G4SiPM Documentation

Release 1.0.0

ntim

Mar 28, 2017

Contents

The purpose of the G4SiPM simulation package is the integration of a detailed SiPM simulation into Geant4 (https://github.com/Geant4/geant4) which is a framework widely used in particle physics for detector simulations.

Further information can be found here: http://dx.doi.org/10.1016/j.nima.2015.01.067

CHAPTER 1

Documentation contents

Getting started

Introduction

G4Sipm uses the CMake to configure a build system for compiling. For more details on Geant4 with CMake itself, the Geant4 documentation should be consulted.

Supported and Tested Platforms

Scientific Linux CERN 6 with gcc 4.7.X (also 64bit).

Required packages

The following packages are required for a successful build of G4Sipm.

- Geant ~4.10.0
- Boost ~1.41.0 (required for the reading of properties files)

The tilde signifies that G4Sipm is tested with this version, higher versions should work as well.

Install Geant4 according to the official Building and Installation Guide with CLHEP and Qt if possible.

Optional

Optional dependencies may be installed beforehand to enable certain features.

- ROOT ~5.34 (export of simulation data)
- SQlite3 ~3.6.20 (export of simulation data)
- Pyhton ~2.7.6 (batch invocation of the simulation)

- Matplotlib ~1.2.1 (creation of plots)
- Sphinx ~1.2.2 (building of this documentation)
- Breathe (inclusion of C++ source code in the documentation)
- Doxygen ~1.6.1 (creation of the C++ source code documentation)

Included packages

Several packages are included in the G4Sipm source tree.

- GoogleTest 1.7.0 (automated testing)
- GCovr 2.5-prerelease (generating test coverage reports)
- Jansson 2.6 (export of simulation data)

Building G4Sipm

For illustration only, this guide will assume G4Sipm resides in a directory named */path/to*.

The next step is to create a directory in which we clone the G4Sipm repository:

```
$ cd /path/to
$ mkdir g4sipm
$ cd g4sipm
```

Clone repository

Clone the repository to a directory *source*:

```
$ git clone https://github.com/ntim/g4sipm.git
```

Now, create a directory to configure and run the build and store the build products. This directory should **not** be the same as, or inside, the source directory. In this guide, we create this build directory alongside our source directory:

```
$ mkdir build
```

Configuring the build

To configure the build, i.e. the creation of Unix Makefiles with CMake, change into the build directory and run CMake:

```
$ cd build
$ cmake ../source
```

G4Sipm build options

By running:

```
$ ccmake ../source
```

the CMake build variables can be inspected and modified. The relevant build variables and their purpose are stated in the following:

CMAKE_BUILD_TYPE It is adviced to set this variable to *RelWithDebInfo* to enable debugging with low performance impact which is useful for application developement.

CMAKE_INSTALL_PREFIX Not used.

WITH_COVERAGE_ANALYSIS Enables the coverage analysis.

WITH_ROOT Enable *ROOT* support for the export of simulation data.

WITH_SQLITE Enable *SQLite* support for the export of simulation data.

Compiling of G4Sipm

After the configuration has run, CMake will have generated Unix Makefiles for compiling G4Sipm. To run the build, simply execute make in the build directory:

```
$ make -jN
```

where N is the number of parallel jobs you require (e.g. if your machine has a dual core processor, you could set N to 2). The build will now run, and will output information on the progress of the build and current operations. If you need more output to help resolve issues or simply for information, run make as:

```
$ make -jN VERBOSE=1
```

Additional Make targets

The following additional Make targets are available

help Print all Make targets

runG4sipmTest Runs the GoogleTest tests of the G4Sipm library.

runSampleTest Runs the GoogleTest tests of the sample simulation.

doc Builds this documentation

doc_doxygen Builds a doxygen documentation

G4Sipm directory structure

After finishing the compilation of G4Sipm, the following directories (besides the CMake directories) should appear in your build directory

- doc
- externals
- g4sipm
- sample

whereas the *g4sipm* directory contains the G4Sipm library and the *sample* directory contains a simple simulation which is intended to provide a quick-start for Geant4 beginners.

Developing of G4Sipm using Eclipse

G4Sipm has been developed using Eclipse IDE for C/C++ Developers which I **strongly** recommend for everyone which thinks of developing a serious application.

CMake provides a special Makefile generator:

```
$ cmake ../source -G"Eclipse CDT4 - Unix Makefiles" <other-options>
```

which automatically creates a *.project* and a *.cproject* file in the build directory which will be understood by Eclipse. Import G4Sipm into Eclipse via *File -> Import... -> Existing Projects into Workspace*.

Sample simulation

This is a simple simulation which is intended to provide a quick-start for Geant4 beginners. It resides in the *sample* directory of G4Sipm.

Invocation

A new simulation can be started with:

```
$ ./sample
```

which also creates a GUI window in which you should see a graphical representation of the simulation.

Command line arguments

The following command line arguments are available

help Produce help message.

mac Macro input file which will be processed without creating the GUI.

output The path to the output directory

model The model string of the SiPM model to simulate. Can be one of "HamamatsuS1036211100", "HamamatsuS1036233100", "HamamatsuS1036233050", "HamamatsuS10985100" and "HamamatsuS12651050" or a path to a config file.

housing The type of housing in which the SiPM is packaged. Can be one of "default", "ceramic" and "smd".

GUI mode

If invoked without the *mac* command line argument, the simulation runs in the GUI mode. A QT window with a 3D visualization is created. For more details, please refer to the Geant4 visulization documentation.

