2021 年度卒業論文

シリコンピクセルを用いた プリシャワー検出器の Geant4 シミュレーションによる検討

奈良女子大学理学部
物理学コース
高エネルギー物理学研究室
五屋郁美

2021 年 5 月 7 日

概要

電磁カロリメーターとは、電子や光子のエネルギーを電磁シャワーという現象を用いて測定する測定器である。BelleⅡ実験で用いられる電磁カロリメーターは、γ 線を高いエネルギー分解能で検出する目的でヨウ化セシウム (CsI) にタリウム (Tl) をドープした CsI(Tl) 結晶シンチレータと PIN フォトダイオードを用いている。CsI(Tl) は発光量が大きいが減衰時間が長い。そのため、高輝度ビーム蓄積に伴って発生するビームバックグラウンドで発生した γ 線によりパイルアップが発生し、エネルギー分解能が低下する。

バックグラウンドの γ 線の影響を小さくするための方法として、既存のカロリメーターの前に時間応答が速い検出器を置き、低エネルギーのバックグラウンド γ 線がここで相互作用をして、CsI(Tl) カロリメーターに与える影響を減らすことが考えられる。これをプリシャワー検出器と呼び、本研究では γ 線の到来方向を測定する機能を持たせることについて調べた。Geant4 シミュレーションにより BGO 結晶シンチレータからなるアクティブ吸収層とシリコンピクセル検出器で構成するシャワー中の電子・陽電子の通過位置を検出するプリシャワー検出器についてその特性を検討した。

目次

図目次

1　序論

1.1　Belle Ⅱ 実験

Belle Ⅱ 実験とは、高エネルギー加速器研究機構の SuperKEKB 加速器と Belle Ⅱ 測定器を用いて B 中間子、τ レプトン、チャームハドロンの稀崩壊過程における CP 対称性の破れやレプトン普遍性の破れを測定し、標準理論予言値との差異という形で新しい物理法則の兆候を探索することをミッションとする国際共同実験である。

1.2　SuperKEKB 加速器

SuperKEKB 加速器は 7GeV の電子と 4GeV の陽電子を衝突させ、B 中間子・反 B 中間子対をはじめとする様々な粒子を高ルミノシティで生成する非対称エネルギー衝突型円形加速器である。KEKB 加速器からアップグレードされ、2020 年 6 月 15 日にルミノシティ $2.22 \times 10^{34}\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$ を達成した。これは現在のところ、全ての種類の衝突型加速器の中で世界最高ルミノシティである。

SuperKEKB 加速器全体の模式図を図 1 に示す。各リングの周長は約 3 km で地下約 11 m の深さのトンネル内に設置されている。電子は RF 電子銃で作られ、直線型加速器で加速された後に HER(High Energy Ring) に入射されて図中右回りに周回する。陽電子は 3.5GeV まで加速した電子を金属標的に当てて作り出し、陽電子ダンピングリングでビームのエミッタンスを下げた後に LER(Low Energy Ring) に入射されて図中左回りに周回する。各リングで加速された電子と陽電子が交差する衝突点を囲うように Belle Ⅱ 測定器が設置されている。

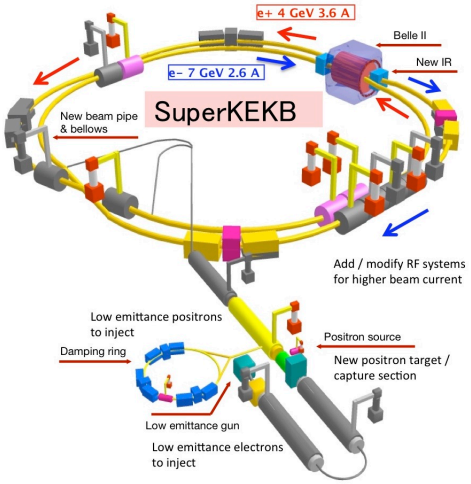


図 1　SuperKEKB 加速器全体の模式図

1.3 Belle Ⅱ 測定器

Belle Ⅱ 検出器は、衝突点まで電子・陽電子ビームを導く真空系の一部をなすビームパイプを中心として、異なる 7 種類の検出器を組み合わせた構造を持つ。検出器は内側からまず、ピクセル型検出器とシリコンバーテックス検出器が置かれ、これらが粒子の崩壊点を再構成するバーテックス検出器を構成するとともに、その外側の中央飛跡検出器とともに荷電粒子の飛跡検出系を形成する。続いてエアロゲルリングイメージチェレンコフ検出器、TOP カウンター、電磁カロリメーター、ミュー粒子・中性 K 中間子検出器である。図 2 で示す電磁カロリメーターに関して更に述べる。

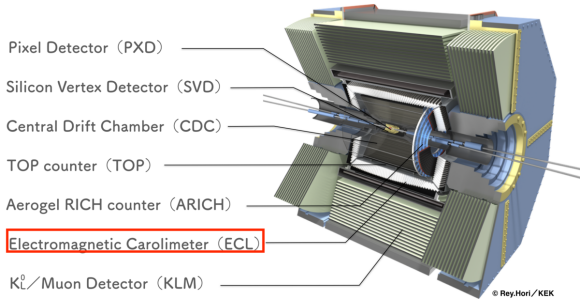


図 2　Belle Ⅱ 測定器全体の模式図

1.4 電磁カロリメーター

Belle Ⅱ 実験で用いる電磁カロリメーターはシンチレータとして CsI(Tl) 結晶を使用し、そこで粒子が起こしたエネルギー損失によるシンチレーション光を読み出す。このシンチレータに光検出器として PIN-PD を組み合わせた検出器 8736 本から構成される。高エネルギーの電子や光子が物質中に入射すると電磁シャワーを形成し、エネルギーを失う。検出器有感部となる CsI(Tl) 結晶は 30cm の長さがあり、入射粒子のほとんどのエネルギーを内部で失わせることができるため、エネルギー損失を最終的に電気信号パルスに変換し、その信号電荷の大きさにより入射粒子のエネルギーを測定する。

1.4.1 CsI(Tl) シンチレータ

シンチレータとは、荷電粒子が入射してエネルギー損失を起こすと、それに起因してシンチレーション光と呼ばれる発光をする物質である。シンチレータには大別するとポリスチレンを全体とした有機シンチレータと無機シンチレータがある。Belle Ⅱ 測定器の電磁カロリメーターに使用されている CsI(Tl) は無機結晶シンチレータである。無機結晶シンチレータは発光量が大きいためエネルギー分解能が良く、密度、原子番号が大きいため γ 線の阻止能が高いといった特徴を持つ。そのために γ 線の検出に適している。CsI(Tl) は密度 4.53 g/cm^3、発光量が約 50,000 光子 /MeV

と豊富である。しかしながら発光の減衰時間が約 1μs と長いため、高ルミノシティ環境ではバックグラウンドによるパイルアップの効果によりエネルギー分解能が制限される問題が発生する。

1.4.2 光検出器 (PIN-PD)

光検出器はシンチレータで発生した光を電気信号に変換し読み出す検出器である。中でもフォトダイオードは PN 接合した半導体に電極をつけた構造を持つ。電流が流れない向きに電圧をかけると N 型半導体の電子が一部 P 型半導体に移動し、正孔と結合して空乏層ができる。これにより N 型半導体は＋に、P 型半導体ではーに帯電し、内部に電界が発生する。空乏層で光子が光電効果を起こし、電子・正孔対を作ると、内部の電界により電子は N 型半導体側の電極へ、正孔は P 型半導体側の電極へ移動することにより電位変化が発生して電気信号が形成される。これがフォトダイオードの原理である。

PIN 型フォトダイオードは P 型半導体と N 型半導体の間に真性半導体の I 層を挟んだ構造を持つ。この I 層が主として空乏化し、光検出器として動作するための印加電圧の必要値を低くできる。

1.5 プリシャワー検出器の動機

実験中の検出器には、電磁カロリメーターに使用されている電子・陽電子衝突による物理現象に起因する γ 線だけでなく、ビーム蓄積に伴って生じるビームバックグラウンドの γ 線も達する。CsI(Tl) 結晶は記述したように発光減衰時間が長いため、バックグラウンドのパイルアップが雑音レベルの上昇をもたらすことが問題となる。既存のカロリメーターの前に時間分解能や位置分解能に優れた検出器を置き、そこでエネルギーの低い γ 線がとらえられるようにすればバックグラウンドの効果を軽減することができる。この検出器をプリシャワー検出器と呼ぶ。

BelleⅡ実験ではバックグラウンド γ 線が高輝度で発生するホットスポットが、ビーム衝突点から約 1cm 離れた位置にあることが判明しており、プリシャワー検出器に到来方向に対する感度を持たせることができれば、衝突点から来る γ 線とバックグラウンドの γ 線を識別することができると考えられる。

1.6 電磁シャワーによるエネルギーの測定

γ 線は物質中に入射すると光電効果、コンプトン散乱、電子・陽電子対生成という相互作用をする。BelleⅡ実験のような素粒子実験で検出すべき γ 線のエネルギーは数十 MeV 以上であるので、物質に入射後に最初に起こす相互作用は電子・陽電子対生成が主である。

電子・陽電子対生成とは入射光子のエネルギーが 1.022MeV（電子、陽電子の静止質量 0.511MeV の 2 倍のエネルギー）以上の場合に、物質の原子核近傍で電子・陽電子を生成する反応のことである。入射光子は全エネルギーを失って消滅し、電子と陽電子に全てのエネルギーを与える。また、生成された電子・陽電子が十分なエネルギーを持つ場合は物質中で更に制動放射を起こす。

制動放射とは、荷電粒子が物質中のクーロン場で減速し失ったエネルギーを γ 線として放出する現象であり、質量の小さい電子や陽電子で顕著に現れる。電子のエネルギーが数十 MeV 以上の場合には荷電粒子に共通に発生する。物質を構成する分子、原子をイオン化または励起することによる電離損失を制動放射によるエネルギー損失が上回る。制動放射で生じた γ 線のエネルギーが十分に高い場合は更に電子・陽電子対生成を起こす。

このように制動放射と電子・陽電子対生成が連鎖的に反応することを電磁シャワーという。シンチレータ内で電磁シャワーが形成されると電子や陽電子のエネルギー損失をシンチレーション光に変換し読み出すことでエネルギーを測定する。

2 Geant4 によるシミュレーション

2.1 Geant4 ソフトウェア

Geant4 は物質中を通過する粒子の相互作用をシミュレーションするツールキットである。高エネルギー物理学や医療、宇宙科学などの分野で利用されている。使用言語は C++ である。Geant4 は役割に応じて様々なクラスとして書かれており、クラスをまとめたパッケージのことをツールキットと呼ぶ。Geant4 は複数の例題プログラムを持っており、ユーザーの目的に応じて例題プログラムを改変しシミュレーションを実行することができる。

2.2 環境

使用した Geant4 ソフトウェアのバージョンは 10.6 であり、64bit アーキテクチャーの CentOS7 Linux 向けバイナリパッケージと例題プログラムを含むソースコードをインストールしたものである。シミュレーションにより作成された root ファイルの解析は個人所有の Macintosh に転送して行った。

2.3 シミュレーションの設定

Geant4 ソースコードパッケージ中に収録されているサンプリングカロリメーターの例題である exampleB4a（以下 B4a）を改変して使用した。

プリシャワー検出器の構成はアクティブ吸収体となる BGO 結晶シンチレータとシャワー中の電子や陽電子の通過位置を検出する Si 検出器を交互に配置したサンドイッチ型である。BGO 結晶シンチレータと Si 検出器を組み合わせたものを 1layer とし、3layer 構造とした。1layer の縦と横の長さは既存の電磁カロリメーターの CsI(Tl) 結晶の断面と同じ 5.5cm とした。BGO の厚みは 1 放射長である 11.2mm、Si 検出器の厚みは一般的な厚みである 0.3mm とした。ここで放射長とは、高エネルギーの電子が物質中で制動放射により、そのエネルギーが平均で $1/e$ に減少する長さである。また、Si 検出器は碁盤の目状のピクセル構造を持つものとした。

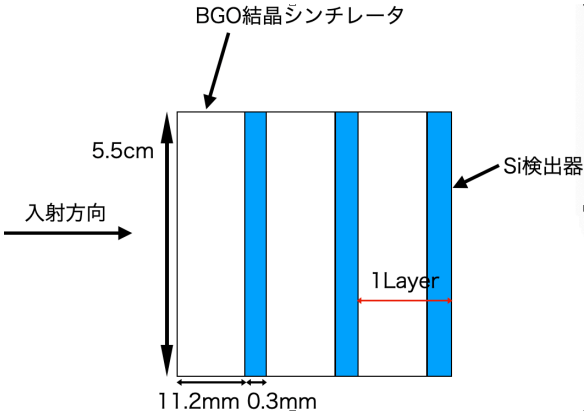


図 3　プリシャワー検出器の装置ジオメトリ

src ディレクトリに置かれているソースコードのうち、`B4DetectorConstruction.cc`、`B4RunAction.cc`、`B4aEventAction.cc`、`B4aSteppingAction.cc`、及び適宜対応するヘッダーファイルを改変した。

シミュレーションを実行すると root ファイルが作成される。root ファイルを解析するマクロファイル作成し、解析を行った。解析用マクロファイルについては次の章で述べる。

2.3.1　BGO 結晶シンチレータ

BGO（ゲルマニウム酸ビスマス）は酸化物無機結晶シンチレータの一種である。BGO シンチレータは粒子が入射するとエネルギー損失が起こりピーク波長 480nm の緑色蛍光の光を発する。発光減衰時間は 300ns、密度は 7.13g/cm^3、放射長は 11.2mm である。プリシャワー検出器 1layer の BGO 結晶シンチレータは縦 5.5cm、横 5.5cm、厚み 11.2mm の直方体の構造を持つ。プリシャワー検出器のアクティブ吸収体をなす。

