モンテカルロシミュレーションによる宇宙γ線検出器EGRETの応答関数の再評価

広島大学 理学部 物理科学科
高エネルギー宇宙・素粒子実験研究室
山本 和英

2005年02月10日

概 要

宇宙では、超高エネルギーへの粒子加速、ブラックホールや中性子星に付随する高温プラズマなど、目で見ることが出来ないような高エネルギー現象が至る所で起こっている。そのような現象を観測するのには、X 線、γ 線による観測が不可欠である。そのγ線領域での観測を進展させるべく、γ線衛星 GLAST の開発が、日本を含む国際協力で行われている。GLAST は 1990 年代に活躍した CGRO 衛星搭載の EGRET 検出器の後継で、数 10MeV から数 100GeV までの広いエネルギー領域をカバーし、EGRET に比べ感度も数 10 倍に向上される。GLAST による天体観測の計画や、解析方針においては、EGRET での観測結果が重要な役割りを担っている。その一方、EGRET の応答関数にはいくつか不定性があることも知られている。そこでγ線の観測に慣れ、GLAST のデータ解析の方法の開発につなげる事を目的として、GEANT4 を利用した EGRET 検出器のシミュレータを開発した。またこれを用いて EGRET の応答関数の再評価を行った。

目 次

第1章　Introduction

1962年のジャコーニ、ロッシらによるX線天体の発見により、X線やγ線を放射するような高エネルギー天体の観測が1960年代から始まり、以下のような高エネルギー現象が至る所で起こっていることが分かった。すなわち活動銀河核(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、γ線バースト、太陽フレアなどである。これらの天体では多様なメカニズムで粒子が加速され高エネルギー粒子が宇宙空間に振りまかれており、宇宙線の加速現場と考えられている。まさに地上では実現できないような巨大な加熱器あるいは加速器なのである。さらに、高密度天体は、地上では決して実現することができないような極端な強重力、高密度、高温、強磁場のもとでの物理学を検証する貴重な場でもある。こうした現象は可視光の観測では見えないことが多く、X線やγ線での観測が不可欠となる。また、高エネルギー現象を観測することによって、宇宙の進化や構造形成などの解明にも迫ることができるであろう。

このγ線天文学を大きく進展させるべく次世代γ線衛星GLAST(Gamma-ray Large Area Space Telescope)の開発が行われている。GLASTは1991から2000年に活躍したCGRO衛星に搭載されたEGRET検出器の後継で、数10MeVから数100GeVの広いエネルギー領域をカバーし、EGRETの30倍以上もの感度をもつ。それによりEGRETで検出した天体の数（約300）の数10倍もの天体を検出できると期待されておりこのエネルギー領域の観測が一気に進展すると思われている。

現在、2007年度の打ち上げを目指し、フライトモデルの製作やGEANT4を用いた検出器シミュレータの開発、装置の動作を理解するためのビーム試験の準備が進められている。GLASTで期待されるサイエンスの予想は、EGRETの観測結果に大きく依存している。またGLASTのような巨大で複雑な検出器のデータ解析手法の開発は、打ち上げ前から行う必要が有り、その入力データとしてEGRETの結果が用いられている。しかしながらEGRETの時代には、GEANT4のような、複雑な検出器をシミュレーションする道具が十分でなく、ビーム試験自身にも問題がいくつかあり、その応答関数には、不定性があることが知られている。そこで本研究では、γ線での観測に慣れ、GLASTのデータ解析手法の確立につなげる目的でGEANT4

を用いて EGRET 検出器のシミュレータを開発した。またこれを用いて、EGRET の応答関数の再評価を行った。

第2章　EGRET検出器

2.1　CGRO衛星搭載EGRET検出器

2.1.1　EGRET検出器の概要と成果

EGRETは1991年から2000年に活躍したCGRO衛星に搭載された総重量1830 kgの大型検出器である (図2.1 左)。次世代γ線衛星GLASTでも観測を目指す数10 MeVから数10 GeVのエネルギー領域で全天探査を行い、活動銀河核 (AGN)、太陽フレア、パルサー、大マゼラン星雲 (LMC) などの高エネルギー天体からのγ線を検出した。EGRETはGLASTと同じく、γ線の対生成を利用した電子陽電子対生成型検出器で、当時としては画期的な広い視野、高い角度分解能と低バックグランドを実現し, ほぼ未開であったエネルギー領域で271個ものγ線天体の観測に成功した (図2.1 右)。そしてEGRET検出器による銀河面からのγ線放射の観測は、宇宙陽子線と星間物質との相互作用を利用して、銀河面内の物質分布と宇宙線の分布を直接測定出来ることを示した。(S.D. Hunter et al. 1997)

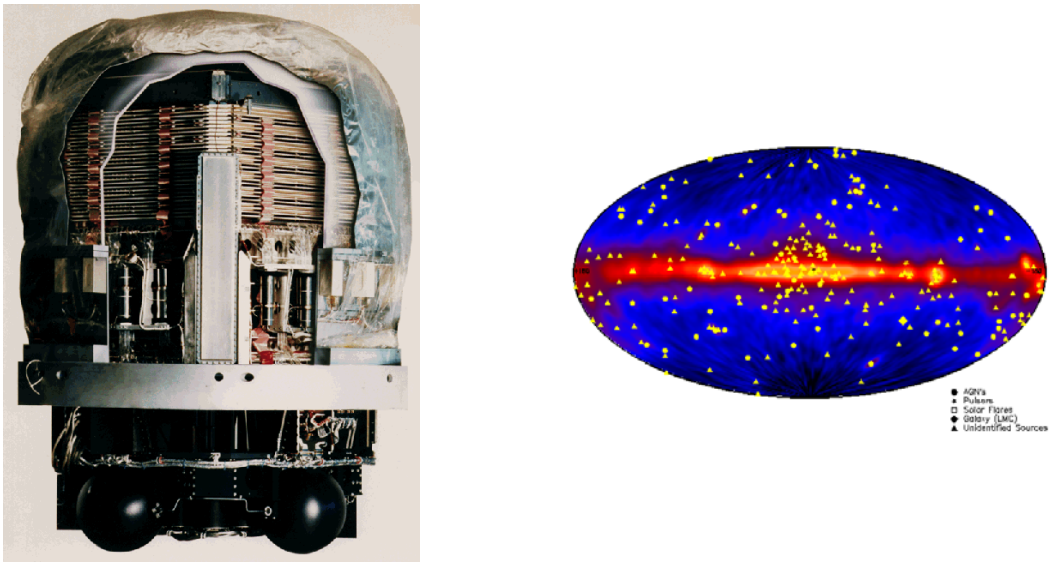

図 2.1:　(左)EGRET検出器 (右)EGRETで見つかったγ線天体 (EGRET 3rd Catalog)

2.1.2 EGRET 検出器の構造

EGRET は、GLAST と同じく電子陽電子対生成を利用した検出器で、γ線の方向を測る飛跡検出部と、エネルギーを測るカロリメータ部からなり、荷電粒子事象を落とすためのプラスチックシンチレータで覆われている (図 2.2)。

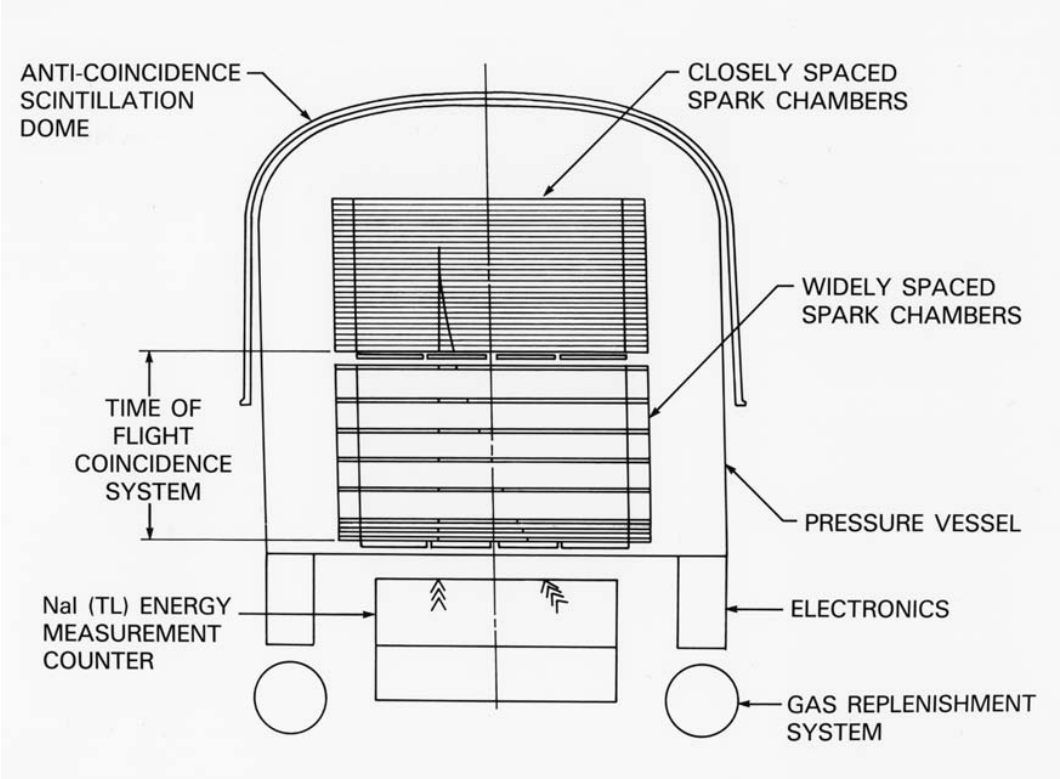

図 2.2: EGRET 検出器の概念図

飛跡検出部分は、x,y 方向に各層 992 本のワイヤが張られた 36 層のスパークチェンバーで構成されており二層目から下は各層の上にコンバータ (タンタルの薄い板) が置かれていてる。28 層と 29 層の間と 36 層には Time Of Flight (TOF) Coincidence System と呼ばれるプラスチックシンチレータが置かれて、飛跡検出部分の下には、厚い NaI(Tl) のカロリメータが置かれている。

γ線が入射してくると、飛跡検出部のコンバータで対生成を起こし、電子陽電子対がつくられ、スパークチェンバー中のガスを電離しながら進むとともに、TOF シンチレータおよびカロリーメータ (NaI シンチレータ) にエネルギーを落とす (図 2.3)。TOF シンチレータの時間情報から、粒子が下向きに走ったことが分かるとトリガとなり、スパークチェンバーに高電圧がかかる。そしてガスの電離で生じた電子がワイヤに引き寄せられ、その際にねずみ算式に増幅することで、スパーク (火花) が発

生し、それを記録することで電子・陽電子の飛跡、ひいては入射γ線の方向が分かる。またNaIシンチレータで電子・陽電子対の総エネルギー、つまり入射γ線の総エネルギーを測ることができる。

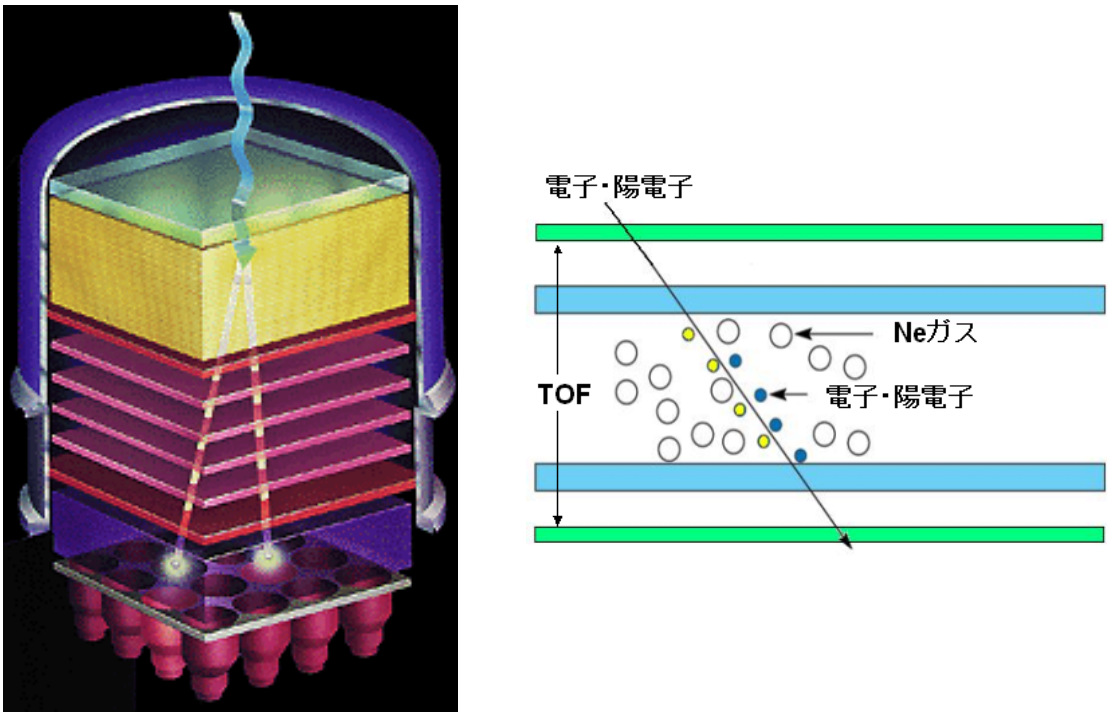

図 2.3: (左) スパークチェンバー中でγ線が電子陽電子対生成を起こす様子。(右) スパークチェンバー中での反応の様子。

2.2 GLAST衛星

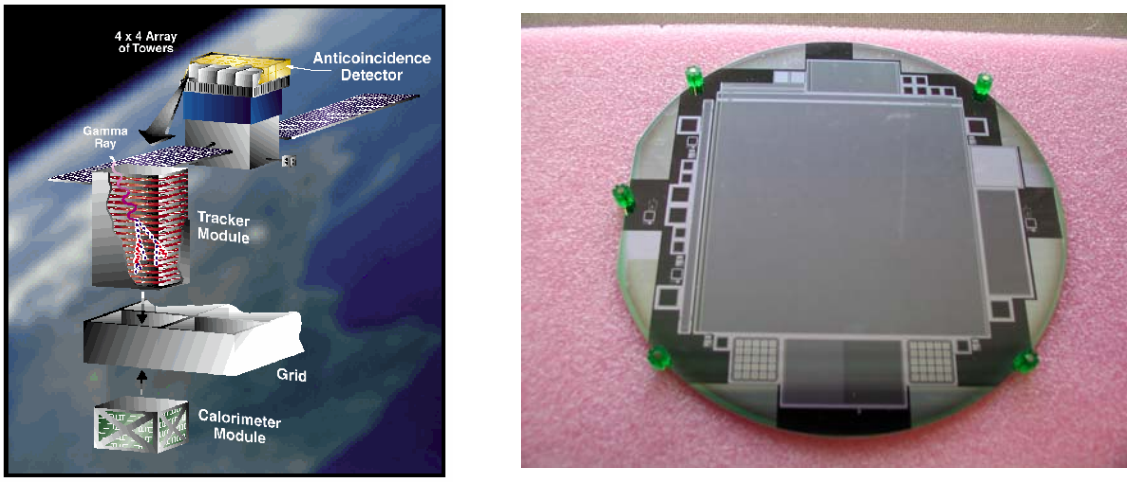

図 2.4: (左)GLAST 衛星の LAT 検出器。 (右) シリコンストリップ検出器の写真

GTAST 衛星は EGRET 検出器の後継の LAT(Large Area Telescope) 検出器を搭載した衛星で、アメリカ、日本、フランス、イタリアの国際強力で 2007 年度打ち上げ予定の宇宙 γ 線衛星である。LAT 検出器は半導体検出技術を利用して作られ、総重量 3000kg にも及ぶ大型検出器である。これはタワーと呼ばれるモジュール構造を $4 \times 4 = 16$ 個並べたものになっていて、各モジュールはトラッカー部とカロリーメータ部に分かれ、全体をプラスチックシンチレータでできた荷電粒子アンチカウンター部が覆っている (図 2.4 左)。モジュール化はトリガーシステムの負担を減少し、直接関係の無いモジュールでのイベントによる不感時間を減少している。トラッカー部は 18 層の約 200μm 幅の細い電極をたくさん並べた Si 検出器と薄い鉛のシートでできている。この Si 検出器 (シリコンストリップ検出器) は、当研究室が中心となって開発したものである (図 2.4 右)。そしてエネルギーを測定する電磁カロリーメータ部は CsI(Ta) シンチレータのブロックで構成されている。トラッカー部がシリコン検出器になったため、電子・陽電子が通過する際の位置を約 200μm という高い精度で記録でき、また各層の間隔を小さくできるようになったため数 GeV の γ 線で 0.1 度という優れた角度分解能を得られるようになった。さらにカロリーメータ部はセグメント化されておりシャワーの形状を測定することで、カロリーメータから抜け出るような高いエネルギーの γ 線の測定も可能となった。またモジュールを横に並べた横長の構造をもつことで大きな角度で入射してきた γ 線からの電子・陽電子対の飛跡も再構築できるようになった。それにより、大きな有効面積と常時全天の 20%を一度にカバー出来る広い視野をもつ。それゆえ EGRET と同じ検出方法

を用いているが、エネルギー範囲、視野、角度分解能の全てでEGRET検出器を大きく上回る性能を持ち、長期的なサーベイで得られる検出感度はEGRETの数10倍に達すると予測されている。(表2.1, 図2.5, 図2.6)

表 2.1: EGRETとGLASTの性能比較表

	EGRET	GLAST
エネルギーバンド	30 MeV-10 GeV	30 MeV-100 GeV
Field of View	0.5 sr	2.4 sr(20% of 4π)
有効面積	1,500 cm^2	11,000 cm^2
エネルギー分解能	10%	10%
1イベントデッドタイム	100 ms	20 μs
点源位置決定精度	5–30分角	0.5–5分角
点源感度	$\sim 1\times10^{-7}$ $cm^{-2}s^{-1}$	$\sim 2\times10^{-9}cm^{-2}s^{-1}$
検出天体数	271	>10000
重量	1820 kg	2560 kg
電力	160 W	600 W
Orbit(28.5度incl.)	350 km	550 km
寿命	9年	>5年

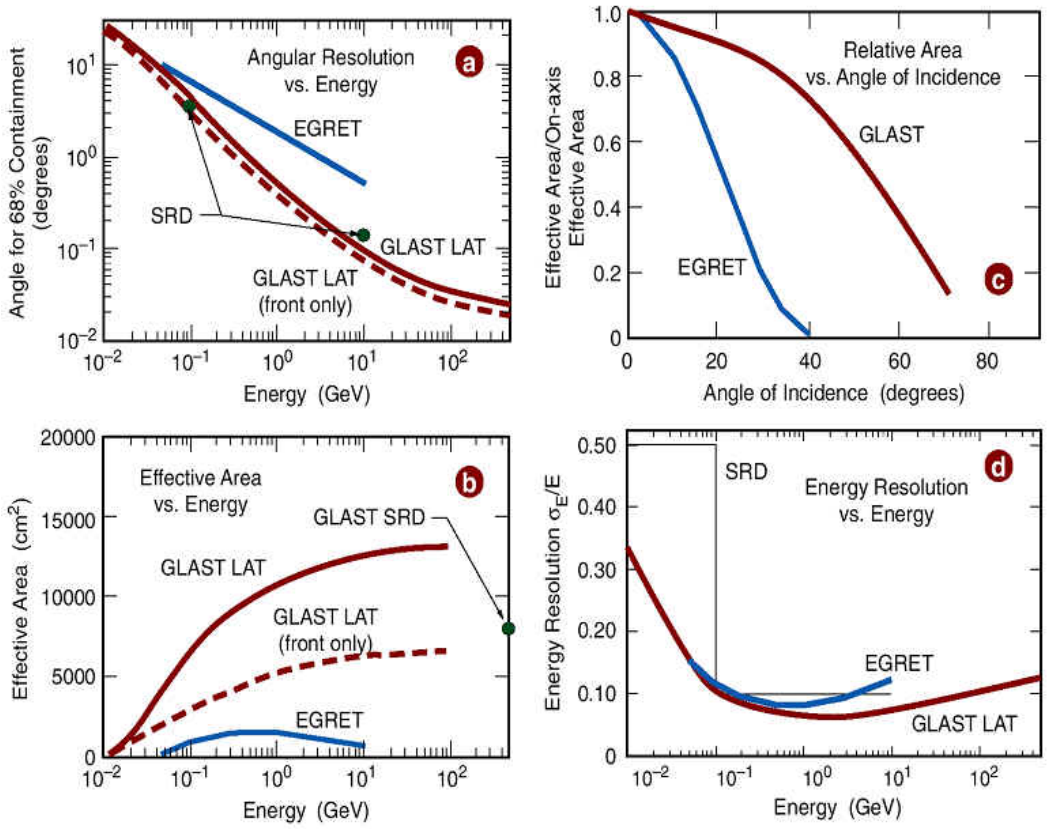

図 2.5: GLAST と EGRET の性能比較グラフ (a) 角度分解能、(b) 有効面積、(c) 有効面積の入射角依存性、(d) エネルギー分解能

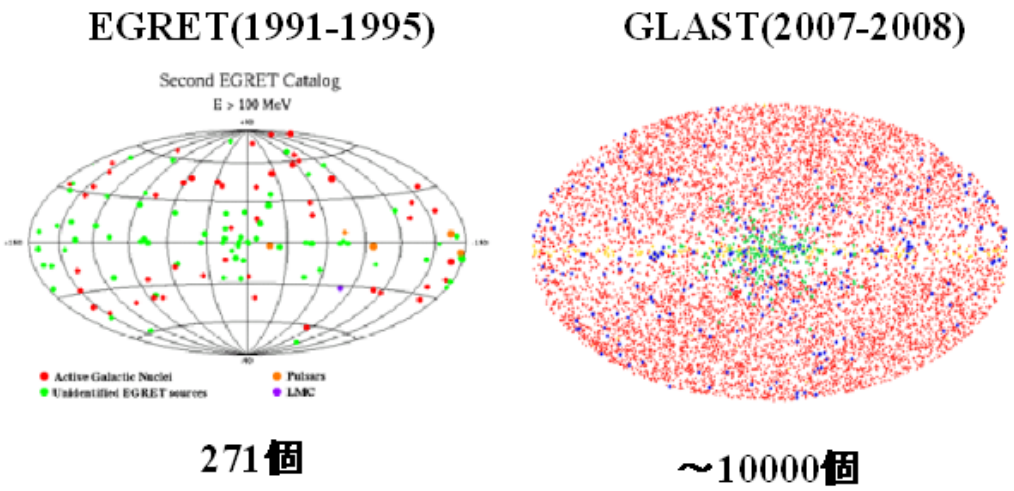

図 2.6: (左)EGRET で検出された γ 線天体 (右)GLAST で検出が期待される天体

2.3 GEANT4を用いたEGRET/GLASTシミュレーター

GLAST の開発の一環として、ビーム試験の準備とともに、モンテカルロ法を用いたシミュレーターの開発がなされている。これまで開発が進められてきたGLAST用シミュレーターは、その核心部となるシミュレーションツールキットのサポートが失われたために、新たに検出器の構造を精密に記述でき信用のおけるシミュレーターとして国際的に共同開発が行われているGEANT4を使うこととなった。我々はEGRETのシミュレーターとしても同様にGEANT4を用いることにした。

