解析ツールとのインターフェイス

Geant4 10.3.p03 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

目標

- HEP実験などでは、膨大な量の実験生データ(raw data)からエッセンスだけを絞り出したDST(Data Summary Tape)を作り、それを元にデータ解析をする
  - 例えば、数百万から数千万チャンネルのADCデータのほとんどは意味あるデータを持たないし、圧縮したDSTでも1イベントあたりのデータ量は大きくばらつく。
  - このようなデータの編成の方法はいろいろ
- 本講義では、SteppingActionやEventActionの中で、ヒストグラム(一次元や二次元)、Ntupleなどに直接データを書き出す方法を学ぶ
- 解析ツールROOTは仮想マシンにインストールされているので自習のこと

目次

1. Geant4とanalysis カテゴリー
    1. analysis カテゴリー
    2. 解析マネージャの働き
2. 解析の方針を決める
    1. 解析マネージャと解析ツールへの接続の選択
    2. Sensitive Detector とHit Collection との接続
3. コードの流れ
4. 解析マネージャーの関数群
    1. ヒストグラムとntupleの定義、Fill, 出力など
    2. /analysis/ コマンドとの使い分け
5. 解析ツールでの利用例
    - ROOTと表計算

[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analysis カテゴリー

analysisカテゴリの位置付け： 外部ツールからの独立

- Geant4は外部ソフトを使う立場であって、解析ツール／外部ソフトには依存しないのが基本方針。
  - 解析ツールはユーザの分野によって様々なのでなんらかの抽象インターフェースが必要だった。
- Geant4の10.0版からはanalysis カテゴリーを設け、解析マネージャを置いた
  - 次のファイル形式に対応するファイルを出力する
    - CSV 形式: 表計算ソフト(Office, LibreOffice/OpenOffice), GnuPlot
    - ROOT形式 : ROOT
    - XML形式 : JAS, iAIDA, OpenScientist, rAIDA

注意　10.3からHBOOKはサポートされなくなった

  - ただし、一つのファイルに複数のファイル形式を入れることはできない
- G4toolsパッケージへの統一的なインターフェイスとして提供されているので、解析ツールによって異なる外部コードへの依存関係はない
  - ユーザコードでは、出力形式に関係なく解析マネージャへのポインタを参照するだけで良い。

解析マネージャの働き

- 解析マネージャG4AnalysisManager は以下の操作のための関数群を提供する
  - ヒストグラムの定義と非活性化（RunAction のコンストラクターで使用）
  - Ntuple の定義と非活性化 （ RunAction のコンストラクターで使用）
  - 既定の出力ファイル名指定（RunAction のコンストラクターで使用）
  - 出力ファイルのオープン （BeginOfRunActionで使用）
  - ヒストグラムへの記入（EventAction, SteppingActionなどで使用）
  - ntupleへの追加 （同上）
  - ファイルへの出力 （EndOfRunActionで使用）
  - など
- 解析マネージャの関数をコードする代わりに、会話型のコマンドも利用できる
  - /analysis/setFileName name　　既定値のファイル名を変更する
  - /analysis/h1/SetActivation id true/false　ヒストの活性化／不活性化
  - /analysis/h1/set id nbin min max　　ヒストのパラメータ設定
- どの形式を選ぶかはanalysis.hh定義ファイルのなかで
  - g4root.hh, g4xml.hh または g4csv.hh の一つを　include　するだけ
- マルチスレッドでもスレッドごとの結果を自動的に集めてくれる（今回は扱わない）

現在の制限事項

- ✧ Histogram の種類: 1D, 2D, 3D 倍精度数データのみ
- ✧ Profile の種類: 1D, 2D　倍精度数データのみ
- ✧ Ntuple のカラムデータの種類: int, float, double, G4String　のデータのみ
  - ✧ 新規追加　std::vector<int>, std::vector<float>, std::vector<double>
- ◆ 以上のものをディレクトリに分けて管理できる。
  - ✓ ただし、ヒストグラム用とNtuple用に各々一つに限られる。例えば
    - myHisto/energy1, energy2, position1, …..
    - myNtuple/hit1, hit2, …..
- ◆ このカテゴリーは精力的に開発が進められているので最新のリリースノートを見ておくべし
  - ✓ 10.3 で名前変更：G4Parameter => G4Accumulable など
  - ✓ HBOOKはサポート停止
  - ✓ 軸の上下限を指定せずにヒストグラムを作り、設定変更できるようになった。いままでは、既定値かダミーの値を書かねばならなかった