Batch mode

If the *mac* command line argument is supplied, the simulation runs in batch mode:

```
./sample --mac run.mac
```

After executing all commands stored in the *macro*-file, the program exits. For more details on *macro*-files, please refer to the Geant4 documentation. This is particularly useful for executing many simulations consecutively as e.g. a dynamic range simulation.

The most basic example of a mac file is the *run.mac*:

```
# Generate event.
/run/beamOn 1
```

Run scripts

The repository of G4Sipm also contains several so-called *run-scripts*. It is a collection of python scripts which invoke the simulation with temporarily created *macro*-files for the generation of particular plots. The run scripts should be invoked from the *sample* directory:

```
python run/pde.py
```

pde.py

Varies the photon wavelength for the investigation of the photon detection efficiency as a function of the photon wavelength. The output is written to *./results/pde*. A temporary *macro*-file will be created:

```
/g4sipm/filter/timing 0
/g4sipm/digitize/hits 0
/g4sipm/digitize/trace 0
/ps/energy/eMin %g eV
/ps/energy/eMax %g eV
/ps/nParticles %d
/run/beamOn 1
```

with the string replacements "%g" and "%d".

relative_pde.py

Similar to the *pde.py* script, this script investigates the photon detection efficiency as a function of the angle of incidence of the light. The output is written to *./results/relative_pde*. Instead of only changing the angle of incidence, the light source is rotated around the y-axis whereas the surface normal points to the center of the SiPM.

Plot scripts

G4Sipm also provides several python scripts for plotting simulation data with matplotlib. Depending on the activated export method (Json, ROOT or SQlite), the plot scripts can be found in either one of the following directories:

```
plots/json
plots/root
plots/sqlite
```

The scripts are thought to be invoked from the sample simulation directory:

```
python plots/sqlite/pde.py
```

Concepts

This section describes the core concepts which have been used to build G4Sipm.

Properties files

The class *Properties* provides the parsing of simple properties files.

Definition of a property

A simple property can be defined by:

```
key: value
```

or with a unit (the CLHEP system of units is used):

```
key: value * unit
```

Also available as a unit is the "%" sign.

Definition of a properties table

You can think of many properties which have to be supplied as a function of a certain variable. For this, a table can be defined. The header reads:

```
table-name: tabular
   field1   field2   field3   ....
```

Or again with units::

table-name: tabular field1 / unit field2 / unit field3 / unit ...

whereas the columns should be separated by tabulators to increased readability. Additionally, the properties could be edited with a spreadsheet software when importing / exporting as CSV file.

Example

For SiPMs, the photon detection efficiency is a function of the photon wavelength:

```
photonDetectionEfficiency: tabular
   entry wavelength / nm   efficiency / %
   1  319.490  9.18187959596
   1  320.882  9.79401373737
   1  322.274  10.2530812121
   1  323.666  12.7016177778
   1  326.450  14.3849535354
   1  327.842  14.9970876768
   1  329.234  16.2213559596
   1  330.626  16.833490101
   ...
```

Source

class `Properties`

This class can parse ".properties" files containing key-value-pairs and tables.

Uses boost::regex to match the contents.

Public Types

typedef std::map<std::string, std::vector<double>> **`tabular`**

Type represents a table in the property file.

Public Functions

bool **`load`**(std::string *filename*)

Return boolean - true if the file could be parsed completely.

Parameters

- `filename`: - the properties file name.

double **`getNumber`**(std::string *key*) **const**

Return double - the number.

Parameters

- `key`: - the key.

std::string **`getString`**(std::string *key*) **const**

Return string - the string.

Parameters

- `key`: - the key.

Properties::*tabular* **`getTabular`**(std::string *key*) **const**

Return map - the tabular data.

Parameters

- `key`: - the key.

bool **`containsNumber`**(std::string *key*) **const**

Return boolean - true if the key-number-pair exists.

Parameters

- `key`: - the key.

bool **`containsString`**(std::string *key*) **const**

Return boolean - true if the key-string-pair exists.

Parameters

- `key`: - the key.

bool **`containsTabular`**(std::string *key*) **const**

Return boolean - true if the key-tabular-pair exists.

Parameters

- `key`: - the key.

std::string **`toString`**`()` **const**

Return prints all properties to a string which is compatible with the file format.

void **`print`**`()` **const**
Print all read properties.

std::string **`getFilename`**`()` **const**

Return string - the file name.

G4Sipm model

The *`G4SipmModel`* is one of the most important classes of G4Sipm. It stores all properties of the SiPM.

Working conditions

The model stores the working conditions of the SiPM: environmental temperature and bias voltage.

The environmental temperature can be set via the *`G4SipmModel::setTemperature()`* function, the bias voltage with *`G4SipmModel::setBiasVoltage()`*. The over-voltage of the SiPM is the difference between the bias voltage and the breakdown voltage

SiPM properties

The model also stores the following properties of SiPMs

- breakdown voltage
- number of cells
- cell pitch
- geometrical fill factor
- thermal noise rate
- effective dead time of the cells during the avalanche breakdown
- recovery time of the cells
- optical crosstalk probability
- time constants and probabilities of the long and the short afterpulsing component
- variance of the gain
- photon detection efficiency as a function of the photon wavelength
- thickness of the entrance window

It also holds references to a *`G4SipmGainMapModel`* and a *`G4SipmVoltageTraceModel`*.

SiPM properties as a function of the temperature and over-voltage

Please refer to the `G4SipmConfigFileModel` documentation.

