経過報告

田中 隆己

2012 年 10 月 4 日

1 連絡

- カメラが届きました (ヴァンデで主に使う)。
- HIME DAQ PC が届きました。
- 500GByte, 3TByte, HDD 外付けケース, それぞれ一つずつ学生室で在庫してるので必要なら使ってください。

2 今後の方針

日中はヴァンデに赴いて HIME をやり、それ以外は NEBULA・卒論をやるつもり。

2.1 NEBULA

NEBULA の解析としては、efficiency の妥当性の評価 (ビームの数かぞえ、シミュレーションの妥当性) と cross talk の定量的な評価が大きな課題として残っている。cross talk に関しては最終的に 2n(14Be) のデータを評価しないとパッとしないので、自分も NEBULA 以外の解析を進めて相対エネルギーを組むみたいと考えている。

2.2 HIME

HIME の実験は今年度は無理ということだが、最低限組み上げて宇宙線のデータ取得 (宇宙線のトラックを見たい) まではやるつもり。HIME のコンストラクション・宇宙線のテストについては B4 の卒論になる可能性もあるので、そろそろ誰が何をやるのか fix してほしい。現在は遮光作業をちまちまやりながら DAQ PC のセットアップ等を進めている。

2.3 卒論の内容

卒論の内容は以下を考えいている。

- 中性子検出器のいろは
- neutron detector のシミュレータ作成

- NEBULA の基本性能シミュレーション
- HIME の基本性能シミュレーション
- 2n simulation
- HIMAC
- SM-COM(NEBULA 評価、1n・2n 物理測定評価)
- RCNP での実験について

3 解析ミーティングで言われたこと

- 物質表の共有。小林さんとやりとりしていくつか修正しました。upload してあるファイルはまだ古いままです。
- 見込むポールの面積が大きいので、その影響が知りたい。実験的には x 方向の分布と y 方向の分布を比較すると見えるかも。
- geometry 情報のまとめ。現在少しずつ進めています。
- C による散乱の影響。respons function を出したりするのはどうか。なまった成分は分離できないので efficiency には含めた方がよい。
- SUS 窓などに当たった成分がどう見えるのか。
- 500ch-2000ch で切ったときの resolution と数
- クラスター解析において dt-dr の分布のうち、連続する成分は斜め右上に伸びるのではないか？
- dt の 1D の見え方から定量的にクラスター解析の範囲を決定するべき
- 2n は dt-dr の分布ではどう見える？
- 2mm 開いている、キャスト面、どっちが長い面？ x 軸方向に長く z 軸方向に短い。シールがはってある面がキャスト面らしく、入射面がキャスト面となっているはず。
- p,n における multiplicity
- 反射光の影響は？反射光があると A は y に対して変な相関をもつ。また、y 方向の位置と treso の相関も知りたい。

以上のことを解決する必要がある。まずは小林さんとやりとりをして物質表を更新した (別紙)。

4 Geant4 まわり

4.1 解析マシンに Geant4 を入れました

解析マシンに最新 (4.9.5.p01) の Geant4 を入れました。マイナーな関数の引数とか変わっていたりするので、昔のコードを使うには一部手を入れる必要があるかも。Install のメモ等は web に上げました。4.9.5 はいくつか致命的なバグがあったので 4.9.6beta の src を使うなどして適当に対処しています。

4.2 nebula(というか neutron detector) のシミュレータの開発

neutron detector のシミュレータが大体できた気がします。シミュレータは二段階に分かれていて、

1. beam 入射から素情報取得まで: geant4 の user src。
2. 素情報から実験情報生成・解析: 個々人が解析コードを書く。

となっています。そのままで HIME, NEBULA, LNEUT, どれでも src を変更すること無くシミュレーションできます。VETO の段差やモジュールの隙間も等間隔だけど再現できます。

前も話した通り、上記の様に二つに分かれていると後半何をすればよいのか分からないという懸念があったので、一つにくっつけたコードも作りました。練習用にこちらを使ってもらってもかまいません。

シミュレーションで得られる素情報 (各ステップごとのエネルギーロスなど) は ROOT を使って回収します。また、これを格納する入れ物は別途のライブラリで定義します。入れ物の種類は 3 種類で、beam 情報、scntillator 中の情報、detector の parameter 情報です。これをまとめて ROOT file に入れておくことで一つのファイルで parameter 情報まで入ったシミュレーション結果が得られます。

5 Geant4 での visualize

visualize、及び UI に Qt(キュート) が使えるようになったが、リモートだと無駄に遅くなるので微妙。一方、vrml の viewer で view3dscene という free で且つ linux 可、使い勝手・見た目が良好なものが見つかった。解析 PC に入れてあるが、3D を X で飛ばすのは辛いので、生成される.wrl を local にコピーして見る方法をおすすめする (/usr/bin/remote-view3dscene というスクリプトを置きました)。