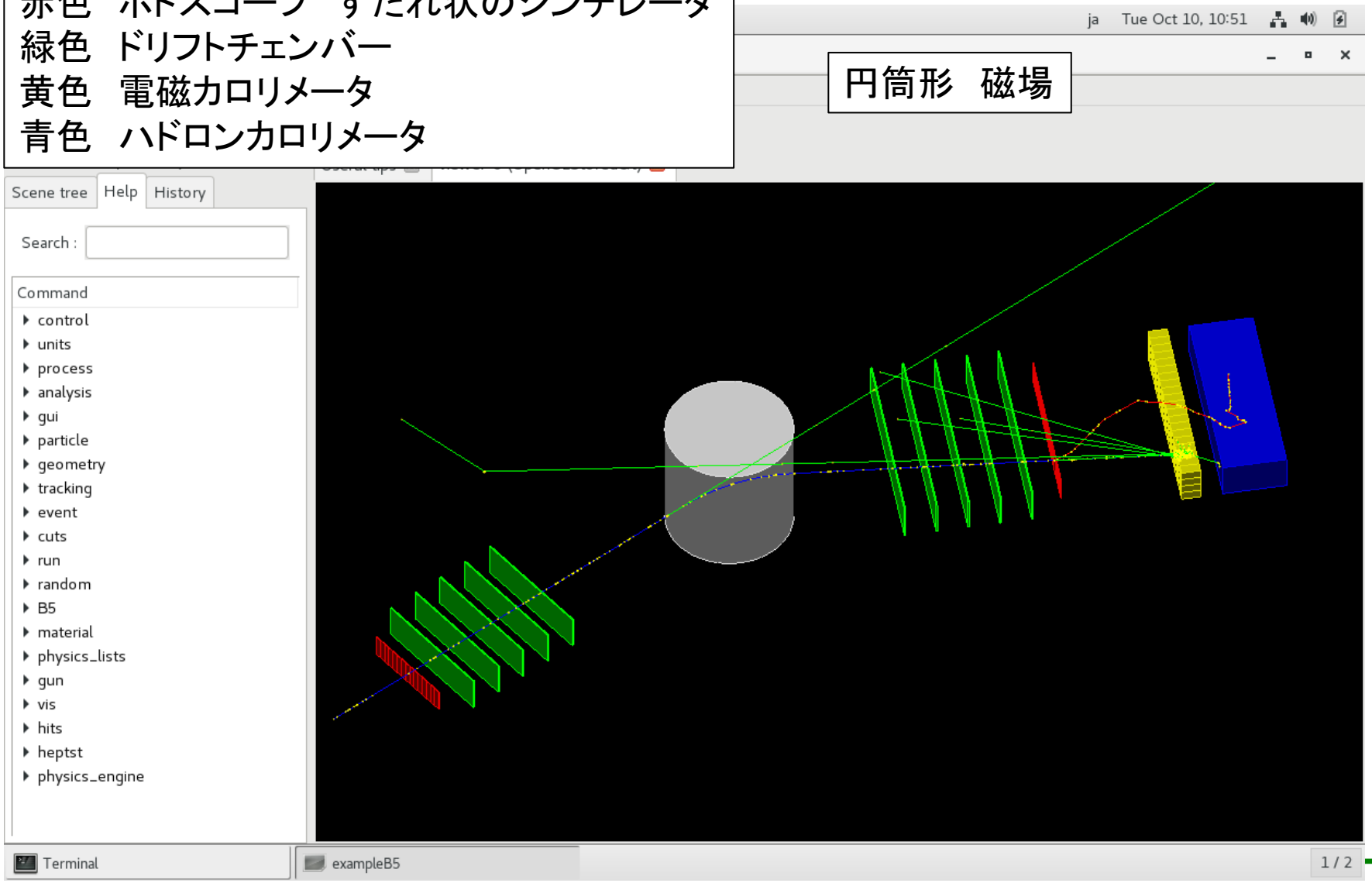
解析の方針を決めよう

選択肢

1. バッチか会話型か
    1. 多量のデータならバッチ
        1. 処理の流れを考えてマクロファイルを準備する。例えば
            1. ラン毎にデータを取るヒストを選択する
            2. ランの条件を変えては別の出力ファイルに書き出す
        2. Ntupleを使うことを考える
        3. 解析ツールはROOTの一択　など
    2. 試験的に使いたいなら会話型、例えば
        1. バッチ処理の前に、ヒストなどのパラメータを変えて最適化したい
        2. 簡単に使える解析ツールから始めて、直ぐに結果を見たい
        3. ROOTを使って見たい　など
2. C++コーディングと /analysis/ コマンドの使い分け
    1. 解析マネージャはC++でそのインスタンスを呼び出す
    2. ヒストなどの定義は予めC++コーディングせねばならない
        - ✓ ヒストを非活性状態で定義しておくと、実行開始後、 /analysis コマンドで様々な設定ができ、それがC++コーディングで記述した既定値を上書きできる
            - ヒストのパラメータ
            - 出力ファイル名　など

Geant4/examples/basic/B5：single arm spectrometerでは

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でのSensitive detectors と Hits

- 4種類の有感検出器とヒットクラス
  - HodoscopeSD/Hit
  - DriftChamberSD/Hit
  - EmCalorimeterSD/Hit
  - HadCalorimeterSD/Hit
- 有感とされる6つの論理物体
- イベント毎に6回のProcessHits(), Initialize(), EndOfEvent()のコールが発生
- イベントユーザアクション　EventAction
  - 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
- 実行用のマクロ
  - 入射粒子の種類などのラン条件を変えては別のファイルに書き出す

コードの流れ:

Handsonの例題では

- 一番簡単な事例として、ヒット情報、ヒットコレクションは使わないことにする。
- P05_Scoring/source-SD は
  - シリコン　Pixel 検出器で、粒子の入射位置及びシリコン中でのエネルギー損失を求める
  - ピクセルを有感検出器とする
    - ProcessHits()で入射位置を求めヒストに記入する。また、ステップでのエネルギー損失を積算する。
    - EndOfEvent() で積算エネルギーデポジットをヒストに記入する。