GEANT4は、陽子・中性子・電子・γ線・π粒子・μ粒子などの素粒子が物質中で起こす複雑な振舞や反応電離や光子放射によるエネルギー損失、対生成、カスケードシャワーなど）を正確にシミュレートするためのツールである。このような粒子と物質の相互作用のシミュレーションを行うことは、高エネルギー・原子核実験だけでなく、宇宙科学、放射線医学などの分野においても不可欠な要素である。そうしたシミュレーションプログラムとして、GEANT4以前にGEANT3というものがあり、これは1980年代にCERNを中心とするメンバーによって、FORTRAN言語を用いて開発された。GEANT3 は、多くの実験で標準の測定器シミュレーターとして使われるようになり、現在では世界で最も広く使用されているシミュレーターの一つである。一方、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、オブジェクト指向技術が、様々な分野でますます定着してきている。このことは、高エネルギー実験の分野でも例外ではなく、これまでのGEANT3での経験を可能な限り生かした上で、オブジェクト指向技術とC++言語を用いて再構築を行ったものが相互作用のモンテカルロシミュレータGEANT4である。

2.4 EGRET検出器の応答関数の問題点

検出器の出力信号から、入射粒子 (EGRET検出器やGLAST衛星のLAT検出器の場合は高エネルギーγ線) の情報を得るために、応答関数というものが用いられる。これは、γ線検出器の場合は入射γ線の方向、エネルギーがある値をとる場合に、検出器で測定されたエネルギーや方向の分布で表される。EGRET検出器の開発が行われた当時 (1990年以前) はGEANT4は存在せず、キャリブレーションのためのビーム試験の結果を再現するのに、EGRETチームはEGS4と呼ばれるシミュレーションツールを用いていた。これは電磁相互作用のシミュレーションツールとして、1985年に発表されたもので、20年以上の歴史を持ち、反応プロセスのシミュレーターとしては、信頼性が非常に高い。しかし複雑な検出器ジオメトリを記述、

シミュレートする能力は十分ではなく、また物理プロセスも電磁相互作用に限られることから、EGRET のような高エネルギー γ 線の複雑な検出器をシミュレートするには、必ずしも十分ではなかった。実際、検出器の応答関数を調べるビーム試験は 35 MeV から 10 GeV のエネルギー範囲で行われているが、100 MeV 以下の低エネルギー領域、1 GeV 以上の高エネルギー領域ではビームに低エネルギー光子が混入していたため、不定性が大きいとされている (A.Walker 1990, PhD Thesis)。このうち 1 GeV 以上の領域は 2.1.1 節で述べた銀河面からの γ 線放射において、宇宙陽子線の寄与による成分と電子による成分を分けるのに決定的な情報であり、これは γ 線宇宙物理学で最も重要なサイエンスを左右するエネルギー帯である (図 2.7)。また 100 MeV 以下のエネルギーは他の X 線、γ 線検出器の結果と比較するさいに用いられる領域であるが、かに星雲の観測などで不整合がみられたため、ビーム試験で得られた有向面積は 2-3 倍高すぎたと解釈することで対処している (D. Thompson et al. 1993, ApJ 415, L13)。また EGRET が感度をもつエネルギーの上限は 30 GeV 程度であるが、ビーム試験は 10 GeV までしか行われていないため、高エネルギー領域の応答関数は適当な関数で外挿することで決められている (S.D. Hunter et al. 1997, ApJ 481, 205; A. Strong et al. 2004, ApJ 613, 962 など)。そこで本研究では、GLAST につなげることを目的として EGRET 検出器のシミュレーターを開発すると同時に、それを用いて EGRET 検出器の応答関数の再評価を行った。

2.5 本研究の目的

既に述べたように、EGRET 以前は数 10 MeV 以上の γ 線の分野は未開に等しく、断片的な知識が得られているに過ぎなかった。GLAST で観測される γ 線宇宙の予測やその解析ツールの開発は EGRET のデータに大きく依存しており、EGRET 検出器を理解しデータを再解析することは、γ 線のデータに慣れ GLAST におけるデータ解析の手法を確立する上で極めて重要である。また EGRET の応答関数にはいくつか問題があることも知られている。よって EGRET の応答関数を再現することは、GLAST の観測計画で用いられる EGRET のデータの信頼性を保証することにつながり、また応答関数に修正が必要な事が分かれば、新しいサイエンスにもつながることになる。そこで最新のシミュレーションツールである GEANT4 を用いて、EGRET のデータの再解析、GLAST における解析ツールの開発、EGRET の応答関数の再評価を目的として、我々は EGRET 検出器のシミュレーターの開発を行っている。本論文では特に、GEANT4 によるモンテカルロシミュレーターの物理座標で記述された、スパークチェンバー中のヒット情報を EGRET 実機のデータ形式のワイヤ番号に置き換える Digitization のアルゴリズムの実装、ヒット情報を元に電子陽電子対を認識し γ 線の方向を求めるリコンストラクションのアルゴリズムの実装を行った。

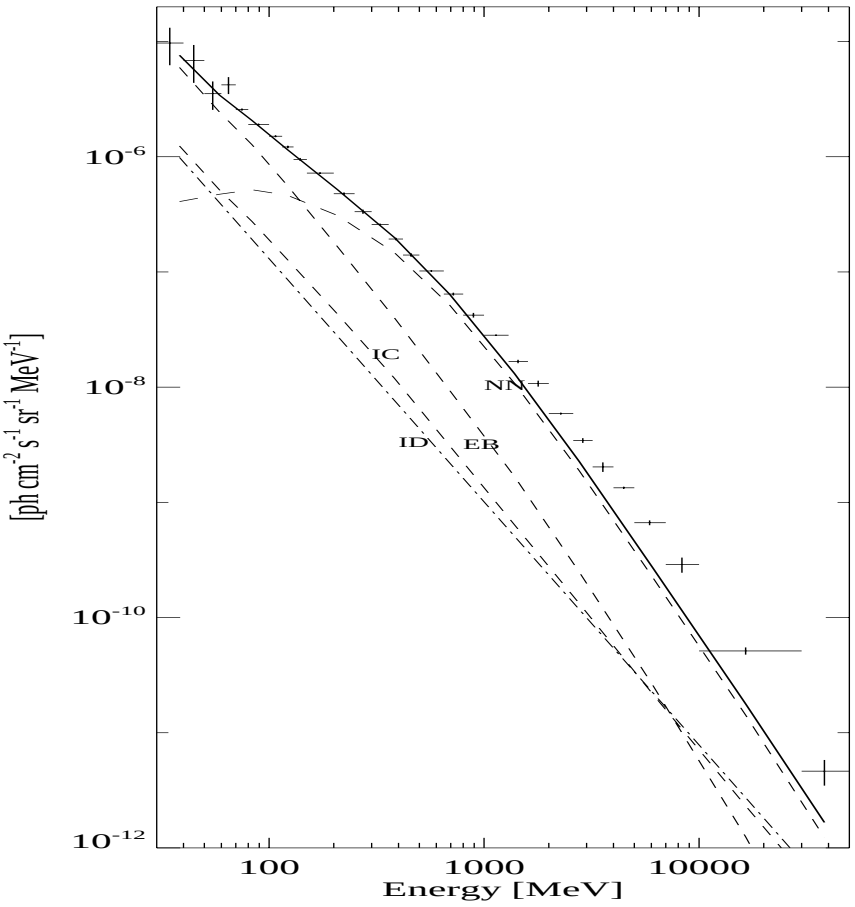

図 2.7: EGRET で観測された、銀河面拡散 γ 線放射のエネルギースペクトル (Hunter et al. 1997)。十字がデータ点、実線がモデル関数であり、各々の成分の寄与を点線で示してある。このうち ID が銀河系外からの拡散 γ 線放射、EB が電子の制動放射による成分、IC が逆コンプトン散乱、NN が陽子と銀河系内物質との相互作用による γ 線放射を示す。

第3章　EGRETシミュレーターの開発(I)

前節で述べた目的を達成するために我々はEGRETのシミュレーターの開発を行ってきた。このうち検出器ジオメトリの記述、および初期の応答関数の再評価は、当研究室の先輩である水嶋、河嶋によって進められてきたものであり、その後を受けてDigitization、リコンストラクションプログラムの開発や詳しい応答関数の評価を、本研究で行った。シミュレーターの全体像、検出器ジオメトリの概要とこれまでなされてきた応答関数の評価をこの第3章で述べる。Digitization、リコンストラクションプログラムの詳細と、応答関数の詳しい評価は第4章で述べる。

3.1　EGRETシミュレーターの全体像

シミュレーションとは、計算機を用いて天体からの信号や実際の検出器での反応を模擬(シミュレート)する作業のことである。本研究で開発しているEGRETシミュレータの全体像を図示すると図3.1のようになる。まず粒子生成部において、γ線天体からの信号や宇宙線バックグラウンド、ビーム試験における入射粒子を生成する。この入射粒子とEGRET検出器との反応は、GEANT4を用いたモンテカルロシミュレーターでシミュレーションしてやる。こして得られた検出器でのヒット情報をもとに、実機と同じトリガ条件判定を行い、条件を満たした事象における検出器でのヒット情報(物理座標で記述)を実機の出力と同じ離散的な数値に直してやるのがDigitizationである。つまりDigitizationの出力は実際の観測データを模擬としたものとなる。この情報をもとにγ線到来方向などを求めるのがリコンストラクション(再構成)と呼ばれる作業で、シミュレーションデータをリコンストラクションした結果と、実データのリコンストラクション結果を比較することで、観測したγ線天体のエネルギースペクトルや空間分布を求めることが出来る。またビーム試験のシミュレーションの場合は、入射粒子のエネルギーや方向が既に分かっているので、モンテカルロシミュレーター(検出器ジオメトリ)やDigitization、リコンストラクションが正しく実装されているかの確認を行うことができる。

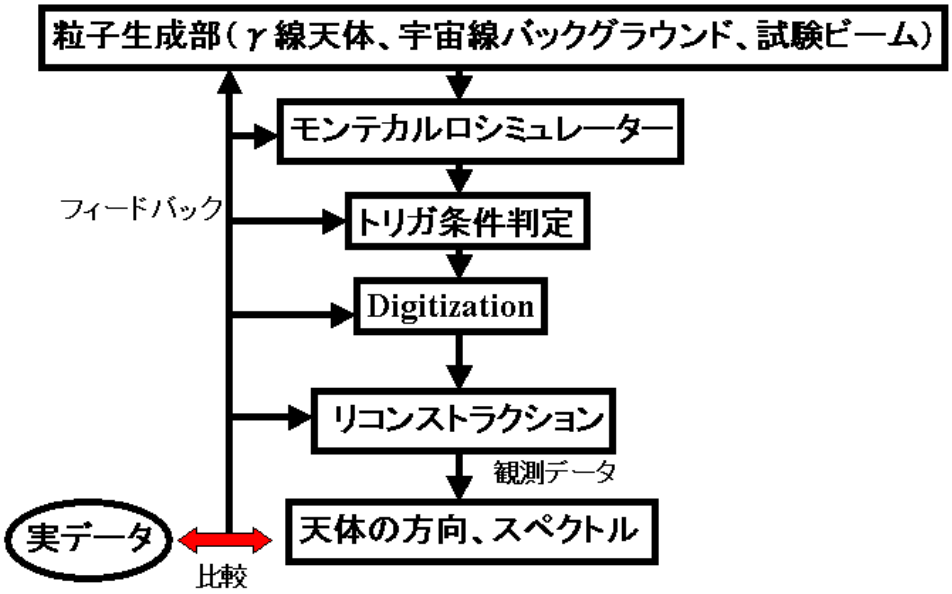

図 3.1: EGRET シミュレーターの全体像

3.2 シミュレーター中の検出器ジオメトリの概要

EGRET は、2.1.2 章で既に述べたように、荷電粒子事象を落とすための Scinti-Dome と呼ばれる部分、γ 線の方向を測るための飛跡検出部であるスパークチェンバー、γ 線のエネルギーを測るカロリーメーター、スパークチェンバーにかける高圧、データ取得のトリガーを生成する TOF(Time Of Flight) シンチレータ、およびエレクトロニクスや支持構造体などから構成される (図 2.2 参照)。我々は GEANT4 を用いて、この EGRET 検出器を必要に応じて簡単化した上で、可能なかぎり実装した。実装したジオメトリの概略を図 3.2 に示す。Scinti-Dome、スパークチェンバーといった放射線に感度を持つ領域 (アクティブな領域) だけでなく、主要な支持構造体 (パッシブな領域) も実装していることが分かる、以下に各々のコンポーネントについて詳しく述べる。

Scinti Dome

宇宙空間から降りそそぐ宇宙線は、大部分が荷電粒子であり、宇宙 γ 線の観測のためにははこの荷電粒子事象を効率良く落とす必要がある。この目的のため、プラスチックシンチレーターで主検出器を覆うことが広く行われている。プラスチックシンチレーターは密度および原子番号が小さいため、欲しい信号である γ 線との反応が最小限に押さえられる上、減衰時定数が短い (数 ns 程度) ため、極めて高いレー

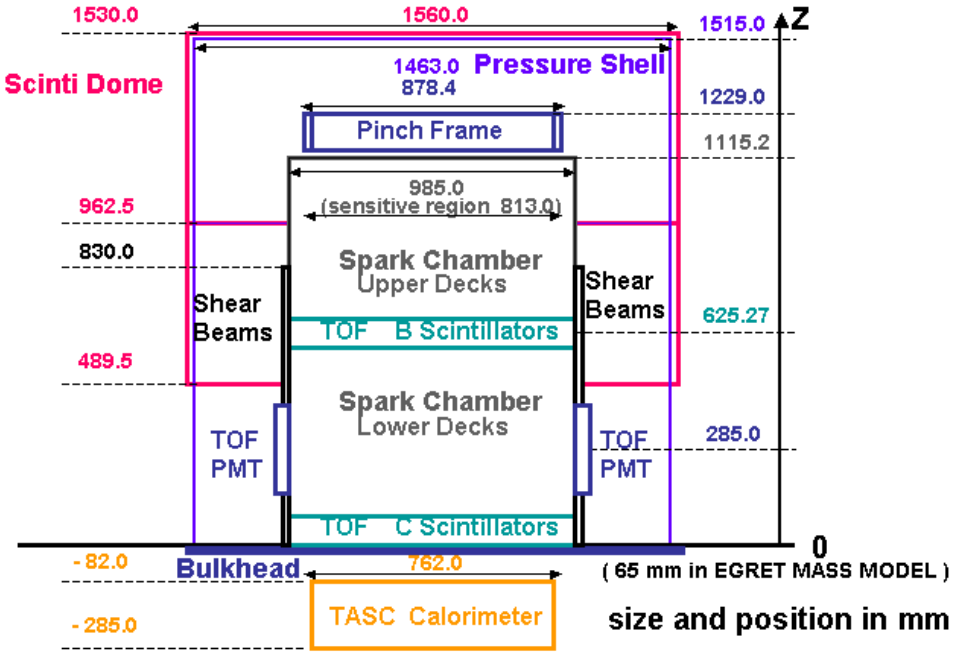

図 3.2: EGRET 検出器シミュレーターにおける検出器ジオメトリの概要

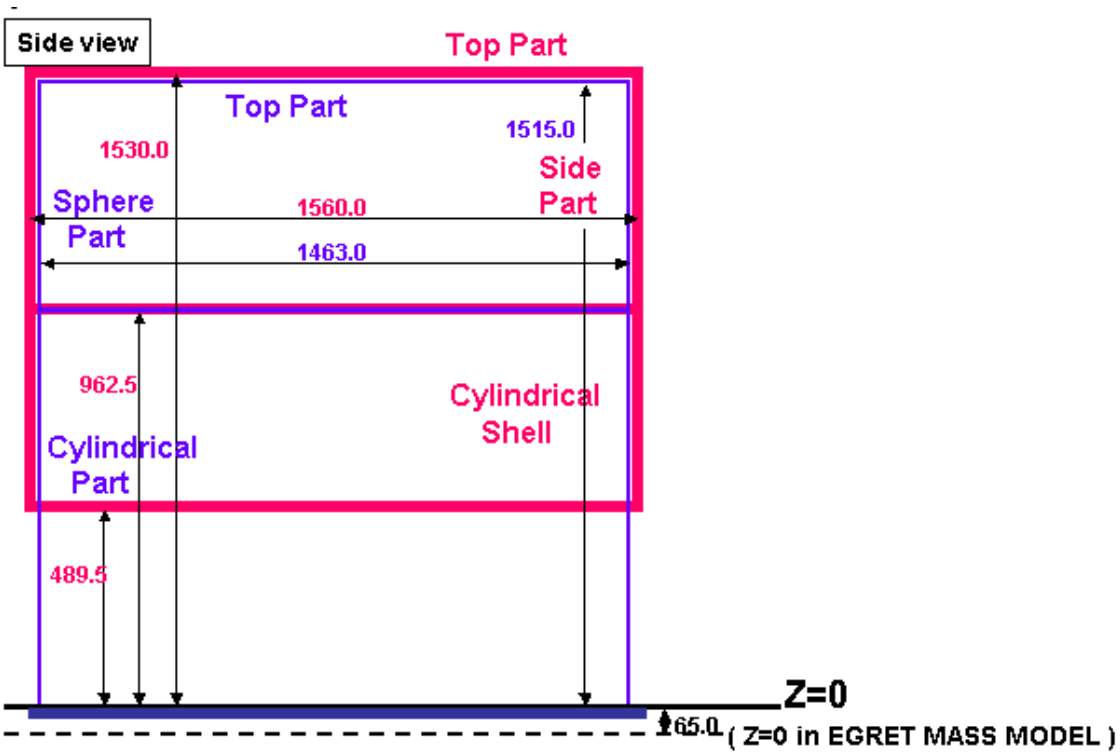

図 3.3: シミュレーター中の Scinti Dome のジオメトリ

トで降り注ぐ荷電粒子からの信号を、パイルアップなど起こさずに処理できるからである。このためEGRETでも、大きな一枚のプラスティックシンチレータ(Scinti Dome)で、ガススパークチェンバーをすっぽり覆った構造をしている。実際のScinti Domeは図2.2で分かるように、円柱状の部分と湾曲した部分がつながった一枚のプラスティックシンチレーターで構成されているが、この形状を完全にシミュレーターで再現するのは難しいので、円柱で近似を行った。円柱のサイズは、高さ1040.5 mm、厚さ20.0 mm、半径780.0 mmである(図 3.3 参照)。将来の拡張のため、EGRETの実機では曲げられた形状をしている"Side part" と実機でも円柱状の"Cylindrical part" に分けて実装している。またスパークチェンバーのガスを封入する圧力容器はScinti Domeのすぐ内側に位置し、厚さ3.5 mmのアルミでできている。

スパークチェンバー

スパークチェンバーは、電子・陽電子の飛跡、ひいてはγ線の到来方向を決定するための、いわばEGRET検出器の核となる部分である。スパークチェンバー全体は、圧力容器の中におかれ、Neガスが封入されている。γ線が対生成を起こしてできた、電子・陽電子がスパークチェンバー中を走る際にこのガスを電離し、TOFシンチレーター(後述)がトリガーとなって高電圧をかけることで火花放電が起こる。チェンバー中に張られたワイヤで火花の場所を記録することで電子陽電子対の飛跡、ひいてはγ線の到来方向を測るのが、スパークチェンバーの役割である。スパークを記録するワイヤの間隔は約0.813 mmで、これが横方向に992本張られており、感度を持つ領域は約80.6 cmとなる。x方向を測るワイヤとy方向を測るワイヤの上下間隔は4.0 mmであり、このx方向、y方向を測るワイヤ群の組が縦に36層並べられた構造をしている。層と層の間(z方向)の間隔は、TOFシンチレータの上側と下側とで異っており、TOFシンチレーターの上の領域では約1.67 cm、下の領域では約11.1 cmである。γ線が電子陽電子対生成を起こすためのコンバーターとしては、原子番号の大きいタンタル(Ta)が使われており、横方向のサイズは813.0 mm×813.0 mm、厚みは0.831〜0.109 mmである。このコンバーターは、スパークチェンバーの第2層から第28層までにおかれており、TOFシンチレータの下側には置かれていない。これらのジオメトリを図 3.4 に示す。図から分かるように、様々な支持構造体も可能な限り実装してある。

