Geant4 基础知识

G4模拟粒子过程:

建立一次模拟，在 G4 中称为一次 Run；Run 建立后，需要对几何结构、物理过程进行初始化；初始化完成后就开始模拟过程了，首先发射一个粒子。在 G4 中，发射一个（或一系列）粒子到所有次级粒子死亡的过程成为一次 Event。而每次发射的初始粒子则有粒子发射器进行控制。而在每一个 event 过程中，粒子与材料反应后会可能生成多个次级粒子，每个粒子都会有一条径 迹，称之为 track；而每一个粒子（初始的或次级的）的径迹又是由很多步组成的，称之为 step。

G4模拟的基本算法：

A Run Start -> 初始化物理模型/几何模型-> An Event Start -> 调用粒子发射器发射粒子

-> A Track Start

-> A Step Start

-> A Step End

-> Next Step Start

-> ……

-> All Step End

-> A Track End

-> Next Track Start

-> ……

-> All Track End

-> An Event End -> Next Event Start

-> ……

-> All Event End(All Primaries Shot) -> A Run End -> Next Run Start

-> ……

1) main()中应该包括的内容

Geant4是一个探测器模拟工具，但它对于某个特定的模拟程序没有固定的main()函数，用户在建立模拟程序的过程中需要提供自己的 main()函数. 一个最基本的 main()函数需要包括以下几个方面:

G4RunManager(模拟整个过程)

G4VUserDetectorConstruction(定义探测器材料，几何形状，灵敏区和读出方案)

G4VUserPhysicsList(定义粒子种类和物理过程，还有截断参数)

G4VUserPrimaryGeneratorAction(定义了源粒子的种类，能量，出射方向等)

一个最简单的 main()函数如下:

```
#include "G4RunManager.hh"

#include "G4UImanager.hh"

#include "ExN01DetectorConstruction.hh"

#include "ExN01PhysicsList.hh"

#include "ExN01PrimaryGeneratorAction.hh"

int main()

{

  // Construct the default run manager

  G4RunManager* runManager = new G4RunManager;


  // set mandatory initialization classes
```

```
runManager->SetUserInitialization(new ExN01DetectorConstruction);
runManager->SetUserInitialization(new ExN01PhysicsList);

// set mandatory user action class
runManager->SetUserAction(new ExN01PrimaryGeneratorAction);

// Initialize G4 kernel
  runManager->Initialize();
// get the pointer to the UI manager and set verbosities
  G4UImanager* UI = G4UImanager::GetUIpointer();
  UI->ApplyCommand("/run/verbose 1");
  UI->ApplyCommand("/event/verbose 1");
  UI->ApplyCommand("/tracking/verbose 1");

  // start a run
  int numberOfEvent = 3;
  runManager->BeamOn(numberOfEvent);
  // job termination
  delete runManager;
  return 0;
}
```


main()首先生成一个 G4RunManager 类，这个类是在主程序中用以初始化模拟信息，用来连接子程序，连接方式是通过 Set 函数来完成。 例如：

```
BDetectorConstruction* detector = new BDetectorConstruction;
```

```
runManager->SetUserInitialization(detector);
```

