cfluxim
cosmic ray simulation tool

Kamil Wójcik

University of Silesia

2020

kamil.wojcik@us.edu.pl

Contents

General information

About cfluxim

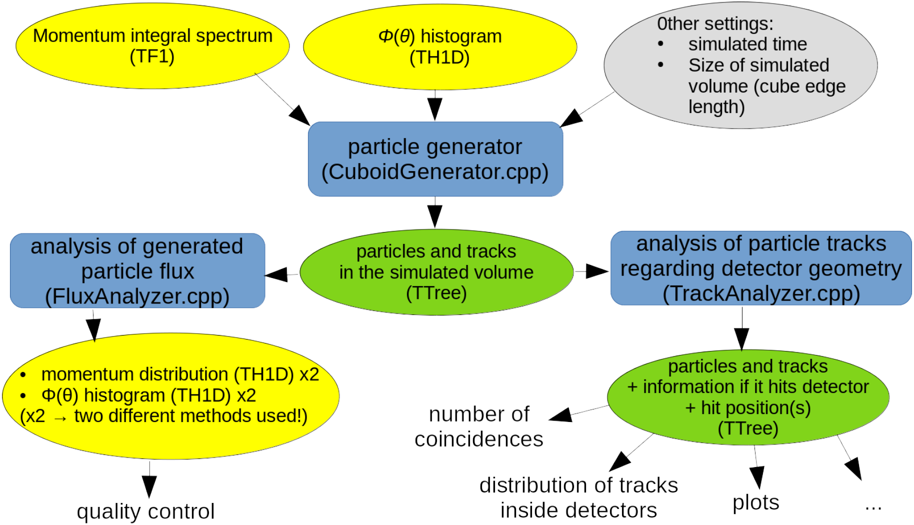
The `cfluxim` project was created to make a simple geometrical simulation of cosmic muons passing through some modules of the MPD detector (NICA MPD website).

- Cosmic muons are generated inside the given cubic volume.
- $\Phi(\theta)$ and momentum distribution of the simulated flux fits the the experimental data with good accuracy.
- For every generated particle, it is checked if it hits the defined detector modules, and the hit position is saved.
- Any energy cutoff can be applied.

General notes

- `cfluxim` uses CERN's ROOT libraries.
- `cfluxim` consists of 3 tools: `CuboidGenerator`, `FluxAnalyzer` and `TrackAnalyzer`.
- Only muons at ground level are implemented, however, implementation of other cosmic ray components is possible.
- `FluxAnalyzer` generates momentum distribution and $\Phi(\theta)$ normalized histogram, so it can be compared wirh the experimental data as a simple quality check.
- `TrackAnalyzer` does the simple geometrical analysis of the tracks, regarding the defined detector geometry. It does not run the full physical analysis as Geant4 does.
- Generated cosmic muons can be, however, put into the Geant4 simulation.

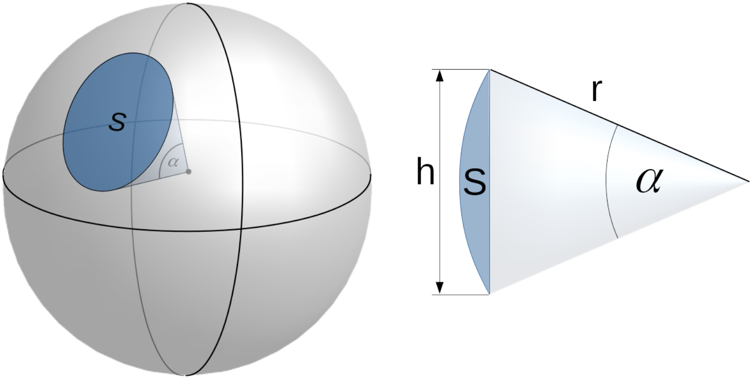
Project scheme



s

Basic definitions

Solid angle

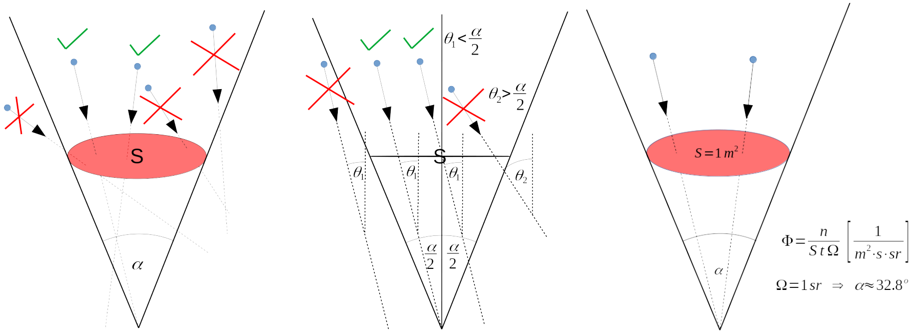


area: $S = 2\pi rh = 2\pi r^2(1 - cos\alpha)$

solid angle: $\Omega = \frac{S}{r^2} = 2\pi(1 - cos\alpha)$ [sr]

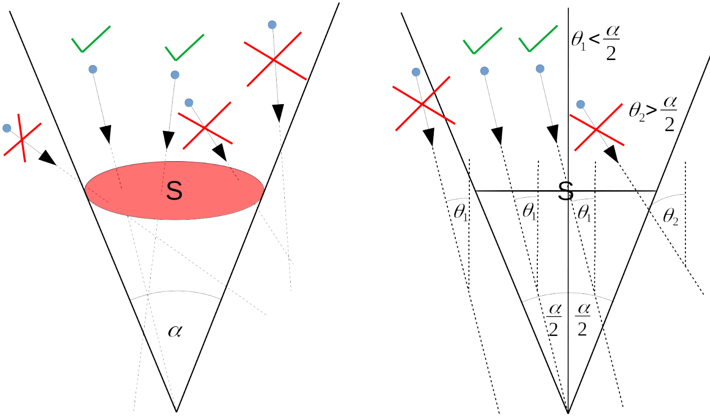
Cosmic ray flux

Cosmic ray flux $=$ *number of particles that come from unit solid angle, passing through unit area, per unit of time*