Source

Inherited by `G4SipmConfigFileModel`, `G4SipmGenericSipmModel`, `HamamatsuS1036211100`, `HamamatsuS1036233050`, `HamamatsuS1036233100`, `HamamatsuS10985100` and `HamamatsuS12651050`.

class `G4SipmModel`

SiPM model.

Subclassed by G4SipmConfigFileModel, G4SipmGenericSipmModel, HamamatsuS1036211100, HamamatsuS12573100C, HamamatsuS12573100X, HamamatsuS12651050

Public Functions

`G4SipmModel`(*G4SipmGainMapModel* *gainMapModel, *G4SipmVoltageTraceModel* *voltageTraceModel)

Constructor.

Parameters

- `gainMapModel`: - the gain map model.
- `voltageTraceModel`: - the voltage trace model.

double **`getPitch`**() **const**

Return double - the complete pitch of the SiPM.

CLHEP::Hep2Vector **`getCellPosition`**(G4SipmCellId *cellId*) **const**

Return Hep2Vector - the position relative to the center.

Parameters

- `cellId`: - the id of the cell.

G4SipmCellId **`getCellId`**(**const** double *x*, **const** double *y*, bool *respectFillFactor* = false) **const**

Return G4SipmCellId - the corresponding cell id.

Parameters

- `x`: - the x coordinate.
- `y`: - the y coordinate.
- `respectFillFactor`: - set true to return an invalid cell id if the coordinates are not in an active area.

bool **`isValidCellId`**(**const** G4SipmCellId *cellId*) **const**

Return bool - true if the cellId is valid.

Parameters

- `cellId`: - the cellId.

double **getGain**(**const** G4SipmCellId *cellId*) **const**

Return double - the gain of the cell.

Parameters

- `cellId`: - the cellId.

double **getOverVoltage**() **const**

Return double - the current overvoltage.

virtual std::string **getName**() **const** = 0

Return string - the name of the model.

virtual double **getBreakdownVoltage**() **const** = 0

Return double - the current breakdown voltage.

virtual unsigned int **getNumberOfCells**() **const** = 0

Return unsigned int - the number of cells.

virtual double **getCellPitch**() **const** = 0

Return double - the pitch of a single cell.

virtual double **getThermalNoiseRate**() **const** = 0

Return double - the thermal noise rate.

virtual double **getDeadTime**() **const** = 0

Return double - the dead time of a SiPM cell.

virtual double **getRecoveryTime**() **const** = 0

Return double - the recovery time of the SiPM cell gain.

virtual double **getCrossTalkProbability**() **const** = 0

Return double - the crosstalk probability.

virtual double **getApProbLong**() **const** = 0

Return double - the probability for long time constant afterpulses.

virtual double **getApProbShort**() **const** = 0

Return double - the probability for short time constant afterpulses.

virtual double **getApTauLong**() **const** = 0

Return double - the time constant of long time constant afterpulses.

virtual double **getApTauShort**() **const** = 0

Return double - the time constant of short time constant afterpulses.

virtual double **getFillFactor**() **const** = 0

Return double - the fill factor.

virtual double **getGainVariation**() **const** = 0

Return double - the gain variance.

virtual double **getPhotonDetectionEfficiency**(double *wavelength*) **const** = 0

Return double - the photon detection efficiency.

Parameters

- `wavelength`: - the photon wavelength.

G4Material ***getMaterial**() **const**

Return G4Material - the material of the chip.

double **getThickness**() **const**

Return double - the thickness of the chip.

G4Material ***getWindowMaterial**() **const**

Return G4Material - the material of the window.

double **getWindowThickness**() **const**

Return double - the window thickness.

double **getTemperature**() **const**

Return double - the current temperature of the chip.

void **setTemperature**(double *temperature*)

Parameters

- `temperature`: - the temperature to set.

double **getBiasVoltage**() **const**

Return double - the bias voltage.

void **setBiasVoltage**(double *biasVoltage*)

Parameters

- `biasVoltage`: - the biasVoltage to set.

G4SipmGainMapModel ***getGainMapModel**() **const**

Return *G4SipmGainMapModel* - the gain map model.

G4SipmVoltageTraceModel ***getVoltageTraceModel**() **const**

Return *G4SipmVoltageTraceModel* - the voltage trace model.

G4Sipm config file model

The *`G4SipmModel`* is one of the most important classes of G4Sipm. It stores all properties of the SiPM.

Working conditions

The model stores the working conditions of the SiPM: environmental temperature and bias voltage.

The environmental temperature can be set via the *`G4SipmModel::setTemperature()`* function, the bias voltage with *`G4SipmModel::setBiasVoltage()`*. The over-voltage of the SiPM is the difference between the bias voltage and the breakdown voltage

SiPM properties

The model also stores the following properties of SiPMs

- breakdown voltage
- number of cells
- cell pitch
- geometrical fill factor
- thermal noise rate
- effective dead time of the cells during the avalanche breakdown
- recovery time of the cells
- optical crosstalk probability
- time constants and probabilities of the long and the short afterpulsing component
- variance of the gain
- photon detection efficiency as a function of the photon wavelength
- thickness of the entrance window

It also holds a reference to *G4SipmGainMapModel* and *G4SipmVoltageTraceModel*.

