目次

第 1 章　序章

我々は、ガンマ線天体の発見以来、エックス線・ガンマ線衛星の活躍によって、宇宙の至る所で高エネルギー現象が起きていることを知った。活動銀河核(AGN)、ブラックホール候補天体、パルサーなどに付随する超高エネルギーへの粒子の加速とジェット噴射、超新星残骸におけるショックフロントの存在とその構造、銀河や銀河団に閉じ込められた大量の高温プラズマの存在、ガンマ線バースト、そして GeV 領域にまで粒子が加速している太陽フレアなどがある。これらの高エネルギー天体は宇宙線の加速現場の最有力候補である。これらのガンマ線観測を行い、他波長観測と比較することは、粒子の加速機構を解明するための最も確実な手段である。

1.1 GLAST 計画

次世代ガンマ線衛星 GLAST (Gamma-ray Large Area Space Telescope)は、2005 年に打ち上げ予定の、アメリカ、日本、イギリス、フランス、イタリアの共同プロジェクトであり、GLAST に搭載されるシリコンストリップ検出器は、広島大学が中心となり開発している。GLAST は、10 MeV から数 100 GeV 以上にわたる広いエネルギーバンドでの天体観測を目的とする高エネルギーガンマ線衛星であり、入射ガンマ線の到来方向とエネルギーを同時に決定することができる電子陽電子対生成型ガンマ線検出器をそなえる。検出器には、ガンマ線天文学において優れた実績をあげた CGRO 衛星(1991~2000)に搭載されている EGRET のデザインをベースにして、高エネルギー素粒子実験分野で開発され、位置分解能に優れたシリコンストリップ検出器を使用している。その結果、これまでのガンマ線天体観測ミッションに比べて、視野の広さ、検出感度、空間分解能が同時に飛躍的に向上するために、観測天体数の急激な増加が予想される。実際 EGRET と比べ観測可能なエネルギー範囲、空間分解能などが大幅に向上しており(図 1.1)、長期的なサーベイで得られる検出感度は、EGRET の 50-100 倍に達すると予測される。また、GLAST は広視野(一度に全天の 20%、1 日に 80%程度)であるために、任意の天体を継続的あるいは断続的に視野内に捉えつづけることが容易で、長期にわたる激しい時間変動を伴う天体のモニターも自動的に行うことができる。このことにより、ガンマ線天文学はその様相を一変させ、系統的で定量的なガンマ線観測に基づいた、宇宙高エネルギー現象のより精密で、統一的な描像を目指すことができる。

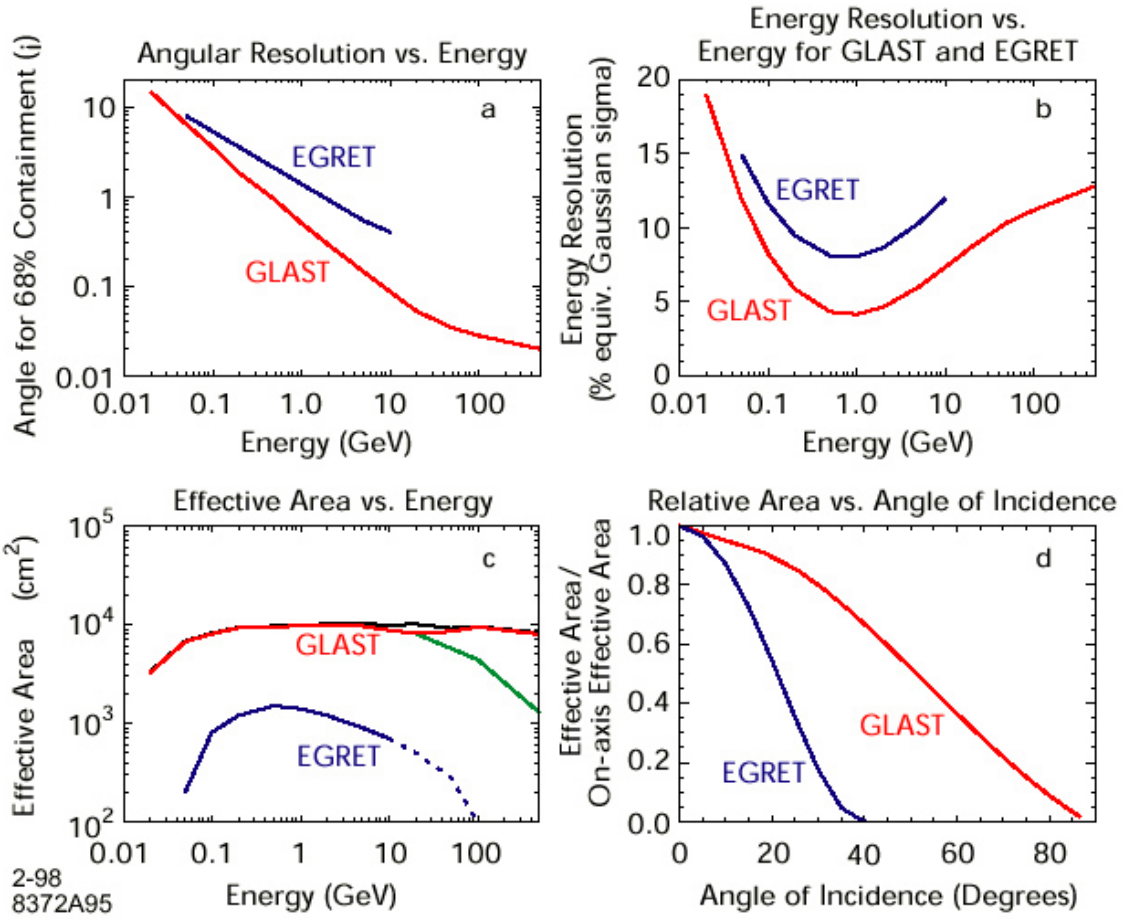

図 1.1: GLAST と EGRET の性能比較。角分解能 a)、エネルギー分解能 b)、有効面積 c)、有効面積の入射角依存性 d)。いずれの性能も飛躍的に向上していることがわかる。

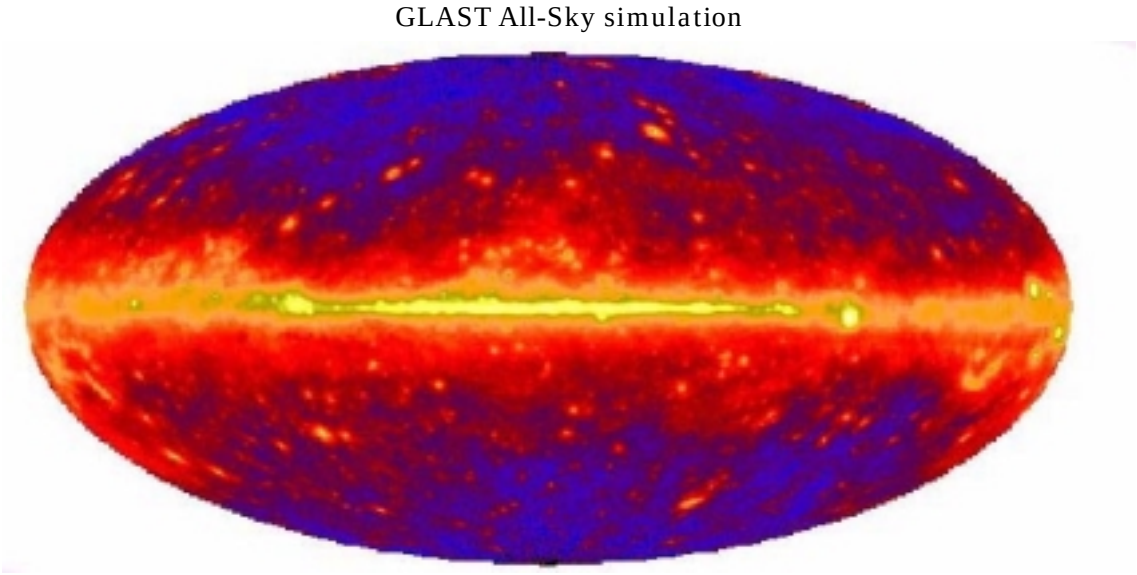

図 1.2: GLAST で見た銀河面からの広がった放射。

1.2 GLAST 加速器ビーム実験

1999~2000 年にかけて SLAC (Stanford Linear Accelerator Center)において GLAST のための加速器ビーム実験が行われた。この実験は、GLAST 衛星のシステムの動作確認、キャリブレーションデータの取得、モンテカルロシミュレーションの確認のために行われた。この実験のために、検出器には GLAST で使用予定の検出器(図 1.3)が用いられた。実際の GLAST では、この検出器を 4×4 のアレイ状に配置することによって、全検出器を構成する。このようなモジュール化は、トリガーシステムへの負担を減少し、直接因果関係のないイベントによる不感時間を減少する。検出器は、バックグランドを除去するアンチコインシデンスディテクター(ACD)、入射ガンマ線の到来方向を測定するシリコントラッカー(TKR)、入射ガンマ線のエネルギーを測定するカロリメーター(CAL)で構成されている。トラッカーは、入射ガンマ線に電子陽電子対生成を起こさせるための鉛フォルムと、対生成された電子、陽電子の飛跡を捉えるシリコンマイクロストリップ検出器からなる。検出器は、16 の層からなり隣り合うストリップが直交するように配置されている。また、トラッカーで飛跡を記録された電子と陽電子は、トラッカーの下部の CsI で構成されたカロリメーターに入射してシャワーを起こし、エネルギーが記録される。カロリメーターは 8 つの層からなり、90 度ずつ回転して配置される。ACD は、プラスチックシンチレーターで主検出器の上面と側面を覆う。

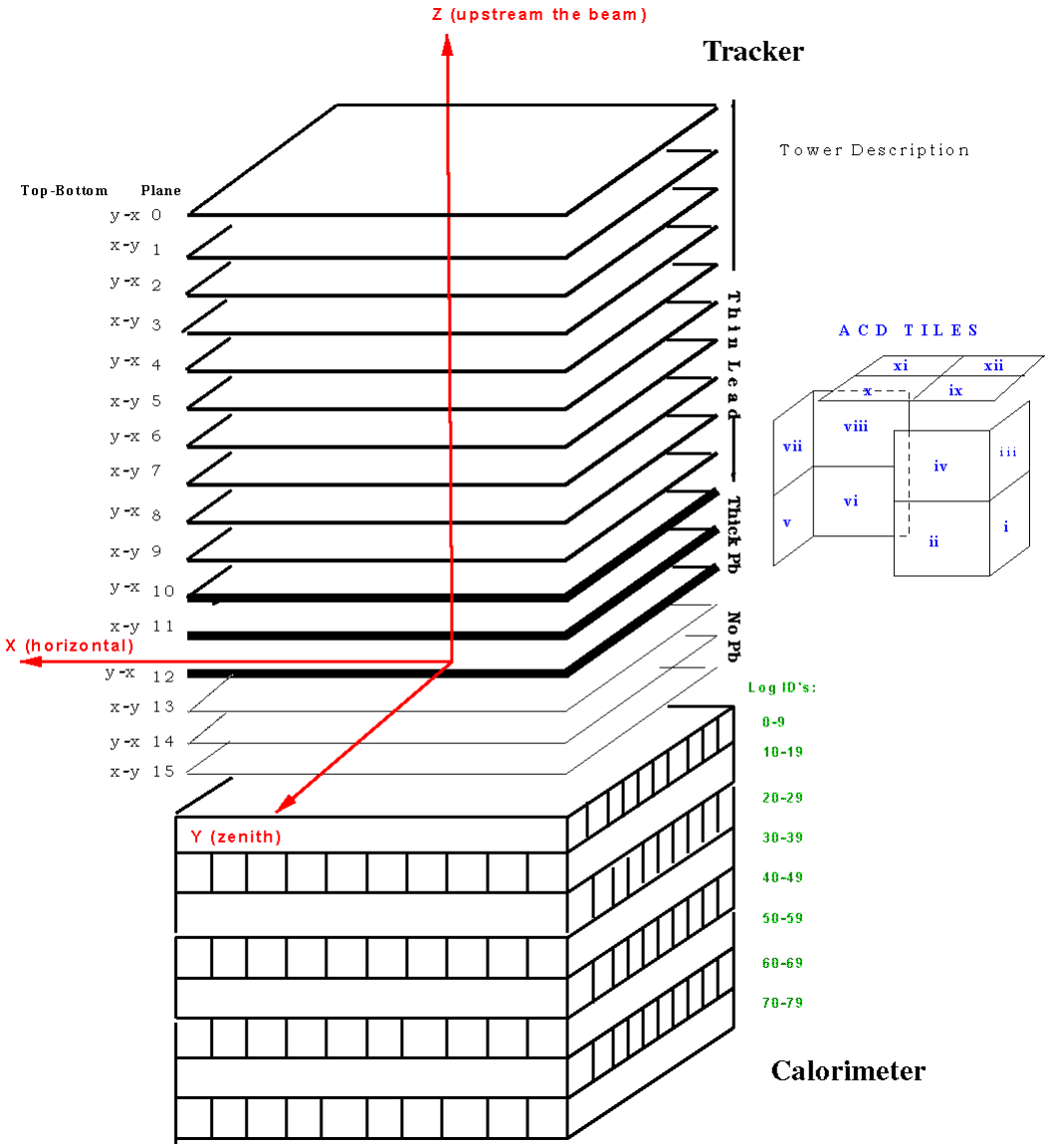

図 1.3: GLAST 加速器ビーム試験に使用された検出器。

第 2 章 GLAST ガンマ線検出器シミュレーターとデータ構造

検出器の性能を評価し、サイズや配置を最適化するためには、検出器に入射した高エネルギー粒子が引きおこす相互作用のシミュレーションを行う必要がある。さらに、検出器シミュレーターは、軌道上での検出器の動作を理解し、応答関数を作成する際にも重要な役割をもち、実際の観測をシミュレーションして観測データの確認を行うために欠かせないものである。GLAST のシミュレーター開発は、これまで EGS４や GISMO という検出器シミュレーションソフトを用いて開発されており、GLASTSIM と呼ばれている。しかし、シミュレーションソフトのサポート体制が弱くなってきたことや、複雑なジオメトリの再現が困難なことなどの理由で、オブジェクト指向の検出器シミュレーションのツールキットである GEANT4 が用いられることになった。開発は広島大学を中心に始まり、1999~2000 年に SLAC (Stanford Linear Accelerator Center)で行われた加速器ビーム実験の結果をもとに、シミュレーターの評価が行われている。

