
演習H06 Scoring

(使用ファイル: P05_Scoring)

Geant4 10.3.3 準拠

Geant4 HEP/Space/Medicine 講習会資料



大学共同利用機関法人
高エネルギー加速器研究機構

本資料に関する注意

- 本資料の知的所有権は、高エネルギー加速器研究機構およびGeant4 collaborationが有します
- 以下のすべての条件を満たす場合に限り無料で利用することを許諾します
 - 学校、大学、公的研究機関等における教育および非軍事目的の研究開発のための利用であること
 - Geant4の開発者はいかなる軍事関連目的へのGeant4の利用を拒否します
 - このページを含むすべてのページをオリジナルのまま利用すること
 - 一部を抜き出して配布したり利用してはいけません
 - 誤字や間違ないと疑われる点があれば報告する義務を負うこと
- 商業的な目的での利用、出版、電子ファイルの公開は許可なく行えません
- 本資料の最新版は以下からダウンロード可能です
 - <http://geant4.kek.jp/lecture/>
- 本資料に関する問い合わせ先は以下です
 - Email: lecture-feedback@geant4.kek.jp

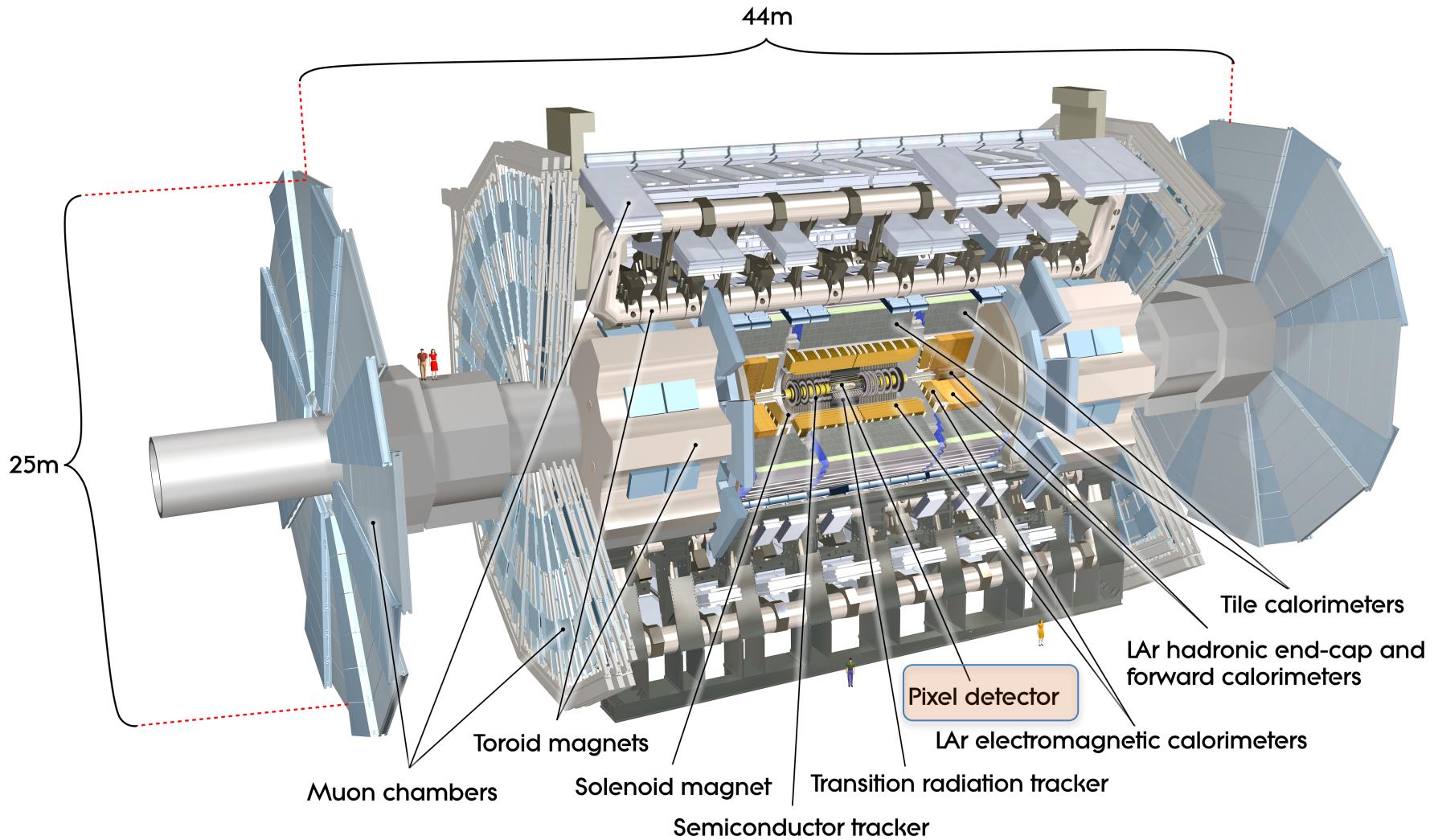
P05_Scoring/source-SD 演習の目標

P05_Scoring には4つのソースがあるが、そのうちsource-SDを使う。他の3つは自習用である。

- P02_Geometry のピクセル検出器を使って、粒子の入射位置およびシリコン中のエネルギー損失を求める。 P05_Scoring/source-SD/
 - ピクセル検出器をSensitive Detector 有感検出器とする
 - ピクセル検出器でのヒット情報を得る方法を知る
 - 解析ツールとの接続のコードと解析コマンドの使い方
 - 入射ピクセルの番号と入射位置の世界座標を求め、ヒストグラムを作成する
 - シリコン中の dE/dx を求めヒストグラムを作成する
- 余裕があれば、予備の自習教材が用意されている
 - BGO結晶中のエネルギー付与について、2通りのスコアリング手法と解析ツールとの接続を実習する。電磁シャワーの発達を調べる
 1. 自前のユーザフックを使って求める手法を理解する: P05_Scoring/source
 2. 多機能検出器と原始スコアラを使って解決する: P05_Scoring/source-PS
 - コマンドラインスコアラ: P05_Scoring/source-CS

