結果の取得
Scoring

Geant4 10.3.p3準拠

Geant4　HEP/Space/Medicine 講習会資料

KEK 大学共同利用機関法人 高エネルギー加速器研究機構

本資料に関する注意

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

本講の目標

Scoringのやり方はいろいろだが、本稿では有感検出器に焦点を当てる

- Geant4のScoring手法を、ユーザフックだけを使った自前のスコアリングの概要から理解する
  - SteppingAction, EventAction, TrackingAction, RunAction
- Geant4が提供する汎用のスコアリングの手法を知る
  - G4VSensitiveDetector: 有感検出器の機能とその使い方
  - G4VHits: ヒット、ヒットコレクション、ヒットマップ　データの保管
  - Handson P05_Scoringのコードによる具体的な説明

予備資料

- Geant4が提供する既製のスコアリングの概要
  1. Primitive Scorer 原始スコアラの簡単な紹介
  2. Command-line Scorer　について少し

参考する他のコース

「構造体の形状定義」「ユーザクラスの記述方法」「解析」
Standard Template Libraryの教科書も

目次

1. Geant4のスコアリングとは
    1. スコアリングの課題
2. 物理量はどこから取得するか
3. Scoringの前準備
4. ユーザアクションによるScoring
5. Sensitive Detector 有感検出器
6. Hands on P05_Scoring/source-SDに沿って解説
7. ヒット情報とヒットコレクション

予備資料

A) 多機能検出器(Multi-Functional Detector)と具象原始スコアラPrimitive Scorer

B) コマンドベーススコアリング

[謝辞] 本講義で使用している資料は、過去にSLAC、CERN、IN2P3、ESAなどが主催したGeant4チュートリアルで使用されたスライドの内容を多く含みます。
これらのスライド作成に寄与したGeant4 Collaborationメンバーに謝意を表します。

Geant4のScoringとは

検出器から得られる物理量のいろいろ

- HEP、宇宙物理や医学応用で求めたい典型的な物理量の例
  - 粒子の位置検出器
    - 位置情報から飛跡を求める
      - 入射粒子の種類、位置、時刻など
  - 粒子のエネルギー
    - 全吸収型カロリメータ、一様物質やサンドウィッチ型
      - 入射粒子のエネルギースペクトル
      - エネルギーデポジットの総和
      - 人体組織のどこにどれだけエネルギーが吸収されたか
- 本講習会で扱うHands on 演習問題
  - Pixel 検出器　pixel 素子への入射位置分布とエネルギー損失
    - /source-SD

- 予備の自習問題
  - BGO結晶カロリメータ　エネルギーデポジットとそのスペクトル
    - /source/ (ユーザアクションを使って自前で)
    - /source-PS/ (原始スコアラ)
  - 水ファントムでの深度線量分布
    - /source-CS/ (コマンドラインスコアラ)

スコアリング

- Geant4の既定実行モードとは
  - 3つの必須情報(Geometry, Physics, Primary Particle)が*G4RunManager*に渡されれば、粒子輸送(tracking)を実行できる
    - ✓ /run/beamOnでシミュレーションが開始され、指定したイベント数が終わったら、実行の最小情報を書き出した後、黙ってシミュレーションを終了してしまう
    - ✓ Verbose モードを上げれば詳しい経過情報を文字方法として得ることはできるし、
    - ✓ 可視化をすれば様子がわかるが、
- シミュレーションの定量的な結果を得るにはスコアリングが不可欠である
  - 検出器システムのどこでどれだけのどのような物理量が発生したかを求めるために、ユーザは自分の要求に応じるコード(通常C++)を書く必要がある => 通称 Scoring
- すべての情報はステップ(G4Step)にあるのだが、それには二つの課題がある
  1. ステップが特定の検出器の中にあることの判定
  2. そこで欲しい物理情報を取得すること

課題その1： ステップからヒットオブジェクトを得る

- 検出器の中での相互作用により発生する物理量
  - 必要なすべての情報はステップ(G4Step)にある
    - G4Stepは様々な物理量を獲得する get メソッドを用意している
  - 欲しい物理量を得るには => ヒット情報
    - ステップはシミュレーションを進めるための人工物なので、ステップ単位での物理量そのものが欲しいものとは限らないこともある。
    - ステップから直接得られる意味ある物理量； 例 入射位置
    - 複数のステップからの物理量の和がイベント毎に欲しい場合： 例 energy depojit
    - イベント毎の物理量は不要で、ランでの積算量が欲しい場合： 例 吸収線量
  - ヒットオブジェクトの取得、保存と処理は、自分の要求に応じて自作するのが基本。
- 取得したヒットオブジェクトの保存
  - 取得したオブジェクトの一時保存のやり方
    - 簡単なデータなら自前で処理できる
    - 複雑なデータなら、Geant4が用意するデータ構造を使って格納できる => ヒットコレクション
- ヒット情報の処理 ：講義「解析ツールとのインターフェイス」
  - スコアリングとは直接関係ないが、ヒット情報と解析ツールとの接続ができる
  - 取得したエネルギー損失などの物理量をあらかじめ定義しておいたヒストグラムや散布図Ntupleなどに書き込む

課題その2:ステップが検出器内にあることの判定

- 複雑な構造物の中で、支持体のようなドンガラなどの構造物と、物理情報を提供する検出器本体部分を識別し、ステップがその中にあることを判別してヒットを得る

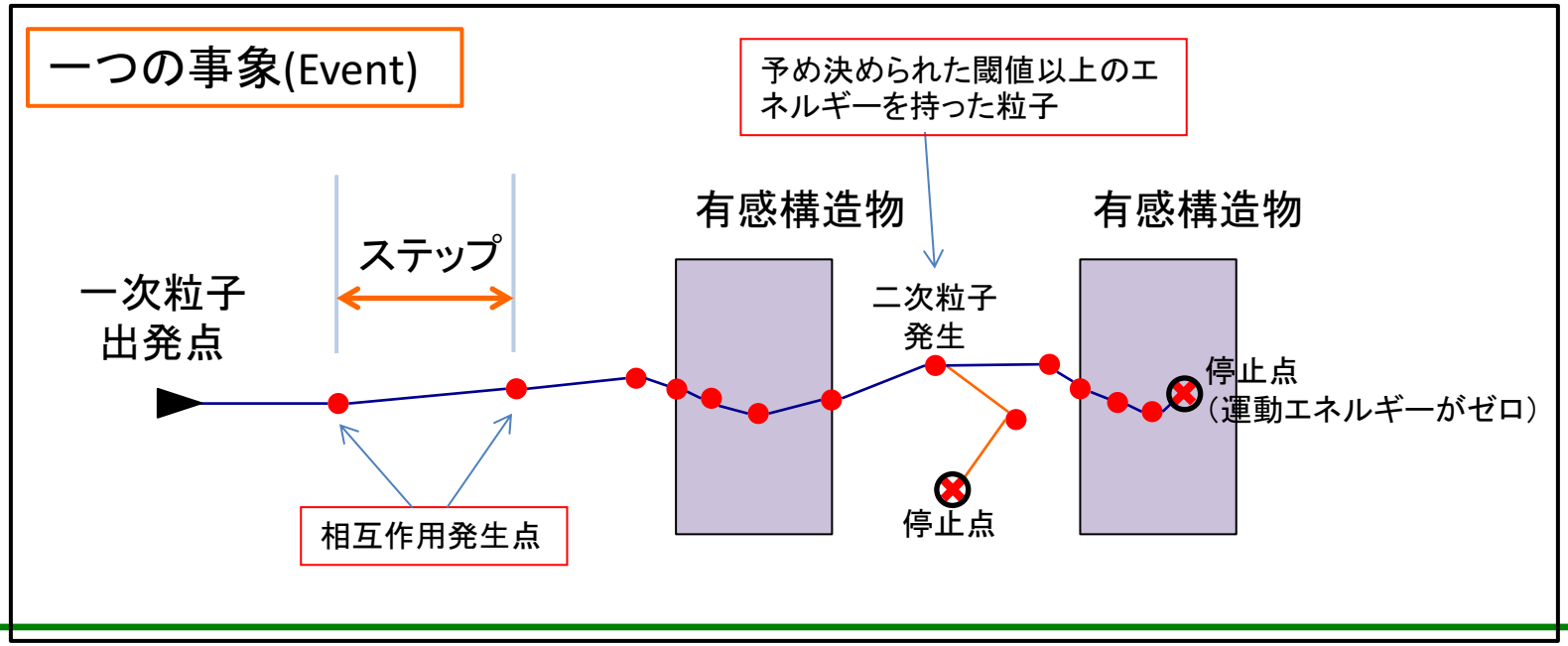
1. 自前のユーザアクションを使う方法:
   - 構造物定義クラスにおいて、対象とする検出器に自分なりに印をつけておき、シミュレーションの進行につれて、ステップがそこにあるかどうかを毎回ユーザアクションでチェックするというやり方
2. 有感検出器を使う方法: Sensitive Detector 本稿のテーマ
   - Geant4が提供する有感検出器クラスは、ステップが有感検出器と割付された構造体の中にあればヒットオブジェクトを処理するための手続関数が自動的に呼ばれる
   - イベント単位でヒットオブジェクトをイベントアクションへ渡すことができる
   - MySensitiveDetecorを作成し、それを構造物定義クラスの中で登録しておけば、ヒットオブジェクトの処理手続きを実装するだけで良い。