2.1 ROOT　　http://root.cern.ch/

GLAST では、全天サーベイ観測が中心になるため観測データは膨大なものとなる。そして、観測データからその科学的成果を効率よく引き出すことが科学的に重要である。これまで宇宙物理では FITS フォーマットというものが一般に使用され、また高エネルギー物理では HBOOK フォーマットが主流であった。これらのデータフォーマットを扱うソフトとして長年の実験から、FTOOLS、PAW、PIAF、DIS45 などの解析ソフトが発達しており、FORTRAN / C で構築されている。さらに、これらは、十分に使い込まれ、機能も豊富であるが、しかし、今後実験がより大規模になり、データ構造が複雑化し、データ解析もさらに大規模になることを予想すると、ソフトウェアの拡張性が重要になる。そこで、GLAST では近年高エネルギー物理の分野で用いられるようになった ROOT フォーマットが使用されることになった。このフォーマットは、ファイルシステムのディレクトリーのように Tree 階層構造をもち、ROOT というソフトウェアパッケージで処理できる。ROOT は、CERN (欧州原子核研究機構)のチームにより、オブジェクト指向技術に基づき開発が進められ、対話的データ処理、オンラインのデータ収集、大規模なバッチデータ処理など、あらゆる実験の局面においてデータを処理することが可能である。現在 2000 年 12 月にリリースされたバージョン 3.00 が最新であるが、本研究ではバージョン 2.00 を使用した。

ROOT の枠組みは、350 以上もの C++言語のクラスからなっており、11 のクラスカテゴ

リーに分かれている。

1. The basic ROOT classes
2. The container classes
3. The histograms and Minimization classes
4. The tree and ntuple classes
5. The 2D graphics classes
6. The 3D graphics classes
7. The MOTIF graphical user interface classes
8. The interactive interface classes and C++ interpreter
9. The operating system interface
10. The networking classes
11. The documentation classes

2.2 GEANT4　　http://wwwinfo.cern.ch/asd/geant4/

粒子と物質とのシミュレーションを行うことは、高エネルギー・原子核実験などの分野において不可欠な要素である。現在までは、EGS4 / GISMO や CERN を中心とするメンバーによって開発された GEANT3 が用いられてきた。これらは FORTRAN 言語で構築されており、多くの実験で標準の測定器シミュレーターとして世界各国の研究者に利用されてきた。しかし、最近の急速な計算機技術の発達に伴い、ソフトウェア開発技術の分野では、近年注目を集めているオブジェクト指向技術がさまざまな分野でますます定着しつつある。これは高エネルギー実験の分野でも例外ではなく、オブジェクト指向技術を用い、C++言語で書かれた相互作用のモンテカルロシミュレーターが開発された。GEANT4 (geometry and tracking)である。GEANT4 は、GEANT3 での経験を生かし、国際協力のもとで根本的に新しく開発されたものであり、CERN の主な LHC 実験や、SLAC の BABAR 実験で、本格的な使用が開始されるなど、現在標準となりつつある。GLAST においてもこの GEANT4 を用いて検出器シミュレーターを開発することとなった。本研究では、GEANT4.2 で開発されたシミュレーターを使用しているが、現在までに GEANT4.3 がリリースされている。

2.3 RECONSTRUCTION

GLAST では、入射粒子がトラッカー、カロリメーターで物質と相互作用を起こし、その情報がデータに記録される。そして、トラッカーでの各シリコンストリップ検出器の層にヒットした位置(チャンネル)と、カロリメーターの各 CsI でデポジットされたエネルギーの情報を元に、入射粒子の到来方向、粒子の種類、エネルギーを決定する。これが reconstruction と呼ばれるプロセスで、実際の観測で天体からのガンマ線情報を得るために不可欠である。

GLAST では、トラッカーでの電子陽電子の飛跡の情報から、Kalman Filter を用いて入射ガンマ線の到来方向を決定する。Kalman Filter(参考文献[4]、[5])は、Kalman によって提唱され、Fr wirth が高エネルギー物理学に応用した理論であり、フィルターとスムーサーの 2 ステップで到来方向を決定している。

GLAST 検出器の優れた性能のひとつに、荷電粒子バックグランドの除去能力が挙げられる。銀河系外からの微弱な宇宙ガンマ線バックグランド放射を検出するためには、宇宙線に含まれる高エネルギーの陽子や中性子の入射を識別しなければならないが、これらの粒子の飛跡とエネルギー測定のパターンは、ガンマ線のパターンとは容易に分別が可能である(図 2.1、図 2.2)。

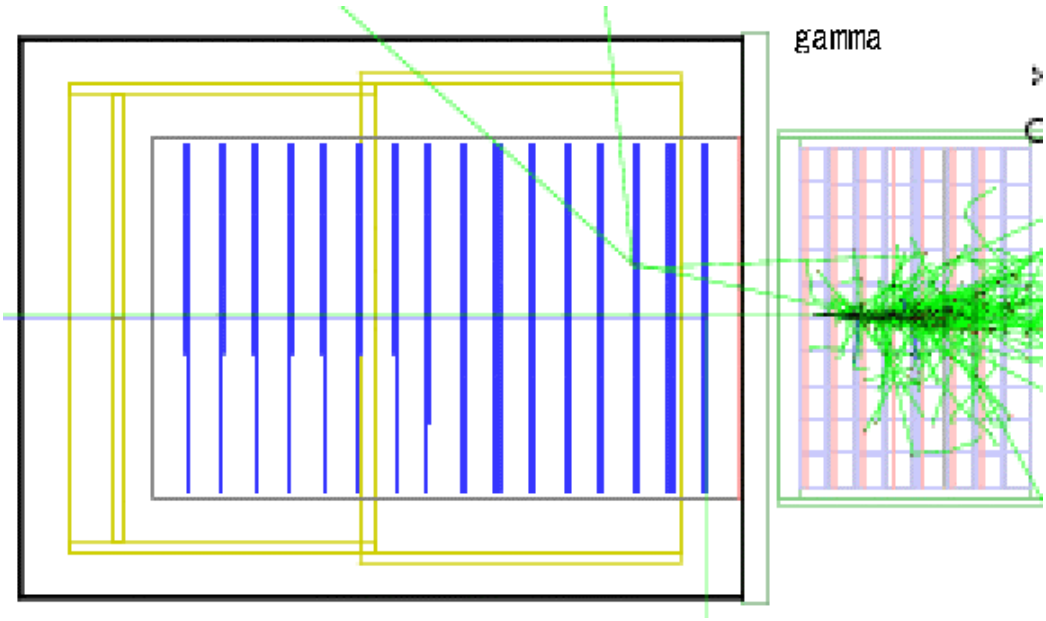

図 2.1: 1 GeV のガンマ線のシミュレーターでの反応。

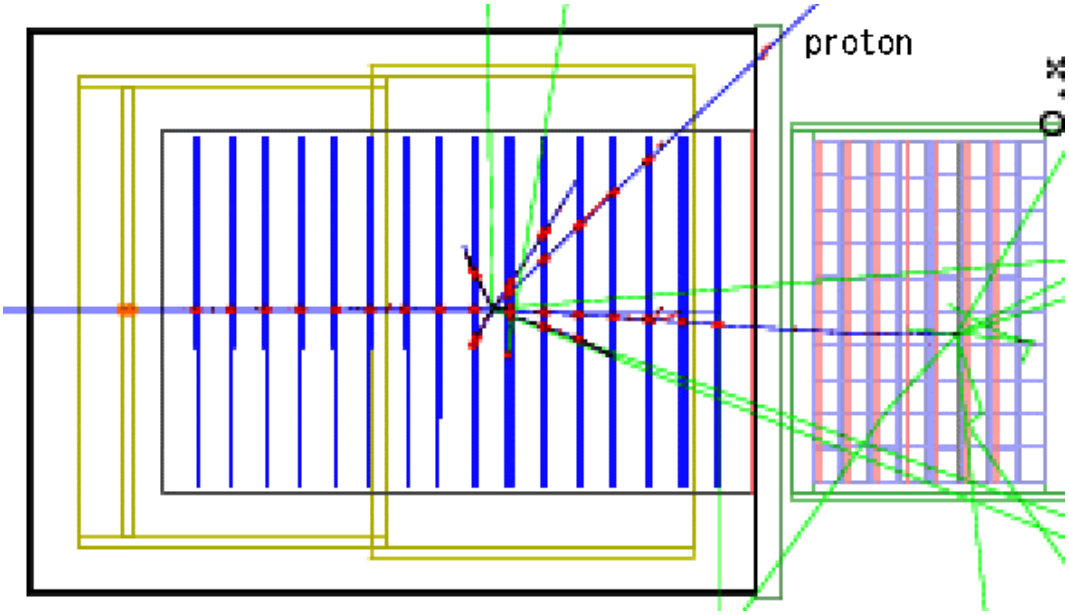

図 2.2: 1 GeV の陽子のシミュレーターでの反応。

GLAST の reconstruction プログラムは、UCSC (Universal of California Santa Cruz) の Jose A Hernando と Bill Atwood らによって開発されており、tb_recon (Centella)と呼ばれている。

2.4 GEANT4 シミュレーターの開発と本論文の目的

GEANT4 による GLAST 検出器のシミュレーター開発は、広島大学を中心に行われており、粒子発生部、検出器コンポーネントを模擬するジオメトリ部、入射ガンマ線が反応してできた粒子と検出器の相互作用を記述する物理プロセス部からなる。

本研究では、新しく開発された GEANT4 シミュレーターの出力を reconstruction するための必要なソフトを開発し、さらに reconstruction の結果得られたデータを、GLAST 検出器ビーム実験のデータと比較して、シミュレーターの性能評価を行う。

第 3 章 シミュレーションのデータ出力部の開発

GEANT4 によるシミュレーターの出力はテキスト形式である。GLAST は、ROOT をデータフォーマットと決めており、reconstruction プログラムである tb_recon (Centella)は ROOT フォーマットを要求している。そこで、シミュレーターのデータを reconstruction するためには、GEANT4 の出力を ROOT フォーマットに変換することが不可欠である。

3.1 GLAST データ構造

GLAST のデータは ROOT フォーマットをとる。GLAST 計画の際に行われた、ビーム実験では、ROOT の階層構造を利用し、トラッカーや、カロリメーターなどといった、データの種類によって効率よくデータを処理している。図 3.1 に各データ処理の段階でのデータフォーマットの流れを示す。ビーム実験と GLASTSIM のデータは IRF 形式で提供され、ROOT への変換プログラムは確立しているが、GEANT4 の出力データはテキスト形式であり、いまだに ROOT への変換プログラムは確立されていない。GEANT4 の出力は、テキストフォーマットから ROOT フォーマットへの変換を経て、tb_recon プログラムで reconstruction されることになり、その変換プログラムを用意しなければならない。このため、GLAST のデータ構造をさまざまなドキュメント(参考文献、Web ページ)やソースファイルを用いて調べた。

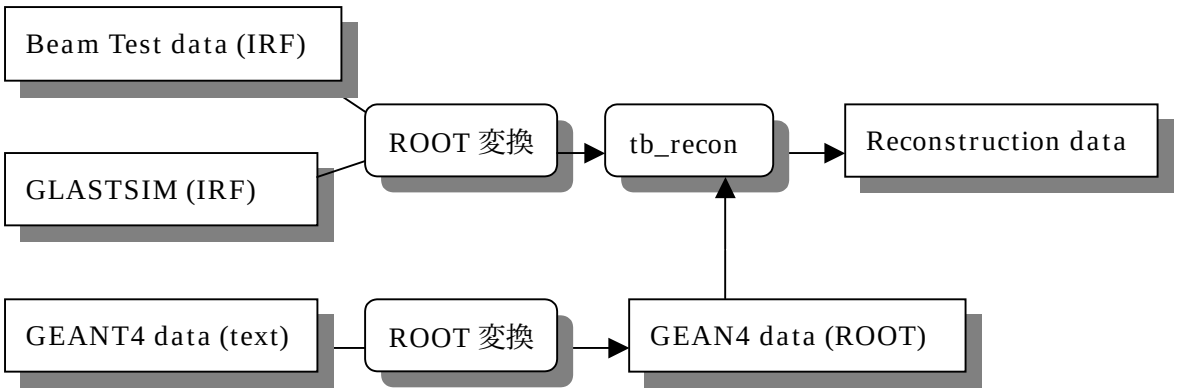

図 3.1: GLAST データの流れ。

ビーム実験のデータは、ACDTile をはじめとする 10 の最上流クラスからなっている(図 3.2)。そのうち EVENT、MCPart、Run、Cluster、Track の 4 つクラスを除きデータ構造は確立している。getACD()によってアンチコインシデンスディテクターの情報(ACDTile クラス)、getCAL()によりカロリメーターの情報(CalHit クラス)、getL1T()によりトリガー情報(L1T クラス)、getTKR()によりトラッカー情報(TkrLayer クラス)が得られる。今回、ROOT フォーマットへの変換は、GEANT4 のトラッカーの出力を用いて TkrLayer クラスに合わせて変換している。