2.3.2　Si 検出器

プリシャワー検出器 1layer の Si 検出器は縦 5.5cm、横 5.5cm、厚み 0.3mm の平板の形状をもち、BGO 結晶シンチレータの背後に配置する。Si 検出器の密度は 2.33g/cm^3 である。

Si 検出器をピクセル構造を持つものとすることで粒子の通過位置を検出する。本研究のシミュレーションでは各 Si 検出器をピクセルに分割し、ピクセル毎のエネルギー損失の値を root の Ntuple で該当する変数に保存するようにプログラムを改変した。

2.3.3　シミュレーションコードの改変

オリジナルの Geant4 バージョン 10.6 のソースコードパッケージにサンプリングカロリメータの例題として収録されている exampleB4 はシャワーを生成・発達させる吸収体部分を鉛、電子または陽電子のエネルギー損失を検出する有感部分としてのギャップを液体アルゴンとした構成に

なっている。吸収体部分でのエネルギー損失も測定可能になるよう、密度の大きい結晶シンチレータの典型例として BGO 結晶に置き換えるとともに、ギャップ部分をシリコン半導体検出器とする改変は、今野の卒業研究 [6] で既に実施されている。

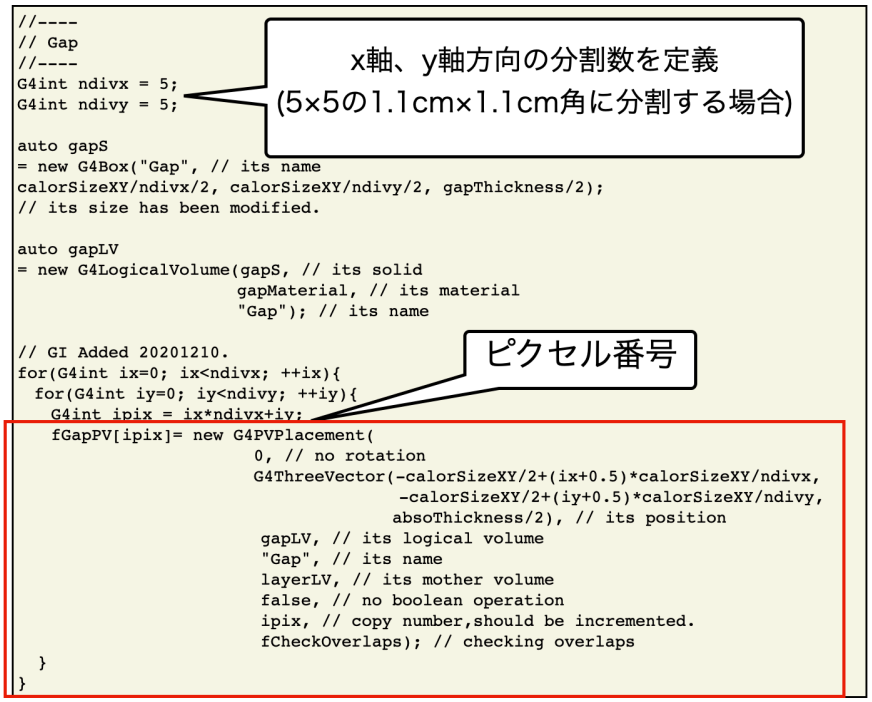
BGO、シリコンとも、Geant4 パッケージに付属の NIST マテリアルデータベース中に記載されているので、そのレコードを `G4NistManager` クラスの `FindOrBuildMaterial` 関数で呼び出す手法をとっている。最大の `Mother Volume` つまり検出器ジオメトリを入れ子構造で実装する際の、全てを含む最も外側のボリュームである `World` は、地表の大気で満たしている。

`B4DetectorConstruction.cc` で Si 検出器は `Gap` 部分に当たり、x 軸、y 軸方向の分割数を決定する変数 `ndivx`、`ndivy` を宣言し、各 Si ピクセルの情報を一次元配列で扱うため通し番号にあたるピクセル番号である `ipix` は列の番号 `ix` と行の番号 `iy` を用いて表した。そして、ピクセル毎にジオメトリを宣言して扱うために `Gap` 部分の `PV`(PhysicalVolume) である `fGapPV` を配列とした。`PV` とは Physical Volume のことを指し、設定した物質がシミュレーション上で設置される。

`B4RunAction.cc` で各ピクセルのエネルギー損失等の情報を与える Ntuple を作成し、`B4EventAction.cc` で各 Ntuple にピクセル毎のエネルギー損失を Fill する。

`B4aSteppingAction.cc` ではピクセル中で生じた step のエネルギー損失が 50keV 以上の場合にレイヤー番号とピクセル番号とともにエネルギー損失の値を `AddGap` 関数に引数として渡す。ここで 50keV の制限をつけたのは厚さ 300μm の Si 検出器を Minimum Ionizing Particle(MIP) が通過する際のエネルギー損失が約 0.1MeV であることから、その 50% である 50keV を考慮したからである。[*1] `AddGap` 関数とは `B4aEventAction.hh` 中で定義されている関数であり、`Gap` 部分で生じたエネルギー損失を step 毎、volume 毎、つまり Si 検出器ではピクセル毎に足し合わせる関数である。step とは、1 つの event 中での 1 つの相互作用のことを言う。

[*1] 厳密には、エネルギー損失に閾値を設けるにはピクセル毎のエネルギー損失の積算値が確定した後に行うべきであるので、この制御はシミュレーション中で行わずに、解析マクロの中で行うのが正しい。




```
//----
// Gap
//----
G4int ndivx = 5;
G4int ndivy = 5;

auto gapS
= new G4Box("Gap", // its name
calorSizeXY/ndivx/2, calorSizeXY/ndivy/2, gapThickness/2);
// its size has been modified.

auto gapLV
= new G4LogicalVolume(gapS, // its solid
                      gapMaterial, // its material
                      "Gap"); // its name

// GI Added 20201210.
for(G4int ix=0; ix<ndivx; ++ix){
  for(G4int iy=0; iy<ndivy; ++iy){
    G4int ipix = ix*ndivx+iy;
    fGapPV[ipix]= new G4PVPlacement(
                        0, // no rotation
                        G4ThreeVector(-calorSizeXY/2+(ix+0.5)*calorSizeXY/ndivx,
                                      -calorSizeXY/2+(iy+0.5)*calorSizeXY/ndivy,
                                     absoThickness/2), // its position
                         gapLV, // its logical volume
                         "Gap", // its name
                         layerLV, // its mother volume
                         false, // no boolean operation
                         ipix, // copy number,should be incremented.
                         fCheckOverlaps); // checking overlaps
  }
}
```

図 4　B4DetectorConstruction.cc の抜粋。Si 検出器部分に当たる Gap 部分のオリジナルプログラムからの変更点を示す。

```
if ( volume == fDetConstruction->GetAbsorberPV() ) {
// KM Added 20200216.
G4int lyrid
= step->GetPreStepPoint()->GetTouchableHandle()->GetReplicaNumber(1);
fEventAction->AddAbs(edep,stepLength,lyrid);
// fEventAction->AddAbs(edep,stepLength); // original
}
for(G4int ipix=0; ipix<3025 ; ++ipix){
 if ( volume == fDetConstruction->GetGapPV(ipix) ) {
  if( edep > 0.05 ){ //GI Added 20210202.
  // get layer number (replica Number)
  // KM Added 20200216.
  G4int lyrid
   = step->GetPreStepPoint()->GetTouchableHandle()->GetReplicaNumber(1);
  // GI Added 20201030.
  fEventAction->AddGap(edep,stepLength,lyrid,ipix); // GI Added 20201030.
  // fEventAction->AddGap(edep,stepLength,lyrid); // KM Added 20200216.
  // fEventAction->AddGap(edep,stepLength); // original.
    }
  }
}
```

図 5　B4aSteppingAction.cc の抜粋。Si 検出器部分に当たる Gap 部分のオリジナルプログラムからの変更点を示す。

3　データの処理と解析

3.1　解析マクロファイル

Geant4 シミュレーションを実行すると各 layer の BGO アクティブ吸収層や Si ピクセル毎のエネルギー損失の値を書き込んだ root ファイルが作成される。この root ファイルを読み込み、γ 線の到来方向を再構成する解析を行うマクロファイルを 2 種類作成した。root マクロは C++ 言語で作成する仕様であり、数値的に求めるものと、最小二乗法を適用して解析的に最小の条件を解いた結果の式を実装したものの 2 種類である。以下でこの 2 つのプログラムについて述べる。

3.2　解析方法

解析マクロの内容は定数になる情報の宣言や初期化と、event 毎に変化する情報を取り扱う部分に大別できる。前者は以下の 1. であり、後者は 2. は 2 種類の解析マクロに共通であるが、3. の手法が異なる。

1. 各ピクセル毎の中心位置の座標を宣言する。
   ピクセルがエネルギー損失を検出したとき全ての event に対して共通である。ここで定義した位置を電磁シャワー中の電子または陽電子の通過位置とする。
2. Si 検出器の各 layer 毎に電磁シャワーの中心位置を決定する。
   Si ピクセルのエネルギー損失で重みをつけた重心を求め、その layer の電磁シャワーの中心位置とした。
3. 3layer とも 2. で電磁シャワーの中心位置を得られた場合に、得た点を結ぶことにより γ 線の到来方向を決定する。
   γ 線の到来方向を決定する際に、数値計算による方法と最小二乗法による方法で解析を行う 2 種類のプログラムを作成した。

3.2.1　数値的に解析する方法

前述の段階で得た点からの距離の二乗和が最小となる直線を決定し、これを γ 線の到来方向とする。

γ 線の入射位置はシミュレーションの際に $(x, y, z) = (0, 0, 0)$ と設定しており、これが既知のものとする仮定を置いて図 6 に示す。入射位置を $(x, y, z) = (0, 0, 0)$ に固定した。この場合、再構成する γ 線の到来方向は極角 θ と方位角 ϕ の 2 変数で表せる。(0,0,0) に固定した入射位置から θ と ϕ で決まる方向に直線を伸ばし、各層で求めた電磁シャワーの中心位置からの距離 $d[0], d[1], d[2]$ の二乗和 χ^2 が最小となる条件を、$0\,\mathrm{rad}<\theta<\frac{\pi}{2}\mathrm{rad}$,$-\frac{\pi}{2}\mathrm{rad}<\phi<\frac{\pi}{2}\mathrm{rad}$ の範囲で $0.001\,\mathrm{rad}$ ずつ変化させて、$\theta - \phi$ 平面状にグリッドを設定し、その各格子点状での χ^2 を計算し、最小のものをとる。本研究では、γ 線が $(x, y, z) = (0, 0, 1)$ の方向に入射する条件でシミュレーションを行ったので ϕ

の範囲を結果を簡潔にするために上記の範囲として解析を行った。

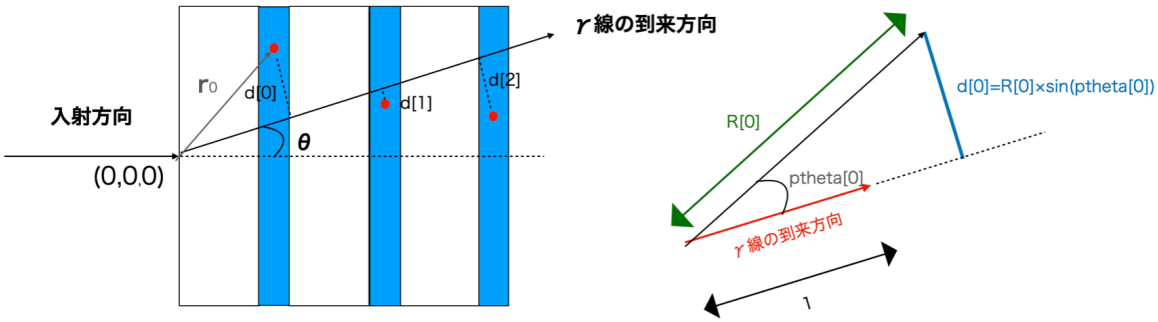


図 6　数値的に解析する場合。(左)γ 線の到来方向を示す直線を延長し、各 layer の電磁シャワーの中心位置からの距離 $d[0], d[1], d[2]$。(右)γ 線の到来方向を示す直線と電磁シャワーの中心位置を示す点との距離の求め方について第 0layer における $d[0]$ を例にとって示した。

θ と ϕ を用いて、再構成する γ 線の方向を示す単位ベクトルは $\boldsymbol{V}_\gamma$=(sinθcosϕ,sinθsinϕ,cosθ) と表される。$(x, y, z) = (0, 0, 0)$ の点から i 番目 layer で得た電磁シャワーの中心位置までのベクトルを $\boldsymbol{r}_i$ とし、このベクトルと到来方向を示すベクトルのなす角を $ptheta[i]$ とする $(i = 0, 1, 2)$。$ptheta[i]$ は、

$$ptheta[i] = \arccos\left(\frac{\boldsymbol{r}_i}{R[i]} \cdot \boldsymbol{V}_\gamma\right)(i = 0, 1, 2)$$

により求められる。ここで、$R[i]$ は $\boldsymbol{r}_i$ の大きさである。シャワーの中心位置と γ 線の到来方向を示す直線との距離 $d[i]$ は

$$d[i] = R[i]\sin(ptheta[i])(i = 0, 1, 2)$$

より求められる。

$d[i]$ の二乗和が χ^2 であり、これが最小となる時の θ、ϕ を θ は 0rad から $\frac{\pi}{2}$rad の範囲で、ϕ は-$\frac{\pi}{2}$rad から $\frac{\pi}{2}$rad の範囲でそれぞれ 0.001 rad ずつ増加させるループを回して決定した。