物理量をどこから取るか

ユーザはどうやってカーネルに介入できるか

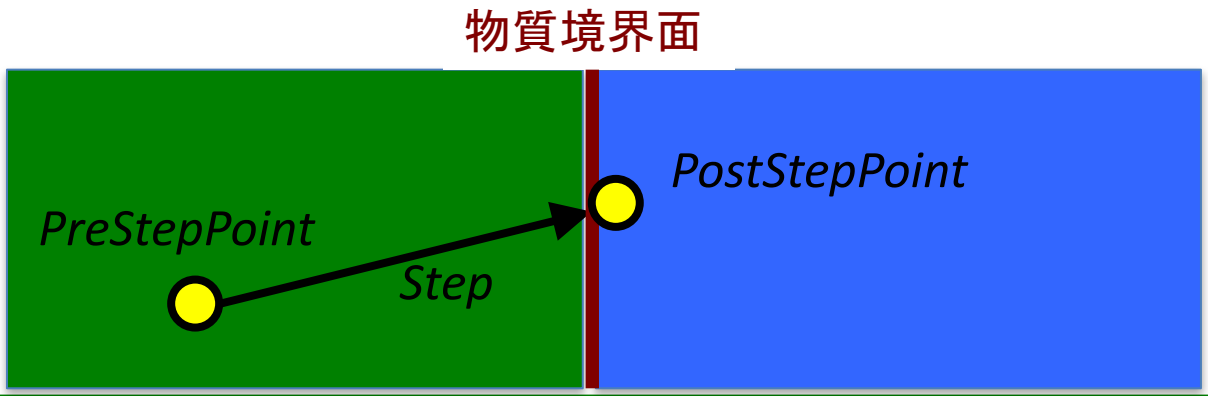
復習　情報取得はステップから

- シミュレーションは粒子進路に沿ったステップ（step）単位で進行
  - ✓ ステップ単位で粒子と物質の相互作用（物理：physics）が考慮される
  - ✓ 粒子が構造物の境界面を通過すれば、通過点で一つのステップは終了
  - ✓ 粒子の1ステップ長だけの時空間移動（transportation）も相互作用として扱う
- 粒子は運動エネルギーがゼロになれば停止し、ステップの進行は終了
- 一つの粒子を発生から停止まで進行させることをtrackingとよぶ



ステップの両端点で得られる情報

- Stepの始点と終点は'*G4StepPoint*'クラスで表現される。
  - ✓ Stepの始点と終点はそれが属する物体・物質情報を保持する。
  - ✓ Stepの終点が物質境界面にある場合、終点は境界面の先にある物体の中に入ったと解釈
- G4Stepは、始点(*Pre-step Point*)と終点(*Post-step point*)の間での粒子の物理量の変化（エネルギー損失、経過時間、等）の情報を保持。
- '*G4UserSteppingAction*'クラスは stepping中に必要に応じてユーザが介入するためのユーザ・フック



Stepの状態からわかること

- Stepの状態は'*G4StepPoint*'が保持し、ユーザは現stepが決定された原因の情報をそこから得る事が出来る
  - ✓ 現stepの状態は'*PostStepPoint*'から得られる
  - ✓ 前stepの状態は'*PreStepPoint*'から得られる
- Stepの状態は*G4StepStatus*列挙型(Enumerator)

  で表現され、以下の列挙定数が定義されている:
  - ✓ fWorldBoundary: Stepがworld境界に到達した
  - ✓ fGeometryBoundary: Stepが物質境界面に到達した
  - ✓ fAtResDoItProc, fAlongStepDoItProc, fPostStepDoItProc

    Stepがそれぞれの相互作用過程で決定された
  - ✓ fUndefined: Stepはまだ決定されていない
- Stepが在る物体の情報を得るには
  - ✓ 現stepの始点の物体(volume)の情報を入手したい場合: *PreStepPoint*にある'*fGeomBounary*'を参照する
  - ✓ 次のstepの始点の物体(volume)の情報を入手したい場合:*PostStepPoint*にある'*fGeomBounary*'を参照する

[注] G4StepStaus値の使用例は、例えばGeant4 home pageのFAQのFAQ.4参照

具体例　Geant4 Web Q&Aから抜粋

ステップが入った物体のコピー番号、名前、論理物体、物質、親物体などなど
G4Step * step として　このステップが通過した物体を「現物体」と呼ぶことにする

```
G4StepPoint* point1 = step->GetPreStepPoint();                    ステップの開始点をGet
if (point1->GetStepStatus() == fGeomBoundary)                     ステップが現物体に入った
                        つまり、このステップが現物体中での最初のステップであることの判定
```

- G4TouchableHandle touch1 = point1->GetTouchableHandle();　タッチャブルハンドル
- G4VPhysicalVolume* volume = touch1->GetVolume();　現物体（物理物体）
- G4int　copyNumber = touch1->GetCopyNumber();　そのコピー番号

```
   G4String   name = volume->GetName();                           その名前
```

- G4LogicalVolume* lVolume = volume->GetLogicalVolume();　その論理物体

```
   G4Material*          material = point1->GetMaterial();         現物体の物質
   G4Material*        material = lVolume ->GetMaterial();         上に同じ
```

- G4ThreeVector　worldPos1 = point1->GetPosition();　開始点の世界座標

```
   G4ThreeVector     localPos1  = touch1->GetHistory()->          物体に相対的な
                        GetTopTransform().TransformPoint(worldPos1); ローカル座標
```

- G4VPhysicalVolume* mother = touch1->GetVolume(depth=1);　現物体の親物体
- G4int　copyNumber = touch1->GetCopyNumber(depth=1);　そのコピー番号
- 印のついているものはhandsonで使う

続き

ステップから得られる物理量

- G4double eDeposit = step->GetTotalEnergyDeposit(); 全エネルギー付与量
  エネルギー損失と、カット以下で生成されなかった二次粒子のエネルギー損失の和

G4double sLength = step->GetStepLength();
多重散乱でのジグザグも含むステップ長
G4double deltaEne = step->GetDeltaEnergy(); ステップ前後でのエネルギー変化
G4ThreeVector displace = step->GetDeltaPosition(); “ 位置座標の変化量
G4double tof = step->GetDeltaTime(); “ 飛行時間

PostStepPoint から得られるもの

G4StepPoint* point2 = step->GetPostStepPoint(); ステップの終了点
if (point2->GetStepStatus() == fGeomBoundary) ステップが現物体の境界に達した
つまり、現ステップが現物体内での最後のステップである
G4TouchableHandle touch2 = point2->GetTouchableHandle(); 次の物体のタッチャブル