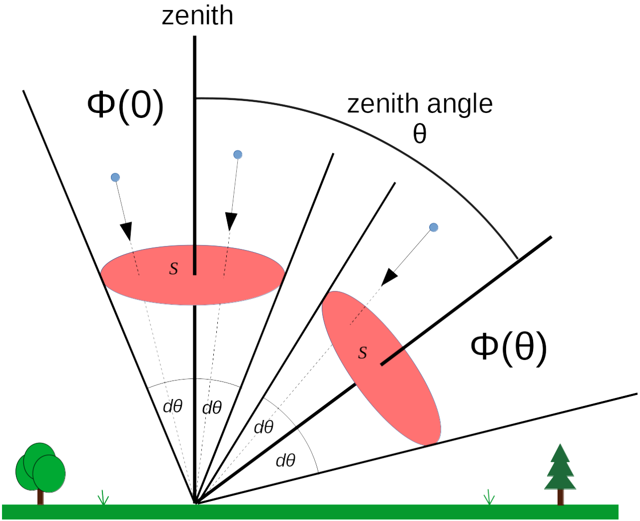
particles coming from a given solid angle passing through area S

standard normalization

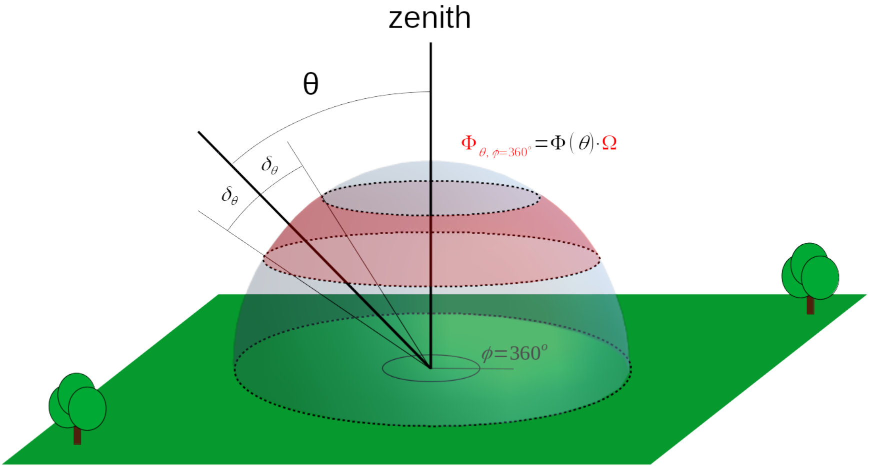


Solid angle limits the *direction* of incoming particle momentum, but not the position on the 'probing area' S. To count a particle as coming from the given solid angle, momentum angular limitations must be fulfilled and the particle must hit the probing area.

Cosmic ray flux dependant on zenith angle



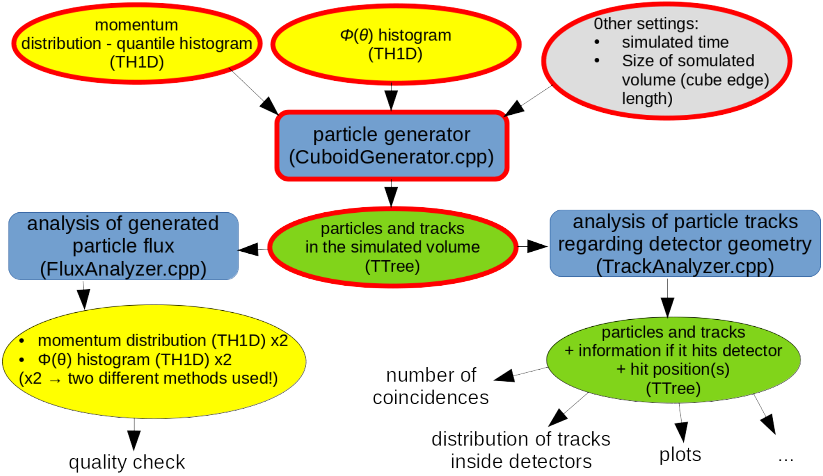
Cosmic ray flux dependant on zenith angle – full azimuth angle case



$$\Omega = 2\pi(cos(\theta - \delta_\theta) - cos(\theta + \delta_\theta))$$

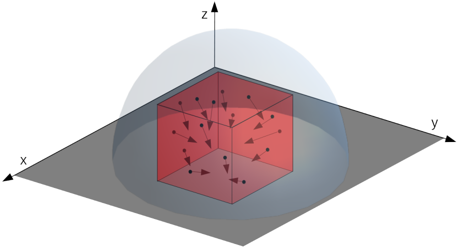
Simulation design

CuboidGenerator



CuboidGenerator – the idea

The idea: generation of particles and its momenta, coming from half-sphere ($\Omega = 2\pi$), that would pass through a cubic volume.