3.2.2　解析的な最小二乗法

前小節で述べた方法では必要な数値計算の量が多く、入射位置を固定せずにフィットで得る拡張も現実的でないので χ^2 最小の条件を解析的に求めた結果の式を使用することにした。

γ 線の方向を表すベクトルを $(e_1, e_2, 1)$ で表す。γ 線の入射点の座標を $(x_0, y_0, 0)$ とし、ここを起点として γ 線の方向を表す。ベクトルを伸ばし、各 layer で求めた電磁シャワーの中心位置から Si 検出器の平面状での距離 $d[0], d[1], d[2]$ の二乗和を χ^2 とし、これが最小となる条件を計算する。これは x_0, y_0, e_1, e_2 の 4 パラメーターフィットである。

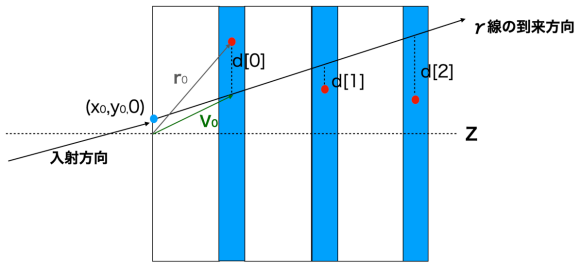


図 7　解析的に最小二乗法を適用する場合。γ 線の方向を示すのベクトル $(e_1, e_2, 1)$ を延長し、各 layer の Si 検出器の平面状で再構成した電磁シャワーの中心位置からの距離を $d[0], d[1], d[2]$ とする。

γ 線の方向を示すベクトルを $\boldsymbol{V}_\gamma = (e_1, e_2, 1)$、$\gamma$ 線の入射位置の座標を $(x_0,y_0,0)$ とする。i 番目の layer の Si 検出器で得られた電磁シャワーの中心位置を示すベクトルを $\boldsymbol{r}_i = (x_{i+1}, y_{i+1}, z_{i+1})$ とする (i=0,1,2)。また、$\boldsymbol{V}_\gamma$ の延長した直線上で $\boldsymbol{r}_i$ と z 座標が等しくなる位置を示すベクトル $\boldsymbol{V}_i = (z_{i+1}e_1 + x_0, z_{i+1}e_2 + y_0, z_{i+1})$ とする $(i = 0, 1, 2)$。この時、直線とシャワーの中心位置の距離 d は以下のように表される。

$$d[i] = |\boldsymbol{r}_i - \boldsymbol{V}_i| \qquad (i = 0, 1, 2)$$

$$Q = \sum_{i=0}^{2} d[i]^2 = \sum_{i=0}^{2} (x_{i+1} - x_0 - z_{i+1}e_1)^2 + (y_{i+1} - y_0 - z_{i+1}e_2)^2$$

と表すことにすると、Q が最小かつ極小になるときのパラメーター x_0, y_0, e_1, e_2 は以下の式を満たす。

$$\frac{\partial Q}{\partial x_0} = 6x_0 - 2\sum x_i + 2e_1 \sum z_i = 0 \tag{1}$$

$$\frac{\partial Q}{\partial y_0} = 6y_0 - 2\sum y_i + 2e_2 \sum z_i = 0 \tag{2}$$

$$\frac{\partial Q}{\partial e_1} = 2e_1 \sum z_i^2 - 2\sum x_i z_i + 2x_0 \sum z_i = 0 \tag{3}$$

$$\frac{\partial Q}{\partial e_2} = 2e_2 \sum z_i^2 - 2\sum y_i z_i + 2y_0 \sum z_i = 0 \tag{4}$$

この連立 4 元一次方程式を解くにあたり、簡単のために $A = \sum x_i, B = \sum y_i, C = \sum z_i, D = \sum x_i z_i, E = \sum y_i z_i, F = \sum z_i^2$ と表記する。(1)-(4) 式よりそれぞれのパラメーターは以下の式により求められる。

$$x_0 = \frac{AF - CD}{3F - C^2} \tag{5}$$

$$y_0 = \frac{BF - CE}{3F - C^2} \tag{6}$$

$$e_1 = \frac{3D - AC}{3F - C^2} \tag{7}$$

$$e_2 = \frac{3E - BC}{3F - C^2} \tag{8}$$

ただし、ここで得られた $\boldsymbol{V}_\gamma$ の大きさは 1 ではないので単位ベクトルにするには $\sqrt{e_1^2+e_2^2+1}$ で割ればよい。

4　結果

エネルギー 1GeV の γ 線を $(x, y, z) = (0, 0, 0)$ の位置に 10,000 イベント垂直入射させたシミュレーションデータを前章に記述した手法で解析した結果について述べる。

4.1　エネルギー損失を検出したピクセル数

エネルギー損失を検出したピクセル数の分布をピクセルのサイズが異なる場合について示す。

図 8 に示す 1.1cm 角ピクセルの場合は、エネルギー損失を検出したピクセル数が 0 の割合は layer0 は約 48%、layer1 は約 23%、layer2 は約 11% である。これは γ 線がアクティブ吸収体の BGO シンチレーター中で相互作用する確率によって決まっているため、ピクセルのサイズとは無関係である。エネルギー損失を検出したピクセル数が 1 の割合は layer0 は約 49%、layer1 は約 62%、layer2 は約 65% である。エネルギー損失を検出できなかった場合を除くと、エネルギー損失を検出できたピクセル数が 1 の場合が占める割合が大きい。信号が出たピクセルについてエネルギー損失で重みをつけた重心位置により電磁シャワーの中心位置を決める方法を取ることを想定すると、特にシャワーが発達を始めた後の layer1 や layer2 で複数のピクセルがエネルギー損失を検出していることが望ましい。したがって 1.1cm 角はピクセルの大きさが最適値よりも大きいことを示唆している。

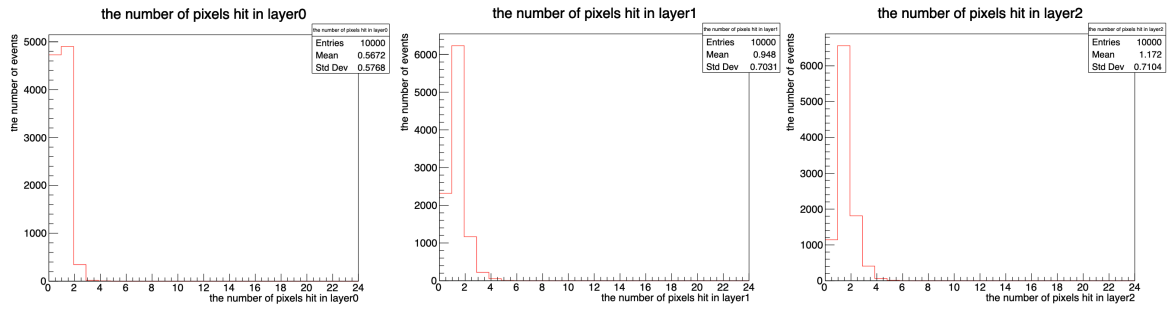


図 8　1.1cm 角ピクセルの場合にエネルギー損失を検出したピクセル数の分布。左から順に layer0、layer1、layer2。

図 9 に示す 5mm 角ピクセルの場合はエネルギー損失を検出したピクセル数が 0 の割合は変わらないが、ピクセル数が 1 の割合は layer0 は約 45%、layer1 は約 45%、layer2 は約 37% となり、1.1cm 角の時よりも複数枚検出される割合が増加している。

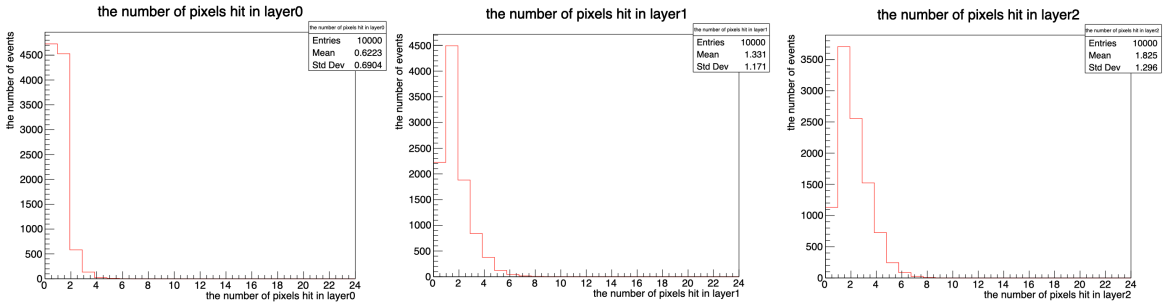


図 9　5mm 角ピクセルの場合にエネルギー損失を検出したピクセル数の分布。左から順に layer0、layer1、layer2。

図 10 に示す 2.5mm 角ピクセルの場合はエネルギー損失を検出したピクセル数が 1 の割合が layer0 が約 10%、layer1 が約 9%、layer2 は約 8% と 5mm 角のピクセルの分布と比較するとかなり小さくなる一方、ピクセル数が 2 の頻度が顕著に増加している。これは Si 検出器を x 方向および y 方向に偶数枚に分割したために、プリシャワー検出器の中心への γ 線の入射位置がピクセルの境界に当たるために起こったと考えられた。図 11 はこれを確かめるためにピクセル中央に γ 線が入射するように入射位置を 1.25mm ずつ縦、横に変化させた場合である。γ 線の入射位置が検出器中央のピクセルの中心に当たるプリシャワーピクセルや検出器 1.1cm 角のピクセルや 5mm 角のピクセルのときと同様、信号が出たピクセルが存在する場合はピクセルの数が 1 つの場合が最も多い分布をしている。これより、図 10 に示した分布は入射位置がピクセルの境界位置に来ていたことが原因だと言える。また、図 11 より 2.5mm 角にするとエネルギー損失を検出するピクセル数が全体に増加している。

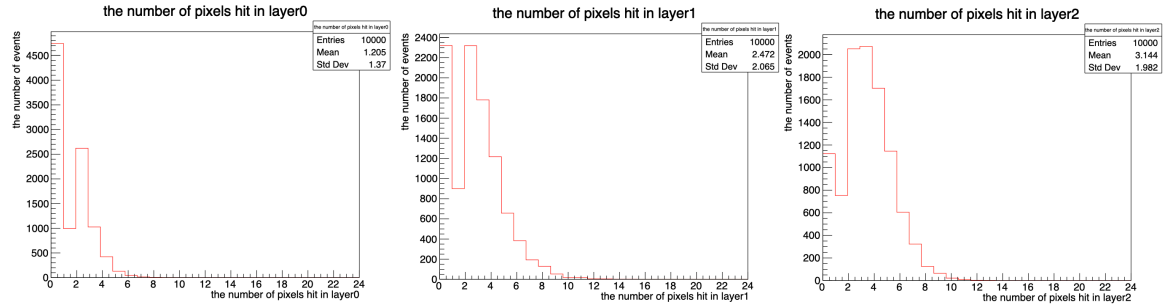


図 10　2.5mm 角ピクセルの場合にエネルギー損失を検出したピクセル数の分布。左から順に layer0、layer1、layer2。

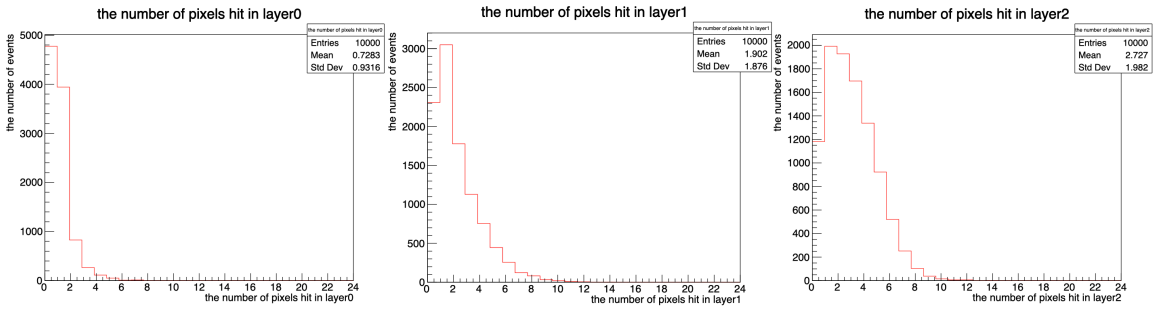


図 11　2.5mm 角ピクセルの場合に入射位置を 1.25mm ずつ縦・横にずらした場合にエネルギー損失を検出したピクセル数の分布。左から順に layer0、layer1、layer2。

1mm 角までピクセルを細分化した場合は、エネルギー損失を検出したピクセル数が 1 の割合は layer0 が約 30%、layer1 が約 18%、layer2 が約 12% である。これは入射した光子のうち、layer1 で 59%,layer2 で 77% が複数ピクセルに信号を作ることを意味する。