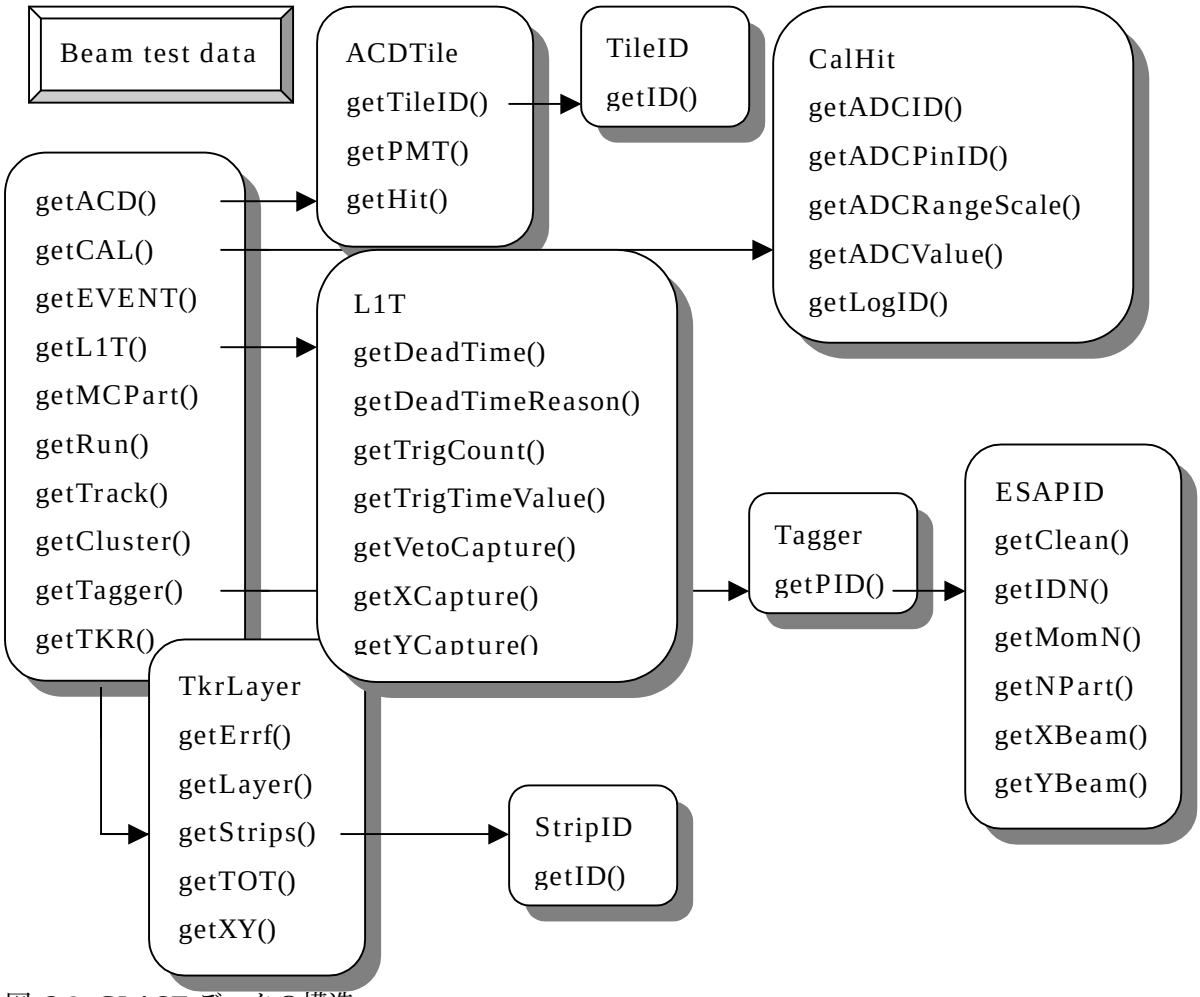

図 3.2: GLAST データの構造。

さらにこれらのデータを用いて、tb_recon (Centella)で reconstruction を行って得られるデータはやはり階層構造をとる。reconstruction で得られるデータは、これまで公式のデータ構造を示すドキュメントがなかったこともあり、tb_recon プログラムのソースプログラムから解読した。図 3.3 である。最上流の CalRecon、TkrRecon の 2 つのクラスからなり、getCalRecon()でカロリメーターの reconstruction の情報を、getTkrRecon()でトラッカーの reconstruction 情報を得ることができる。

ビーム実験のデータ、reconstruction 後のデータのどちらも get~()を用いてデータの値を取り出すことができ、あるいは、ポインタを返す関数が用意されている。

例として、reconstruction 後のデータは次のようにすることでアクセスが可能である。

```
TkrLocator *locator = (TkrLocator *)(track->getLocator()->At(iLocator);
```

でポインタを指定し、

```
slx = locator ->getSlopeX();
```

とすると、slx に SlopeX のデータが詰め込まれる。

本研究では、さらにこれら ROOT フォーマットへ変換し reconstruction を行ったデータを解析するためのプログラムの作成(巻末の付録を参照)も行った。

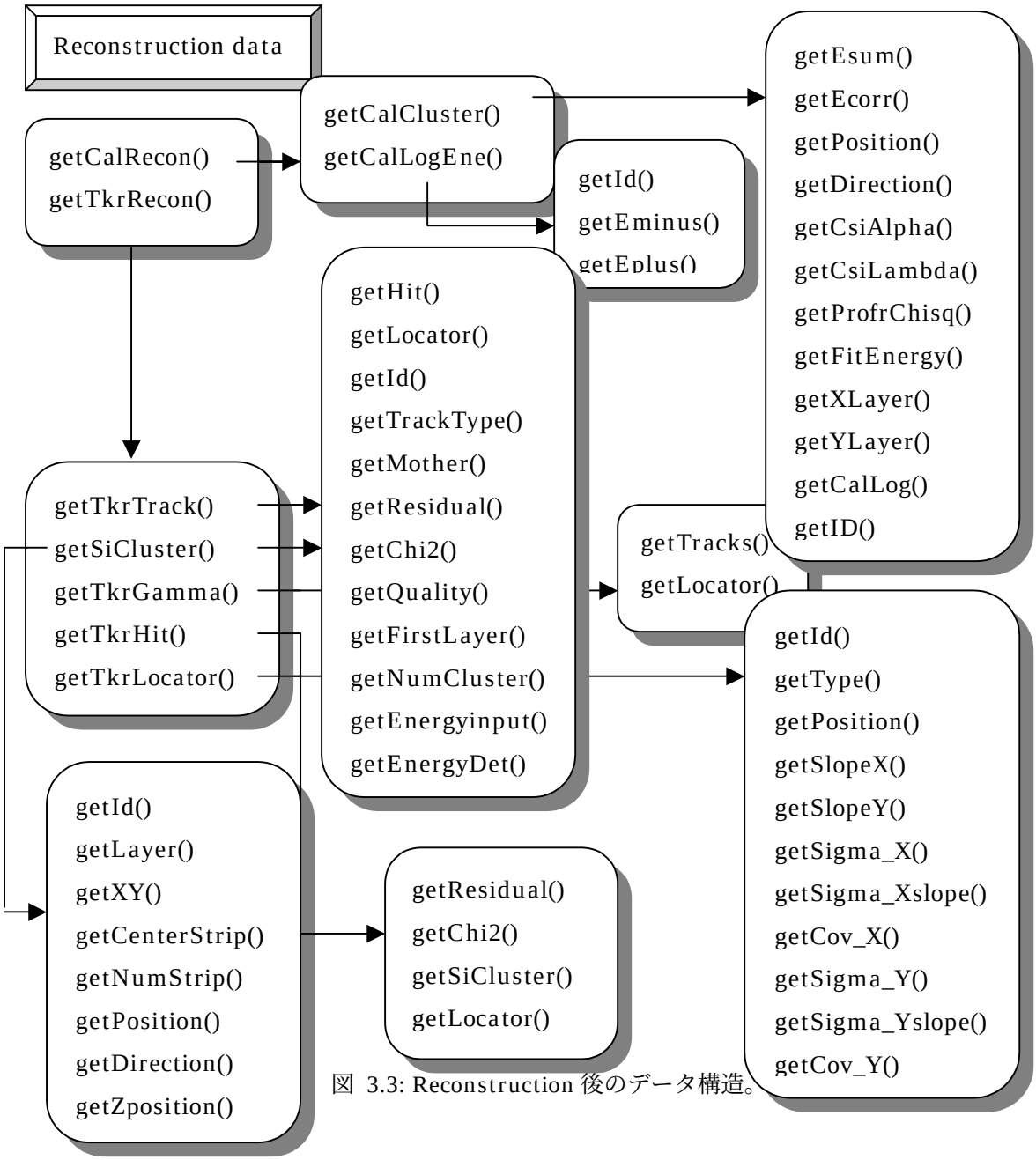

図 3.3: Reconstruction 後のデータ構造。

3.2 GEANT4 の出力から ROOT フォーマットへの変換

GEANT4 の出力データを tb_recon (Centella)プログラムに入力するために、出力データを ROOT フォーマットへ変換しなければならない。シミュレーションの出力データは、アンチコインシデンスディテクター、トラッカー、カロリメーターについてそれぞれテキストフォーマットでデータが記録され、さらに GEANT4 の機能である verbose と呼ばれる全相互作用を記録するオプションもある。シミュレーションのトラッカー出力データの例として図 3.4 を示す。tb_recon (Centella)で、角分解能を求めるために必要なデータは、粒子の飛跡の情報であり、ヒットがあったストリップ検出器のチャンネルリストである。

GEANT4 の出力を ROOT に変換するには 2 段階のステップがある。GEANT4 の出力は、連続した値をとる座標とエネルギーであるが、実際に得られるデータは飛び飛びの値を取るシリコンストリップチャンネル(座標に相当する)と、ADC 値(エネルギーに相当する)であり、それぞれ変換する必要がある。これは一般に DIGI 変換と呼ばれる。

まず、図 3.4 のシリコンストリップ検出器に粒子が入射した位置(XYZi)と、出た位置(XYZo)からヒットしたチャンネルへ変換した(図 3.5)。その際、シリコンストリップのピッチは 0.194 mm、検出器と検出器とのギャップは 0.2 mm、検出器の端の不感領域は 0.96mm とした。トラッカーはシリコン検出器が 90 度ずつ回転して配置されておりそれぞれ X layer と Y layer に区別されている。粒子がヒットした位置が X layer ならば、XYZi と XYZo の X 座標の値から、Y layer ならば Y 座標の値から、それぞれ粒子が入射し、ヒットしたと思われるストリップチャンネル A と、検出器から出ていったストリップチャンネル B を求める。次に A、B のチャンネルが異なった場合は、A、B 間のすべてのストリップにもヒットしたとみなし、チャンネルを加える。このようにヒットしたストリップ検出器のチャンネルリストを作り、図 3.5 のようにテキストファイルに出力した。

```
EventNo= 1
 NoTrackerHit= 44
  ID= 27  ParSpc= e+  TrkLen= 0.397142  XYZi= -0.015036  0.00607509  419.193
   XYZo= -0.0543465  0.0157908  418.798   DepE= 0.109876  ParE= 44.7755
  ID= 26  ParSpc= e+  TrkLen= 0.396638  XYZi= -0.263236  0.12801  416.137
   XYZo= -0.294256  0.144675  415.743   DepE= 0.13645  ParE= 44.6656
```

図 3.4: シミュレーションの出力データ。GEANT4 でシミュレーションされた粒子のステップ毎に情報を記録している。ID はレイヤーの ID。ParSpc は粒子の種類。TrkLen は粒子の走った飛跡。XYZi は粒子の入射または生成位置(ｘ軸、ｙ軸、ｚ軸、単位は mm)。XYZo は粒子のレイヤーから出た、または消滅位置(x 軸、y 軸、ｚ軸、単位は mm)。DepE はエネルギーデポジット。ParE は入射位置での全エネルギーを表す。

```
EventNo= 1
 0 2 1408 1409
 1 5 1476 1477 1478 1479 1480
 2 15 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597
 3 10 1366 1367 1368 1369 1370 1371 1372 1373 1374 1375
 4 7 632 633 634 635 636 637 638
 5 3 447 448 449
 6 2 494 495
```

図 3.5: GEANT4 出力データを DIGI 変換で変換した例。順にレイヤーナンバー、ヒットしたストリップ検出器のチャンネル数とチャンネルリスト。

次に DIGI 変換を行い得られた図 3.5 の出力を ROOT フォーマット(図 3.2)に変換する。ROOT に変換する手続きは次のようになる。

```
TkrLayer *xLayer;        ***(1)
xLayer->setXY(TkrLayer::X);        ***(2)
xLayer->setLayer(lay);        ***(3)
StripID *strip = new StripID();        ***(4)
Strip->setStrip(channel);        ***(5)
xLayer->getStrips()->Add(strip);        ***(6)
event->getTKR()->Add(xLayer);        ***(7)
```

(1) layer 情報をつめる TkrKayer クラスのポインタを指定する。
(2) x layer であることを指定。
(3) シリコンストリップの layer 番号(lay)をセットする。
(4) ストリップオブジェクトの作成。
(5) シリコンストリップでヒットしたチャンネル(channel)の番号をセットする。
(6) (5)でセットしたストリップの情報を TkrLayer クラスにつめる。
(7) TkrLayer クラスの情報を 1 つのイベントとして登録。
(1)から(7)は 1 イベントあたり 32 回(32Layer)行われる。また、(4)から(6)はヒットしたチャンネルの数だけ各 Layer ごとに行われる。詳しいプログラムは巻末の付録に譲る。

第 4 章 シミュレーションデータの RECONSTRUCTION

4.1 入射ガンマ線の到来方向の決定と評価

本節では、SLAC での GLAST 加速器ビーム実験と、GEANT4 シミュレーターで得られた入射ガンマ線の位置決定精度を比較してシミュレーターの評価を行う。そのために、GLAST の目指す 4 桁にわたるエネルギー領域でガンマ線をシミュレーターに入射し位置決定精度を求めた。入射エネルギーは、50 MeV から 500 GeV までの 9 つのエネルギー領域で、0 ° (検出器の真上)から 5000 発ずつガンマ線を入射し、トラッカーでのテキスト出力データを得た。次に 3 章で述べた変換プログラムを用いて ROOT フォーマットに変換し、tb_recon (Centella)プログラムで reconstruction し解析した。

まず、50 MeV のガンマ線を入射し、シミュレーションを行い、reconstruction した結果得られる入射方向と実際の入射方向との角度のずれをイベント毎に求め、図 4.1 のようにヒストグラムにした。また、ガンマ線の入射エネルギーが 10 GeV と 500 GeV の場合については、入射方向とのずれの x と y の 2 乗平均の分布のみ、図 4.2 と図 4.3 に示す。50 MeV のときに比べ 10 GeV、500 GeV のほうが到来方向の決定精度はよくなっているのがわかる。これは対生成で生じた電子陽電子が、高エネルギー領域で多重散乱をうけにくくなるからである。荷電粒子が多重散乱を受け方向を変える角度は、

$$\vartheta_0 = \frac{13.6MeV}{\beta cp} z \sqrt{\frac{L}{L_r}\left[1 - 0.038 \ln\left(\frac{L}{L_r}\right)\right]} \qquad \text{(rad)}$$

