

# cfluxim cosmic ray simulation tool

Kamil Wójcik

University of Silesia

2020

[kamil.wojcik@us.edu.pl](mailto:kamil.wojcik@us.edu.pl)

# Contents

## 1 General information

## 2 Cosmic ray flux – definition

## 3 Simulation design

- CuboidGenerator
- TrackAnalyzer.cpp
- FluxAnalyzer.cpp and quality check

## 4 Example simulation results

# General information

# About cfluxim

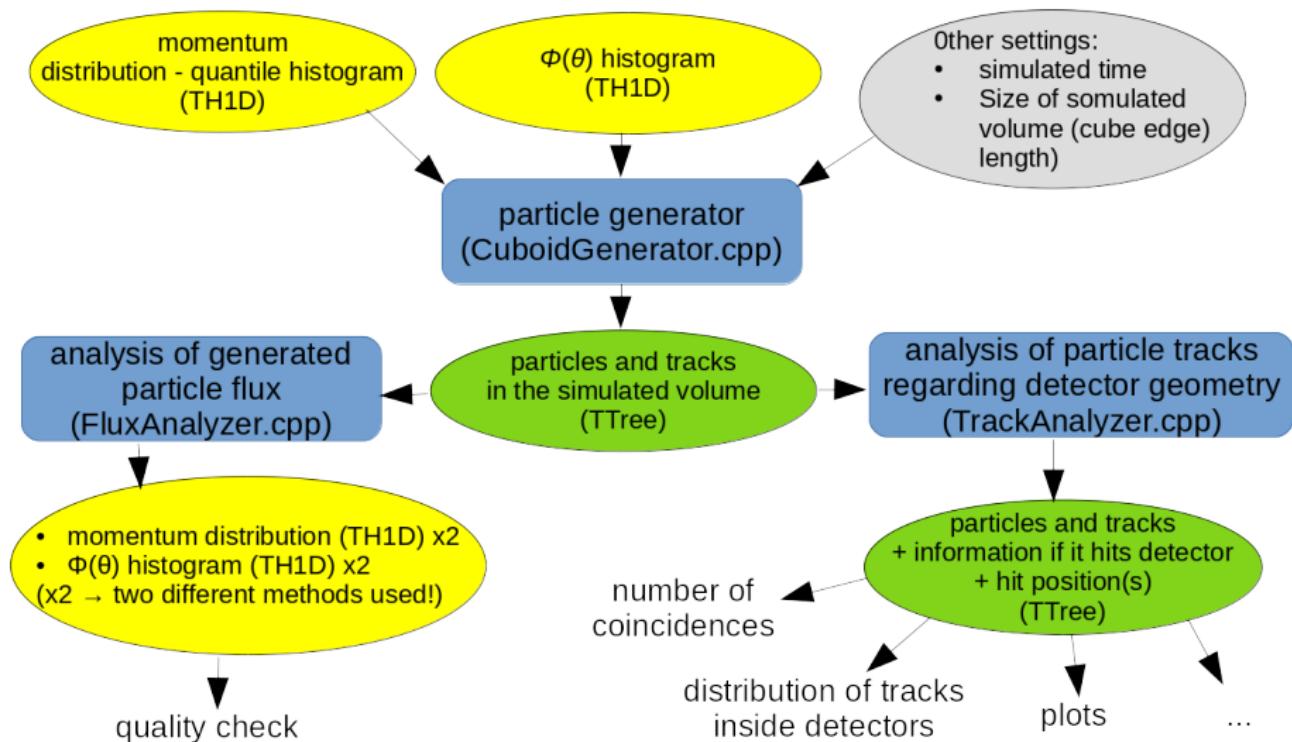
The `cfluxim` project was created to make a simple geometrical simulation of cosmic muons passing through some modules of the MPD detector ([NICA MPD website](#)).

- Cosmic muons are generated inside the given cubic volume.
- $\Phi(\theta)$  and momentum distribution of the simulated flux fits the 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

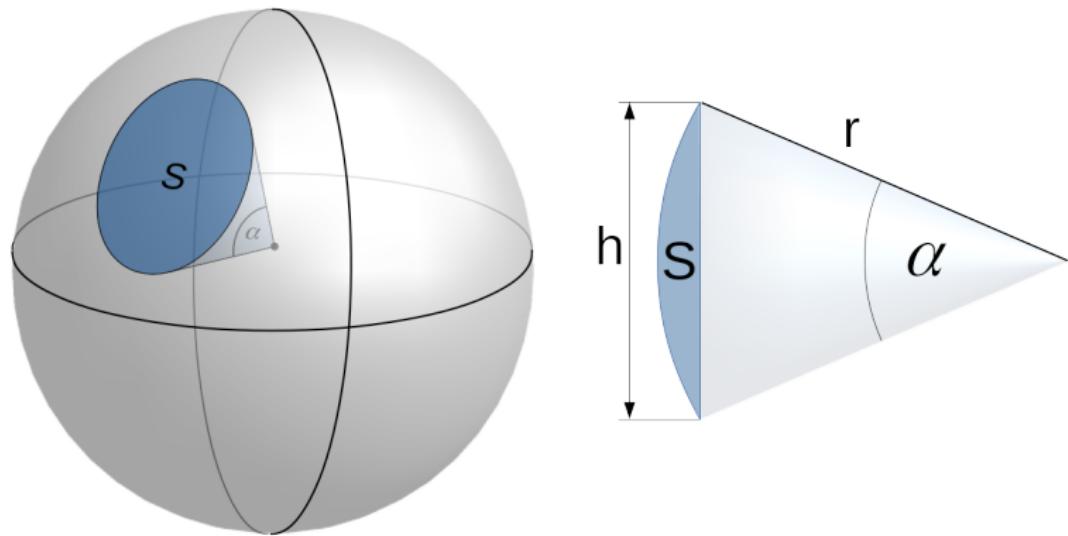
- `cfluxim` uses [CERN's ROOT](#) libraries.
- `cfluxim` consists of 3 tools: `CuboidGenerator`, `FluxAnalyzer` and `TrackAnalyzer`.
- Only muons at ground level are implemented, however, implementation of other cosmic ray components is possible.
- `FluxAnalyzer` generates momentum distribution and  $\Phi(\theta)$  normalized histogram, so it can be compared with 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