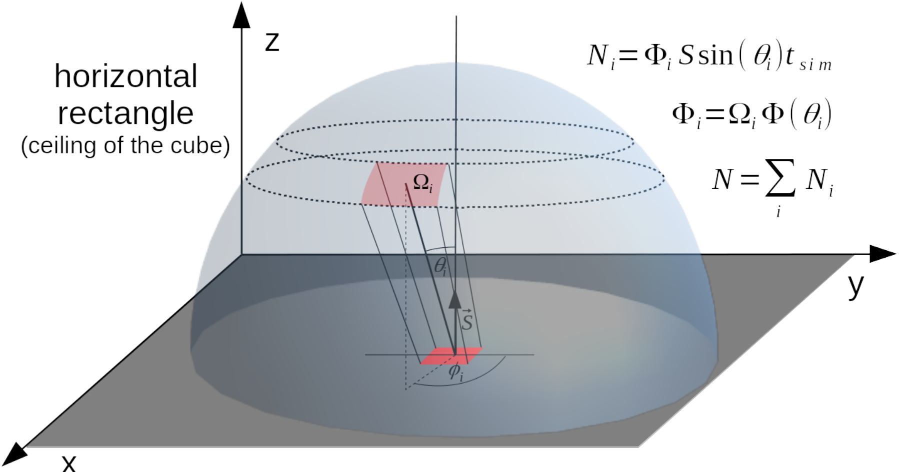
Particle's 'initial' position on the cube wall and its momentum vector define the track inside the cube!

Key problems:

- $\Phi(\theta)$ of generated particles must reproduct the experimental data with sufficient accuracy.
- Same for momentum distribution.

Horizontal area problem

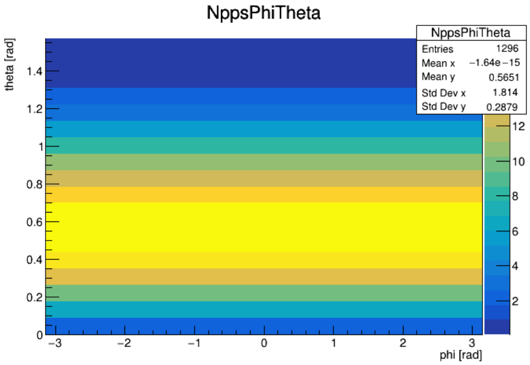
1. 2π solid angle is divided into small Ω_i solid angles
2. for each Ω_i: the number of incoming particles from this Ω_i in simulated time t_{sim} is calculated – N_i



Horizontal area problem

Number of incoming particles from Ω_i per second can be mapped, regarding θ_i and ϕ_i:

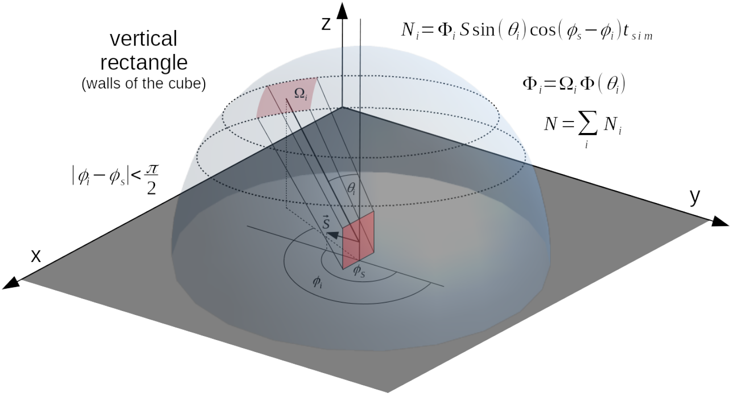
$$Npps_i = \Omega_i \Phi(\theta_i) S sin(\theta_i)$$



The cube ceiling is horizontal $\Rightarrow$ flux depends only on θ

Vertical area problem

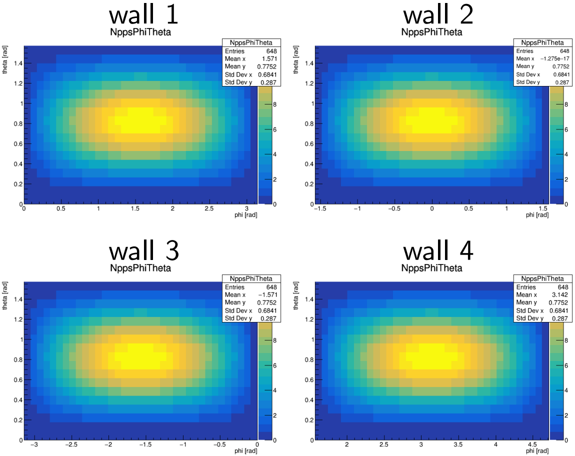
1. Ω_i and N_i – same as for horizontal area
2. Only particles that come from one side of the wall are generated $\Rightarrow$ ϕ_i range is limited!
3. N_i depends on both θ_i and ϕ_i



Vertical area peoblem

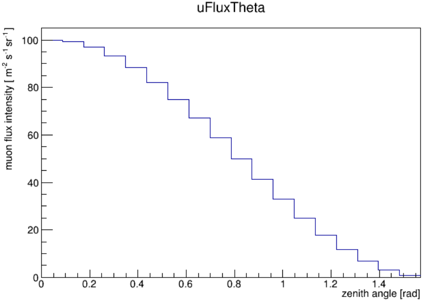
$Npps_i$ formula for vertical walls of the cube:

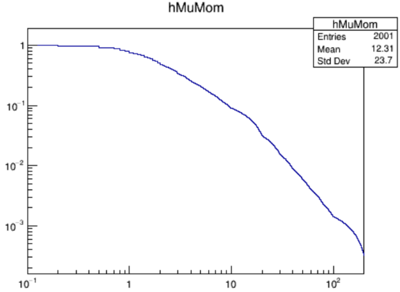
$$Npps_i = \Omega_i \Phi(\theta_i) S sin(\theta_i) cos(\phi_s - \phi_i)$$