先构造一个探测器几何，再用 SET 函数初始化。

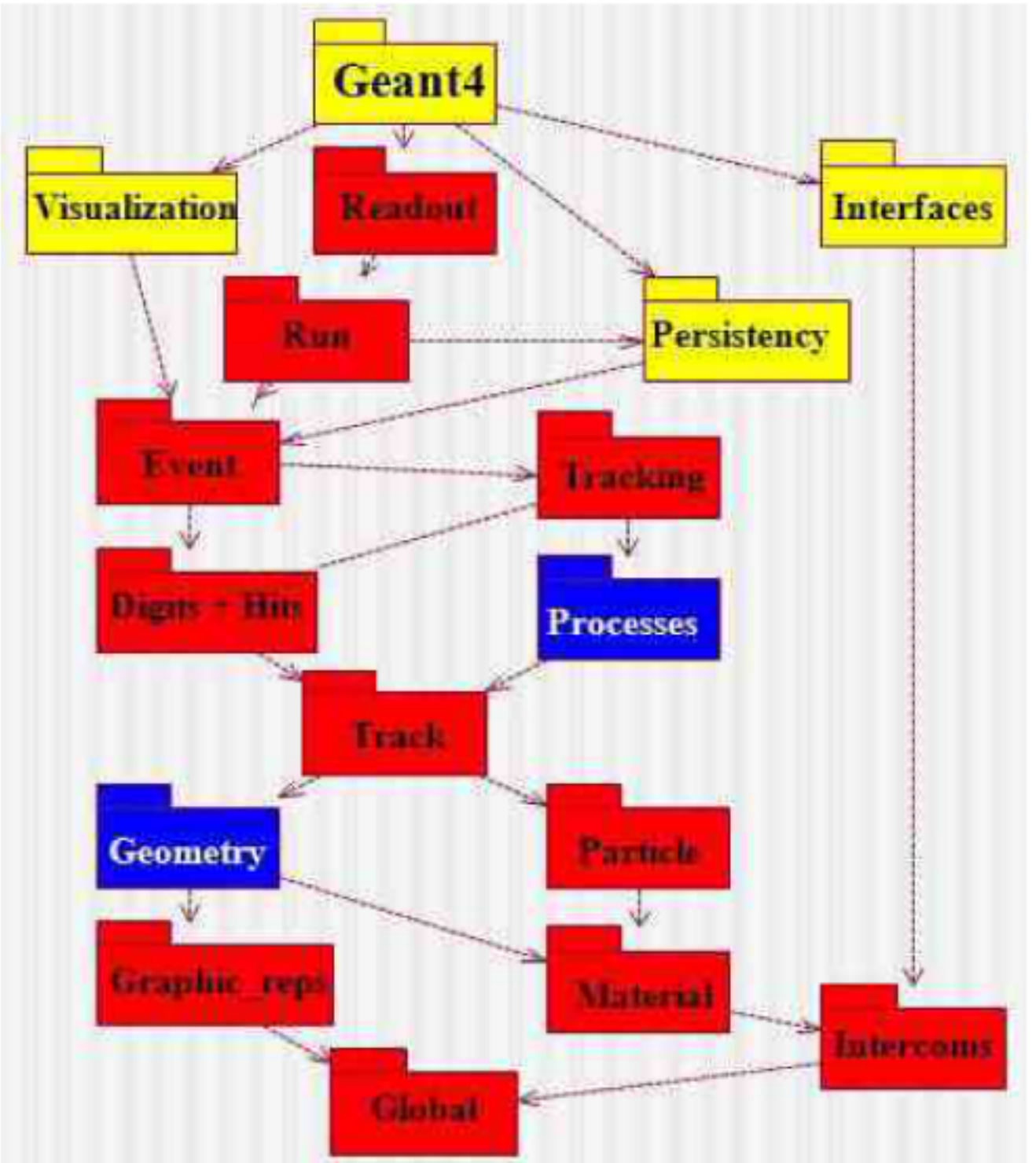
用 new 申请的空间，在程序完成后需要释放空间，否则出错。注意：GEANT4中，在 runManager 上面说明的其他类会自动释放。所以不需要一一释放。

其他类按照重要性分为强制类和可选类。其中几何结构类（DetectorConstruction）、物理设定类（PhysicsList）、源描述类（PrimaryGenerator）都 属于强制类，缺少任一个程序都无法运行。而事件处理类（EventAction）、步数据处理类（SteppingAction）、径迹处理类（TrackingAction）、运行处理类（RunAction）都属于可选类，用户可以通过设定这些类来获取感兴趣的数据，可以按照自己的需要添 加。

包含强制类之后程序就可以运行了，但只能检验几何结构的完备性，所以为了获得数据必须设定可选类。

1. Geant4的类结构

Geant4程序代码是根据这个类结构建立起来的, 类结构如下图所示. 下面就是 Geant4中每个类的简单说明.