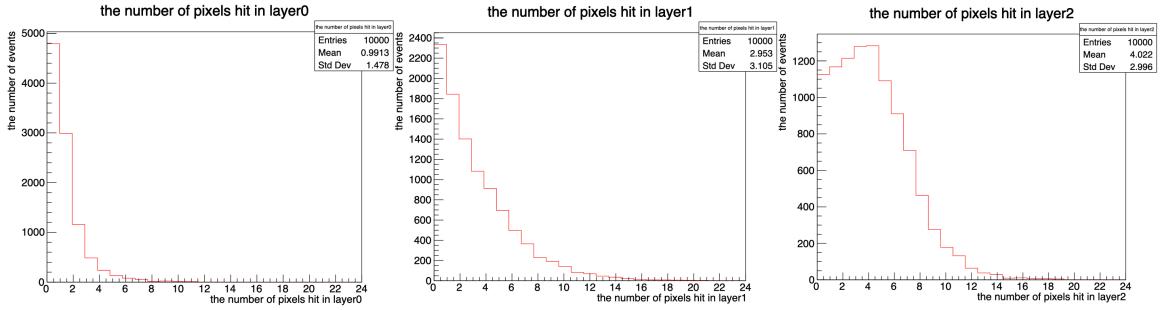


図 12　1mm 角ピクセルの場合にエネルギー損失を検出したピクセル数の分布。左から順に layer0、layer1、layer2。

以上のことより 1mm 角ピクセルの場合は複数のピクセルから信号を得て、高輝度で電磁シャワーの中心位置を決定することができると期待できる。そこで、1mm 角ピクセルの場合に入射位置 (x_0, y_0) と γ 線の方向を再構成する分解能を調べた。

4.2　1mm 角ピクセルの場合の性能評価

4.2.1　電磁シャワーの中心位置

シミュレーションではプリシャワー検出器の中央に垂直入射させたため、x と y 方向は対称になる。各 layer で再構成した電磁シャワーの中心位置の x 座標の分布を図 13 に示す。分布の標準偏差は 1.6mm であった。

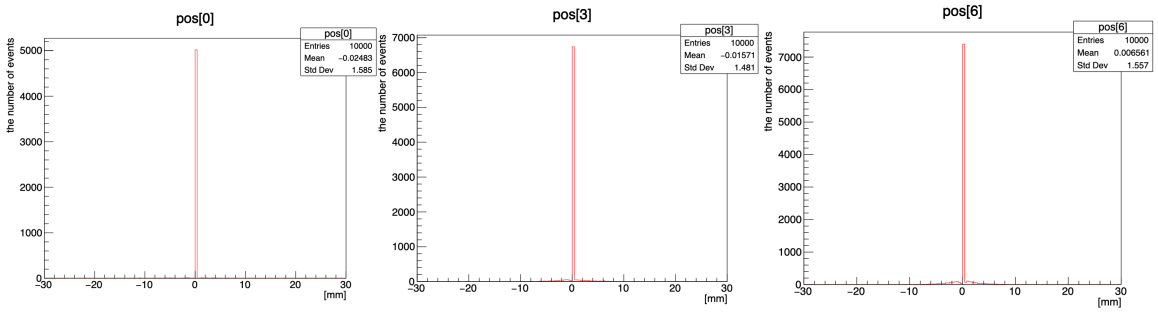


図 13　1mm 角ピクセルの場合の電磁シャワーの中心位置の x 座標の分布。左から順に layer0,layer1,layer2。

4.2.2　γ 線の方向の再構成

数値的および解析的な式による 2 つの方法で γ 線の方向を求めた。両者の手法に共通に実行したのは $(x_0, y_0) = (0, 0)$ に固定した場合であり、その条件下での極角 θ の分布を図 14 示す。2 つの分布はほぼ同じであり、分布の標準偏差はいずれの手法を用いた場合も 0.03 rad であった。さらに、入射位置 (x_0, y_0) もフィットで求めるパラメーターとした 4 パラメーターのフィットを実行して得た結果を図 15 および図 16 に示す。

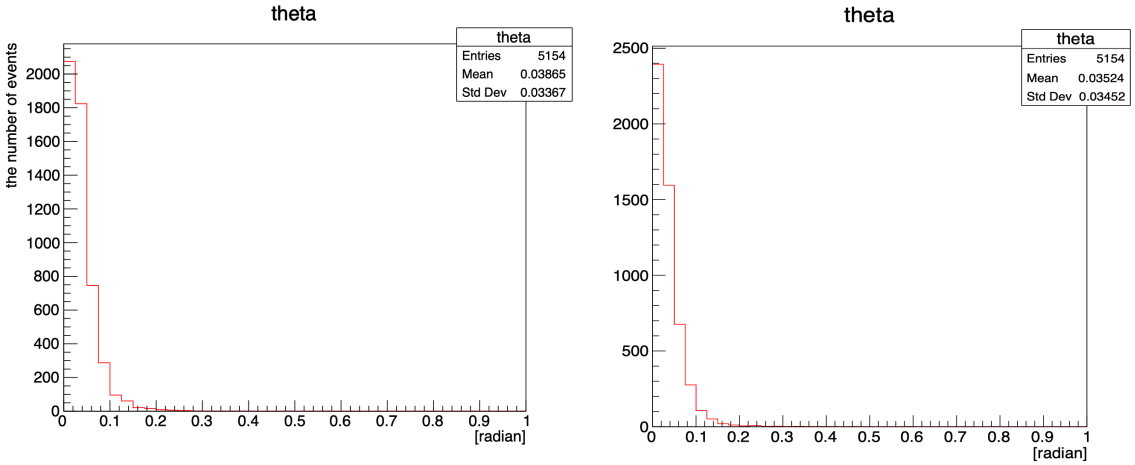


図 14　1mm 角ピクセルの場合の再構成した γ 線の到来方向を示す極角 θ。最小二乗法により求めた分布（左）、数値計算により求めた分布（右）

再構成した γ 線の方向を示すベクトル $\boldsymbol{V}_\gamma = (e_1, e_2, e_3)$ として、この x 成分および y 成分 e_1 と e_2 の分布を図 15 に示す。分布の標準偏差により xz 平面内または yz 平面内に投影した方向の分解能を評価すると、0.08 rad であった。

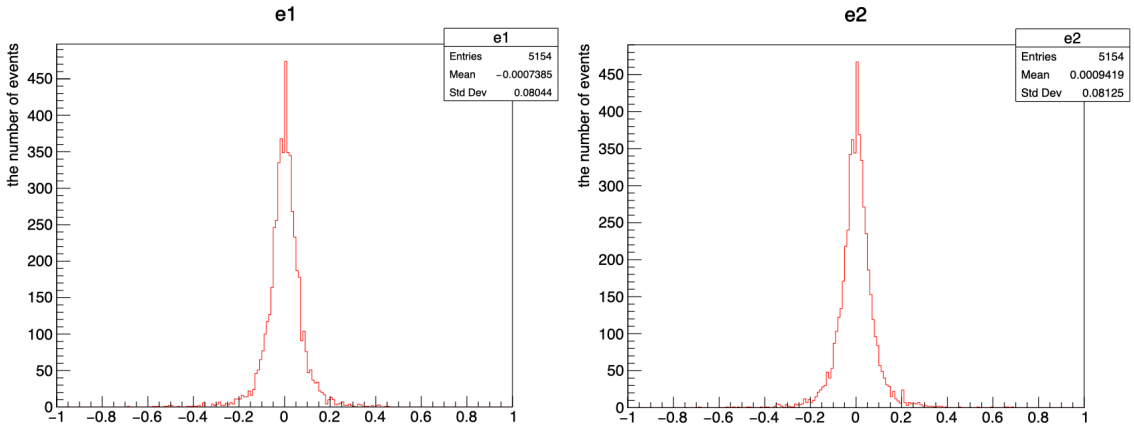


図 15　1mm 角ピクセルの場合の再構成した γ 線の到来方向を示すベクトル。x 方向の要素 e_1（左）、y 方向の要素 e_2（右）

また、再構成した γ 線の入射位置 x 成分 x_0 と y 成分の y_0 の分布を図 16 に示す。分布の標準偏差により位置分解能を評価したところ、1.9mm であった。

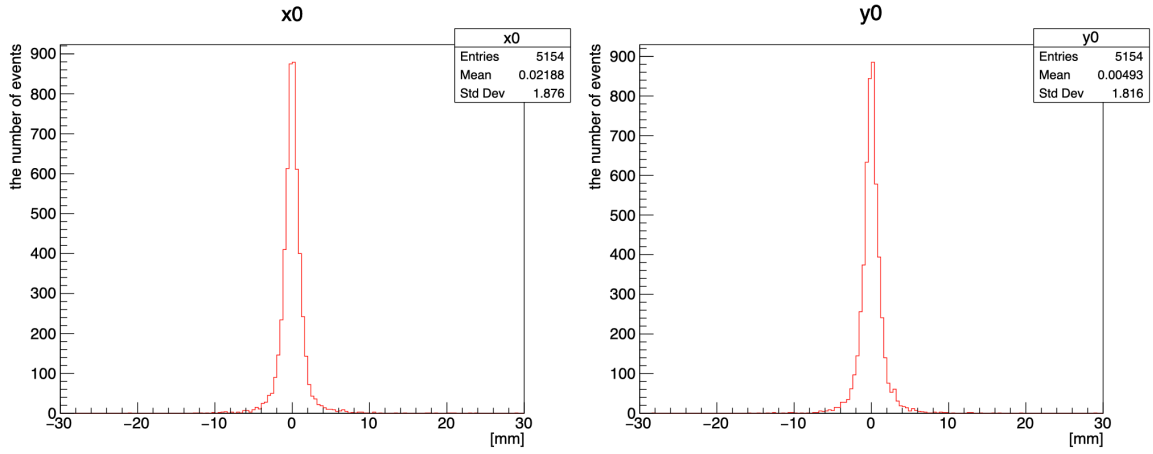


図 16　1mm 角ピクセルの場合の再構成した γ 線の入射位置。x 方向の要素 x_0（左）、y 方向の要素 y_0（右）

5　まとめ

BelleⅡ実験において CsI(Tl) シンチレータを用いた電磁カロリメーターは高ルミノシティ環境下ではビームバックグラウンドに起因するパイルアップによりエネルギー分解能が低下する。この問題への対応をする選択肢の一つとしてプリシャワー検出器を導入することを検討した。今年は特に、γ 線の到来方向を再構成するために Si 検出器をピクセル構造を持つよう設定し、シミュレーションと解析を行った。Si 検出器は 1.1cm 角に分割したものから、最小で 1mm 角に分割した場合までシミュレーションを行った。Si 検出器を 1mm 角に分割した場合にプリシャワー検出器中心にエネルギー 1GeV の γ 線を垂直入射させたとき、Si 検出器の面上における電磁シャワーの中心位置の分解能は約 1.5mm であり、最小二乗法を用いて入射位置 (x_0, y_0) と方向を示す単位ベクトルの x 成分と y 成分の 4 パラメーターフィットにより入射 γ 線の再構成を行ったところ、入射位置の分解能は 1.9mm、方向の分解能は 0.08 rad と得た。今後は γ 線の入射方向、入射位置、エネルギーが変化した場合のシミュレーションと解析を行ってプリシャワー検出器の特性について包括的な研究を行うべきである。

謝辞

本研究において、ご指導いただいた宮林謙吉教授に心より感謝申し上げます。研究の方針や Geant4 の使い方、グラフの見方など拙い私に丁寧にサポートしてくださいました。また、最後まで導いてくださったおかげで、１年間本研究に取り組み、本論文を書き終えることができました。

ミーティングの時などに様々なコメントやアドバイスをいただいた林井久樹教授、下村真弥助教、蜂谷崇助教にも感謝申し上げます。そして、研究室の先輩方、同回生の皆様にも感謝申し上げます。新型コロナウイルスの影響もあり、一緒に過ごす時間は多くはありませんでしたが、真剣に研究に取り組まれる姿に刺激を受けると共に楽しく研究室で過ごすことができました。

最後に、４年間大学に通わせ、時に励まし支えてくれた家族にも心より感謝申し上げます。

短い期間ではありましたが、プログラムや電磁カロリメータなどの検出器の知識、グラフの見方など多くのことを学び、経験させていただきました。全ては本研究に携わってくださった方々のおかげです。皆様への感謝の意を表して、謝辞と致します。

参考文献

[1] KEK　ニュースルーム　”SuperKEKB 加速器が世界最高ルミノシティ（衝突性能）を達成しました”　https://www.kek.jp/ja/newsroom/2020/06/26/1400/

[2] Belle Ⅱ Project ”SuperKEKB and Belle Ⅱ” https://www.belle2.org/project/super_kekb_and_belle_ii

[3] 株式会社リーティングエッジアルゴリズム ”CsI”　https://www.marutsu.co.jp/contents/shop/marutsu/datasheet/LEC3M101010.pdf

[4] 株式会社ハナムラオプティクス ”BGO 結晶” http://www.hanamuraoptics.com/scintillatorcrystal/scintillatorcrystals/bgo.htm

[5] 関口信忠, 阪井英次　”半導体検出器による放射線測定”　1966 年 7 月 26 日　https://www.jstage.jst.go.jp/article/radioisotopes1952/15/6/15_6_393/_pdf

[6] 今野つかさ　奈良女子大学理学部物理学コース高エネルギー加速器研究室　卒業論文 (2020 年)　https://webhepl.cc.nara-wu.ac.jp/old_HP/thesis/4kaisei/2019/20200507_konno_ug_thesis_final.pdf