カロリーメーター

カロリーメーター部は、電子・陽電子対のエネルギー、ひいては入射γ線のエネルギーを測る部分であり、全エネルギーをここで吸収する必要がある。そのため一

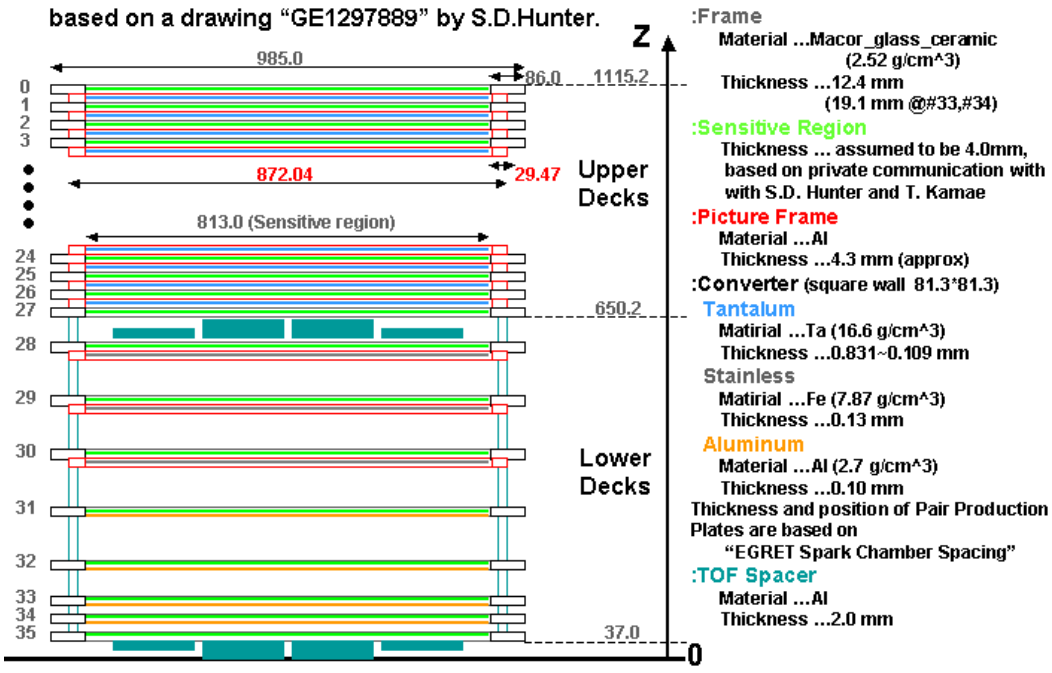

図 3.4: シミュレーター中のスパークチェンバーのジオメトリ

般には厚い無機シンチレーターが使われる。そして EGRET では最も代表的かつ光量の多い無機シンチレータである NaI(Cl) が用いられた。実際のカロリーメーターは、斜め方向の γ 線をとらえるためわずかに湾曲しているが、ここでは横方向 762.0 mm×762.0 mm、厚さ 203.0 mm なる直方体で近似した。

TOF シンチレーター

TOF シンチレーターは、粒子の走った方向が下向きか上向きかを判定し、スパークチェンバーにかける高電圧、およびイベント取得のトリガーを生成する、重要な部分である。これは $4\times4 = 16$ 枚のプラスティックシンチレータが 2 セットからなり、図 3.5 に示されているように上部の TOF シンチレーターはスパークチェンバーの 28 層目の下に、下部の TOF シンチレータは 36 層目の下に位置する。各々のサイズは、XY 方向の長さが 200 mm, 厚さが 6.4 mm である。

TOF シンチレータの置かれている位置、および番号付けを図 3.5 に示す。上下の 16 枚ずつのタイルの組合せ 256 通りのうち、z 方向に平行、および隣接するタイル同士でヒットがあった場合の 96 種類の組合せをトリガー条件としている。より詳しく説明すると、以下のようになる。

- 4 つの角の場合は、並行ないしは辺で隣接する組合せの 3 通り。例えば上部の

14 番と、下部の 14 ないしは 13 ないしは 24 番

- 8 つの辺の場合は、並行ないしは、隣接する組合せの 6 通り。例えば上部の 24 番と、下部のの 14、24、34、13、23、33 番のどれか
- 中央の 4 つの場合は、並行ないしは隣接する組合せの 9 通り。例えば、上部 23 番と、下部の 13、24、34、13、23、33、12、22、32 のどれか

我々のシミュレーターにおいては、TOF シンチレーターのヒットが起きた時刻も記録しており、かつ、上で述べた 96 種類の組合せのいずれかが満たされた時にトリガー条件が満たされたとみなしている。これは、実機の条件と同じものである。

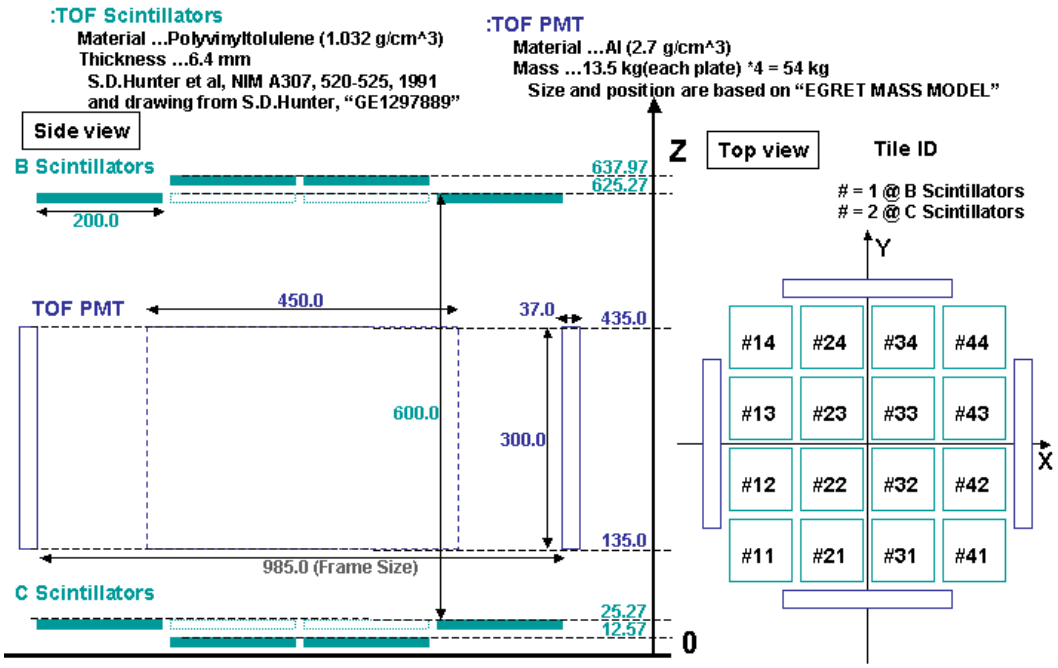

図 3.5: シミュレーター中の TOF シンチレーターのジオメトリ

シミュレーターの出力フォーマット

EGRET シミュレーターの出力は、人間が目で見てすぐに意味がわかり、またどのようなツールでも処理しやすいように、バイナリでなくテキストデータの形で出力させることとした。その例を図 3.6 に示す。ここで “EventNo” の行がイベント番号を示し、“Primary Particle” 以下の行で入射粒子の情報、具体的には粒子の種類、質量、三次元運動量、および粒子を発生させた三次元の座標を記録している。続いて”ScintiDome”、“Spark Chamber”、“TOF” 以下の行で各々の検出器でのヒット情

報、具体的には粒子の種類、運動エネルギー、検出器に落としたエネルギーとステップの長さおよび三次元位置 (ステップの開始位置と終了位置)、そして反応プロセス名が記録されている。また検出器の種類に応じて、適切な ID 番号も記録している。例えば、スパークチェンバーであれば何番目の層でエネルギーを落としたかを記録している。カロリーメーター部は、単一の NaI で出来ているので、落とされたエネルギーの総和を記録するだけである。このシミュレーターからの出力に基づき、トリガー判定や Digitization、リコンストラクション (後述) を行うことで、EGRET 検出器の応答関数を再評価してやるのが、本研究の内容である。

```
### EventNo: 0
## Primary Particle
# ParSpc, Mass(MeV/c2), PXYZi(MeV/c), XYZi(mm)
gamma 0 0 0 -10000 0 0 2000
## ScintiDome
# ParSpc, TrackID, parentID, KinE(MeV), DepE(MeV), StepLength(mm), XYZi(mm), XYZo(mm), ProcName
e- 619 610 1.12615 0.0762123 0.579508 -704.813 -307.695 971.796 -705.182 -307.785 972.226 eIoni
e- 619 610 0.934046 0.192099 1.20585 -705.182 -307.785 972.226 -705.792 -308.688 972.683 eIoni
e- 619 610 0.780619 0.153428 0.982069 -705.792 -308.688 972.683 -705.843 -309.639 972.588 eIoni
e- 619 610 0.655012 0.125607 0.807708 -705.843 -309.639 972.588 -706.247 -310.234 972.885 eIoni
## Spark Chamber
# FrameID, ParSpc, TrackID, parentID, KinE(MeV), DepE(MeV), StepLength(mm), XYZi(mm), XYZo(mm), ProcName
0 e+ 3 1 9081.65 0.000914679 4 -0.0159402 0.000302078 1111 -0.0162021 0.000328668 1107 Transportation
1 e+ 3 1 8887.18 0.000901302 4 -0.0152797 0.000352284 1094.24 -0.0144172 0.000323304 1090.24 Transportation
2 e+ 3 1 8886.71 0.000383007 4 -0.0107769 -0.000149178 1077.47 -0.00934959 -0.00038787 1073.47 Transportation
3 e+ 3 1 8886.58 0.00046579 4 -0.00496668 -0.00235157 1060.71 -0.00366268 -0.00336061 1056.71 Transportation
4 e+ 3 1 8886.42 0.00298459 4 0.00349101 -0.00579563 1043.95 0.00659077 -0.00629987 1039.95 Transportation
5 e+ 3 1 8886.28 0.0006357 4 0.0192287 -0.00945473 1027.18 0.0240127 -0.0109646 1023.18 Transportation
## TOF
# TileID, ParSpc, TrackID, parentID, KinE(MeV), DepE(MeV), StepLength(mm), XYZi(mm), XYZo(mm), HitTime(ns), ProcName
123 e+ 9755 18 81.4022 0.731304 5.20712 -2.25019 6.74111 637.97 -2.3869 7.28912 632.794 4.56178 eIoni
123 e+ 9755 18 81.2345 0.167644 1.23066 -2.3869 7.28912 632.794 -2.40555 7.41444 631.57 4.56588 Transportation
223 e+ 9755 18 79.4954 0.9208 6.4137 -6.90232 54.869 12.57 -6.91547 55.2684 6.17 6.65845 Transportation
223 e- 9763 9755 0 0.0730353 0.0797633 -6.91488 55.2504 6.45797 -6.93206 55.2108 6.4438 6.6578 eIoni
123 e- 9756 9755 0 0.0532334 0.0458663 -2.26577 6.86106 636.892 -2.24575 6.8534 636.886 4.5482 eIoni
123 e- 9754 18 35.1067 0.425707 2.61491 -6.58268 3.45723 637.97 -6.92564 3.49077 635.378 4.55403 eIoni
## TASC
# DepE(MeV)
7697.6
### End of the event
```

図 3.6: EGRET シミュレーターの出力

3.3 シミュレーターのこれまでの評価

こうして作られたモンテカルロシミュレーターが正しく動くことを確認するために、EGRET のビーム試験 (D. Thompson et al. 1993, ApJS 86, 629) で得られた有効面積の結果と比較を行った。有効面積とは、γ 線を検出する実効的な面積のことで、これが大きいほど性能の良い検出器といえる。具体的には、幾何学面積に、電子陽電子を対生成する割合、トリガー条件を満たす割合、γ 線事象と認識される割合をか

けあわせたものである。参照データとして、NASA が正式に配布しているキャリブレーションファイル (ftp://cossc.gsfc.nasa.gov/pub/data/egret/calib/sarfil01.fits) を用いた。このファイルは、有効面積および角度分解能 (Point Spread Function;PSF) が入射 γ 線のエネルギーおよび角度 (天頂角と方位角) の関数として格納されている。最も単純な場合として、垂直入射の有効面積を、我々のシミュレーションの結果とキャリブレーションデータとで比較を行った。

これまで述べてきたように、シミュレーターは γ 線と EGRET 検出器との相互作用、および TOF シンチレータによるトリガーをシミュレートする。EGRET による実際の γ 線観測においては、スパークチェンバー中のヒット情報 (γ 線が対生成して出来た電子陽電子対の軌跡を反映) をみて、電子陽電子対のつくるトラックを再構成 (リコンストラクション) し、γ 線事象とみなせるか否かの判定を行っている。各エネルギーの γ 線に対し、どのくらいの割合で γ 線事象と判定されるかの割合、すなわちリコンストラクションの効率は、リコンストラクションのアルゴリズムに依存する。この効率は既に EGRET チームによって、ビーム試験のデータを用いて求められているので、この効率のテーブルを EGRET チームから入手し用いた。これにより、実データに対するのと同じ効率の値を我々のシミュレーションにも用いた。ビーム試験の行われた 35 MeV、 60 MeV、 100 MeV、 200 MeV、 500 MeV、1 GeV、 3 GeV、 10 GeV の各エネルギーの γ 線を、シミュレーションで垂直方向から一様に検出器に照射し、トリガー条件を満たした事象を数え上げ既知の『リコンストラクションの効率』をかけることで、EGRET の有効面積を γ 線のエネルギーの関数として求めた。

結果を図 3.7 に示す。この図で+印がキャリブレーションファイルから得た値 (ビーム試験の結果)、○印が我々のシミュレーションで得られた値である。高いエネルギーの γ 線を打ち込んだ場合は、カロリーメーター (NaI シンチレータ) で電磁シャワーが発達し、一部の数 10 から数 100 keV 程度の X 線が上方向にもどる「バックスプラッシュ」と呼ばれる現象により、Scinti Dome である程度のエネルギーが落とされ、荷電粒子事象として棄却されることがある。このため有効面積は Scinti Dome のエネルギースレショルドに依存するので、スレショルドをいくつか変えて有向面積を調べた。9 keV および 100 keV にスレショルドを設定したときの有効面積を、各々赤および青色の○印で示してある。

この図から分かるように高いエネルギー側の有効面積はスレショルドの値に大きく依存し、100 keV を仮定したとき、ビーム試験のデータをよく説明する。この値は、EGRET の実機で設定されたスレショルドと同じ値である。また、電子陽電子対によるトラックが発達しにくいため、有効面積が次第に下がる低いエネルギー側もよく再現しており、我々が実装したシミュレーターおよびトリガ条件は正しいことが確認された。よってビーム試験の無いエネルギー領域の応答関数を、適当な関

数で内挿、外挿することなしに、我々のシミュレーターを用いてじかに求めてやることは、有効であるといえる。

以上のようにEGRET検出器におけるγ線の反応およびトリガーをシミュレートする作業は、はぼ完成した。応答関数の再評価のために次に行うべきことは、γ線と判定される事象の数と、トリガー条件を満たす事象の数の比、すたわち「リコンストラクションの効率」を、任意のエネルギー、角度で知ることである。しかしEGRETチームによって測られた既知の効率は、ビーム試験の行われた限られたエネルギーと角度についてのみしか分かっていない。そこでEGRETの実際の観測で用いられているSAGE(Search and Analysis of Gamma-ray Events)と呼ばれるリコンストラクションのアルゴリズムを我々自身でシミュレーションの出力に適用することとした。プログラム自体はEGRETチームによりFortran言語を用いて書かれているが、このプログラムをそのまま利用するには互換性のあるFortranコンパイラを準備し、データフォーマットを細部まで一致させねばならず、これは簡単な作業ではない。また、たとえプログラムを見かけ上動かせたとしても、アルゴリズムの中身が理解出来ていないと、思い通りの動作をしないときに問題が何処にあるのかを突き止めることも難しいであろう。そこで我々はプログラムをそのまま動かすのではなく、EGRETチームから入手したFortranコード、及びドキュメントを元に、アルゴリズムを我々自身で実装することとした。言語としては、テキスト、数値処理の機能の豊富さ、開発効率の良さを考え、スクリプト言語であるPythonを用いることとした。また実際にシミュレーション出力にリコンストラクションをあてがうには、検出器のヒット情報(物理座標で記録)を、ワイヤ番号(離散的な番号で記録)に置き換える、Digitizationと呼ばれる作業も必要である。このDigitizationも合わせて行った。

次章でこのDigitizationとリコンストラクションの中身について、詳しく述べていく。

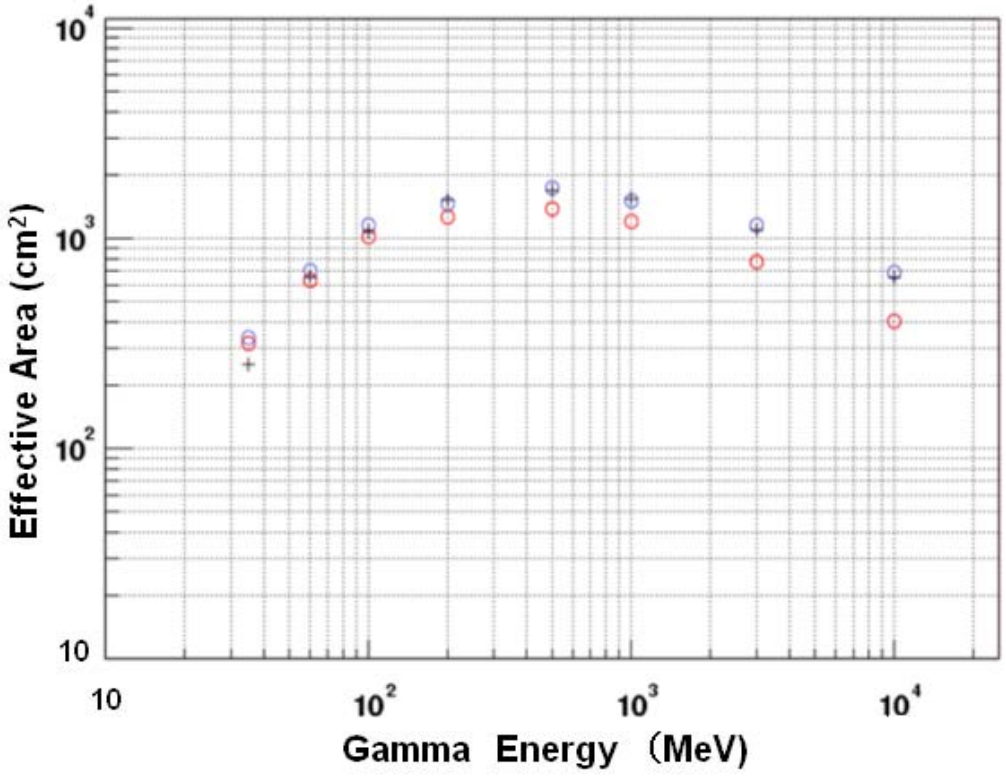

図 3.7: ビーム試験とシミュレーションによる EGRET の有効面積の比較。十字がビーム試験のデータを、丸印がシミュレーションを結果を示す。ここで青の丸印が Scinti Dome でのエネルギースレッショルドが 100 keV の場合、赤の丸印がスレショルドが 9 keV の場合である。

第4章　EGRETシミュレーターの開発(II)

4.1　Digitizationプログラムの開発

Digitization は、物理座標で記録された、シミュレーションによるヒット情報を、ワイヤの番号に置き直す作業であり、いわばEGRET の実機のデータ形式を模擬する作業である。電子陽電子対の再構成 (リコンストラクション) はこの Digitization の出力に対して行われることとなる。3.2 章で述べたように、EGRET においては、0.813 mm 間隔で 992 本のワイヤが張られており、このワイヤの番号でヒットのあった位置を記録している。よって Digitization としては逆の作業、つまりヒットの位置をワイヤの番号に直す作業を行ってやればよい。座標値の小さいほうから大きいほうへと順に、ワイヤの番号を、1 から 992 番と番号付けをしてやり、各ワイヤの位置からワイヤ間隔の半分、つまり ± 0.4065 mm の範囲にあったヒットを、そのワイヤのヒットとした。また我々の GEANT4 シミュレーターにおいては、全 992 本のワイヤの中心、つまり 496 本目と 497 本目のワイヤの真中を 0 と決めた。座標値とワイヤの番号の関係は、以下の式で表される。ここで wireDistance はワイヤの間隔を表し 0.813 mm、L は 992 番目のワイヤとみなされる最大の領域の座標で 403.248 mm、X_a はヒットの物理座標である。座標値が 0 番目のワイヤよりも小さい場合、ないしは 992 番目のワイヤよりも大きい場合には、不感領域であるとしてヒットを捨てることにした。

まずある wireNumber(ワイヤ番号) のヒットであるときは, 次のような不等式が成り立つ。

$$-L + (\text{wireNumber} - 1) \times \text{wireDistance} \leq Xa \leq -L + (\text{wireNumber}) \times \text{wireDistance}$$

これを変形すると

$$(L + X_a)/\text{wireDistance} \leq \text{wireNumber} \leq (L + X_a)/\text{wireDistance} + 1$$

ここで $(L + X_a)/\text{wireDistance}$ の小数部分を切捨てると以下のように書ける。

$$\text{wireNumber} = (L + X_a)/\text{wireDistance} + 1$$

この式を用いて物理座標をワイヤ番号に置き直した。シミュレーションにおいては、各スパークは始点と終点を持つステップとして表される。この始点と終点の座標をワイヤ番号に直し、その間にあるワイヤ全てでスパークが起きるとして、カウントしていった。複数のスパークでワイヤ番号が重複した場合は、重複を除いて記録することとした。シミュレーターの出力、および対応する Digitization 後の出力を、図 4.1 に示す。シミュレーター出力のうち、Digitization で用いられるのは、“EventNo:” で示されるイベントの番号、“FrameID” で示される、スパークチェンバーの何層目かの番号、XYZi(始点), および XYZo(終点) で示されるスパークの物理座標である。