# Basic definitions

# Solid angle

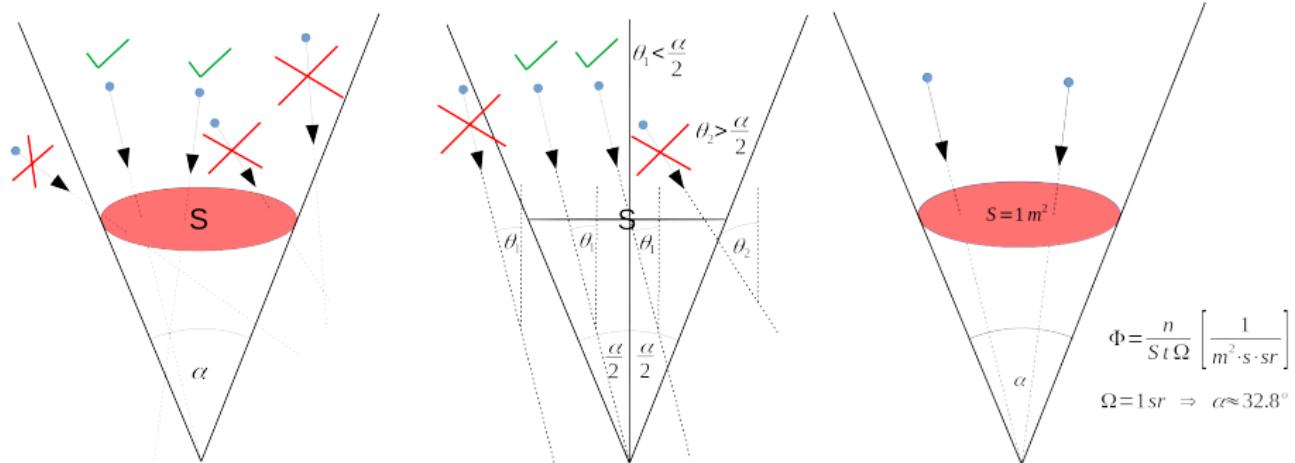


$$\text{area: } S = 2\pi r h = 2\pi r^2(1 - \cos\alpha)$$

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

# Cosmic ray flux

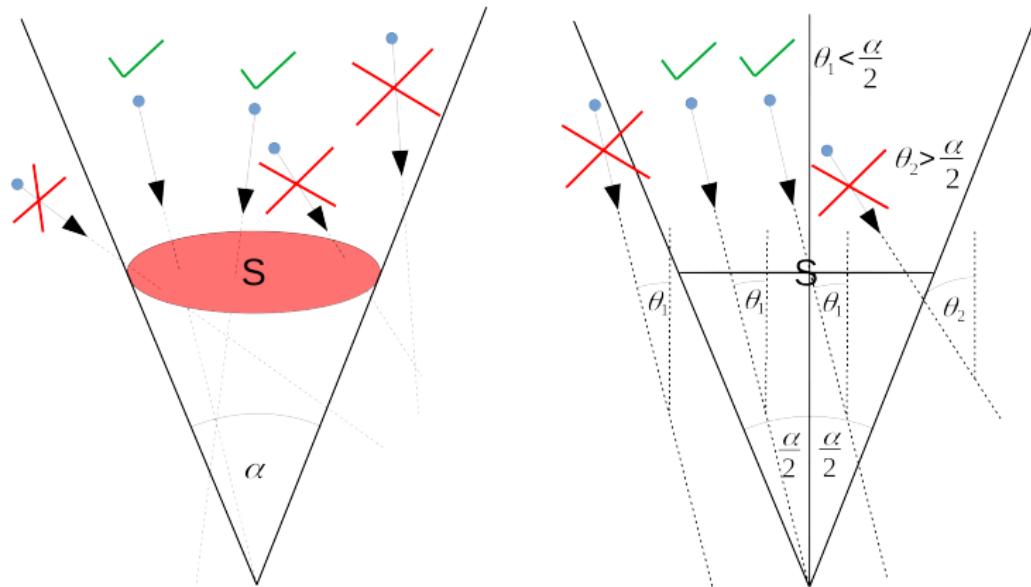
*Cosmic ray flux = number of particles that come from unit solid angle, passing through unit area, per unit of time*



particles coming from a given solid angle  
passing through area  $S$

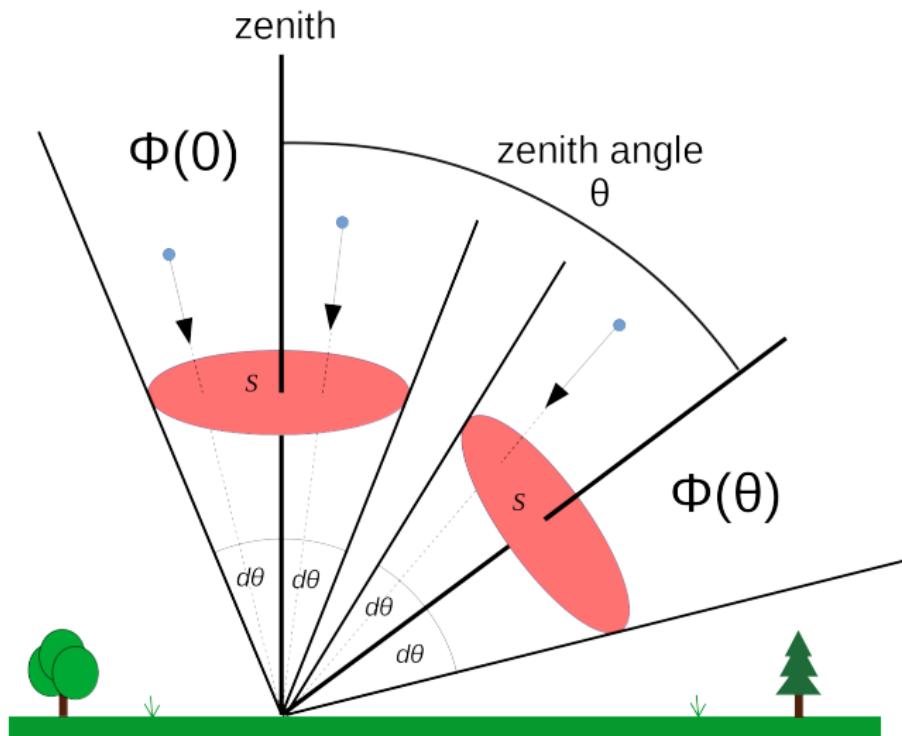
standard normalization

# Cosmic ray flux

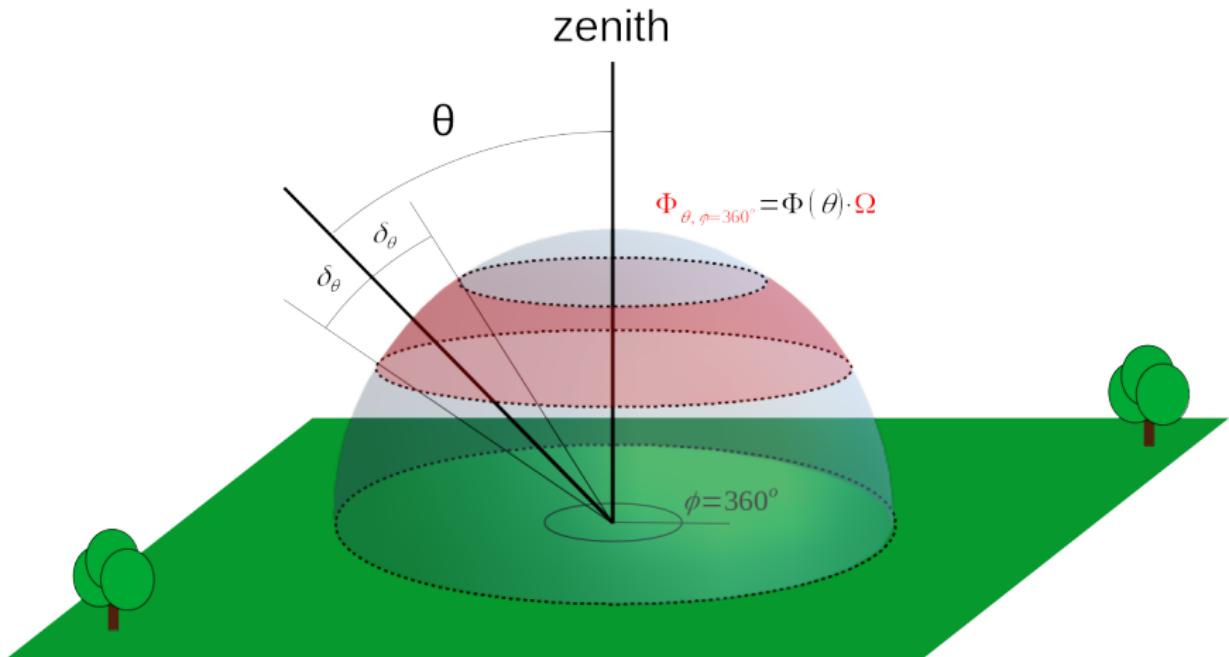


Solid angle limits the *direction* of incoming particle momentum, but not the position on the ‘probing area’  $S$ . To count a particle as coming from the given solid angle, momentum angular limitations must be fulfilled and the particle must hit the probing area.

# Cosmic ray flux dependant on zenith angle



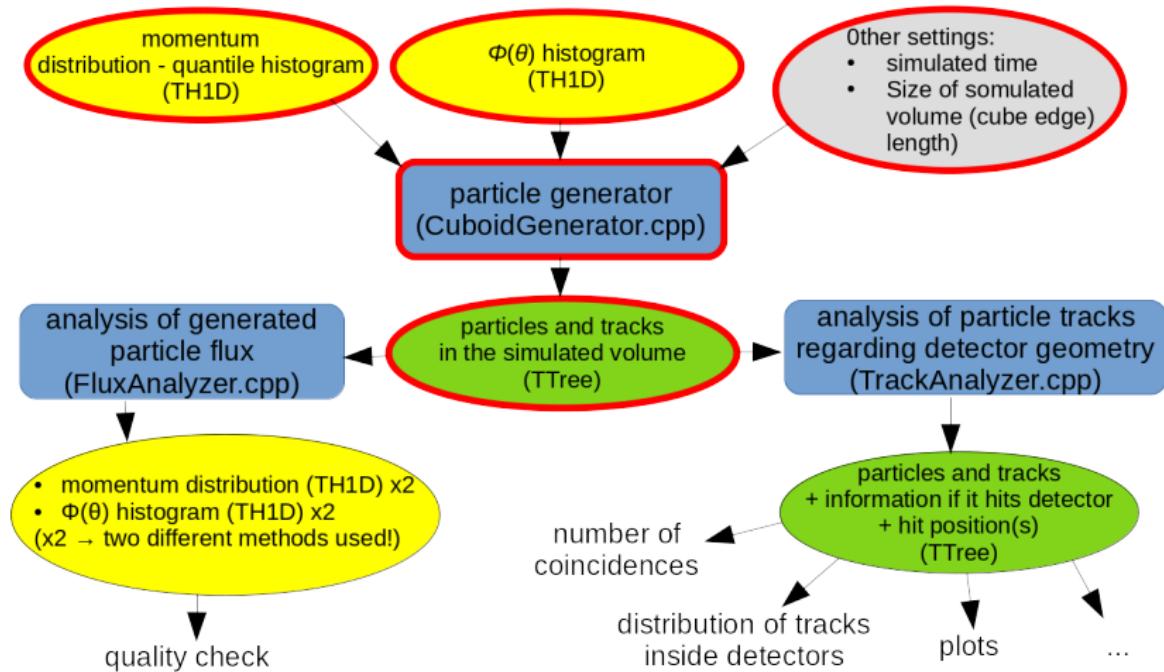
# Cosmic ray flux dependant on zenith angle – full azimuth angle case



