main** ()

5.13.1 Detailed Description

Superluminal octocopter example program for artificial light sources using the CORSIKA/IACT interface.

This program is for demonstration purposes only and has no practical use.

If not build by default, use the Makefile dependencies with
```
make octo
```

or compile it manually (will actually need paths to source files and for header files included, here assuming we are under the sim_telarray directory):
```
g++ -I. \
    -I../corsika -I../common -I../../hessioxxx/include -I../../stdtools \
   -DIACT_NO_GRID  -DCORSIKA_VERSION=6999 -DWITH_STDTOOLS \
   -DWITH_RPOLATOR_SEPARATE -DWITH_MC_ATMPROF_SEPARATE \
   -Wall -Wextra  -o octo \
   octo.cc IactLightEmission.cc \
   ../corsika/iact.c ../corsika/atmo.c ../corsika/sampling.c \
   ../../hessioxxx/src/io_simtel.c \
   ../../hessioxxx/src/eventio.c \
   ../../hessioxxx/src/fileopen.c \
   ../../hessioxxx/src/warning.c \
   ../../hessioxxx/src/straux.c \
   ../../hessioxxx/src/mc_atmprof.c \
   ../common/rndm2.c ../common/rpolator.c ../common/sim_absorb.c \
   -lm
```

Run as:
```
./octo
```

Process the resulting 'octo.iact.gz' through the usual telescope simulation, e.g.
```
../sim_telarray -c cfg/CTA/CTA-PROD6-SST.cfg -DNUM_TELESCOPES=1 -Icfg/CTA \
-C Altitude=2150 -C iobuf_maximum=1000000000 -C maximum_telescopes=1 \
-C atmospheric_transmission=atm_trans_2150_1_10_0_0_2150.dat \
-C telescope_theta=0 -C telescope_phi=0 -C power_law=2.68 \
-C output_file=octo.simtel.zst octo.iact.gz
```

and plot the resulting camera signal trace images.

Author

Konrad Bernloehr

5.14 pixled.cc File Reference

A program to set an assumed LED in front of a specified subset of pixels in a camera as defined for sim_telarray. The main purpose is to test that trigger definitions work as intended.

```
#include "IactLightEmission.hh"
#include <straux.h>
#include <list>
```

Classes

- struct PixPos
- struct PixLed

Functions

- string **clean_trigger_list** (const string &plist, bool keep_slaves)
- int read_cam (const string &fname, vector< PixPos > &cam, bool keep_slaves, bool read_triggers)
- int get_simple_use (const char *use, vector< PixLed > &pl)
- int get_complete_use (const char *use, vector< PixLed > &pl)
- int **get_trg_use_random** (vector< PixLed > &pl)
- int **get_trg_use** (size_t j, vector< PixLed > &pl)
- void syntax (int argc, char **argv, int iarg=0, const string &msg="")

  Report syntax errors and program usage.
- void emit_all_events (vector< LightSource > &vls, size_t nreq, const vector< size_t > &selected, const vector< size_t > &available, Run &run, double nphot_per_pos, double bunch_size, double dt, vector< LightSource > &vlss, const vector< size_t > &nsl, const vector< size_t > fsl)

  Recursive function to produce as many events as possible pixel combinations.
- int main (int argc, char **argv)

  The main program.

Variables

- vector< string > **trglist**

5.14.1 Detailed Description

A program to set an assumed LED in front of a specified subset of pixels in a camera as defined for sim_telarray. The main purpose is to test that trigger definitions work as intended.

Compile with (will actually need paths to source files):

```
g++ -DCTA_PROD3 -DIACT_NO_GRID -Wall -Wextra  -o pixled \
   pixled.cc IactLightEmission.cc  iact.c io_simtel.c eventio.c \
   fileopen.c warning.c  straux.c atmo.c sampling.c \
   rndm2.c sim_absorb.c -lm
```

Run as:

```
./pixled [ --camera-file <name> ] [ --use <pixel-list ]
    [ --require <npix> ] [ --random <npix> ] [ --events <nev> ] \
    ... [ -o <output_file> ]
```