G4VProcess* aProcess = point2->GetProcessDefinedStep(); 今のステップを決めたプロセス

復習　G4Track クラス

- Trackとは粒子飛跡のスナップショット
  - ✓ Stepはtrackingの最小単位であり、G4Trackへのポインタを持つ
  - ✓ Trackはtrackingされている粒子の現時点での物理情報を保持。
  - ✓ Trackは複数のStepを記述しないし、過去の情報も保持しないで、対象粒子が以下の条件になると廃棄される
    - 粒子がワールドボリュームから出た時、粒子が崩壊、非弾性散乱等で消滅した時、粒子の運動エネルギーがゼロになり（すなわち静止状態）、さらに静止状態で相互作用（例えば、崩壊）が生じない場合
    - ユーザが強制的に粒子のtrackingを停止した時
  - ✓ Trackの保持する情報は事象(event)処理の終了時に破棄
    - Track情報を保持したい場合は、'G4Trajectory'クラスを用いる
- *'G4UserTrackingAction'*クラスはユーザがtrackingに介入するためのユーザ・フック

G4Trackから得られる情報

- G4Track* track = step->GetTrack(); 今のステップが終わった時点で得る

G4String ParticleName = track->GetDynamicParticle()-> 粒子名
GetParticleDefinition()->GetParticleName();

G4ThreeVector momentum = track->GetMomentum(); 運動量

G4double kinEnergy = track->GetKineticEnergy(); 運動エネルギー

G4double globalTime = track->GetGlobalTime(); 世界時間
現イベントの開始時からの経過時間

track->GetNextVolume() 次のVolume

- track->SetTrackStatus(fStopAndKill) 今のステップでtracking をお終いとする
- Track->GetTrackID() TrackID=1 なら一次粒子

G4String creatorProcName = 自分を産んだ親プロセス名をGet
track->GetCreatorProcess()>GetProcessName();

Geant4/track/include/G4Step.hh, G4StepPoint.hh, G4Track.hh を読むこと

復習　Eventはヒットオブジェクトを取りまとめる

- ‘Event‘（事象）の基本的な意味
  - 入射粒子（一次粒子：複数可）が物質を通過した際に生じる全ての物理現象をさす
  - シミュレーションにおける表現の基本単位
- *’G4Event’*クラスは一つの‘Event‘全体を表現。このオブジェクトは一つの‘Event‘シミュレーションの正常終了時に以下のオブジェクトを保持できる
  - 全ての一次粒子の情報及びそれらの空間始点情報　（初期情報）
  - Hits(後述)とTrajectory(後述)の総合情報　（結果情報）
- *’G4UserEventAction’*クラスは‘Event‘処理中に必要に応じてユーザ介入するためのユーザ・フック。

Scoring の前準備

考えるべきこと

- どの物体／検出器でなにをスコアリングするか
  - 検出器の外にあるステップは興味ない。
  - ステップが検出器に入り、反応を起こしながらステッピングを進めて、検出器から出るのをどうやって調べるか
  - 検出器の種類(位置、エネルギーなど)と検出器の論理物体の数が多いときは次のヒット情報のことも考えねばならない
- 検出器で何を得るのか
  - ステップから多種多様なヒット情報が得られるが
  - 欲しい条件を満たす情報をどうやって得るか
- ヒット情報をどのように処理するか
  - イベント単位でヒット情報が一時保存されるが、
  - それを後で解析するためのデータ構造をどうするか
  - 外部の解析ツールには何を選ぶか

Geant4/examples/basic で見るスコアリングの具体例

- Geant4/examples/basic は全てスコアリングの例である。検出器とスコア量

1. B1 : 二つの物体での全吸収線量
2. B2 : 飛跡検出器 (tracking chambers) 飛跡情報
3. B3 : PET ファントムと円筒状配列のガンマ線検出器 対のヒット情報
4. B4 : サンドウッチカロリメータ エネルギーdeposit 情報
5. B5 : single arm spectrometer , 位置及びエネルギー検出器

- 中でもB4は4種類のスコアリング手法の例

B4では4つのスコアリング手法を紹介している

A. B4a : ユーザアクション (stepping, event, run)を使って全て自前でやる。ステップがabsorber またはgapに含まれて居るなら、ステップごとに得られたエネルギーデポをイベントに渡し、ランの終わりに全absorberと全gapの全デポ量を得る。

B. B4b : B4aと同様にステップが物体に含まれて居るかを判定し、エネルギーデポを自前の二次元配列に記入し、ランの終わりにすることはB4aと同じ。

C. B4c : absorberとgapの論理物体をSensitive Detector (有感検出器)とする。ステップが有感検出器に在れば自動的にヒット処理が呼ばれる。ランの終わりに、各吸収層と検出層毎のデポジット量も求める。

D. B4d : G4のエネルギーデポのためのPrimitive Scorerを使う

■ ステップから情報を引き出すのは共通だが、大別して

- ユーザアクションでステップの在り処を調べ自前のデータ処理を行うか
- 有感検出器とヒット情報を使うか
- データオブジェクトをどうするか

UserActionによる　Scoringのポイント

簡単な検出器の場合

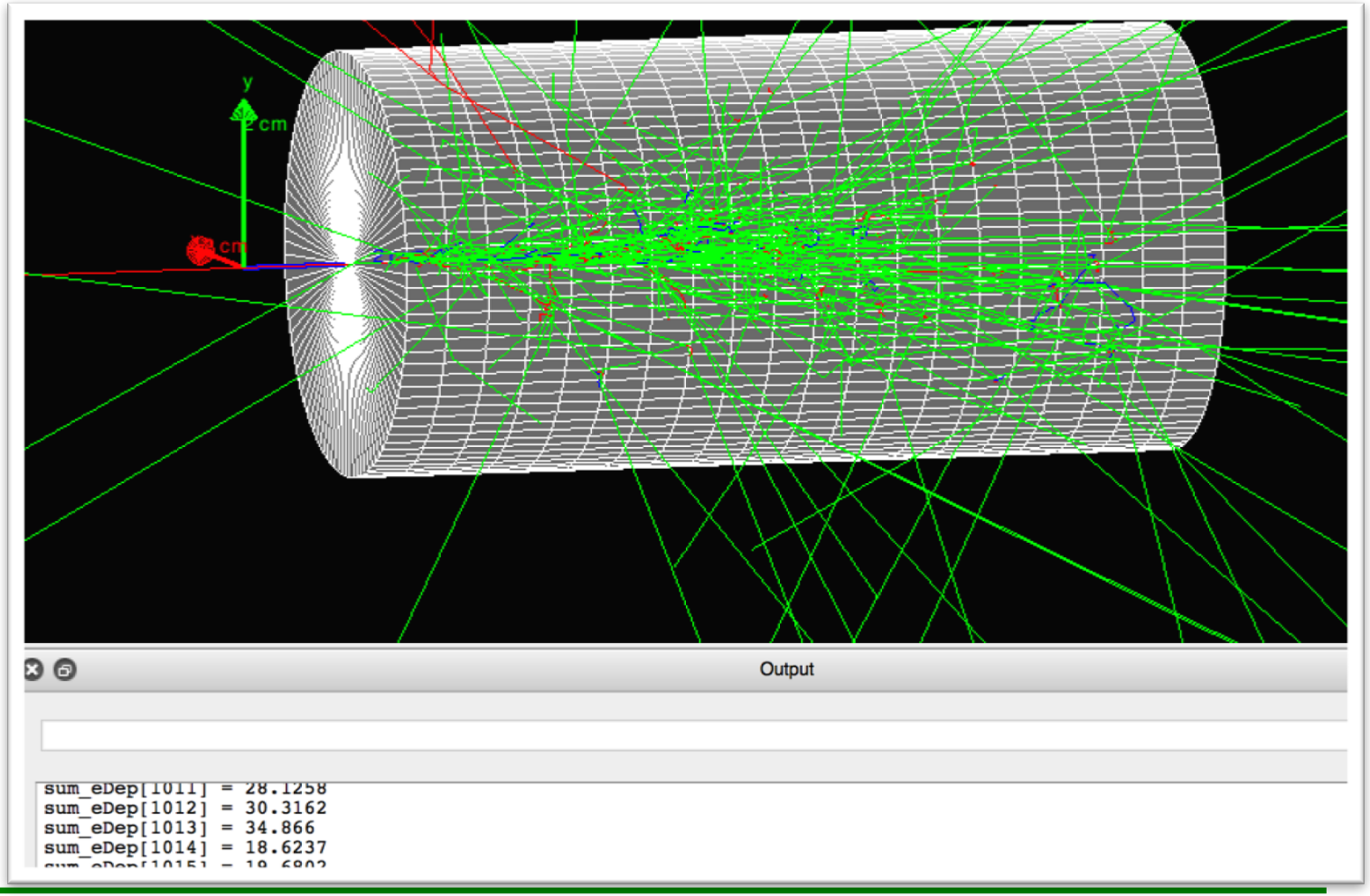
ユーザフックでScoringするには

1. ユーザフックのために5つの基底クラスが用意されている。それらのメンバー関数はダミーなので、必要ならそれを実装する
   1. ステップアクション　G4UserSteppingAction
      - ✓ Stepが目的の検出器/物体に入ったかどうかを調べ、そこにステップがあるなら、欲しい物理量を取得する
   2. G4UserTrackingAction, G4UserStackingActionで記述できること
      - ✓ この二次粒子はトラッキングしないなど
      - ✓ このトラック情報はスタックに入れないなど
   3. G4UserEventAction
      - ✓ イベントごとにデータの初期化や集計などの処理をする
   4. G4UserRunAction
      - ✓ ランの前準備とまとめを行う
   5. イベントやランのユーザアクションはそれぞれ最初と最後に何をするかを記述する
      - ✓ 例 BeginOfRun(Event)Action, EndOfRun(Event)Action