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

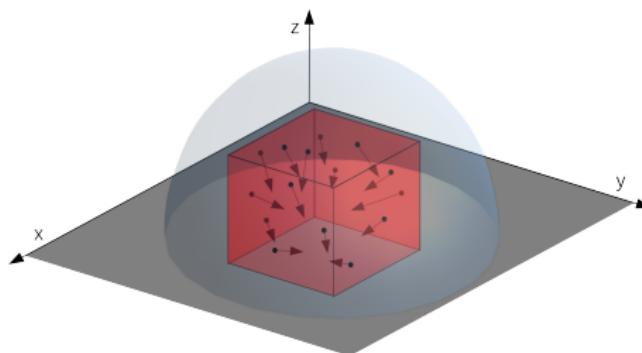
# Simulation design

# 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.



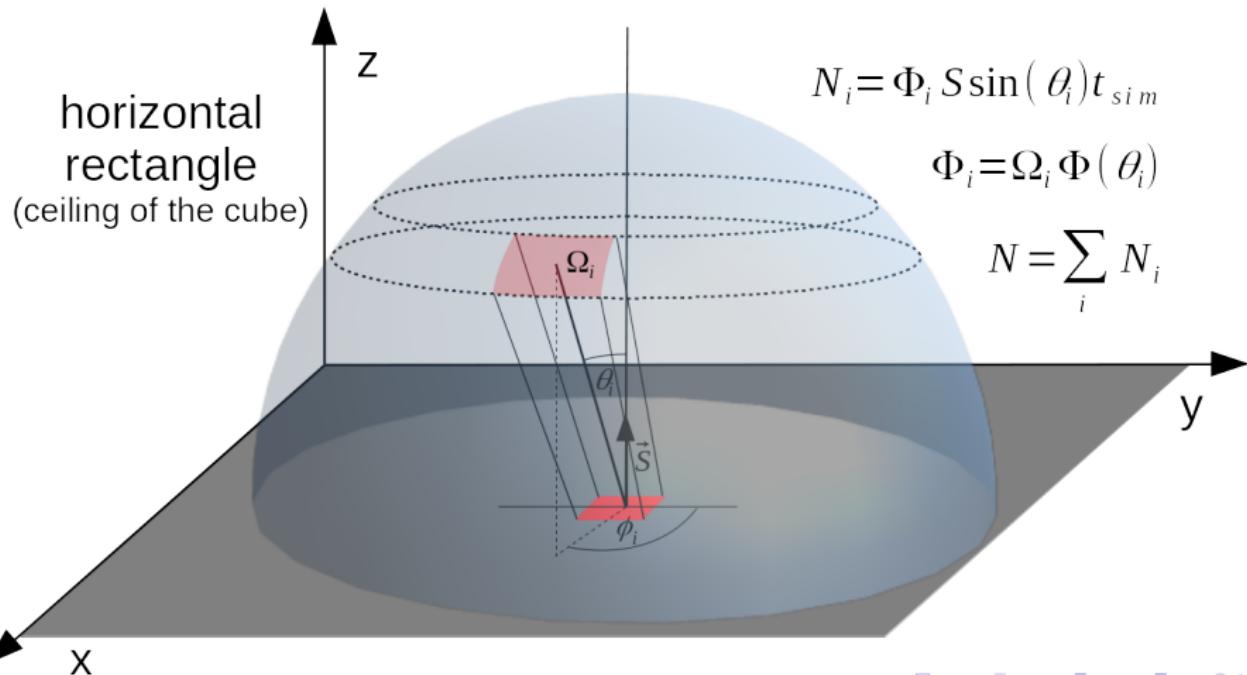
Particle's 'initial' position on the cube wall and its momentum vector define the track inside the cube!

Key problems:

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

# Horizontal area problem

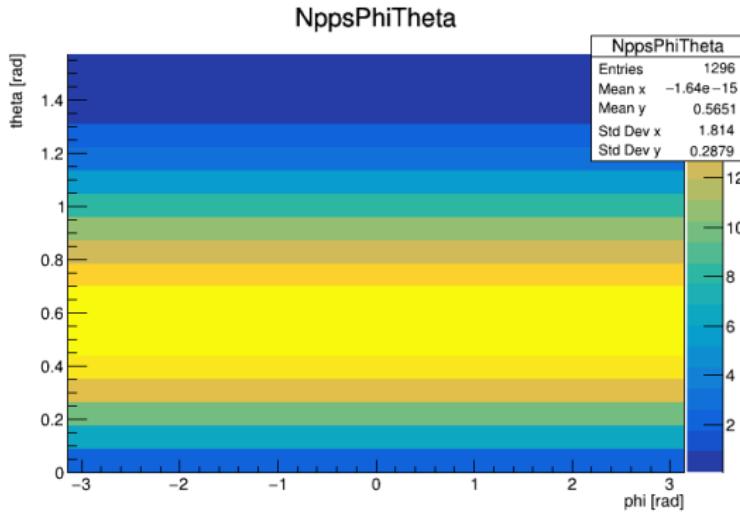
- ①  $2\pi$  solid angle is divided into small  $\Omega_i$  solid angles
- ② for each  $\Omega_i$ : the number of incoming particles from this  $\Omega_i$  in simulated time  $t_{sim}$  is calculated –  $N_i$



# Horizontal area problem

Number of incoming particles from  $\Omega_i$  per second can be mapped, regarding  $\theta_i$  and  $\phi_i$ :

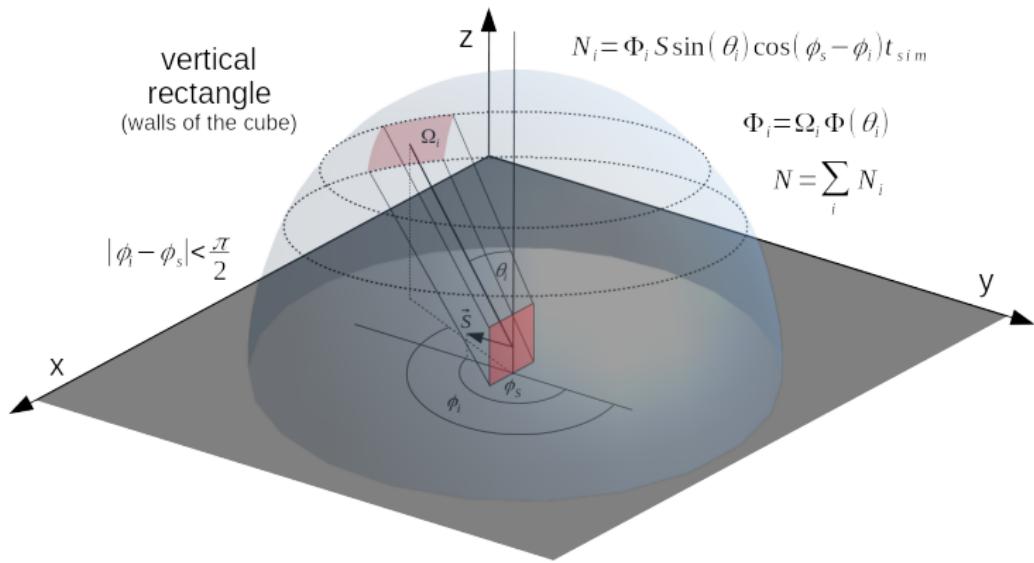
$$N_{pps} = \Omega_i \Phi(\theta_i) S \sin(\theta_i)$$



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

# Vertical area problem

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



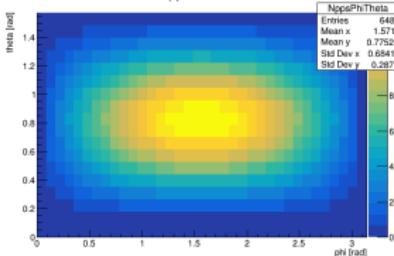
# Vertical area peoblem

$N_{pps_i}$  formula for vertical walls of the cube:

$$N_{pps_i} = \Omega_i \Phi(\theta_i) S \sin(\theta_i) \cos(\phi_s - \phi_i)$$

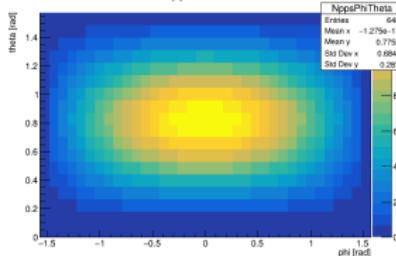
wall 1

$N_{ppsPhiTheta}$



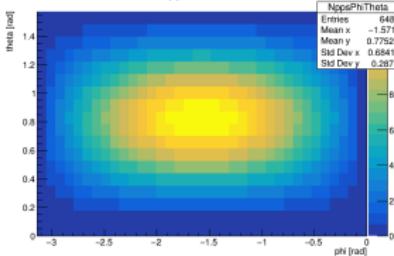
wall 2

$N_{ppsPhiTheta}$



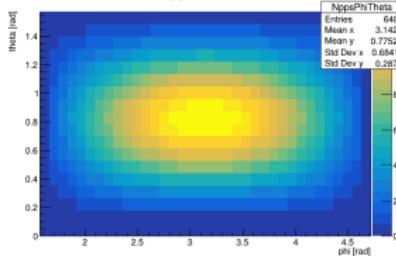
wall 3

$N_{ppsPhiTheta}$



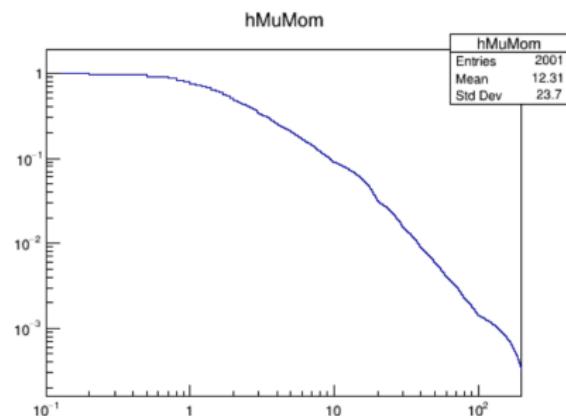
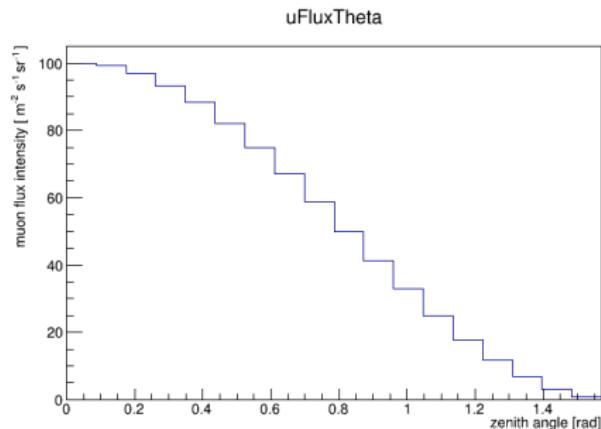
wall 4

$N_{ppsPhiTheta}$



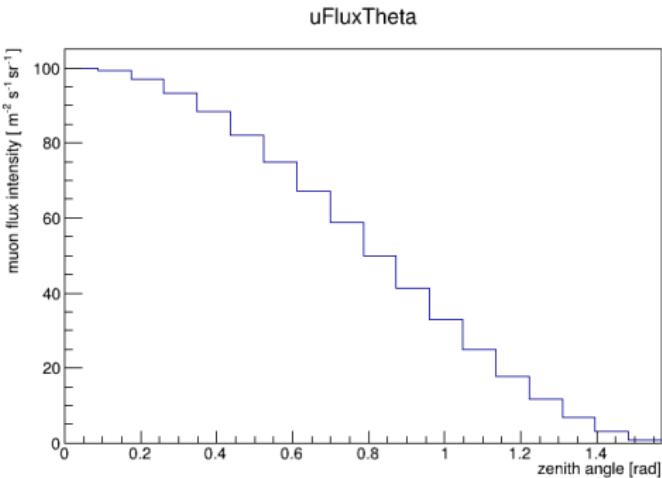
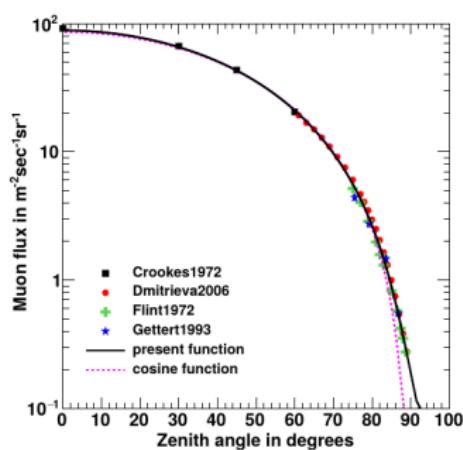
# How momentum is generated

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



# $\Phi(\theta)$ – input histogram

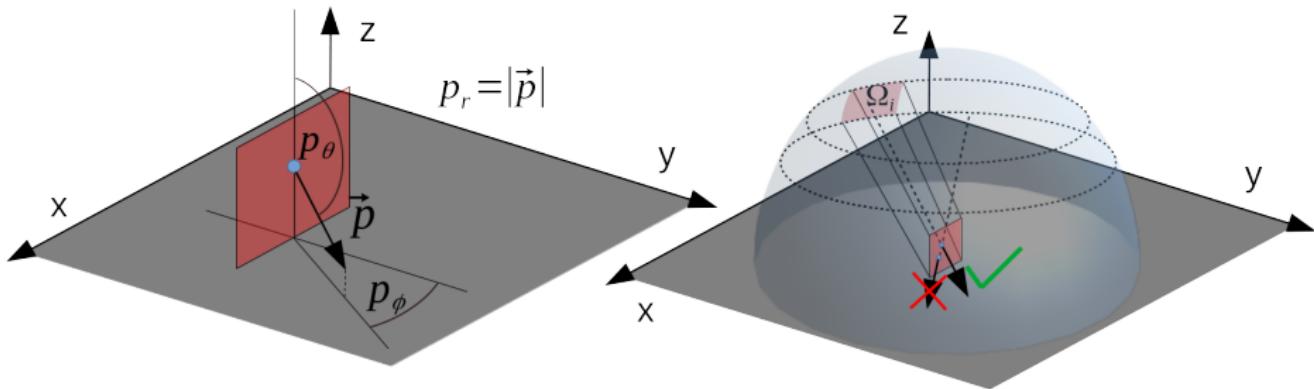
On the left: measured vertical muon flux at the sea level (log scale), fitted with ‘cosine function’:  $\Phi(\theta) = \cos^2(\theta)$ . Data source:  
<https://arxiv.org/pdf/1606.06907.pdf>



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

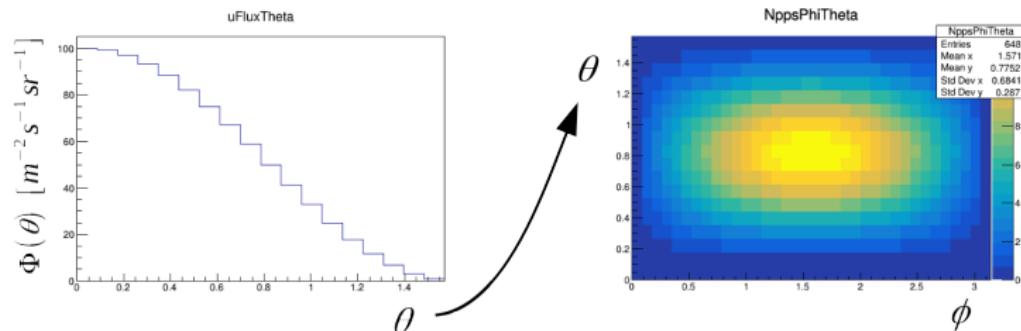
$\Phi(\theta)$  and flux mapping  $\rightarrow p_\theta$  and  $p_\phi$

1. Momentum coordinates  $p_\theta$  and  $p_\phi$  are limited by the solid angle  $\Omega_i$ :



$\Phi(\theta)$  and flux mapping  $\rightarrow p_\theta$  and  $p_\phi$

2. Flux is mapped into  $\theta-\phi$  space regarding  $\Phi(\theta)$  histogram:



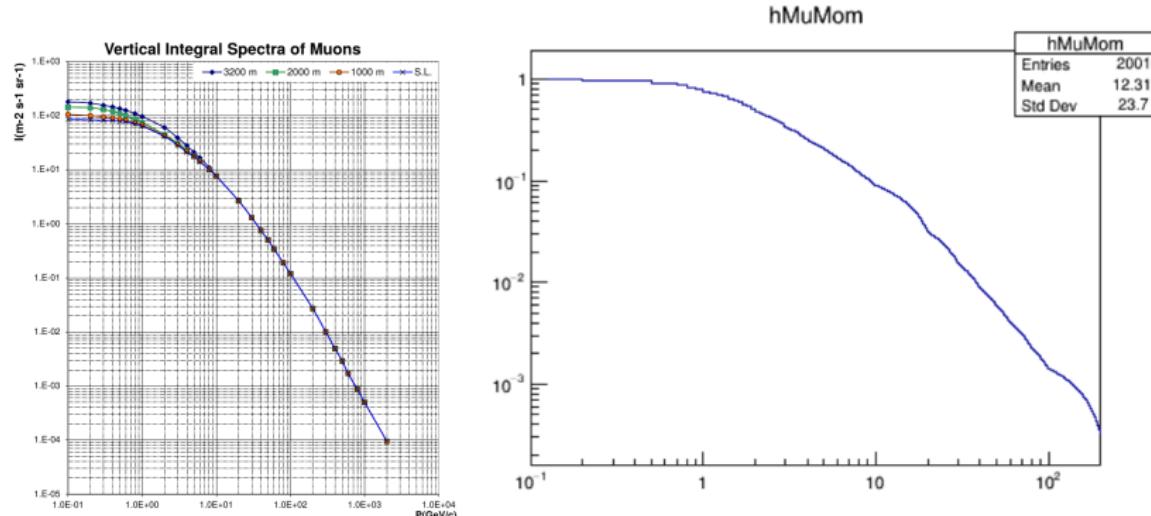
$$Npps_i \propto \Phi(\theta_i)$$

3.  $\Omega_i$  corresponds with the intervals:  $[\theta_i, \Delta_\theta)$  and  $[\phi_i, \Delta_\phi)$ . Within the intervals,  $p_\theta$  and  $p_\phi$  is drawn from uniform distribution.