How momentum is generated

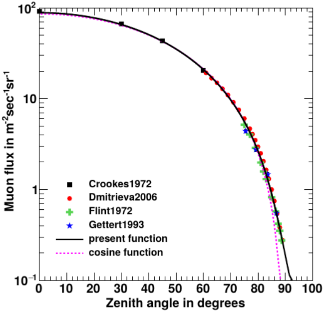
These two histograms made from experimental data are necessary for momentum generation

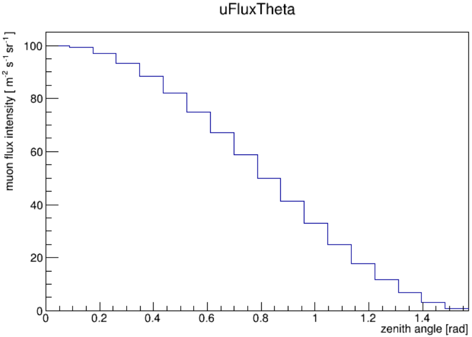




$\Phi(\theta)$ – input histogram

On the left: measured vertical muon flux at the sea level (log scale), fitted with 'cosine function': $\Phi(\theta) = cos^2(\theta)$. Data source: https://arxiv.org/pdf/1606.06907.pdf

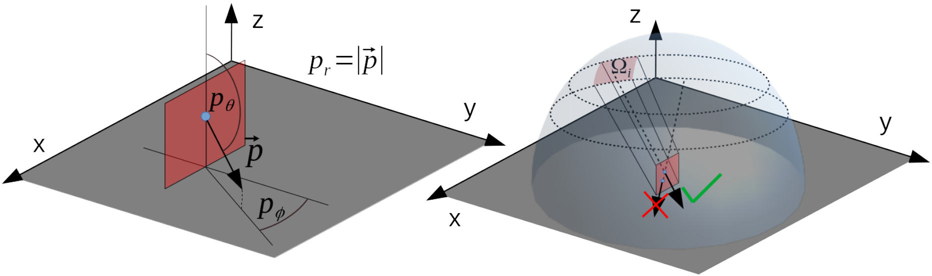




On the right: the input hostogram of $\Phi(\theta)$ (linear scale). Since $cos^2(\theta)$ fits the data precisely enough, the histogram is just filled with $cos^2(\theta)$ distribution.

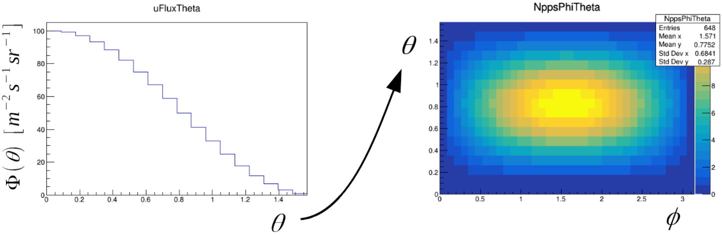
$\Phi(\theta)$ and flux mapping $\rightarrow$ p_θ and p_ϕ

1. Momentum coordinates p_θ and p_ϕ are limited by the solid angle Ω_i:



$\Phi(\theta)$ and flux mapping $\to$ p_θ and p_ϕ

2. Flux is mapped into θ–ϕ space regarding $\Phi(\theta)$ histogram:



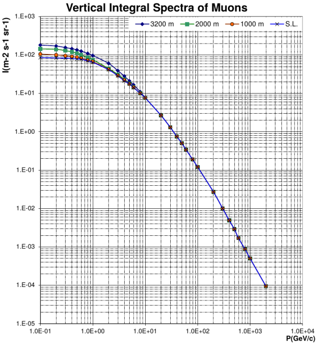
$$Npps_i \propto \Phi(\theta_i)$$

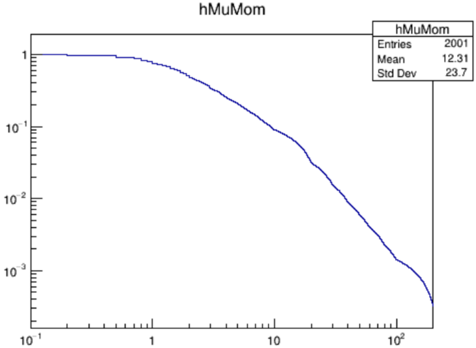
3. Ω_i corresponds with the intervals: $[\theta_i,\ \Delta_\theta)$ and $[\phi_i,\ \Delta_\phi)$. Within the intervals, p_θ and p_ϕ is drawn from uniform distrubution.

4. **Generation of $N_i \propto Npps_i$ particles from each Ω_i solid angle guarantees that the given $\Phi(\theta)$ is reconstructed by the generated particles.** The accuracy of this reconstruction is sufficient if $\Delta\theta$ is small enough.

p_r distribution

On the left: measured vertical integral spectra of muons. Data source: http://crd.yerphi.am/Muons



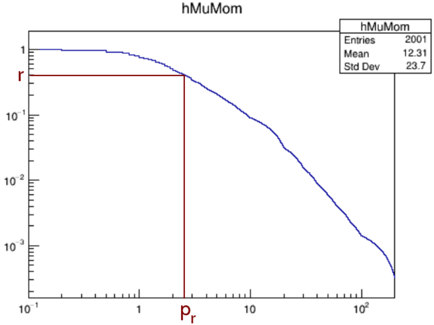


On the right: historgam made from measured integral momentum spectra od muons. Linear interpolation between data points was applied. It is scaled, so the maximum equals 1.

p_r generation

The algorithm:

1. a random number $r \in [0, 1)$ is generated (uniform distribution),
2. hMuMom: finding last bin which value is greater than r,
3. bin center of the bin ('x value') is the drawn p_r [GeV].