- ➢ これらのアクションはG4RunManagerに登録するには
  - ✓ G4VUserActionInitializationを実装する

P05_Scoring/source 自習Handson問題

円筒形のBGO結晶を20個並べた全吸収型カロリメータ
　　各結晶でのエネルギー付与(deposit)を得て電磁シャワーの発達を見る



P05_Scoring/source ユーザフックの実装

■ ユーザフック: SteppingAction, EventAction, RunAction

1. Geometry.cc クラス
   - ✓ fScoringVol = logVol_BGO; //BGO論理物体をスコアリング対象とする印をつける
   - ✓ それをコピー番号を付けて20個並進配置する(G4PVPlacement)
2. RunAction.cc クラス
   - ✓ 20個のヒストグラムの初期化と結果の書き出し
3. SteppingAction.cc クラス
   1. ステップがGeometryで指定されたfScoringVolの中にあるかどうか、そうならば、そのコピー番号は?
   2. ステップでのエネルギー付与を獲得
   3. コピー番号ごとのエネルギー付与に加算。
4. EventAction.cc クラス
   - ✓ EndOfEventActionでは積算エネルギー付与を各ヒストグラムに記入
5. UserActionInitialization.cc クラス
   - ✓ ユーザアクションの登録と初期化

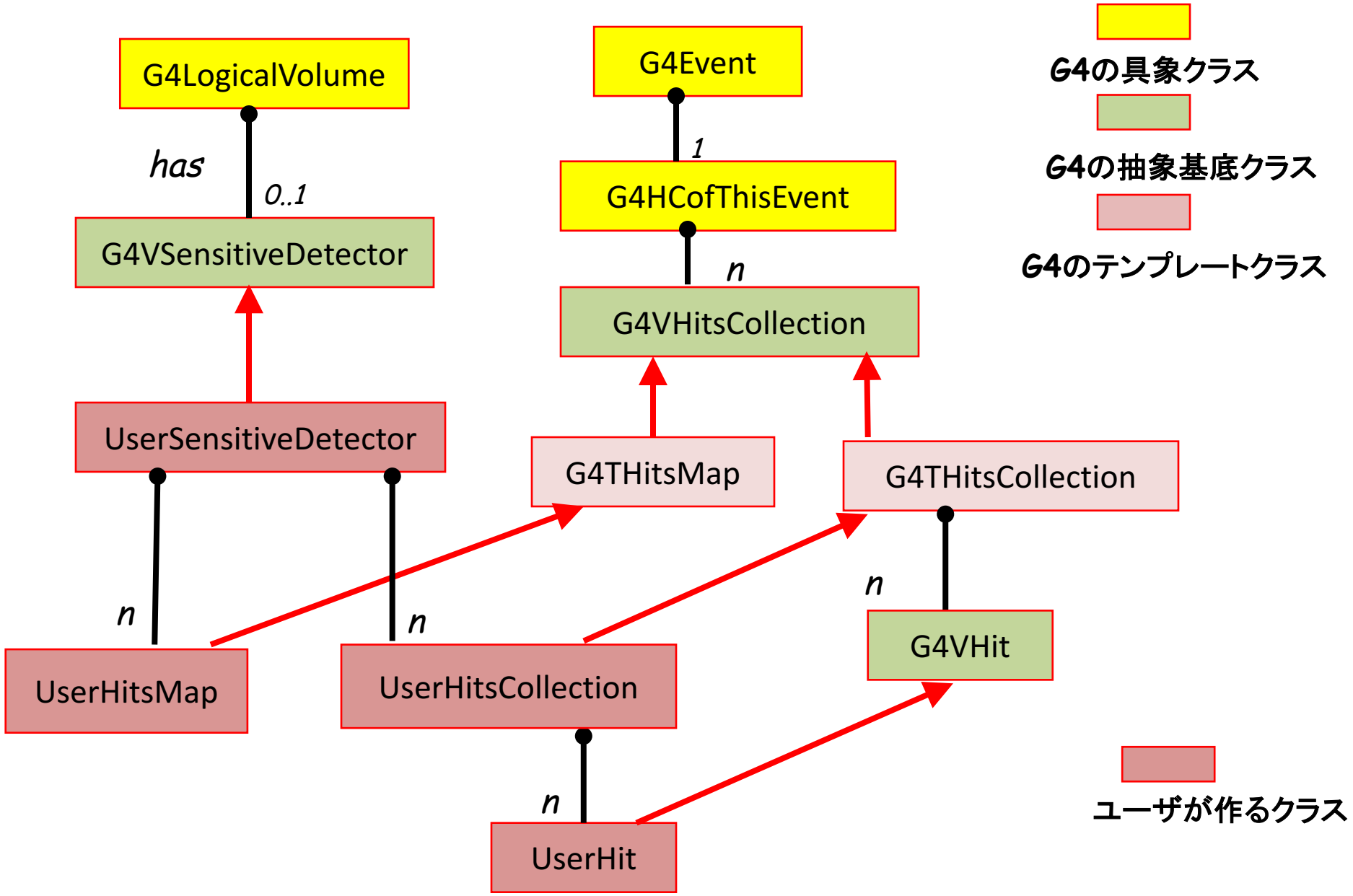
SensitiveDetector: 有感検出器

Geant4が提供する汎用のスコアリング機能

Sensitive Detector 有感検出器

- 大規模な実験装置では、
  - 同じ検出器(論理物体)が多数、コピー、レプリカ、パラメトリゼーションなどを使って空間に配置される（物理物体）のが普通である。
  - 種類の異なる検出器群がサブシステムを構成するのが普通である
- G4VSensitiveDetectorは検出器を表現する抽象クラスで、検出器内の粒子の飛跡に沿ってステップからヒットオブジェクトを集めるのが仕事である。
  - 論理物体に有感検出器を割付できる
    - この論理物体のコピー、レプリカ、パラメトリゼーションで作られる多数の物理物体は全て同じ有感検出器として扱える
  - ステップが有感検出器に在れば、自動的にヒットを処理する手続きが呼ばれる
    - ヒット情報の蓄積格納機能を有し、イベントからデータ構造へのアクセスができる
- 本講では主にこれを説明する。
  - 次のクラス図がクラス間の関係を示す。また、ユーザが実装すべきクラスも。
    - 有感検出器、 論理物体
    - ヒットデータ、 ヒットデータのコレクション
    - イベント

