Geant4 и особенности его применения

Курганов Александр

25 апреля 2019 года

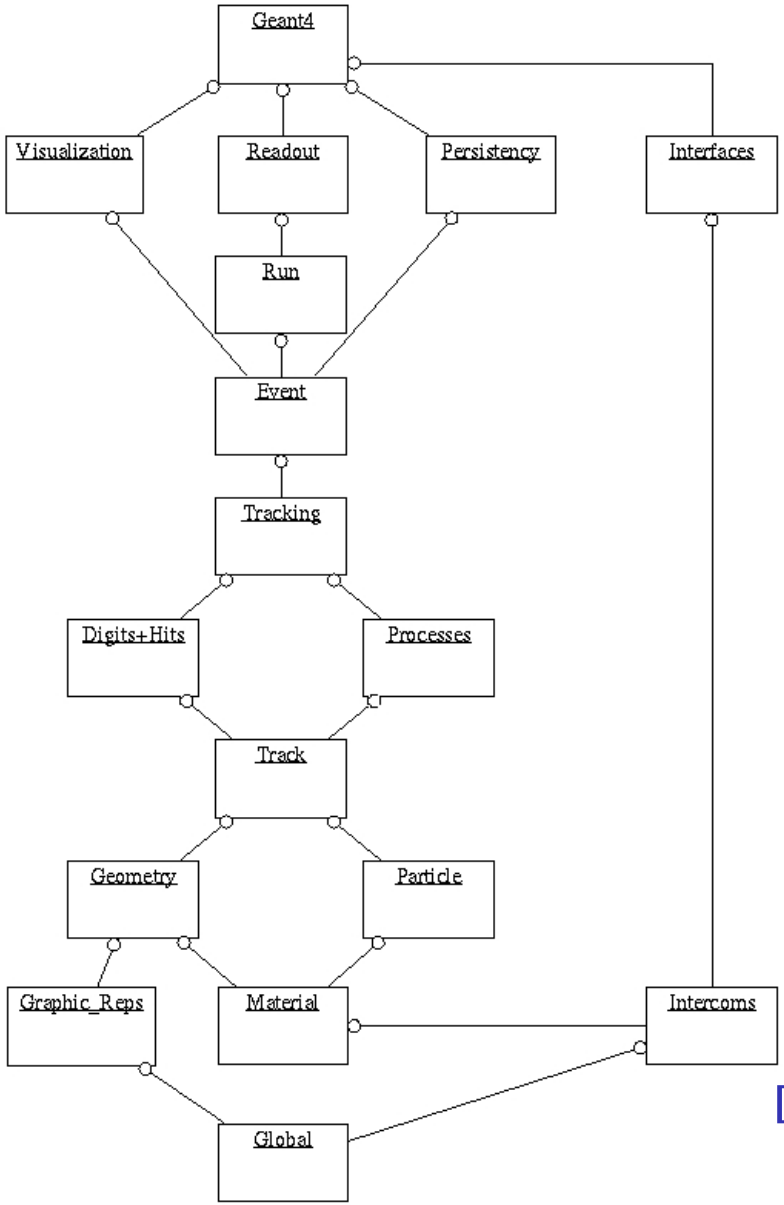
Необходимые мануалы

- User guide for application developers
  Описание классов, принципов работы, алгоритмов

  `http://geant4-userdoc.web.cern.ch/geant4-userdoc/UsersGuides/ForApplicationDeveloper/html/index.html`
- Physics reference manual
  Описание конкретных формул для процессов и ссылки на работы `http://cern.ch/geant4-userdoc/UsersGuides/PhysicsReferenceManual/html/index.html`
- Physics list guide
  Когда какие physics list применять

  `http://cern.ch/geant4-userdoc/UsersGuides/PhysicsListGuide/html/index.html`

Структура и основные понятия



Run Набор событий для заданных параметров и геометрии

Event Событие - один запуск первичных частиц

Tracking Отслеживание частицы, изменение ее параметров

Process Физический процесс, способный изменять параметры частицы

Geometry Геометрия детектора, магнитные поля

Material Определение материалов и их свойств

Digi+Hits Затронем позже

Основные классы-интерфейсы

Run Manager
Инициализация всех необходимых объектов (геометрия, физические процессы и т. д.), запуск Run
Единственный класс, который надо создавать непосредственно

Event Manager
Управляет одним событием: создание первичной частицы, отслеживание до конца

Tracking manager
Управляет одним треком: отслеживание шагов, отслеживание, когда он умирает и т. д.