Config file example

```
# Properties file for a Hamamtsu S10362-11-100C SiPM.
#
# References:
# P. Hallen, Determination of the Recovery Time of Silicon Photomultipliers, bachelor␣
↪thesis, III. Phys. Inst. A, RWTH Aachen University, Sep 2011.
# M. Lauscher, Characterisation Studies of Silicon Photomultipliers for the Detection␣
↪of Fluorescence Light from Extensive Air Showers, III. Phys. Inst. A, RWTH Aachen␣
↪University, Jan 2012.
#

name:        Hamamatsu_S10362-11-100C

# Setup parameters.
temperature:        20. * Celsius
biasVoltage:        73. * V
deadTime: 3. * picosecond
gainVariation: 1. * %

# Geometry.
thickness:        0.1 * mm
numberOfCells:        100
cellPitch:        0.1 * mm
fillFactor:        78.5 * %
windowThickness: 0.5 * mm
windowRefractiveIndex: 1.41

# Voltage trace parameters.
# The amplitude of the pulse.
voltageTrace-amplitude:        14.1e-3 * volt
# The time constants of the rising and the falling component.
```

```
voltageTrace-tauFall:        43.6 * ns
voltageTrace-tauRise:        8.1 * ns
# The time difference between two voltage entries of the trace.
voltageTrace-timeBinWidth:        1. * ns
# Baseline.
voltageTrace-v0:        4.7e-3 * volt
# White noise.
voltageTrace-whiteNoiseSigma:        1e-3 * volt
# Precision in bit of the voltage trace measurement.
voltageTrace-precision:        12

# Operation parameters.
operatingParameters:        tabular
        entry        overVoltage / V        breakDownVoltage / V        temperature /␣
↪Celsius        thermalNoiseRate / Hz        crossTalkProbability /
↪%        afterPulseProbLong / %        afterPulseProbShort /
↪%        afterPulseTauLong / ns        afterPulseTauShort / ns        recoveryTime /
↪ ns
        1        0.8        71.5        25.5        494000        8.63891        9.
↪424030        7.710570        129.4        43.7        41.1
        1        0.9        71.5        25.5        508000        10.4349        11.
↪22517        9.184230        162.0        48.4        41.1
        1        1.0        71.5        25.5        648000        12.1126        13.
↪65793        11.17467        112.8        40.3        41.1
        1        1.1        71.5        25.5        655000        13.7848        16.
↪03894        13.12277        127.9        42.1        41.1
        1        1.2        71.5        25.5        775000        15.4505        18.
↪57405        15.19695        119.2        45.0        41.1
        1        1.3        71.5        25.5        799000        16.9318        21.
↪56655        17.64536        125.9        40.8        41.1
        1        1.4        71.5        25.5        956000        18.1540        23.
↪64654        19.34717        121.3        43.7        41.1
        1        1.5        71.5        25.5        924000        19.3662        26.
↪01786        21.28734        123.1        44.5        41.1
        1        1.6        71.5        25.5        1057000        20.3590        28.
↪65891        23.44820        111.7        40.6        41.1
        1        1.7        71.5        25.5        995000        21.0974        30.
↪57439        25.01541        129.6        44.7        41.1
        1        1.8        71.5        25.5        1103000        21.6948        33.
↪25718        27.21042        110.1        36.0        41.1


# Photon detection efficiency for each entry in tabular "operationParameters".
photonDetectionEfficiency:        tabular
        entry        wavelength / nm        efficiency / %
        1        319.490        9.18187959596
        1        320.882        9.79401373737
        1        322.274        10.2530812121
        1        323.666        12.7016177778
        1        326.450        14.3849535354
        1        327.842        14.9970876768
        1        329.234        16.2213559596
        1        330.626        16.833490101
        1        332.019        18.0576921212
        1        332.019        18.0576921212
        1        333.411        19.281960404
        1        334.803        19.8940945455
        1        336.195        20.9652961616
```

```
        1        337.587        21.577430303
        1        338.979        22.1895644444
        1        340.371        22.8016985859
        1        343.155        23.4138327273
        1        344.548        24.0259668687
        1        345.940        24.4850343434
        1        348.724        25.0971684848
        1        351.508        25.5562359596
        1        355.684        26.015369697
        1        358.469        26.4744371717
...
```

Source

Inherited by `G4SipmConfigFileModel`, `G4SipmGenericSipmModel`, `HamamatsuS1036211100`, `HamamatsuS1036233050`, `HamamatsuS1036233100`, `HamamatsuS10985100` and `HamamatsuS12651050`.

class `G4SipmModel`

SiPM model.

Subclassed by G4SipmConfigFileModel, G4SipmGenericSipmModel, HamamatsuS1036211100, HamamatsuS12573100C, HamamatsuS12573100X, HamamatsuS12651050

Public Functions

`G4SipmModel`(*G4SipmGainMapModel* **gainMapModel*, *G4SipmVoltageTraceModel* **voltageTraceModel*)

Constructor.

Parameters

- `gainMapModel`: - the gain map model.
- `voltageTraceModel`: - the voltage trace model.

double **`getPitch`**() **const**

Return double - the complete pitch of the SiPM.

CLHEP::Hep2Vector **`getCellPosition`**(G4SipmCellId *cellId*) **const**

Return Hep2Vector - the position relative to the center.

Parameters

- `cellId`: - the id of the cell.

G4SipmCellId **`getCellId`**(**const** double *x*, **const** double *y*, bool *respectFillFactor* = false) **const**

Return G4SipmCellId - the corresponding cell id.

Parameters

- `x`: - the x coordinate.
- `y`: - the y coordinate.

- `respectFillFactor`: - set true to return an invalid cell id if the coordinates are not in an active area.

bool **`isValidCellId`**(**const** G4SipmCellId *cellId*) **const**

Return bool - true if the cellId is valid.

Parameters

- `cellId`: - the cellId.

double **`getGain`**(**const** G4SipmCellId *cellId*) **const**

Return double - the gain of the cell.