Digitization の出力 (図 4.1) はより簡略化されており、最初の行ではイベントの最初の行であることを示す 0 の後にイベントの番号と入射粒子の名前、質量、三次元運動量 (全て MeV 単位) と粒子を発生させた三次元位置 (mm) が記録され、続く行では全 36 層の内の何番目のワイヤ層かを示す番号、X 方向を測るワイヤ群か Y 方向を測るワイヤ群かを示すフラグに続いて、スパークのあったワイヤの数が示される。これが 1 ないしはそれ以上のときは、スパークのあったワイヤの番号を全て出力している。最後の”-1” は、後のリコンストラクションプログラムで処理しやすいように、イベントの最後の行であることを示すフラグである。

```
### EventNo: 5
## Primary Particle
# ParSpc, Mass(MeV/c2), PXYZi(MeV/c), XYZi(mm)
gamma 0 0 0 -1000 0 0 2000
## ScintiDome
# No Hit
## Spark Chamber
# FrameID, ParSpc, TrackID, parentID, KinE(MeV), DepE(MeV), StepLength(mm), XYZi(mm), XYZo(mm), ProcName
5 e+ 3 1 261.672 0.000401613 4.0003 0.0587905 0.0537483 1028.08 0.094894 0.0866287 1024.08 Transportation
6 e+ 3 1 261.497 0.00218467 4.00074 0.241742 0.221588 1011.32 0.298955 0.273052 1007.32 Transportation
7 e+ 3 1 261.365 0.000772476 4.00033 0.420754 0.43238 994.56 0.43919 0.480615 990.56 Transportation
8 e+ 3 1 261.166 0.0011339 4.00047 0.54257 0.617323 977.8 0.590079 0.656407 973.8 Transportation
9 e+ 3 1 234.494 0.00106005 4.00021 0.717466 0.758008 961.03 0.750742 0.782557 957.03 Transportation
11 e+ 3 1 234.151 0.00132539 4.00009 0.941607 1.00673 927.51 0.94963 1.03304 923.51 Transportation
12 e+ 3 1 233.977 0.000623411 4.00078 0.962127 1.20519 910.74 0.963037 1.28412 906.74 Transportation


25 e- 2063 1476 3.13616 0.00139774 4.49879 8.09875 -12.2721 692.82 10.1405 -12.5298 688.82 Transportation
26 e- 2063 1476 2.96485 0.00144819 4.85088 15.4679 -9.16886 676.06 16.7594 -6.74854 672.06 Transportation
27 e- 2063 1476 2.78531 0.00179198 5.31396 17.7544 -8.24507 659.3 16.9781 -11.655 655.3 Transportation
## TOF
# TileID, ParSpc, TrackID, parentID, KinE(MeV), DepE(MeV), StepLength(mm), XYZi(mm), XYZo(mm), HitTime(ns), ProcName
122 e+ 3 1 229.041 0.939683 6.41064 -8.72042 -10.6185 637.97 -9.01051 -10.8419 631.57 4.56625 Transportation
222 e+ 3 1 219.985 0.780872 4.92018 -41.3693 -31.1865 12.57 -41.6572 -31.358 7.66128 6.65138 eIoni
222 e+ 3 1 219.767 0.217303 1.4937 -41.6572 -31.358 7.66128 -41.7212 -31.4136 6.17 6.65636 Transportation
222 e- 28 3 0.192455 0.0791606 0.288662 -41.6572 -31.358 7.66128 -41.8686 -31.2278 7.56337 6.65245 eIoni
222 e- 28 3 0.123548 0.0689068 0.22118 -41.8686 -31.2278 7.56337 -41.9991 -31.3502 7.52698 6.65321 eIoni
222 e- 28 3 0.0535942 0.0699537 0.158301 -41.9991 -31.3502 7.52698 -42.0732 -31.3382 7.47857 6.65362 eIoni
222 e- 28 3 0 0.0535942 0.0464438 -42.0732 -31.3382 7.47857 -42.0842 -31.3184 7.47787 6.6538 eIoni

132 e- 2080 2063 0.0323466 0.0672208 0.128419 13.411 -28.0119 634.025 13.3661 -28.0196 634.08 4.61453 eIoni
132 e- 2080 2063 0 0.0323466 0.019105 13.3661 -28.0196 634.08 13.3596 -28.0141 634.082 4.61461 eIoni
## TASC
# DepE(MeV)
869.932
### End of the event
```

```
0 5 gamma 0 0 0 -1000 0 0 2000
0 x 0
0 y 0
1 x 0
1 y 0
2 x 0
2 y 0
3 x 0
3 y 0
4 x 0
4 y 0
5 x 1 497
5 y 1 497
6 x 1 497
6 y 1 497
7 x 1 497
7 y 1 497
8 x 1 497
8 y 1 497
9 x 1 497
9 y 1 497
10 x 1 497
10 y 1 496
11 x 2 497 498
11 y 2 496 498
…
31 x 3 361 463 478
31 y 5 103 105 106 468 514
32 x 4 370 371 455 475
32 y 4 26 29 464 521
33 x 2 451 473
33 y 2 461 525
34 x 3 449 472 473
34 y 2 460 526
35 x 2 448 472
35 y 4 459 460 527 528
-1
```

図 4.1: (左)EGRET シミュレーターの出力データ (右)Digitization 後の出力データ

4.2 リコンストラクションアルゴリズムの概要

EGRET のリコンストラクションは、SAGE(Search and Analysis of Gamma-ray Events) と呼ばれるプログラムが担っており、EGRET チームにより Fortran で書かれた。我々はこのソースコード、およびドキュメントを元に、アルゴリズムを実装した。プログラムの開発はいきなり完全なものを作ろうとするのは逆効果なことが多く、実際に走らせてフィードバックをかけることが何よりも重要であるので、アルゴリズムが複雑なところは一部単純化を行っている。本節ではリコンストラクションのアルゴリズムの概要を述べる。続いて次節以降で、具体的なアルゴリズムの説明、ソースコードの主要部の説明、リコンストラクションを走らせた結果の図示などを行う。

SAGE は数 10 のファイル、関数から構成されているが、EGRET チームによるドキュメントでは 10 個の phase に分けて説明されている。我々もこれにならい、各々の phase に対応するプログラムを phase0, phase1, ..., phase9 というように名付けて開発した。以下、各々の phase で行われる作業を説明する。

スパークチェンバー中での γ 線の反応の様子を図 4.2 に示す。まず phase0 において、隣接するスパークのあったワイヤを一つにまとめる作業を行う。続いて phase1 で、スパークの数が極端に少なく、明らかに γ 線事象とはみなせないであろう事象を棄却している。その上で電子陽電子対のトラックの再構成 (リコンストラクション) を行う phase2 以降に移る。phase2 では最初の準備段階として、トラックの先頭とみなしうる 3 つのスパークのあったワイヤの組 (トリプレットと呼ばれる) を探しだす。phase3 でトリプレットの有無やその特質から、γ 線とみなせないであろう事象を落としている。phase4 でトリプレットを電子陽電子対の飛跡であるトラックに延長し、phase5 で最も素性の良いトラック (primary トラック) と 2 番目に素性の良いトラック (secondary トラック) の選定を行う。この primary トラックと secondary トラックから、入射 γ 線の方向を求めることになる。phase6 では primary と secondary トラックの分かれる場所 (vertex) の位置をみることで、より適切な secondary トラックが他に無いかの確認を行う。phase7 は X 座標を測るワイヤ群での primary トラック、secondary トラックと、Y 座標を測るワイヤ群での primary トラック、secondary トラックとの組合せを探す作業である。phase8 はイベントの様々な特質を計算し、最後に phase9 で γ 線事象とみなせるか否かの判定、および γ 線の到来方向の計算を行う。

このうち phase6 は結果に大きな影響を与えないことが知られており (D.Thompson et al. 1993, ApJS 86, 629)、また phase8 はフラグを立てるだけの箇所なので、現段階では省略している。次節で、各々の phase のアルゴリズムの詳細と、ソースコードの主要部を述べ、またプログラムをデータにあてがった結果の例を紹介する。一

部のアルゴリズムは単純化しており、これについては各々の箇所で述べる。上で述べた作業をまとめると以下のようになっている。

- phase 0: 隣接するスパークのあったワイヤを一つにまとめる
- phase 1: 簡単なセレクションでイベントを捨てる。
- phase 2: トリプレット (track の先頭になりうる、3 つのスパークの組) を探す
- phase 3: セレクションをかけてトリプレット、イベントを落とす
- phase 4: トリプレットをトラックに拡張
- phase 5: primary トラックと secondary トラック (あれば) を選定
- phase 6: vertex を探し、必要なら secondary トラックを選定し直す。
- phase 7: X 座標を測るワイヤ群、Y 座標を測るワイヤ群との間での primary トラック、secondary トラックの対応の確認と関連付け
- phase 8: イベントの特質を出す
- phase 9: γ 線事象か否かの判定を行い γ 線の到来方向を求める

4.3 リコンストラクションプログラムの開発

前節で述べたように EGRET のリコンストラクションのドキュメントと Fortran で書かれたプログラムとをもとに、スクリプト言語 Python を使ってリコンストラクションプログラムの作成を行った。本研究では 4.2 節の 10 個の phase を一部単純化して実装した。具体的に行った内容を以下に示す。

1. 隣接するスパークのあったワイヤを 1 つにまとめる
2. 簡単なセレクションでイベントを捨てる
3. トリプレットを探す
4. トリプレットをトラックに拡張する
5. primary トラックと secondary トラックを選定する

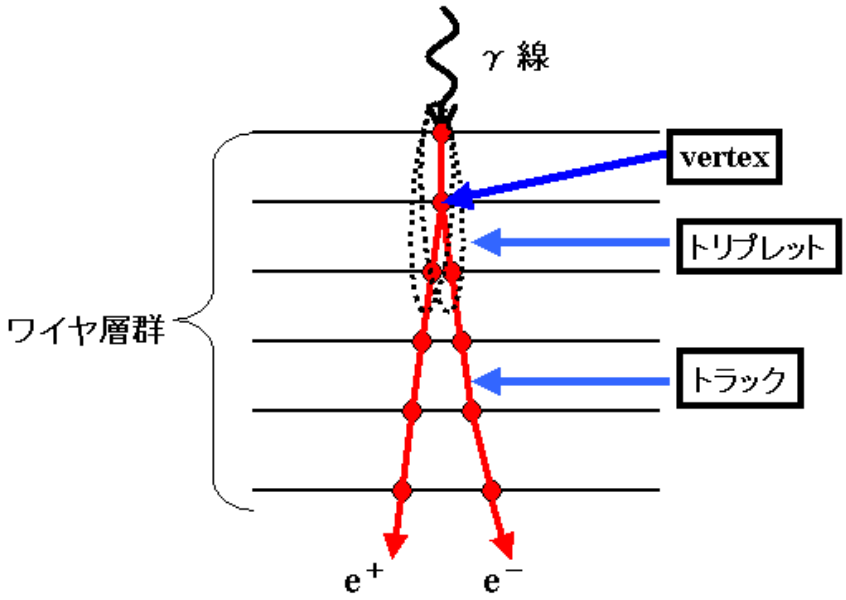

図 4.2: スパークチェンバーにおける、γ 線の反応イメージ。赤の丸印がスパークの有ったワイヤを示す。この例では、e^+ の作るトラックが直線に近く、これが primary トラック、e^- の作るトラックが secondary トラックとなる。

6. X 座標を測るワイヤ群、Y 座標を測るワイヤ群との間での primary トラック、secondary トラックの対応の確認と関連付け

7. γ 線事象か否かの判定を行い γ 線の到来方向を求める

ここで 1. から 5. までは X 座標を測るワイヤ群、Y 座標を測るワイヤ群で別々に処理を行い、6. 以降で両者をまとめて処理していく。以下具体的なアルゴリズムを述べる。

4.3.1 隣接するスパークのあったワイヤを１つにまとめる

ワイヤの横方向の間隔は 0.813 mm と非常に狭くなっており、そのため単一の電子や陽電子が通過しても、複数のワイヤににスパークを残すことがある。そのため隣接した Spark を図 4.3 の様にして１つにまとめ、中心のワイヤの番号で位置を代表させる。

このような操作を図 4.4 のプログラムを使って行った。まず sequence という変数を用意し、これでいくつスパークが連続しているかを数えると共に、連続したスパークをひとまとめにしたワイヤ番号を格納するための newlist というリスト[1]を用意す

[1] リストとは、要素 (項目) が一定順序に並ぶ特性を持ったオブジェクトで、数値、文字列、リスト自身などあらゆるオブジェクトを構成要素とできるものである。

る。squence の値を見て、連続になっていない場合はそのまま、連続したスパークであれば中心のワイヤ番号を求めて、append [2]という操作を用いて newList に格納する。最後のスパークに達したら、この newList を返してやる。この操作を行う前後の出力データの例を図 4.5 に示す。

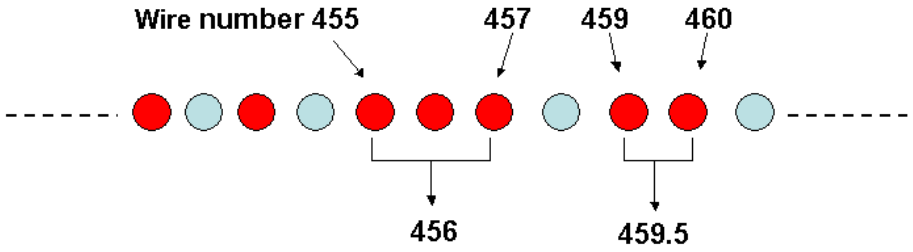

図 4.3: 隣接するスパークのあったワイヤ番号をまとめる作業のイメージ。

4.3.2 簡単なセレクションでイベントを捨てる

X 座標を測るワイヤ群、Y 座標を測るワイヤ群共に MINSPARK 以上のスパークをもたないイベントは捨てる。ここで MINSPARK は、リコンストラクションのパラメータの 1 つで、デフォルトの値は 6 である。以下リコンストラクションパラメータが出てくる時は MINSPARK(6) のように表記することとする。このセレクションを設けるのは、ヒットのあるスパークの数が少ないイベントは γ 線事象でない可能性が高く、またたとえ起源が γ 線であっても方向を正しく求めることができないためである。

図 4.6 のようなプログラムを用いてスパークの数を数え上げた。まず nhitX および nhitY なる変数を用意してそれぞれ X 座標を測るワイヤ群、Y 座標を測るワイヤ群の各層のスパークの数をカウントしてイベントの最後に達したら MINSPARK と比較を行い、nhitX と nhitY の両方が MINSPARK 以上となっていればそのイベントはそのまま出力するようにした。

[2]append とは、リストの末尾に要素を追加する操作を行うものである。

```
if nhit > 1 :
  sequence = 0  # the number of combined wires -1
  for wireID in range(nhit-1):
    # check if the next wire is sequence or not
    if list[1+wireID] == list[wireID] +1:
      sequence = sequence + 1
    elif list[1+wireID] > list[wireID] +1:
      if sequence == 0 : # independet hit
        newList.append(list[wireID])
      else:
        # the combined wire position
        combinedWirePosition = float(list[wireID])-float(sequence)/2
        newList.append(float(combinedWirePosition))
        sequence = 0
      if wireID == nhit-2: # the last wire
        newList.append(list[wireID+1])
  if sequence != 0: # sequence include the last hit
    combinedWirePosition = list[wireID+1]-float(sequence)/2
    newList.append(float(combinedWirePosition))
elif nhit==1:   # the number of hit is one
  newList.append(list[0])

# gives back the combined list of wire number
return newList
```

図 4.4: python 言語を用いた連続するスパークを結合するプログラム

4.3.3 トリプレットを探す

続いて電子陽電子対の飛跡であるトラックを見つける準備として、トラックの先頭の 3 つのスパークの組であるトリプレットを探す。1 つの γ 線から複数の電子・陽電子対が生成されうるので、トリプレットも 3 つ以上になることがある。トリプレットを探す流れを図 4.7 に示す。まずスパークのの当ったワイヤの層のうち、1 番上のワイヤの層のスパーク全てを、トリプレットの先頭 (1st スパーク) の候補とする。続いて 1st スパークのワイヤ層から JUMP2(2) 層まで下のワイヤ層から、後述の条件に従ってトリプレットの 2 番目スパーク (2nd スパーク) を探す。ここで 2nd スパークが見つからなければ、1st スパークのワイヤ層から 1 つ下のワイヤ層に移り、1st スパークを探し直す。2nd スパークが見つかれば、そこから JUMP3(3) 層まで下のワイヤ層から後述の条件に従い 3rd スパークを探す。3rd スパークが見つからないときは、1st スパークの 1 つ下のワイヤ層に移り、再び 1st スパークを探す所からやり直す。

2nd スパークを探す条件は、1st スパークと 2nd スパークを結ぶ直線と鉛直方向

```
0 40 gamma 0 0 0 -1000 0 0 2000
0 x 0
0 y 0
1 x 0
1 y 0
2 x 0
2 y 0
3 x 0
3 y 0
…
25 x 3 497 522 523
25 y 4 475 476 496 497
26 x 2 497 524
26 y 3 473 474 497
27 x 2 498 525
27 y 3 470 471 497
28 x 2 499 528
28 y 3 461 462 497
29 x 3 502 537 538
29 y 2 445 498
30 x 2 505 546
30 y 3 428 429 498
31 x 3 508 555 556
31 y 3 413 414 499
32 x 3 510 564 565
32 y 3 398 399 499
33 x 2 511 570
33 y 3 389 390 500
34 x 2 512 572
34 y 3 386 387 500
35 x 3 512 573 574
35 y 3 383 384 500
-1
```

```
0 40 gamma 0 0 0 -1000 0 0 2000
0 x 0
0 y 0
1 x 0
1 y 0
2 x 0
2 y 0
3 x 0
3 y 0
…
25 x 2 497 522.5
25 y 2 475.5 496.5
26 x 2 497 524
26 y 2 473.5 497
27 x 2 498 525
27 y 2 470.5 497
28 x 2 499 528
28 y 2 461.5 497
29 x 2 502 537.5
29 y 2 445 498
30 x 2 505 546
30 y 2 428.5 498
31 x 2 508 555.5
31 y 2 413.5 499
32 x 2 510 564.5
32 y 2 398.5 499
33 x 2 511 570
33 y 2 389.5 500
34 x 2 512 572
34 y 2 386.5 500
35 x 2 512 573.5
35 y 2 383.5 500
-1
```

図 4.5: (左)Digitization 後の出力データ。(右) 隣接するスパークをひとまとめにするプログラムに通した後の出力データ

```
if re.match(".*gamma.*", line):
  eventLine = line # particle information
  listLines = [] # list to store tracker data
  nhitX = 0  # the number of hit in x face
  nhitY = 0  # the number of hit in y face
 # if the line shows the tracker hit, combine adjacent hits
 if re.match(".*x.*", line):
  listLines.append(line)
  nhitX = nhitX+int(columns[2])
 if re.match(".*y.*", line):
  listLines.append(line)
  nhitY = nhitY+int(columns[2])


 # only events with sufficient number of hits
 # (more than MINSPK) are printed.
 if (columns[0]=="-1"):
  if nhitX >= MINSPK and nhitY >= MINSPK:
    print eventLine,
    for i in range(len(listLines)):
      print listLines[i],
    print line,
```

図 4.6: スパーク数を数えあげるプログラム

のなす角 (つまり天頂角) を θ として

$$\tan\theta \leq 2.0 \tag{4.1}$$

である。また 3rd スパークは、TOF シンチレーターの上か下かでワイヤ層間の間隔が異るので (図 2.2 参照) 判定方法を変えている。TOF シンチレーターの上のワイヤ層に 3rd スパークの候補があるときは 1st スパークと 2nd スパークを結ぶ直線と 3rd スパークのあるワイヤ層との交点が、3rd スパークから見てどれだけ離れているかで判定しており、2nd スパークのすぐ下のワイヤ層であれば 8 ワイヤ以内、2 つ下のワイヤ層なら 15 ワイヤ以内、3 つ下のワイヤ層なら 22 ワイヤ以内となる (図 4.8)。ワイヤ同士の間隔は 0.813 mm、ワイヤ層同士の間隔は 4.0 mm なので、1st スパーク、2nd スパークを結ぶ線が鉛直方向であれば、これと 2nd スパーク、3rd スパークを結ぶ直線のなす角が約 18 度以下ということに対応する。

3rd スパークの候補が TOF シンチレーターの下に有るときは、1st スパーク、2nd スパークを結ぶ線と、2nd スパーク、3rd スパークを結ぶ線のなす角 θ で条件をつけ

ており、3rd スパークが TOF シンチレーターの１つ下ないしは２つ下のワイヤ層にあるときは、

$$\tan\theta \leq \mathrm{TTHMID} \tag{4.2}$$

それより下のワイヤ層なら

$$\tan\theta \leq \mathrm{TTHBOT} \tag{4.3}$$

である。ここで TTHMID、TTHBOT のデフォルトの値はそれぞれ 0.32、0.24 である。

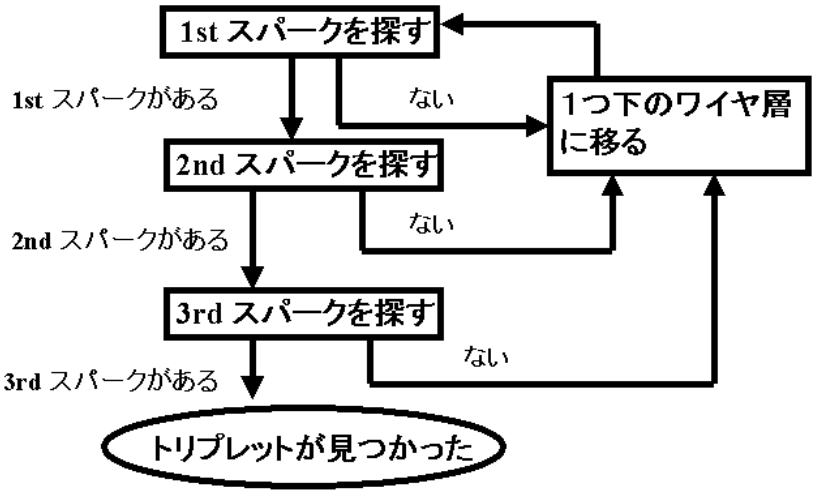

図 4.7: トリプレットを探す作業の流れ

4.3.4 トリプレットをトラックに拡張する

これまでの作業で見つかったトリプレットをもとに、スパークを次々と加えていき、電子陽電子対の飛跡の候補であるトラックに延長する作業をここで行う。

作業の流れ図を図 4.9 に示す。トラックを構成するスパークのうち、一番最後の 3 つのスパークに最小自乗法を適用し、求まった直線と JUMP(4) 層まで下のワイヤ層との交点を求める。この交点との距離が、NWIRE(i) 以内であれば、スパークを加えトラックを延長する。ここで i は延長前のトラックの最後のスパークのワイヤ層の番号と、加えようとしているスパークのあるワイヤ層の番号での差で、トリプレットを探す作業で用いたのと同じく、NWIRE(1)=8、NWIRE(2)=15、NWIRE(3)=22、NWIRE(4)=29 となっている。