  - ヒストは非活性化して定義しておき、/analysis/コマンドで設定値を調整して活性化する
- P05_Scoring/source は
  - 単結晶シャワーカロリメータのエネルギーデポジットの分布を求める
    - UserSteppingActionでステップがカロリメータの中にあることを判定し、ステップ毎のエネルギーデポジットを積算する
    - UserEventAction で積算エネルギーデポジットをヒストに記入する

段取り： ユーザアクションがやること

- 段取り
  1. 解析に関わるユーザアクション定義クラス(例えばRunAction.hh)のなかで、使用する解析ツールに対応するヘッダファイルanalysis.hh をincludeし、
  2. Step/Event/Runなどユーザアクションの中でその都度解析マネージャG4AnalysisManagerのインスタンスを作り、解析マネージャのメソッド(定義, Fill, 保存など)を呼び出す
- 関係するユーザアクション
  1. ユーザRunActionのコンストラクタで解析マネージャインスタンスを呼び出してヒストなどを定義
  2. UserRunAction::BeginOfRun()で出力ファイル名前の指定とファイルオープンを行う
  3. ユーザSteppingActionやEventActionのSensitiveDetector::ProcessHits(), EndOfEvent()でヒストなどをFillする
  4. ユーザEventAction::EndOfEventでヒストなどをフィル(fill)する
  5. UserRunAction::EndOfRunでランの終わりにヒストを書き出す
  6. ユーザRunActionのデストラクタで解析マネージャをキルする
- ランを終了したら結果がファイルに出力される

解析マネージャの選択

例えばROOTマネージャを使うには
analysis.hh　で対応する行を生かすだけ！！

```
#include "g4root.hh"
//#include "g4xml.hh”
//#include “g4csv.hh”
```

解析マネージャのメソッドを使うときは、その都度インスタンスを呼び出す

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

複数の解析マネージャを同時に使うには明示的に記述する。

```
#include "G4CsvAnalysisManager.hh"
#include "G4XmlAnalysisManager.hh"
#include “G4RootAnalysisManger”

G4CsvAnalysisManager* csvManager = G4CsvAnalysisManager::Instance();
G4RootAnalysisManager* rootManager = G4RootAnalysisManager::Instance();
```