6 Geant4 によるシミュレーションコードの開発仕様

こんなことを意識して開発するときっと幸せになれるという妄想。以下を念頭にしてコードを書いたので、この思想を何となく引き継いで他の方もシミュレーションをしていただきたい。

6.1 命名規則

ROOT には厳格な命名規則があり、Geant4 には何となくな命名規則しかない。Geant4 によるシミュレーションを確立するにあたって命名規則を ROOT の規則に統一する。詳しくは `http://root.cern.ch/drupal/content/c-coding-conventions` を参照。といっても、完全に準拠するのは大変なので、以下を大

体守ればよい。

- クラスの名前は T 始まり (url 中 Mixin は気にしなくてよい)。ただし、Geant4 のクラスを継承した場合は特につけない。TContainerView, EventAction
- メンバ変数は f はじまり。fViewLits
- グローバル関数内の static な変数とグローバル変数は g 始まり。gDeviceList
- local 変数は小文字始まり。seed, theCurrentArea
- メンバ関数は大文字始まり。DrawSelf()
- 複数単語になる名前はアンダーバーを使わずに大文字を使う。

一応規則として以下も書いておきますが、わかる人は以下にも従ってください。

- Enum の空間の名前は E 始まり。EFreezeLevel
- Enum や定数 (static cosnt 指定された変数のこと。メンバ変数含む) は k 始まり。kMenuCommand
- static なメンバ変数は fg 始まり。fgTokenClient
- typedef した場合は末尾に_t をつける。Int_t
- struct は末尾に_t をつける。SimpleStructure_t

以上に加えて、ROOT の命名規則にかかれていないが多分一般的な規則。

- 変数の名前は省略せずそれを読んで意味がわかるようにする (eloss ではなく energyLoss とか)。特にオブジェクトの場合は基本的に接頭辞を除いたクラスの名前をそのまま使う (TContainerView なら containerView)。

6.2 ROOT と Geant のどちらのクラスを使うか

ROOT のクラスは基本的にデータ回収用にのみ使う。例えばベクトルの計算は G4ThreeVector を使い、それを格納する変数のみに TVector3 を使う。といっても複雑な beam の初期化等で ROOT のクラスを使いたくなるはずで、そいういうときは混在はできるだけ避けて、ROOT のクラスを使ってまとめて書くようにする。

6.3 思想

漠然とした設計思想。

- ユーザーは NEBULA の実験を行いたい人全て。
- ユーザーがブラックボックスとしてとりあえず使用可能になるための src とテキストを用意する。
- 開発者にはある程度のレベルを要求する。つまり初心者用に冗長に書かない。
- 書き方・ファイル構造・クラス設計等は Geant4 の example や src を参考にする。
- Geant4 で計算するのは step 情報まで。
- data や parameter などは全て単一の ROOT file に保存する。

6.4 具体的なお約束

- 要求する環境: Geant4 と ROOT をライブラリとして呼べる環境。
- user src の基本構成は RunAction, EventAction, DectectorConstruction, PrimaryGeneratorAction, SensitiveDetector, VisManager からなる。
- 自前物理クラスは別のライブラリとしてコンパイルする
- シミュレーションに必要な情報 (Data) の保存には専用のクラスを用意し、このクラスを動的配列 (vector など) に格納した上で tree に保存する。
- Geant4 で使用した Parameter は必要に応じて専用のクラスを用意し、Data を保存した ROOT file と一緒に保存する。
- Data と Parameter を格納する専用のクラスは ROOT 用のライブラリとしてコンパイルする (dictionary とか)。

6.5 意識してほしいテクニカルなこと

- シミュレーションに自由度を出すためには HogeManager を用意し、if 文やコメントアウトによる src の書き換えは避ける。
- warning は無視しない。
- 配列よりも vector を使う (vector を使うとほぼ配列と互換性のある書き方ができます)。
- segmentation fault はデバッガで追えるのでいちいち Debug 用コードを書かない。例えば無駄にポインタがゼロでないかのチェックをするとか。
- 残す必要のある DEBUG コードは#ifdef DEBUG...#endif で囲む。
- #DEFINE ではなく static const を使うこと。
- 行ごとにコメントを書くなどのコメントの乱用はしない。できるだけ変数名やコードの書き方で意図を伝える。
- コードの可読性は重視する。関数化したりコメント残したりしないで memory を直接いじるとかシフト演算するとかは死ね。
- その.cc 用の header file の include は一番最初に書く。
- using namespace を header file に書かない。
- pointer の宣言は double* x; と、アスタリスクを型側にくっつける。