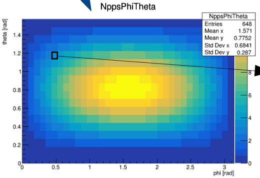
This procedure is very similar to drawing random number using quantile function.

Generation of particles step by step

BEGIN: setting generator parameters
example:
simulated time $t_{sim} = 10$ s
cube edge = 2m

loop over walls

loop over map bins



Example bin: ϕ=0.4 θ=1.14
Npps = 3.46

calculating the number of particles per sedond, that comes from the solid angle defined by θ and ϕ:

$$N_i = N_{pps} \cdot area \cdot t_{sim} = 3.46 \cdot 4 \cdot 10 = 138.4$$

non-interger? N_i is rounded up or down:
→ chance for rounding up = N_i – floor(N_i)

loop over N_i
(generation of N_i particles)

drawing momentum magnitude p_r (implementation of quantile function method)

drawing momentum coordinates p_θ and p_ϕ corresponding to the current solid angle

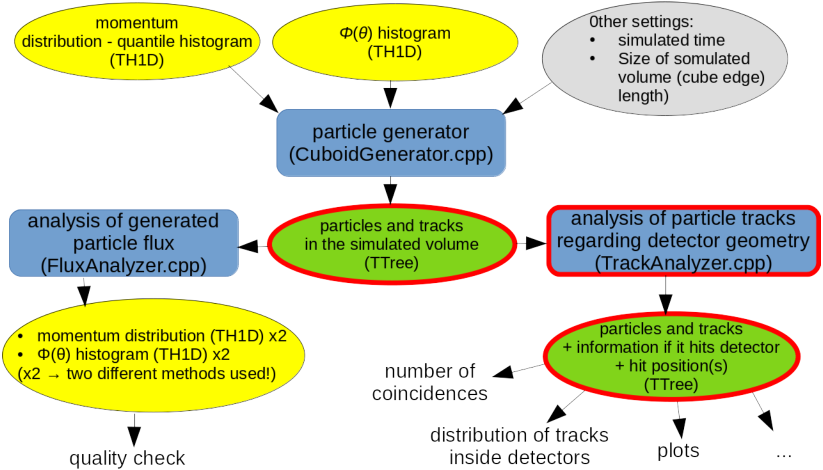
transforming momentum coordinates from spherical to Carthesian coordinate system:

$$(p_r,\ p_\theta,\ p_\phi) \Rightarrow (p_x,\ p_y,\ p_z)$$

drawing random position on the wall (uniform distribution)

writing the particle to the output file

END


momentum distribution - quantile histogram (TH1D)
$\Phi(\theta)$ histogram (TH1D)
Other settings:
• simulated time
• Size of somulated volume (cube edge) length)
particle generator (CuboidGenerator.cpp)
analysis of generated particle flux (FluxAnalyzer.cpp)
particles and tracks in the simulated volume (TTree)
analysis of particle tracks regarding detector geometry (TrackAnalyzer.cpp)
• momentum distribution (TH1D) x2
• Φ(θ) histogram (TH1D) x2
(x2 → two different methods used!)
quality check
particles and tracks + information if it hits detector + hit position(s) (TTree)
number of coincidences
distribution of tracks inside detectors
plots
...

TrackAnalyzer.cpp

Detector elements are defined in the source code (no input config file) – so far it is good enough for my needs.

Implemented shapes (C++ classes):

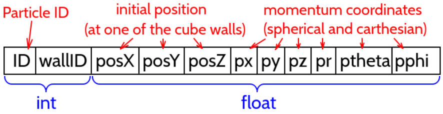
- Rectangle
- Disk
- Cylinder

Every spahe has a methods that detects if the particle hits the detector module (shape instance) and calculates hit position(s).

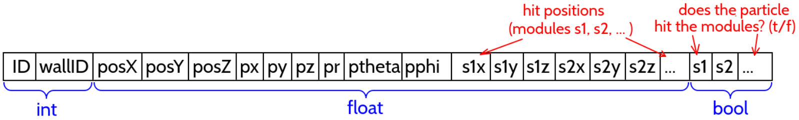
More detailed description of the classes is presented in the section Technical delails and class description.

TrackAnalyzer.cpp – input and output tree

Input tree from CuboidGenerator – every entry represents one particle. Variables are stored in different branches:



Output tree = inout tree + information if the particle hits each module and hit positions:



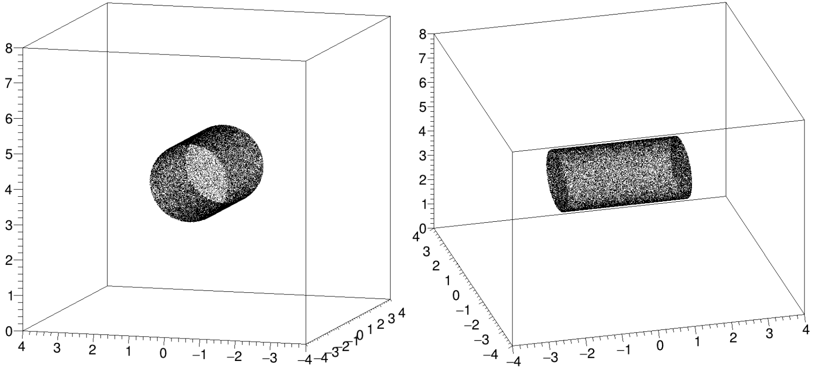
This output tree pattern works fine for simple detector lauoyt, but is not optimal regarding the output file size. For more complicated detector geometries (hundreds of elements), a different way of creating output tree may be needed.