加えるスパークがなくなれば、延長済みトラックができたという事であり、既存の延長済みトラック全てと比べ、一方が他方に含まれるときそれを捨てる。具体的

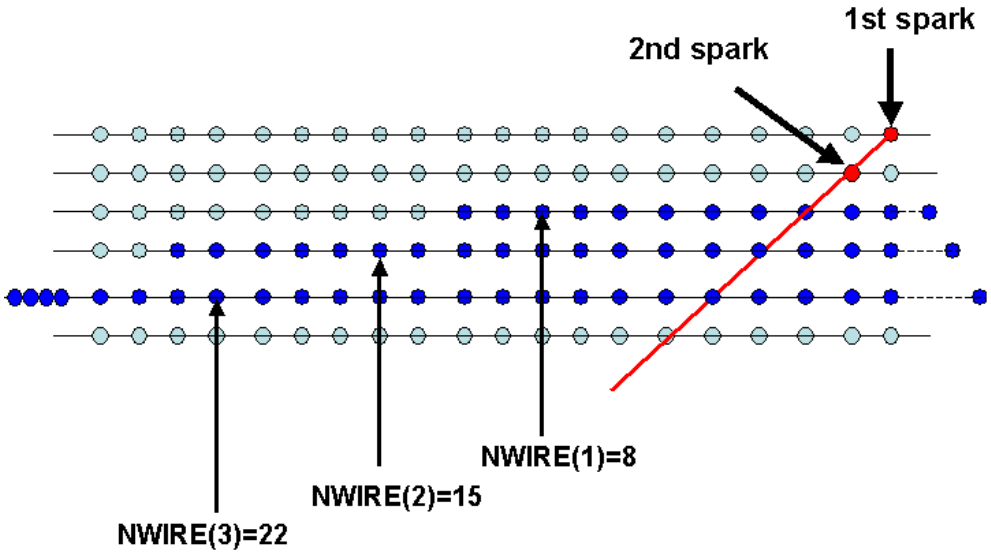

図 4.8: 3rd スパークを探す条件。1st スパーク、2nd スパークを結ぶ線とワイヤ層の交点から、3rd スパークの候補が NWIRE(i) 以内なら 3rd スパークとみなす。ここで i は 2nd スパークのあるワイヤ層と 3rd スパークのあるワイヤ層の番号の差であり、NWIRE(1)=8、NWIRE(2)=15、NWIRE(3)=22 が条件である。

には、スパークを全て比較し、一方のトラックが他方に完全に含まれればそれを捨てる。また先頭のスパークを共有し、一度枝分かれした後、再び 2 つ以上のスパークを共有すれば、同種のトラックとみなし片方を捨てる。このときスパークの数の差が 3 つ以上ならスパークの少い方を落とす。スパークの差が 2 つ以内であれば、トラック中のスパークを結んだ折れ線のなす角の平均値である average turning angle

$$\theta_{\rm av} = \frac{1}{k}\sum_{i=1}^{k} |\theta_i| \tag{4.4}$$

(ここで θ_i は i 番目の交点でのなす角、k は交点の数) を用いて、以下の条件で判定する。スパークの数の多い方のトラックを A、その $\theta_{\rm av}$ を $\theta_{\rm av\_A}$、スパークの数の少い方のトラックを B、その $\theta_{\rm av}$ を $\theta_{\rm av\_B}$ として、スパークの数の差が 2 のとき

$$\theta_{\rm av\_B} \geq 0.8\,\theta_{\rm av\_A} \tag{4.5}$$

なら B を、逆の不等号なら A をすてる。またスパークの数が差が 1 のとき

$$\theta_{\rm av\_B} \geq 0.9\,\theta_{\rm av\_A} \tag{4.6}$$

なら B を、逆の不等号なら A を捨てる。

このようにして、全てのトラックを延長し終わったら、延長済みトラック同士を比べる。一方のトラックの最後のスパークだけで分離している場合、および先頭の

スパークと共有し、一度枝分かれした後、一方のトラックの最後のスパークで合流しているときは、良く似たトラックであるとみなし、トラック中のスパークの数や $\theta_{\rm av}$ を元にさらなるトラックの選別を行っている。この作業は複雑なので、現在は省略している。

こうしてトリプレットをトラックに拡張した結果の例を図 4.10 に示す。図中で×印がスパークのあったワイヤを示し、直線で結ばれた物が再構成されたトラックを示しており、電子陽電子の飛跡と思われる物を、我々のプログラムがトラックとして認識できていることが分かる。

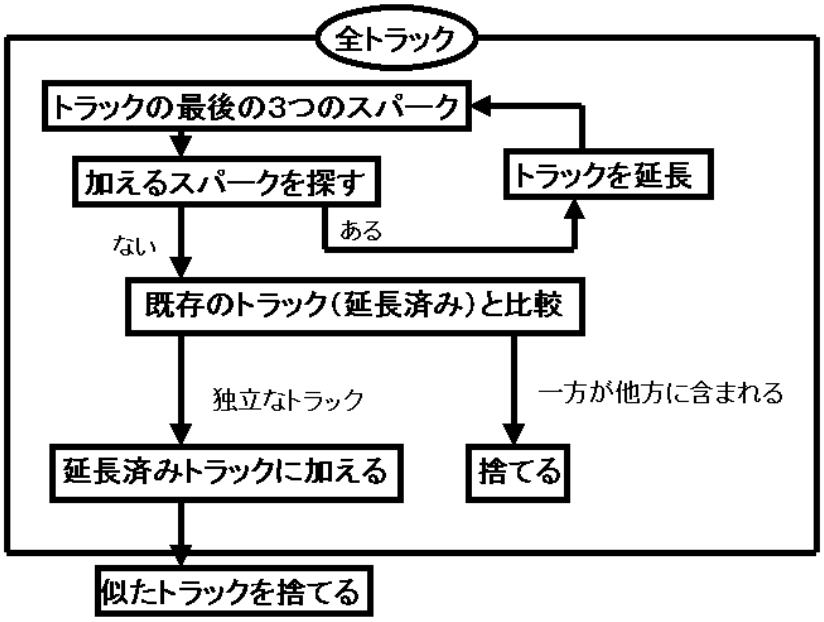

図 4.9: トラックに延長する作業の流れ図

4.3.5 Primary トラックと Secondary トラックを選定する

前節までで、電子陽電子対の飛跡と考えられるトラックが探し出された。1 つの γ 線から電子陽電子対は複数作られることがあり、この場合はもとの γ 線の方向を反映しているのは、エネルギーの高い電子ないしは陽電子である。エネルギーが高いと、トラックがよりまっすぐになると期待されるので、ここでは一番素状の良い (直線に近くかつ長い)primary トラックと、2 番目に素状の良い secondary トラックを探す。

1. Primary トラックを選定する

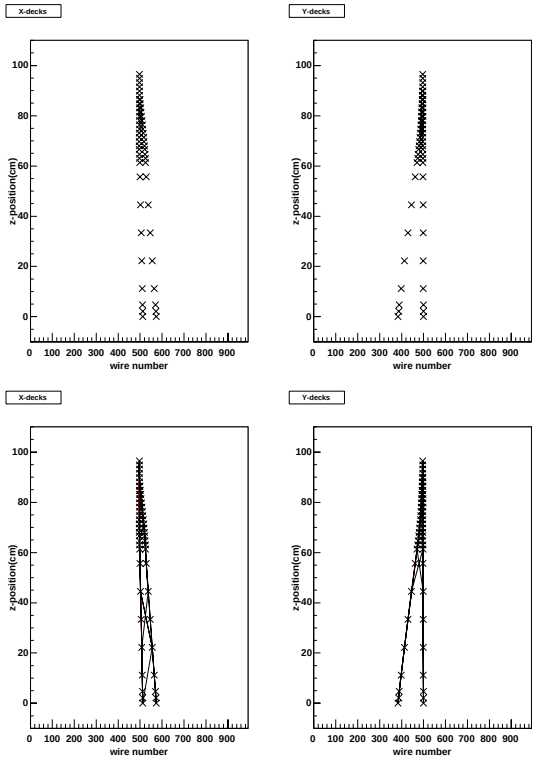

図 4.10: (上) トラック拡張前のスパーク。(下) トラック拡張プログラムを通した後。ここで左が X 座標を測るワイヤ群に右が Y 座標を測るワイヤ群に投影した図である。×印はスパークのあったワイヤ、直線で結ばれた物が再構成されたトラックを示す。

まず見つかったトラックをスパークの数の多い順に並べ換える。もしスパークの数が同じ場合は、前節の $\theta_{\rm av}$ の小さい順に並べる。このような並び替えを行うのに” バブルソート” と呼ばれる並び替えのアルゴリズムを用いた。このアルゴリズムは、並び替えの済んでいないデータ配列の隣り合う要素を比べ、条件を満たしていれば、並べ替えるという作業を配列の先頭から末尾まで順に繰り返すものである。要素の最大値や最小値 (値の大きさでソートする場合) が末尾に移動する様子が水中の泡 (バブル) が水面に上がる様子に似ていることからバブルソートと呼ばれている。具体的に図 4.11 のプログラムを見ながら説明する。まず並べ替えるトラックを listTrackX というリストに格納している。j 番目のスパークの数が j+1 番目のそれより少い場合ないしはスパークの数が同じで折れ曲がりの度合を表す average turning angle(プログラム中の ata1、ata2) の大きい場合は、j 番目と j+1 番目のトラックを入れ替えるようにしている。したがって最初のトラックと 2 番目のトラックの比較から始めて 2 番と 3 番、3 番と 4 番、…とやっていけば最終的に一番スパークの数が少いト

ラックがリストの最後に移動することになる。この一連の比較操作を内側の for 文で行っている。このの操作が終わると一番数の少いトラックは確定するので比較対象から外す。こうした操作を繰り返すとスパークの数の多い順に並ぶということになる。

並べ換えが済んだら一番最初、つまり最もスパークの数が多くよりまっすぐなトラックと二番目のトラックの比較を行い『二番目のトラックが最初のトラックよりスパークの数が 1 つだけ少なく、$\theta_{\rm av}$ が 10%以上小さい』という条件をみたしたら二番目のトラックを、そうでなければ最初のトラックを primary トラックとみなす。

```
for i in range(len(listTrackX)-1):
 for j in range(len(listTrackX)-i-1):
  num1 = len(listTrackX[j])/2-1 # jth track
   num2 = len(listTrackX[j+1])/2-1 # (j+1)th track
   ata1 = listTrackX[j][-1] # jth track
   ata2 = listTrackX[j+1][-1] # (j+1)th track
   tmp1 = listTrackX[j]
   tmp2 = listTrackX[j+1]
   if (num1<num2 or (num1==num2 and ata1>ata2)):
     listTrackX[j] = tmp2
     listTrackX[j+1] = tmp1
```

図 4.11: バブルソートのアルゴリズムを用いた、トラックの並べ換えのプログラム

2. secondary トラックを選定する

以下の 2 つの条件を満たすトラックを secondary トラックの候補とする。

2-1 primary トラックと一番上のスパークを共有している

γ 線が対生成してできる電子・陽電子対は、始めは接近しており、徐々に離れていくからと考えられる。よって一番上のスパークが離れているようなトラックは電子・陽電子対では無い別の何かによるものだと判断される。よって一番上のスパークを比較して、両者が同じであれば secondary トラックの候補とみなす。

2-2 primary トラックに含まれないスパークを NDIFF(5) 以上もつ。

独立なスパークをほとんど持たない 2 つのトラックは電子・陽電子対ではなく、電子や陽電子が作ったトラックとそれが δ 線 (電離で生じた低エネ

ルギー電子) を出したことによる、疑似トラック対の可能性が高いと考えられる。そこで独立なスパークの数が NDIFF(5) 以上であることを要求する。primary トラックに含まれないスパークの数を数え上げるプログラムを図 4.12 に示す。ここで t1 は primary トラックの情報が、t2 は比較されるトラック情報の入ったリストである。まず track1、track2 というリストを用意して、t1、t2 それぞれのワイヤ層の番号 (deck という変数) とワイヤ番号 (wire という変数) を、append を使い track1 と track2 の後ろにひとまとめにして格納する。全て格納し終わったら track1 と track2 の比較を行う。

if(track2[i] in track1):の部分は track2 の i 番目の要素が track1 に含まれるかどうか調べるもので、もし含まれないならカウンター変数である n に 1 を加えることで track1 に含まれないスパークの数を数え上げる。そしてこの関数での返り値と NDIFF(5) を比較することで、secondary トラックの候補とみなせるかどうかを判定する。

以上の二つの条件を満たしたトラックを θ_{av} の小さい順にバブルソートのアルゴリズムを用いて並べ換え、一番スパークの数が多く真直なトラックを secondary トラックとする。このようにして選定されたトラックは図 4.13 の下のようになる。ここで赤色で示されたのが primary トラック、黒の線で示されたのが secondary トラックである。尚、EGRET に実際に適用されたリコンストラクションでは θ_{av} の他に、primary トラックと secondary トラックがお互いどれだけ離れているか、また secondary トラック中に、スパークのないワイヤ層がどれだけあるかも考慮して、secondary トラック候補の並べかえを行っているが、本論文ではまだ省略している。

4.3.6 X 座標を測るワイヤ群と Y 座標を測るワイヤ群の間でのトラックの関連付け

前節までで、X 座標を測るワイヤ群と Y 座標を測るワイヤ群で独立に、primary トラック、secondary トラックが選定された。この先は、両者をまとめて処理していく。本節は X 座標を測るワイヤ層群と Y 座標を測るワイヤ層群での、primary トラックと secondary トラックの対応付けであり、両ワイヤ層群とも secondary トラックを持つ場合に対応付けを行う。

デフォルトでは、X-ワイヤ層群の primary トラックは Y-ワイヤ層群のそれに対応すると考えている。これが実際は、X-ワイヤ層群の primary トラックが Y-ワイヤ層群のの secondary トラックに、X-ワイヤ層群の secondary トラックが Y-ワイヤ層群

```
def compare2(t1, t2):
  track1 = []
  track2 = []
  nSparks = len(t1)/2-1 # the number of sparks of the track
  for i in range(nSparks):
    deck = t1[i+1]
    wire = t1[i+1+nSparks]
    track1.append([deck, wire])

  nSparks = len(t2)/2-1 # the number of sparks of the track
  for i in range(nSparks):
    deck = t2[i+1]
    wire = t2[i+1+nSparks]
    track2.append([deck, wire])
  n = 0
  for i in range(nSparks):
    if (track2[i] in track1):
      n = n
    else :
      n = n +1

  return n
```

図 4.12: 異なるスパークの数を数え上げるプログラム

の primary トラックに対応していないか確認して、そうなっている場合には Y-ワイヤ層群の primary トラックと secondary トラックを入れ換える。

入れ換えの条件は次のようになる。X-ワイヤ層群の primary トラック、secondary トラックをそれぞれ X1、X2、Y-ワイヤ層群の primary トラック、secondary トラックをそれぞれ Y1、Y2 とし、C1 を (X1 と Y1、X2 と Y2) の組合せで、共通のワイヤ層にスパークを持つ数と定義し、また C2 を (X1 と Y2、X2 と Y1) の組合せで、共通のワイヤ層にスパークを持つ数と定義する。このとき

$$\frac{(\mathrm{C2}-\mathrm{C1})}{\mathrm{C1}} \geq \mathrm{THRSH}(0) \tag{4.7}$$

を満たせば、Y-ワイヤ層群の primary トラック、secondary トラックを入れかえる。

共通のワイヤ層にスパークを持つ数を返す関数を図 4.14 に示す。具体的には、まずはそれぞれのトラックでスパークのあるワイヤ層の番号を格納するため、listDeck_tx1 などのリストを初期化している。そしてワイヤ層の番号をそれぞれ append を使って対応するリストに格納していく。そして変数 c をカウンターとし、tx1,ty1 とで共通のワイヤ層にスパークを持つ数、および tx2,ty2 とで共通のワイヤ層にスパークを持つ数を数え上げている。この関数を用いて C1、C2 を求め (4.7) 式の条件を満たせばの Y-ワイヤ層群の primary トラック、secondary トラックを入れかえるようにしている。

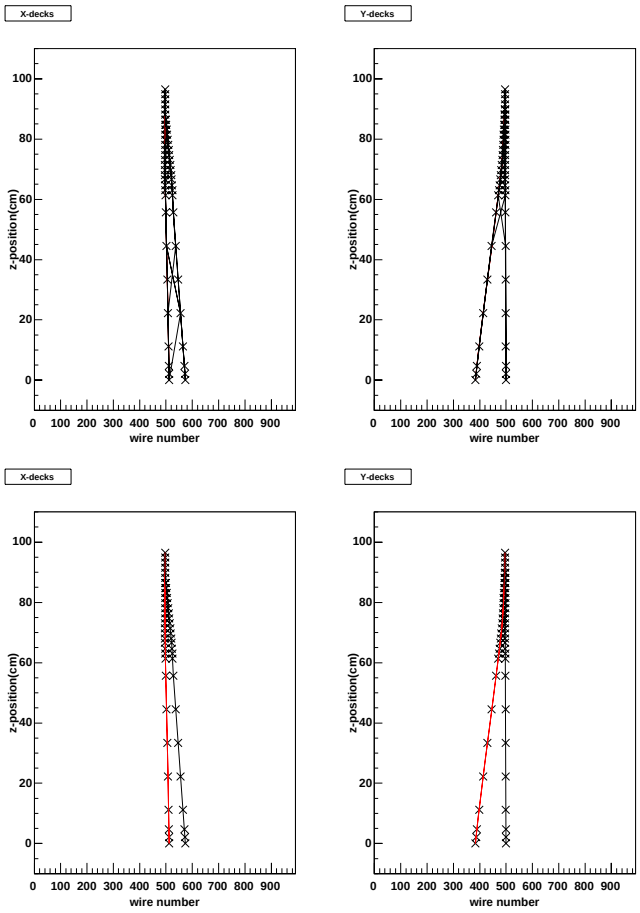

図 4.13: (上)primary トラックと secondary トラックの選定前のトラック。(下)primary トラックと secondary トラックの選定のプログラムを通した後。ここで、左が X 座標を測るワイヤ群で右が Y 座標を測るワイヤ群、×印はスパークのあったワイヤ、直線は再構成されたトラックを示す。

尚、EGRET の実機では、Y-ワイヤ層群の 36 層のうち、34 番目、35 番目は 45° 傾けてワイヤが張られており、これを用いて X-ワイヤ層群と Y-ワイヤ層群のトラックの対応付けを見ることもできる。オリジナルのリコンストラクションはこの 45° 傾いたワイヤ層の情報も用いているが、条件判定が複雑なので本論文ではひとまず省略している。

4.3.7 γ 線事象か否かの判定を行い γ 線の到来方向を求める

γ 線か否かの判定

以下の条件を両方満たすような事象を γ 線事象とする

```
def corr(tx1, tx2, ty1, ty2):
 listDeck_tx1 = []
  listDeck_tx2 = []
  listDeck_ty1 = []
  listDeck_ty2 = []
  nSparkstx1 = len(tx1)/2-1 # the number of sparks of the tracktx1
  for i in range(nSparkstx1):
    deck = tx1[i+1]
    listDeck_tx1.append(deck)
  nSparkstx2 = len(tx2)/2-1 # the number of sparks of the tracktx2
  for i in range(nSparkstx2):
    deck = tx2[i+1]
    listDeck_tx2.append(deck)
  nSparksty1 = len(ty1)/2-1 # the number of sparks of the trackty1
  for i in range(nSparksty1):
    deck = ty1[i+1]
    listDeck_ty1.append(deck)
  nSparksty2 = len(ty2)/2-1 # the number of sparks of the trackty2
  for i in range(nSparksty2):
    deck = ty2[i+1]
    listDeck_ty2.append(deck)
  c = 0
  for i in range(len(listDeck_tx1)):
    if (listDeck_tx1[i] in listDeck_ty1):
      c = c+1
  for i in range(len(listDeck_tx2)):
    if (listDeck_tx2[i] in listDeck_ty2):
      c = c+1

  return c
```

図 4.14: トラック間の共通 deck の数をカウントするプログラム

- X-ワイヤ層群、Y-ワイヤ層群の少くともどちらか一方では primary トラックと secondary トラックを持つ。
- ユニークなスパークの数をが NPTMIN(7) より大きい

ここでユニークなスパークの数とは、primary トラックと secondary トラックの各々のスパークの数の和から、両者で共有するスパークの数を引いたものである。具体的なコードを図 4.15 に示す。

4.3.5 節の図 4.12 とほぼ同様の方法を用いていて、ここでは逆に primary と secondary でスパークを共有していたらカウンター c を 1 つずつ増やして共有するスパークの数を求め、それと primary トラック、secondary トラックのスパークの数からユニークなスパークの数を求めている。

また実機に対するリコンストラクションでは、トラックの始点が、スパークチェンバーの壁に近い事象 (スパークチェンバーの外で対生成を起こした可能性が高い) や、トラックの構成要素として認識されたスパークの割合が少い事象 (正しくγ線

の方向が求まらない可能性が高い) も非γ線事象として落とすが、本論文では単純化のため、まだ実装していない。

```
def Unisparks(t1,t2):
  listDeckSpark_t1 = []
  listDeckSpark_t2 = []
  nSparkst1 = len(t1)/2-1 # the number of sparks of the trackt1
  for i in range(nSparkst1):
    deck = t1[i+1]
    spark = t1[i+nSparkst1+1]
    listDeckSpark_t1.append([deck, spark])
  nSparkst2 = len(t2)/2-1 # the number of sparks of the trackt2
  for i in range(nSparkst2):
    deck = t2[i+1]
    spark = t2[i+nSparkst2+1]
    listDeckSpark_t2.append([deck, spark])
  c = 0 # the number of matches for the pair of (t1,t2)
  for i in range(nSparkst1):
    if (listDeckSpark_t1[i] in listDeckSpark_t2):
      c = c+1
  return  nSparkst1 + nSparkst2 - c
```

図 4.15: ユニークなスパークの数を返す関数

γ線の到来方向を求める

上記の判定でγ線事象と判断されたら、primary トラックと secondary トラックの方向を最小自乗法で出し、両者の重みつき平均をγ線到来方向とする。

最小自乗法でγ線の到来方向を求めるのにまず、X-ワイヤ層群、Y-ワイヤ層群各々でトラックの方向を出し (つまり、XZ 面および YZ 面に投影した方向を出し) それから 3 次元の方向を求める。各々の方向を求めるのにまず先頭の 3 つ (または、4 つ) のスパークを最小自乗法を用いて直線フィットして方向を出す。続くスパークでこの直線に近いものがあればどんどん加え最小自乗法で直線を計算し直す。これを加えるスパークがなくなるまで続け XZ 面、YZ 面の方向を重みをつけて足し、これをγ線の方向とする。大まかな流れを図 4.16 に示す。

具体的には次の様な手順で方向決定を行う。XZ 面の方向を出す場合を例に説明する。

(1) トラックの先頭 4 つのスパークを最小自乗フィットする。この際 $z = a + bx$ の表式を用いると、b が無限大になりうるので、$x = a + bz$ の表式でフィットを行った。すると a, b は以下のように求まる。

$$
\begin{aligned}
a &= \frac{1}{\Delta}(\sum_i z_i^2 \sum_i x_i - \sum_i z_i \sum_i z_i x_i) \\
b &= \frac{1}{\Delta}(N \sum_i z_i x_i - \sum_i z_i \sum_i x_i) \\
\Delta &= N \sum_i z_i^2 - (\sum_i x_i)^2
\end{aligned}
$$