有感検出器関係のクラス図



SensitiveDetectorの実装とヒットオブジェクト処理の実装

- 抽象基底クラスG4VSensitiveDetectorを継承してUserSensitiveDetectorクラスを実装する。
- G4VSensitiveDetectorはヒットオブジェクトを作るための3つの仮想関数を持つ
  - ProcessHits() 関数
    - 検出器中でステップの発生毎にProcessHits(G4Step*)関数が自動的に呼ばれるので、G4StepやG4Trackから情報を集められる
    - ステップが有感検出器に出入りした回数だけ実行される
  - Initialize()とEndOfEvent()関数
    - イベントの最初と最後に呼ばれ、普通は複数回実行される*ProcessHits()*で得られるヒット情報の取り扱いを担当する
      - Initialize()でヒットオブジェクトを初期化し
      - EndOfEvent()で取りまとめる
      - UserEventActionとのタイミング関係
        - » BeginOfEventActionの後でInitialize()
        - » EndOfEventAction より前にEndOfEvent()
- ヒットオブジェクトについて、簡単な場合は自前でやれるがGeant4が提供するヒットオブジェクト機能を使うと、
  - UserHitMapは連想配列G4VHitMapを実装し、検出器要素番号とそこでの物理量を保存する
  - UserHitCollectionはG4VHitsCollectionを実装し、任意の物理量を格納するヴェクトルの集合である。
- 複数の有感検出器を必要とする場合には、検出器毎にヒットオブジェクトを扱うクラスを用意するのがわかりやすい。(examples/basic/B5の例)

SensitiveDetectorの割付:

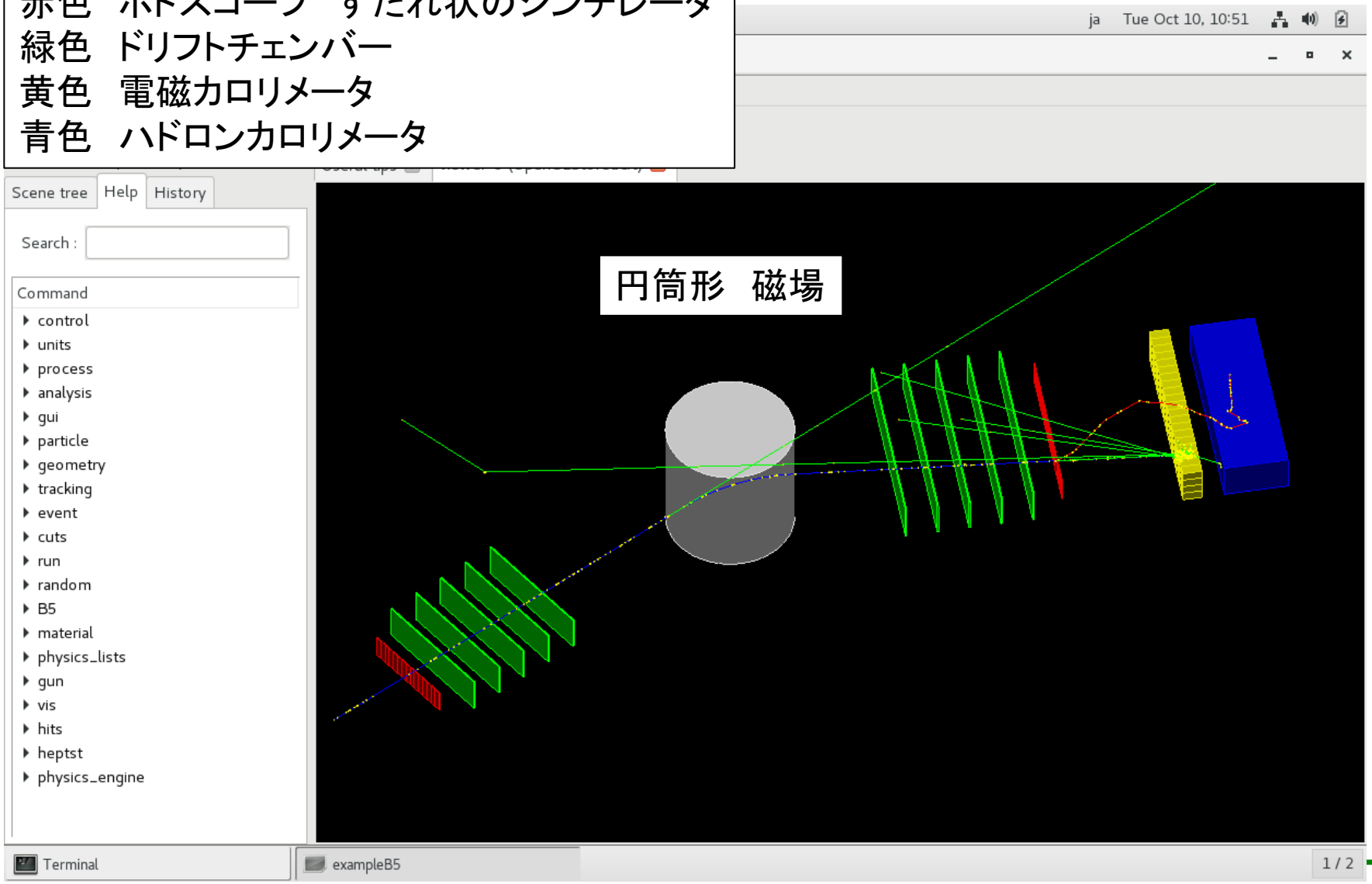
- G4LogicalVolumeは一個のG4VSensitiveDetectorを持てる
  - ✓ 10.3 から一個の論理物体が複数の有感検出器を持てるようになった
- 論理物体の割付は構造物幾何形状クラスのConstruct()またはCreateSDandField() 関数の中でおこなう

1. UserSensitiveDetectorのインスタンスを作り、
   - インスタンスに検出器の階層構造を取り入れて名前を与えられる
     - myEMcal = new MyEMcal("/myDet/myCal/myEMcal");
     - 複数の有感検出器からのデータは名前で識別できる
2. それを論理物体のSetSensitiveDetector()関数の引数にすると、その論理物体が有感となる

3. それを有感検出器マネージャG4SDManagerのAddNewDetector()関数の引数とすると、Initialize(), EndOfEvent()関数を有効にできる
   1. 注意　10.2まではG4VUserDetectorConstruction::SetSensitiveDetector()すれば暗黙でG4SDManagerに登録できていたが、10.3からは明示的に登録せねばならなくなった。 G4SDManager::AddNewDetector(mySenDetector)のように。このため、10.2 のexamplesの中には10.3でコンパイルできても実行時にコアダンプするものがある。