で与えられることがわかっている。ここで、β, z, p はそれぞれ入射粒子の速度を高速 c で割ったもの、電荷、運動量(単位は MeV/c)であり、$L/\ L_r$ は物質の厚さを radiation length で割った量である。この式からわかるように、粒子のエネルギーが高いほど散乱される角度は小さくなるので位置決定精度はよくなる。

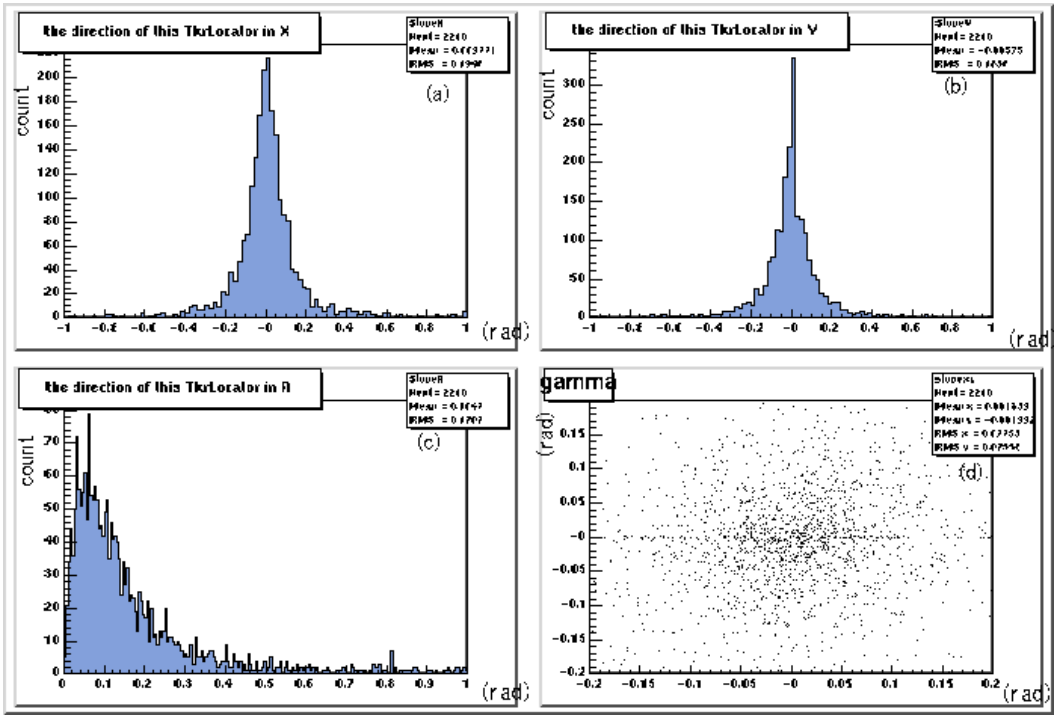

図 4.1: 50 MeV のガンマ線を入射し reconstruction して、得られた方向と入射方向とのずれ。(a)、(b)はそれぞれ x 軸、y 軸での到来方向からのずれ。(c)は x と y の 2 乗平均。単位は x 軸が radian、y 軸がカウントである。(d)は x 方向、y 方向のずれを 2 次元プロットしたもので単位は radian。

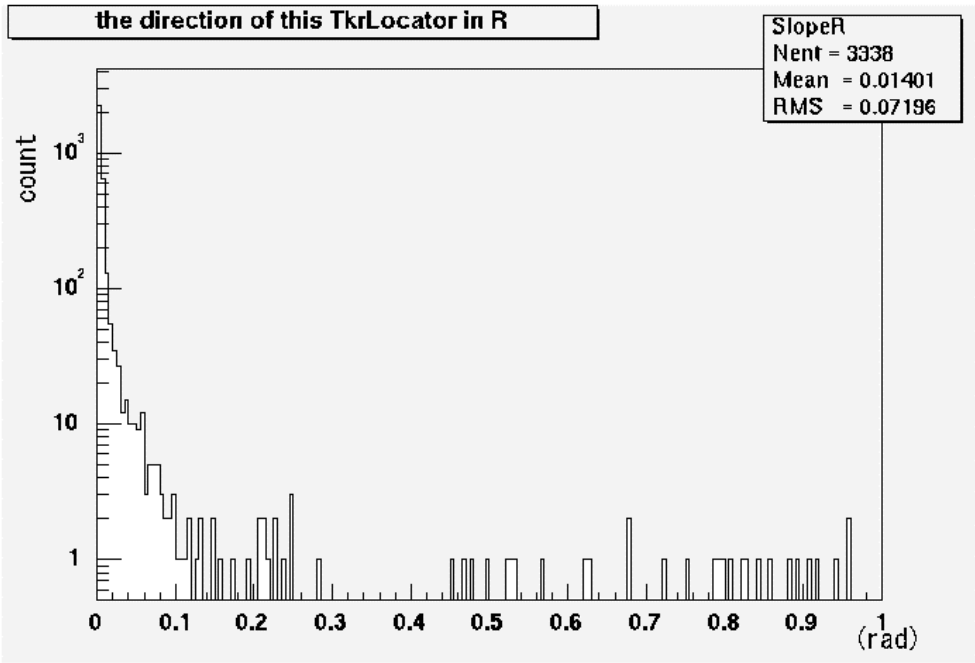

図 4.2: 10 GeV のガンマ線を入射し reconstruction して得られた方向と実際の入射方向のずれを x、y で 2 乗平均したもの。x 軸は radian、y 軸はカウントである。図 4.1(c)に相当する。

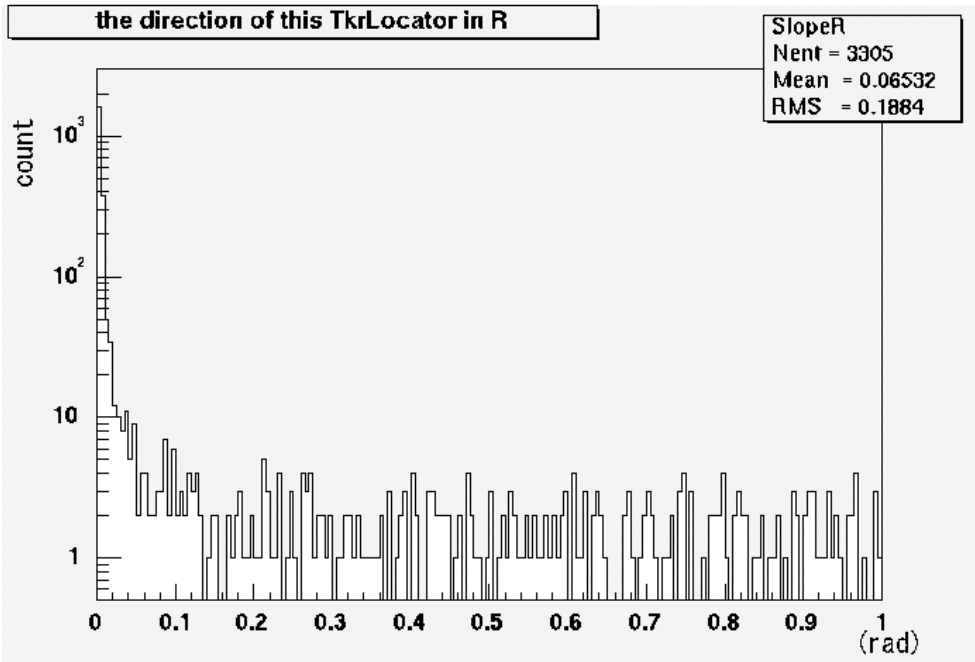

図 4.3: 図 4.2 と同様。ただし、入射ガンマ線のエネルギーは 500 GeV である。

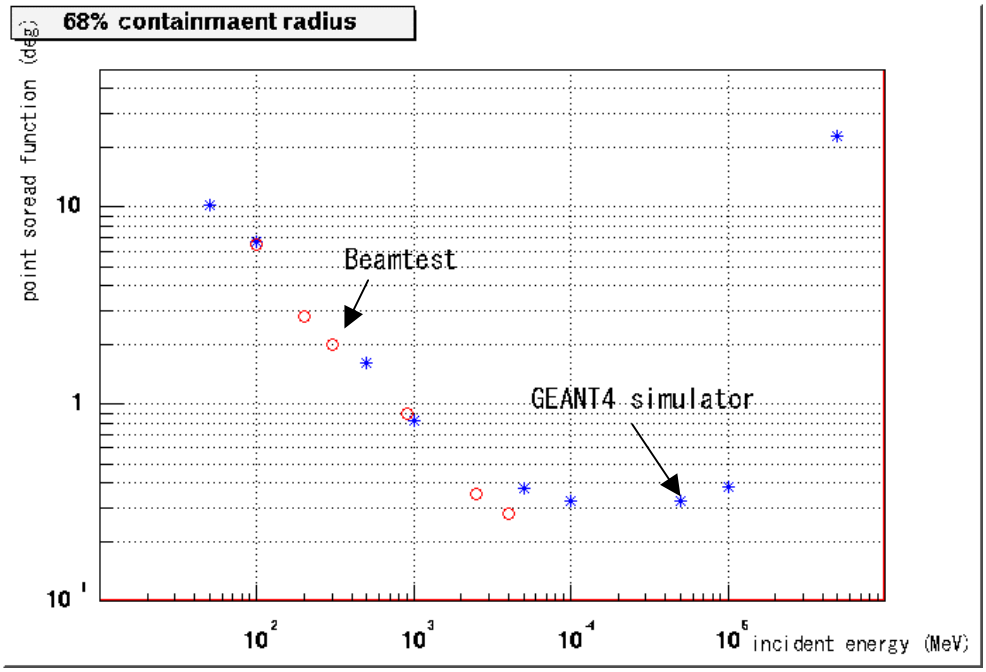

図 4.4: GLAST 加速器ビーム実験における PSF(赤)と GEANT4 シミュレーターによる PSF(青)。

ここで、reconstruction して得られる入射方向と実際の角度が、ある値 A 以下のイベントが全イベントに対して 68%であるとき A を PSF(Point Spread Function)という。PSF は位置決定精度を表す指標であり、値が小さいほど位置決定精度がよいことを意味する。

図 4.4 に、GLAST ビーム実験、及び GEANT4 シミュレーターにおける PSF(68%のイベントが入る)をのせる。100 MeV から 5 GeV の範囲では GEANT4 での PSF はビーム実験の PSF とほぼ一致しており、GEANT4 シミュレーターは現実を正しく再現しているといえる。しかし、GEANT4 シミュレーターの角分解能は、10 GeV から 50 GeV をピークに 100 GeV あたりで悪化し始め、500 GeV での角分解能は、23.1 ° と非常に悪かった。ビーム実験ではこれほど高エネルギー領域での実験は行われていないので比較は行えなかった。この原因はまだ明らかではないが、カロリメーターからのバックスプラッシュが効いているのかもしれない。カロリメーターに入射した粒子のエネルギーが高くなると反応してできる生成物が多いので、トラッカーに再度入射し、シリコンストリップでヒットする。Reconstruction ではこの情報も用いるので、単純な reconstruction では、高エネルギー領域で角分解能が悪くなることもありうる。バックスプラシュは下方からくるので、データから取り除くことが可能であり、今後の課題である。

PSF は、ガンマ線 1 つに対する位置決定精度を表すが、実際の観測では、同一方向の天体から多数のガンマ線が検出器に入射する。このとき、天体から入射したガンマ線の数を n とすると、$1/\sqrt{n}$ に比例して角分解能はよくなるので、実際の観測ではさらに天体の位置決定精度はあがると考えられる。

4.2 入射ガンマ線のエネルギーの決定方法

GLAST で用いられるカロリメーターは、1 つのブロックのサイズが、23.5 mm* 30.5 mm*310.5 mm であり、計 188 mm の厚さ(10.2 radiation length)の CsI 結晶で、1 つの layer に 10 本ずつ並べられ、総計 80 本のブロックで構成されている。また、それぞれの layer は 90 度ずつ回転して配置されている(図 1.3 参照)。GLAST のカロリメーターはこれまでの検出器と異なり、カロリメーターにデポジットしたエネルギーを CsI 結晶のブロックの両側で読み出すことにより、デポジットしたエネルギーの正確な値と粗い位置情報を同時に知ることができる。

入射エネルギーの決定は、カロリメーターにデポジットしたエネルギーによって行う。しかし、GLAST 検出器では、入射エネルギーのすべてがカロリメーターでデポジットされるとは限らない。入射エネルギーが低エネルギー領域である場合は、入射エネルギーに対して多くの割合がトラッカーでデポジットする(図 4.5)。また高エネルギー領域のガンマ線になると、カロリメーターの外に漏れ出す(リークする)エネルギーが増加する(図 4.6)。