ここで N はフィットを行うスパークの数の総和であり、i は何番目のスパークかを示す変数である。

(2) フィッティングを行った直線から大きくずれるスパークがあればそれを外し、残り三つのスパークで再びフィッティングを行う。

実際のスパーク (i 番目とする) の位置と最小自乗法で求めた直線とのずれが、スパークの位置の不定性である $\sqrt{\delta^2 + \Delta x_i^2}$ の DEVCUT(2.54) 倍より大きい時、直線から大きくずれているとみなす。ここで δ はワイヤ間隔が有限なことに基因する不定性、$\Delta\ x_i$ は最小自乗法で求めた係数の誤差に基因する不定性であり、誤差の伝播により、両者の自乗和の平方根を取った。δ は、ワイヤの間隔 0.813 mm に比例する量で、EGRET で用いられたプログラムのコードに習い

$$
\delta = 0.0813\sqrt{\frac{\pi}{2}} \tag{4.8}
$$

とした。また最小自乗法で求めた係数 a、b の不定性は各々

$$
\Delta a^2 = (\delta^2/\Delta)\Sigma z^2
$$

$$
\Delta b^2 = N\delta^2/\Delta^2
$$

なので、それによる不定性は

$$
\Delta x^2 = \Delta a^2 + z^2 \Delta b^2
$$

となる。

(3) (1),(2) のようにして、先頭の 3 つ (又は 4 つ) のスパークと、それから求まる直線がきまったら、続く nset(3) 個までのスパークと直線とのずれを、(2) と同様の方法で判定する。ずれが DEVCUT$\times\ \sqrt{\delta^2 + \Delta x_i^2}$ 以内におさまるスパーク

があれば、それを加えて新たに直線を計算し直し、ここまでのスパーク全てで再び直線とのずれの判定を行う。ずれが小さい (DEVCUT$\times\sqrt{\delta^2+\Delta x_i^2}$ 以内) のスパークのみで再び直線を求め、これを新たな直線部の方向とする。そしてまた新たに nset(3) 個までのスパークと直線とのずれの判定に戻る。この操作を繰り返し、新たに加えたスパークが全て直線からのずれが大きいか、 加えるスパーク自体が無くなったときやめる。

このようにして primary、secondary トラックの方向が求まったら、直線フィットに用いたスパークの数の 3 乗と、直線とスパークのずれの自乗和平均の逆数で重みをつけ、XZ 平面、YZ 平面各々での γ 線の方向を求める。つまりスパークの数の多いトラック、真直なトラックほど重みを大きくして方向を求めるのである。最後に XZ 平面、YZ 平面での方向から、3 次元的な方向を求め、これを γ 線の方向とみなす。具体的には、以下のようになる。

今 XZ 面、YZ 面のトラックは各々$x=a+bz$ ないしは $y=a+bz$ で直線部が記述される。XZ 面内の primary トラックの直線部のスパークの数をを n_1、傾きを b_1、残差の自乗和の平均を g_1、secondary トラックのそれらを n_2、b_2、g_2 とすると、primary トラックの向きは

$$\frac{dz}{dx_1}=\frac{1}{b_1} \tag{4.9}$$

secondary トラックの向きは

$$\frac{dz}{dx_2}=\frac{1}{b_2} \tag{4.10}$$

となる。各々に平行な単位ベクトルは

$$\overrightarrow{e}_1=\frac{1}{\sqrt{1+(\frac{dz}{dx_1})^2}}\begin{pmatrix}1\\ \frac{dz}{dx_1}\end{pmatrix} \tag{4.11}$$

および

$$\overrightarrow{e}_2=\frac{1}{\sqrt{1+(\frac{dz}{dx_2})^2}}\begin{pmatrix}1\\ \frac{dz}{dx_2}\end{pmatrix} \tag{4.12}$$

であるから、XZ 面に投影した γ 線の方向はスパーク数の三乗と残差の自乗和平均の逆数で重みをつけて

$$\frac{dz}{dx} \equiv \frac{n_1^3}{g_1}\overrightarrow{e}_1 + \frac{n_2^3}{g_2}\overrightarrow{e}_2 \tag{4.13}$$

となる。同様にYZ面に投影したγ線の方向$\frac{dz}{dy}$が求まり、これらから3次元空間におけるγ線の方向が

$$\overrightarrow{r}_{\mathrm{recon}} = (\frac{dx}{dz}, \frac{dy}{dz}, -1) \tag{4.14}$$

と求まる。

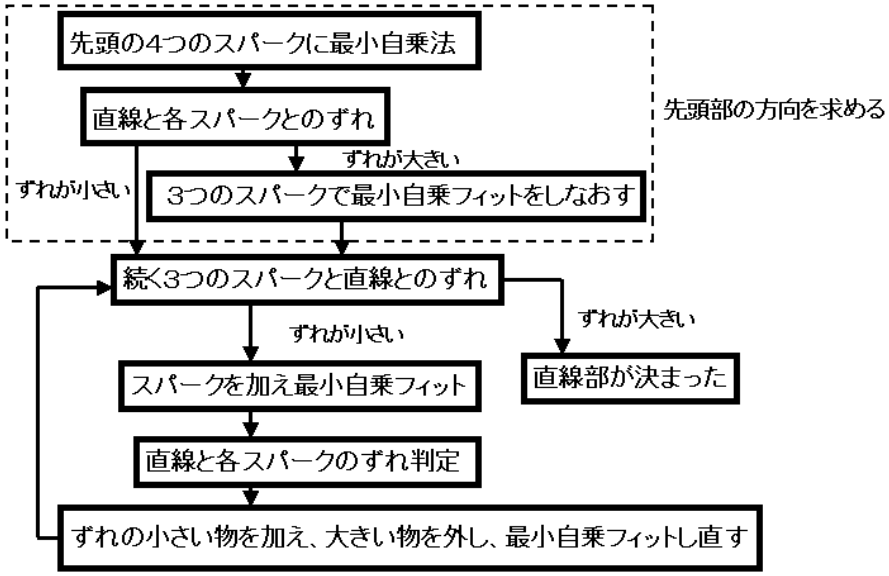

図 4.16: トラックの直線部を求める作業の流れ

4.4 ビーム試験との比較

前章までで作成したDigitization、リコンストラクションプログラムを評価するために、EGRETシミュレーターのシミュレーション結果にこれらのプログラムを適用し、有効面積、角度分解能を求めビーム試験で得られている有効面積、角度分解能との比較を行った。

有効面積の比較

前節の「γ線か否かの判定」の結果を用いて有効面積を求める。この判定を行うことで最終的にγ線であると判断されるイベントが出てくるので、そのイベントの

数を数え上げる。シミュレーションはビーム試験の行われた各エネルギー毎に 1 万発の γ 線を打ち込んで行った。但し低エネルギー側は γ 線と判定されるイベントの数が少く、統計誤差が大きくなるので 35 MeV は 100 万発、60 MeV、100 MeV、200 MeV は 10 万発で行った。γ 線は斜め方向から γ 線を打ち込んでもスパークチェンバーの有感領域に当るように、対角線の長さより大きい直径 160 cm の円内に一様に打ち込んだ (図 4.17)。このとき、有効面積は以下のように計算される。

$$\text{有効面積} = \pi(80\text{cm})^2 \times \left[\frac{\gamma\text{線と判定されたイベント数}}{\text{打ち込んだ}\gamma\text{線の数}} \right] \tag{4.15}$$

上記の式で天頂角 0、10、20 度からエネルギー 30 MeV,60 MeV,100 MeV、200 MeV、500 MeV、1 GeV、3 GeV、10 GeV の有効面積をそれぞれ出し、それと 3.2 節で述べたキャリブレーションファイルのデータをグラフにプロットしての比較を行った。結果を図 4.18、4.19、4.20 に示す。

図 4.18 の天頂角が 0 度の結果をみると、ビーム試験、シミュレーションともに約 500 MeV で有効面積が最大となり、エネルギーが高く、ないしは低くなるにつれ下がっていく。このエネルギー領域での電子・陽電子対生成の反応断面積はエネルギーにあまりよらず測定された有効面積の低下 (高エネルギーで約 1/3、低エネルギーで約 1/4 に落ちる) は単なる反応確率だけでは説明出来ない。高エネルギー、低エネルギーで有効面積が下がるのは、以下のように理解できる。低エネルギーの γ 線の場合は、生成した電子・陽電子のエネルギーも小さいため、十分な長さのトラックを形成出来なかったり、大きく曲げられたり、電子・陽電子対の軌跡のなす角が大きくなってしまったりするため、γ 線事象と判定されないイベントが多くなり有効面積が下がる。一方高エネルギーの γ 線は、カロリーメーターで電磁シャワーが起きるので、低エネルギーの光子 (X 線) が Scinti Dome へ後方散乱 (バックスプラッシュ) してエネルギーを落とすため、荷電粒子事象としてイベントが捨てられ、有効面積が下がると考えられる。

次にシミュレーションとビーム試験の結果を詳しく見てみるとビーム試験で信頼がおけるとされる 200 MeV から 1 GeV の結果をほぼ完全に再現し、また 10GeV までの高エネルギーの結果も 10% 以内で再現できている。よってバックスプラッシュも含めて我々のシミュレーションツールは実機を正しく再現しており、EGRET の応答関数が正しいことを確認するとともに、高エネルギー (10 GeV 以上) の応答関数を求めるのに、我々のシミュレーター・リコンストラクションが有効であることが分かった。

一方 100 Mev 以上でビーム試験の結果を大きく上回っている原因を考えると、低エネルギーの γ 線は上述のように様々な理由から棄却される可能性があるのに対し、我々はリコンストラクションを多少単純化して行っているため条件が甘くなり、本来ならば落とされるであろうイベントも拾っていることが考えられる。

また、天頂角を 10 度、20 度と変えた図 4.19,4.20 の有効面積をみると、入射角度を大きくするとビーム試験の結果からのずれが大きくなっている。これは本来のリコンストラクションでは壁からくるトラックを捨てているが、我々のプログラムはまだそうしておらず (4.3.7 参照)、これにより本来捨てられてしまうであろうイベントも γ 線事象として拾っていることが疑われる。入射角度を大きくすれば壁にあたるイベントも多くなるのでそれが拾われる分 γ 線イベントが増え、ビーム試験で得られた有効面積より大きな値が得られたものと考えられる。

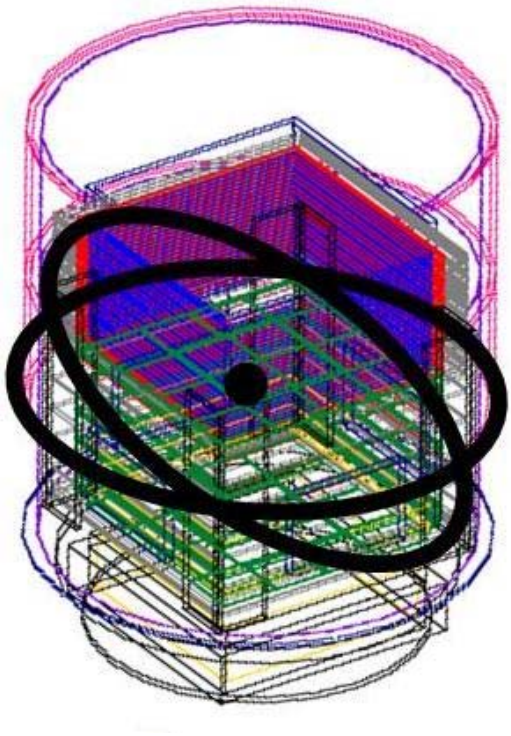

図 4.17: スパークチェンバーの有感領域を覆う γ 線を打ち込む直径 160 cm の円。

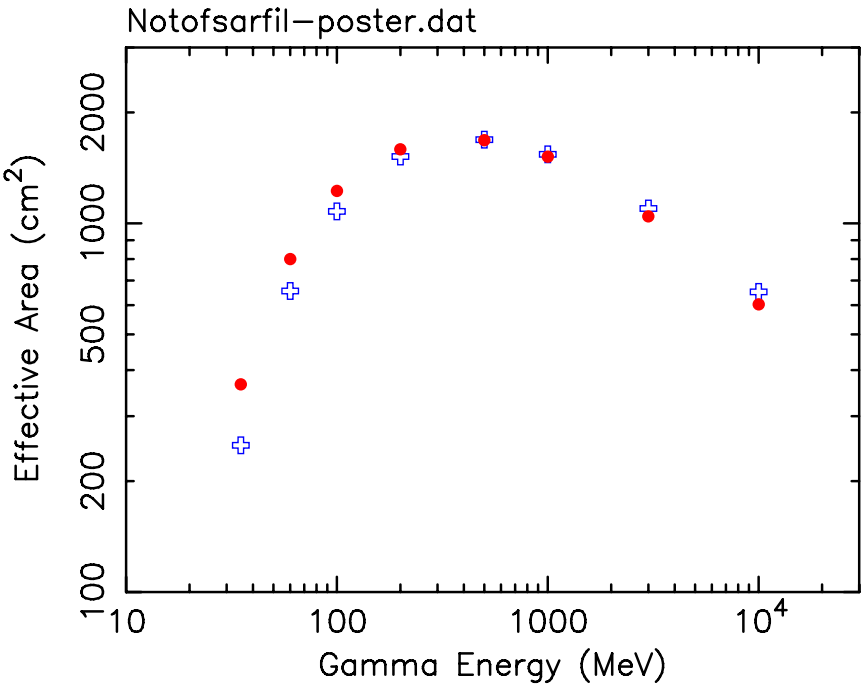

図 4.18: 天頂角 0 度から γ 線を入射したときの有効面積の比較 (赤●:シミュレーション結果、青＋:ビーム試験の結果)

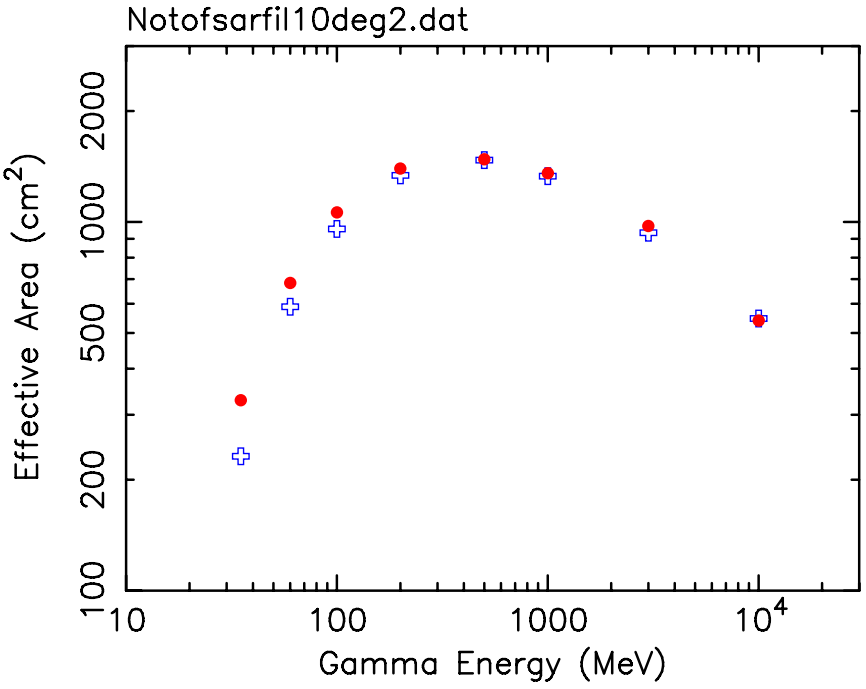

図 4.19: 天頂角 10 度から γ 線を入射したときの有効面積の比較 (赤●:シミュレーション結果、青＋:ビーム試験の結果)

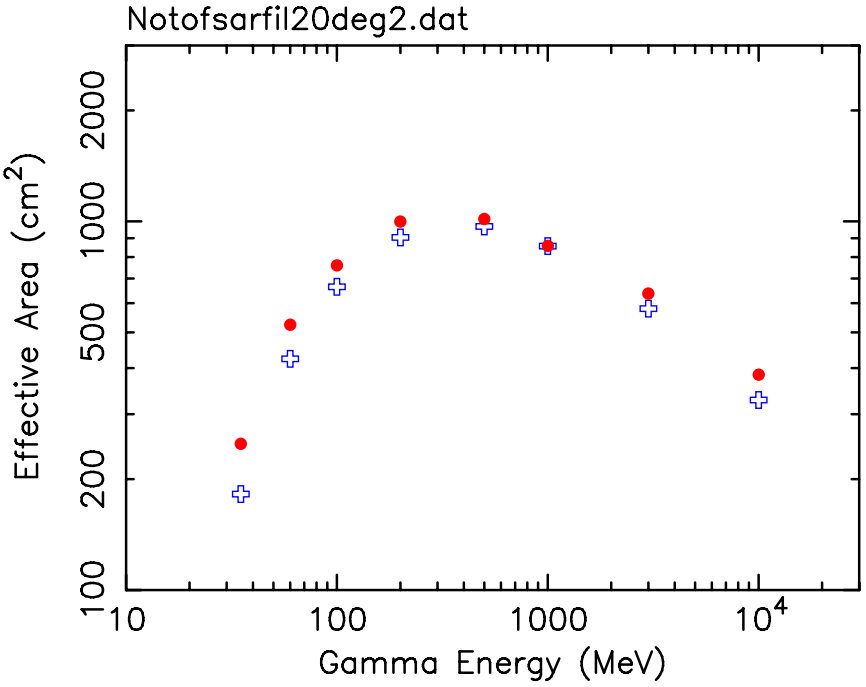

図 4.20: 天頂角 20 度から γ 線を入射したときの有効面積の比較 (赤●:シミュレーション結果、青＋:ビーム試験の結果)

角度分解能の比較

EGRET の角度分解能は、真の γ 線の到来方向と、リコンストラクションで求めた γ 線の方向のなす角の分布で決まる。4.3.7 節に従ってリコンストラクションによる γ 線の方向 $\overrightarrow{r}_{\rm recon}$ を求め、これと真の γ 線の方向 $\overrightarrow{r}_{\rm true}$ より、両者のなす角を

$$\cos\theta = \frac{\overrightarrow{r}_{\rm true}\cdot\overrightarrow{r}_{\rm recon}}{|\overrightarrow{r}_{\rm true}||\overrightarrow{r}_{\rm recon}|} \tag{4.16}$$

として求めた。

このようにして求めた、「真の γ 線の方向」と「リコンストラクションで得られた γ 線の方向」のなす角の頻度分布を、我々のシミュレーションとビーム試験とで比べたのが図 4.21 である。ここでは天頂角 0 °(垂直下向き) の場合について、ビーム試験の行われた 8 つのエネルギー全てで比較を行っている。

これを見ると 500 MeV 以上ではビーム試験の結果を大体再現できていると思われる。一方 200 MeV 以下の低エネルギー側では、シミュレーションで得られた角度の分布がビーム試験の結果よりも大きく広がり、エネルギーが下がるにつれその傾向が大きくなっていることが見てとれる。

この原因として考えられるのは、4.2 節の phase7 において X-ワイヤ層群と Y-ワイヤ層群の間でのトラックの関連付けを単純化していることである。高エネルギー

のγ線の場合、生成した電子陽電子対もほぼγ線の方向に沿って走るので、primary トラックと secondary トラックの対応を取り違えてもほとんど影響がないが、低エネルギーは 2 つのトラックのなす角が大きいので、対応関係を取り違えると本来のγ線方向から大きくずれた方向を求める可能性が高い。よってこの部分を最後までプログラムに入れ込むことで、低エネルギー側も正しい角度分布に近づくと期待される。

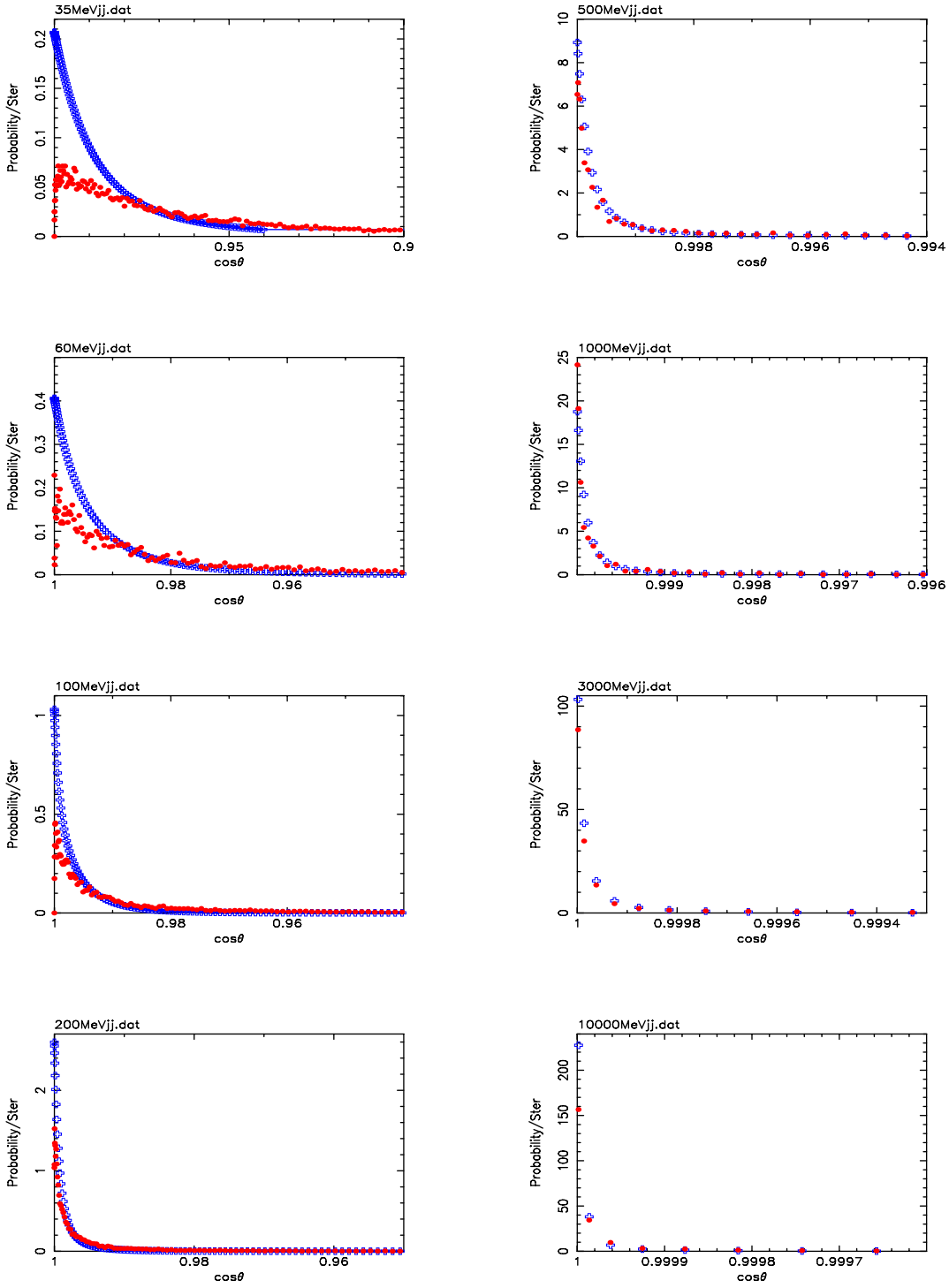

図 4.21: 天頂角 0 度からγ線を入射したときの「真のγ線方向」と「リコンストラクションで得られたγ線の方向」のなす角の確率分布。左側は上から順に 35 MeV、60 MeV、 100 MeV、 200 MeV、右側は上から順に 500 MeV、1 GeV、3 GeV、10 GeV。青＋はキャリブレーションデータ、赤●はシミュレーション結果を示す。