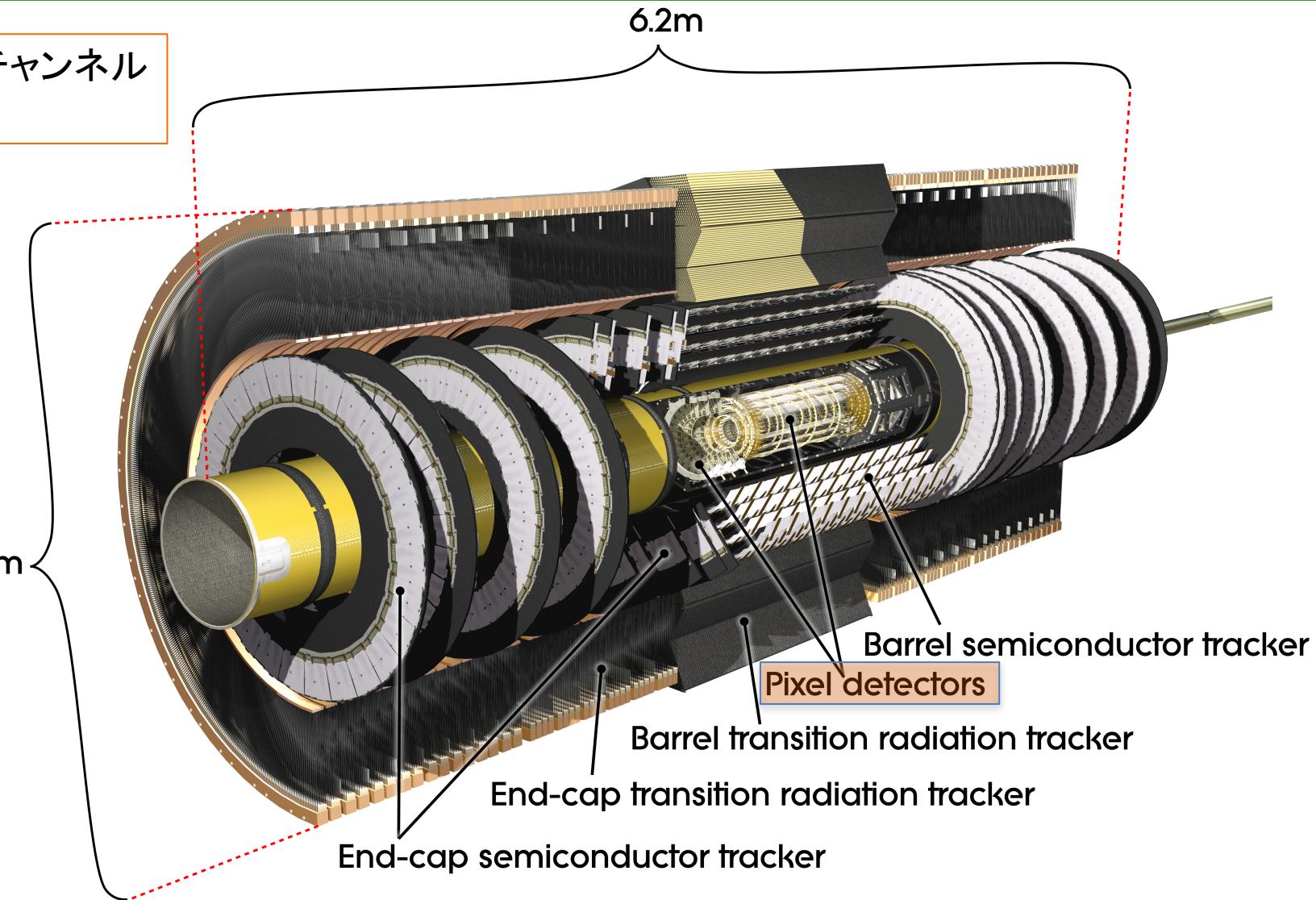
予備知識

LHC Atlas

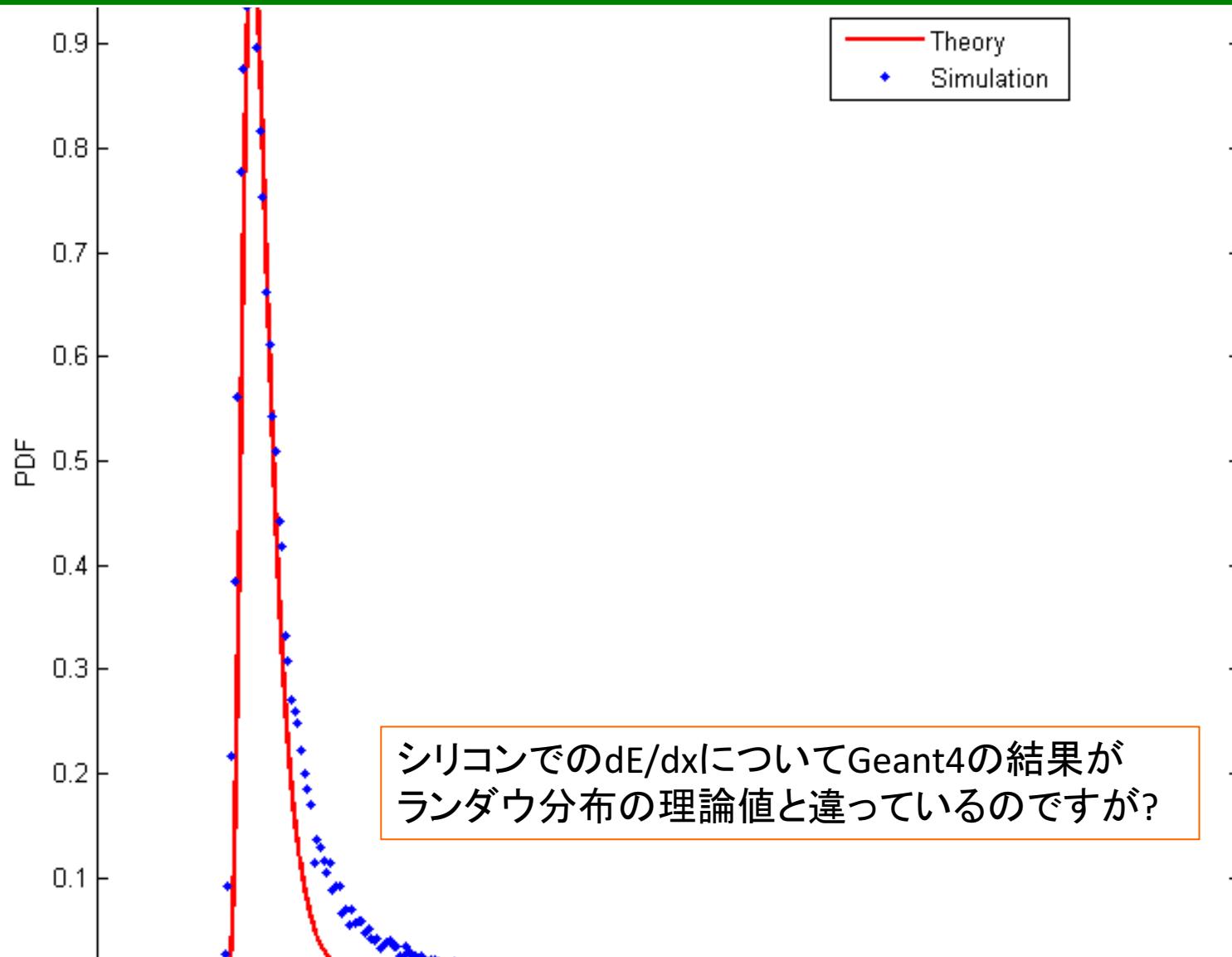


シリコン検出器拡大図

80Mチャンネル
15kW



Geant4 User Forumの質問から (electromagneticの部)



P05_Scoring/source-SD 演習

ピクセル検出器のヒットと dE/dx

P05/source-SD で調べること

■ P02のピクセル検出器を元にして、

- 粒子のヒット位置(ピクセル番号と空間座標値)を求める。
 - G4Stepからどうやってこれらの情報を入手するか
- 粒子のエネルギー損失を求める。
 - 薄いシリコンでの dE/dX について考察する。
 - Bethe-Bloch dE/dX は平均的なエネルギー損失を与える
 - 薄い物質でのエネルギー損失はランダウ分布が予測する
 - ただし、デルタ線の発生は考慮されていない
 - Geant4では Restricted Bethe-Bloch + delta raysを使い、二重勘定を防ぐためにGeant4 標準EM には G4UniversalFluctuationsを組み込んである。
- Geant4 のレンジカット
 - レンジに相当するエネルギー以上を持つ二次粒子は発生させるが、それ以下の二次粒子発生は抑止され、連続的にエネルギー損失をして止まるまでを1ステップで行う。
 - 例えば、/run/setCut 10 kmとすると、実質上、デルタ線などの二次粒子の発生は抑止され、Bethe-Bloch dE/dx だけで走る

課題1 P05_Scoring/source-SD プログラムの確認

P05_Scoring/source-SD の全体をユーザのワークディレクトリにコピーし、そのファイル構造を確認する

1) 演習プログラム全体を自分のワークディレクトリにコピーする

```
$ cd ~/Geant4Tutorial20171129  
$ cd UserWorkDir  
$ cp -r ..TutorialMaterials/P05_Scoring .
```

2) 演習プログラムのファイル構造の確認

```
$ ls P05_Scoring  
source/ source-CS/ source-PS/ source-SD/ util/  
$ ls P05_Scoring/source-SD  
Application_Main.cc      CMakeLists.txt  
include/                 src/
```

```
$ ls P05_Scoring/source-SD/include  
Analysis.hh    Geometry.hh    PrimaryGenerator.hh  
UserActionInitialization.hh RunAction.hh  SensitiveVolume.cc
```

Analysis.hh では
ROOTファイル出力を選択

```
$ ls P05_Scoring/util/  
Macros/ Macros-CS/ Macros-PS/ Macros-SD/
```

基本マクロ、実行用マクロ
ROOT解析用マクロなど

課題2 コードを読んで今からやることを確認する

■ エネルギー損失の求める

- Initialize() 積算エネルギー損失を0とする
- ProcessHits() 全てのステップでのエネルギー損失を求め、加算する。レンジカット以下のエネルギーの二次粒子のエネルギーも含める。
- EndOfEvent() 積算エネルギー損失をヒストグラムに記入する

■ ProcessHits() 粒子の入射位置を求める

- 一次粒子のステップであることの判定
 - TrackID() == 1 ならprimary particle である
- ステップのPreStepPointからピクセル素子のコピー番号を知る
- その世界座標を知る

作成するヒストグラム一覧

■ 1D, 2DヒストグラムとNtuple

| 種類 | Hist ID | 物理量 | 初期状態 | ROOTの呼び名 |
|--------|---------|----------------|------|----------|
| 1D | 0 | MotherCopyNo | 非活性 | “h1:0” |
| 1D | 1 | CopyNo | 非活性 | “h1:1” |
| 1D | 2 | Sum_eDep | 非活性 | “h1:2” |
| 2D | 0 | worldX, worldY | 非活性 | “h2:0” |
| Ntuple | 0 | コピー番号対 | | |

実行時に活性化するためのコマンド例
/analysis/h1/set 0 100 151 250
/analysis/h1/set 2 100 0. 1. MeV
/analysis/h2/setX 0 100 -1. 1. mm
/analysis/h2/setY 0 100 -1. 1. mm

RunActionでは解析ツールとの接続のお仕事

- RunActionコンストラクタでヒスト定義、非活性としておき、実行時のコマンドで設定パラメータを随時変更する

```
RunAction::RunAction(): G4UserRunAction()
```

```
{
```

```
G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();  
analysisManager->SetActivation(true);
```

コマンドで活性化できる

```
// analysisManager->SetFirstHistold(1);  
analysisManager->SetFileName("Pixel");  
// Creating histograms
```

CreateH1/2 で任意に定めた名前 "h1:1"などは
FillH1/2では使えないが、ROOTで使える。

```
G4int id = analysisManager->CreateH1("h1:0","motherCopyNumber", 100, 1, 100);
```

```
analysisManager->SetH1Activation(id, false);
```

```
id = analysisManager->CreateH1("h1:1","CopyNumber", 400, 1, 400);
```

```
analysisManager->SetH1Activation(id, false);
```

```
id = analysisManager->CreateH1("h1:2","sum_eDep", 100, 0., 5.*MeV);
```

```
analysisManager->SetH1Activation(id, false);
```

```
id = analysisManager->CreateH2("h2:0","world_x vs y", 200, -10.*mm, 10*mm, 200, -10*mm,
```

```
analysisManager->SetH2Activation(id, false);
```

ヒストのパラメータは仮のもの
コマンドで変えられる

```
...
```

SensitiveVolumeクラスの実装

- ProcessHits()では
 - TouchableHistoryを手繰って入れ子になっているレプリカのヒットピクセル番号を求める
 - ただし、シリコンに入射した一次粒子の場合のみ
 - 入射位置の世界座標を求める
 - これらをヒストグラムにfillする
 - シリコン中でのすべてのステップのenergy deposit を求め積算する
- Initialize()では
 - シリコン中で積算する量、energy deposit の和を初期化する
- EndOfEvent()では
 - 積算量をヒストグラムにfillする

SensitiveVolumeのコードスケッチ

■ 定義ファイル

```
#include "G4VSensitiveDetector.hh"
class G4Step;
//-----
class SensitiveVolume : public G4VSensitiveDetector
//-----
{
public:
    SensitiveVolume(G4String);
    ~SensitiveVolume();

    void Initialize(G4HCofThisEvent* );
    G4bool ProcessHits(G4Step*, G4TouchableHistory* );
    void EndOfEvent(G4HCofThisEvent* );

private:
    G4double sum_eDep;
    G4int no_Step;
};
```

お約束の継承／実装

シリコン中でのdE/dxの和
及び一次粒子のステップ数

SensitiveVolumeクラスのProcessHits()

■ P05_Scoring/source-SD のSensitiveVolumeクラスのヒット処理

- 一次粒子の入射位置は最初のステップだけからProcessHits()の中で求める。
 - トランクIDを取得して一次粒子かどうかを判定する。反応で生成された二次粒子は扱わない。
 - 入射位置のワールド座標を得る
 - 入射位置が属するピクセル番号を得る

```
SensitiveVolume::SensitiveVolume(G4String name) 自作の SensitiveDetector  
: G4VSensitiveDetector(name)
```

```
void SensitiveVolume::Initialize(G4HCofThisEvent*){}
```

```
void SensitiveVolume::EndOfEvent(G4HCofThisEvent*) {}
```



ヒットコレクションは
今は使わない

```
G4bool SensitiveVolume::ProcessHits(G4Step* aStep, G4TouchableHistory*)
```

```
{
```

```
    G4int nTrack = aStep->GetTrack()->GetTrackID();    トランクID==1ならprimary 一次粒子  
    G4String particleName = aStep->GetTrack()->GetDefinition()->GetParticleName(); 粒子名  
    G4StepPoint* preStepPoint=aStep->GetPreStepPoint();  
    G4ThreeVector position_World = preStepPoint->GetPosition(); ワールド座標
```

ProcessHits() で情報を得る

1行ずつ読んでゆく

```
G4bool SensitiveVolume::ProcessHits(G4Step* aStep, G4TouchableHistory*)
{
// energy deposit in this step and its accumulation over steps
    G4double edep = aStep->GetTotalEnergyDeposit();
    sum_eDep = sum_eDep + edep;
```

dE/dx とその和

```
// Retrieve information from the track object
    G4int nTrack = aStep->GetTrack()->GetTrackID();
// Primary track is picked up
    if (nTrack != 1) return false;
```