RunAction　コンストラクタでヒストなどを用意する

- P05_Scoring/source-SD のコードの例。
- ランアクションのコンストラクタで解析マネージャを呼び出して準備する

```
RunAction::RunAction()
 : G4UserRunAction()
{
 // analysis.hh で選んだ xxxAnalysisManager が使われる　xxx={root, xml, csv}

G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
              G4cout << "Using " << analysisManager->GetType() << G4endl;
// 既定の設定
       analysisManager->SetVerboseLevel(1);
       analysisManager->SetFirstHistoId(1);          ヒストのidを１から始める
       analysisManager->SetFileName(fFileName);   ファイル名はコマンドで上書きできる
       analysisManager->SetActivation(true);        ヒストなどの非活性化を有効とする
```

ヒストのIDについては後述

続き：　ヒストグラムとNtupleの定義

```
// Creating histograms
    G4int id = analysisManager->CreateH1(”MotherCopyNo","motherCopyNumber",
100, 1, 100);
    analysisManager->SetH1Activation(id, false);    //このヒストは不活性としておく
```

/analysis/h1/set コマンドでnbin, vnin, vmax などを指定し、活性化できる

```
// Creating ntuple
//

    analysisManager->CreateNtuple("Pixel", "motherCopy Number copyNumber");
    analysisManager->CreateNtupleIColumn("motherCopyNumber");
    analysisManager->CreateNtupleIColumn("copyNumber");
    analysisManager->FinishNtuple();       // 忘れないように
}
```

ランの初めに出力ファイルを準備する

```
Void RunAction::BeginOfRunAction(const G4Run* run)
{
  // Get analysis manager
  G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
  // Open an output file
  analysisManager->OpenFile();
}
```

Tips!　コマンドでファイル名を変更するには
OpenFile()に引数を与えない

コンストラクタRunAction::RunAction中で、SetFileName(fName)で既定値のファイル名を決めておける。ファイルの種別を表す記述子
root, csvなどが自動的に付加される。

アプリの実行開始後、Idle> 状態でファイル名を指定することができ、
/analysis/setFileName　“newFname”
コマンドで指定する名前が既定値を上書きする

ステップやイベントの終わり毎にfill

- ProcessHits()関数ではステップからヒット情報を記入する

```
G4bool SensitiveVolume::ProcessHits(G4Step* aStep, G4TouchableHistory*)
{
analysisManager->FillH1(1,motherCopyNo);
…
analysisManager->FillNtupleIColumn(0, motherCopyNo);
…
}
```

- EndOfEvent() 関数では積算したエネルギデポジットを記入する

```
void SensitiveVolume::EndOfEvent(G4HCofThisEvent*)
{
        analysisManager->FillH1(3, sum_eDep);
}
```

ランの後始末とランアクションのデストラクタ

- EndOfRunAction

```
Void RunAction::EndOfRunAction(const G4Run* aRun)
{
// ファイルへの保存
  //
  G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
    analysisManager->Write();
    analysisManager->CloseFile();     ファイル名は既定値またはコマンド指定
}
```

```
RunAction::~RunAction()
          デストラクタで解析マネージャのシングルトンを殺す
{
  delete G4AnalysisManager::Instance();
}
```

解析マネージャの関数群

定義して、Fillして、出力する
関数の引数が多いので、マニュアル
9.2 Analysis Manager Class 参照

一次元ヒストグラム

- 書式　CreateH1(“name” , “title”, nBins, xmin, xmax
  [, unitName=“none”, fcnName=“none”])
  unitName, fcnNameは省略可能で、使い方はマニュアル9.2.1.5
- ヒストグラム識別子　G4int　id
  - G4AnalysisManager::CreateH1()でヒストグラムを作った時に自動的に作られ、戻り値となる。既定値は0から始まり、新しいヒストを作る毎に1増える。
  - この識別子がfillするときに使われる
    - FillH1(id, value, weight)
  - 注意！CreateH1() の引数で与える文字列のnameは識別子とは無関係だが、識別子に合わせて置くと紛れが少ない。
- ヒストグラムオブジェクト
  - G4AnalysisManager::GetH1(G4int id) でオブジェクトが得られる
  - ヒストグラムオブジェクトではmean(), rms()　などが使える
- ヒストグラムの活性化
  - /analysis/h1/set id nbins xmin xmax unitName を実行するとCreateH1()の引数地を上書きして活性化される