/examples/basic/B5 : single arm spectrometerの例

赤色　ホドスコープ　すだれ状のシンチレータ
緑色　ドリフトチェンバー
黄色　電磁カロリメータ
青色　ハドロンカロリメータ



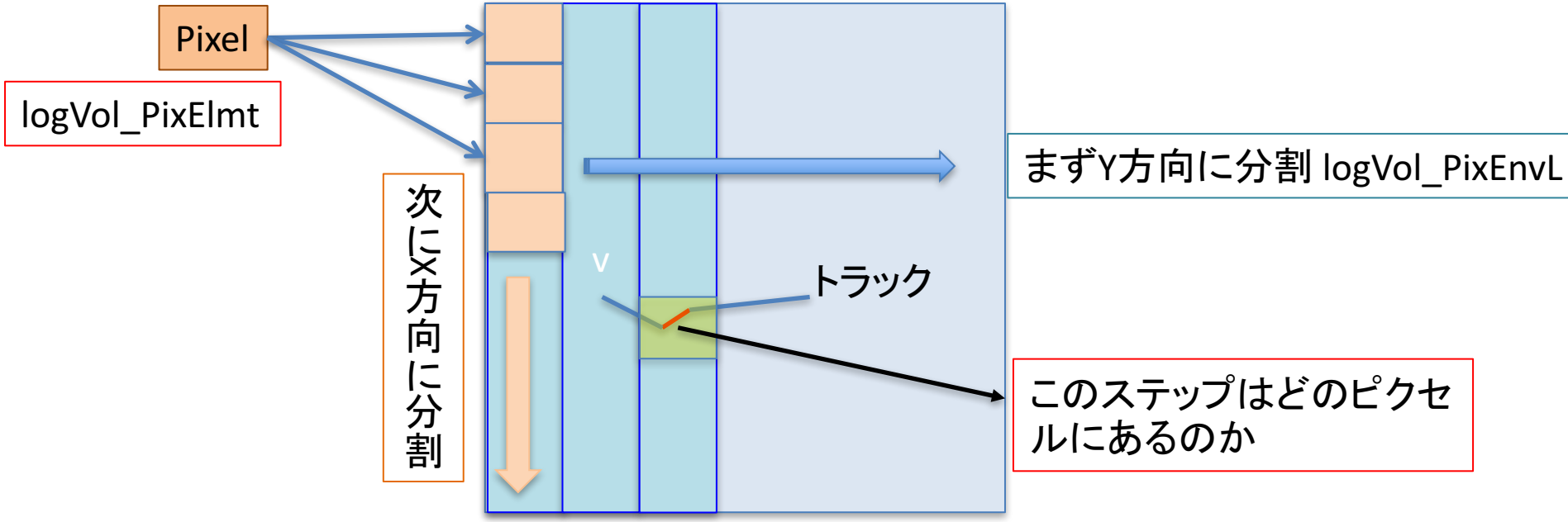
B5のSensitive detectors と対応する Hits クラス

- 4種類の有感検出器クラスとヒットクラスのペア
  - HodoscopeSD, HodoscopeHit
  - DriftChamberSD, DriftChamberHit
  - EmCalorimeterSD, EmCalorimeterHit
  - HadCalorimeterSD, HadCalorimeterHit
- 有感とされる6つの論理物体
  - ホドスコープとドリフト箱はそれぞれ2個あるので
- どのように処理が進むか
  - イベント毎に6回のProcessHits(), Initialize(), EndOfEvent()のコールが発生
    - 4つの有感検出器のProcessHits()では位置やエネルギーデポ情報を取得しヒットオブジェクトとする
  - イベントユーザアクション EventAction では
    - BeginOfEventAction
      - ヒットコレクションの初期化
    - EndOfEventAction
      - 6つのヒットコレクションを取得してヒストやNtupleなどに記入

Handson P05/source-SDの場合

P05_Scoring ピクセル検出器の作り方

- P02/source/src/Geometry.cc の二次元分割レプリカをもとにする
  - ✓ 検出器のエンベロープ logVol_PixEnvG をY方向にエンベロープlogVol_PixEnvL でレプリケートする.



- ✓ 次にlogVol_PixEnvL をX方向にピクセル要素logVol_PixElmtでレプリケートした
- ✓ このピクセル要素を有感とし、ヒットしたらそのピクセル要素番号 (i,j) や世界座標を知りたい

P05/source-SDは非常に簡単な有感検出器の例

- P02のピクセル検出器ジオメトリの個々のピクセルを有感検出器として、

  目的１：粒子のヒット位置（ピクセル番号と空間座標値）を求める。

  目的２：シリコンを通過する粒子のエネルギー損失を求め、薄いシリコンでのdE/dXについて考察する。

  - Bethe-Bloch dE/Dxは平均的なエネルギー損失を与える
  - 薄い物質でのエネルギー損失はランダウ分布が予測する
    - ただし、デルタ線の発生は考慮されていない
  - Geant4では　Restricted Bethe-Bloch + delta raysを使い、二重勘定を防ぐためにGeant4 標準EM には G4Fkuctuationを組み込む
  - Geant4 のレンジカットレンジに相当するエネルギーまでは二次粒子の発生をするが、それ以下だと連続的なエネルギー損失を止まるまで行う。setCut 10 kmとすると、実質上、二次粒子の発生は抑止され、Bethe-Bloch　dE/dxだけで走る

ピクセル要素を有感検出器とする

名前をつけて論理物体を有感検出器とする

P05_Scoring/source-SD/src/Geometry.cc は

ジオメトリ実装クラスであるP02/src/Geometry.cc に次の4行を追加をするだけでよい。

G4VSensitiveDetectorの実装クラスSensitiveVolume

```
Geometry::Construct(){            //の中で追加

 SensitiveVolume* aSV = new SensitiveVolume("/Pixel");     //Pixelという名前のインスタンス

 logVol_PixElmt->SetSensitiveDetector(aSV);       // 論理物体logVol_PixElmtにアサインする

G4SDManager* SDman = G4SDManager::GetSDMpointer();  // SDマネージャのインスタンス
 SDman->AddNewDetector(aSV);     //SDマネージャに登録
```

SensitiveVolume.cc　ProcessHits() のスケッチ

- 一次粒子の入射位置を入射のピクセル内の最初のステップから求めることにする。（反応で生成された二次粒子は無視する）
  - トラックIDを取得して一次粒子かどうかを判定する。
  - ステップのワールド座標を得る
  - ステップが属するピクセル番号を得る

```
SensitiveVolume::SensitiveVolume(G4String name)                    //自作の SensitiveDetector
        : G4VSensitiveDetector(name)
G4bool SensitiveVolume::ProcessHits(G4Step* aStep, G4TouchableHistory*)
{
G4double edep = aStep->GetTotalEnergyDeposit();                    //閾値以下の二次粒子も含む
sum_eDep = sum_eDep + edep;                                        // イベント単位の積算

G4int nTrack = aStep->GetTrack()->GetTrackID();                    //トラックから
If(nTrack!=1) return;                                              //トラックID==1 一次粒子だけを選ぶ
    G4String particleName = aStep->GetTrack()->GetDefinition()->GetParticleName();   粒子名
    G4StepPoint* preStepPoint=aStep->GetPreStepPoint();            //入射ポイント
    G4ThreeVector position_World = preStepPoint->GetPosition();    //そのワールド座標
```

ProcessHit() のスケッチ続き

- レプリカの入れ子になっているピクセル番号の対を得る

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

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

木構造の階層を指定するには引数に整数を与える

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

入射位置の世界座標も得て、これらをヒストグラムに記入する

P05_Scoring ピクセルヒット情報の扱い

- ピクセル検出器の場合は簡単な構造
  - 有感検出器は一種類だけ
  - 論理物体も一つだけ
- ヒットデータも簡単
  - 一次粒子の入射点のコピー番号の対
  - 一次入射点の世界座標
  - シリコン中でのエネルギー損失
- データの処理方法
  - 位置情報はProcessHits() のなかで得られるのでその場でヒストへ記入する
  - エネルギー損失はProcessHits()の中で積算し、EndOfEvent()でヒストに記入する
- 今回はEventActionは使わないが、複雑な測定器では
  - EndOfEventActionで全ての有感検出器からのヒット情報を集めて処理するのが定石である

ヒットオブジェクトとヒットコレクション

Hits データの格納（ヒットとヒットコレクション）

- Hitとは、有感領域中でのトラックの物理相互作用のスナップショットで、G4Stepオブジェクトに属する種々の物理量を集める
  - ✓ ステップの位置と時刻、トラックの運動量、エネルギー付与、ジオメトリ情報など
- ヒット情報が持つべき機能を定めている基底クラスG4VHitを継承して自分のヒットオブジェクト”userHit”を作る
- コンテナ**HitsCollections** とは、各種多量のヒットオブジェクト(G4VHitの具象オブジェクト)を1イベントに渉って寄せ集めるための容器　G4VHitsCollection
  - ✓ G4Eventは、ヒットコレクションのコンテナであるG4THCofThisEvent (Template Hit Class of this Event)クラスへのポインタを持っている。これから、事象の終わりにその全ヒット情報を取得する
  - ✓ **HitsMap** : コンテナG4VHitsCollectionの代わりにSTLの連想リスト形式のコンテナG4THitsMapも使える。後述のPrimitive Scorerはこれを使う
- G4Runは、ユーザ定義のヒットをベクトル形式で格納するG4VHitsCollection へのポインタを持っている

ヒットに関するユーザクラスの実装方針

- フル装備のスコアリングをするには
  1. G4VHitを実装してUserHitクラスを作る
  2. G4THitsCollectionを実装してUserHitsCollectionを作り、それにUserHitを持たせる
  3. G4EventはUserHitsCollectionから最終的に見たい情報を得るよう処理する
     - G4HCofThisEvent (G4_Hits_Collection_of_This_Eventの意)
  4. G4SensitiveDetector を実装してUserHitsCollectionを持つUserSenstiveDetectorを作る
  5. UserSensitiveDetectorをG4LogicalVolumeに持たせる
     - 有感検出器として選んだG4LogicalVolumeに名前を付ける
- 場合によってはUserHitクラスを作らず、連想配列G4THitsMapを実装してUserHitsMapを作り、UserSensitiveDetectorに持たせるか
  - Geant4が提供する既製のスコアラーであるG4VPrimitiveScorerでは(copyNo, eDep)のような連想配列を使っている。