1) Run 和 Event

这些类与模拟事件相关，产生次级粒子，为粒子径迹模拟器提供粒子.

2) Tracking 和 Track

根据粒子在探测器材料中的物理过程模拟粒子运行的轨迹，给出粒子特定时间段在空间的位置，或者粒子空间时间的分布.

3) Geometry, Magnetic Field 和 CAD-Interface

这三个类就是用来描述探测器几何结构和探测器中电磁场的分布. 为了使探测器几何结构可以在 CAD 系统中进行修改，Geant4的几何体模型完全与 ISO STEP 标准一致. 将来 Geant4中几何结构的设计将从程序设计中独立出来. 面向对象设计的方法可以让用户在不影响程序其他部分的同时改变几何参数和场.

4) Particle Definition 和 Matter

这两个类用于定义粒子和探测器材料.

5) Physics

它提供了粒子与探测器材料相互作用时所发生物理过程的模型，允许用户为每次反应或每个反应道添加物理模型. 用户根据粒子的种类，探测器材料和能量范围选择不同的物理模型. Geant4有电磁物理过程和强子物理过程模型，同时也提供了散射截面数据库.

6) Hits 和 Digitization

这个两个类用于对用户定义的灵敏区内的响应进行采样和分析.

7) Visualization

它显示了探测器几何形状，粒子轨迹和碰撞过程. 由于采用了面向对象设计技术来设计可视化部分，这允许用户独立开发可视化工具，如：OpenGL 和 OpenInventor(为 X11和 Windows 设计)，DAWN，Postscript(via DAWN)和 VRML.

8) Interfaces

支持可视化用户界面以及和外加程序的交流(OODBMS，reconstruction 等).

①User Verbose output class，定义了一个 verbosity 变量;

②Run manager，定义了一个运行管理器 runManager;

③User Initialization classes (mandatory)，设置两个强制类 ExN02DetectorConstruction 和 ExN02PhysicsList;

④User Action classes，设置用户干涉类 ExN02PrimaryGeneratorAction、ExN02RunAction、 ExN02EventAction 和 ExN02SteppingAction;

⑤Initialize G4 kernel，“runManager->Initialize()”，初始化 GEANT4 内核。构造探测器，生成物理过程，计算截面以及其他;

⑥Get the pointer to the User Interface manager，获取 UI 管理器的指针;

⑦batch mode，定义了批处理模式;

⑧interactive mode : define visualization and UI terminal; 定义了交互模式的可视化和 UI 终端;

2.构建模型:

在 Geant4 中首先要建立一个最大的盒子，称为 World Volume，然后往这个大盒子里面放各种各样的小盒子（部件），然后每个小盒子 （部件） 里面还可以放更小的盒子 （零件） ， 放入的小盒子将自动代替大盒子原有部分。

在 Geant4中，将大盒子称为 Mother Volume （母体），小盒子称为 Daughter Volume （子体）。除了 World Volume 之外，每个 Volume（体）都必须且只能有一个母体，但可以没有子体，也可以有多个子体。Geant4 的这种盒子模型的要求是：“不交不超”。

Geant4 中每个这样的盒子的安放又分为两步。第一步是构建盒子，称为 Logical Volume（逻辑体）；第二步是将盒子摆放到正确位置，变为 Physical Volume（物理体）。

构 建盒子又分为两步，第一步是确定盒子形状，第二步是确定盒子的材料等属性。 形状在 Geant4 中被称为 Solid。在 Geant4 中提供了多种固有的形状，如球形、长方体、锥体等，可以在$G4INSTALL/source/geometry/solids 里面查找。 用户也可以通过 G4VSolid 类构建自己的形状，参见：

$G4INSTALL/source/geometry/management/include /G4VSolid.hh。

此外，对于一些复杂的形状，用户也可以利用基本形状通过交并补等布尔运算方式完成，布尔运算的方式请参考$G4INSTALL/source/geometry/solids/Boolean。