4. **Generation of  $N_i \propto Npps_i$  particles from each  $\Omega_i$  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>

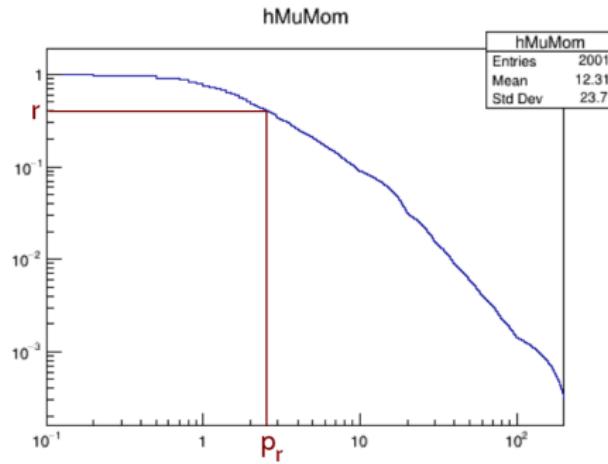


On the right: historgam made from measured integral momentum spectra od muons. Linear interpolation between data points was applied. It is scaled, so the maximum equals 1.

# $p_r$ generation

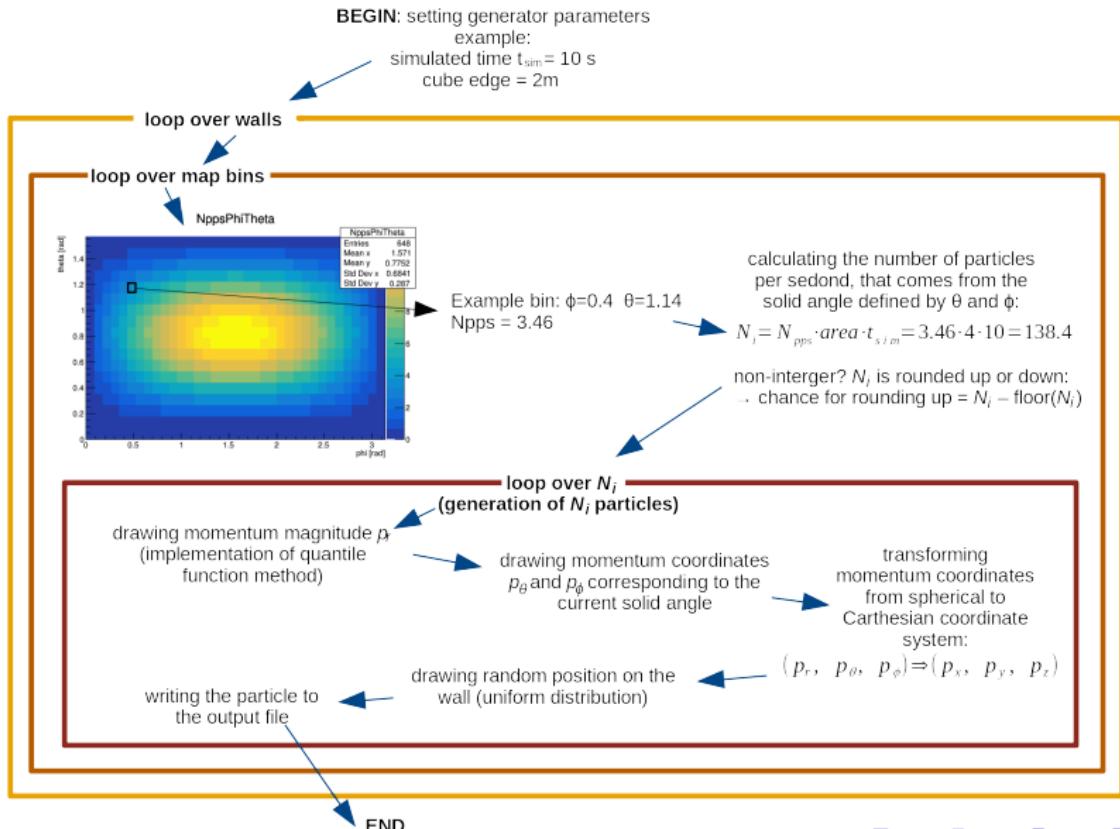
The algorithm:

- 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].

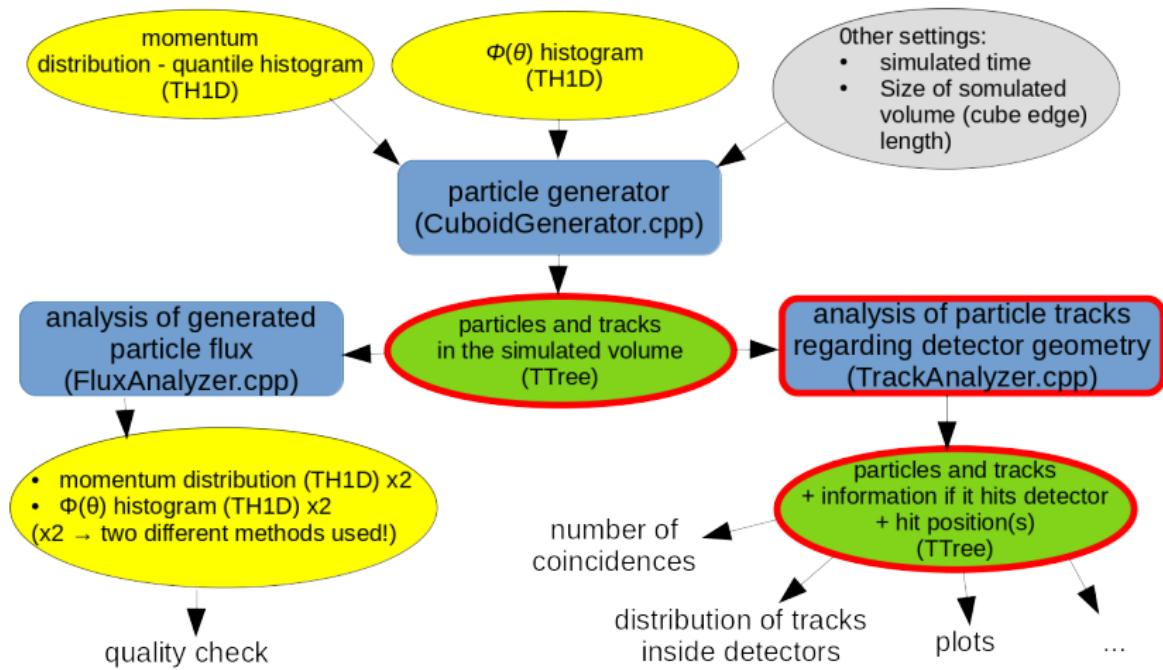


This procedure is very similar to drawing random number using quantile function.

# Generation of particles step by step



# TrackAnalyzer.cpp



# 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

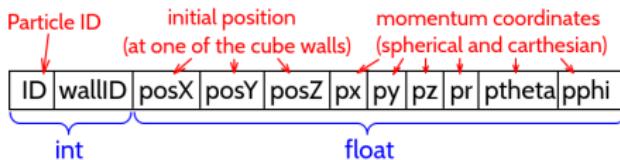
Every shape has a methods that detects if the particle hits the detector module (shape instance) and calculates hit position(s).