付録

1. 装置ジオメトリを設定する B4DetectorConstruction.cc。Geant4 の例題プログラムである exampleB4a から改変したもの。

```
///  B4DetectorConstruction.cc
/// brief Implementation of the B4DetectorConstruction class
// KM modification starts 2020 Jan. 6th.
// GI modification starts 2020 Oct. 14th.
#include "B4DetectorConstruction.hh"

#include "G4Material.hh"
#include "G4NistManager.hh"

#include "G4Box.hh"
#include "G4LogicalVolume.hh"
#include "G4PVPlacement.hh"
#include "G4PVReplica.hh"
#include "G4GlobalMagFieldMessenger.hh"
#include "G4AutoDelete.hh"

#include "G4GeometryManager.hh"
#include "G4PhysicalVolumeStore.hh"
#include "G4LogicalVolumeStore.hh"
#include "G4SolidStore.hh"

#include "G4VisAttributes.hh"
#include "G4Colour.hh"
#include "G4PhysicalConstants.hh"
#include "G4SystemOfUnits.hh"

// Magnetic Field (check whether nullptr means 0)
G4ThreadLocal
G4GlobalMagFieldMessenger*
B4DetectorConstruction::fMagFieldMessenger = nullptr;


// Constructor of B4DetectorConstruction.
B4DetectorConstruction::B4DetectorConstruction()
 : G4VUserDetectorConstruction(),
   fAbsorberPV(nullptr),
   //fGapPV(nullptr),  // original.
   fCheckOverlaps(true)
{
  for(G4int i=0; i<3025 ; ++i){
    fGapPV[i]=nullptr;
  }
}

// Destructor.
B4DetectorConstruction::~B4DetectorConstruction()
{
}


// Construct() function returns G4PhysicalVolume*.
G4VPhysicalVolume* B4DetectorConstruction::Construct()
{
  // Define materials
```

```
  DefineMaterials();

  // Define volumes
  return DefineVolumes();
}

// Define materials.
void B4DetectorConstruction::DefineMaterials()
{
  // NIST is GEANT4 Material Database. Instance() func. is static.
  auto nistManager = G4NistManager::Instance();
  // Air, BGO and Si defined using NIST Manager.
  nistManager->FindOrBuildMaterial("G4_AIR");
  nistManager->FindOrBuildMaterial("G4_BGO");
  nistManager->FindOrBuildMaterial("G4_Si");
  // Liquid argon material (original example).
  // G4double a;  // mass of a mole;
  // G4double z;  // z=mean number of protons;
  // G4double density;
  // new G4Material("liquidArgon", z=18., a=39.95*g/mole,
density=1.390*g/cm3);
  // The argon by NIST Manager is a gas with a different density

  // Print materials
  G4cout << *(G4Material::GetMaterialTable()) << G4endl;
}

// Define Volumes.
G4VPhysicalVolume* B4DetectorConstruction::DefineVolumes()
{
  // Geometry parameters
  G4int nofLayers = 3; // Presampler.
  G4double absoThickness = 11.2*mm; // BGO 1X0.
  G4double gapThickness =  0.3*mm; // Si thickness.
  //G4double gapThickness =  10.*mm; // Si thickness.
  G4double calorSizeXY  = 5.5*cm; // Same cross section as Belle
CsI.

  auto layerThickness = absoThickness + gapThickness;
  auto calorThickness = nofLayers * layerThickness;
  auto worldSizeXY = 1.2 * calorSizeXY;
  auto worldSizeZ  = 1.2 * calorThickness;

  // Get materials, sould be consistent with DefineMaterials().
  auto defaultMaterial = G4Material::GetMaterial("G4_AIR");
  auto absorberMaterial = G4Material::GetMaterial("G4_BGO");
  auto gapMaterial = G4Material::GetMaterial("G4_Si");
  // Get materials (original example).
  // auto defaultMaterial = G4Material::GetMaterial("Galactic");
  // auto absorberMaterial = G4Material::GetMaterial("G4_Pb");
  // auto gapMaterial = G4Material::GetMaterial("liquidArgon");

  // When one of material definitions fails, should exit with a
message.
```

```
  if ( ! defaultMaterial || ! absorberMaterial || ! gapMaterial ) {
    G4ExceptionDescription msg;
    msg << "Cannot retrieve materials already defined.";
    G4Exception("B4DetectorConstruction::DefineVolumes()",
      "MyCode0001", FatalException, msg);
  }

  //------
  // World
  //------
  auto worldS
    = new G4Box("World",           // its name
                worldSizeXY/2, worldSizeXY/2, worldSizeZ/2); // its
size

  auto worldLV
    = new G4LogicalVolume(
                worldS,           // its solid
                defaultMaterial,  // its material
                "World");         // its name

  auto worldPV
    = new G4PVPlacement(
                0,                // no rotation
                G4ThreeVector(),  // at (0,0,0)
                worldLV,          // its logical volume
                "World",          // its name
                0,                // its mother  volume
                false,            // no boolean operation
                0,                // copy number
                fCheckOverlaps);  // checking overlaps

  //------------
  // Calorimeter
  //------------
  auto calorimeterS
    = new G4Box("Calorimeter",     // its name
                calorSizeXY/2, calorSizeXY/2, calorThickness/2); //
its size

  auto calorLV
    = new G4LogicalVolume(
                calorimeterS,     // its solid
                defaultMaterial,  // its material
                "Calorimeter");   // its name

  new G4PVPlacement(
                0,                // no rotation
                G4ThreeVector(),  // at (0,0,0)
                calorLV,          // its logical volume
                "Calorimeter",    // its name
                worldLV,          // its mother  volume
                false,            // no boolean operation
                0,                // copy number
```

```
                fCheckOverlaps);  // checking overlaps

  //------
  // Layer
  //------
  auto layerS
    = new G4Box("Layer",           // its name
                calorSizeXY/2, calorSizeXY/2, layerThickness/2); //
its size

  auto layerLV
    = new G4LogicalVolume(
                layerS,           // its solid
                defaultMaterial,  // its material
                "Layer");         // its name

  new G4PVReplica(
                "Layer",          // its name
                layerLV,          // its logical volume
                calorLV,          // its mother
                kZAxis,           // axis of replication
                nofLayers,        // number of replica
                layerThickness);  // witdth of replica

  //---------
  // Absorber
  //---------
  auto absorberS
    = new G4Box("Abso",            // its name
                calorSizeXY/2, calorSizeXY/2, absoThickness/2); //
its size

  auto absorberLV
    = new G4LogicalVolume(
                absorberS,        // its solid
                absorberMaterial, // its material
                "Abso");          // its name

  fAbsorberPV
    = new G4PVPlacement(
                0,                // no rotation
                G4ThreeVector(0., 0., -gapThickness/2), // its
position
                absorberLV,       // its logical volume

                "Abso",           // its name
                layerLV,          // its mother  volume
                false,            // no boolean operation
                0,                // copy number
                fCheckOverlaps);  // checking overlaps

  //----
  // Gap
  //----
```

```
  //---
  // Modify : 55 x 55 1.0x1.0mm2 Si pads for 5.5x5.5cm2 cross
section.
  // GI modificated 2020 Oct. 26th.
  //---
  G4int ndivx = 55;
  G4int ndivy = 55;


  auto gapS
  = new G4Box("Gap",             // its name
  calorSizeXY/ndivx/2, calorSizeXY/ndivy/2, gapThickness/2);
  // its size has been modified.

  //auto gapS
  //  = new G4Box("Gap",             // its name
  //            calorSizeXY/2, calorSizeXY/2, gapThickness/2);
  // its size has been modified.

  auto gapLV
    = new G4LogicalVolume(
                 gapS,             // its solid
                 gapMaterial,      // its material
                 "Gap");           // its name

  // KM Added
  //  for(G4int ix=0; ix<ndivx; ++ix){
  //   for(G4int iy=0; iy<ndivy; ++iy){
  //    fGapPV
  //       = new G4PVPlacement(
  //          0,                // no rotation
  //          G4ThreeVector(-calorSizeXY/2+(ix+0.5)*calorSizeXY/
ndivx,
  //                        -calorSizeXY/2+(iy+0.5)*calorSizeXY/
ndivy,
  //                        absoThickness/2), // its position
  //          gapLV,            // its logical volume
  //          "Gap",            // its name
  //          layerLV,          // its mother  volume
  //          false,            // no boolean operation
  //          ix*ndivx+iy,                // copy number, should be
incremented.
  //          fCheckOverlaps);  // checking overlaps
  // }
  //}


  // GI Added 20201210.
  for(G4int ix=0; ix<ndivx; ++ix){
    for(G4int iy=0; iy<ndivy; ++iy){
        G4int ipix = ix*ndivx+iy;
        fGapPV[ipix]
         = new G4PVPlacement( 0,                // no rotation
                             G4ThreeVector(-calorSizeXY/2+
```

```
(ix+0.5)*calorSizeXY/ndivx,
                                          -calorSizeXY/2+
(iy+0.5)*calorSizeXY/ndivy,
                                          absoThickness/2), // its
position
                              gapLV,            // its logical
volume
                              "Gap",            // its name
                              layerLV,          // its mother
volume
                              false,            // no boolean
operation
                              ipix,                // copy number,
should be incremented.
                              fCheckOverlaps);  // checking overlaps

    }
  }


  // original.
  // fGapPV
  //   = new G4PVPlacement(
  //               0,                // no rotation
  //               G4ThreeVector(0., 0., absoThickness/2), // its
position
  //               gapLV,            // its logical volume
  //               "Gap",            // its name
  //               layerLV,          // its mother  volume
  //               false,            // no boolean operation
  //               0,                // copy number, should be
incremented.
  //               fCheckOverlaps);  // checking overlaps

  //
  // print parameters
  //
  G4cout
    << G4endl
    <<
"------------------------------------------------------------" <<
G4endl
    << "---> The calorimeter is " << nofLayers << " layers of: [ "
    << absoThickness/mm << "mm of " << absorberMaterial->GetName()
    << " + "
    << gapThickness/mm << "mm of " << gapMaterial->GetName() << " ]
" << G4endl
    <<
"------------------------------------------------------------" <<
G4endl;

  //
  // Visualization attributes
  //
```

```
  worldLV->SetVisAttributes (G4VisAttributes::GetInvisible());

  auto simpleBoxVisAtt= new G4VisAttributes(G4Colour(1.0,1.0,1.0));
  simpleBoxVisAtt->SetVisibility(true);
  calorLV->SetVisAttributes(simpleBoxVisAtt);

  //
  // Always return the physical World
  //
  return worldPV;
}

// Magnetic field.
void B4DetectorConstruction::ConstructSDandField()
{
  // Create global magnetic field messenger.
  // Uniform magnetic field is then created automatically if
  // the field value is not zero.
  G4ThreeVector fieldValue;
  fMagFieldMessenger = new G4GlobalMagFieldMessenger(fieldValue);
  fMagFieldMessenger->SetVerboseLevel(1);

  // Register the field messenger for deleting
  G4AutoDelete::Register(fMagFieldMessenger);
}

(END)
```

2. ヒストグラムの作成や Ntuple の作成などを行う B4RunAction.cc。

```
/// \file B4RunAction.cc
/// \brief Implementation of the B4RunAction class
// KM Modification starts 20200108
// GI Modification starts 20201014
#include "B4RunAction.hh"
#include "B4Analysis.hh"

#include "G4Run.hh"
#include "G4RunManager.hh"
#include "G4UnitsTable.hh"
#include "G4SystemOfUnits.hh"

// GI Added 20201026
#include <sstream>
// RunAction constructor
B4RunAction::B4RunAction()
 : G4UserRunAction()
{
  // set printing event number per each event
  G4RunManager::GetRunManager()->SetPrintProgress(1);

  // Create analysis manager
  // The choice of analysis technology is done via selection of a
namespace
  // in B4Analysis.hh
  auto analysisManager = G4AnalysisManager::Instance();
  G4cout << "Using " << analysisManager->GetType() << G4endl;

  // Create directories
  //analysisManager->SetHistoDirectoryName("histograms");
  //analysisManager->SetNtupleDirectoryName("ntuple");
  analysisManager->SetVerboseLevel(1);
  analysisManager->SetNtupleMerging(true);
    // Note: merging ntuples is available only with Root output

  // Book histograms, ntuple

  // Creating histograms
  analysisManager->CreateH1("Eabs","Edep in absorber", 100, 0.,
800*MeV);
  analysisManager->CreateH1("Egap","Edep in gap", 100, 0., 100*MeV);
  analysisManager->CreateH1("Labs","trackL in absorber", 100, 0.,
1*m);
  analysisManager->CreateH1("Lgap","trackL in gap", 100, 0., 50*cm);

  // Creating ntuple
  analysisManager->CreateNtuple("B4", "Edep and TrackL");
  analysisManager->CreateNtupleDColumn("Eabs");
  analysisManager->CreateNtupleDColumn("Egap");
  analysisManager->CreateNtupleDColumn("Labs");
  analysisManager->CreateNtupleDColumn("Lgap");
  // KM Add 20200216, energy deposit in each layer.
  analysisManager->CreateNtupleDColumn("EabsL0");
  analysisManager->CreateNtupleDColumn("EgapL0");
```

```
  analysisManager->CreateNtupleDColumn("EabsL1");
  analysisManager->CreateNtupleDColumn("EgapL1");
  analysisManager->CreateNtupleDColumn("EabsL2");
  analysisManager->CreateNtupleDColumn("EgapL2");
  // GI Add 20201026, energy deposit in each pixel
  G4int nl = 3;
  G4int ndivx = 55;
  G4int ndivy = 55;
  std::stringstream ss;
  for (G4int il=0; il<nl; ++il){
    for (G4int ix=0; ix<ndivx; ++ix){
      for(G4int iy=0; iy<ndivy; ++iy){
        ss.str("");
        ss<<"EgapL"<<il<<"P"<<ndivx*ix+iy;
        analysisManager->CreateNtupleDColumn(ss.str().c_str());
      }
    }
  }
  analysisManager->FinishNtuple();
}

// Destructor
B4RunAction::~B4RunAction()
{
  delete G4AnalysisManager::Instance();
}

// Begin of RunAction
void B4RunAction::BeginOfRunAction(const G4Run* /*run*/)
{
  //inform the runManager to save random number seed
  //G4RunManager::GetRunManager()->SetRandomNumberStore(true);

  // Get analysis manager
  auto analysisManager = G4AnalysisManager::Instance();

  // Open an output file
  //
  G4String fileName = "B4";
  analysisManager->OpenFile(fileName);
}

// End of RunAction
void B4RunAction::EndOfRunAction(const G4Run* /*run*/)
{
  // print histogram statistics
  //
  auto analysisManager = G4AnalysisManager::Instance();
  if ( analysisManager->GetH1(1) ) {
    G4cout << G4endl << " ----> print histograms statistic ";
    if(isMaster) {
      G4cout << "for the entire run " << G4endl << G4endl;
    }
    else {
```

```
      G4cout << "for the local thread " << G4endl << G4endl;
    }

    G4cout << " EAbs : mean = "
       << G4BestUnit(analysisManager->GetH1(0)->mean(), "Energy")
       << " rms = "
       << G4BestUnit(analysisManager->GetH1(0)->rms(),  "Energy") <<
G4endl;

    G4cout << " EGap : mean = "
       << G4BestUnit(analysisManager->GetH1(1)->mean(), "Energy")
       << " rms = "
       << G4BestUnit(analysisManager->GetH1(1)->rms(),  "Energy") <<
G4endl;

    G4cout << " LAbs : mean = "
      << G4BestUnit(analysisManager->GetH1(2)->mean(), "Length")
      << " rms = "
      << G4BestUnit(analysisManager->GetH1(2)->rms(),  "Length") <<
G4endl;

    G4cout << " LGap : mean = "
      << G4BestUnit(analysisManager->GetH1(3)->mean(), "Length")
      << " rms = "
      << G4BestUnit(analysisManager->GetH1(3)->rms(),  "Length") <<
G4endl;


  }

  // save histograms & ntuple
  //
  analysisManager->Write();
  analysisManager->CloseFile();
}
(END)
```