P05_Scoring Hands-on 4つの演習題

- 本講習会では次の4つの方法のうち方法2だけを演習する。方法1、3、4のハンズオンは時間があれば自習する。

1. 方法1： ユーザフックだけでやりくりする
   - P05_Scoring/source/ 自習演習 BGOカロリメータ
2. 方法2: 有感検出器クラスを実装し、ヒット収集
   - P05_Scoring/source-SD/ 演習 ピクセル検出器
     - ただしこの例題ではヒットコンテナは使わない
3. 方法3: 原始スコアラ(Primitive scorer) 方法1と同じ問題
   - G4MultiFunctionalDetector （多機能スコアラ)を登録出来る原始スコアラと原始フィルタを使う
   - P05_Scoring/source-PS/ 演習例題（自習用） BGO カロリメータ
4. 方法4： コマンドベーススコアラ（付録のページ参照）
   - ユースケースが対応しているなら、良く使われる物理量のスコアを取得できる
   - P05_Scoring/source-CS/ 水ファントムでの吸収線量分布

予備の講義録とhandson解説

原始スコアラーを使う例　P05_Scoring/source-PS/ と、
コマンドラインスコアラーを使う例 P05_Scoring/source-CS/

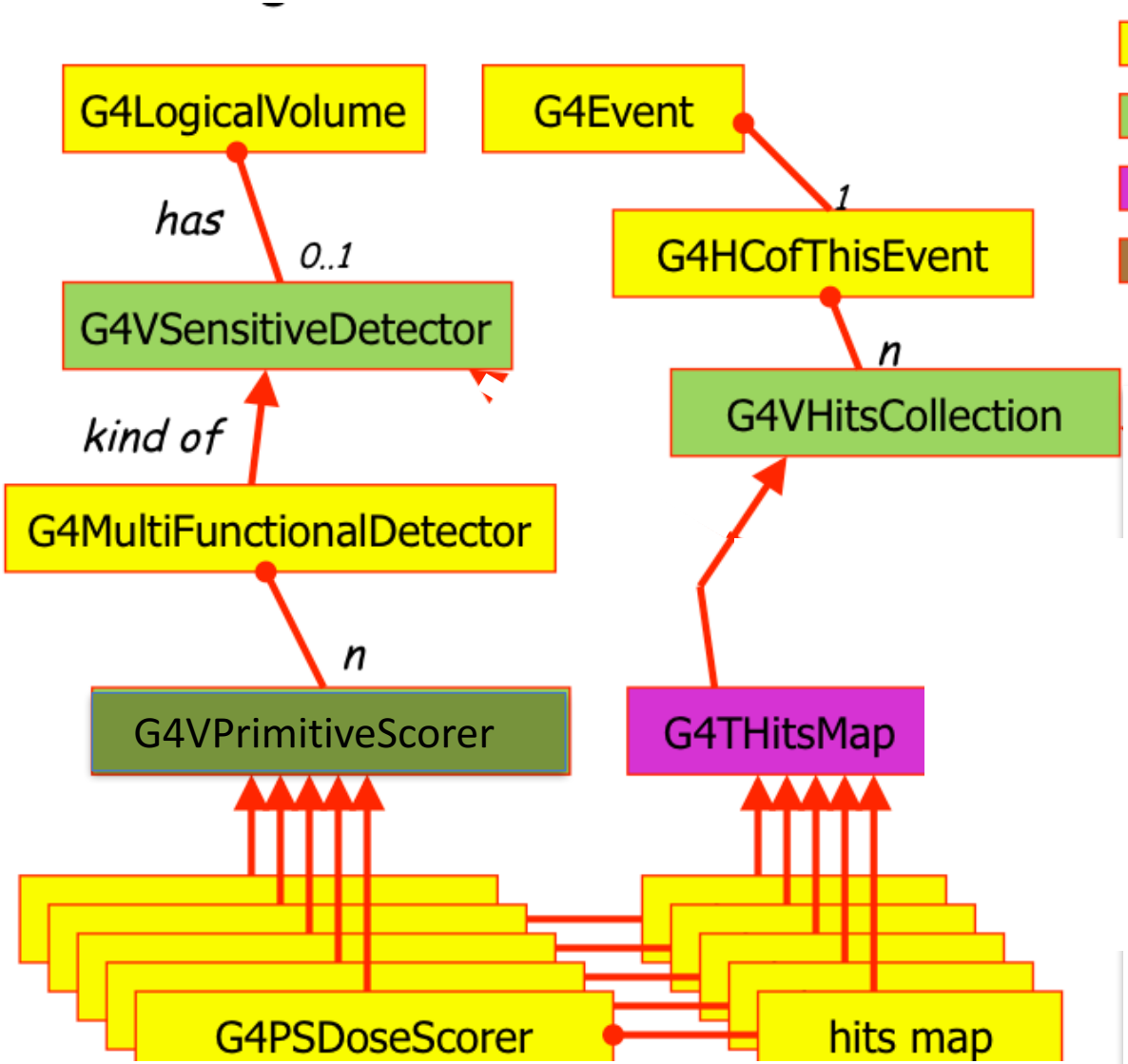
多機能検出器(MultiFunctionalDetector)と原始スコアラ(primitive scorer)

原始スコアラ(Primitive scorer)

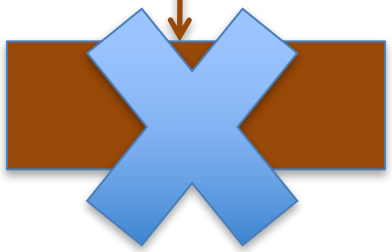
- G4VSensitiveDetectorを継承した多機能検出器G4MultiFunctionalDetectorを使えば、制限付きだが自前の有感検出器クラスを作らなくてよい場合がある
- Geant4が既成の具象スコアラを用意している
  - ✓ スコア量はエネルギー付与や吸収線量のようなよく使われているもので
  - ✓ 個々のステップでのスコアリングは必要ではなく、イベント単位またはラン毎でよい
  - ✓ 多機能検出器に(一つまたは複数の)原始スコアラを割り付ける。ただし、G4VPrimitiveScorerのインスタンスは一つのG4MultiFunctionalDetectorインスタンスだけにしか割り当てできない。
- ヒットデータは有感物体のcopy番号やレプリカ番号(キー)とそこでのスコア量の対を要素とする連想リスト(STLのmap オブジェクト)に格納される。
  - キーから値を取り出すイテレータを使えば、容易にデータ集合を操作できる
- 抽象基底クラスG4VPrimitiveScorerを具体化すれば自分のスコアラが作れるので、具象原始スコアラーのコードを読んで勉強すれば、自前のスコアラーを作るのための勉強になる。

P05_Scoring/source-PS 演習に自習用のコードがある。
カロリメータのエネルギーデポジットを求める。

原始スコアラとヒットマップ　：　クラス図



ユーザは有感検出器と
ヒットコレクションの
具象クラスを
作らなくて良い



用意されている具象原始スコアラ 一覧から抜粋

- 具象原始スコアラ(Guide 4.4.6 参照)
  - ✓ トラック長 : Weighted()メソッドでステップ毎に重みを付けられる
    - G4PSTrackLength
    - G4PSPassageTrackLength　通過したトラックだけ
  - ✓ エネルギー付与　:　粒子の重みはステップ毎に付ける
    - G4PSEnergyDeposit　　➔ P05_Scoring/source-PS自習例題で使用
    - G4PSDoseDeposit
    - G4PSChargeDeposit
  - ✓ Current:　G4BoxのZ平面またはG4Sphereの内側の球面に到達したトラック数
    - G4PSFlatSurfaceCurrent
    - G4PSPassageCurrent
    - **G4PSSphereSurfaceCurrent**
    - **G4PSCylinderSurfaceCurrent**
  - ✓ Flux:　　W / cos ϕ /A 重み、法線への射影、面積
    - G4PSFlatSurfaceFlux
    - **G4PSSphereSurfaceFlux**
    - **G4PSCylinderSurfaceFlux**