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

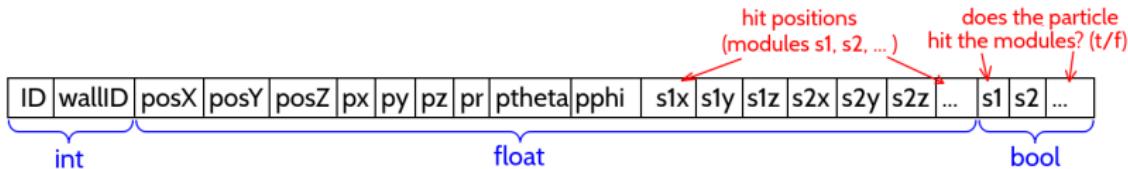
## TrackAnalyzer.cpp – input and output tree

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:



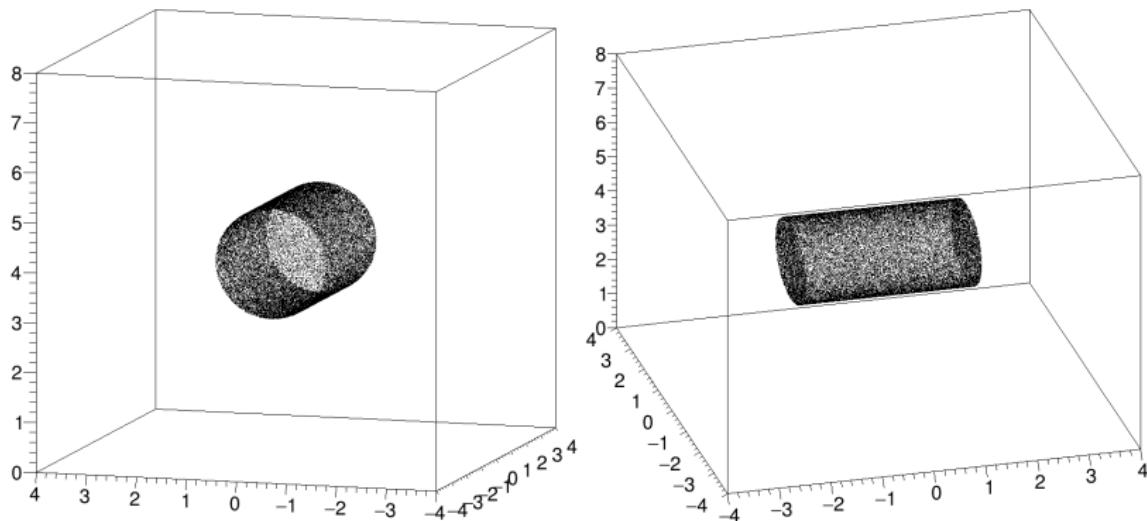
This output tree pattern works fine for simple detector lauoyt, but is not optimal regarding the output file size. For more complicated detector geometries (hundreds of elements), a different way of creating output tree may be needed.

# TrackAnalyzer.cpp – plot of example output

An example cylinder was defined:

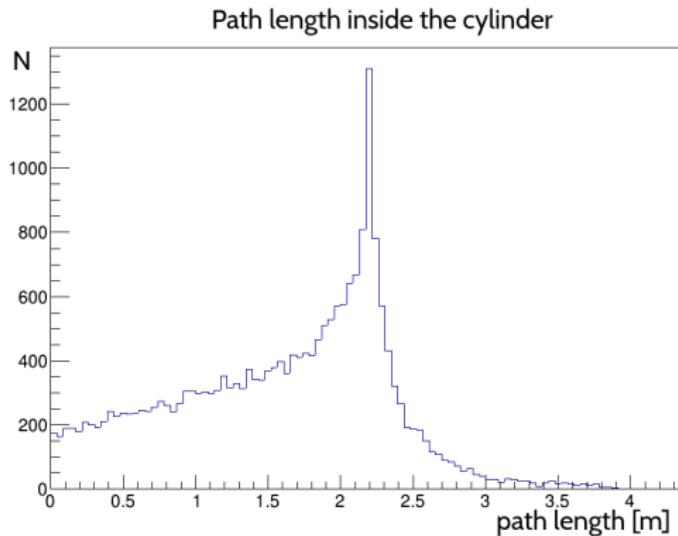
- radius:  $r = 1.1 \text{ m}$
- length  $L = 3.4 \text{ m}$

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



# TrackAnalyzer.cpp – plot of example output

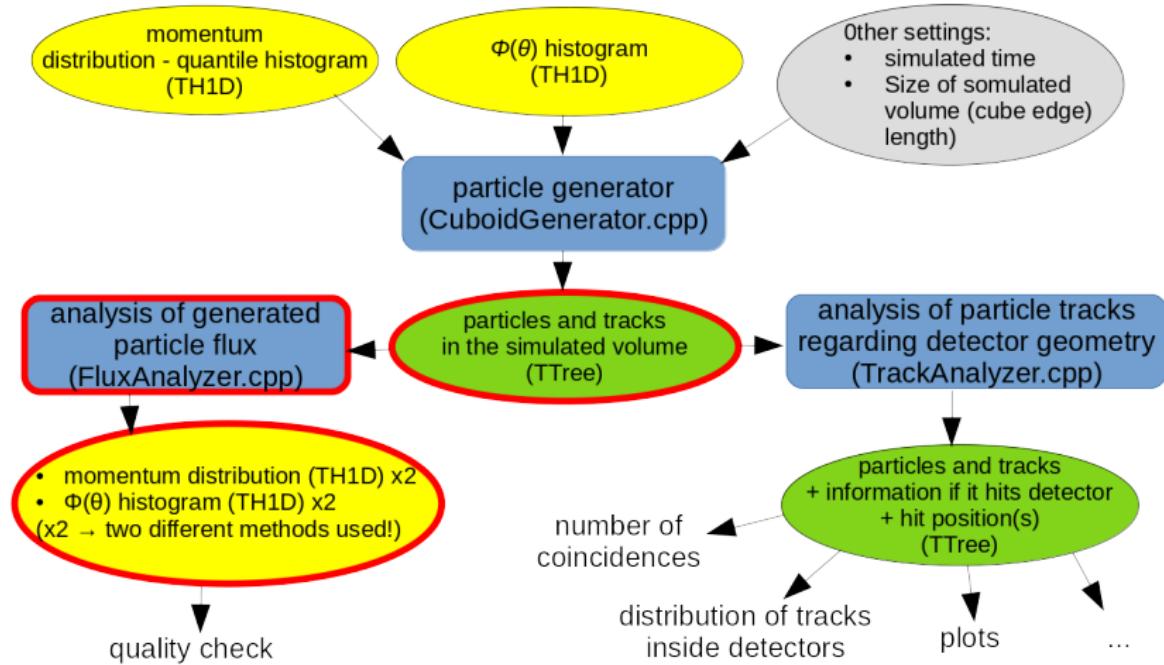
One can also plot a distribution of the path length inside the cylinder:



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

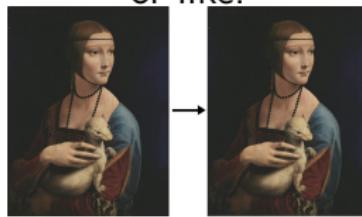
Quality check of generated particles:

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

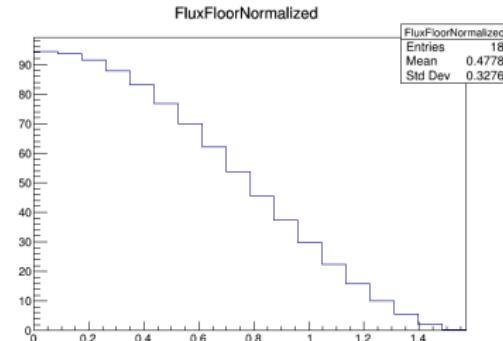
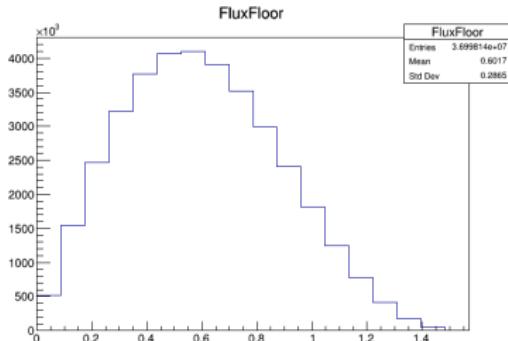
$$f_n(\theta) = \frac{1}{S \times t_{sim} \times \cos(\theta) \times \Omega(\theta)}$$

## ② rotating rectangle method:

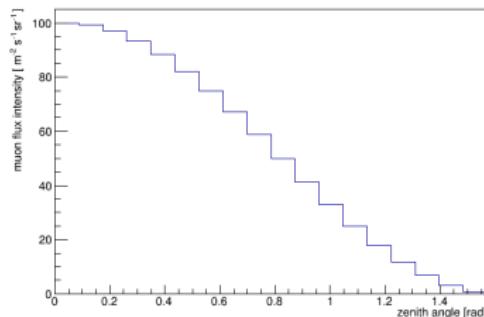
- ① Initializing a horizontal rectangle inside the cube the cube. The center of the cube is also the center of this rectangle.
- ② Rotating it through angle  $\Delta\theta$ . Rotation axis: contains the center of the cube, parallel to the x axis.
- ③ filling  $\theta$  distribution histogram with particles 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: $\theta$ distribution – floor method