3.Ntuple に各情報を Fill する B4aEventAction.cc。

```
/// \file B4aEventAction.cc
/// \brief Implementation of the B4aEventAction class
// KM modification starts 20200108
// GI modification starts 20201014

#include "B4aEventAction.hh"
#include "B4RunAction.hh"
#include "B4Analysis.hh"

#include "G4RunManager.hh"
#include "G4Event.hh"
#include "G4UnitsTable.hh"

#include "Randomize.hh"
#include <iomanip>

// Constructor
B4aEventAction::B4aEventAction()
 : G4UserEventAction(),
   fEnergyAbs(0.),
   fEnergyGap(0.),
   fTrackLAbs(0.),
   fTrackLGap(0.)
{}

// Destructor
B4aEventAction::~B4aEventAction()
{}

// Initialization per event
void B4aEventAction::BeginOfEventAction(const G4Event* /*event*/)
{
  // initialisation per event
  fEnergyAbs = 0.;
  fEnergyGap = 0.;
  fTrackLAbs = 0.;
  fTrackLGap = 0.;
  // KM Added 20200216, Energy Deposit in each layer.
  for(G4int i=0; i<3; ++i){
     fEnergyAbsLyr[i] = 0.;
     fEnergyGapLyr[i] = 0.;
  }
  // GI Added 20201026, Energy Deposit in each pixel.
  G4int ndivx = 55;
  G4int ndivy = 55;
  for(G4int ix=0; ix<ndivx; ++ix){
    for(G4int iy=0; iy<ndivy; ++iy){
      G4int ipix = ndivx*ix + iy;
      fEnergyGapL0Pix[ipix] = 0.;
      fEnergyGapL1Pix[ipix] = 0.;
      fEnergyGapL2Pix[ipix] = 0.;
    }
  }
}
```

```
// Event function.
void B4aEventAction::EndOfEventAction(const G4Event* event)
{
  // Accumulate statistics
  //

  // get analysis manager
  // KM Histogram and Ntuple creation is done inside RunAcrion.
  // GI Ntuple creation of each pixel is done inside RunAcrion.
  // G4AnalysisManager::Instance() returns corresponding the pointer
(static).
  auto analysisManager = G4AnalysisManager::Instance();

  // fill histograms
  analysisManager->FillH1(0, fEnergyAbs);
  analysisManager->FillNtupleDColumn(1, fEnergyGap);
  analysisManager->FillNtupleDColumn(2, fTrackLAbs);
  analysisManager->FillNtupleDColumn(3, fTrackLGap);
  // KM Added 20200216, Energy Deposit in each Layer.
  for(G4int i=0; i<3; ++i){
     G4int iptrAbs = i*2 + 4;
     G4int iptrGap = i*2 + 5;
     analysisManager->FillNtupleDColumn(iptrAbs, fEnergyAbsLyr[i]);
     analysisManager->FillNtupleDColumn(iptrGap, fEnergyGapLyr[i]);
  }
  // GI Added 20201026, Energy Deposit in each pixel.
  G4int ndivx = 55;
  G4int ndivy = 55;
  for(G4int ix=0; ix<ndivx; ++ix){
    for(G4int iy=0; iy<ndivy; ++iy){
       G4int ipix = ix*ndivx + iy;
       G4int iptrL0 = ipix + 10;
       analysisManager->FillNtupleDColumn(iptrL0,
fEnergyGapL0Pix[ipix]);
    }
  }
  for(G4int ix=0; ix<ndivx; ++ix){
    for(G4int iy=0; iy<ndivy; ++iy){
      G4int ipix = ix*ndivx + iy;
      G4int iptrL1 = ipix + 10 + ndivx*ndivy;
      analysisManager->FillNtupleDColumn(iptrL1,
fEnergyGapL1Pix[ipix]);
    }
  }
  for(G4int ix=0; ix<ndivx; ++ix){
    for(G4int iy=0; iy<ndivy; ++iy){
      G4int ipix = ix*ndivx + iy;
      G4int iptrL2 = ipix + 10 + 2*ndivx*ndivy;
      analysisManager->FillNtupleDColumn(iptrL2,
fEnergyGapL2Pix[ipix]);
    }
  }
  analysisManager->AddNtupleRow();
```

```
  // Print per event (modulo n)
  //
  auto eventID = event->GetEventID();
  auto printModulo = G4RunManager::GetRunManager()-
>GetPrintProgress();
  if ( ( printModulo > 0 ) && ( eventID % printModulo == 0 ) ) {
    G4cout << "---> End of event: " << eventID << G4endl;

    G4cout
       << "   Absorber: total energy: " << std::setw(7)
                                        <<
G4BestUnit(fEnergyAbs,"Energy")
       << "       total track length: " << std::setw(7)
                                        <<
G4BestUnit(fTrackLAbs,"Length")
       << G4endl
       << "        Gap: total energy: " << std::setw(7)
                                        <<
G4BestUnit(fEnergyGap,"Energy")
       << "       total track length: " << std::setw(7)
                                        <<
G4BestUnit(fTrackLGap,"Length")
       << G4endl;
  }
}
```

4. エネルギー損失を step 毎に足し合わせる計算をする B4aSteppingAction.cc。

```
/// \file B4aSteppingAction.cc
/// \brief Implementation of the B4aSteppingAction class
// KM : Modification starts 20200107.
// GI : Modification starts 20201030.
#include "B4aSteppingAction.hh"
#include "B4aEventAction.hh"
#include "B4DetectorConstruction.hh"

#include "G4Step.hh"
#include "G4RunManager.hh"


B4aSteppingAction::B4aSteppingAction(
                      const B4DetectorConstruction*
detectorConstruction,
                      B4aEventAction* eventAction)
  : G4UserSteppingAction(),
    fDetConstruction(detectorConstruction),
    fEventAction(eventAction)
{}


B4aSteppingAction::~B4aSteppingAction()
{}


void B4aSteppingAction::UserSteppingAction(const G4Step* step)
{
// Collect energy and track length step by step

   // get volume of the current step
   auto volume = step->GetPreStepPoint()->GetTouchableHandle()-
>GetVolume();

   // energy deposit
   auto edep = step->GetTotalEnergyDeposit();

   // step length
   G4double stepLength = 0.;
   if ( step->GetTrack()->GetDefinition()->GetPDGCharge() != 0. ) {
      stepLength = step->GetStepLength();
   }

   if ( volume == fDetConstruction->GetAbsorberPV() ) {
      // KM Added 20200216.
      G4int lyrid
         = step->GetPreStepPoint()->GetTouchableHandle()-
>GetReplicaNumber(1);
      fEventAction->AddAbs(edep,stepLength,lyrid);
      // fEventAction->AddAbs(edep,stepLength); // original
   }

   for(G4int ipix=0; ipix<3025 ; ++ipix){
   if ( volume == fDetConstruction->GetGapPV(ipix) ) {
```

```
     if( edep > 0.05 ){  //GI Added 20210202.
      // get layer number (replica Number)
      // KM Added 20200216.
      G4int lyrid
         = step->GetPreStepPoint()->GetTouchableHandle()-
>GetReplicaNumber(1);

      // get pixel number in the layer
      // GI Added 20201030.
      //G4int Pixid
      //        = step->GetPreStepPoint()->GetTouchableHandle()-
>GetCopyNumber(0);

      // check program
      //      G4cout << "lyrid:" << lyrid << " " ;
      //      G4cout << "Pixid:" << ipix << "\n" ;

        fEventAction->AddGap(edep,stepLength,lyrid,ipix); // GI
Added 20201030.
      //  fEventAction->AddGap(edep,stepLength,lyrid); // KM Added
20200216.
      // fEventAction->AddGap(edep,stepLength); // original.
     }
   }
   }
}
```

5. シミュレーションを実行させる時に読ませるマクロファイルである run2.mac。

```
# Macro file for example B4
#
# To be run preferably in batch, without graphics:
# % exampleB4[a,b,c,d] run2.mac
#
#/run/numberOfWorkers 4
/run/initialize
#
# Default kinemtics:
# electron 50 MeV in direction (0.,0.,1.)
/gun/particle gamma
/gun/energy 1000 MeV
#/gun/position 600 600 0 mm
#/gun/direction 0 0 1
#
# 1000 events
#
/run/printProgress 100
/run/beamOn 100
```