确定了盒子形状后，就是设置盒子的材料、磁场等属性。这些属性的设置通过 G4LogicalVolume 类来完成，设置方法如下：

```
G4LogicalVolume(G4VSolid* pSolid, //形状
                G4Material* pMaterial, //材料
                const G4String& name, //逻辑体名字
                G4FieldManager* pFieldMgr=0, //场管理
                G4VSensitiveDetector* pSDetector=0, //是否 SD 探测器
                G4UserLimits* pULimits=0, //用户限制
 G4bool optimise=true); //是否优化
```

盒子造完了就该摆放盒子了。摆放盒子也有两种方法，一种是直接构建物理体，另一种是指定摆放方法。

直接构建物理体是通过 G4VPhysicalVolume 类，其定义方法如下：

```
 G4VPhysicalVolume(G4RotationMatrix *pRot, //旋转方式
```

```
                const G4ThreeVector &tlate, //摆放坐标

                const G4String &pName, //物理体名字

                G4LogicalVolume *pLogical, //对应的逻辑体

                G4VPhysicalVolume *pMother); //母体
```

如果 pMother=0就表明这个体是一个 World Volume，World Volume 必须且只能有一个。

在实际应用中，我们通常采用指定摆放方法的方式来完成物理体的构建。

指定摆放方法是通过 G4PVPlacement 类完成。G4PVPlacement 类是 G4VPhysicalVolume 的派生类，该类提供了多种方法描述 Logical Volume 的摆放方法。具体可以参考：

$G4INSTALL/source /geometry/volumes/include /G4PVPlacement.hh。

用这种方法可以建立具有相同 Logical Volume 的物理体，同时给可以给每个物理体分配一个编号，以便区分具有相同 Logical Volume 的物理体。这些编号在 UserSteppingAction 等类中处理数据时有时会非常有用处。

需要注意的是，在 Geant4中摆放坐标都是指的相对坐标，是子体中心相对母体中心的坐标。而 World Volume 建立后就等于建立了几何模型的绝对坐标系。

3. 定义材料

1) 基本概念

自然界中，材料(化合物，混合物)一般是由元素组成，而元素由同位素组成. Geant4中有三个类用来定义探测器材料的.

类 G4Element 描述了原子属性:原子量，原子核中质子数，原子质量，壳层能量，还有每个原子的散射截面等.

类 G4Material 描述了材料的宏观属性:密度，状态，温度，压强，还有辐射长度，平均自由程，dE/dx 等.

2) 定义单质材料

如下例，给出材料名称，密度，摩尔质量和原子量，定义了液态氩.

```
//////////////////////////////////////////////////////////////////////////////////////////////////
```

```
    G4double density =  1.390*g/cm3;

    G4double a = 39.95*g/mole;

    G4Material* lAr = new G4Material(name="liquidArgon", z=18., a,
density);

//////////////////////////////////////////////////////////////////////////
///////////////
```

这就可以用来确定所定义的逻辑块的材料:

```
    G4LogicalVolume* myLbox = new G4LogicalVolume(abox, lAr, "Lbox", 0,
0, 0);
```

3) 定义化合物

如例, 定义水(H2O):

```
//////////////////////////////////////////////////////////////////////////
///////////////

a = 1.01*g/mole;

    G4Element* elH = new G4Element(name="Hydrogen", symbol="H", z=1, a);

    a = 16.00*g/mole;

    G4Element* elO = new G4Element("Oxygen", "O", z=8, a);

    density=1.0*g/cm3;

    G4Material* H2O = new G4Material("water", density, ncomponents=2);

    H2O->AddElement(elH, natoms=2);

    H2O->AddElement(elO, natoms=1);

//////////////////////////////////////////////////////////////////////////
///////////////
```

4) 定义混合物

如例，定义空气(Air):

```
//////////////////////////////////////////////////////////////////////////////
///////////////
```

```
a = 14.01*g/mole;
G4Element* elN = new G4Element("Nitrogen", "N", z=7, a);

a = 16.0*g/mole;
G4Element* elO = new G4Element("Oxygen", "O", z=8., a);

density = 1.290*g/cm3;
G4Material* Air = new G4Material("Air", density, ncomponents=2);
Air->AddElement(elN, fractionmass=70*perCent);
Air->AddElement(elO, fractionmass=30*perCent);
```

```
//////////////////////////////////////////////////////////////////////////////
///////////////
```


在 /home/username/geant4.7.0/examples/novice/N03/ExN03DetectorConstruction.cc 文件中介绍了所有定义材料的方法.