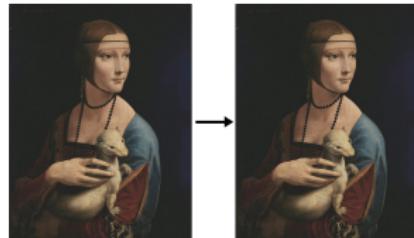


Before normalization



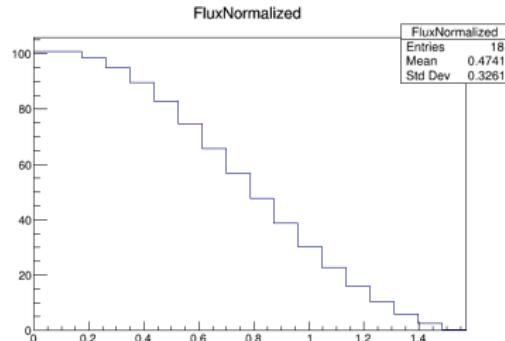
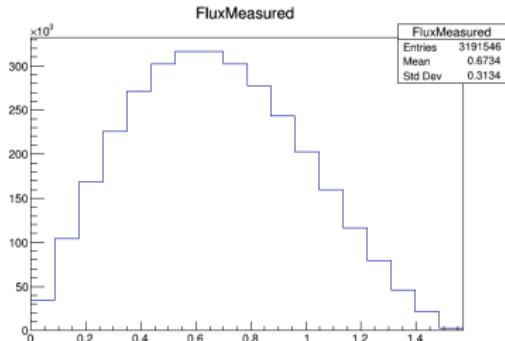
Input  $\Phi(\theta)$

Normalized

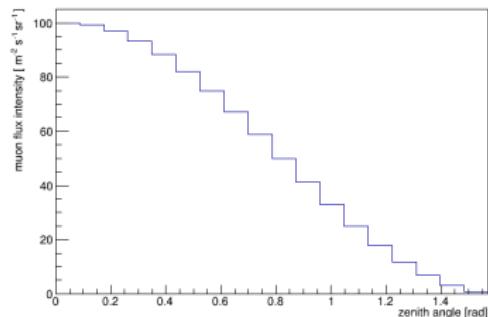


Very well!

# Quality check: $\theta$ distribution – rotating rectangle method

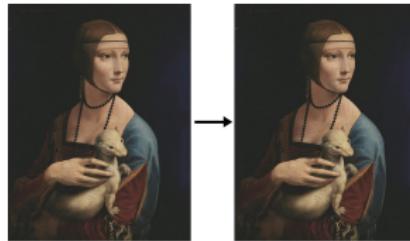


Before normalization



Input  $\Phi(\theta)$

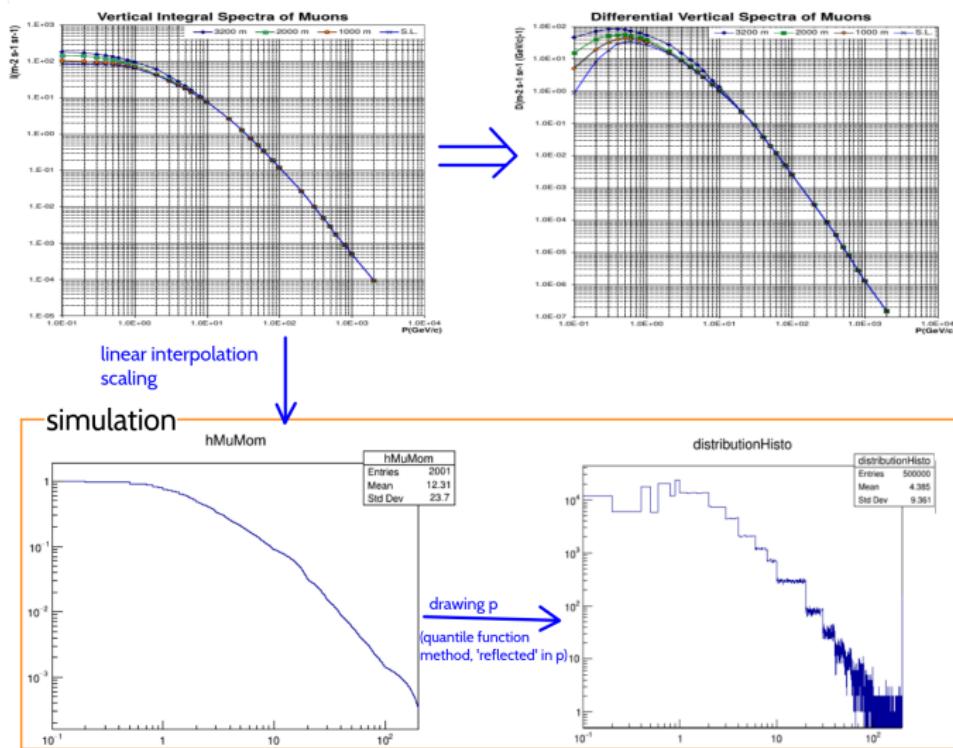
Normalized



Even better!

# Quality check: $p_r$

experimental data



## 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.

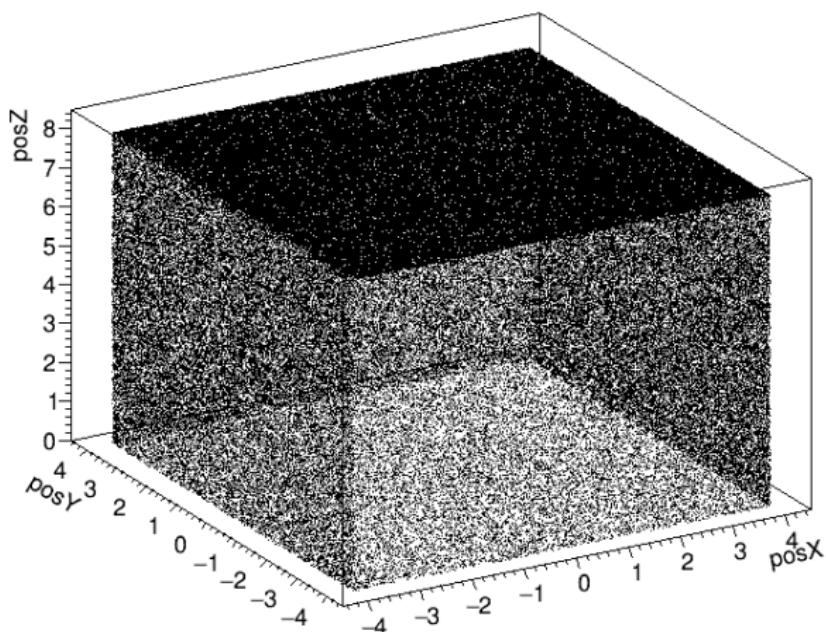
It is yet not obvious whether the  $p_r$  reconstruction is as bad as:



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

## Quality check: initial positions

Initial positions are uniformly distributed on each wall of the cube. Also, the highest particle 'surface density' of particles is observed 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)