一次粒子

```
G4int nStep = aStep->GetTrack()->GetCurrentStepNumber();
G4String particleName = aStep->GetTrack()->GetDefinition()->GetParticleName();
G4StepPoint* preStepPoint=aStep->GetPreStepPoint();
G4TouchableHandle theTouchable = preStepPoint->GetTouchableHandle();
```

```
// Touchable infomation: Start position of the current step
    G4ThreeVector position_World = preStepPoint->GetPosition();
    G4ThreeVector position_Local = theTouchable->GetHistory()
                                ->GetTopTransform().TransformPoint(position_World);
```

世界座標

ヒットしたピクセル番号と座標を得る

GetCopyNumber()の引数で入れ子となった構造体階層を識別する

```
G4StepPoint* preStepPoint=aStep->GetPreStepPoint(); //ステップの始点を得て  
G4TouchableHandle theTouchable = preStepPoint->GetTouchableHandle();
```

```
G4String volumeName = theTouchable->GetVolume()->GetName(); //“LogVol_PixElmt”  
G4int copyNo = theTouchable->GetCopyNumber();  
G4String motherVolumeName;  
G4int motherCopyNo;  
if (volumeName != “LogVol_PixElmt” ) {  
    motherVolumeName = theTouchable->GetVolume(1)->GetName(); //“LogVol_PixEnvL”  
    motherCopyNo = theTouchable->GetCopyNumber(1);  
}
```

木構造の階層を指定する
には引数に整数を与える

ピクセルのレプリケーションは先ずY方向ついでX方向としているので、
X方向はcopyNo
Y方向はmotherCopyNo

ヒストのFill

■ Initialize(), EndOfVolume(), ProcessHits() の仕事

```
void SensitiveVolume::Initialize(G4HCofThisEvent*)
{
    sum_eDep = 0.;

}

void SensitiveVolume::EndOfEvent(G4HCofThisEvent*)
{
    G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
    analysisManager->FillH1(2, sum_eDep);
}

G4bool SensitiveVolume::ProcessHits(G4Step* aStep, G4TouchableHistory*)
{
    ...
    sum_eDep = sum_eDep + edep;
    //一次粒子ならヒスト
    analysisManager->FillH1(1,motherCopyNo);
    analysisManager->FillH1(0,copyNo);
    analysisManager->FillH2(0,position_World.x(), position_World.y());
    ...
}
```

FillH1, FillH2 では RunAction で
決めた ID 番号を使う

Geometry クラスではSensitive Detectorの割り付け

P05_Scoring/source-SD では、P02_Geometry のピクセル検出器のピクセル要素論理物体(logVol_PixElmt)を有感検出器とする。

そのために、P02/src/Geometry.ccにつぎのような4行を追加をするだけでよい。

ピクセル要素を有感検出器とする

G4VSensitiveDetectorの実装クラスSensitiveVolume

Geometry::Construct(){

//の中で追加

```
SensitiveVolume* aSV = new SensitiveVolume("Pixel"); //Pixelという名前をつける  
logVol_PixElmt->SetSensitiveDetector(aSV); // 論理物体logVol_PixElmtにアサインする  
G4SDManager* SDman = G4SDManager::GetSDMpointer(); // SDマネージャのインスタンス  
SDman->AddNewDetector(aSV); // SDマネージャに登録
```

課題3 P05_Scoring/source アプリケーションのビルド

■ ビルドとコンパイル

1) P05_Scoringへ移動しビルドするための作業デレクトリを作成

```
$ cd P05_Scoring  
$ mkdir build
```

2) buildデレクトリでビルドを実行

```
$ cd build  
$ cmake ../source-SD  
$ make  
$ make install
```

cmakeでビルドを実行する時の慣用句

3) ビルド実行後のデレクトリ構造確認

```
$ cd ..  
$ ls  
bin/ build/ source-SD/ ...  
$ ls bin  
Application_Main*
```

4) 実行用ディレクトリ作成とマクロのコピー

```
$ mkdir TestBench  
$ cd TestBench  
$ cp ../util/Macros/* .
```

課題4 Application_Mainの実行準備

Application_Main を実行する準備

- 1) プログラム実行を行う作業デレクトリ TestBenchを新たに作成して、実行に必要なマクロファイルをコピーする

```
$ pwd  
...../P05_Scoring ←  
$ ls  
TestBench/ bin/ build/ source-SD/  
$ cd TestBench  
$cp ../utils/Macros/* .  
$cp ../util/Macros-SD/* . ←
```

P05_Scoringへ移動する
このアプリ実行のマクロ ROOTのマクロ をTestBenchにコピー

- 2) \$./Application_Main

Qt Tips!

Zoom upしてRGB色の座標軸がうるさい時は、Qt画面左上の SceneTree を展開してAxesのチェックを外すと良い。

課題5 端末出力を読む

■ アプリを立ち上げ、run0.mac マクロを実行する

- a. Gunを粒子mu-, エネルギー1 GeV, 位置(0., 1., -1.) cm, 方向(0., -1., 1)
- b. /tracking/verbose 0 としている
- c. 可視化の設定はviewpoint と zoomを調節している
- d. beamOn 1 で端末への出力を読む

- ✓ なぜmu-が何回も表示されるのか
- ✓ copyNumber, motherCopyNumber, worldX, worldY は？
- ✓ Sum_eDepがpixelを通過するごとにどうなるか
 - SensitiveVolume.cc コードを読む

1. /tracking/verbose 1 として出力を読む

■ 次に、各ピクセルでの信号がどうなるかを考える(頭の体操だけ)

1. 各ピクセルごとにエネルギー損失を求める
2. ピクセルごとのエネルギー損失から電子の数を見積もる
 - 平均一電子当たり3eVのイオン化エネルギーとする
3. 電子の位置をミュー粒子の飛跡に沿ってばら撒く。(G4はここまで止めるか)
4. 電極間の電場を求めておいて、電子のドリフトをシミュレーションし、検出信号パルスの波形を求める。(そこまでやるか？)
5. などなど

課題6 エネルギー損失のヒストグラムを作成して表示する

- Run1.mac を実行する
 - マクロを実行するとPixel.rootファイルが出力される。
 - アプリは実行状態のままとして起き、別の端末コンソールからROOTを立ち上げる。その際、PlotPixel1.CC マクロをROOTコマンドの引数に与えてプロットの様子を見る。
- Run1.macの改善
 - シリコン中での dE/dx はおよそ83keV/300μmであることから、ヒストグラム "h1:2" の binning、最小値、最大値、単位 を決め、run2.macを編集する。
 - Run1.macを実行すると、Pixel.rootファイルが新しいもので上書きされる。
 - ROOTでPlotPixel.CCを実行する。
- RunAction.ccでヒストの初期設定をしたときに、不活性化を出来るようにして、個別のヒストの不活性化をしておいたので、上のような反復によって最適なヒスト出力を得られる。
 - 統計を稼ぐには、Application_Mainをtcshモードで立ち上げ(`~/.g4session` を編集)、/vis/disableにしてbeamOn すると良い。

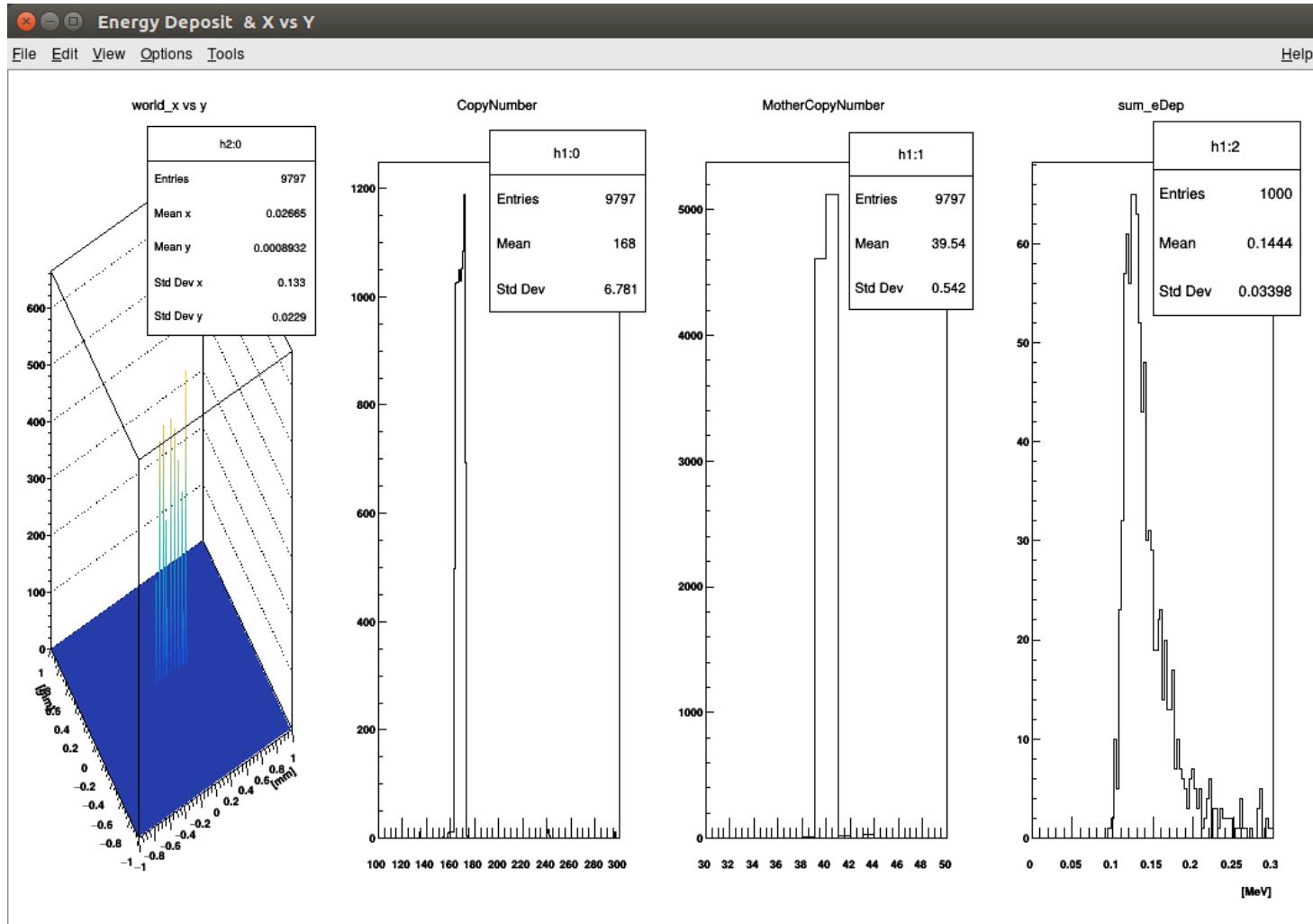
薄い物質中でのdE/dx とレンジカット

■ 粒子のエネルギー損失を求める。

- 薄いシリコン中のdE/dXは「厚い」物質の場合と同じだろうか。
 - Bethe-Bloch dE/dXは平均的なエネルギー損失を与える
 - 薄い物質でのエネルギー損失はランダウ分布が予測するが、デルタ線の発生は考慮されていない
 - Geant4では Restricted Bethe-Bloch + delta raysを使い、二重勘定を防ぐためにGeant4 標準EMには G4UniversalFluctuationsを組み込んである。
- Geant4 のレンジカット
 - レンジに相当するエネルギー以上を持つ二次粒子は発生させるが、それ以下の二次粒子発生は抑止される。
 - 例えば、/run/setCut 10 kmとすると、実質上、デルタ線などの二次粒子の発生は抑止され、Bethe-Bloch dE/dxだけで走る
 - Geant4 Cross Referenceで /run/setCut 文字列検索をすると /examples/extended/electromagnetic/TestEm* のマクロファイルにkm ほどのレンジカットを設定する例がある。その意図は何かを推定すると勉強になる。もちろん、1nmのカットの例もある。