4. 指定粒子

1）一般概念

Geant4能够模拟多种粒子.

基本粒子:电子，质子，和 gamma 等.

短寿命的共振粒子:媒质介子和 delta 重子.

核子:氘，alpha 粒子和重离子等.

夸克，D 夸克和胶子.

G4ParticleDefinition 类提供了粒子的定义，而且每种粒子都有自己的类.粒子主要分为6个大类:

轻子(lepton)

介子(meson)

强子(baryon)

波色子(boson)

短寿命粒子(shortlived)

离子(ion)

G4ParticleDefinition 类包含了单个粒子的属性:名称，质量，带电电荷，自旋等. 其中大部分的属性都不可修改，除非重新建立库. Geant4定义了上百种粒子用于不同的物理过程，在一般应用中，用户不需要自己定义粒子. 粒子在连接的过程中注册启用了，用户不须要(也不能)自行执行注册粒子.

2) 指定粒子和物理过程

G4VUserPhysicsList 类让用户可以指定粒子以及它们的物理过程，用户还可以指定截断参数.

一个用户从 G4VUserPhysicsList 中创建自己的类要利用以下几个步骤:

ConstructParticle(): 调用粒子

ConstructPhysics(): 调用粒子的物理过程

SetCuts(): 设置粒子的截断值

用户利用 ConstructParticle()来调用所需要的粒子的函数，如用户需要质子和 geantino，代码如下:

```
//////////////////////////////////////////////////////////////////////////////
///////////////

    void ExN01PhysicsList::ConstructParticle()

      {

        G4Proton::ProtonDefinition();

        G4Geantino::GeantinoDefinition();
```

```
        }
//////////////////////////////////////////////////////////////////////////////
///////////////
```

Geant4中预先定义的粒子有100多个，用户不可能用这种方法来调用每一个粒子的函数，Geant4中有一些类可以让用户调用一类粒子，相对于6个粒子大类，总共有6个类来调用这批粒子.他们是:

G4BosonConstructor

G4LeptonConstructor

G4MesonConstructor

G4BarionConstructor

G4IonConstructor

G4ShortlivedConstructor

使用方法如下例:

```
//////////////////////////////////////////////////////////////////////////////
///////////////
    void ExN04PhysicsList::ConstructLeptons()
    {
        //Construct all leptons

        G4LeptonConstructor pConstructor;
        pConstructor.ConstructorParticle();

    }
//////////////////////////////////////////////////////////////////////////////
///////////////
```


3) 设定截断值

用户必须为每个粒子设定截断值，所谓截断值，就是程序在粒子输运距离为截断值时记录粒子位置，能量的信息. 在一般的模拟程序中，用户可以为所有的粒子设定一个截断值. 如下:

```
//////////////////////////////////////////////////////////////////////////////
///////////////

     void ExN04PhysicsList::SetCuts()

        {

          // the G4VUserPhysicsList::SetCutsWithDefault() method sets
// the default cut value for all particle types

          SetCutsWithDefault();

        }

//////////////////////////////////////////////////////////////////////////////
///////////////
```

系统默认的截断值为1*mm, 当然用户可以为自己调用的粒子和物理过程设定不用的默认值.如下:

```
//////////////////////////////////////////////////////////////////////////////
///////////////

     ExN04PhysicsList::ExN04PhysicsList():  G4VUserPhysicsList()

        {

          // default cut value (1.0*mm)

          defaultCutValue = 1*mm;

        }

//////////////////////////////////////////////////////////////////////////////
///////////////
```

用户可以为 gamma 射线, 电子, 正电子, 或者为不用的几何区域设定不同的截断值. 这种情况下, 由于 Geant4的物理过程(特别是计算能量损失的物理过程)是根据特定的截断值编写的, 它们输出就会改变, 所以用户应该特别小心.