1次元ヒストグラムの操作関数一覧

```
G4int CreateH1(const G4String& name, const G4String& title,
               G4int nbins, G4double xmin, G4double xmax,
               const G4String& unitName = "none",
               const G4String& fcnName = "none",
               const G4String& binSchemeName = "linear");
G4bool SetH1(G4int id,
               G4int nbins, G4double xmin, G4double xmax,
               const G4String& unitName = "none",
               const G4String& fcnName = "none",
               const G4String& binSchemeName = "linear");
G4bool SetH1Title(G4int id, const G4String& title);
G4bool SetH1XAxisTitle(G4int id, const G4String& title);
G4bool SetH1YAxisTitle(G4int id, const G4String& title);
G4bool FillH1(G4int id, G4double value,  G4double weight = 1.0);
G4int GetH1Id(const G4String& name, G4bool warn = true) const;
```

対応するコマンド
/analysis/h1/set
/analysis/h1/setTitle
/analysis/h1/setXaxis
/analysis/h1/setYaxis

二次元ヒストグラム

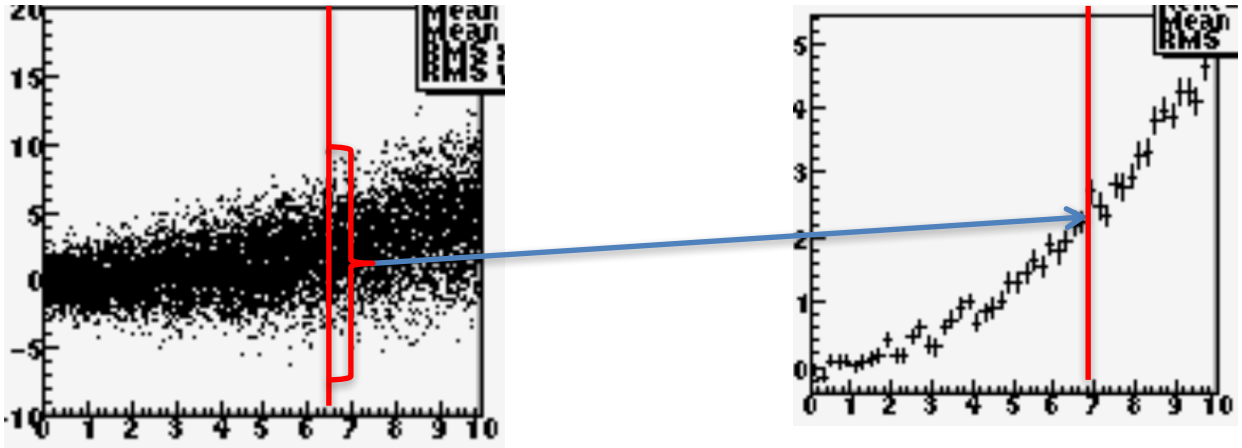
- CreateH2, SetH2

ID 番号は H1とは別につく

```
G4int CreateH2(const G4String& name, const G4String& title,
               G4int nxbins, G4double xmin, G4double xmax,
               G4int nybins, G4double ymin, G4double ymax,
               const G4String& xunitName = "none",
               const G4String& yunitName = "none",
               const G4String& xfcnName = "none",
               const G4String& yfcnName = "none",
               const G4String& xbinScheme = "linear",
               const G4String& ybinScheme = "linear");

G4bool SetH2(G4int id,
               G4int nxbins, G4double xmin, G4double xmax,
               G4int nybins, G4double ymin, G4double ymax,
               const G4String& xunitName = "none",
               const G4String& yunitName = "none",
               const G4String& xfcnName = "none",
               const G4String& yfcnName = "none",
               const G4String& xbinSchemeName = "linear",
               const G4String& ybinSchemeName = "linear");
```

プロファイル　P1

- P1では二次元プロットのYの平均値とYのσをXのビンへ入れる
  （KEK藤井さんの「猿にも使えるROOT」からの実例



```
G4int CreateP1(const G4String& name, const G4String& title,
               G4int nbins, G4double xmin, G4double xmax,
               G4double ymin = 0, G4double ymax = 0,
               const G4String& xunitName = "none",
               const G4String& yunitName = "none",
               const G4String& xfcnName = "none",
               const G4String& yfcnName = "none",
               const G4String& xbinSchemeName = "linear");
```