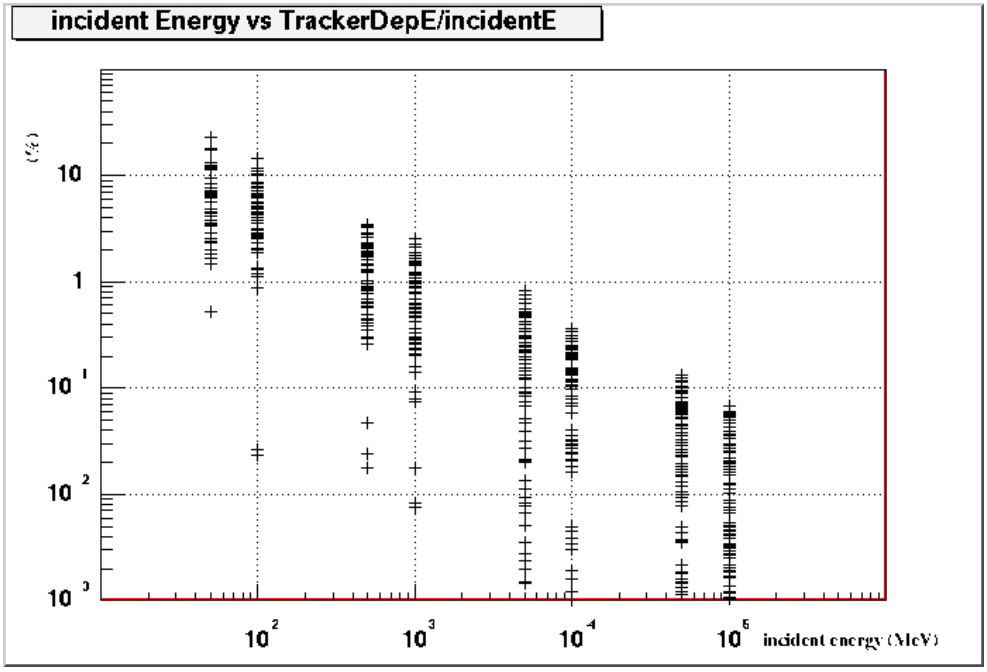

図 4.5 : 入射エネルギー(MeV)に対するトラッカーエネルギーデポジットの割合(%)。

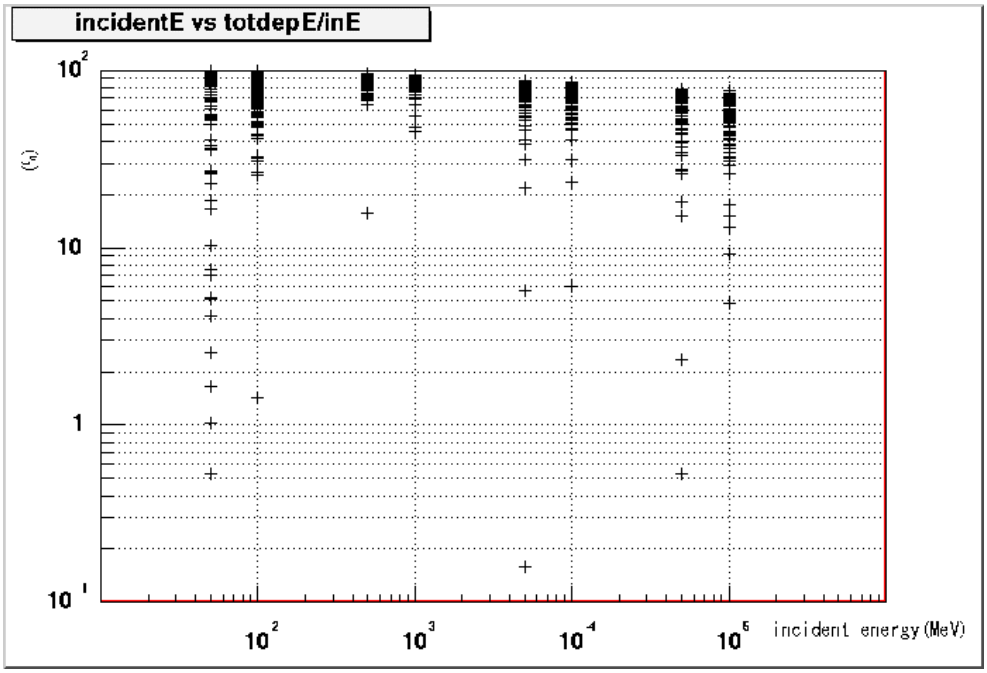

図 4.6 : 入射エネルギー(MeV)とカロリメーターでデポジットしたエネルギーの割合(%)。

入射エネルギーを reconstruction する方法には、エネルギー領域によって 3 種類の方法があることがわかっている。入射エネルギーを E として、

1. $E < 1GeV$ ・・・トラッカーでのヒット数を利用する。
2. $1GeV < E < 50GeV$ ・・・CsI カロリメーターの最下層でのエネルギーデポジットを利用する。
3. $E > 50GeV$ ・・・カロリメーター各 layer でのエネルギーデポジットの分布を利用する。

以上の 3 つであり、詳細は 4.2.1~4.2.3 で述べる。これら、3 つの方法の適用は図 4.7 に示すようにガンマ線の入射角度にも依存する。

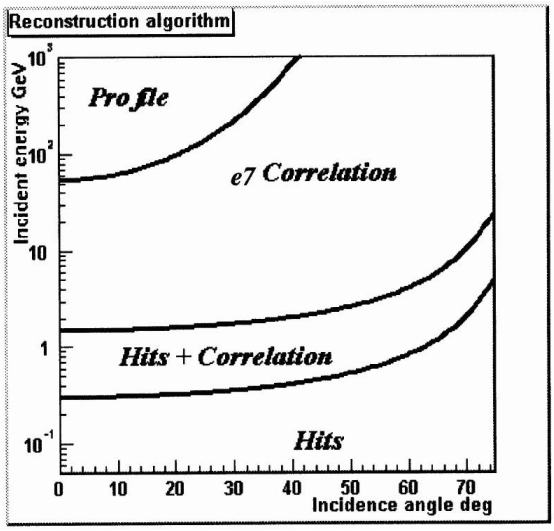

図 4.7 : エネルギーの reconstruction の各方法の適用範囲。横軸が入射角、縦軸が入射エネルギー。

本節ではカロリメーターの実際の reconstruction には間に合わなかったが、これらの方法でシミュレーションデータの reconstruction が可能かを調べて、シミュレーターの動作確認も行う。

本実験では、50 MeV から 100 GeV までの 8 つのエネルギー領域で 0 ° からガンマ線を入射してシミュレーションを行った。カロリメーターにデポジットしたエネルギー、リークしたエネルギーなどさまざまな量を柔軟に扱うために GEANT4 の機能の 1 つである verbose 出力を用いた。これは粒子の全相互作用を出力する GEANT4 のオプションであり、データが莫大になることから、ガンマ線はそれぞれのエネルギー領域で 100 発ずつ入射した。

4.2.1 入射エネルギーとトラッカーヒット数の関係

図 4.5 に示したように入射エネルギーの低エネルギー領域では、トラッカーでのデポジットエネルギーが入射エネルギーに対して寄与が大きくカロリメーターでのエネルギー決定に関して無視できない。これは、シリコン検出器 1 ヒットでデポジットされるエネルギーが電子陽電子のエネルギーにあまり依存しないため、入射ガンマ線のエネルギーが低いときは、トラッカーでデポジットするエネルギーの割合が増加してしまうからである。入射エネルギーが 1 GeV 以下の場合、入射エネルギーを決定するためには、トラッカーでデポジットしたエネルギーも補正する必要がある。

トラッカーでデポジットされたエネルギーは、トラッカーでのヒット数に依存すると期待される。図 4.8(a)は入射エネルギーに対してトラッカーでのヒット数を調べたものである。さらに図 4.8(b)は入射エネルギーとトラッカーでデポジットしたエネルギーの関係である。

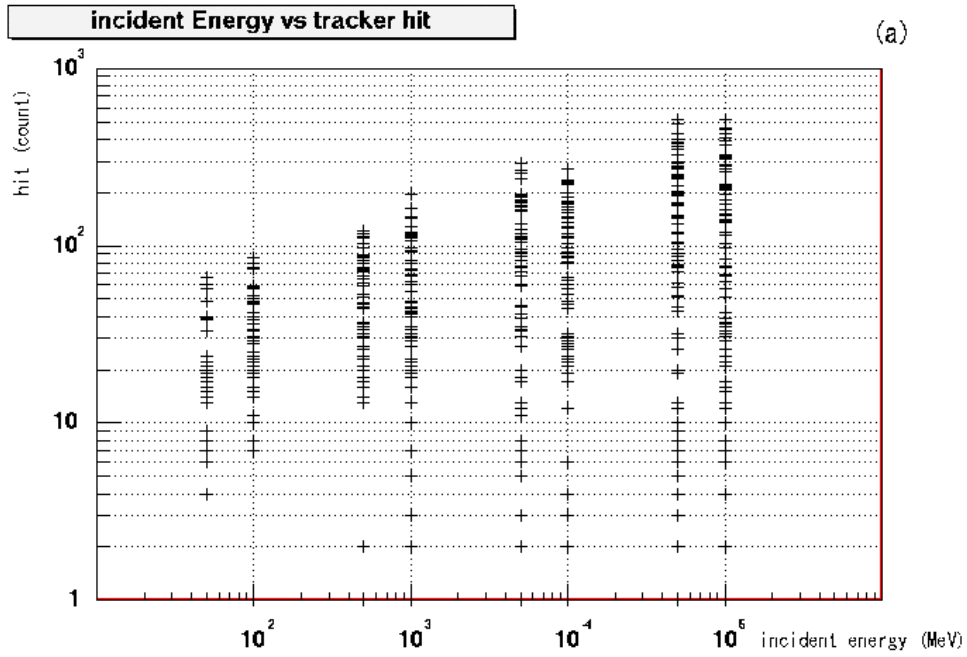

図 4.8(a)：入射エネルギー(MeV)に対するトラッカーでのヒット数。

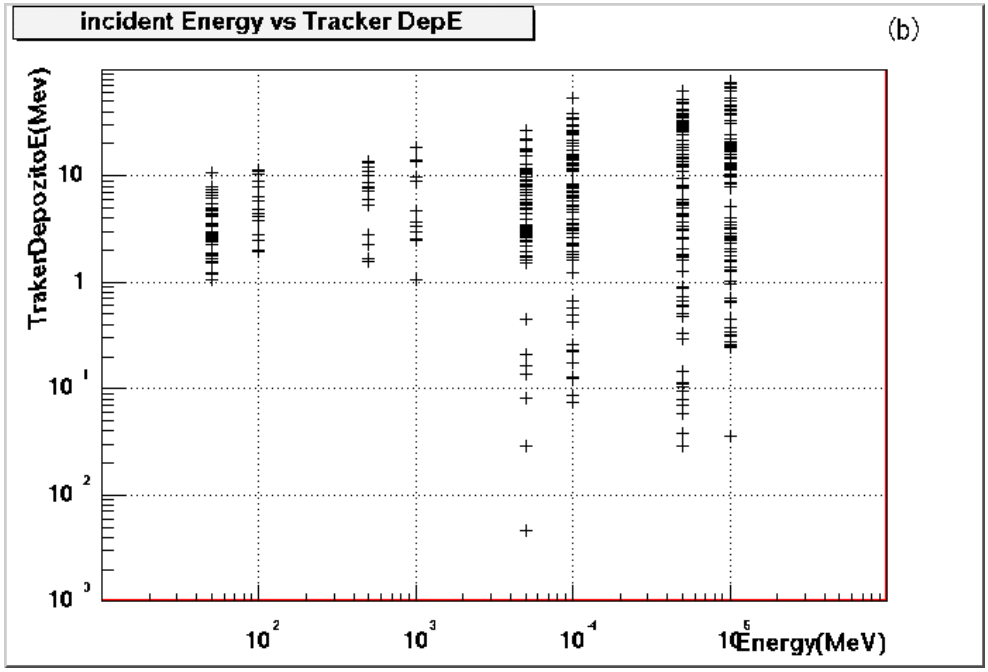

図 4.8(b)：入射エネルギーに対するトラッカーでのエネルギーデポジット。

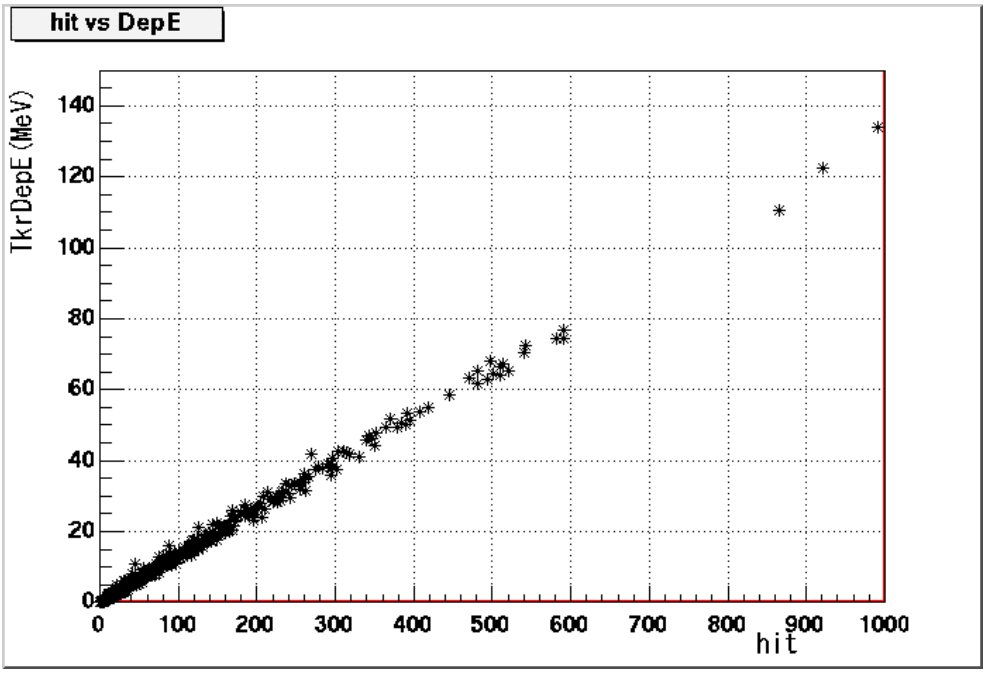

図 4.9 : トラッカーでのヒット数(横軸)とデポジットエネルギー(縦軸)の関係。

図 4.9 からトラッカーでのヒット数とエネルギーデポジットがほぼ比例していることがわかる。したがって、入射ガンマ線のエネルギーの決定の方法は次のように補正でき、決定される。

入射ガンマ線のエネルギーを E とすると、$E < 1GeV$ のときは、トラッカーでのエネルギーデポジットが無視できないので

$$E_{in} = E_{cal} + AN_{tkr} \qquad \cdots(1)$$

となる。ここで、E_{in} は入射エネルギー、E_{cal} はカロリメーターでデポジットしたエネルギー、A は 1 ヒットにつきトラッカーでデポジットするエネルギー、N_{tkr} はトラッカーでヒットした数である。

4.2.2 カロリメーター最下層でのエネルギーデポジット

図 4.5 から入射ガンマ線のエネルギーが 1 GeV 以上の場合、トラッカーでデポジットしたエネルギーは無視できる。しかし、だからといって、入射エネルギーが 1 GeV 以上のガンマ線のエネルギーは正確にもとまるとは限らない。図 4.6 から、入射ガンマ線の高エネルギー領域において、エネルギーデポジットと入射エネルギーが一致せず、ひどいときでは入射エネルギーの半分以下程度しかデポジットしないことがわかる。これは、電子陽電子