Parameters

- `cellId`: - the cellId.

double **`getOverVoltage`**() **const**

Return double - the current overvoltage.

virtual std::string **`getName`**() **const** = 0

Return string - the name of the model.

virtual double **`getBreakdownVoltage`**() **const** = 0

Return double - the current breakdown voltage.

virtual unsigned int **`getNumberOfCells`**() **const** = 0

Return unsigned int - the number of cells.

virtual double **`getCellPitch`**() **const** = 0

Return double - the pitch of a single cell.

virtual double **`getThermalNoiseRate`**() **const** = 0

Return double - the thermal noise rate.

virtual double **`getDeadTime`**() **const** = 0

Return double - the dead time of a SiPM cell.

virtual double **`getRecoveryTime`**() **const** = 0

Return double - the recovery time of the SiPM cell gain.

virtual double **`getCrossTalkProbability`**() **const** = 0

Return double - the crosstalk probability.

virtual double **`getApProbLong`**() **const** = 0

Return double - the probability for long time constant afterpulses.

virtual double **`getApProbShort`**() **const** = 0

Return double - the probability for short time constant afterpulses.

virtual double **`getApTauLong`**() **const** = 0

Return double - the time constant of long time constant afterpulses.

virtual double **getApTauShort**() **const** = 0

Return double - the time constant of short time constant afterpulses.

virtual double **getFillFactor**() **const** = 0

Return double - the fill factor.

virtual double **getGainVariation**() **const** = 0

Return double - the gain variance.

virtual double **getPhotonDetectionEfficiency**(double *wavelength*) **const** = 0

Return double - the photon detection efficiency.

Parameters

- `wavelength`: - the photon wavelength.

G4Material ***getMaterial**() **const**

Return G4Material - the material of the chip.

double **getThickness**() **const**

Return double - the thickness of the chip.

G4Material ***getWindowMaterial**() **const**

Return G4Material - the material of the window.

double **getWindowThickness**() **const**

Return double - the window thickness.

double **getTemperature**() **const**

Return double - the current temperature of the chip.

void **setTemperature**(double *temperature*)

Parameters

- `temperature`: - the temperature to set.

double **getBiasVoltage**() **const**

Return double - the bias voltage.

void **setBiasVoltage**(double *biasVoltage*)

Parameters

- `biasVoltage`: - the biasVoltage to set.

G4SipmGainMapModel ***getGainMapModel**() **const**

Return *G4SipmGainMapModel* - the gain map model.

G4SipmVoltageTraceModel ***getVoltageTraceModel**() **const**

Return *G4SipmVoltageTraceModel* - the voltage trace model.

G4Sipm voltage trace model

The voltage trace model is used to mimic the SiPM signals read with an flash ADC or oscilloscope. A pulse is parameterized as a double exponential (falling and rising edge) function.

Voltage trace properties

The following properties can be set:

- Amplitude of 1 p.e. pulse
- The time bin width, i.e. the time difference between two voltage readings
- The time constant τ_{rise} of the rising edge
- The time constant τ_{fall} of the falling edge
- The baseline offset v_0
- The white noise sigma
- The precision of the voltage trace in bits.

Source

class `G4SipmVoltageTraceModel`
Voltage trace model.

TODO: parameters should be overvoltage dependent.

Subclassed by G4SipmConfigFileVoltageTraceModel, G4SipmGenericVoltageTraceModel, HamamatsuS12573100C::VoltageTraceModel, HamamatsuS12573100X::VoltageTraceModel

Public Functions

`G4SipmVoltageTraceModel``()`
Constructor.

double **`pulse`**(**const** double *t*, **const** double *gain* = 1.) **const**
Pulse parameterization (double exponential by default).

Return double - the current amplitude.

Parameters

- `t`: - the time after trigger.
- `gain`: - the gain in percent.

virtual double **`getAmplitude`**`()` **const** = 0

Return double - the amplitude of the pulse.

virtual double **`getTauRise`**`()` **const** = 0

Return double - the rising edge time constant.

virtual double **`getTauFall`**`()` **const** = 0

Return double - the falling edge time constant.

virtual double **getV0** () **const** = 0

Return double - the baseline height.

virtual double **getWhiteNoiseSigma** () **const** = 0

Return double - the white noise variance of the baseline.

virtual int **getPrecision** () **const** = 0

Return int - the precision in bit.

virtual double **getTimeBinWidth** () **const** = 0

Return double - the time bin width.

G4Sipm gain map model

Due to imperfections and electrical noise, the gain is not homogeneous over the whole SiPM surface. Two implementations are available: `G4SipmRandomGainMapModel` and `G4G4SipmGaussianGainMapModel`.

Random gain map model

Picks the gain out of a Gaussian distribution with mean $\mu = 1$ and standard deviation σ equal to the *G4SipmModel::getGainVariation()*. The individual gains stay the same over the whole simulation time.

Gaussian gain map model

Some SiPM models have a lower gain to the edges of the chip. The gain is a 2D Gaussian function $f(x, y)$ with $f(0, 0) = 1 + 3\sigma$ and at in the corners $f(d, d) = 1 - 3\sigma$ with the standard deviation σ equal to the *G4SipmModel::getGainVariation()*.

Source

class G4SipmGainMapModel

Gain map model base class for the G4Sipm model.

Measurements did reveal a non-constant gain of the SiPM cells. This automatically contributes to the width of the finger spectra recorded with an QDC.

TODO: lazy initialization via the refresh mechanism is too dangerous.