第5章　まとめと今後の課題

本研究では、宇宙γ線衛星GLASTの解析ツールの開発などにつなげ、またEGRET検出器自身の応答関数の再評価を行うことを目的として、EGRETのシミュレーションプログラムの開発を行った。具体的には既に開発された(3章)モンテカルロシミュレーター(ジオメトリ部)の出力結果ををDigitization、リコンストラクションするプログラムの開発をスクリプト言語pythonを用いて行った(4.1から4.3節)。現段階で既に、垂直入射の場合で200 MeV以上のエネルギーの有効面積を10%以内で再現することに成功しており、我々のシミュレーター、リコンストラクションプログラムが信頼できる事をしめしている(図4.18)。特に高エネルギー側はビーム試験において低エネルギー光子の混入のため不定性が大きく、またカロリーメーター中の電磁シャワーが反同時係数用のScinti Domeに戻るバックススプラッシュの影響で有効面積が下がっていく領域であり、この領域で有効面積を精度良く再現できたことは、我々のシミュレーターを用いて、10 GeV以上の領域での正しい応答関数を求めることが可能であることを示している。高エネルギー側は有効面積のみならず角度分解能も概ねビーム試験の結果を再現できており(図4.21)、信頼性は高いと言える。

一方、最も低エネルギー(35 MeV)で有効面積を約1.5倍過大評価しており、また角度分解能も200MeV以下ではシミュレーションによる角度分解能の方がビーム試験で得られたものよりも広がった分布をしている。この原因としては4.4節で述べたように、γ線事象か否かの判定部分、およびX-ワイヤ層群とY-ワイヤ層群の間のトラックの関連付けを単純化していることが上げられる。今後はこれらを最後まで入れ込み、またその他単純化した箇所も順次実装することで、有効面積、角度分解能とも精度良く再現できるようにすることが目標である。最終的にはビーム試験の行われていないエネルギーでの応答関数を、飛び飛びの実験結果をもとに適当な関数で内挿、外挿して求める従来の方法ではなく、シミュレーションをもとに正しく求めてやり、EGRETのデータの再解析を通して、新しいサイエンスやGLASTの解析手法の開発につなげていきたい。

付 録A　リコンストラクションに用いたプログラム

以下 4.2 節の各 phase に対応して、私が主に作成に携わったプログラムを示す。

Digitizationのプログラム

　シミュレーターの出力の物理座標を EGRET 実機のワイヤ番号に置き直すプログラム

```
import sys, string, re
inputName = sys.argv[1]
input = open(inputName, "r")

# Spark chamber parameter
wireDistance = 0.813 #(mm) spacing of adjacent wires in mm
nWires = 992 # the number of wires in a deck
# rightmost position of the wire number 992, 992*0.813/2 = 403.248mm
l = nWires*wireDistance/2

# boolean
true = 1
false = 0

# match charactor to see if line read is for comment or not
beginComment_match = re.compile("/\*")
endComment_match = re.compile("\*/")
# boolean to show if the current line is comments or not
isComment = false

#---------- flag initialization command ----------
# flags to indicate what kind of detector we are reading
isParticle = false
isAnti = false
isSpark = false
isTof = false
isTask = false
def initFlag():
  global isParticle, isAnti, isSpark, isTof, isTask
  isParticle = false
  isAnti = false
  isSpark = false
  isTof = false
  isTask = false
#---------- flag initialization command ----------
```

```
#---------- main routine ----------
# read the data
while 1:
  line = input.readline()
  if not line:
    break

  # if the line read is for comment, go to the next line
  if beginComment_match.search(line):
    isComment = true
  if endComment_match.search(line):
    isComment = false
    continue
  if isComment == true:
    continue

  # begin of the event
  if re.match("^###\ EventNo", line):
    columns = string.split(line)
    eventNo = int(columns[2])
    print  '0', eventNo,

  if re.match("^gamma", line):
    if (isParticle==true):
      print line,

  # end of the event
  if re.match("^###\ End", line):
    initFlag()
    for i in range(36):
      # sort the data
      listWireNx[i].sort()
      listWireNy[i].sort()

      # remove the "[" "]" and ","
      lengthX = len(listWireNx[i])
      lengthY = len(listWireNy[i])
      print i, "x" ,
      print lengthX ,
      for m in range(lengthX):
       print listWireNx[i][m],
      print ""
      print i, "y" ,
      print lengthY ,
      for n in range(lengthY):
       print listWireNy[i][n],
      print ""
    # print deliminator
    print "-1"

  # what kind of detector we are reading?
  if re.match("^##\ Primary Particle", line):
    initFlag()
    isParticle = true

  if re.match("^##\ ScintiDome", line):
    initFlag()
    isAnti = true
```

```
if re.match("^##\ Spark Chamber", line):
  initFlag()
  isSpark = true
  # initialize list to store ids of wire with hit
  listWireNx = []
  listWireNy = []
  for i in range(36):
    listWireNx.append([])
    listWireNy.append([])

if re.match("^##\ TOF", line):
  initFlag()
  isTof = true

if re.match("^##\ TASC", line):
  initFlag()
  isTasc = true

# skip the comment line
if re.match("^#\ ", line):
  continue

# read the data
if not re.match("^#", line):
  columns = string.split(line)
  if (isSpark==true):
     id = int(columns[0])
     Xi = float(columns[7])
     Yi = float(columns[8])
     Xo = float(columns[10])
     Yo = float(columns[11])
     if Xi <= l and Xi >= -l:
       NumXi = (l+Xi)/wireDistance+1
     else:
       continue
     if Xo <= l and Xo >= -l:
       NumXo = (l+Xo)/wireDistance+1
     else:
       continue

     if Yi <= l and Yi >= -l:
       NumYi = (l+Yi)/wireDistance+1
     else:
       continue

     if Yo <= l and Yo >= -l:
       NumYo = (l+Yo)/wireDistance+1
     else:
       continue

     NumXi = int(NumXi)
     NumXo = int(NumXo)
     NumYi = int(NumYi)
     NumYo = int(NumYo)

     if NumXi not in listWireNx[id]:
       listWireNx[id].append(NumXi)
     if NumXo not in listWireNx[id]:
       listWireNx[id].append(NumXo)
```

```
      if NumYi not in listWireNy[id]:
        listWireNy[id].append(NumYi)
      if NumYo not in listWireNy[id]:
        listWireNy[id].append(NumYo)
```

phase0 隣接したスパークひとまとめにするプログラム

連続したワイヤのスパークをひとまとめにするプログラム。

```
import sys, string, re
inputName = sys.argv[1]
input = open(inputName, "r")

# boolean
true = 1
false = 0

#---------- combine command----------
# This function combines adjacent wires and
# gives back the new list
def combine(list):
  newList = [] # new list to store the combined wire numbers
  nhit = len(list)
  if nhit > 1 :
    sequence = 0  # the number of combined wires -1
    for wireID in range(nhit-1):
      # check if the next wire is sequence or not
      if list[1+wireID] == list[wireID] +1:
        sequence = sequence + 1
      elif list[1+wireID] > list[wireID] +1:
        if sequence == 0 : # independet hit
          newList.append(list[wireID])
        else:
          # the combined wire position
          combinedWirePosition = float(list[wireID])-float(sequence)/2
          newList.append(float(combinedWirePosition))
          sequence = 0
        if wireID == nhit-2: # the last wire
          newList.append(list[wireID+1])
    if sequence != 0: # sequence include the last hit
      combinedWirePosition = list[wireID+1]-float(sequence)/2
      newList.append(float(combinedWirePosition))
  elif nhit==1:   # the number of hit is one
    newList.append(list[0])

  # gives back the combined list of wire number
  return newList
#---------- combine command----------

#---------- main routine ----------
# read data
while 1:
  line = input.readline()
  if not line:
    break
```

```
  columns = string.split(line)

  # print the particke info (Monte Carlo truth)
  if re.match(".*gamma.*", line):
    print line,

  # if the line shows the tracker hit, combine adjacent hits
  if re.match(".*x.*", line) or re.match(".*y.*", line):
    deckId = int(columns[0])
    face = columns[1]
    nhit = int(columns[2])
    listWireN = [] # list to store hit wire number
    for i in range(nhit):
      listWireN.append(int(columns[3+i]))
    combinedListWireN = combine(listWireN)
    print deckId, face,
    print  len(combinedListWireN) ,
    for i in range(len(combinedListWireN)):
      print  combinedListWireN[i],
    print

  # print the particke info (Monte Carlo truth)
  if (columns[0]=="-1"):
    print line,
```

phase1 イベントセレクションをするプログラム

イベント数の少いイベントを捨てるプログラム

```
import sys, string, re
inputName = sys.argv[1]
input = open(inputName, "r")

# boolean
true = 1
false = 0

# reconstruction parameters
MINSPK = 6

#---------- main routine ----------
# read data
while 1:
  line = input.readline()
  if not line:
    break
  columns = string.split(line)
  # print the particke info (Monte Carlo truth)
  if re.match(".*gamma.*", line):
    eventLine = line # particle information
    listLines = [] # list to store tracker data
    nhitX = 0  # the number of hit in x face
    nhitY = 0  # the number of hit in y face
  # if the line shows the tracker hit, combine adjacent hits
  if re.match(".*x.*", line):
```

```
    listLines.append(line)
    nhitX = nhitX+int(columns[2])
  if re.match(".*y.*", line):
    listLines.append(line)
    nhitY = nhitY+int(columns[2])


  # only events with sufficient number of hits
  # (more than MINSPK) are printed.
  if (columns[0]=="-1"):
    if nhitX >= MINSPK and nhitY >= MINSPK:
      print eventLine,
      for i in range(len(listLines)):
        print listLines[i],
      print line,
```

phase5 primary トラック secondary トラックを選定する 2 つのプログラム (1)

バブルソートのアルゴリズムを用いて primary トラックを選定するプログラム

```
import sys, string, re
inputName = sys.argv[1]
input = open(inputName, "r")

# boolean
true = 1
false = 0

#---------- main routine ----------
# read data
while 1:
  line = input.readline()
  if not line:
    break
  columns = string.split(line)
  # print the particke info (Monte Carlo truth)
  if re.match(".*gamma.*", line):
    print line,
    listLines = [] # list to store the spark chamber
    listTrackX = [] # list to store the tracks in XZ-view
    listTrackY = [] # list to store the tracks in YZ-view

  # if the line shows the tracker hit, store them
  if re.match(".*x.*", line) or re.match(".*y.*", line):
    listLines.append(line)

  # if the line show the track, store them
  if re.match(".*TRKX.*", line) or re.match(".*TRKY.*", line):
    type = columns[0]
    tmpList = []
    for i in range(len(columns)):
      tmpList.append(columns[i])
```

```
  if (type=="TRKX"):
    listTrackX.append(tmpList)
  else:
    listTrackY.append(tmpList)

# at the end of the event
if (columns[0]=="-1"):
  # print out the spark chamber hits
  for i in range(len(listLines)):
    print listLines[i],

  # sort the list of track in XZ-view
  # Here, we utilize the "bubble sort algorithm".
  # Although not so fast, this algorithm is simple and robust.
  # Reference:
  #   http://www.is.oit.ac.jp/~naka/server/~taniguchi/a4.html
  # print len(listTrackX)
  for i in range(len(listTrackX)-1):
    for j in range(len(listTrackX)-i-1):
      # compare the jth track and (j+1)th track.
      # if the (j+1)th track is longer/straighter than jth track,
      # swap them

      # the number of sparks and the average turning angle
      # of jth and (j+1)th track
      num1 = len(listTrackX[j])/2-1 # jth track
      num2 = len(listTrackX[j+1])/2-1 # (j+1)th track
      ata1 = listTrackX[j][-1] # jth track
      ata2 = listTrackX[j+1][-1] # (j+1)th track
      # store the jth and j+1th track
      tmp1 = listTrackX[j]
      tmp2 = listTrackX[j+1]
      if (num1<num2 or (num1==num2 and ata1>ata2)):
        listTrackX[j] = tmp2
        listTrackX[j+1] = tmp1

  # sort the list in YZ-view
  for i in range(len(listTrackY)-1):
    for j in range(len(listTrackY)-i-1):
      # compare the jth track and (j+1)th track.
      # if the (j+1)th track is longer/straighter than jth track,
      # swap them

      # the number of sparks and the average turning angle
      # of jth and (j+1)th track
      num1 = len(listTrackY[j])/2-1 # jth track
      num2 = len(listTrackY[j+1])/2-1 # (j+1)th track
      ata1 = listTrackY[j][-1] # jth track
      ata2 = listTrackY[j+1][-1] # (j+1)th track
      # store the jth and j+1th track
      tmp1 = listTrackY[j]
      tmp2 = listTrackY[j+1]
      if (num1<num2 or (num1==num2 and ata1>ata2)):
        listTrackY[j] = tmp2
        listTrackY[j+1] = tmp1
```

```
# swap the first (provisional primary) and 2nd track
# if the 2nd track has one less sparks but the average turning angle
# is blow 0.9*(average turning angle of the first track)
if (len(listTrackX)>=2):
  num1 = len(listTrackX[0])/2-1
  num2 = len(listTrackX[1])/2-1
  ata1 = float(listTrackX[0][-1])
  ata2 = float(listTrackX[1][-1])
  if (num1==num2+1) and ((0.9*ata1)>=ata2):
    tmp1 = listTrackX[0]
    tmp2 = listTrackX[1]
    listTrackX[0] = tmp2
    listTrackX[1] = tmp1

if (len(listTrackY)>=2):
  num1 = len(listTrackY[0])/2-1
  num2 = len(listTrackY[1])/2-1
  ata1 = float(listTrackY[0][-1])
  ata2 = float(listTrackY[1][-1])
  if (num1==num2+1) and ((0.9*ata1)>=ata2):
    tmp1 = listTrackY[0]
    tmp2 = listTrackY[1]
    listTrackY[0] = tmp2
    listTrackY[1] = tmp1


# print listTrackX
for i in range(len(listTrackX)):
  lengthX = len(listTrackX[i])
  print "TRKX" ,
  # print lengthX,
  for m in range(lengthX-1):
   print listTrackX[i][m+1],
  print ""

# print listTrackY
for i in range(len(listTrackY)):
  lengthY = len(listTrackY[i])
  print "TRKY" ,
  # print lengthY,
  for m in range(lengthY-1):
   print listTrackY[i][m+1],
  print ""

# print deliminator
print "-1"
```

phase5 primary トラックsecondary トラックを選定する2つのプログラム(2)

primary トラックとの比較等によって secondary を選定するプログラム

```
import sys, string, re
inputName = sys.argv[1]
input = open(inputName, "r")

# boolean
true = 1
false = 0

# reconstruction constant
NDIFF = 5

#----------function compare1----------
def compare1(t1, t2):
# This function compares track1 and track2 and see if they
# share the common top spark or not.
# This function retunrs 1 if they have common top spark. 0 otherwize.

  # list to store track data (in a different format)
  track1 = []
  track2 = []
  # rearrange the track data so that we can comapre two tracks easily.
  # [TRKX, 0, 1, 2, 480.0, 481.0, 482.0, 1.0] ->
  # [[0,480.0], [1,481.0], [2,482.0]]
  nSparks = len(t1)/2-1 # the number of sparks of the track
  for i in range(nSparks):
    deck = t1[i+1]
    wire = t1[i+1+nSparks]
    track1.append([deck, wire])

  nSparks = len(t2)/2-1 # the number of sparks of the track
  for i in range(nSparks):
    deck = t2[i+1]
    wire = t2[i+1+nSparks]
    track2.append([deck, wire])
  if track1[0] == track2[0]:
    return true
  else:
    return false
#--------------------

#----------function compare2----------
def compare2(t1, t2):
# This function compares track1 and track2 and see how many sparks
# are in track2 but not in track1

  # list to store track data (in a different format)
  track1 = []
  track2 = []
  # rearrange the track data so that we can comapre two tracks easily.
  # [TRKX, 0, 1, 2, 480.0, 481.0, 482.0, 1.0] ->
  # [[0,480.0], [1,481.0], [2,482.0]]
  nSparks = len(t1)/2-1 # the number of sparks of the track
  for i in range(nSparks):
    deck = t1[i+1]
    wire = t1[i+1+nSparks]
    track1.append([deck, wire])

  nSparks = len(t2)/2-1 # the number of sparks of the track
```

```
  for i in range(nSparks):
    deck = t2[i+1]
    wire = t2[i+1+nSparks]
    track2.append([deck, wire])

  # the number of sparks which is included in track2
  # but not included in track1
  n = 0
  for i in range(nSparks):
    if (track2[i] in track1):
      n = n
    else :
      n = n +1

  return n
#--------------------
#---------- main routine ----------
# read data
while 1:
  line = input.readline()
  if not line:
    break
  columns = string.split(line)
  # print the particke info (Monte Carlo truth)
  if re.match(".*gamma.*", line):
    print line,
    listLines = [] # list to store the spark chamber
    listTrackX = [] # list to store the tracks in XZ-view
    listTrackY = [] # list to store the tracks in YZ-view

  # if the line shows the tracker hit, store them
  if re.match(".*x.*", line) or re.match(".*y.*", line):
    listLines.append(line)

  # if the line show the track, store them
  if re.match(".*TRKX.*", line) or re.match(".*TRKY.*", line):
    type = columns[0]
    tmpList = []
    for i in range(len(columns)):
      tmpList.append(columns[i])
    if (type=="TRKX"):
      listTrackX.append(tmpList)
    else:
      listTrackY.append(tmpList)

  # at the end of the event
  if (columns[0]=="-1"):
    # print out the spark chamber hits
    for i in range(len(listLines)):
      print listLines[i],

    # primary track in XZ-view and YZ-view
    if (len(listTrackX)>=1):
      primaryX = listTrackX[0]
    if (len(listTrackY)>=1):
      primaryY = listTrackY[0]
    if len(listTrackX)==0 or len(listTrackY)==0:
      continue
```

```
# list of 2ndary track candidates
listCandidateX = []
listCandidateY = []

# store the track if it satisfies the criteria as a secondary track
for i in range(len(listTrackX)-1):
  isCommon = false # if they share the common top spark or not
  numDiff = 0 # the number of different sparks

  isCommon = compare1(primaryX, listTrackX[i+1])
  numDiff = compare2(primaryX, listTrackX[i+1])
  if (isCommon==true and numDiff>=NDIFF):
    listCandidateX.append(listTrackX[i+1])

# store the track if it satisfies the criteria as a secondary track
for i in range(len(listTrackY)-1):
  isCommon = false # share the common top spark
  numDiff = 0 # the number of different sparks

  isCommon = compare1(primaryY, listTrackY[i+1])
  numDiff = compare2(primaryY, listTrackY[i+1])
  if (isCommon==true and numDiff>=NDIFF):
    listCandidateY.append(listTrackY[i+1])

# output the primary track in XZ-view
for i in range(len(primaryX)):
  print primaryX[i],
print

# sort listCandidateX (candidate of the 2nd track)
# based on the straightness of the track and
# output the 2ndary track in XZ-view
if (len(listCandidateX)>0):
 if (len(listCandidateX)>1):
  for i in range(len(listCandidateX)-1):
    for j in range(len(listCandidateX)-i-1):
      # compare the jth track and (j+1)th track.
      # if the (j+1)th track is longer/straighter than jth track,
      # swap them

      # the number of sparks and the average turning angle
      # of jth and (j+1)th track
      ata1 = listCandidateX[j][-1] # jth track
      ata2 = listCandidateX[j+1][-1] # (j+1)th track
      # store the jth and j+1th track
      tmp1 = listCandidateX[j]
      tmp2 = listCandidateX[j+1]
      if (ata1>ata2):
        listCandidateX[j] = tmp2
        listCandidateX[j+1] = tmp1

 for i in range(len(listCandidateX[0])):
   print listCandidateX[0][i],
 print

# output the primary track in YZ-view
```

```
    for i in range(len(primaryY)):
      print primaryY[i],
    print

    # sort listCandidateY (candidate of the 2nd track)
    # based on the straightness of the track and
    # output the 2ndary track in YZ-view
    if (len(listCandidateY)>0):
     if (len(listCandidateY)>1):
      for i in range(len(listCandidateY)-1):
        for j in range(len(listCandidateY)-i-1):
          # compare the jth track and (j+1)th track.
          # if the (j+1)th track is longer/straighter than jth track,
          # swap them

          # the number of sparks and the average turning angle
          # of jth and (j+1)th track
          ata1 = listCandidateY[j][-1] # jth track
          ata2 = listCandidateY[j+1][-1] # (j+1)th track
          # store the jth and j+1th track
          tmp1 = listCandidateY[j]
          tmp2 = listCandidateY[j+1]
          if (ata1>ata2):
            listCandidateY[j] = tmp2
            listCandidateY[j+1] = tmp1

     for i in range(len(listCandidateY[0])):
       print listCandidateY[0][i],
     print

   # print deliminator
    print "-1"
```