TrackAnalyzer.cpp – plot of example output

An example cylinder was defined:

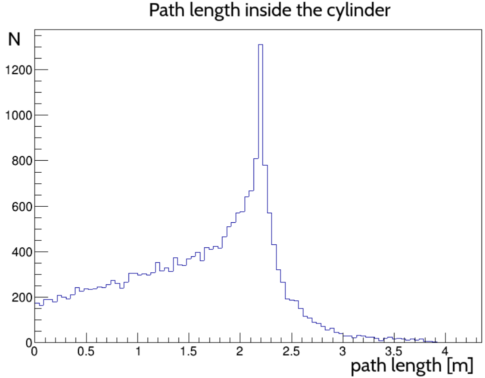
- radius: r= 1.1 m
- length L= 3.4 m

One can print the points, where the particle track intersects the cylinder surface. Cylinder shape reveals ⇒ geometry implementation works correctly



TrackAnalyzer.cpp – plot of example output

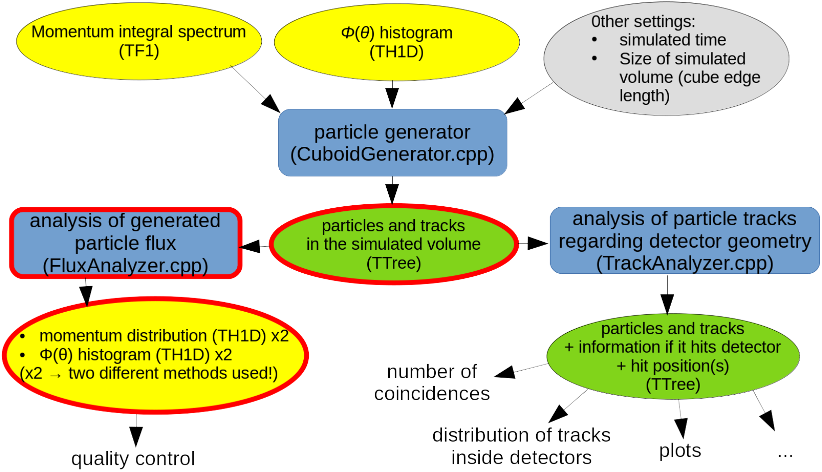
One can also plot a distribution of the path length inside the cylinder:



The distribution is correct:

- The longest possible path length is $s_{max} = \sqrt{(4r^2 + L^2} \approx 4.1$ [m]
- Regarding most particles come from the 'ceiling', the most common track length should be approximately equal to $2r = 2.2$ [m]

FluxAnalyzer.cpp and quality check



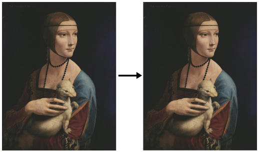
FluxAnalyzer.cpp and quality check

Quality check of generated particles:

- Are the initial positions ok?
- How well $\Phi(\theta)$ and momentum distribution resembles the given experimental data?

Is it like:

or like:



?

FluxAnalyzer.cpp – two methods of investigating $\Phi(\theta)$

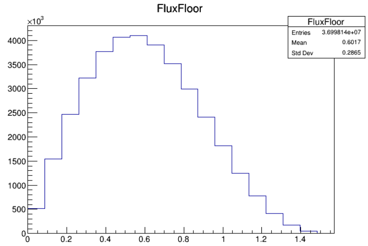
1. floor method:
   1. Filling θ distribution histogram with particles that hits the cube floor (bin width: $\Delta\theta$).
   2. Normalizing the distribution to obtain $\Phi(\theta)$. Normalization function:

$$f_n(\theta) = \frac{1}{S \times t_{sim} \times cos(\theta) \times \Omega(\theta)}$$

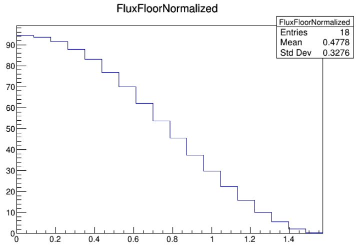
2. rotating rectangle method:
   1. Initializing a horizontal rectangle inside the dube the cube. The center of the cube is also the center of this rectangle.
   2. Rotating it through angle $\Delta\theta$. Rotation axis: contains the center of the cube, parallel to the x axis.
   3. filling θ distribution histogram with particles that hit the rectangle and come from the limited solid angle $\Omega(\theta)$ in front of the rectangle.
   4. Normalizing the distribution to obtain $\Phi(\theta)$. Normalizaing function:

$$f_n(\theta) = \frac{1}{S \times t_{sim} \times \Omega'(\theta)}$$

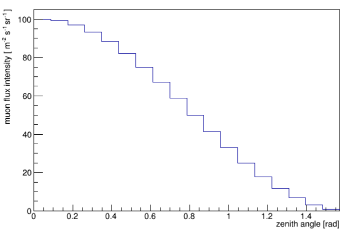
Quality check: θ distribution – floor method



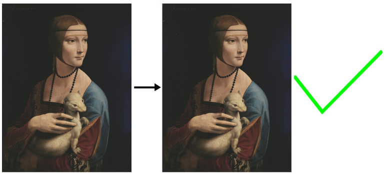
Before normalization



Normalized

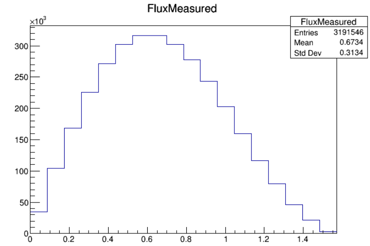


Input $\Phi(\theta)$

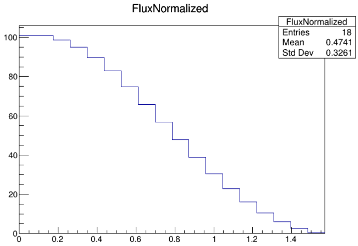


Very well!