Note that the name of the camera definition file and the pixel list are necessary. If the number of required or random pixels is missing, all specified pixels will get fired. Process the resulting output file through the usual telescope simulation, bypassing ray-tracing in the telescope, e.g.

```
/bin/rm pixled.simtel.gz; zcat pixled.dat.gz | \
   ./sim_telarray -c cfg/CTA/CTA-PROD4-LST.cfg -DNUM_TELESCOPES=1 -Icfg/CTA \
   -C Bypass_Optics=2 \
   -C Altitude=2150 -C iobuf_maximum=1000000000 -C maximum_telescopes=1 \
   -C atmospheric_transmission=atm_trans_2150_1_10_0_0_2150.dat \
   -C telescope_theta=0 -C telescope_phi=0 -C power_law=2.68 \
   -C output_file=pixled.simtel.gz
```

and plot the resulting camera signal trace images with

```
read_cta -p pixled.ps pixled.simtel.gz
```

Author

Konrad Bernloehr

5.14.2 Function Documentation

5.14.2.1 get_complete_use()

```
int get_complete_use (
            const char * use,
            vector< PixLed > & pl )
```

Complete list of pixels to illuminate and how to illuminate can only be obtained from a separate file (one line per pixel).

5.14.2.2 get_simple_use()

```
int get_simple_use (
            const char * use,
            vector< PixLed > & pl )
```

Get a simple list of pixels to use, that is illuminate by light. The simple list is provided on the command line and its only pixel-specific option is to say that a pixel is always required.

5.14.2.3 read_cam()

```
int read_cam (
            const string & fname,
            vector< PixPos > & cam,
            bool keep_slaves,
            bool read_triggers )
```

Read a camera definition file as it gets used by sim_telarray, extracting the x/y projected positions of each pixel center.

5.15 xyls.cc File Reference

Test program for artificial light sources using the CORSIKA/IACT interface. This program is not intended to be of practical use.

```
#include "IactLightEmission.hh"
```

Functions

- int **main** (int argc, char **argv)

5.15.1 Detailed Description

Test program for artificial light sources using the CORSIKA/IACT interface. This program is not intended to be of practical use.

Compile with (will actually need paths to source files):

```
g++ -DCTA_PROD3 -DIACT_NO_GRID -Wall -Wextra  -o xyls \
   xyls.cc IactLightEmission.cc  iact.c io_simtel.c eventio.c \
   fileopen.c warning.c  straux.c atmo.c sampling.c \
   rndm2.c sim_absorb.c -lm
```

Run as:

```
./xyls [ -r rcam ] [ -x xpos ] [ -y ypos ] [ -h height] [ -o fname ]
```

Author

Konrad Bernloehr

5.16 xyzls.cc File Reference

Lightsource example program for artificial light sources using the CORSIKA/IACT interface.

```
#include "IactLightEmission.hh"
#include "straux.h"
```

Functions

- int **read_telpos_file** (const char *fname, bool corsika_fmt, vector< TelPos > &tpv)
- int **main** (int argc, char **argv)

5.16.1 Detailed Description

Lightsource example program for artificial light sources using the CORSIKA/IACT interface.

Compile with (will actually need paths to source files):

```
g++ -DCTA_PROD3 -DIACT_NO_GRID -Wall -Wextra  -o xyls \
   xyzls.cc IactLightEmission.cc  iact.c io_simtel.c eventio.c \
   fileopen.c warning.c  straux.c atmo.c sampling.c \
   rndm2.c sim_absorb.c -lm
```

Run as:

```
./xyzls
```

Process the resulting 'xyzls.dat.gz' through the usual telescope simulation, bypassing ray-tracing in the telescope, e.g.

```
/bin/rm xyzls.simtel.gz; zcat xyzls.dat.gz | \
   ./sim_telarray -c cfg/CTA/CTA-ULTRA6-MST-NectarCam.cfg -DNUM_TELESCOPES=1 -Icfg/CTA \
 [ -C Bypass_Optics=2 ] \
   -C Altitude=2150 -C iobuf_maximum=1000000000 \
   -C maximum_telescopes=1 -C trigger_telescopes=1 \
   -C atmospheric_transmission=atm_trans_2150_1_10_0_0_2150.dat \
   -C telescope_theta=0 -C telescope_phi=0 -C power_law=2.68 \
   -C output_file=xyzls.simtel.gz
```

and plot the resulting camera signal trace images with

```
read_cta -p xyzls.ps xyzls.simtel.gz
```

For single-reflector telescopes it makes no difference if Bypass_Optics=2 or Bypass_Optics=1.

Author

Konrad Bernloehr

Index