Ntuple

- 4つのデータ型またはベクトル型のカラムが作れる
  - NtupleのIDと名前
    1. CreateNtuple(“name”, “title”)　　ntuple ID = 0から
  - カラムの定義
    1. CreateNtupleDColumn(“column_1_Name”) column ID = 0から
    2. CreateNtupleIColumn(“column_2_Name”) column ID = 1
    3. CreateNtupleFColumn(“column_3_Name”) column ID = 2
    4. CreateNtupleSColumn(“column_4_Name”) column ID = 3
    5. G4int CreateNtupleDColumn(
       “column_5_Name”, std::vector<double>& vector); column ID = 4
  - このNtuple 定義のおしまい
    2. FinishNtuple()
  - Ntuple column ID は作られた順序に付けられる
  - カラムに入れるデータの型に応じてメソッドが異なる。現在4つのデータ型だけD倍精度、I整数、F単精度、S文字列型をサポートしている
  - 複数のNtupleを作るときはCreateNtuple(G4int id, const G4String& name);
- Fill
  - FillNtupleDColumn(id, value) ここでもデータ型を明示的に含む関数名となっている

ヒストグラムの活性化／不活性化

- 最初にすべてのヒストグラムを活性化せずに作っておいて、アプリの実行時にコマンドで必要なものだけを活性化することができる。こうすれば、必要で無い結果の出力を抑制できる。

```
//非活性化できるようにする。True とすると非活性化ができるのに注意

analysisManager->SetActivation(true);

// ヒストグラム　nbinsなどは後で設定するので非活性として置く
//
G4int id = analysisManager->CreateH1(name, title, nbins, vmin, vmax);
  analysisManager->SetH1Activation(id, false);
……
……
……
```

“name” はヒストグラムのidではないが、ROOTでヒストグラムの識別子として使えるので全てのヒストやntupleを通じてユニークな名前を使うと便利

アプリの起動後、set コマンドを使ってビンニングなどを与えると活性化する

```
/analysis/h1/set 1   100  0   50  cm
```