Subclassed by G4SipmGaussianGainMapModel, G4SipmRandomGainMapModel

Public Functions

G4SipmGainMapModel ()

Constructor.

virtual void **refresh** (**const** *G4SipmModel* ***const** *model*) = 0

Refresh mechanism.

Parameters

- `model`: - the SiPM model.

virtual bool **`needsRefresh`**`()` = 0

Return bool - true if a refresh is required.

virtual double **`getGain`**`(`**const** G4SipmCellId *cellId*`)` **const** = 0

Return double - the gain in percent.

Parameters

- `cellId`: - the id of the cell.

G4Sipm hit

A *`G4SipmHit`* is created by a *`G4SipmSensitiveDetector`* when a Geant4 step of a photon goes through it. It stores the following properties of the step:

- `G4Sipm` id
- track id
- kinetic energy
- global time
- weight (normally equals 1 but can be used to track photon bunches)
- local position on the SiPM
- absolute position in the simulation world
- momentum
- start position
- start momentum

Please also refer to the Geant4 hits documentation.

G4Sipm hit collection

Each `G4SipmHitCollection` is uniquely identified by the `G4SipmId`. Thus, given the id is zero:

```
g4sipmHits-0
```

Source

class `G4SipmHit`

Hit class for the G4Sipm.

It implements G4VHit::Draw() to visualize hits as red dots on the SiPM surface.

Inherits from G4VHit

Public Functions

G4SipmHit()
Constructor.

G4ParticleDefinition ***getParticleDefinition**() **const**

Return G4ParticleDefinition - the particle definition for the underlying PDG-Id.

double **getEKin**() **const**

Return double - the kinetic energy.

void **setEKin**(double *eKin*)

Parameters

- `eKin`: - the kinetic energy to set.

const CLHEP::Hep3Vector &**getMomentum**() **const**

Return Hep3Vector - the momentum.

void **setMomentum**(**const** CLHEP::Hep3Vector &*momentum*)

Parameters

- `momentum`: - the momentum to set.

int **getPdgId**() **const**

Return int - the PDG particle id.

void **setPdgId**(int *pdgId*)

Parameters

- `pdgId`: - the PDG particle id to set.

const CLHEP::Hep3Vector &**getPosition**() **const**

Return Hep3Vector - the position.

void **setPosition**(**const** CLHEP::Hep3Vector &*position*)

Parameters

- `position`: - the position to set.

G4SipmId **getSipmId**() **const**

Return G4SipmId - the SiPM id to set.

void **setSipmId**(G4SipmId *sipmId*)

Parameters

- `sipmId`: - the SiPM id to set.

int **getTrackId**() **const**

Return int - the track id.

void **setTrackId**(int *trackId*)

Parameters

- `trackId`: - the track id.

const CLHEP::Hep3Vector &**getStartMomentum**() **const**

Return Hep3Vector - the start momentum.

void **setStartMomentum**(**const** CLHEP::Hep3Vector &*startMomentum*)

Parameters

- `startMomentum`: - the start momentum to set.

const CLHEP::Hep3Vector &**getStartPosition**() **const**

Return Hep3Vector - the start position.

void **setStartPosition**(**const** CLHEP::Hep3Vector &*startPosition*)

Parameters

- `startPosition`: - the start position to set.

double **getTime**() **const**

Return double - the time.

void **setTime**(double *time*)

Parameters

- `time`: - the time to set.

double **getWeight**() **const**

Return double - the weight of the particle step.

void **setWeight**(double *weight*)

Parameters

- `weight`: - the weight to set.

const CLHEP::Hep3Vector &**getWorldPosition**() **const**

Return Hep3Vector - the position relative to the origin of the world.

void **setWorldPosition**(**const** CLHEP::Hep3Vector &*worldPosition*)

Parameters

- `worldPosition`: - the position relative to the origin of the world.

G4Sipm sensitive detector

The *G4SipmSensitiveDetector* is automatically created for each `G4Sipm` and uniquely identified by the `G4SipmId`. Thus, given the id is zero:

```
g4sipmSd-0
```

It is attached to the silicon chip representation of the `G4Sipm`. The sensitive detector is responsible for the creation of *G4SipmHit* objects.

To respect the effects of dead space and the photon detection efficiency, each *G4SipmSensitiveDetector* has a *G4SipmSensitiveDetectorFilter*.

Source

class G4SipmSensitiveDetector

This sensitive detector handles the sensitive parts of a G4Sipm and creates *G4SipmHit*.

Registers itself to the G4SDManager. Makes use of the *G4SipmSensitiveDetectorFilter*.

Inherits from G4VSensitiveDetector

Public Functions

G4SipmSensitiveDetector(**const** G4Sipm **sipm*)

Constructor.

Parameters

- `sipm`: - the SiPM to be attached to.

G4Sipm sensitive detector filter

The *`G4SipmSensitiveDetectorFilter`* implements the virtual method:

```
G4bool Accept(const G4Step*)
```

of its base class `G4VSDFilter` which returns true if the photon should be scored by the *`G4SipmSensitiveDetector`* while considering the effects of dead space and the photon detection efficiency.

Dead space

The photon should not hit the dead space on the Sipm. This is checked by the `G4SensitiveDetectorFilter::acceptGeometry()` function.

In G4Sipm, the cells of a Sipm are modeled as squares surrounded by a dead space border.