課題7 全てのヒストのプロットを表示する

- Run2.mac を実行し、PlotPixel2.CCでROOTを動かす。



Pixel : 次の発展の例

■ AIDA 2020 Workshop on HV/HR CMOS Simulation

■ CPPM, May 12th-13th 2016

Simulation in Particle Physics: The ATLAS Pixel Detector Case

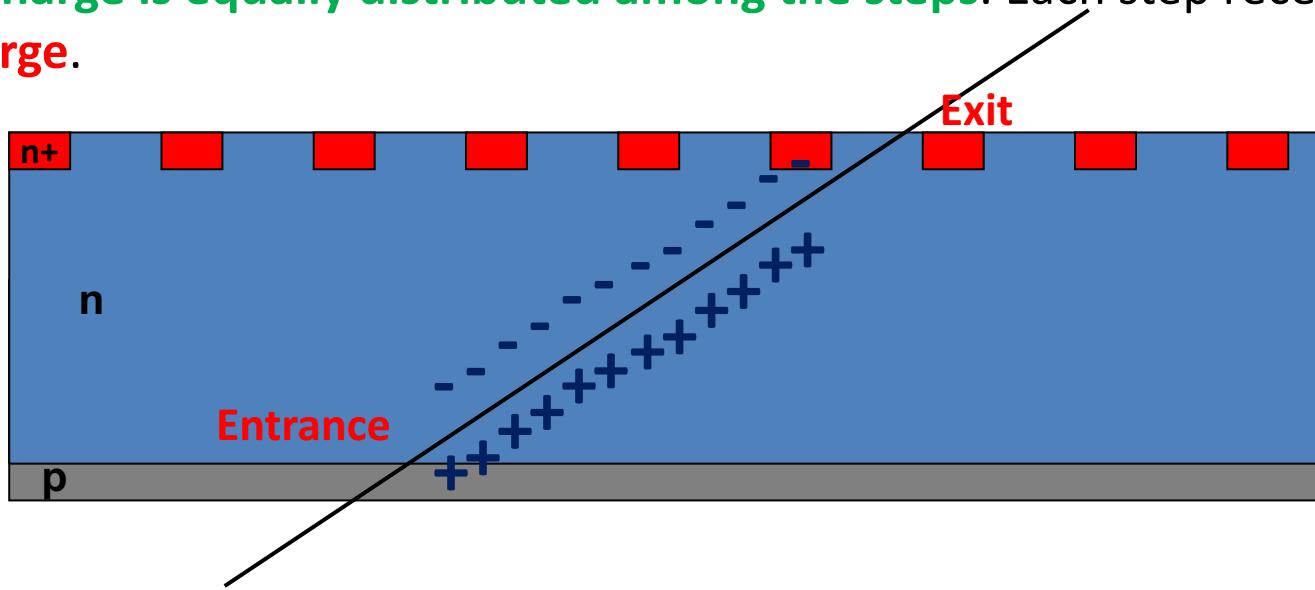
■ Farès Djama (CPPM Marseille)
■ May 12th 2016

- **Digitization** is not universal like **particle physics (Pythia)** or **particle interaction with matter (Geant4)**. It is **detector dependent**.
- So we need an example.
- Let's look into the digitization of the **ATLAS Pixel Detector**.

信号のデジタル化を目的として、次ページの考え方なら、ピクセル検出器は、今までのようにピクセルの集合としてではなく、一体ものと考えて良い、

The Drift Model (1)

- The straight line between the entrance and exit points is divided on a **number of steps** (typically 50-100) of equal length.
- **The total charge is equally distributed among the steps.** Each step receives a **local charge**.



- Each **local charge** is divided into **subcharges** (typically 5-10).
- Each **subcharge** is “**drifted**” towards the pixel plan, taking into account Electric field, magnetic field and thermal diffusion.

3つの予備の演習課題

予備演習例題 その1

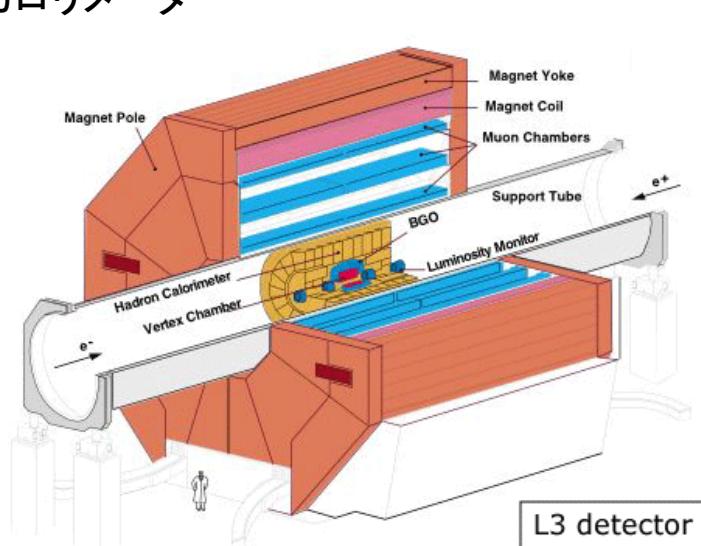
BGOカロリメータでのエネルギー測定

BGOの物理特性

密度 7.13 g/cm^3
輻射長 $X_0 = 1.12 \text{ cm}$
Moliere 半径 $R_M = 2.4 \text{ cm}$
臨界エネルギー $E_c = 10.6 \text{ MeV}$
エネルギー損失 $dE/dx = 9.2 \text{ MeV/cm}$
光減衰時間 300 ns
発光Peak波長 480 nm
屈折率 $2.20 @ 480 \text{ nm}$
 Δ Light yield $8 \times 10^3 \gamma/\text{MeV}$
耐環境性能、Rad hard

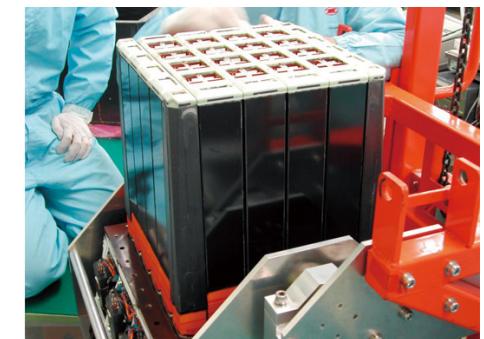
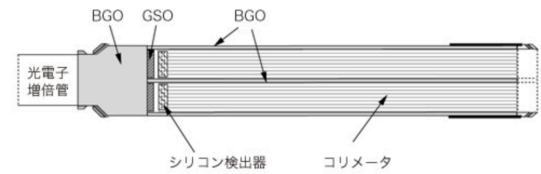
LEP L3

11488 BGOs 電磁カロリメータ
($22 X_0$)



L3 detector

すぐくの硬X線検出器
BGOはVeto役



BGO結晶：均質型全吸収力ロリメータの評価

- 均質型全吸収力ロリメータでは入射粒子の運動エネルギーが全てそのなかで吸収されるのが理想である。Geant4を使って実際の力ロリメータの性能評価を試みよう
 - 電磁シャワーの発達の様子から、縦方向及び横方向へのシャワーの漏れを調べ、
 - エネルギー分解能を調べる
 - 全吸収型力ロリメータとしての評価のためには力ロリメータが吸収したエネルギーを見積もるだけで良い。
 - シャワーでは電子、陽電子、光子が関わる電磁物理プロセスが必要である
 - シャワーが関わる物理過程ではエネルギー保存が保証されていないと全吸収エネルギーが得られない
 - Geant4では各ステップ毎にエネルギー付与が得られる。これらを積算すれば、入射粒子から発生するすべての二次、3次粒子などの相互作用の結果、結晶中にdepositされる全エネルギーが求められる。
 - どれだけの光電子から電気信号を発生させるかのシミュレーションは本講では扱わない。Geant4ではDigiというカテゴリでこの課題を扱う。
 - シンチレーション検出器では光電増倍管やシリコンフォトセンサーを用いてシンチレーション光やチエレンコフ光などを電気信号に変換し、AD変換をしてデジタル情報を得る。
 - ◆ シャワーで発生する粒子による発光／吸収は原子過程である複雑な原子構造が関わる現象であって、Geant4では直接はシミュレーションできない。
 - ✓ Geant4は可視光の反射屈折減衰などを扱えるので、シンチレータと光検出器の間の光伝送を扱える。
 - ◆ 光検出器では、光電効果の量子効率を入れて光電子数が得られ、電子增幅を経てアナログな電気量が得られる。
 - ✓ シンチレーション検出器のエネルギー分解能は光電子数の平方根に逆比例するという経験則
 - 課題) P05_Scoring/source, source-PS のBGO検出器は20個の小さいBGO結晶からなる。
 - 電磁シャワーの発達を調べ、入射粒子のエネルギーのうちどれだけがdepositされるか、漏れ出すエネルギー量はどれくらいかを調べる
-

演習の準備

P05_Scoring 予備課題プログラムファイル一覧

P02_Geometry Geometry.cc_BGO を下敷きにして寸法を大きくし、結晶の数を増やしている。
スコアリング手法として3通りを用意している

- Source: すべてをユーザアクションで行う
- Source-PS: 多機能検出器(MFD) + 原始スコアラ(PS)を使う
- Source-CS: コマンドラインスコアラ

| クラス実装 | source | Source-PS | Source-CS |
|-----------------------------|------------|-----------|------------------|
| Application_Main.cc | P02_BGOと同一 | 同左 | ScoringManager追加 |
| PhysicsList | P02_BGOと同一 | 同左 | 同左 |
| PrimaryGenerator.cc | P02_BGOと同一 | 同左 | GPS |
| Geometry.cc | 有感定義 | MFD+PS定義 | スコアリング無関係 |
| SteppingAction.cc | エネルギー付与 | 不要 | 不要 |
| EventAction.cc | 自前の積算関数 | マップの処理 | 不要 |
| RunAction.cc | ヒスト準備と始末 | 同左 | 不要 |
| UserActionInitialization.cc | 上の4つを初期化 | 上の3つを初期化 | 上の1つを初期化 |
| Analysis.hh | ROOT | 同左 | 不要 マクロでCSV |