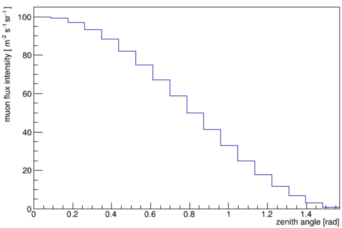
Quality check: θ distribution – rotating rectangle method



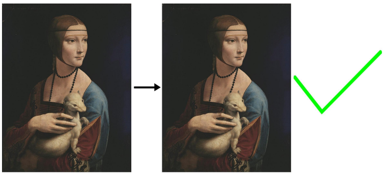
Before normalization



Normalized



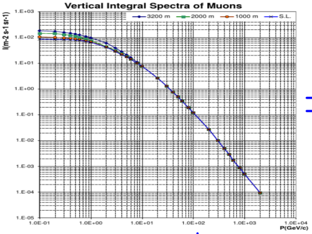
Input $\Phi(\theta)$

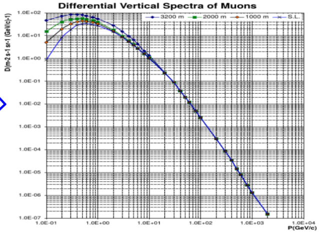


Even better!

Quality check: p_r

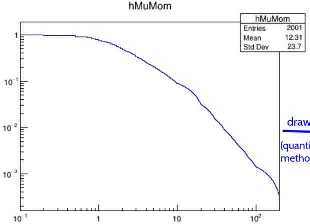
experimental data





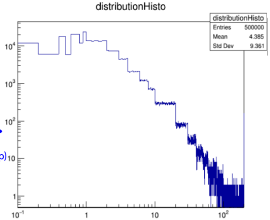
linear interpolation
scaling

simulation



drawing p

(quantile function
method, 'reflected' in p)



Quality check: p_r evaluation

Key notes:

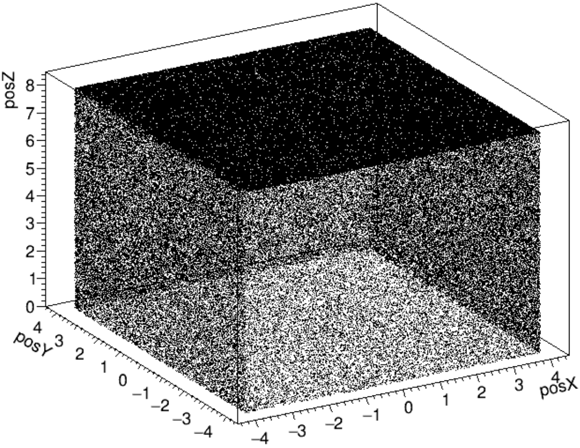
- No normalization implemented for reconstructed p_r $\Rightarrow$ measured and reconstructed p_r cannot be directly compared.
- The shape of reconstructed p_r seems to be correct, but the reason for size of steps will be investigated.

It is yet not obvious whether the p_r reconstruction is as bad as:

and it was good enough to produce some preliminary results, however, the quantile function implementation is planned for improvements.

Quality check: initial positions

Initial posions are uniformly distributed on each wall of the cube. Also, the highest particle 'surface density' of particles is onserved on the cube ceiling.



Initial positions are correctly generated!

Example simulation

Example simulation - general parameters

General parameters (CuboidGenerator.cpp):

- cube edge: 8 [m]
- central point: x=0, y=0, z=4
- simulated time: 1 [h]
- simulated $p_{min} = 0.1$ [GeV/c]

For detection, $p_{min} = 1.6$ [GeV/c] was assumed. Tracks with lower momenta are ignored (TrackAnalyzer.cpp)

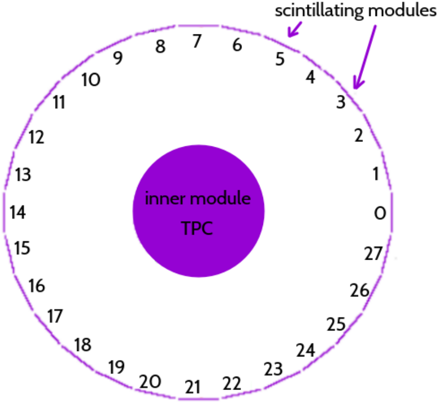
TPC parameters

- a single cylinder – length = 3.4 [m]; radius = 1.1 [m]
- axis of symmetry: parallel to the X axis, in the center of the cube (y=0, x=4)
- efficiency $\eta = 1$

Simulated detector geometry

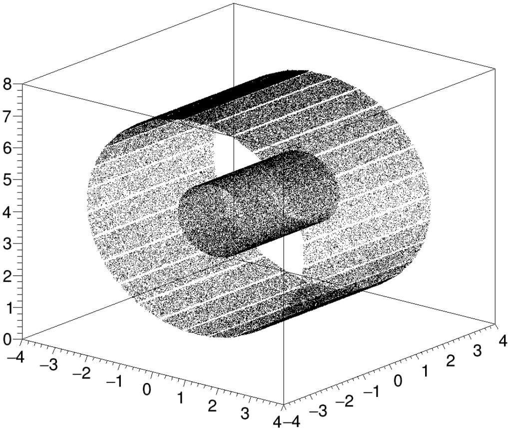
Scintillating modules

- rectangular modules – length = 4.784 [m]; width = 0.675 [m]
- efficiency $\eta = 0.9$
- placed around TPC axis