为不同粒子设定不同截断值的例子如下:

```
//////////////////////////////////////////////////////////////////////////////
///////////////
```

```
void ExN03PhysicsList::SetCuts()
    {
      // set cut values for gamma at first and for e- second and next
for e+,
      // because some processes for e+/e- need cut values for gamma
      SetCutValue(defaultCutValue, "gamma");
      SetCutValue(defaultCutValue, "e-");
      SetCutValue(defaultCutValue, "e+");

      // set cut value for proton and anti_proton before all other
hadrons
      // because some processes for hadrons need cut value for
proto/anti_proton
      SetCutValue(cutForProton, "proton");
      SetCutValue(cutForProton, "anti_proton");

      // set cut value for other particles
      SetCutValueForOthers(defaultCutValue);
    }
//////////////////////////////////////////////////////////////////////////
//////////////
```

5. 指定物理过程

1）物理过程

物理过程描述了粒子与材料的相互作用. Geant4中提供了电磁, 强子和其他的反应过程. 有7 个物理过程大类:

电磁物理过程(electromagnetic)

强子物理过程(hadronic)

输运过程(transportation)

衰减(decay)

光学过程(optical)

光轻子-强子过程(photolepton_hadron)

参数化过程(parameterisation)

G4VProcess 是最基本的物理过程的类, 每一个物理过程有三个"DoIt"方式来描述:

AtRestDoIt

AlongStepDoIt

PostStepDoIt

还有三个响应方式:

AtRestGetPhysicalInteractionLength

AlongStepGetPhysicalInteractionLength

PostStepGetPhysicalInteractionLength

下面给出了用于单个物理过程的基本的类:

G4VAtRestProcess (仅适合于"AtRestDoIt"方式的物理过程)

G4VContinuousProcess (仅适合于"AlongStepDoIt"方式的物理过程)

G4VDiscreteProcess (仅适合于"PostStepDoIt"方式的物理过程)

另外4个类(如 G4VContinuousDiscreteProcess)用于复杂的物理过程.

2) 指定物理过程

用户通过 G4VUserPhysicsList 来调用模拟程序中的粒子以及它们的物理过程. 用户通过 ConstructPhysics()方式利用 G4VUserPhysicsList 为每一个所调用的粒子创建自己的物理过程类.

i）添加输运方式

如果没有添加粒子的输运方式，程序就不能描述粒子在时空的输运，以及不能记录粒子输运过程中的轨迹，所以用户就必须为所有的粒子调用 G4Transportation. 如下:

```
////////////////////////////////////////////////////////////////////////////////
////////////////

    void G4VUserPhysicsList::AddTransportation()

        {

          // create G4Transportation

          G4Transportation*     theTransportationProcess     =     new
G4Transportation();


          // loop over all particles in G4ParticleTable and register the

          // transportation process

theParticleIterator->reset();

          while((*theParticleIterator)()){

            G4ParticleDefinition*              particle              =
theParticleIterator->value();

            G4ProcessManager*                  pmanager              =
particle->GetProcessManager();


          // adds transportation to all particles except shortlived
particles

          if (!particle->IsShortLived()){

            pmanager -> AddProcess(theTransportationProcess);

          // set ordering to the first for AlongStepDoIt
```

```
            pmanager                                                    ->
SetProcessOrderingToFirst(theTransportationProcess,
                                                        idxAlongStep);
          }
        }
      }
////////////////////////////////////////////////////////////////////////
///////////////
```

ii) 调用物理过程

以 ConstructProcess()方式为 G4Geantino 类调用输运物理过程如下:

```
////////////////////////////////////////////////////////////////////////
///////////////
 void ExN01PhysicsList::ConstructProcess()
    {
      // Define transportation process
      AddTransportation();
    }
////////////////////////////////////////////////////////////////////////
///////////////
```