6. 数値計算により解析を行う解析マクロファイル

```
//root_macro2.cpp
//20201126 Written by GI.
//This program analyzes 3layer's case.

#include <iostream>
#include "TFile.h"
#include "TTree.h"
#include <sstream>
#include <math.h>
#include <cassert>  //Added to use vector.
#include <vector>   //Added to use vector.
#include <numeric>  //Added to calculate inner products.
#include <iterator> //Added to calculate inner products.


//analyze function----------------------------------------------//
std::vector<double> LeastSquares(double pos0,double pos1,double pos3,double
pos4,double pos6,double pos7)
{
  double PI = acos(-1.0);

  double calorSizeXY = 55.0 ;
  double absoThickness = 11.2 ;
  double gapThickness = 0.3 ;
  double pos2 = absoThickness+gapThickness/2;
  double pos5 = absoThickness*2+gapThickness*3/2;
  double pos8 = absoThickness*3+gapThickness*5/2;

  double R[3];    //length from incident position
  double d[3];    //length from Vgamma vector
  for(int i=0; i<3; ++i){
    R[i] = 0.;
    d[i] = 0.;
  }

  double d2 = 9999.0;
  std::vector<double> angle = {100.,100.};
  std::vector<double> Vgamma = {0.,0.,0.};

  // for(double px=-0.30; px<=0.30; px+=0.1){
  //  for(double py=-0.30; py<=0.30; py+=0.1){
     for(double phi=-PI/2; phi<=PI/2; phi+=0.001){
      for(double theta=0.0; theta<PI/2; theta+=0.001){

      Vgamma[0] = sin(theta)*cos(phi);
      Vgamma[1] = sin(theta)*sin(phi);
      Vgamma[2] = cos(theta);

      //      double posd0 = pos0 - px;
      //      double posd1 = pos1 - py;
```

```
      //      double posd3 = pos3 - px;
      //      double posd4 = pos4 - py;
      //      double posd6 = pos6 - px;
      //      double posd7 = pos7 - py;

      R[0] = sqrt( pow(pos0,2.0) + pow(pos1,2.0) + pow(pos2,2.0));  //layer0
      R[1] = sqrt( pow(pos3,2.0) + pow(pos4,2.0) + pow(pos5,2.0));  //layer1
      R[2] = sqrt( pow(pos6,2.0) + pow(pos7,2.0) + pow(pos8,2.0));  //layer2

      double ptheta[3];
      ptheta[0] = acos((pos0*Vgamma[0]+pos1*Vgamma[1]+pos2*Vgamma[2])/
R[0]);  //layer0
      ptheta[1] = acos((pos3*Vgamma[0]+pos4*Vgamma[1]+pos5*Vgamma[2])/
R[1]);  //layer1
      ptheta[2] = acos((pos6*Vgamma[0]+pos7*Vgamma[1]+pos8*Vgamma[2])/
R[2]);  //layer2

      double pd2 = 0.;
      for(int i=0; i<3; ++i){
        d[i] = R[i]*sin(ptheta[i]);
        pd2 += d[i]*d[i];
      }

      if(pd2 < d2){
        d2 = pd2;
        angle[0] = theta; // theta
        angle[1] = phi;   // phi
        //       x = px;
        //       y = py;
       }

        }
      }
     //    }
     //  }
  std::cout << "d2=" << d2 << std::endl;
  std::cout << "theta=" << angle[0] << std::endl;
  std::cout << "phi=" << angle[1] << std::endl;

  return angle;
}

//main program----------------------------------------------------------------//
void root_macro2(){

  // Set pixel positions.
  int nl = 3;
  int ndivx = 55;
  int ndivy = 55;
  double calorSizeXY = 55.0 ;
```

```
  double absoThickness = 11.2 ;
  double gapThickness = 0.3 ;
  double pos_define[3025][2];

  for(int ix=0; ix<ndivx; ++ix){
    for(int iy=0; iy<ndivy; ++iy){
      int ipix = ix*ndivx+iy;
      pos_define[ipix][0] = -calorSizeXY/2+(ix+0.5)*calorSizeXY/ndivx;
      pos_define[ipix][1] = -calorSizeXY/2+(iy+0.5)*calorSizeXY/ndivy;
        }
  }

  // readfile.
  TFile* f = new TFile("B4test_3L3025P_1000MeVgamma.root");
  TTree* tt = (TTree*)f -> Get("B4");
  int nEvent = tt -> GetEntries();  // Get the number of total entris.
  //  std::cout << "the number of event : " << nEvent << std::endl;

  // Set histogram.
  TH1D* hpos0 = new TH1D("pos[0]","pos[0];[mm];",150.0,-30.0,30.0);
  TH1D* hpos1 = new TH1D("pos[1]","pos[1];[mm];",150.0,-30.0,30.0);
  TH1D* hpos3 = new TH1D("pos[3]","pos[3];[mm];",150.0,-30.0,30.0);
  TH1D* hpos4 = new TH1D("pos[4]","pos[4];[mm];",150.0,-30.0,30.0);
  TH1D* hpos6 = new TH1D("pos[6]","pos[6];[mm];",150.0,-30.0,30.0);
  TH1D* hpos7 = new TH1D("pos[7]","pos[7];[mm];",150.0,-30.0,30.0);

  TH1D* hpixl0 = new TH1D("hit in layer0","hit in layer0",3025.0,0.0,3024.0);
  TH1D* hpixl1 = new TH1D("hit in layer1","hit in layer1",3025.0,0.0,3024.0);
  TH1D* hpixl2 = new TH1D("hit in layer2","hit in layer2",3025.0,0.0,3024.0);

  TH1D* hnofpixl0 = new TH1D("the number of pixels hit in layer0","the number of
pixels hit in layer0",25.0,0.0,24.0);
  TH1D* hnofpixl1 = new TH1D("the number of pixels hit in layer1","the number of
pixels hit in layer1",25.0,0.0,24.0);
  TH1D* hnofpixl2 = new TH1D("the number of pixels hit in layer2","the number of
pixels hit in layer2",25.0,0.0,24.0);

  TH1D* htheta = new TH1D("theta","theta;[radian];",40.0,0.,1.0);
  //  TH1D* hphi = new TH1D("phi","phi",64.0,-1.6,1.6);

  //Set Canvas.
  TCanvas* hcpos0 = new TCanvas("hcpos0","hcpos0",600,600);
  TCanvas* hcpos1 = new TCanvas("hcpos1","hcpos1",600,600);
  TCanvas* hcpos3 = new TCanvas("hcpos3","hcpos3",600,600);
  TCanvas* hcpos4 = new TCanvas("hcpos4","hcpos4",600,600);
  TCanvas* hcpos6 = new TCanvas("hcpos6","hcpos6",600,600);
  TCanvas* hcpos7 = new TCanvas("hcpos7","hcpos7",600,600);

  TCanvas* hcpixl0 = new TCanvas("hcpixl0","hcpixl0",600,600);
  TCanvas* hcpixl1 = new TCanvas("hcpixl1","hcpixl1",600,600);
```

```
TCanvas* hcpixl2 = new TCanvas("hcpixl2","hcpixl2",600,600);

TCanvas* hcnofpixl0 = new TCanvas("hcnofpixl0","hcnofpixl0",600,600);
TCanvas* hcnofpixl1 = new TCanvas("hcnofpixl1","hcnofpixl1",600,600);
TCanvas* hcnofpixl2 = new TCanvas("hcnofpixl2","hcnofpixl2",600,600);

TCanvas* hctheta = new TCanvas("hctheta","hctheta",600,600);
//  TCanvas* hcphi = new TCanvas("hcphi","hcphi",600,600);


// Initialize array (Egap in each pixel).
double Epix[nl*ndivx*ndivy];
for(int i=0; i<nl*ndivx*ndivy; ++i){
  Epix[i] = 0.;
}

// Give address to Branch of each pixel's Egap.
std::stringstream ss;

for(int il=0; il<nl; ++il){
  for(int ix=0; ix<ndivx; ++ix){
    for(int iy=0; iy<ndivy; ++iy){
      int ipix = ndivx*ix+iy;
      int iEpix = ipix + il*ndivx*ndivy;
      ss.str("");
      ss << "EgapL" << il << "P" << ipix ;
      tt -> SetBranchAddress(ss.str().c_str(),&Epix[iEpix]);
    }
  }
}

// Give address to Branch of each layer's Egap.
double EgapL[3];
for(int il=0; il<nl; ++il){
  EgapL[il] = 0.;
}
tt -> SetBranchAddress("EgapL0",&EgapL[0]);
tt -> SetBranchAddress("EgapL1",&EgapL[1]);
tt -> SetBranchAddress("EgapL2",&EgapL[2]);


//-----------------------------------------------------//
// Analyze.
// 1.search for position electron went through in each layer
// 2.connect two positions
//-----------------------------------------------------//

// 1.search for position electron went through in each layer ----------------------------//
double pos[9];
pos[2] = absoThickness+gapThickness/2;
```

```
 pos[5] = absoThickness*2+gapThickness*3/2;
 pos[8] = absoThickness*3+gapThickness*5/2;

 for(int event=0; event<nEvent; ++event){
  tt->GetEntry(event);
  std::cout << "****************************************" << std::endl;
  std::cout << "event:" << event << std::endl;
  pos[0] = 0.;
  pos[1] = 0.;
  pos[3] = 0.;
  pos[4] = 0.;
  pos[6] = 0.;
  pos[7] = 0.;
  std::vector<double> angle = {110.,110.};

  int npix[3];
  for(int i=0; i<3; ++i){
   npix[i] = 0.;
  }

 for(int il=0; il<nl; ++il){

  for(int ix=0; ix<ndivx; ++ix){

   for(int iy=0; iy<ndivy; ++iy){
    int ipix = ndivx*ix+iy;
    int iEpix = ipix + il*ndivx*ndivy;

    if(Epix[iEpix]>0.05){
     ss.str("");
     ss << "EgapL" << il << "P" << ipix;
      std::cout << ss.str().c_str() << ":" << Epix[iEpix] << std::endl;
      pos[il*3] = pos[il*3] + pos_define[ipix][0]*Epix[iEpix]/EgapL[il];            // being
divided by Egap of the layer, later.
      pos[il*3+1] = pos[il*3+1] + pos_define[ipix][1]*Epix[iEpix]/EgapL[il];   // being
divided by Egap of the layer, later.
      ++npix[il];

      if(il==0){
      hpixl0 -> Fill(ipix);
      }else if(il==1){
        hpixl1 -> Fill(ipix);
      } else if(il==2){
          hpixl2 -> Fill(ipix);
        }

    }
   }
  }
```

```
 if(npix[il]==0){
       pos[il*3] = -9999.0;
       pos[il*3+1] = -9999.0;
     }
 }

 std::cout << "npix[0]:" << npix[0] << std::endl;
 std::cout << "npix[1]:" << npix[1] << std::endl;
 std::cout << "npix[2]:" << npix[2] << std::endl;

 hnofpixl0 -> Fill(npix[0]);
 hnofpixl1 -> Fill(npix[1]);
 hnofpixl2 -> Fill(npix[2]);

   for(int i=0; i<9; ++i){
      std::cout << "pos[" << i << "]=" << pos[i] << std::endl;
   }

 // 2.connect two positions ------------------------------------------------------------//

   if( npix[0] != 0 && npix[1] != 0 && npix[2] != 0){
     angle = LeastSquares(pos[0],pos[1],pos[3],pos[4],pos[6],pos[7]);
     htheta -> Fill(angle[0]);
     // hphi -> Fill(angle[1]);

   std::cout << "theta=" << angle[0] << ",phi=" << angle[1] << std::endl;
   }


hpos0 -> Fill(pos[0]);
hpos1 -> Fill(pos[1]);
hpos3 -> Fill(pos[3]);
hpos4 -> Fill(pos[4]);
hpos6 -> Fill(pos[6]);
hpos7 -> Fill(pos[7]);

}


hcpos0 -> cd();
hpos0 -> Draw();
hpos0 -> SetLineColor(2);
//  hcpos0 -> Print("pos[0]_B4test_3L3025P_1000MeVgamma.pdf");

hcpos1 -> cd();
hpos1 -> Draw();
hpos1 -> SetLineColor(2);
//hcpos1 -> Print("pos[1]_B4_3L121P_1000MeVgamma.pdf");

hcpos3 -> cd();
```

```
hpos3 -> Draw();
hpos3 -> SetLineColor(2);
//  hcpos3 -> Print("pos[3]_B4test_3L3025P_1000MeVgamma.pdf");

hcpos4 -> cd();
hpos4 -> Draw();
hpos4 -> SetLineColor(2);
//hcpos4 -> Print("pos[4]_B4_3L121P_1000MeVgamma.pdf");

hcpos6 -> cd();
hpos6 -> Draw();
hpos6 -> SetLineColor(2);
//  hcpos6 -> Print("pos[6]_B4test_3L3025P_1000MeVgamma.pdf");

hcpos7 -> cd ();
hpos7 -> Draw();
hpos7 -> SetLineColor(2);
//hcpos7 -> Print("pos[7]_B4_3L121P_1000MeVgamma.pdf");

hcpixl0 -> cd ();
hpixl0 -> Draw();
hpixl0 -> SetLineColor(2);
//hcpixl0 -> Print("hitl0_B4_3L121P_1000MeVgamma.pdf");

hcpixl1 -> cd ();
hpixl1 -> Draw();
hpixl1 -> SetLineColor(2);
//hcpixl1 -> Print("hitl1_B4_3L121P_1000MeVgamma.pdf");

hcpixl2 -> cd ();
hpixl2 -> Draw();
hpixl2 -> SetLineColor(2);
//hcpixl2 -> Print("hitl2_B4_3L121P_1000MeVgamma.pdf");

hcnofpixl0 -> cd ();
hnofpixl0 -> Draw();
hnofpixl0 -> SetLineColor(2);
//  hcnofpixl0 -> Print("the number of pixels hit in layer0_B4check2.pdf");

hcnofpixl1 -> cd ();
hnofpixl1 -> Draw();
hnofpixl1 -> SetLineColor(2);
//  hcnofpixl1 -> Print("the number of pixels hit in layer1_B4check2.pdf");

hcnofpixl2 -> cd ();
hnofpixl2 -> Draw();
hnofpixl2 -> SetLineColor(2);
//  hcnofpixl2 -> Print("the number of pixels hit in layer2_B4check2.pdf");

hctheta -> cd();
```

```
htheta -> Draw();
htheta -> SetLineColor(2);
//  hctheta -> Print("theta in B4_3L121P_1000MeVgamma.pdf");

//  hcphi ->cd();
//  hphi -> Draw();
//  hphi -> SetLineColor(2);
//  hcphi -> Print("phi in B4_3L121P_1000MeVgamma.pdf");

//  tt -> Draw("EgapL0P60");
//  tt -> Draw("EgapL1P60");
//  tt -> Draw("EgapL2P60");

}
```