The dead space check can be enabled/disabled by a macro command:

```
/g4sipm/filter/geometry 1
```

Photon detection efficiency

This is handled by the `G4SensitiveDetectorFilter::acceptPde()` function. The photon detection efficiency (PDE) is a function of the photon wavelength. A random number between zero and one is picked from a uniform distribution. If the random number is smaller than the PDE for the photon wavelength, the method returns true.

To account for the reflectivity of the entrance window of the SiPM, the PDE is divided by its transmittance. The transmittance is calculated from a Fresnel-equation for three-layer transmittance whereas for inclined light the transmittance is calculated for unpolarized light. For perpendicular incidence, the transmittance is calculated for parallel polarization following Jackson's convention.

The PDE check can be enabled/disabled by a macro:

```
/g4sipm/filter/pde 1
```

Source

class `G4SipmSensitiveDetectorFilter`
The detector filter for the *G4SipmSensitiveDetector*.

Handles the loss in sensitivity by the fill factor and the photon detection efficiency.

Inherits from G4VSDFilter

Public Functions

`G4SipmSensitiveDetectorFilter`(**const** G4Sipm **sipm*)
Constructor.

Parameters

- `sipm`: - the SiPM.

`~G4SipmSensitiveDetectorFilter`()

G4bool **`Accept`**(**const** G4Step **step*) **const**

Protected Functions

bool **`acceptPde`**(double *eKin*, double *theta*) **const**
Dices the acceptance according to the PDE.

Return bool - true if the photon should be accepted.

Parameters

- `eKin`: - the energy of the photon.
- `theta`: - the angle of incidence of the photon.

bool **`acceptGeometry`**(double *x*, double *y*) **const**
Checks if the photon hits dead space between the single cells.

Return bool - true if the photon should be accepted.

Parameters

- `x`: - the x position of the photon.
- `y`: - the y position of the photon.

G4Sipm digi

A *`G4SipmDigi`* is created by the *`G4SipmDigitizer`*. It stores the following properties:

- `G4Sipm` id
- `G4SipmCellId`
- The time of the breakdown
- The weight, i.e. the gain of the cell at the breakdown whereas one equals to 1 p.e.

- *G4SipmDigiType*, can be either *UNDEFINED*, *THERMAL*, *PHOTON*, *CROSSTALK* or *AFTERPULSE*

Please also refer to the Geant4 digitization documentation.

G4Sipm digi collection

Each `G4SipmDigiCollection` is uniquely identified by the `G4SipmId`. Thus, given the id is zero:

```
g4sipmDigis-0
```

Source

enum G4SipmDigiType
Enumeration signifying the cause of the cell trigger of the Sipm.

Values:

UNDEFINED

PHOTON

THERMAL

CROSSTALK

AFTERPULSE

class G4SipmDigi
Geant4 Digi for the G4Sipm.

Signifies one cell trigger.

Inherits from G4VDigi

Public Functions

G4SipmDigi()
Constructor.

G4SipmDigi(**const** *G4SipmDigi* &*right*)
Copy constructor.

G4SipmId **getSipmId**() **const**

Return G4SipmId - the SiPM id.

void **setSipmId**(G4SipmId *sipmId*)

Parameters

- `sipmId`: - the SiPM id to set.

G4SipmCellId **getCellId**() **const**

Return G4SipmCellId - the cell id.

void **setCellId**(G4SipmCellId *cellId*)

Parameters

- `cellId`: - the cellId to set.

double **`getTime`**`()` **const**

Return double - the global time of the trigger.

void **`setTime`**(double *time*)

Parameters

- `time`: - the time to set.

G4SipmDigiType **`getType`**`()` **const**

Return G4SipmDigiType - the type of the trigger.

void **`setType`**(*G4SipmDigiType type*)

Parameters

- `type`: - the type of the trigger.

double **`getWeight`**`()` **const**

Return double - the weight of the trigger which is identical to the gain of the cell.

void **`setWeight`**(double *weight*)

Parameters

- `weight`: - the weight of the trigger.

G4Sipm digitizer

The purpose of the *`G4SipmDigitizer`* is to simulate the SiPM on a single cell basis creating a list of cell breakdowns using the `G4SipmHitCollection` and the properties of the *`G4SipmModel`*. Since the digitization in Geant4 takes place after the completion of the particle tracking, in this case ray tracing, all needed information is available. Cell breakdowns or cell triggers are represented by the *`G4SipmDigi`* class. All *`G4SipmHit`* instances are transformed into *`G4SipmDigi`* instances and sorted chronologically into a list.

The digitization can be enabled/disabled with a macro command:

```
/g4sipm/digitize/hits 1
```

In case no *`G4SipmHit`* was created, the creation of dark noise can be enabled/disabled:

```
/g4sipm/noise/ifNoSignal 1
```

Please also refer to the Geant4 digitization documentation.

Thermal noise

Given, the time stamps of the :cpp:class‘G4SipmHit‘s vary between t_0 and t_1, thermal noise is created. To ensure a quasi stable state of the SiPM Monte Carlo model, thermal noise is created in a larger time window. Each cell of the SiPM is allowed to produce a mean number of thermal noise triggers before t_0 and after t_1:

```
/g4sipm/noise/preThermal 3
/g4sipm/noise/postThermal 1
```

The thermal noise creation can be enabled/disabled with a macro command:

```
/g4sipm/digitize/hits 1
```

All thermal noise triggers are added to the chronologically sorted list.

Correlated noise

Once the thermal noise triggers have been created, the chronologically sorted list is traversed. For each trigger, correlated noise, i.e. optical crosstalk and afterpulsing, may be created.