具象フィルタ　抜粋

- 原始スコアリングに条件をつけられる
- G4VSDFilterを継承した具象フィルタクラスも提供されている。
  - ✓ G4SDChargedFilter　:荷電粒子を受理
  - ✓ G4SDNeutralFilter　:非荷電粒子を受理
  - ✓ G4SDParticleFilter　:　特定の粒子を受理　Add(“particle_name”)
  - ✓ G4SDKineticEnergyFilter
  - ✓ G4SDParticleWithEnergyFilter
- フィルタの使い方
  1. 原始スコアラに固有名を与えてインスタンスを作る
  2. フィルターに固有名を与えてインスタンスを作る
  3. フィルター　-> Add(“粒子名など”)
  4. 原始スコアラ　-> SetFilter(フィルターのポインタ)
  5. 原始スコアラをSDマネージャに　RegisterPrimitive(原始スコアラポインタ)

ユーザコード

- 原始スコアラを使うためにユーザは以下のコードを作成する
  - 構造物　DetectorConstruction::construct()中で
    1. 名前を付けてG4MultiFunctionalDetector のインスタンスmyDetを作る
    2. そのインスタンスをG4SDManager->AddNewDetector(myDet)で登録する
    3. myDetを論理物体に割り付けて有感検出器とする
    4. 名前を付けてG4VPrimitiveScorersのインスタンスprimを作る
    5. primitiveを多機能検出器に割付する　myDet->RegisterPrimitive(prim)
       ✧ヒットマップの名前は上の２つを “/ “で繋いだものになる
- ユーザアクションクラス
  - RunActionクラスの始めと終わりのメソッド
  - EventActionクラスの始めと終わりのメソッド(G4Runクラスでも代用可)
  - TrackingActionクラスは使わなくてよい
  - SteppingActionクラスは使わなくてよい
- Run クラスでヒットの記録

ヒットコレクションの処理　：　G4HitsMap

- Hitクラスオブジェクトの作成
  - ✓ 一つの原始スコアラはイベント毎に一つのG4THitsMap<G4double>を作る。マップオブジェクトの名前は原始スコアラの名前と同じ
  - ✓ G4THitsMap<G4double>はキー（どの物体）とG4doubleの値（どれだけ）を対応付ける
    - 既定ではG4MultiFunctionDetectorを割り当てた論理物体のコピー番号をキーとして用いる　P05_Scoring/source-PS
      - キーが入れ子になっているコピー番号を考慮せねばならない構造物階層（ある物体の何番目の娘の物体のように）では、今のステップに対応するキーを得るためにGetIndex(G4Step*)を自前で実装する必要が有る
- G4THitsMapへのアクセス
  - ✓ ヒット収集の命名規則　二つの名前をスラッシュで繋いだもの
    - <MultiFunctionalDetector name>=""></<PrimitiveScorer name>="">
  - ✓ G4HitsMapオブジェクトへのアクセスのためにはG4HCofThisEvent()の引数にこのヒット収集IDを渡すとよい
  - ✓ G4THitsMapは []配列添え字演算子を持つ。これはキーの値を引数に取り、その値へのポインタを返す。戻り値が0は、キーが存在しないことを意味する。
  - ✓ G4THitsMapは+= 演算子を持つので、事象ごとのデータを積算することができる

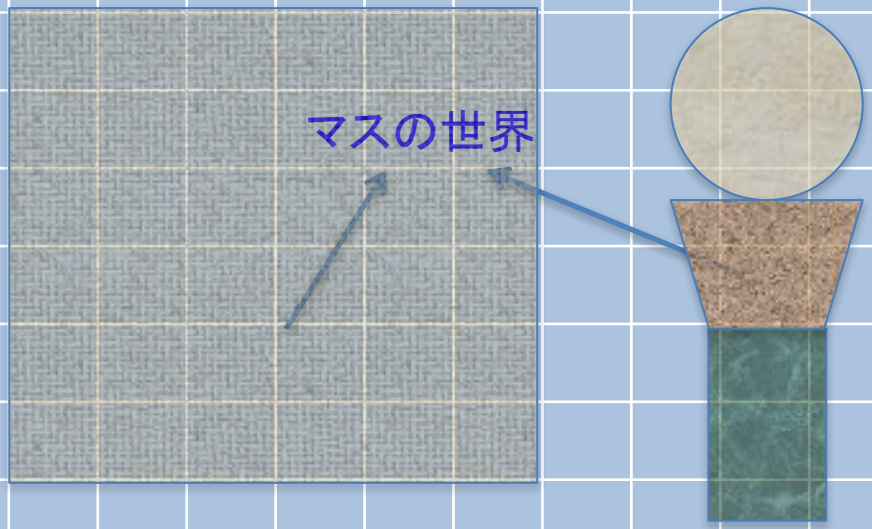
(予備)方法4
コマンドベーススコアリング

最も手軽にスコアリングが出来る方法.
参照マニュアル　4.8. Command-based scoring

コマンドベーススコアラの予備知識

- コマンドベーススコアラの特長
  - パラレルワールドにスコアリングのためのメッシュを切って空間を分割する.
  - コマンドだけですべてのスコアリング処理ができるように作られている
- パラレルワールド(parallel world)とは
  - マスワールドはGeant4の構造物幾何形状で定義されるもので、構造物を構成する立体の形状、物質、寸法、それらの空間配置などの情報を持つ
  - パラレル(parallel)ワールドはマスワールドとは独立に定義出来る(手引き4.7章)
    - マスワールドと同様にパラレルワールドで粒子輸送（navigation)が行われる
    - マスワールドと同様パラレルワールドの境界でもステップは制約される
    - パラレルワールドでもスコアする有感物体を設定しスコアリング出来る
    - 複数のパラレルワールドが設定出来、それらは重なっても良い
    - 例えば、シャワーのパラメータ化では、マスワールドで有感物体を定義するのは隔靴掻痒の感があるが、パラレルワールドではシャワーエンベロープに合わせてやるとよい

マスワールドとパラレルワールド



線量などマスワールドの情報を利用するスコアリングでは
パラレルワールドのメッシュとマスワールドの物体の境界が
一致しない場合メッシュでは正しいスコアが得られない

機能

- コマンドベーススコアラ (command-based scorers)
  - パラレルワールドの応用の一つ
  - スコアできる物理量と適応出来るフィルタは、一定のユースケースを想定して実装されている。汎用ではないが、利用出来れば、C++コード無しでスコアできる.
    - メッシュの設定 （mass geometryとは無関係に自由にメッシュを切れる）
    - スコアラの割り付け （今のところ20ヶ）
    - フィルタの適用 （今のところ5ヶ）
    - 結果の出力（CSVフォーマット）
- 線量や粒子フルーエンスのようなよく使われる物理量を計数したり、計数用メッシュを設定するなど種々のコマンドが用意されている
  - この機能は開発中のものを含んでいる。
  - 開発者はユーザからのコメントや意見を待っている
  - 先ず、コマンドで使えるスコアラを調査し、自分の用途に使えるかどうかを調査するのが吉

最小限のC++コード: main()

- この機能を使うには、main()の中で、G4RunManagerのインスタンスを作った直後にG4ScoringManagerのインスタンスへのポインタを入手するだけ

```
#include “G4ScoringManager.hh”
int main()
{
  G4RunManager* runManager = new G4RunManager;
  G4ScoringManager* scManager =
                              G4ScoringManager::GetScoringManager();
// Un-comment this line for user defined score writer
//    scManager->SetScoreWriter(new RE03UserScoreWriter());
  …
以下通常の必須初期化
```

C++で書くのは
これだけ

- コマンドはすべて/score/の下にある
- /examples/extended/runAndEvent/RE03 にマクロファイルの実例

陽子線ビーム+ 水 + 吸収線量　マクロ

- Proton_200MeV.mac
  - 水ファントムは一辺の長さが1000mmで世界の中心に合わせておいてある
  - 陽子の入射面から500mmの深さまで5x5x1mmのセルを切る
  - 吸収線量とエネルギー付与を求める

```
# Example of proton pencil beam into the water phantom
# define scoring mesh
/score/create/boxMesh boxMesh_1
/score/mesh/boxSize 5. 5. 250. mm
/score/mesh/translate/xyz 0 0 -250. mm
/score/mesh/nBin 1 1 500
#
/score/quantity/doseDeposit dDep
/score/quantity/energyDeposit eDep
#
/score/close
/score/list
#
```