対生成によってできたシャワーがカロリメーターから漏れ出すときに持ち出すエネルギーが無視できなくなるためである。GLAST の目指す上限である数 100 GeV の高エネルギー領域では、すべてのエネルギーがカロリメーターでデポジットされるわけではなく多くのエネルギーがリークする。よって、入射ガンマ線のエネルギーを正確に求めるためには、カロリメーターからどれだけのエネルギーがリークしたかを何らかの方法を用いて知る必要がある。一般的に、入射ガンマ線のエネルギー E が、$1GeV < E < 50GeV$ の場合、カロリメーターの最下層の layer でのエネルギーデポジットがリークエネルギーとの間に相関があることが知られており、次のように reconstruction する。

$$E_{in} = E_{cal} + aE_{last} + b \qquad \cdots(2)$$

$$a = 1.111 + 0.557 \log\left(\frac{E_{cal}}{1GeV}\right)$$

$$b = 210 + 112\left[\log\left(\frac{E_{cal}}{10GeV}\right)\right]^2$$

E_{in} は入射エネルギー、E_{cal} はカロリメーターでデポジットしたトータルエネルギー、E_{last} は最下層でデポジットしたエネルギーである。

図 4.10 は入射エネルギーに対するカロリメーターでリークしたエネルギーの関係である。

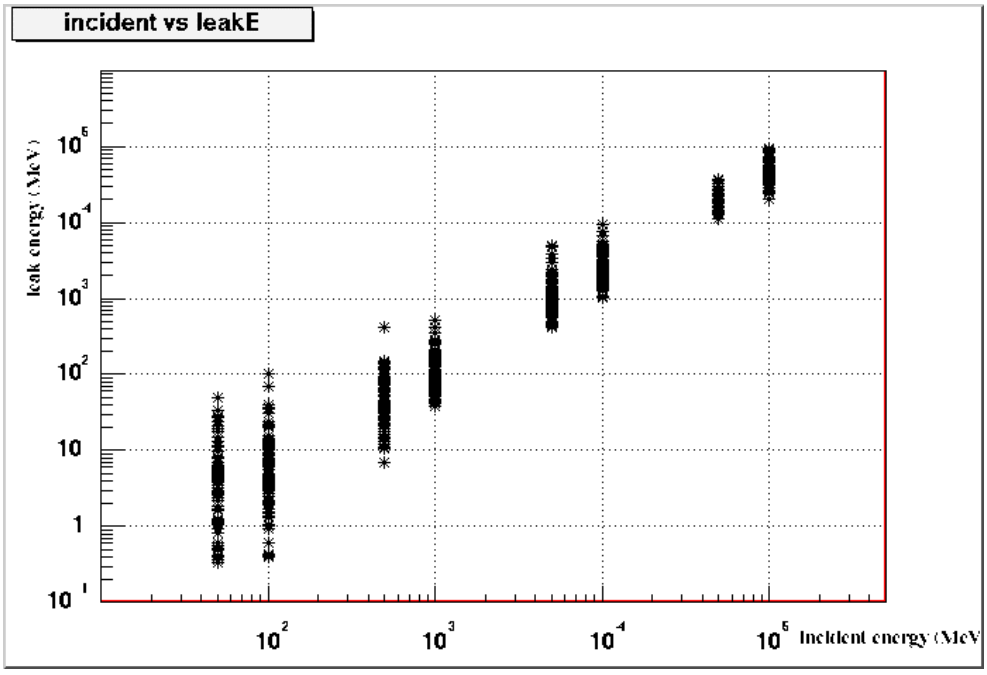

図 4.10 : 入射エネルギー(MeV)に対するリークしたエネルギー(MeV)の関係。

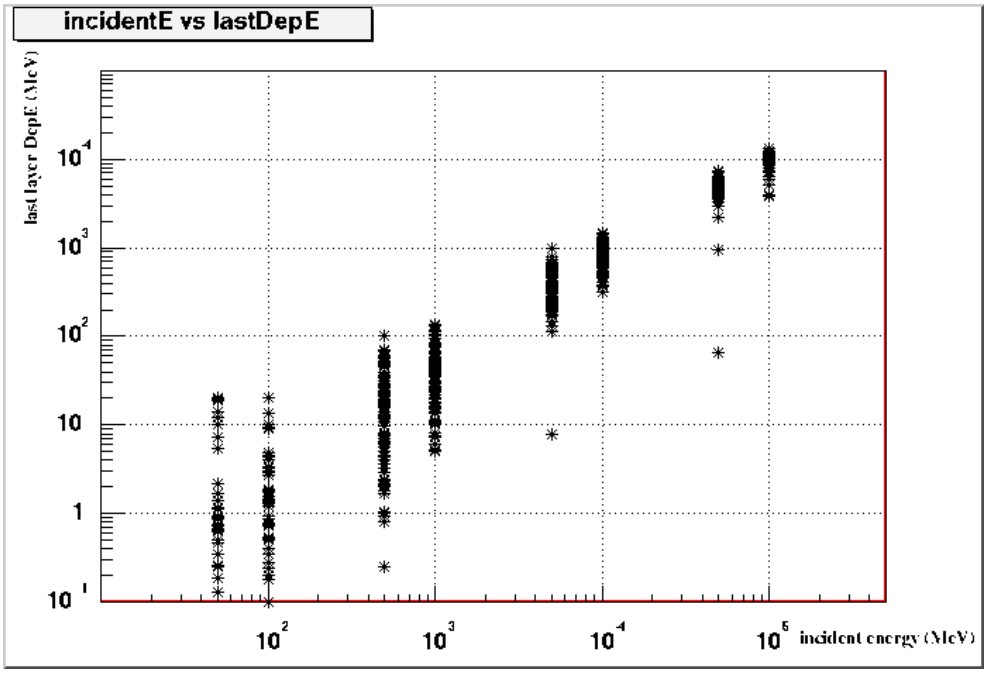

図 4.11:入射エネルギーに対するカロリメーター最下層でデポジットしたエネルギー。

さらに図 4.11 は入射エネルギーに対するカロリメーターの最下層でデポジットしたエネルギーの関係である。図 4.12 は、図 4.10、図 4.11 からカロリメーターの最下層にデポジットしたエネルギーとリークしたエネルギーの関係を見たものである。

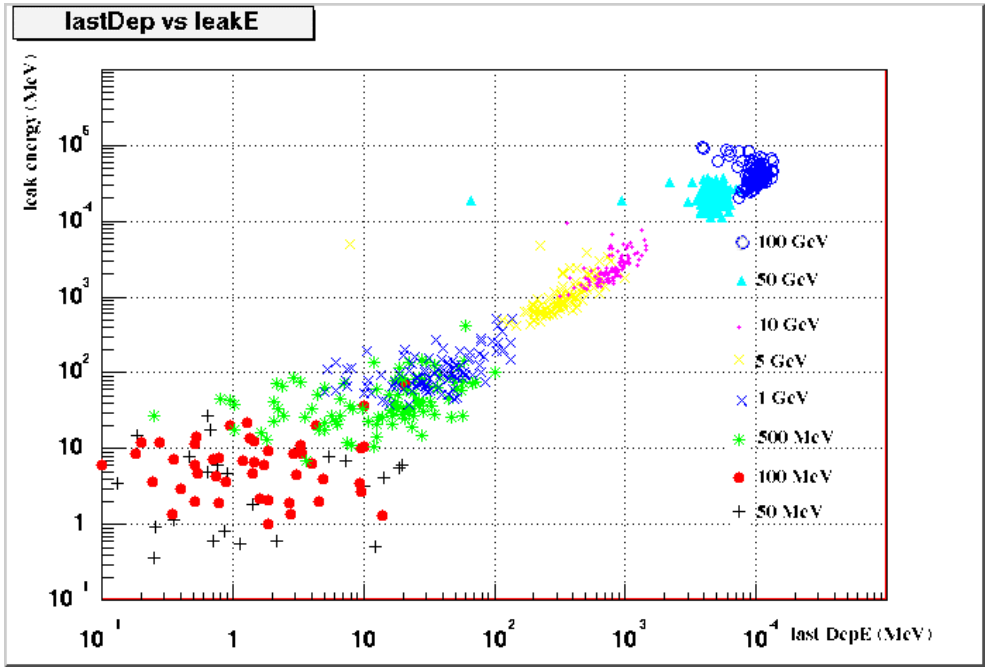

図 4.12：カロリメーター最下層にエネルギーデポジットとリークしたエネルギーの関係。

図 4.12 からカロリメーター最下層にデポジットしたエネルギーとリークしたエネルギーには相関がみられ、狭いエネルギー帯に限れば 1 次関数で表せるのがわかる。

4.2.3 カロリメーターでのプロファイルフィット

入射ガンマ線のエネルギー E が、$E > 50GeV$ の場合、シャワーがカロリメーターの各 layer でデポジットするエネルギーの様子が入射エネルギーに依存することを利用する。Z を、ガンマ線が電子陽電子になった場所からの距離とすると、シャワーの落すエネルギーは距離 Z に対して、次の Γ 関数で与えられる。α、λ は入射エネルギーに依存するパラメーターであり、E_i は i 番目の layer でデポジットしたエネルギーである。

$$E_i = E_{in}\left[P\left(\alpha, \frac{z_i}{\lambda}\right) - P\left(\alpha, \frac{z_{i-1}}{\lambda}\right)\right] \qquad \cdots(3)$$

$$f(z) = \frac{1}{\lambda}\left(\frac{z}{\lambda}\right)^{\alpha-1} e^{-\frac{z}{\lambda}}$$

$$P\left(\alpha, \frac{z_i}{\lambda}\right) = \int f(z)dz$$

これらを用い、実際に得られた各 layer における E_{in} の分布を式(3)でフィッティングすることにより E_{in} を求める。

入射エネルギー50 MeV から 100 GeV の範囲で、カロリメーターの各 layer でデポジットしたエネルギーを調べたのが図 4.13 から図 4.20 である。GLAST では、最下層から、最上層まで、layer から layer31 である。それぞれの図では、入射位置(layer)によって分けてプロットしてある。カロリメーター各 layer でのエネルギーデポジットが、ガンマ線の電子陽電子対生成した位置からの距離に依存するためである。