phase7 X-ワイヤ層群と Y-ワイヤ層群の間のトラックの関連付けプログラム

X-ワイヤ層群と Y-ワイヤ層群の間のトラックの関連付けを確認し、もし X-ワイヤ層群の primary と Y-ワイヤ層群の secondary トラックが対応していれば Y-ワイヤ層群の primary と secondary トラックを入れかえるようにするプログラム

```
import sys, string, re
inputName = sys.argv[1]
input = open(inputName, "r")

# boolean
true = 1
false = 0

# reconstruction parameter
THRS = 0

#----------function corr----------
def corr(tx1, tx2, ty1, ty2):
```

```
# This function calculate the match
# (i.e., the strength of the correlation) of track1s in XZ-view and YZ-view
# and track2s in XZ-view and YZ-view
  # list to store the deck number
  listDeck_tx1 = []
  listDeck_tx2 = []
  listDeck_ty1 = []
  listDeck_ty2 = []
  # store the deck number in lists
  nSparkstx1 = len(tx1)/2-1 # the number of sparks of the tracktx1
  for i in range(nSparkstx1):
    deck = tx1[i+1]
    listDeck_tx1.append(deck)
  nSparkstx2 = len(tx2)/2-1 # the number of sparks of the tracktx2
  for i in range(nSparkstx2):
    deck = tx2[i+1]
    listDeck_tx2.append(deck)
  nSparksty1 = len(ty1)/2-1 # the number of sparks of the trackty1
  for i in range(nSparksty1):
    deck = ty1[i+1]
    listDeck_ty1.append(deck)
  nSparksty2 = len(ty2)/2-1 # the number of sparks of the trackty2
  for i in range(nSparksty2):
    deck = ty2[i+1]
    listDeck_ty2.append(deck)

  # counts the number of mathes for the pair of (X1,Y1)
  # and the pair of (X2,Y2).
  # Here, a match is defined as that case where both tracks have a spark on
  # the same deck.
  c = 0
  for i in range(len(listDeck_tx1)):
    if (listDeck_tx1[i] in listDeck_ty1):
      c = c+1
  for i in range(len(listDeck_tx2)):
    if (listDeck_tx2[i] in listDeck_ty2):
      c = c+1

  return c
#--------------------

#---------- main routine ----------
# read data
while 1:
  line = input.readline()
  if not line:
    break
  columns = string.split(line)
  # print the particke info (Monte Carlo truth)
  if re.match(".*gamma.*", line):
    print line,
    listLines = [] # list to store the spark chamber
    listTrackX = [] # list to store the tracks in XZ-view
    listTrackY = [] # list to store the tracks in YZ-view
  # if the line shows the tracker hit, store them
  if re.match(".*x.*", line) or re.match(".*y.*", line):
    listLines.append(line)
```

```
# if the line show the track, store them
if re.match(".*TRKX.*", line) or re.match(".*TRKY.*", line):
  type = columns[0]
  tmpList = []
  for i in range(len(columns)):
    tmpList.append(columns[i])
  if (type=="TRKX"):
    listTrackX.append(tmpList)
  else:
    listTrackY.append(tmpList)

# at the end of the event
if (columns[0]=="-1"):
  # print out the spark chamber hits
  for i in range(len(listLines)):
    print listLines[i],


  # flag to indicate we need to swap two tracks or not
  isSwap = false

  # only when we have two tracks in XZ-view and YZ-view,
  # see if we need to swap them or not
  if (len(listTrackX)==2 and len(listTrackY)==2):
    # variables to indicate how strongly two tracks
    # (one in XZ-view and the other in YZ-view) are correlated
    c1 = 0
    c2 = 0
    # correlation when we don't swap the tracks
    c1 = corr(listTrackX[0], listTrackX[1], listTrackY[0], listTrackY[1])
    # correlation when we swap the tracks
    c2 = corr(listTrackX[0], listTrackX[1], listTrackY[1], listTrackY[0])
    val = (c2-c1)/c1
    if (val>THRS):
      isSwap = true
    else:
      isSwap = false
  # output track information
  for i in range(len(listTrackX)):
    for j in range(len(listTrackX[i])):
      print listTrackX[i][j],
    print

  if (isSwap==true):
    for i in range(len(listTrackY[1])):
      print listTrackY[1][i],
    print
    for i in range(len(listTrackY[0])):
      print listTrackY[0][i],
    print
  else:
    for i in range(len(listTrackY)):
      for j in range(len(listTrackY[i])):
        print listTrackY[i][j],
      print
```

```
    print "-1"
```

phase9 γ線到来方向を求めるプログラム

XZ面、YZ面でのγ線方向を求めて3次元的な到来方向を求めるプログラム

```
import sys, string, re, math
inputName = sys.argv[1]
input = open(inputName, "r")

# boolean
true = 1
false = 0

### reconstruction parameter
NPTMIN = 7
# mean absolute reading or digitization uncertainty in unit of
# wire number, scaled to an RMS value.
DELTA = 1.0*math.sqrt(3.1415/2)
# Threshold value to see if the spark is close to the best-fit line or not.
# When the deviation is larger than DEVCUT*uncertainty, the spark is
# regarded to deviate from the line
DEVCUT = 2.54

# position of each deck in unit of cm
listDeckPos = \
 [106.58, 104.904, 103.227, 101.551, 99.875, 98.198, 96.522, 94.846, \
  93.170, 91.493, 89.817, 88.141, 86.464, 84.788, 83.112, 81.436, \
  79.759, 78.083, 76.407, 74.730, 73.054, 71.378, 69.701, 68.025, \
  66.639, 64.672, 62.996, 61.320, 55.630, 44.510, 33.390, 22.270, \
  11.150, 4.68, 2.12, 0.00]

#-----------------function Unisparks-----------------------------------
def Unisparks(t1,t2):
# This function calculate the number of unique sparks in
# track1 and track2 (t1 and t2). The format of t1 and t2 is, e.g.,
# "[ [TX], [0], [1], [2], [496.0], [493.0], [490.0], [0.0] ]"
# Here, TX or TY indicates if the track is in XZ-view or YZ-view,
# [0], [1], and [2] are the deck numbers and
# [496.0], [493.0], [490.0] are the wire numbers.
# The last value indicates straightness of the track.
  # list to store the deck and wire number
  listDeckSpark_t1 = []
  listDeckSpark_t2 = []
  # store the deck number in lists
  nSparkst1 = len(t1)/2-1 # the number of sparks of the trackt1
  for i in range(nSparkst1):
    deck = t1[i+1]
    spark = t1[i+nSparkst1+1]
    listDeckSpark_t1.append([deck, spark])
  nSparkst2 = len(t2)/2-1 # the number of sparks of the trackt2
  for i in range(nSparkst2):
    deck = t2[i+1]
    spark = t2[i+nSparkst2+1]
```

```
    listDeckSpark_t2.append([deck, spark])

  c = 0 # the number of matches for the pair of (t1,t2)
  for i in range(nSparkst1):
    if (listDeckSpark_t1[i] in listDeckSpark_t2):
      c = c+1

  return  nSparkst1 + nSparkst2 - c
#-------------------------------------------------------------------------

#----------class lsq2----------
# This class performs the least-square fitting (x=a+b*z)
# to the data array and gives back the best fit parameters
# input: [[x1,z1],[x2,z2],[x3,z3],....], where x* and z* are the
# x-coordinate and z-coordinate.
# In this file, x is given in unit of wire number
# (so is the constant named "DELTA")
# and z in physical value of the height of the deck in cm.
# You may give x and z in different unit, but please make it sure that
# x and DELTA are given in the same unit if you will calculate the
# error of the fitting parameters.
class lsq2:
  # read data and perform the least-square fitting
  def __init__(self, list):
    self.m_sigmaX = 0.0 # summation of x
    self.m_sigmaZ = 0.0 # summation of z
    self.m_sigmaZ2 = 0.0 # summation of z^2
    self.m_sigmaZX = 0.0 # summation of z*x
    self.m_num = 0 # the number of sparks in a track
    self.m_mDev = 0.0 # sum of the square of the deviation,
                      # devided by the number of points

    self.m_num = len(list)
    for i in range(self.m_num):
      x = list[i][0]
      z = list[i][1]
      self.m_sigmaX = self.m_sigmaX + x
      self.m_sigmaZ = self.m_sigmaZ + z
      self.m_sigmaZ2 = self.m_sigmaZ2 + z*z
      self.m_sigmaZX = self.m_sigmaZX + z*x
    det = (self.m_num * self.m_sigmaZ2) - (self.m_sigmaZ * self.m_sigmaZ)
    # a of (x = a+b*z)
    self.m_offset = \
      (self.m_sigmaZ2*self.m_sigmaX - self.m_sigmaZ*self.m_sigmaZX)/det
    # b of (x = a+b*z)
    self.m_coeff = \
      (self.m_num*self.m_sigmaZX - self.m_sigmaZ*self.m_sigmaX)/det
    # mean of the square of deviation
    for i in range(self.m_num):
      x = list[i][0]
      z = list[i][1]
      x0 = self.m_offset + z*self.m_coeff
      self.m_mDev = (x-x0)*(x-x0)
    self.m_mDev = self.m_mDev/self.m_num

  # gives back the best-fit parameters
  def offset(self):
    return self.m_offset
```

```
  def coeff(self):
    return self.m_coeff

  # gives back the error of best-fit parameters
  def dOffset(self):
    det = self.m_num * self.m_sigmaZ2 - self.m_sigmaZ*self.m_sigmaZ
    val = DELTA*DELTA/det*self.m_sigmaZ2
    return math.sqrt(val)
  def dCoeff(self):
    det = self.m_num * self.m_sigmaZ2 - self.m_sigmaZ*self.m_sigmaZ
    val = self.m_num*DELTA*DELTA/det
    return math.sqrt(val)

  # gives back the mean of the square of deviation
  def mDev(self):
    return self.m_mDev
#--------------------

#----------class lsq3----------
# This class utilized the lsq2 and performs the least-square fitting (x=a+b*z)
# to the data array and gives back the best fit parameters
# input: Two list. one is
# [[x1,z1],[x2,z2],[x3,z3],....], where x* and z* are the
# x-coordinate and z-coordinate, and the other is the list of flags
# (e.g, [1,1,0,1,0,1,1]), where "1" in ith column means that the
# ith data of the first list is used for the fitting.
class lsq3:
  # read data and perform the least-square fitting
  def __init__(self, list1, list2):

    # list to store the data (list of [x, z]) for the fitting.
    # Only when the status bit is 1 (true), we use the spark
    # for the fitting
    list = []
    for i in range(len(list2)):
      if (list2[i]==true):
        list.append(list1[i])

    # perform the fitting and get the best fit values and errors
    l = lsq2(list)
    self.m_offset = l.offset()
    self.m_dOffset = l.dOffset()
    self.m_coeff = l.coeff()
    self.m_dCoeff = l.dCoeff()
    self.m_mDev = l.mDev()

  # gives back the best-fit parameters
  def offset(self):
    return self.m_offset
  def coeff(self):
    return self.m_coeff

  # gives back the error of best-fit parameters
  def dOffset(self):
    return self.m_dOffset
  def dCoeff(self):
    return self.m_dCoeff

  # gives back the mean of the square of deviation
```

```
  def mDev(self):
    return self.m_mDev
#--------------------

#----------class dircut----------
# This class corresponds to the "dircut" routine of the original
# sasge (EGRET reconstruction program). It reads the data list
# and the status list
# (to indicate which sparks to be included for the fitting)
# and search the sparks which significantly deviate from the line.
# This class also update the list of the flag based on the
# calculation of the deviation.
# input: Two list. one is
# [[x1,z1],[x2,z2],[x3,z3],....], where x* and z* are the
# x-coordinate and z-coordinate, and the other is the list of flags
# (e.g, [1,1,0,1,0,1,1]), where "1" in ith column means that the
# ith data of the first list is used for the fitting.
class dircut:
  # read data list and status list
  def __init__(self, list1, list2):
    # list of flag to indicate the data
    # to be taken into account for the fitting
    self.m_listFlag = list2[:]
    listData = [] # list of data for the fitting
    for i in range(len(list2)):
      if list2[i]==true:
        x = list1[i][0]
        z = list1[i][1]
        listData.append([x, z])
    l = lsq2(listData)
    # get parameters
    a = l.offset()
    b = l.coeff()
    da = l.dOffset()
    db = l.dCoeff()
    # list of the value of the deviation
    listDev = []
    # the number of points close to the line
    num = 0
    # scan all data points and calculate the deviation
    # and update the list of the flag
    for i in range(len(list2)):
      x = list1[i][0]
      z = list1[i][1]
      dev = abs(x-(a+b*z))
      var = da*da+db*db+z
      var = math.sqrt(var+DELTA*DELTA)
      listDev.append(dev/var) # store deviation divided by the uncertainty
      if (dev/var<=DEVCUT):
        self.m_listFlag[i] = true
        num = num+1
      else:
        self.m_listFlag[i] = false

    # make it sure that we have 3 or more points close to the line
    if (num<3):
      tmpList = listDev[:]
      tmpList.sort
```

```
      for i in range(3):
        index = listDev.index(tmpList[i])
        self.m_listFlag[index] = true

  # gives back the list to indicate the point close to the line
  def listFlag(self):
    return self.m_listFlag
#--------------------

#---------- main routine ----------
# read data
while 1:
  line = input.readline()
  if not line:
    break
  columns = string.split(line)
  # print the particke info (Monte Carlo truth)
  if re.match(".*gamma.*", line):
    eventLine = line
    eventNo = int(columns[1])
    listLines = [] # list to store the spark chamber
    listTrackX = [] # list to store the tracks in XZ-view
    listTrackY = [] # list to store the tracks in YZ-view
  # if the line shows the tracker hit, store them
  if re.match(".*x.*", line) or re.match(".*y.*", line):
    listLines.append(line)

  # if the line show the track, store them
  if re.match(".*TRKX.*", line) or re.match(".*TRKY.*", line):
    type = columns[0]
    tmpList = []
    for i in range(len(columns)):
      tmpList.append(columns[i])
    if (type=="TRKX"):
      listTrackX.append(tmpList)
    else:
      listTrackY.append(tmpList)

  # at the end of the event
  if (columns[0]=="-1"):

    # the variable below indicates if the event is classified as
    # "gamma-ray" or "possible gamma-ray" or "non gamma-ray".
    # If the value is 2, the event is classified as gamma-ray.
    # If the value is 1, the event is classified as "possible gamma-ray".
    # If the value is 0, it is classified as "non gamma-ray"
    # and no further analysis is applied.
    gammaLevel = 2

    # see if XZ-view and YZ-view have primary and 2ndary tracks.
    # If at least one of two views has primary and secondary,
    # the event is classified as gamma.
    # If both view have only primary,
    # the event is classified as "non gamma-ray"

    # when at least XZ-view or YZ-view has primary and secondary
    if len(listTrackX) > 1 or len(listTrackY) >1 :
      gammaLevel = 2
```

```
else:  # when both XZ-view and YZ-view have primary only
  gammaLevel = 0

# see if XZ-view and YZ-view have more than NPTMIN(7)
# unique sparks. Here, the number of unique sparks is defined as
# (the number of sparks in primary track)
# + (the number of sparks in 2ndary track)
# - (the number of sparks shared by primary and 2ndary tracks)
# If at least one of two views has more than NPTMIN,
# the event is classified as gamma. If both view has
# less than NPTMIN, the event is classified as "non gamma-ray"

if gammaLevel != 0:
  # when XZ-view has only primary track, the number of unique sparks
  # is the number of sparks in the track. Otherwize, it is defined as
  # the number of unique sparks in primary and 2ndary tracks.
  if (len(listTrackX)==1):
    NPTX = len(listTrackX[0])/2-1
  else:
    NPTX = Unisparks(listTrackX[0],listTrackX[1])

  # when YZ-view has only primary track, the number of unique sparks
  # is the number of sparks in the track. Otherwize, it is defined as
  # the number of unique sparks in primary and 2ndary tracks.
  if (len(listTrackY)==1):
    NPTY = len(listTrackY[0])/2-1
  else:
    NPTY = Unisparks(listTrackY[0],listTrackY[1])

  # when at least XZ-view or YZ-view has unique sparks more than
  # NPTMIN, the event is classified as gamma. Otherwize,
  # the event is classified as "non gamma-ray"
  if NPTX >= NPTMIN or NPTY >= NPTMIN:
    gammaLevel = 2
  else:
    gammaLevel = 0
else:
  gammaLevel = 0

# when the event is classified as gamma
if (gammaLevel>=1):

  # print out the incident gamma-ray info and the spark chamber hits
  print eventLine,
  for i in range(len(listLines)):
    print listLines[i],

  # calculate the direction of the track
  for i in range(len(listTrackX)):
    # list to store [x,z] data
    list = []
    nSparks = len(listTrackX[i])/2-1
    for j in range(nSparks):
      x = float(listTrackX[i][j+1+nSparks])
      z = listDeckPos[int(listTrackX[i][j+1])]
      list.append([x,z])

    # initial data list for the fitting and status list
```

```
iniList = []
statusList = []
if (len(list)==3):
  iniList = [list[0], list[1], list[2]]
  statusList = [true, true, true]
else:
  iniList = [list[0], list[1], list[2], list[3]]
  statusList = [true, true, true, true]

# initial deviation test. when the number of sparks is 4
# and one of them deviate from the best-fit line significantly,
# not include the spark for the fitting.
d = dircut(iniList, statusList)
statusList = d.listFlag()

# flag to indicate if the sparks are added to form the
# straight line or not.
isAdd = true
while(isAdd==true and len(statusList)<len(list)):
  # perform the fitting
  l = lsq3(iniList, statusList)
  # get parameters
  a = l.offset()
  b = l.coeff()
  da = l.dOffset()
  db = l.dCoeff()
  # the number of sparks (either "true" or "false" status)
  # already used
  nmin = len(statusList)
  # the number of sparks to be used for the fitting
  nmax = min(nmin+3, len(list))
  isAdd = false
  for j in range(nmin, nmax):
    x = list[j][0]
    z = list[j][1]
    iniList.append([x,z])
    # calculate the deviation
    dev = abs(x-(a+b*z))
    var = da*da+db*db+z
    var = math.sqrt(var+DELTA*DELTA)
    if (dev/var<=DEVCUT):
      statusList.append(true)
      isAdd = true
    else:
      statusList.append(false)

# now we know the straight part of the track.
# apply the fitting to get the parameters
l = lsq3(iniList, statusList)
num = 0 # the number of points used for the fitting
for j in range(len(statusList)):
  if (statusList[j]==true):
    num = num+1
print "TRKX",
for j in range(1, len(listTrackX[i])):
  print listTrackX[i][j],
print
if (i==0):
```

```
    print "priX(#,offset,coeff,dOffset,dCoeff,mDev)",
  else:
    print "sndX(#,offset,coeff,dOffset,dCoeff,mDev)",
  print num, l.offset(), l.coeff(), l.dOffset(), l.dCoeff(), l.mDev()

for i in range(len(listTrackY)):
  # list to store [x,z] data
  list = []
  nSparks = len(listTrackY[i])/2-1
  for j in range(nSparks):
    x = float(listTrackY[i][j+1+nSparks])
    z = listDeckPos[int(listTrackY[i][j+1])]
    list.append([x,z])

  # initial data list for the fitting and status list
  iniList = []
  statusList = []
  if (len(list)==3):
    iniList = [list[0], list[1], list[2]]
    statusList = [true, true, true]
  else:
    iniList = [list[0], list[1], list[2], list[3]]
    statusList = [true, true, true, true]

  # initial deviation test. when the number of sparks is 4
  # and one of them deviate from the best-fit line significantly,
  # not include the spark for the fitting.
  d = dircut(iniList, statusList)
  statusList = d.listFlag()

  # flag to indicate if the sparks are added to form the
  # straight line or not.
  isAdd = true
  while(isAdd==true and len(statusList)<len(list)):
    # perform the fitting
    l = lsq3(iniList, statusList)
    # get parameters
    a = l.offset()
    b = l.coeff()
    da = l.dOffset()
    db = l.dCoeff()
    # the number of sparks (either "true" or "false" status)
    # already used
    nmin = len(statusList)
    # the number of sparks to be used for the fitting
    nmax = min(nmin+3, len(list))
    isAdd = false
    for j in range(nmin, nmax):
      x = list[j][0]
      z = list[j][1]
      iniList.append([x,z])
      # calculate the deviation
      dev = abs(x-(a+b*z))
      var = da*da+db*db+z
      var = math.sqrt(var+DELTA*DELTA)
      if (dev/var<=DEVCUT):
        statusList.append(true)
        isAdd = true
```

```
      else:
        statusList.append(false)

  # now we know the straight part of the track.
  # apply the fitting to get the parameters
  l = lsq3(iniList, statusList)
  num = 0 # the number of points used for the fitting
  for j in range(len(statusList)):
    if (statusList[j]==true):
      num = num+1
  print "TRKY",
  for j in range(1, len(listTrackY[i])):
    print listTrackY[i][j],
  print
  if (i==0):
    print "priY(#,offset,coeff,dOffset,dCoeff,mDev)",
  else:
    print "sndY(#,offset,coeff,dOffset,dCoeff,mDev)",
  print num, l.offset(), l.coeff(), l.dOffset(), l.dCoeff(), l.mDev()

print "-1"
```

謝辞

本論文を作成するにあたって、論文の書き方やをはじめ、いろいろと丁寧にご指導くださった、大杉節氏、指導教官の深沢泰司氏に厚く感謝申し上げます。そして水野恒史氏には、Python、論文の書き方等も含め、本研究全般に渡りアドバイスや丁寧な指導をいただいた事に大変感謝致します。
　また、プログラム作成の際に親身になって相談にのって下さった河本卓也氏、河嶋健吾氏、その他いろいろな質問に答えて下さった川埜直美、阿倍由紀子氏をはじめ、御世話になった先輩方にも感謝致します。最後に、改めてこの論文作成の際に御世話になったみなさまありがとうございました。

参考文献

D. Thompson et al. 1993, ApJS 86, 692
D. Thompson et al., 1993, ApJ 415, L13
S.D. Hunter et al. 1997, ApJ 481, 205
A. Strong et al. 2004, ApJ 613, 962

佐藤葉子：2001 年度 広島大学 卒業論文 『加速器ビーム実験による GLAST 衛星搭載ガンマ線検出器のシミュレータの評価』

畦地康弘：2000 年度 広島大学 卒業論文 『宇宙ガンマ線観測衛星 GLAST の検出器シミュレーターの評価と応答関数の研究』