为 gamma 射线添加物理过程如下:

```
////////////////////////////////////////////////////////////////////////
///////////////
void MyPhysicsList::ConstructProcess()
      {
        // Define transportation process
        AddTransportation();
```

```
    // electromagnetic process
    ConstructEM();
  }
void MyPhysicsList::ConstructEM()
  {
    // Get the process manager for gamma
    G4ParticleDefinition* particle = G4Gamma::GammaDefinition();
G4ProcessManager* pmanager = particle->GetProcessManager();


        // Construct process for gamma
        G4PhotoElectricEffect * thePhotoElectricEffect = new
G4PhotoElectricEffect();
        G4ComptonScattering * theComptonScattering = new
G4ComptonScattering();
        G4GammaConversion * theGammaConversion = new
G4GammaConversion();
        // Register process to gamma's process manager
          pmanager->AddDiscreteProcess(thePhotoElectricEffect);
          pmanager->AddDiscreteProcess(theComptonScattering);
          pmanager->AddDiscreteProcess(theGammaConversion);
      }
//////////////////////////////////////////////////////////////////////////
///////////////
```


6. 产生初始事件

1）初始事件

G4VUserPrimaryGeneratorAcrion 可以让用户创建自己的初始事件类. 用户提供初始事件的状态, 如: 出射粒子的种类, 出射能量, 方向等. 如下:

```
//////////////////////////////////////////////////////////////////////////////
///////////////
```

```
#include "ExN01PrimaryGeneratorAction.hh"



#include "G4Event.hh"
#include "G4ParticleGun.hh"
#include "G4ParticleTable.hh"
#include "G4ParticleDefinition.hh"
#include "globals.hh"
ExN01PrimaryGeneratorAction::ExN01PrimaryGeneratorAction()
  {
    G4int n_particle = 1;//设定每次事件出射粒子的数量
    particleGun = new G4ParticleGun(n_particle);

    G4ParticleTable*                    particleTable                    =
G4ParticleTable::GetParticleTable();
      G4String particleName;

particleGun->SetParticleDefinition(particleTable->FindParticle(particle
leName="geantino"));//设定出射粒子的种类
      particleGun->SetParticleEnergy(1.0*GeV);//设定出射粒子的动能
particleGun->SetParticlePosition(G4ThreeVector(-2.0*m,  0.0,  0.0));//
设定出射粒子的出射位置
```

```
}


ExN01PrimaryGeneratorAction::~ExN01PrimaryGeneratorAction()
{
  delete particleGun;
}
    void     ExN01PrimaryGeneratorAction::GeneratePrimaries(G4Event*
anEvent)
      {
        G4int i = anEvent->GetEventID() % 3;
        G4ThreeVector v(1.0,0.0,0.0);
        switch(i)
       {
        case 0:
          break;
        case 1:
          v.setY(0.1);
          break;
         case 2:
          v.setZ(0.1);
          break;
       }
particleGun->SetParticleMomentumDirection(v);//设定出射粒子的方向
```

```
        particleGun->GeneratePrimaryVertex(anEvent);

    }

//////////////////////////////////////////////////////////////////////////
///////////////
```

2) G4VPrimaryGenerator

Geant4中有两个G4VPrimaryGenerator的独立的类, 一个是G4ParticleGun, 另一个是G4HEPEvtInterface. G4ParicleGun用来给用户确定出射粒子的动能和位置. 方式如下:

```
    void SetParticleDefinition(G4ParticleDefinition*)

    void SetParticleMomentum(G4ParticleMomentum)

    void SetParticleMomentumDirection(G4ThreeVector)

    void SetParticleEnergy(G4doule)

    void SetParticleTime(G4double)

    void SetParticlePosition(G4ThreeVector)

void SetParticlePolarization(G4ThreeVector)

    void SetNumberOfParticles(G4int)
```


执行模拟程序

1. 命令总目录

/control/ UI 控制命令.

/units/ 单位.

/geometry/ 几何形状控制命令.

/tracking/ 轨迹和步长控制命令.

/event/ 事件控制命令.

/run/　运行控制命令.

/random/　随机数状态控制命令.

/particle/　粒子控制命令.