課題:A1_P05_Scoring/source プログラムの確認

演習プログラムとして提供されているP05_Scoring/source の全体をユーザのワークディレクトリに
コピーし、そのファイル構造を確認する

1) 演習プログラム全体を自分のワークディレクトリにコピーする

```
$ cd ~/Geant4Tutorial20161129  
$ cd UserWorkDir  
$ cp -r ../TutorialMaterials/P05_Scoring .
```

2) 演習プログラムのファイル構造の確認

```
$ ls P05_Scoring  
source/ source-CS/ source-PS/ source-SD/ util/  
$ ls P05_Scoring/source  
Application_Main.cc      CMakeLists.txt  
include/      src/
```

```
$ ls P05_Scoring/source/include  
Analysis.hh  Geometry.hh      PrimaryGenerator.hh  
UserActionInitialization.hh  SteppingAction.hh  EventAction.hh  
RunAction.hh
```

ROOTファイル出力

```
$ ls P05_Scoring/util/  
Macros/ Macros-CS/ Macros-PS/ Macros-SD/
```

マクロファイルはそれぞれに

課題:A2_P05_Scoring/source アプリケーションのビルド

1) P05_Scoringへ移動しビルドするための作業デレクトリを作成

```
$ cd P05_Scoring  
$ mkdir build
```

2) buildデレクトリでビルドを実行

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

cmakeでビルドを実行する時の慣用句

3) ビルド実行後のデレクトリ構造確認

```
$ cd ..  
$ ls  
bin/ build/ source/ ...  
$ ls bin  
Application_Main*
```

4) 実行用ディレクトリ作成とマクロのコピー

```
$ mkdir TestBench  
$ cd TestBench  
$ cp ../util/Macros/* .
```

電磁シャワーの様子を観察する

課題:A 3 Application_Mainの実行

Application_Main を実行する

1) プログラム実行を行う作業デレクトリを新たに作成する

```
$ pwd  
...../P05_Scoring  
$ ls  
TestBench/ bin/ build/ source/  
$ cd TestBench  
$ ./bin/Application_Main
```

現在いるデレクトリを確認: P05_Scoringでなければ、そこに移動する

TestBenchデレクトリでApplication_Main実行
ここにヒストの出力ファイルが保存される

2) 端末ウインドに開始メッセージが出力され、続いてウインドが開く

課題:A4 Qtウインドでアプリの動作を確認

UI コマンドを使って動作をチェック

1) 以下のUIコマンドを先ず入力

```
/tracking/verbose 0  
/gun/particle e-  
/gun/energy 1 GeV  
/run/beamOn 1
```

2) analysis.hh でROOTを選んだならランの最後に

write Root file : P05.root – done

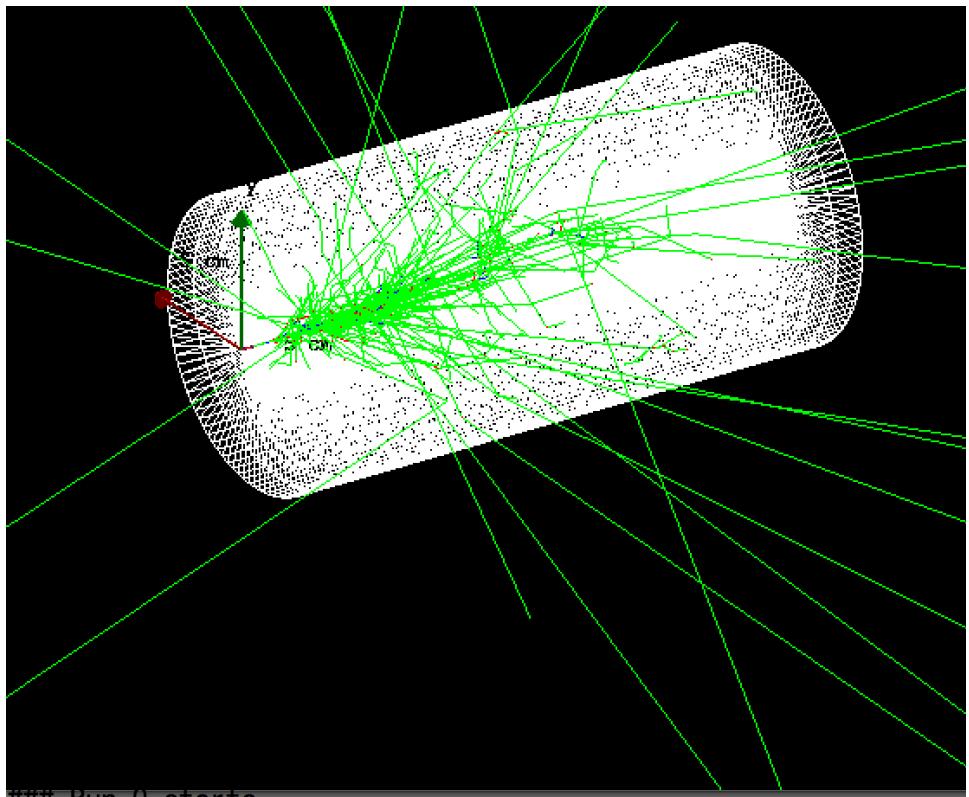
となって、一個のROOTファイルが保存される。

3) 出力ファイルの確認

```
$ls -l
```

注意

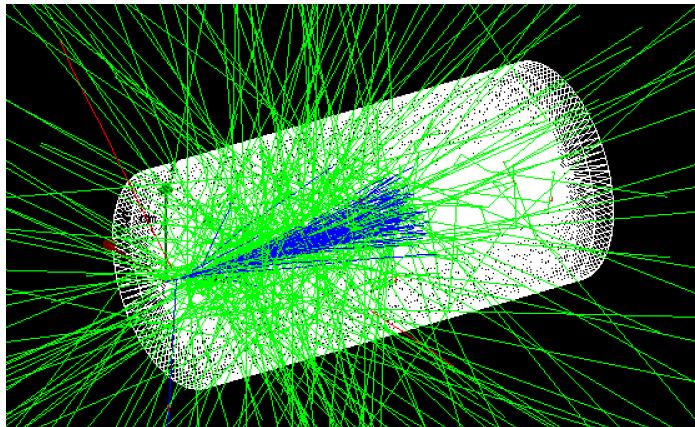
/tracking/verbose の値を上げると
実行と表示に時間がかかる
また可視化をすると時間がかかる



```
#### Run 0 starts.  
... open Root analysis file : P05.root – done  
全てのBG0でのエネルギー付与 : 924.171  
Run terminated.  
Run Summary  
Number of events processed : 1  
User=0.02s Real=0.03s Sys=0s  
... write Root file : P05.root – done  
idle> █
```

課題:A5 粒子による全エネルギー付与の違い

- 以下のコマンドでは/tracking/verbose 0として実行時間を短縮する
- /gun/energy 1 GeV とし、次のように粒子を変えては1イベント発生する
 1. /gun/particle e- “全てのBGOでのエネルギー付与 : 944.594”
 2. /gun/particle mu- “全てのBGOでのエネルギー付与 : 189.357”
 3. Proton “全てのBGOでのエネルギー付与 : 392.878”
 4. Gamma “全てのBGOでのエネルギー付与 : 935.62”
 - この違いはなぜか BGOの長さは20cm、半径は5cmである
- Proton 300 MeV で100events(時間がかかるので注意)



電磁シャワーの発達のヒストグラム

ROOTファイルからグラフ作成

課題A6： 10 GeV 入射粒子と電磁シャワーの発達

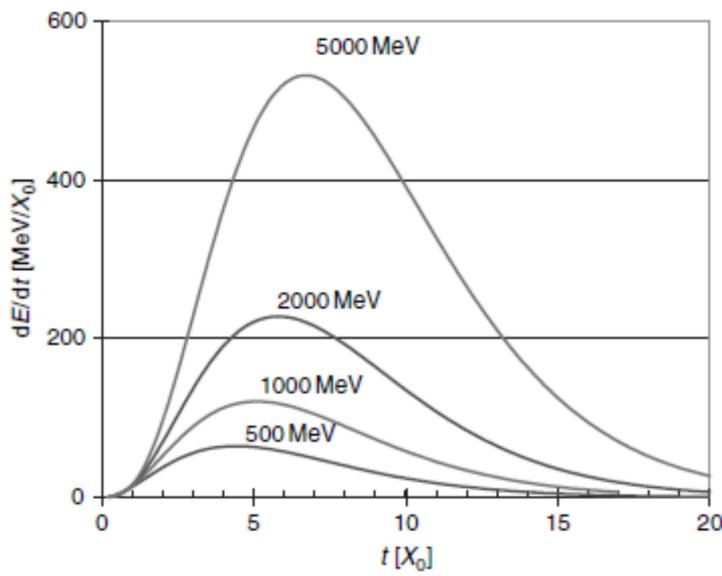
1. シャワーの縦方向と横方向のcontainmentの観察

- 統計を貯めてシャワーの軸方向の発達と総BGOエネルギーデポジットを求める
 - /tracking/verbose 0
 - /vis/disable 可視化で実行速度が遅くなるのを避ける
 - /gun/particle e- /gun/energy 10 GeV /run/beamOn 1000