単位の扱いに注意

/analysis/ コマンド

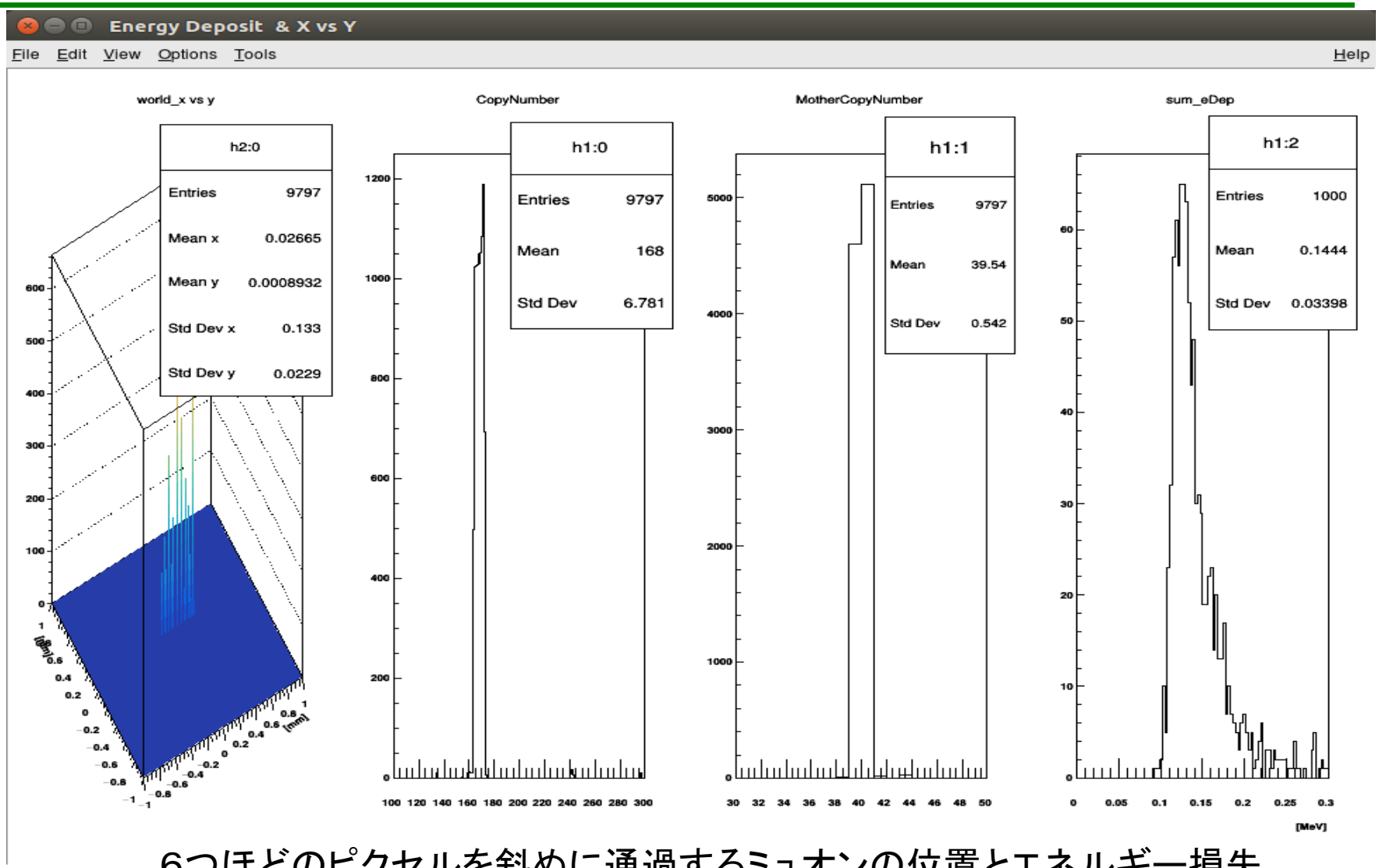
```
ターミナル — exampleB4a — 80×33
exampleB4a
Guidance :
analysis control

 Sub-directories :
   /analysis/h1/   1D histograms control
   /analysis/h2/   2D histograms control
 Commands :
   setFileName * Set name for the histograms & ntuple file
   setHistoDirName * Set name for the histograms directory
   setNtupleDirName * Set name for the ntuple directory
   setActivation * Set activation.
When this option is enabled, only the histograms marked as activated
are returned, filled or saved on file.
No warning is issued when Get or Fill is called on inactive histogram.
   verbose * Set verbose level
Idle> ls /analysis/h1
Command directory path : /analysis/h1/


Guidance :
1D histograms control

 Sub-directories :
 Commands :
   create * Create 1D histogram
   set * Set parameters for the 1D histogram of #Id :
   setAscii * Print 1D histogram of #Id on ascii file.
   setTitle * Set title for the 1D histogram of #Id
   setXaxis * Set x-axis title for the 1D histogram of #Id
   setYaxis * Set y-axis title for the 1D histogram of #Id
   setActivation * Set activation for the 1D histogram of #Id
   setActivationToAll * Set activation to all 1D histograms.
```

参考になるexamples

- Geant4/examples/basic/B4 サンドウィッチカロリメータ
  - ギャップと吸収体でのエネルギー付与
  - RunAction, EventAction の使い方
- Geant4/examples/basic/B5 : 2アームスペクトロメータ
  - ワイヤチェンバ、ホドスコープ　での位置測定
  - カロリメータでのエネルギー測定
  - これらをntupleのカラムにベクトルとして記入
  - ラン毎に出力ファイル名を変える方法
- Geant4/examples/extended/electromagnetic/TestEm
  - まず多数のヒストを定義しておき、
  - UIコマンドで活性化するものを選んでランする方法

解析ツール

P05 Pixel 例題でROOTを使う: PlotPixel2.CC



6つほどのピクセルを斜めに通過するミュオンの位置とエネルギー損失

P05BGO 例題のCSVファイルからExcelでグラフ作成してみたが、、、

1 GeV 電子のeDep分布だが、表計算では色々と足りないものがある。例えば

- 横軸はエネルギーでなく行番号
- 目盛りの数値が気に食わない
- 誤差を重ねて表示できない
- 関数当てはめが単純なものだけ
- などなど

やはりROOTを使うと発表用のグラフが簡単にできるなど、HEP専用の機能が使える。
変更は次のファイルで一行だけ変更する

- Analysis.hh