To enable/disable correlated noise, use one of the following macro commands:

```
/g4sipm/noise/afterpulse 1
/g4sipm/noise/crosstalk 1
```

Source

class `G4SipmDigitizer`

Digitizer module which creates cell triggers (*G4SipmDigi*) from hits (*G4SipmHit*).

Inherits from G4VDigitizerModule

Public Functions

`G4SipmDigitizer`(G4Sipm *_sipm_)

Constructor.

Parameters

- `sipm`: - the SiPM.

const G4SipmHitsCollection ***`getHitCollection`**()

Return G4SipmHitsCollection - the hit collection.

G4Sipm ***`getSipm`**() **const**

Return G4Sipm - the SiPM to which the digitizer is attached to.

G4Sipm voltage trace digitizer

The voltage trace digitizer uses the `G4SipmDigiCollection` and the *`G4SipmVoltageTraceModel`* to create the voltage trace of the SiPM.

The digitizaton can be enabled/disabled by a macro command:

```
/g4sipm/digitize/trace 1
```

The digitization of the hits has to be enabled.

Please also refer to the Geant4 digitization documentation.

G4Sipm voltage trace digi collection

The voltage traces are saved as `G4SipmVoltageTraceDigi` in a `G4SipmVoltageTraceDigiCollection`. Each `G4SipmVoltageTraceDigiCollection` is uniquely identified by the `G4SipmId`. Thus, given the id is zero:

```
g4sipmVoltageTraceDigis-0
```

Source

class G4SipmVoltageTraceDigitizer

Digitizer module converting the list of cell triggers (*G4SipmDigi*) into a single voltage trace (G4SipmVoltageTraceDigi).

Inherits from G4VDigitizerModule

Public Functions

G4SipmVoltageTraceDigitizer(G4Sipm *sipm)

Constructor.

Registers itself to Geant4.

Parameters

- `sipm`: - the SiPM instance.

const G4SipmDigiCollection ***getDigiCollection**()

Return G4SipmDigiCollection - the digi collection.

G4Sipm ***getSipm**() **const**

Return G4Sipm - the SiPM instance.

G4Sipm cell fire controller

The cell fire controller is a container which holds the points in time of the last breakdown all SiPM cells. Thus, the cell fire controller can compute the current gain of the cell and can decide whether the cell can break down or not.

The dead time check can be enabled/disable by a macro command:

```
/g4sipm/filter/timing 1
```

Source

class G4SipmCellFireController

Encapsulates the cell firing mechanism.

Should be recreated for each run.

Subclassed by G4SipmEffectiveOvervoltageCellFireController

Public Functions

G4SipmCellFireController(*G4SipmModel* *model, double t0 = 0.)
Constructor.

Parameters

- `model`: - the underlying SiPM model.
- `t0`: - the initial time of all SiPM cells.

bool **fire**(*G4SipmDigi* *d)
Checks whether the trigger can be successfully made or not.

Return bool - true if the trigger could be fired, false if not.

Parameters

- `d`: - the cell trigger digi.

Application development

Apart from the G4Sipm sample simulation, G4Sipm can be easily integrated into existing projects. This guide explains how to incorporate G4Sipm and use it as a library.

It will be assumed that your project also uses Geant4 and the CMake build system with separate source and build directory.

Create a git submodule

Create a directory inside your source directory:

```
$ mkdir externals/g4sipm
```

Create a git submodule with:

```
$ git add submodule ssh://forge.physik.rwth-aachen.de/git/g4sipm.git externals/g4sipm
```

Initialize and update to the latest version:

```
$ git submodule init
$ git submodule update
```

Add G4Sipm to your CMakeLists.txt

To enable the building of G4Sipm add the following lines to the *CMakeLists.txt* of your project:

```
add_subdirectory(externals/g4sipm)
include_directories(externals/g4sipm/g4sipm/include)
include_directories(externals/g4sipm/externals/jansson/src)
include_directories(externals/g4sipm/externals/gtest/include)
```

If you wish to include the export of the simulation data of the sample simulation in your build process, add the following lines:

```
include_directories(externals/g4sipm/sample/include)
set(LIBS ${LIBS} boost_program_options g4sipm g4sipm_sample boost_date_time jansson)
```

Now, link the G4Sipm library to your project by extending the *target_link_libraries* command of your executable:

```
target_link_libraries(target-name g4sipm boost_date_time jansson [other-libraries])
```

and with the sample simulation library:

```
target_link_libraries(target-name g4sipm g4sipm_sample boost_program_options boost_
↪date_time jansson [other-libraries])
```

Creation of SiPMs

New SiPMs can be created and placed in your `G4VUserDetectorConstruction`:

```
G4Sipm* sipm = new G4Sipm(g4sipmModel);
```

This creates a bare silicon chip without a window. The chip can be placed in a package represented by the `G4SipmHousing`:

```
new G4SipmHousing(sipm);
```

The G4Sipm class automatically creates the :cpp:class‘G4SipmSensitiveDetector‘, the :cpp:class‘G4SipmDigitizer‘ and the *G4SipmVoltageTraceDigitizer* and registers the instances to Geant4.

Digitization

Digitization is not automatically performed by Geant4. The following snippet can be used to execute all registered digitizers in your `G4UserEventAction`:

```
G4DCtable* dcTable = digiManager->GetDCtable();
for (int i = 0; i < dcTable->entries(); i++) {
   G4String dmName = dcTable->GetDMname(i);
   G4VDigitizerModule* dm = digiManager->FindDigitizerModule(dmName);
   if (dm) {
      dm->Digitize();
   }
}
```

CHAPTER 2

License

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Index

P

T

U