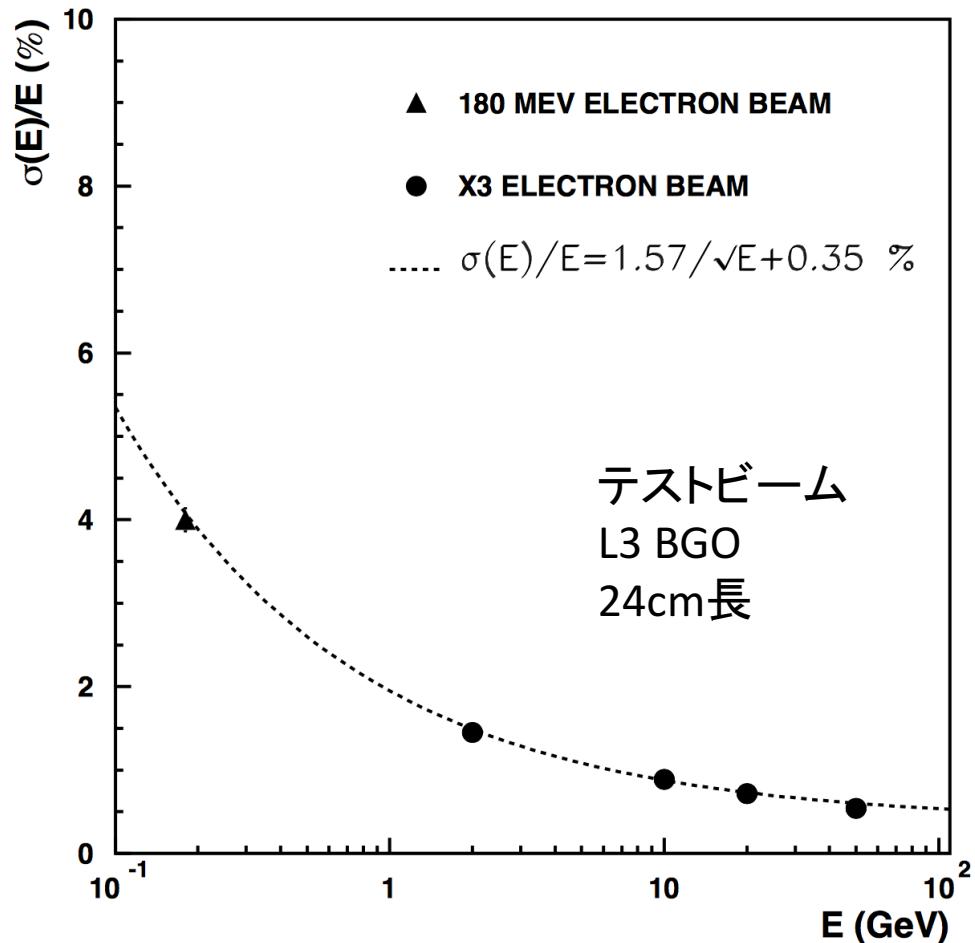
2. 電磁シャワーのシャワー形状に重要な二つの物理特性因子

- 今のGeometry.ccではBGO結晶の半径を現実的でない5cmと大きく設定している。横方向のエネルギー封じ込めRadial containment 95%にはモリエール半径 R_M の2倍が必要である。[Leroy 2.242]
- 軸方向のエネルギー封じ込めには何輻射長(radiation length) X_0 かが関わる。今のGeometryでは 1cm厚さ*20個が隙間なく並んでいる。入射電子のエネルギーEで封じ込めエネルギーの割合は軸方向長さが $3.03(\ln(E/E_c)+0.4)[X_0]$ で98%という経験式がある。[Leroy 2.234] 10GeVなら約 $22X_0$
- BGOの E_c 臨界エネルギー(critical energy)は10.6MeVで、これ以上のエネルギーを持つ電子／陽電子／ガンマは制動輻射と対生成で数を増やし、それ以下では主にイオン化でエネルギーを失う。漏れにはコンプトンが寄与する