/process/　物理过程控制命令.

/hits/　灵敏区和采样

/gun/　粒子源控制命令.

/vis/　可视化命令.

2. 一些常用命令

1) /control/execute

说明:执行一个宏文件.

2) /control/loop

说明:多次执行宏文件

3) /units/list

说明: 列出所有存在的量纲.

4) /tracking/verbose

说明: 设置记录粒子轨迹信息的级别.

5) /run/beamOn

说明:开始运行模拟程序.如果没有初始化 G4内核, 它将自动初始化. 默认运行的事件数为1, 第二第三个参数可以让用户在每次事件结束后执行一个宏文件. 如果第二个参数(例:宏文件名)存在, 但第三个参数不存在, 这样所有的事件都是执行这个宏文件.

6) /process/list

说明: 列出所有的物理过程名称.

7) /process/activate(inactivate)

说明：激活(停止)物理过程，激活(停止)粒子运行.

name：物理过程或类型的名称.

particle：粒子名称 [all：所有的粒子]

8) /hits/list

说明：列出灵敏区.

9) /gun/particle

说明：源粒子设置.(默认为 geantino) (出射离子)

10) /gun/energy

说明：设定出射粒子的动能.

11) /gun/number

说明：设置出射粒子的个数.

12) /vis/open

说明:/vis/open [<graphics-system-name>] [<pixels>]

在画图系统建立一个面，该面的名称自动给出.第二个参数给出窗口大小.

13) /vis/scene/create

说明：/vis/scene/create [<scene-name>] 创建一个当前使用的空的作图面.

14) /vis/scene/add/axes

说明：/vis/scene/add/axes [<x0>] [<y0>] [<z0>] [<length>] [<unit>]

默认值：0 0 0 1 m 在(x0, y0, z0)画出给定长度的坐标轴.

15) /vis/scene/add/hits

说明：在作图面上添加碰撞.

16) /vis/scene/add/trajectories

说明: /vis/scene/add/trajectories [drawing-mode]在作图面上添加粒子轨迹.

17) /vis/scene/add/volume

说明: /vis/scene/add/volume [<physical-volume-name>] [<copy-no>] [<depth-of-descending>]

在作图面上添加物理块.

参数1: 几何块名称 (默认 "world").

参数2: 拷贝数 (默认 -1). 如果是负值, 选择第一个物理块

参数3: 下一级的深度 (默认: G4Scene::UNLIMITED (-1)).

18) /vis/scene/endOfEventAction

说明: /vis/scene/endOfEventAction [accumulate|refresh]

参数1: 下一次事件的轨迹图直接画在上一次事件结束时的图上

参数2: 刷新后画出当前事件轨迹图

19) /vis/viewer/create

说明: /vis/viewer/create [<scene-handler>] [<viewer-name>] [<pixels>]

为指定的作图面设置观察面, 默认为当前的作图面.

20) /vis/viewer/flush

说明: /vis/viewer/flush [<viewer-name>]

是 /vis/viewer/refresh + /vis/viewer/update 的复合命令.用来刷新和初始化作 图系统.用/vis/viewer/list 命令显示的名称来指定观察面.

21) /vis/viewer/refresh

说明: /vis/viewer/refresh [<viewer-name>]

刷新观察面, 使用当前观察面, 可以刷新/vis/viewer/list 命令列出的观察面.

22) /vis/viewer/zoom

说明: /vis/viewer/zoom [<multiplier>]把观察对象放大多少倍.

23) /vis/viewer/zoomTo

说明: /vis/viewer/zoomTo [<factor>]把观察对像放大到多少倍.

24) /vis/viewer/set/style

说明:/vis/viewer/set/style w[ireframe]|s[urface]

25) /vis/viewer/set/viewpointThetaPhi

说明: /vis/viewer/set/viewpointThetaPhi [<theta>] [<phi>] [deg|rad]设定观察方向.

26) /vis/viewer/set/viewpointVector

说明: /vis/viewer/set/viewpointVector [<x>] [<y>] [<z>] 设定观察方向.