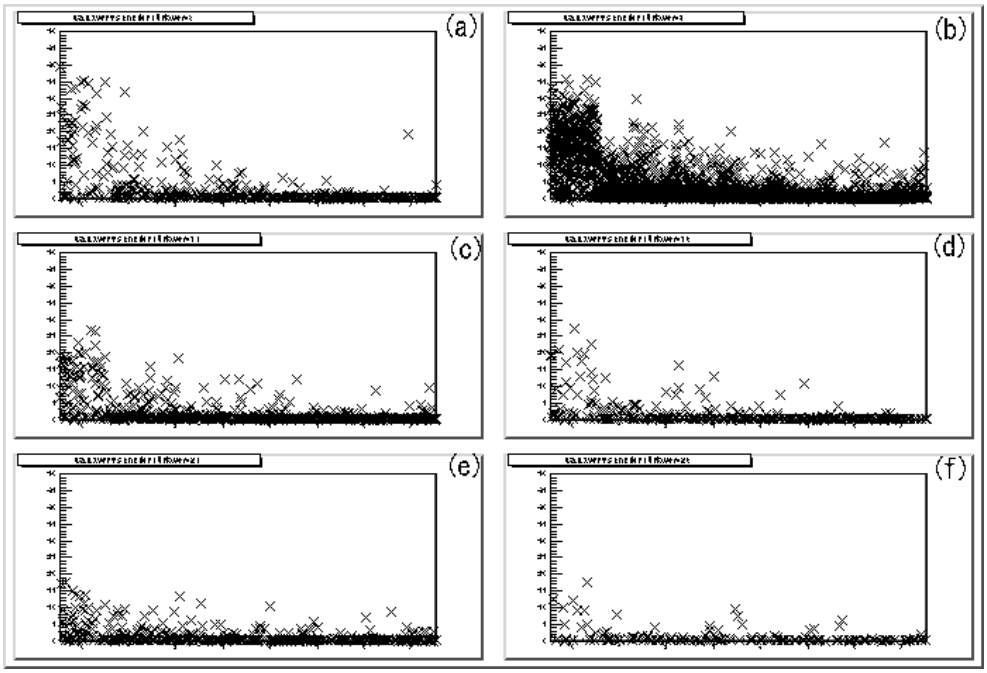

図 4.13 : 入射エネルギー50 MeV。横軸は layer number、縦軸は count。

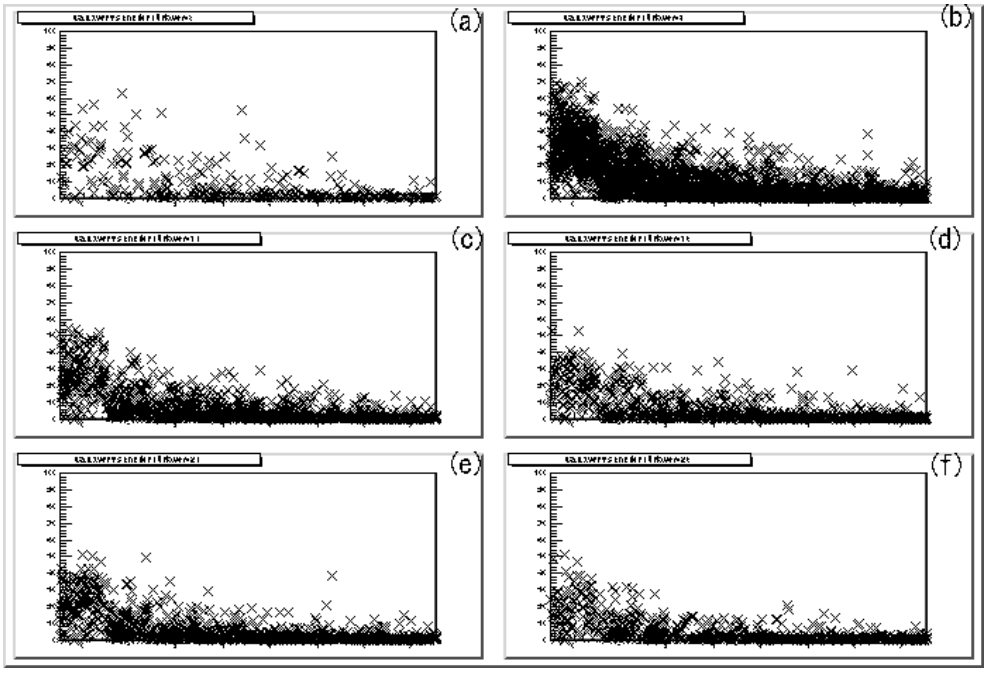

図 4.14 : 入射エネルギー100 MeV。横軸は layer number、縦軸は count。

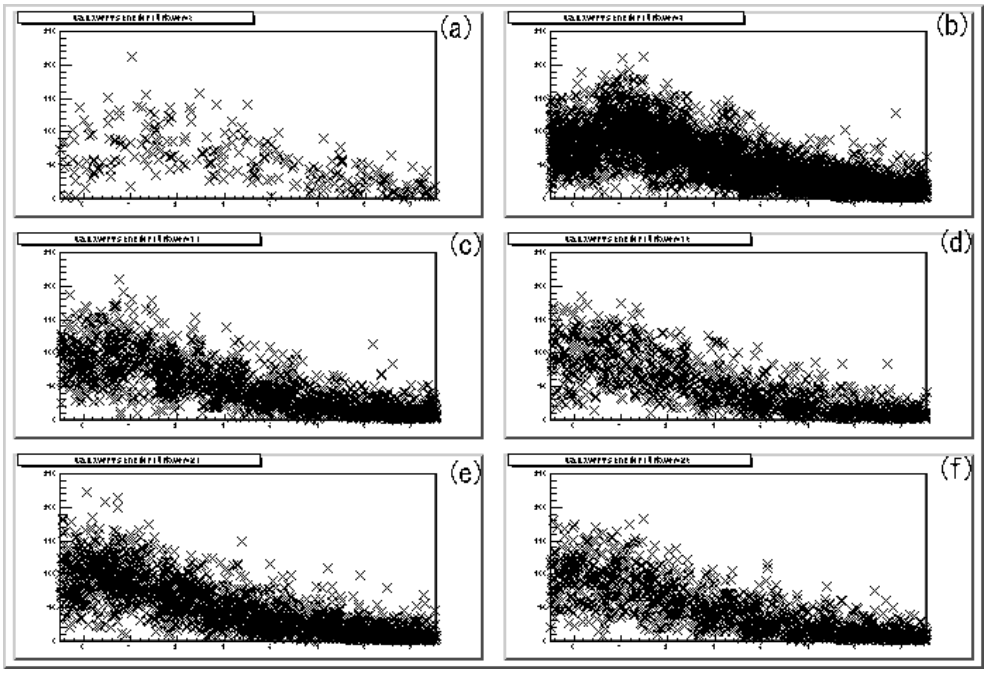

図 4.15 : 入射エネルギー500 MeV。横軸は layer number、縦軸は count。

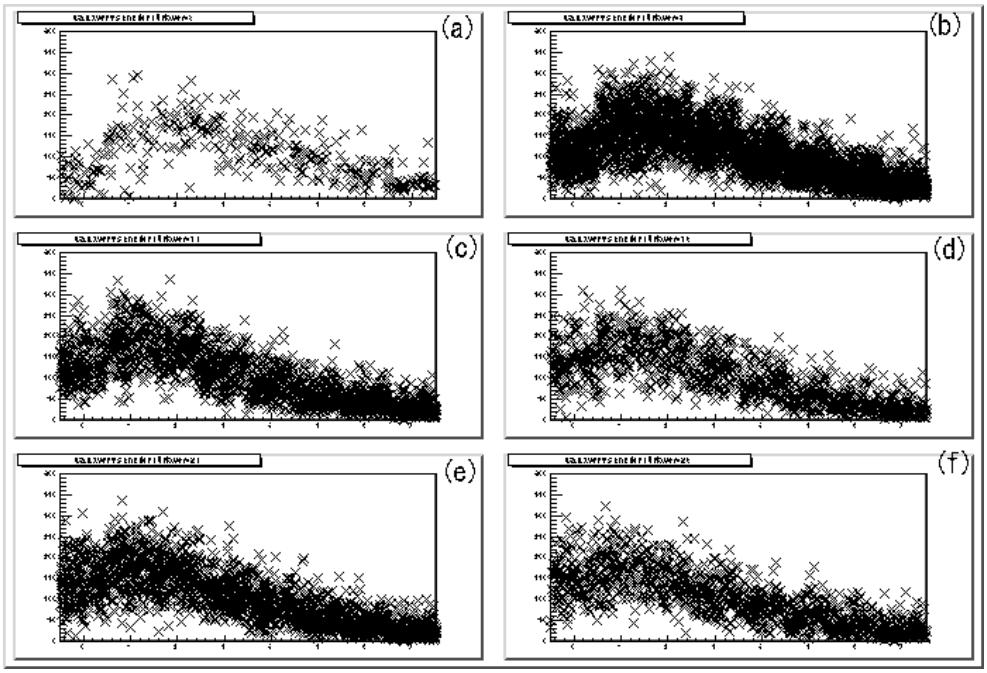

図 4.16 : 入射エネルギー1 GeV。横軸は layer number、縦軸は count。

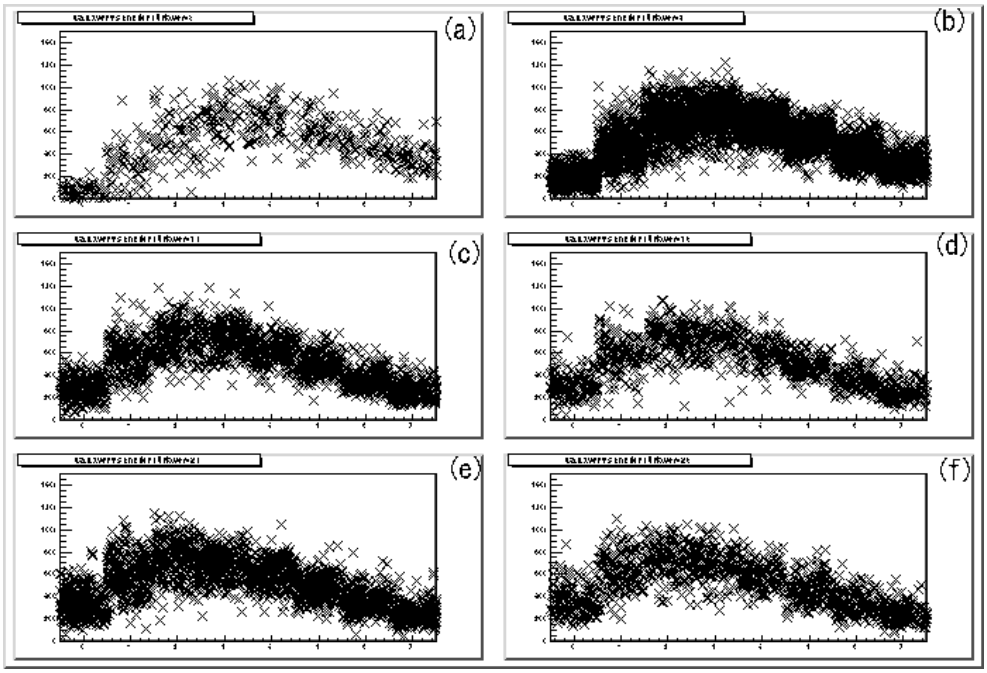

図 4.17 : 入射エネルギー5 GeV。横軸は layer number、縦軸は count。

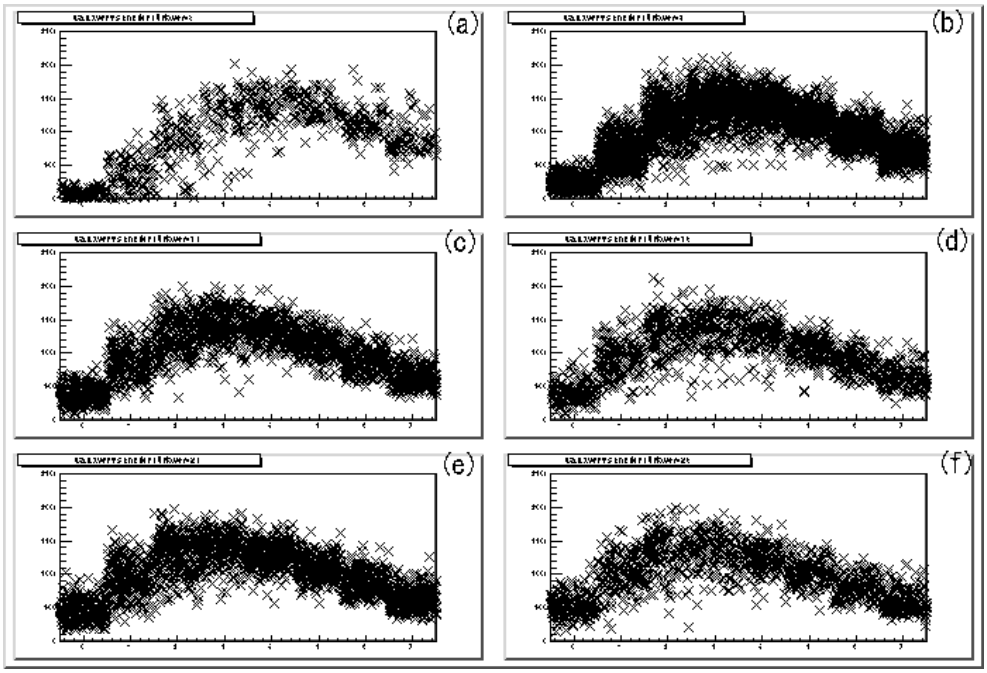

図 4.18 : 入射エネルギー10 GeV。横軸は layer number、縦軸は count。

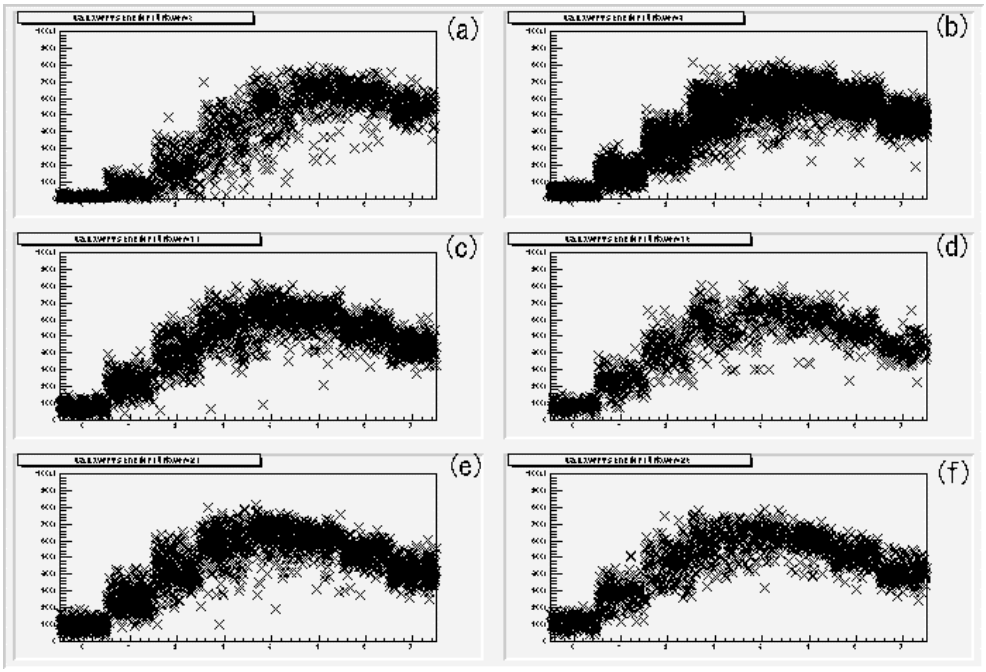

図 4.19 : 入射エネルギー50 GeV。横軸は layer number、縦軸は count。

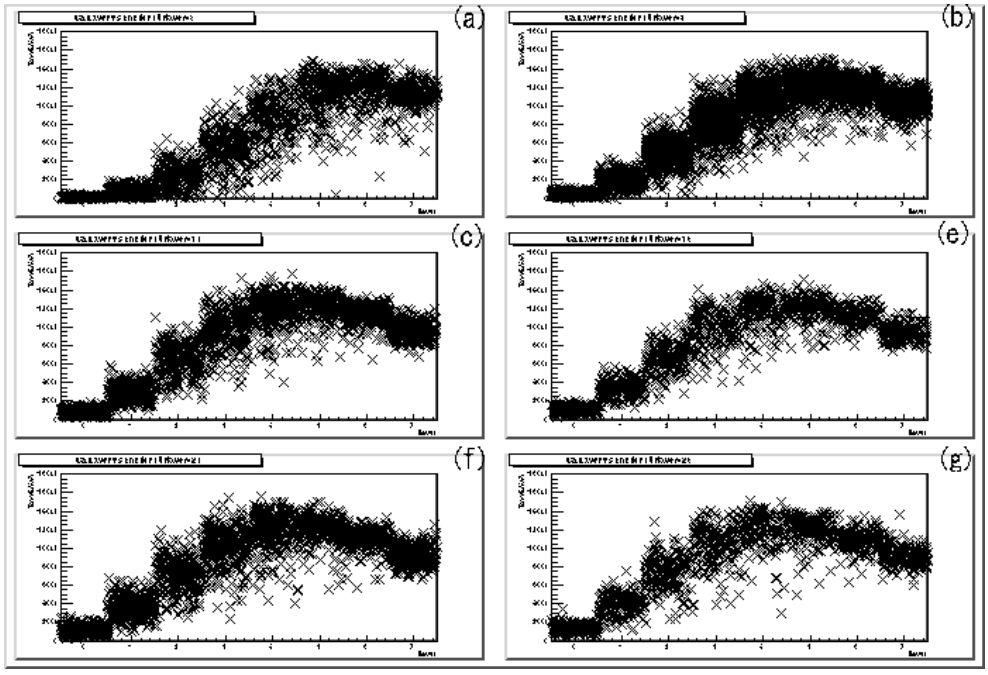

図 4.20 : 入射エネルギー100 GeV。横軸は layer number、縦軸は count。

ガンマ線の電子陽電子対生成の断面積は、ガンマ線のエネルギーによらないにもかかわらず、入射エネルギーによってデポジットしたエネルギーがカロリメーターの layer 毎に異なるのは、シャワーの縦方向の発達が入射エネルギーによるからである。図 4.13 から図 4.20 より、入射ガンマ線が 1 GeV までの低エネルギー領域では、トラッカーで生成された電子陽電子対のシャワーがカロリメーターに入射前、あるいは入射後にほとんどのエネルギーをデポジットしていることがわかり、また 1 GeV から 50 GeV までのエネルギー領域では、シャワーのエネルギーデポジットの最大地点は、カロリメーターの中央付近であることがわかる。50 GeV 以上ではシャワーのエネルギーデポジットの最大地点は、最下層か、もしくはカロリメーターの外である。