シャワーの軸方向発達予測とテストビーム結果例



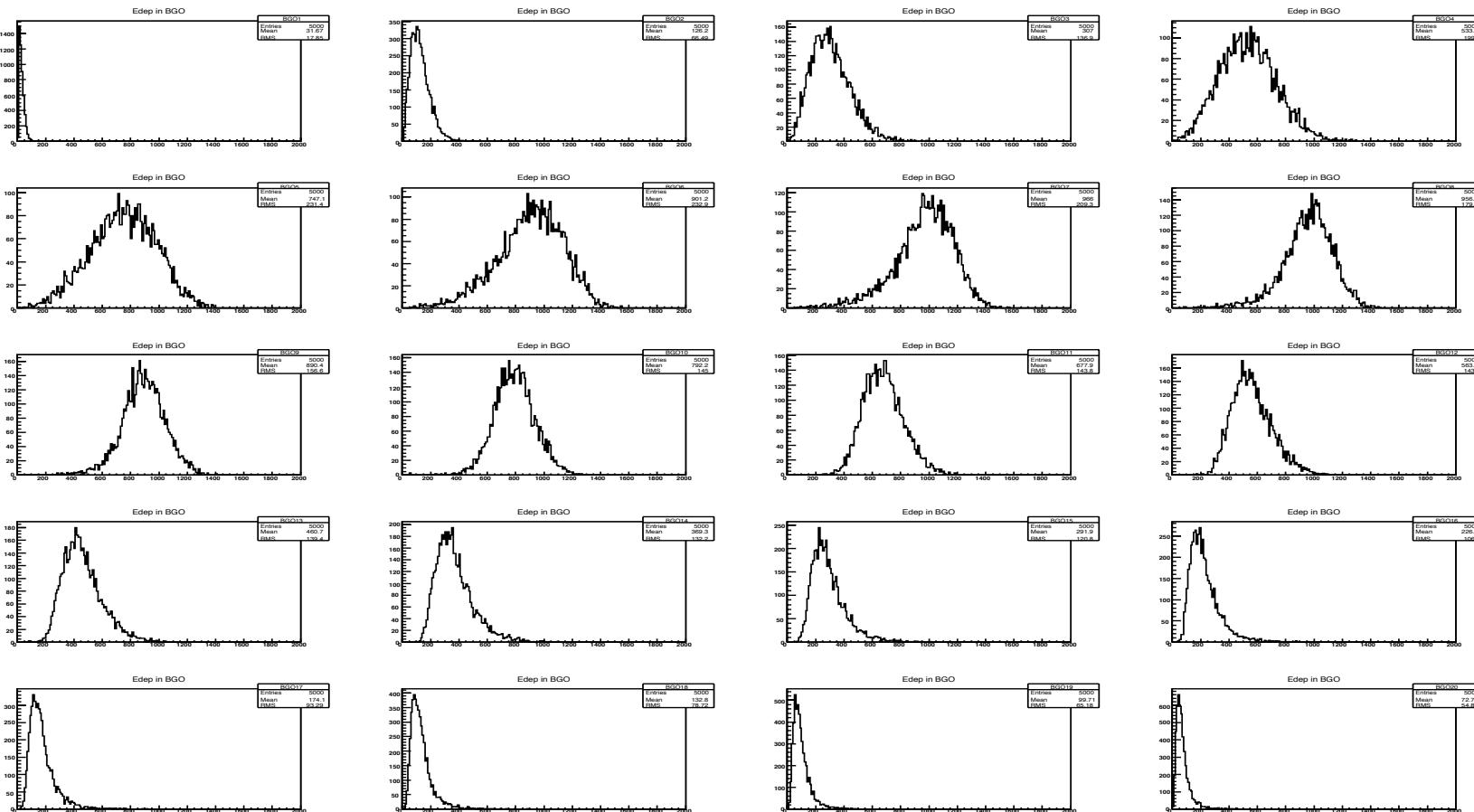
横方向には十分大きいとした時の
「理論」式



課題A7: BGO結晶#1~#20のエネルギー分布作成

10GeV e- 5000evts

ROOTでDepthBGO.ccを実行する



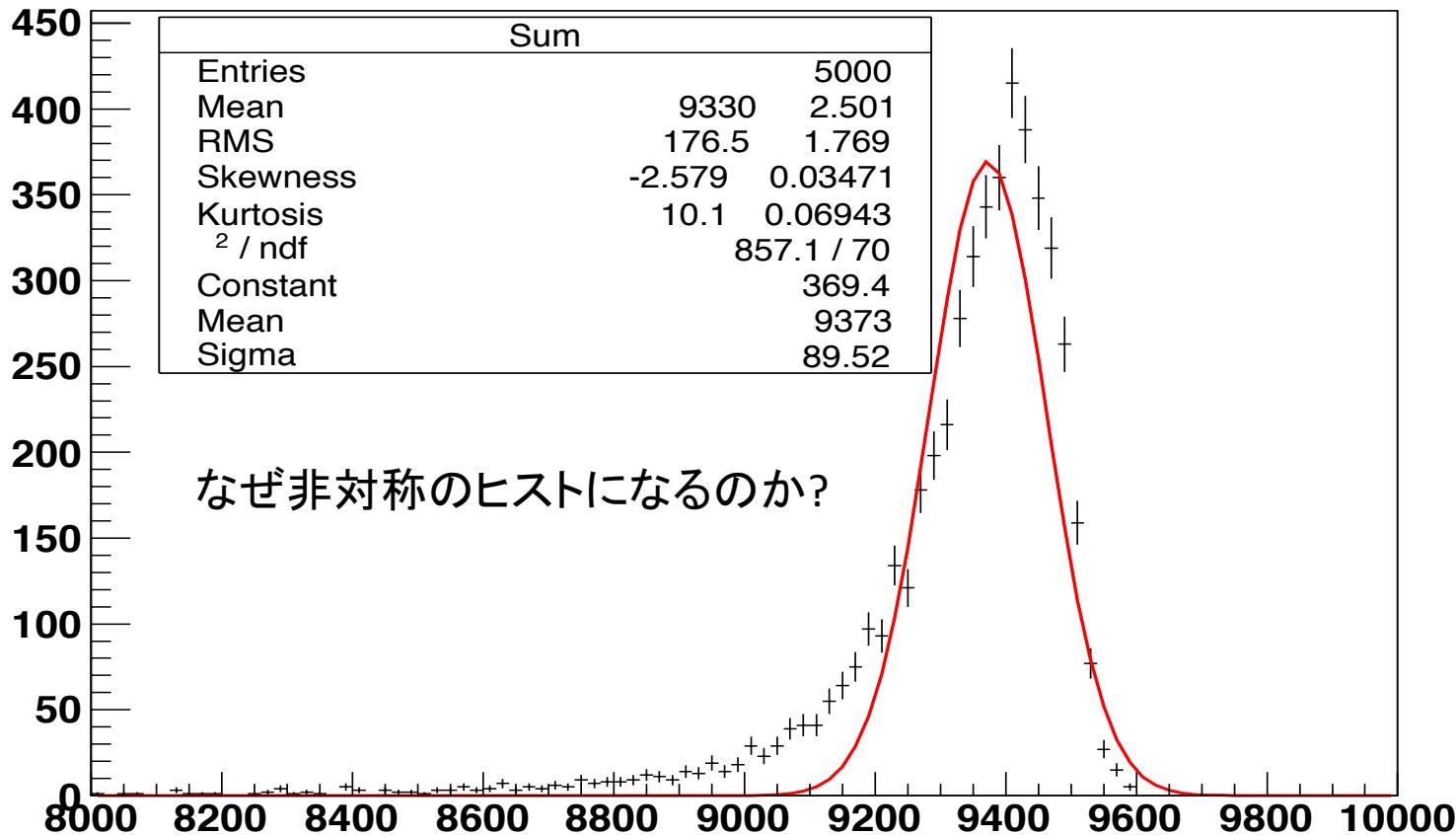
課題A8：エネルギーデポジット総和のヒスト作成

Radius_BGO = 5.0*cm leng_Z_BGO = 1.0*cm

時間があれば1.0 GeVで走らせると $\sigma/E = 1.67\%$ 程度となることを確かめよう

DepthBGO.CCを参考にするとよい

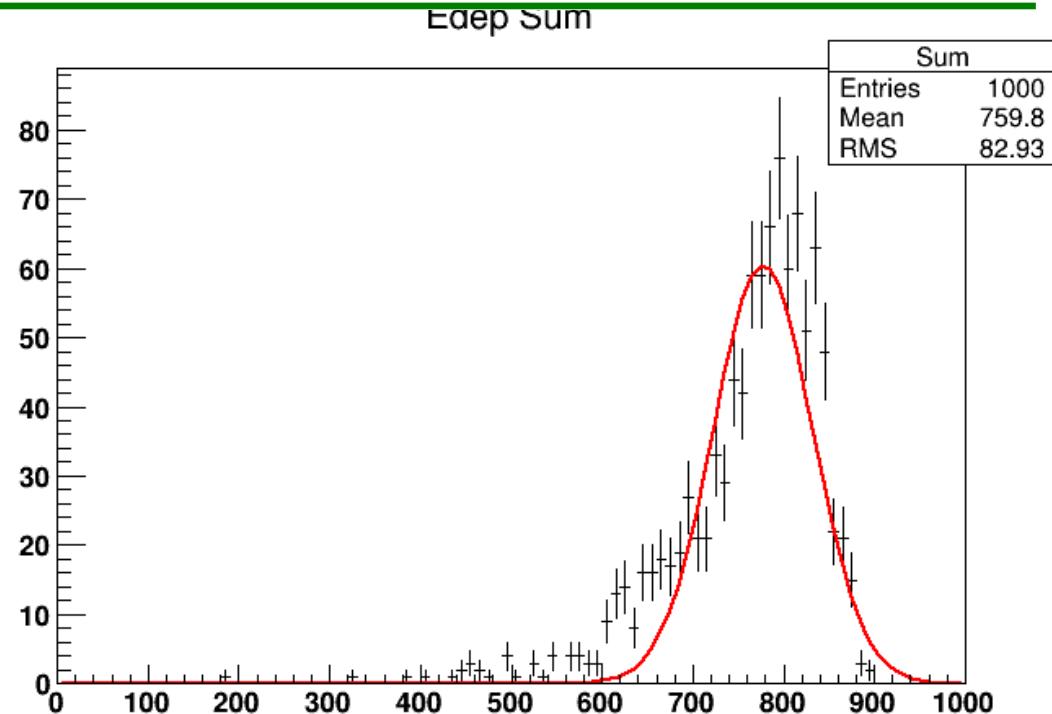
Edep Sum



予備課題A9: 縦漏れ調べ: 長さと半径を半分にすると

BGO
厚さ $0.5\text{cm} \times 20$
半径 2.5cm

電子 1GeV



Tips!

本例に見るようにカスケードシャワーの発達をすべてシミュレーションするのは計算時間がかかる。そこでGeant4ではシャワーのパラメータ表現を使って高速計算できる仕掛けがある。**パラレルワールドにスコア物体を定義**すると、物質世界での粒子のtrackingとは独立にスコア物体でのスコア量を計算できる

P05_Scorer/source プログラム解説

P05_Scoringプログラムのクラス

■ P05_Scoring/source 演習プログラムの概要

- ジオメトリ: P02_Geometryで使用したジオメトリ(Geometry.cc_BGO_One)を再利用する
 - BGO結晶の厚さ1cm半径5cmと大きくし、20個をZ方向に隙間なくCopy番号を付けて並べる。
 - 各BGO結晶をスコアする物体と定義する。(SensitiveDetectorは使わない。)
- SteppingActionでは、プレステップ点がBGOの中にあれば
 - エネルギー付与を求め、Copy番号に応じて積算する
- EventActionでは一イベントで発生したBGO結晶毎の全エネルギー付与を求め、ヒストグラムへ出力する。また全BGOでのエネルギー付与の総和もヒストする。
- RunActionではヒストグラムの準備と結果の出力を行う
 - BGO毎に一次元ヒストを用意する
 - 全BGO積算用にヒストを用意する
- UserActionInitializationをこの3つのユーザアクションを追加する
- Analysis.hh ではROOT出力とする

```
#include "g4root.hh"
//#include "g4csv.hh"
```

Geometry

P02_Geometryの
Geometry.cc_BGO_Oneの拡張

BGO結晶とスコアラー

- BGO ($\text{Bi}_4\text{Ge}_3\text{O}_{12}$)の輻射距離(radiation length)は1.12cmなので、BGO結晶は1.0cmの厚さとする。電磁シャワーの発達を思い出そう。

```
// Define 'BGO Detector'  
// Define the shape of solid  
G4double radius_BGO = 5.0*cm; // > 2*Moliere  
G4double leng_Z_BGO = 1.0*cm; // ~ 1*Rad Length  
G4Tubs* solid_BGO = new G4Tubs("Solid_BGO", 0., radius_BGO, leng_Z_BGO/2.0, 0.,  
                                360.*deg);  
  
// Define logical volume  
G4Material* materi_BGO = materi_Man->FindOrBuildMaterial("G4_BGO");  
G4LogicalVolume* logVol_BGO =  
    new G4LogicalVolume( solid_BGO, materi_BGO, "LogVol_BGO", 0, 0, 0 );  
  
// define my sensitive volume  
fScoringVol = logVol_BGO;                                    これをスコアラーとする
```

並進移動とコピー番号

■ BGO論理物体を並進移動しコピー番号を付けつつコピーする

```
// Placement of logical volume - i-th BGO
G4int noBGO = 20;                                // no of BGO crystals
G4double pos_X_LogV = 0.0*cm;                     // X-location LogV
G4double pos_Y_LogV = 0.0*cm;                     // Y-location LogV

for(G4int i=0; i<noBGO; i++){
    G4double pos_Z_LogV = 2.0*cm + (i-1)*leng_Z_BGO;      // Z-location 移動
    G4ThreeVector threeVect_LogV = G4ThreeVector(pos_X_LogV, pos_Y_LogV,
                                                pos_Z_LogV);
    G4RotationMatrix rotMtrx_LogV = G4RotationMatrix();
    G4Transform3D trans3D_LogV = G4Transform3D(rotMtrx_LogV, threeVect_LogV);

    G4int copyNum_LogV = 1000+i;                         //コピー番号は1000から1019までとする
                                                        //また、iをヒストグラムのIDとする
    new G4PVPlacement(trans3D_LogV, "PhysVol_BGO", logVol_BGO, physVol_World,
                      false, copyNum_LogV); }
```

スコアラ物体をSteppingActionへ渡す用意

- Geometry.hhにIn line 関数GetScoringVol()を用意しておく

Include/Geometry.hh から抜粋

```
G4LogicalVolume* GetScoringVol() const { return fScoringVol; }      //in line 関数  
protected:  
    G4LogicalVolume* fScoringVol;
```

- SteppingActionはこの関数を使ってプレステップポイントがBGO中に在ることを判定する(後で紹介)

SteppingAction

SteppingAction.hh

■ EventActionとG4LogicalVolumeとの間

```
#include "G4UserSteppingAction.hh"
#include "globals.hh"
class EventAction;
class G4LogicalVolume;
// Stepping action class
class SteppingAction : public G4UserSteppingAction
{
public:
    SteppingAction(EventAction* eventAction);
    virtual ~SteppingAction();
    // method from the base class
    virtual void UserSteppingAction(const G4Step*);

private:
    EventAction* fEventAction;
    G4LogicalVolume* fScoringVol;
};
```

SteppingAction.cc

- プレステップポイントがBGO中に在る時、そのステップでのエネルギー付与を求める、イベントに渡ってそれを積算する。イベントの終わりに積算量を出力する

```
SteppingAction::SteppingAction(EventAction* eventAction)
    : G4UserSteppingAction(), fEventAction(eventAction), fScoringVol(0){}

void SteppingAction::UserSteppingAction(const G4Step* step)
{
    if (!fScoringVol) {
        const Geometry* geometry           //Geometry のConstruct()
            = static_cast<const Geometry*>(G4RunManager::GetRunManager()->GetUserDetectorConstruction());
        fScoringVol = geometry->GetScoringVol();
    }
    // 今のステップがある論理物体とコピー番号はプレステップ点のタッチャブルから得る
    G4TouchableHandle theTouchable = step->GetPreStepPoint()->GetTouchableHandle();
    G4LogicalVolume* volume = theTouchable->GetVolume()->GetLogicalVolume();
    G4int copyNo = theTouchable->GetCopyNumber();
    // スコアする論理物体かどうか
    if (volume != fScoringVol) return;
    // このステップのエネルギー付与と積算
    G4double edepStep = step->GetTotalEnergyDeposit();
    fEventAction->AddDep(copyNo-1000, edepStep);      //sum_eDep[] in EventAction.hh
}
```

EventAction

EventAction.hh

- eDepの和 sum_eDep[] を求める関数を用意

```
#include "G4UserEventAction.hh"
#include "globals.hh"
/// Event action class
Enum{noBGO=20};
class EventAction : public G4UserEventAction
{
public:
    EventAction();
    virtual ~EventAction();
    virtual void BeginOfEventAction(const G4Event* event);
    virtual void EndOfEventAction(const G4Event* event);
    inline void AddeDep(G4int i, G4double de) {sum_eDep[i] += de;}
    inline G4double GetSum(G4int i){return sum_eDep[i];}
Private:
    G4double sum_eDep[noBGO];
};
```

EventAction.cc

- ステップに渡って積算したエネルギー付与を出力する

```
EventAction::EventAction()
: G4UserEventAction(){}
EventAction::~EventAction(){}
void EventAction::BeginOfEventAction(const G4Event* /*event*/)
{ for(G4int i = 0; i<noBGO; i++){ sum_eDep[i] = 0.; } }          イベント毎に初期化
void EventAction::EndOfEventAction(const G4Event* /*event*/)      イベント終わり
{
    G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
    G4double sum_BGO = 0.;
    for(G4int i=0; i<noBGO; i++){
        G4cout << "sum_eDep["<<i+1000 << "] = "<< GetSum(i)<<G4endl;
        analysisManager->FillH1(i+1, GetSum(i));    個別のBGO結晶でのヒストグラム
        sum_BGO = sum_BGO + GetSum(i);
    }
    G4cout << "Total energy deposit in BGOs : "<< sum_BGO <<G4endl;
    analysisManager->FillH1(21, sum_BGO);          すべてのBGO結晶での総和
}
```

RunAction

ヒストグラム生成

■ コンストラクタ中で

```
RunAction::RunAction()
: G4UserRunAction()
{
// Create analysis manager
    G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
    analysisManager->SetVerboseLevel(1);
    analysisManager->SetFirstHistold(1);      // Histo Id starts from 1 not from0
// Creating histogram IDs
G4String Hist_Name[20]=
{"BGO1", "BGO2", "BGO3", "BGO4", "BGO5", "BGO6", "BGO7", "BGO8", "BGO9", "BGO10",
 "BGO11", "BGO12", "BGO13", "BGO14", "BGO15", "BGO16", "BGO17", "BGO18",
 "BGO19", "BGO20"};

for(G4int i=0; i<20; i++){    各BGO結晶のeDep
    analysisManager->CreateH1(Hist_Name[i], "Edep in BGO", 100, 0., 1000.*MeV);
}
analysisManager->CreateH1("Sum", "Edep Sum", 100, 0., 10000.*MeV); 全BGO
```

ヒスト保存用ファイルの準備と書き出し

■ Begin/End of RunAction

```
void RunAction::BeginOfRunAction(const G4Run* /*run*/)
{
    // Get analysis manager and open an output file
    G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
    G4String fileName = "P05";
    analysisManager->OpenFile(fileName);
}

void RunAction::EndOfRunAction(const G4Run* /*run*/)
{
    //save histograms & ntuple
    G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
    analysisManager->Write();
    analysisManager->CloseFile();
}
```