7. 最小二乗法により解析を行う解析マクロファイル

```
//root_macro5.cpp
//20201126 Written by GI.
//This program analyzes 3layer's case.
//Try to change analyze function of root_macro2.cpp.

#include <iostream>
#include "TFile.h"
#include "TTree.h"
#include <sstream>
#include <math.h>
#include <cassert>  //Added to use vector.
#include <vector>   //Added to use vector.
#include <numeric>  //Added to calculate inner products.
#include <iterator> //Added to calculate inner products.


//analyze function----------------------------------------------//
std::vector<double> LeastSquares(double pos0,double pos1,double pos3,double
pos4,double pos6,double pos7)
{
  double PI = acos(-1.0);

  double calorSizeXY = 55.0 ;
  double absoThickness = 11.2 ;
  double gapThickness = 0.3 ;
  double pos2 = absoThickness+gapThickness/2;
  double pos5 = absoThickness*2+gapThickness*3/2;
  double pos8 = absoThickness*3+gapThickness*5/2;

  std::vector<double> gamma = {100.,100.,100.,100.,100.}; //{x0,y0,e1,e2,theta}

  double A = 0.;  // X
  double B = 0.;  // Y
  double C = 0.;  // Z
  double D = 0.;  // XZ
  double E = 0.;  // YZ
  double F = 0.;  // Z^2
  double R = 0.;

  A = pos0 + pos3 + pos6;
  B = pos1 + pos4 + pos7;
  C = pos2 + pos5 + pos8;
  D = pos0*pos2 + pos3*pos5 + pos6*pos8;
  E = pos1*pos2 + pos4*pos5 + pos7*pos8;
  F = pow(pos2,2.0) + pow(pos5,2.0) + pow(pos8,2.0);

  gamma[2] = (3*D-A*C)/(3*F-C*C);  //e1
  gamma[3] = (3*E-B*C)/(3*F-C*C);  //e2
  gamma[0] = (A*F-C*D)/(3*F-C*C);  //x0
  gamma[1] = (B*F-C*E)/(3*F-C*C);  //y0
```

```
 //Fix(x0,y0)-----------------------------------
  //gamma[2] = XZ/Z2;  //e1
  //gamma[3] = YZ/Z2;  //e2
  //gamma[0] = 0.;     //x0
  //gamma[1] = 0.;     //y0

  R = sqrt( pow(gamma[2],2.0) + pow(gamma[3],2.0) + 1);
  gamma[2] = gamma[2]/R;
  gamma[3] = gamma[3]/R;

  gamma[4] = acos(1/R);    //theta

  return gamma;

}

//main program-----------------------------------------------------------------//
void root_macro5(){

  // Set pixel positions.
  int nl = 3;
  int ndivx = 55;
  int ndivy = 55;
  double calorSizeXY = 55.0 ;
  double absoThickness = 11.2 ;
  double gapThickness = 0.3 ;
  double pos_define[3025][2];

  for(int ix=0; ix<ndivx; ++ix){
    for(int iy=0; iy<ndivy; ++iy){
      int ipix = ix*ndivx+iy;
      pos_define[ipix][0] = -calorSizeXY/2+(ix+0.5)*calorSizeXY/ndivx;
      pos_define[ipix][1] = -calorSizeXY/2+(iy+0.5)*calorSizeXY/ndivy;
        }
  }

  // readfile.
  TFile* f = new TFile("B4test_3L3025P_1000MeVgamma.root");
  TTree* tt = (TTree*)f -> Get("B4");
  int nEvent = tt -> GetEntries();  // Get the number of total entris.
  //  std::cout << "the number of event : " << nEvent << std::endl;

  // Set histogram.
  TH1D* hpos0 = new TH1D("pos[0]","pos[0];[mm];the number of
events",150.0,-30.0,30.0);
  TH1D* hpos1 = new TH1D("pos[1]","pos[1];[mm];the number of
events",150.0,-30.0,30.0);
  TH1D* hpos3 = new TH1D("pos[3]","pos[3];[mm];the number of
events",150.0,-30.0,30.0);
```

```
  TH1D* hpos4 = new TH1D("pos[4]","pos[4];[mm];the number of
events",150.0,-30.0,30.0);
  TH1D* hpos6 = new TH1D("pos[6]","pos[6];[mm];the number of
events",150.0,-30.0,30.0);
  TH1D* hpos7 = new TH1D("pos[7]","pos[7];[mm];the number of
events",150.0,-30.0,30.0);

  TH1D* hpixl0 = new TH1D("hit in layer0","hit in layer0;pixel ID;the number of
events",3025.0,0.0,3024.0);
  TH1D* hpixl1 = new TH1D("hit in layer1","hit in layer1;pixel ID;the number of
events",3025.0,0.0,3024.0);
  TH1D* hpixl2 = new TH1D("hit in layer2","hit in layer2;pixel ID;the number of
events",3025.0,0.0,3024.0);

  TH1D* hnofpixl0 = new TH1D("the number of pixels hit in layer0","the number of
pixels hit in layer0;the number of pixels hit in layer0;the number of
events",25.0,0.0,24.0);
  TH1D* hnofpixl1 = new TH1D("the number of pixels hit in layer1","the number of
pixels hit in layer1;the number of pixels hit in layer1;the number of
events",25.0,0.0,24.0);
  TH1D* hnofpixl2 = new TH1D("the number of pixels hit in layer2","the number of
pixels hit in layer2;the number of pixels hit in layer2;the number of
events",25.0,0.0,24.0);

  TH1D* htheta = new TH1D("theta","theta;[radian];the number of
events",40.0,0.,1.0);
  //  TH1D* hphi = new TH1D("phi","phi",64.0,-1.6,1.6);
  TH1D* hx0 = new TH1D("x0","x0;[mm];the number of events",150.0,-30.0,30.0);
  TH1D* hy0 = new TH1D("y0","y0;[mm];the number of events",150.0,-30.0,30.0);
  TH1D* he1 = new TH1D("e1","e1;;the number of events",200.0,-1.0,1.0);
  TH1D* he2 = new TH1D("e2","e2;;the number of events",200.0,-1.0,1.0);

  //Set Canvas.
  TCanvas* hcpos0 = new TCanvas("hcpos0","hcpos0",600,500);
  TCanvas* hcpos1 = new TCanvas("hcpos1","hcpos1",600,500);
  TCanvas* hcpos3 = new TCanvas("hcpos3","hcpos3",600,500);
  TCanvas* hcpos4 = new TCanvas("hcpos4","hcpos4",600,500);
  TCanvas* hcpos6 = new TCanvas("hcpos6","hcpos6",600,500);
  TCanvas* hcpos7 = new TCanvas("hcpos7","hcpos7",600,500);

  TCanvas* hcpixl0 = new TCanvas("hcpixl0","hcpixl0",600,500);
  TCanvas* hcpixl1 = new TCanvas("hcpixl1","hcpixl1",600,500);
  TCanvas* hcpixl2 = new TCanvas("hcpixl2","hcpixl2",600,500);

  TCanvas* hcnofpixl0 = new TCanvas("hcnofpixl0","hcnofpixl0",600,500);
  TCanvas* hcnofpixl1 = new TCanvas("hcnofpixl1","hcnofpixl1",600,500);
  TCanvas* hcnofpixl2 = new TCanvas("hcnofpixl2","hcnofpixl2",600,500);

  TCanvas* hctheta = new TCanvas("hctheta","hctheta",600,500);
  //  TCanvas* hcphi = new TCanvas("hcphi","hcphi",600,600);
```

```
TCanvas* hcx0 = new TCanvas("hcx0","hcx0",600,500);
TCanvas* hcy0 = new TCanvas("hcy0","hcy0",600,500);
TCanvas* hce1 = new TCanvas("hce1","hce1",600,500);
TCanvas* hce2 = new TCanvas("hce2","hce2",600,500);


// Initialize array (Egap in each pixel).
double Epix[nl*ndivx*ndivy];
for(int i=0; i<nl*ndivx*ndivy; ++i){
  Epix[i] = 0.;
}

// Give address to Branch of each pixel's Egap.
std::stringstream ss;

for(int il=0; il<nl; ++il){
  for(int ix=0; ix<ndivx; ++ix){
    for(int iy=0; iy<ndivy; ++iy){
      int ipix = ndivx*ix+iy;
      int iEpix = ipix + il*ndivx*ndivy;
      ss.str("");
      ss << "EgapL" << il << "P" << ipix ;
      tt -> SetBranchAddress(ss.str().c_str(),&Epix[iEpix]);
    }
  }
}

// Give address to Branch of each layer's Egap.
double EgapL[3];
for(int il=0; il<nl; ++il){
  EgapL[il] = 0.;
}
tt -> SetBranchAddress("EgapL0",&EgapL[0]);
tt -> SetBranchAddress("EgapL1",&EgapL[1]);
tt -> SetBranchAddress("EgapL2",&EgapL[2]);


//-----------------------------------------------------//
// Analyze.
// 1.search for position electron went through in each layer
// 2.connect two positions
//-----------------------------------------------------//

// 1.search for position electron went through in each layer ----------------------------//
double pos[9];
pos[2] = absoThickness+gapThickness/2;
pos[5] = absoThickness*2+gapThickness*3/2;
pos[8] = absoThickness*3+gapThickness*5/2;

for(int event=0; event<nEvent; ++event){
```

```
    tt->GetEntry(event);
    std::cout << "****************************************" << std::endl;
    std::cout << "event:" << event << std::endl;
    pos[0] = 0.;
    pos[1] = 0.;
    pos[3] = 0.;
    pos[4] = 0.;
    pos[6] = 0.;
    pos[7] = 0.;
    std::vector<double> gamma = {110.,110.,110.,110.};

    int npix[3];
    for(int i=0; i<3; ++i){
      npix[i] = 0.;
    }

   for(int il=0; il<nl; ++il){

    for(int ix=0; ix<ndivx; ++ix){

     for(int iy=0; iy<ndivy; ++iy){
       int ipix = ndivx*ix+iy;
       int iEpix = ipix + il*ndivx*ndivy;

       if(Epix[iEpix]>0.05){
         ss.str("");
         ss << "EgapL" << il << "P" << ipix;
          std::cout << ss.str().c_str() << ":" << Epix[iEpix] << std::endl;
          pos[il*3] = pos[il*3] + pos_define[ipix][0]*Epix[iEpix]/EgapL[il];            // being
divided by Egap of the layer, later.
          pos[il*3+1] = pos[il*3+1] + pos_define[ipix][1]*Epix[iEpix]/EgapL[il];   // being
divided by Egap of the layer, later.
          ++npix[il];

          if(il==0){
          hpixl0 -> Fill(ipix);
          }else if(il==1){
            hpixl1 -> Fill(ipix);
          } else if(il==2){
              hpixl2 -> Fill(ipix);
            }

       }
     }
    }

   if(npix[il]==0){
         pos[il*3] = -9999.0;
         pos[il*3+1] = -9999.0;
       }
```

```
 }

 std::cout << "npix[0]:" << npix[0] << std::endl;
 std::cout << "npix[1]:" << npix[1] << std::endl;
 std::cout << "npix[2]:" << npix[2] << std::endl;

 hnofpixl0 -> Fill(npix[0]);
 hnofpixl1 -> Fill(npix[1]);
 hnofpixl2 -> Fill(npix[2]);

  for(int i=0; i<9; ++i){
     std::cout << "pos[" << i << "]=" << pos[i] << std::endl;
  }

 // 2.connect two positions ------------------------------------------------------------//

  if( npix[0] != 0 && npix[1] != 0 && npix[2] != 0){
    gamma = LeastSquares(pos[0],pos[1],pos[3],pos[4],pos[6],pos[7]);
    htheta -> Fill(gamma[4]);
    // hphi -> Fill(angle[1]);

  hx0 -> Fill(gamma[0]);
  hy0 -> Fill(gamma[1]);
  he1 -> Fill(gamma[2]);
  he2 -> Fill(gamma[3]);
  }


hpos0 -> Fill(pos[0]);
hpos1 -> Fill(pos[1]);
hpos3 -> Fill(pos[3]);
hpos4 -> Fill(pos[4]);
hpos6 -> Fill(pos[6]);
hpos7 -> Fill(pos[7]);
}


hcpos0 -> cd();
hpos0 -> Draw();
hpos0 -> SetLineColor(2);
// hcpos0 -> Print("pos[0]_B4test_3L3025P_1000MeVgamma.pdf");


hcpos1 -> cd();
hpos1 -> Draw();
hpos1 -> SetLineColor(2);
// hcpos1 -> Print("pos[1]_B4test_3L3025P_1000MeVgamma.pdf");

hcpos3 -> cd();
hpos3 -> Draw();
```

```
hpos3 -> SetLineColor(2);
// hcpos3 -> Print("pos[3]_B4test_3L3025P_1000MeVgamma.pdf");

hcpos4 -> cd();
hpos4 -> Draw();
hpos4 -> SetLineColor(2);
// hcpos4 -> Print("pos[4]_B4test_3L3025P_1000MeVgamma.pdf");

hcpos6 -> cd();
hpos6 -> Draw();
hpos6 -> SetLineColor(2);
//hcpos6 -> Print("pos[6]_B4test_3L3025P_1000MeVgamma.pdf");

hcpos7 -> cd ();
hpos7 -> Draw();
hpos7 -> SetLineColor(2);
//hcpos7 -> Print("pos[7]_B4test_3L3025P_1000MeVgamma.pdf");

hcpixl0 -> cd ();
hpixl0 -> Draw();
hpixl0 -> SetLineColor(2);
//hcpixl0 -> Print("hitl0_B4test_3L3025P_1000MeVgamma.pdf");

hcpixl1 -> cd ();
hpixl1 -> Draw();
hpixl1 -> SetLineColor(2);
//hcpixl1 -> Print("hitl1_B4tet_3L3025P_1000MeVgamma.pdf");

hcpixl2 -> cd ();
hpixl2 -> Draw();
hpixl2 -> SetLineColor(2);
//hcpixl2 -> Print("hitl2_B4test_3L484P_1000MeVgamma.pdf");

hcnofpixl0 -> cd ();
hnofpixl0 -> Draw();
hnofpixl0 -> SetLineColor(2);
//hcnofpixl0 -> Print("the number of pixels hit in layer0_B4check2n.pdf");

hcnofpixl1 -> cd ();
hnofpixl1 -> Draw();
hnofpixl1 -> SetLineColor(2);
//hcnofpixl1 -> Print("the number of pixels hit in layer1_B4check2n.pdf");

hcnofpixl2 -> cd ();
hnofpixl2 -> Draw();
hnofpixl2 -> SetLineColor(2);
//hcnofpixl2 -> Print("the number of pixels hit in layer2_B4check2n.pdf");


hcx0 -> cd ();
```

```
hx0 -> Draw();
hx0 -> SetLineColor(2);
hcx0 -> Print("x0right in B4test_3L3025P_1000MeVgamma.pdf");

hcy0 -> cd ();
hy0 -> Draw();
hy0 -> SetLineColor(2);
hcy0 -> Print("y0right in B4test_3L3025P_1000MeVgamma.pdf");

hce1 -> cd ();
he1 -> Draw();
he1 -> SetLineColor(2);
hce1 -> Print("e1right in B4test_3L3025P_1000MeVgamma.pdf");

hce2 -> cd ();
he2 -> Draw();
he2 -> SetLineColor(2);
hce2 -> Print("e2right in B4test_3L3025P_1000MeVgamma.pdf");

hctheta -> cd();
htheta -> Draw();
htheta -> SetLineColor(2);
hctheta -> Print("theta5right in B4test_3L3025P_1000MeVgamma.pdf");

//  tt -> Draw("EgapL0P60");
//  tt -> Draw("EgapL1P60");
//  tt -> Draw("EgapL2P60");

}
```