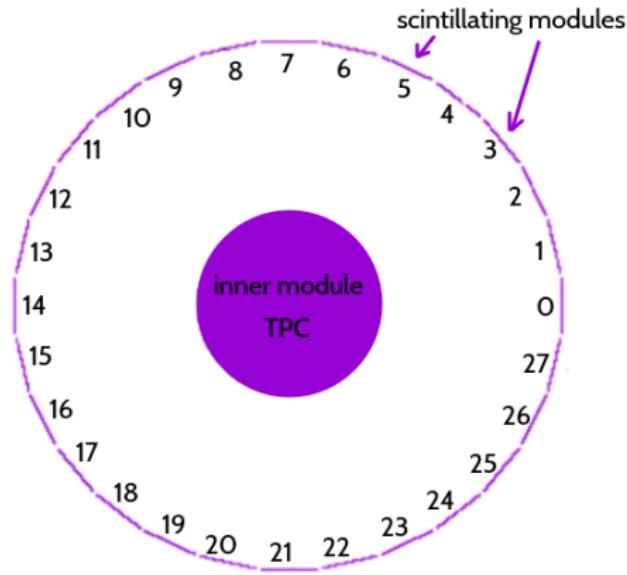
TPC parameters

- 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

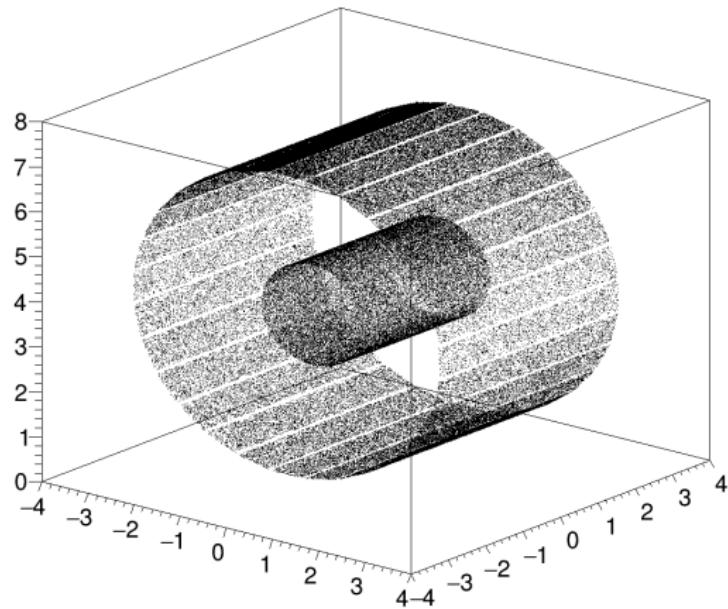
## Scintillating modules

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

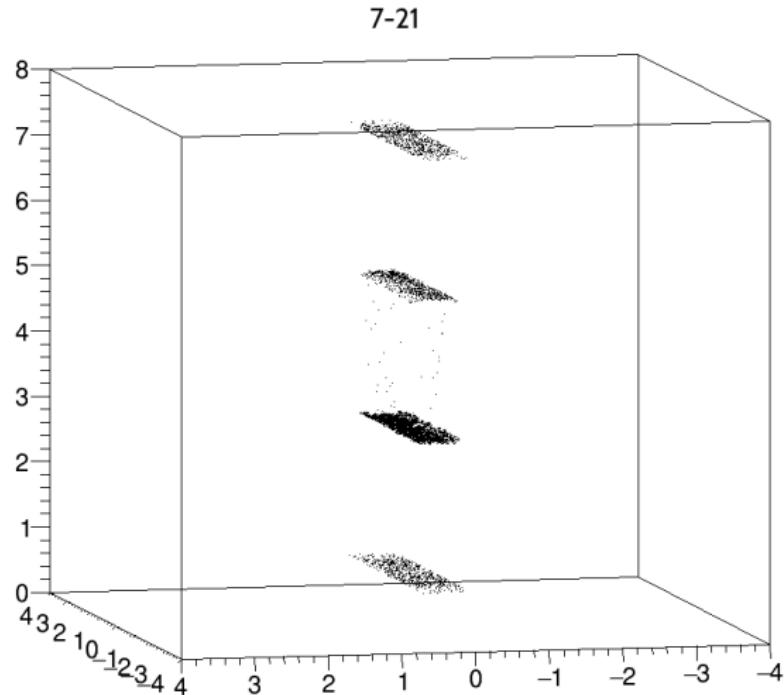


# Obtained detector geometry visualisation

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

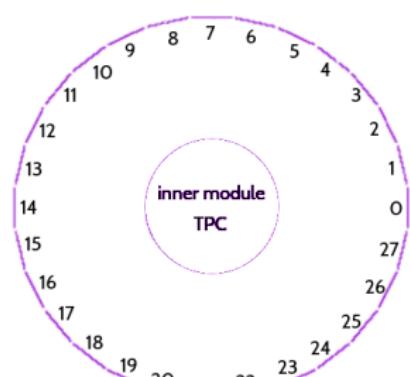
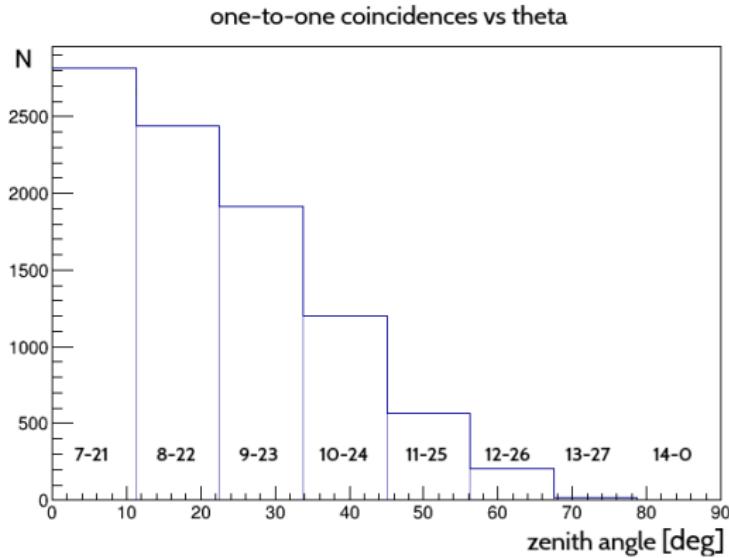


# One-to-one coincidences

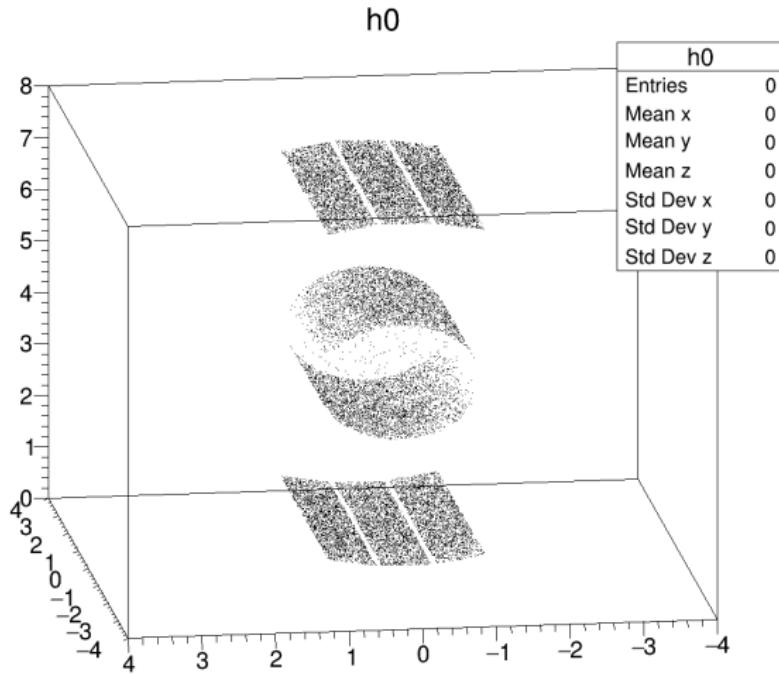


# One-to-one coincidences (with TPC)

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

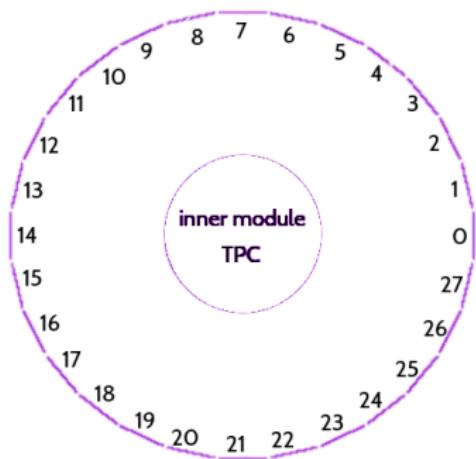


# Layout 1: modules close to each other

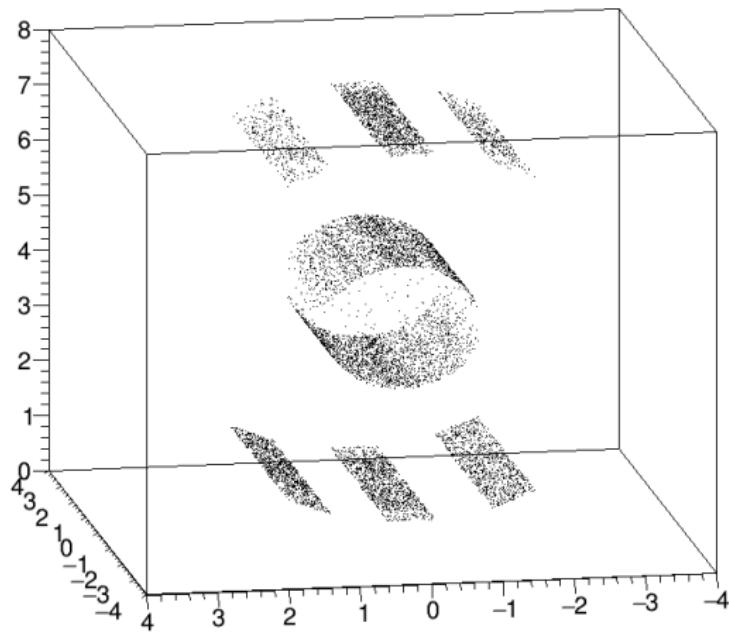


## Layout 1: modules close to each other

- (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

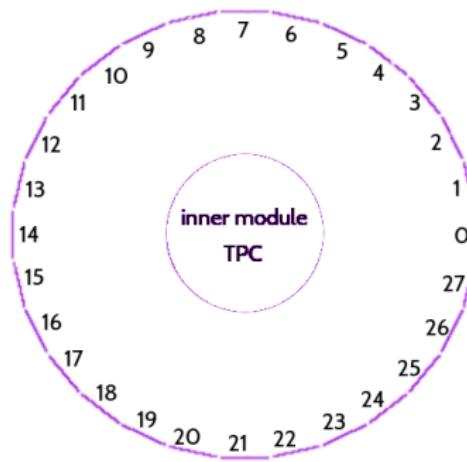


## Layout 2: space between modules



## 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