UserActionInitialization

UserActionInitialization.cc

- オプションのユーザアクションにEventAction, SteppingAction, RunActionを追加

```
void UserActionInitialization::Build() const
//-----
{
    EventAction* eventaction = new EventAction();
    SetUserAction( new PrimaryGenerator() );
    SetUserAction(eventaction);
    SetUserAction( new SteppingAction(eventaction) );
    SetUserAction( new RunAction());
}
```

例題その2：多機能検出器と原始スコア

P05_Scoring/source-PS/

課題B1：原始スコアラ利用コード:コンパイルして実行

- P05_Scoring/source-PSを使う。Build-PSというディレクトリを作りそこでコンパイルする。

```
$ls source-PS/src
EventAction.cc           RunAction.cc
Geometry.cc              UserActionInitialization.cc
PrimaryGenerator.cc
$ mkdir build-PS
$ cd bin; mv Application_Main Application_1
$ cd ../build-PS
$ cmake ..../source-PS
$ make; make install
$cd ../TestBench
$cp ..../util/Macros-PS/* .
$ ../bin/Application_Main
```

Sourceとの違いは少しだけ:

- SteppingActionがない
- Geometryでの有感定義は多機能検出器と原始スコアラに置き換え
- EventActionはマップのキーからコピー番号を得るように変更
- RunActionは同じ

source-PS プログラムの要点

- 多機能検出器と原始スコアラーを使うようにGeometryを変更
 - G4VPhysicalVolume* Geometry::Construct()で次のように書く

```
#include "G4MultiFunctionalDetector.hh"
#include "G4VPrimitiveScorer.hh"
#include "G4PSEnergyDeposit.hh"
略
// Sensitive detector is a MultiFunctionalDetector
G4MultiFunctionalDetector* BGODetector
    = new G4MultiFunctionalDetector("BGO");
                                         名前をつけて多機能検出器を作る
SetSensitiveDetector("LogVol_BGO",BGODetector);  BGO論理物体を多機能検出器に
G4VPrimitiveScorer* primitive;
    primitive = new G4PSEnergyDeposit("Edep");  名前をつけてスコアラを作る
BGODetector->RegisterPrimitive(primitive);      /BGO/Edep でヒットマップを得る
G4SDManager::GetSDMpointer()->AddNewDetector(BGODetector);
```

最後の行が無いと、10.3では実行時にコアダンプする。

EventActionではヒットマップを扱う

- “BGO/Edep”という名前のヒットコレクションが作られている

```
void EventAction::EndOfEventAction(const G4Event* event)
{
fBGOEdepHCID
    = G4SDManager::GetSDMpointer()->GetCollectionID("BGO/Edep");
G4double BGOEdep = GetSum(GetHitsCollection(fBGOEdepHCID, event));
```

```
G4double EventAction::GetSum(G4THitsMap<G4double>* hitsMap) const
{
G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
G4double sumValue = 0;
G4double edep = 0.;                                マップからコピー番号と値を取り出して
std::map<G4int, G4double*>::iterator it;
for ( it = hitsMap->GetMap()->begin(); it != hitsMap->GetMap()->end(); it++) {
    edep = *(it->second);
    sumValue += edep;           //キーから得られる値がedep
    G4int key = (it->first);   //キーの値は1000から始まるコピー番号
    analysisManager->FillH1(key-999, edep);    //ヒストをfill
    //G4cout << "iterator : " << key << " edep: " << edep << G4endl;
}
return sumValue;
}
```

ヒット情報を扱うためのSTL

- なぜStandard Template Libraryを理解せねばならないのか
 - STLは汎用のテンプレート形式のクラスと関数を提供し、よくつかわれるデータ構造とその操作のアルゴリズムを実装している
 - 3つの基本要素はコンテナ、アルゴリズムと反復子
 - STLは複雑で分かりにくいし、構文形式も暗号めいているが、ユーザによって様々で複雑になりうるヒット情報を統一的な手法で扱うために採用されているので、スコアリングでは避けて通れない。
 - 例題のコードを、C++の教科書を横に置いて一行ずつ読むのが一番
- Geant4では次の二つのコンテナを使用している
 - G4HitsCollection std::vector 添字による配列 一般的なヒットを扱う
 - G4HitsMap std::map キーによる検索 原始スコアラで使う

予備の例題 その3 コマンドラインスコアラー

Command-line Scoring

P05_Scoring/source-CS/ は
スコアリングのプログラム無しでもここまでできる例

特長と適用範囲

- パラレルワールドに直方体または円筒形を置き、スコアリングメッシュを切るとメッシュ毎に物理量のランの通しの総和を求められる。
 - 物理量としては原始スコアラーと同様
 - メッシュの設定、分割、スコア物理量、出力ファイルなどをマクロファイルに記述する
 - イベント毎のスコアは得られない。メッシュとマスボリュームの境界が一致しないメッシュでのスコアは正しくない
- ソースコード P05_Scoring/source-CS
 - パラレルワールドでメッシュを切るので、Geometryクラスでは20個の結晶を同じ大きさの単結晶一個で置き換えて簡単化した。
 - ステッピング、イベント、ランアクションは不要
 - メインプログラムで、ランマネージャのインスタンス生成直後にスコアリングマネージャを生成する。C++コードはこれだけでよい。

```
// Construct the default run manager
G4RunManager * runManager = new G4RunManager;
// Activate UI-command base scorer 直後にコマンドベーススコアマネージャ
G4ScoringManager * scManager = G4ScoringManager::GetScoringManager();
scManager->SetVerboseLevel(1);
```

マクロ例

■ Macro-CS/cylMesh.mac

```
# define scoring mesh  
/score/create/cylinderMesh cylMesh_1  
/score/mesh/cylinderSize 50. 100. mm  
/score/mesh/translate/xyz 0 0 0. mm  
# R Z Phi division  
/score/mesh/nBin 50 1 1  
#  
/score/quantity/energyDeposit eDep  
#  
/score/close  
/score/list
```



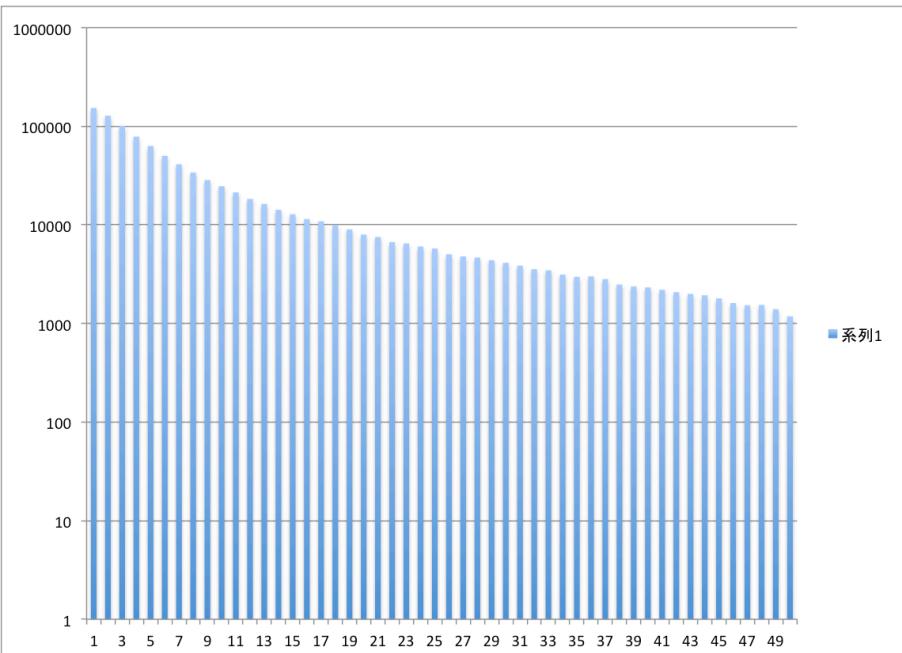
```
/control/verbose 2  
/tracking/verbose 0  
# define gun and start a run  
/gps/particle e-  
/gps/energy 10 GeV  
/gps/position 0 0 -20 cm  
/gps/direction 0 0 1  
/vis/disable  
/run/beamOn 100  
/vis/enable  
#  
  
# Dump scores to a file  
/score/dumpQuantityToFile cylMesh_1 eDep eDep.csv
```

BGO結晶と重なる円筒をRまたはZ方向にメッシュに切って、エネルギーdepositをスコアランの後、eDep.csvという名前のファイルに書き出す。
結果を表計算ソフトで処理すると次のようになる。

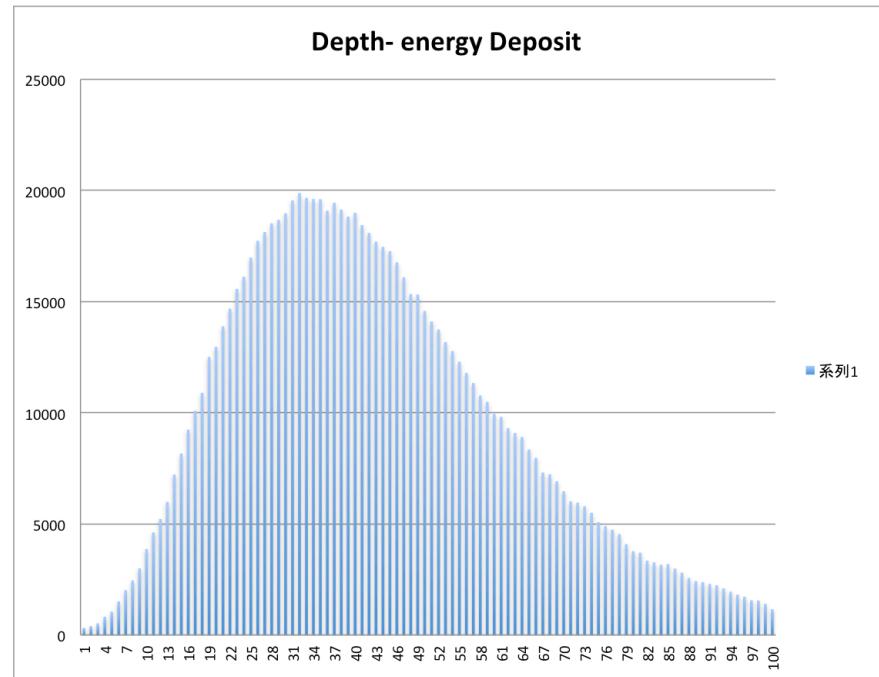
課題C1：コンパイルしてシャワーの発達の様子を調べる

半径 50mm 長さ 200mm 単一結晶
入射電子 10GeV

cylMeshR.mac R分布 1mmステップ



cylMeshZ.mac Z分布 2mmステップ



総デポジットエネルギー 9.369 GeV

課題C2: Bragg Peak proton入射 深さ一吸收線量分布

マクロファイル cylMeshZ-proton.mac を実行する