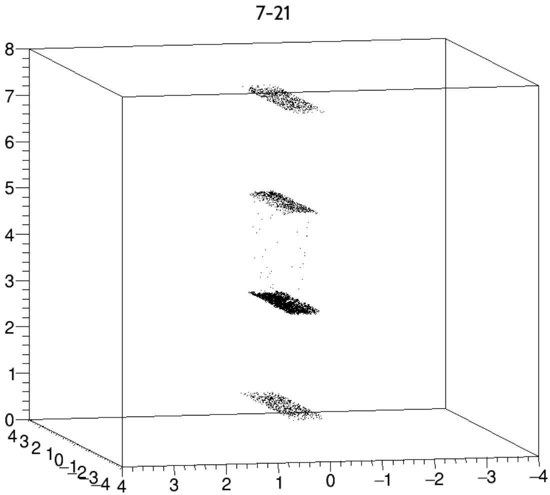


Obtained detector geometry visualisation

Drawing positions where tracks hits the detectors reveals geometry of the detectors

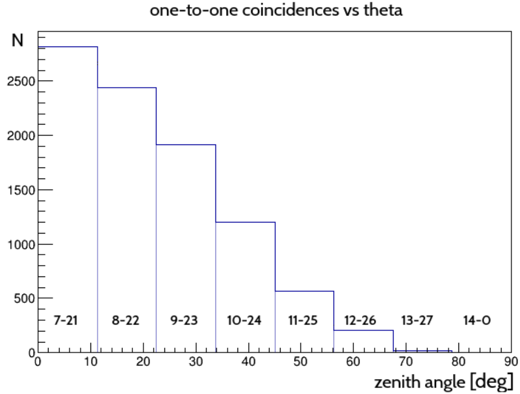


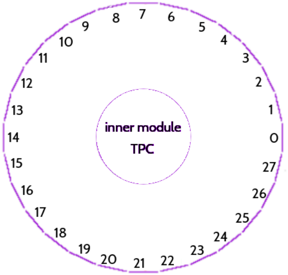
One-to-one coincidences



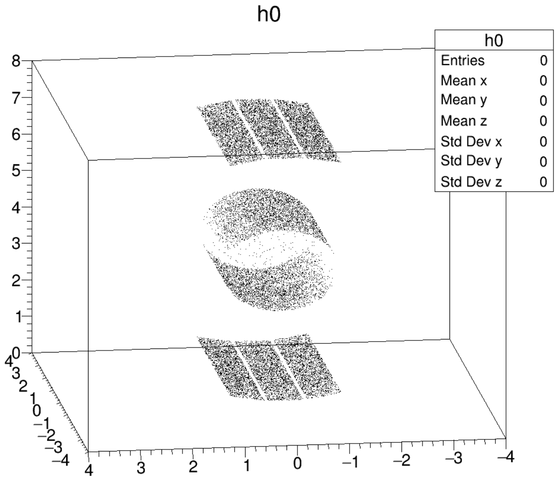
One-to-one coincidences (with TPC)

Similar to $cos^2(\theta)$ function – correct!



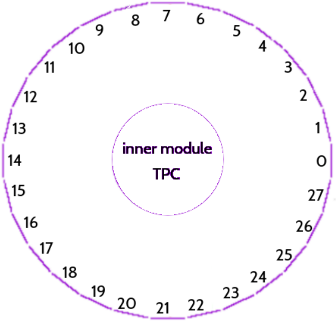


Layout 1: modules close to each other

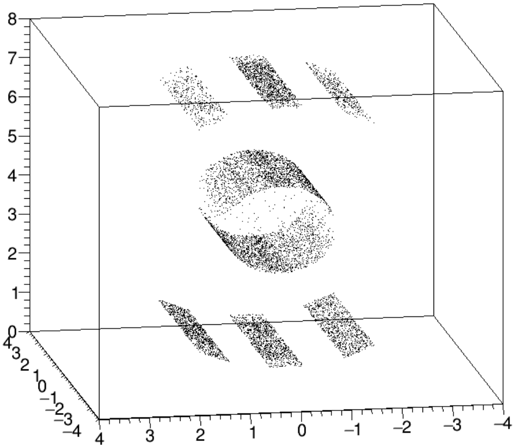


Layout 1: modules close to each other

- (6 or 7 or 8) and (20 or 21 or 22) and TPC – 23341 coincidences per hour
- (9 or 10 or 11) and (23 or 24 or 25) and TPC – 15415 coincidences per hour
- (12 or 13 or 14) and (26 or 27 or 0) and TPC – 1956 coincidences per hour

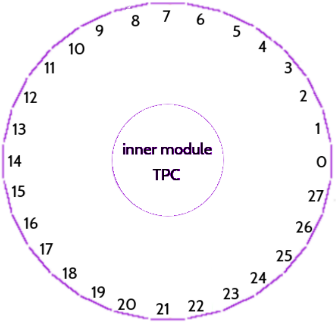


Layout 2: space between modules



Layout 2: space between modules

- (5 or 7 or 9) and (19 or 21 or 23) and TPC – 31402 coincidences per hour
- (10 or 12 or 14) and (24 or 26 or 0) and TPC – 4892 coincidences per hour



Thank you for your attention