よって式(3)にあるようにシャワーの落とすエネルギーは対生成がおきた位置からの距離によって決まりことが、シミュレーションでも確かめられた。低エネルギー領域では、シャワーがカロリメーターに入射前にピークを迎えるため、高エネルギー領域でなければ式(3)を用いることができないことを示す。これを先の Γ 関数でフィッティングすることにより入射エネルギーが求まる。

以上のことより、GLAST 検出器に入射してきたガンマ線のエネルギーは、トラッカーでの情報、カロリメーターでの情報を用いることによって 3 種類の方法で reconstruction が行え、入射エネルギーの決定ができる。

第 5 章 結論と今後の課題

本論文では、GEANT4 シミュレーターを用いて、さまざまなエネルギー領域のガンマ線を入射させ、シミュレーターの位置決定精度や、応答関数を調べた。

シミュレーターの位置決定精度は、入射エネルギーが、100 MeV から 5 GeV の範囲ではビーム実験の角分解能とよく一致していた。しかし、10 GeV から 50 GeV をピークにして、100 GeV あたりから GEANT4 シミュレーターの角分解能は悪化し、500 GeV では 23.1° と非常に悪かった。10 GeV 以降は、ビーム実験での角分解能は求められていなかったため比較はできなかった。これは、カロリメーターからのバックスプラッシュが原因であると考えており、今後、バックスプラッシュを除き比較する必要がある。さらに、トラッカーでは、コンバーターである鉛の層の厚さが違うため、これらが reconstruction に与える影響など、考慮し直す必要があり、さらなる解析が必要である。

カロリメーターでの入射エネルギーの reconstruction は、GLAST ビーム実験によって知られている 3 種類の方法がシミュレーションデータにおいても有効であることを示し、シミュレーターの動作が正しいことを確認した。今後は、得られたデータを実際に reconstruction して入射エネルギーを決定することが課題となる。

今回は GEANT4 のトラッカー出力データを ROOT フォーマットに変換し、reconstruction を行ったが、残るアンチコインシデンスディテクターや、カロリメーターの出力も ROOT フォーマットへ変換し reconstruction を行う必要がある。

謝辞

この論文を製作するにあたり、直接指導をいただきました大杉節教授、深沢泰司助教授、また、適切な助言をいただきました水野恒史氏、半田隆信氏に深く感謝いたします。また、諸事にわたり御協力添えと励ましをいただきました林香苗氏に感謝いたします。

また、本研究を進めるに際して御協力と、御討論をいただいた平野勝也氏、川添哲志氏に深く感謝いたします。さらに、実験その他の面におきまして御協力と激励をいただきました広島大学素粒子実験・高エネルギー宇宙学研究室の皆様に感謝いたします。

参考文献

[1] 日本 GLAST ワーキンググループ　次世代ガンマ線衛星 GLAST で期待される科学 1997

[2] Results from Beam Test of the Engineering Model of the GLAST Large Area Telescope 2000

[3] GLAST Testbeam Users guide 2000

[4] Brian Jones　A physicist's Guide to Kalman Filter

[5] Jose A.Hernando The Kalman Filter Technique applied to Track Fitting in GLAST 1998

[6] SLAC testbeam

URL http://www-sldnt.slac.stanford.edu/glast/testbeam/

[7] ROOT

URL http://root.cern.ch/

[8] GEANT4

URL http://wwwinfo.cern.ch/asd/geant4/

[9] katuya hirano　Development of Simulator for the Gamma-ray Satellite GLAST 2001

付録 A: Reconstruction データを読み出すサンプルプログラム

```
#include <math.h>
#include "TkrReconAnalysis.h"
#include "stuff.h"

void TkrReconAnalysis::HistDefine ()
{
  // define the histgrams
  AnalysisObj::HistDefine ();

  ID = new TH1F("id","ID # of this TkrLocator",10,0,10);
  ID->SetFillColor (38);

  SLOPEX = new TH1F("SlopeX","the direction of this TkrLocator in X",100,-1,1);
  SLOPEX->SetFillColor (38);

  SLOPEY = new TH1F("SlopeY","the direction of this TkrLocator in Y",100,-1,1);
  SLOPEY->SetFillColor (38);

  SLOPER = new TH1F("SlopeR","the direction of this TkrLocator in R",200,0,1);
  SLOPER->SetFillColor (38);

  SLOPE = new TH2D("slopexy","gamma",400,-0.2,0.2,400,-0.2,0.2);
  SLOPE->SetFillColor (2);

  SLOPER1 = new TH1F("SlopeR1","the direction of this TkrLocator in R for
thin-pB",200,0,1);
  SLOPER1->SetFillColor (38);

  SLOPER2 = new TH1F("SlopeR2","the direction of this TkrLocator in R for
thick-pB",200,0,1);
  SLOPER2->SetFillColor (38);
}

void TkrReconAnalysis::RefreshHistList ()
```

```
{
  // refresh your histgram pointer here
  TFile *hf = GetHistFile();

  ID  = (TH1F*)hf->Get("id");
  SLOPEX = (TH1F*)hf->Get("SlopeX");
  SLOPEY = (TH1F*)hf->Get("SlopeY");
  SLOPER = (TH1F*)hf->Get("SlopeR");
  SLOPE = (TH2D*)hf->Get("slopexy");
  SLOPER1 = (TH1F*)hf->Get("SlopeR1");
  SLOPER2 = (TH1F*)hf->Get("SlopeR2");
}
void TkrReconAnalysis::DoAnalysis (Recon* rec)
{
  TkrRecon* tkrRec =rec->getTkrRecon();
  Int_t nClusters = tkrRec->getSiClusters()->GetEntries();
  //NCLUSREC->Fill(nClusters);
  //cout << nClusters << endl;
  for (int iCluster = 0; iCluster < nClusters; iCluster++){
     TkrSiCluster* si = (TkrSiCluster*)(tkrRec->getSiClusters()->At(iCluster));
     UInt_t getL = si->getLayer();
     UShort_t cents = si->getCenterStrip();
     UShort_t nums = si->getNumStrips();
     Float_t posi = si->getPosition();
     Float_t zpos = si->getZPosition();
     UInt_t siid = si->getId();
     //cout  <<getL<<"    "<<cents<<  "    "<<nums<<"    "<<posi<<"    "<<zpos<<"
"<<siid<<endl;
  }

  Int_t nTracks = tkrRec->getTracks()->GetEntries ();
  //cout << nTracks << endl;
  //NTRACKS->Fill (nTracks);
  Float_t ch2[1000];
  Float_t qu[1000];
  UInt_t fi[1000];
```

```
  Float_t slx[1000];
  Float_t sly[1000];

  for (int iTrack = 0; iTrack < nTracks; iTrack++) {
     TkrTrack* tra = (TkrTrack*)(tkrRec->getTracks()->At(iTrack));

     UShort_t tid = tra->getId();
     Float_t resi = tra->getResidual();
     ch2[iTrack] = tra->getChi2();
     qu[iTrack] = tra->getQuality();
     fi[iTrack] = tra->getFirstLayer();
     UInt_t numc = tra->getNumClusters();
     Float_t enei = tra->getEnergyInput();
     Float_t ened = tra->getEnergyDet();
     //cout << "tid= " << tid <<" resi= "<< resi <<" ch2= "<< ch2[iTrack] <<" qu= "<<
qu[iTrack] <<"    "<< fi[iTrack] <<"    "<< numc <<"   "<< enei <<"    "<< ened <<endl;
  }

  Int_t nLocators = 0;
  for (int iTrack = 0; iTrack < nTracks; iTrack++, nLocators++) {
     TkrTrack* tra = (TkrTrack*)(tkrRec->getTracks()->At(iTrack));
     Int_t nLocator = tra->getLocator()->GetEntries ();
     //cout << nLocator << endl;
     //NLOCAT->Fill(nLocator);

     for(Int_t iLocator = 0; iLocator < nLocator; iLocator++){
       TkrLocator *loc = (TkrLocator *)(tra->getLocator()->At(iLocator));

       TVector3* pos = loc->getPosition();
       Double_t X = pos->X();
       Double_t Y = pos->Y();
       Double_t Z = pos->Z();

       UInt_t idn = loc->getId();
       slx[nLocators] = loc->getSlopeX();
       sly[nLocators] = loc->getSlopeY();
```

```
        Float_t sigx = loc->getSigma_X();
        Float_t sigxs = loc->getSigma_Xslope();
        Float_t covx = loc->getCov_X();
        Float_t sigy = loc->getSigma_Y();
        Float_t sigys = loc->getSigma_Yslope();
        Float_t covy = loc->getCov_Y();

        //  cout << X<<"    "<<Y <<"    "<<Z<<endl;
        //cout  <<  "idn=  "  <<  idn<<"    "<<slx[nLocators]<<"    "<<sly[nLocators]<<"
"<<sigx<<"    "<<sigxs<<"    "<<covx<<"    "<<sigy<<"    "<<sigys<<"    "<<covy<<endl;
ID->Fill(idn);
          //SLOPEX->Fill(slx[nLocators]);
          //SLOPEY->Fill(sly[nLocators]);
      }
   }

   if( nTracks != nLocators ) {
     cout << "nTracks != nLocators: " << nTracks << " " << nLocators << endl;
   } else if ( nTracks > 0 ) {
     Float_t max_x=0.0;
     Float_t max_y=0.0;
     Int_t nmx=0;
     Int_t nmy=0;
     for(int iTrack = 0; iTrack < nTracks; iTrack++){
       if(slx[iTrack]!=0 && max_x<qu[iTrack]) {
         nmx=iTrack;
         max_x=qu[iTrack];
       }
       if(sly[iTrack]!=0 && max_y<qu[iTrack]) {
         nmy=iTrack;
         max_y=qu[iTrack];
       }
     }
     //cout << "slxy for max: " << slx[nmx]<< " " << sly[nmy] << endl;
     SLOPEX->Fill(slx[nmx]);
     SLOPEY->Fill(sly[nmy]);
```

```
    SLOPE->Fill(slx[nmx],sly[nmy],1);
    SLOPER->Fill(sqrt(slx[nmx]*slx[nmx]+sly[nmy]*sly[nmy]));

    //azechi wrote
    cout << "K " << sqrt(slx[nmx]*slx[nmx]+sly[nmy]*sly[nmy])<<endl;

    if( fi[nmx]<=4 ) SLOPER2->Fill(sqrt(slx[nmx]*slx[nmx]+sly[nmy]*sly[nmy]));
    else             SLOPER1->Fill(sqrt(slx[nmx]*slx[nmx]+sly[nmy]*sly[nmy]));
    //       cout << "SLOPEXY: " << slx[nmx] << " " << sly[nmy] << " " <<
sqrt(slx[nmx]*slx[nmx]+sly[nmy]*sly[nmy]) << endl;
  }
  //cout << "out\n";
}

void TkrReconAnalysis::Draw ()
{
  if (!m_can2)
    m_can2 = new TCanvas ("TkrLocatRecon2","",700,500);
  TCanvas* can = m_can2;

  can->Divide(2,2);
  can->cd (1);
  SLOPEX->Draw();
  can->cd (2);
  SLOPEY->Draw();
  can->cd (3);
  SLOPER->Draw();
  can->cd (4);
  SLOPE->Draw ();
  can->Print("can.ps");

  if (!m_can3)
    m_can3 = new TCanvas ("TkrLocatRecon3","",700,500);
  TCanvas* can = m_can3;
  /*
  can->Divide(2,2);
```

```
  can->cd (1);
  SLOPER1->Draw();
  can->cd (2);
  SLOPER2->Draw();
  can->Print("can2.ps");
  */
  SLOPER->Draw();

}
void TkrReconAnalysis::AllHistDelete ()
{
  ID = SLOPEX = SLOPEY =  SLOPE = 0;
   AnalysisObj::AllHistDelete ();

}
```

付録 B: ROOT フォーマットへのサンプルプログラム

```
{
  gROOT->Reset ();
  gSystem->Load ("/data1/local/TBEvent/.libs/libTBEvent.so");
  TFile *hfile = new TFile("RAW500GeV.root","RECREATE");
  fTree = new TTree("T","sample ROOT tree");
  fTree->SetAutoSave(10000000);
  event = new Event();
  event->Create();
  fTree->Branch("Event","Event",&event,64000,1);

  ifstream fi("tkr500Gsecond.dat");
  if(!fi)
    cout << "Can't open file!!" << endl;
  char val[10];
  char dam1[10],dam2[10];
  char x1[10],x2[10];

  while(!fi.eof()){
```

```
fi >> dam1 >> dam2;
if (fi.eof()) break;

TkrLayer *xlayer;
TkrLayer *ylayer;
    //  analysis from 0 layer to 15 layer
for ( int  lay = 0; lay < 16; lay++){
  xlayer = new TkrLayer(0);
  ylayer = new TkrLayer(0);
  xlayer->setXY(TkrLayer::X);
  ylayer->setXY(TkrLayer::Y);

  xlayer->setLayer(lay);
  ylayer->setLayer(lay);

  fi >> x1 >> x2;                       //input data for X
  int num = atoi(x2);
  for (int l = 0; l < num; l++){
    fi >> val;              //input strip channel
    int channel = atoi(val);

    StripID *strip = new StripID();
    strip->setStrip(channel);
    xlayer->getStrips()->Add(strip);
  }
  fi >> x1 >> x2;
  int num = atoi(x2);
  for (int l = 0; l < num; l++){
    fi >> val;              //input strip channel
    int channel = atoi(val);

    StripID *strip = new StripID();
    strip->setStrip(channel);
    ylayer->getStrips()->Add(strip);
  }
  event->getTKR()->Add(xlayer);
```

```
      event->getTKR()->Add(ylayer);
      //      cout <<endl;
    } // layer 0..15
  //  cout<<endl;
  event->getTKR()->Sort();
  fTree->Fill();
  event->Clean();
  event->Create();
  }
  hfile->Write();
  hfile->Close();
  fi.close();
}
```