Stepping manager
Управляет одним шагом симуляции для данного трека: выбор длины шага, применение непрерывных и дискретных физических процессов

Что предоставляется пользователю

Выбор, чем инициализировать
Определение конструкции, набора физических моделей:
DetectorConstruction и PhysicsList
Обязательно!

Действия в определенных случаях
UserAction: RunAction, EventAction, StackingAction, TrackingAction, SteppingAction
Обязательно: G4VUserPrimaryGeneratorAction

User interface
Комманды, установка параметров и т. д. в околоконсольном виде

Наследование - !
В принципе, можно унаследовать практически любой класс и, переопределив виртуальные методы, изменить в корне его принцип действия (например, унаследовав RunManager, можно заставить его перезапускать событие, если в нем что-то не понравилось)

Пример инициализации

```
#include "G4RunManager.hh"
#include "G4UImanager.hh"

int main() {
  //Create the runManager
  G4RunManager* runManager = new G4RunManager;

  //Three mandatory init classes
  runManager->SetUserInitialization(new ExDetCnst);
  runManager->SetUserInitialization(new ExPhysList);
  runManager->SetUserInitialization(new ExUserAction);

  //Initialize
  runManager->Initialize();

  //UI
  G4UImanager* UI = G4UImanager::GetUIpointer();
  UImanager->ApplyCommand("/control/execute␣macros.mac");

  //Ending
  delete runManager;
  return 0;
}
```

Создание DetectorConstruction

Создание DetectorConstruction

1. Создать G4VUserDetectorConstruction, определить метод Construct(), при необходимости - конструктор (например, загрузка из конфига)
2. Внутри Construct():

```
//1. Destroy the previous world
G4GeometryManager::GetInstance()->OpenGeometry();
G4PhysicalVolumeStore::GetInstance()->Clean();
G4LogicalVolumeStore::GetInstance()->Clean();
G4SolidStore::GetInstance()->Clean();
//2. Define materials
G4NistManager* man = G4NistManager::Instance();
G4Material* airMat = man->FindOrBuildMaterial("G4_AIR");
//3. Create the world
solidWorld = new G4Box("World", WORLD_SIZE,WORLD_SIZE,
    ↪ WORLD_SIZE);
logicWorld = new G4LogicalVolume(solidWorld, worldMat, "
    ↪ World");
physiWorld = new G4PVPlacement(0, G4ThreeVector(),
    ↪ logicWorld, "World", 0, false, 0);
//4. Create all the detectors
//...
//5. Exit
return physiWorld;
```

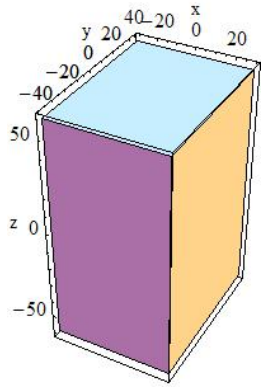
Создание каждого детектора или объема

https://geant4.web.cern.ch/sites/geant4.web.cern.ch/files/geant4/collaboration/workshops/users2002/talks/lectures/geobasics.pdf

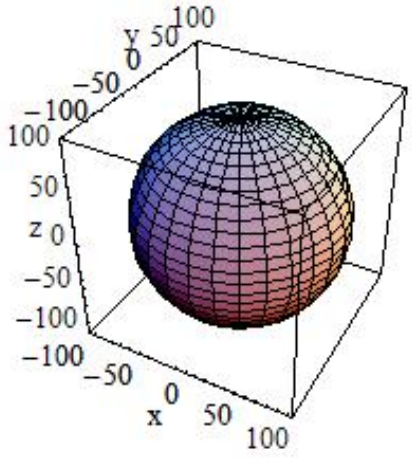
1. **G4VSolid:** Определение размеров и формы объема
2. **G4LogicalVolume:** Определение материала, визуализации, senstive
3. **G4VPhysicalVolume:** определение конкретного расположения, иерархии
4. Если это необходимо: **G4VSensitiveDetector:** определение действия при прохождении частицы (метод ProcessHits)

Объемы не должны пересекаться, но могут быть вложены друг в друга, если верно определен родитель

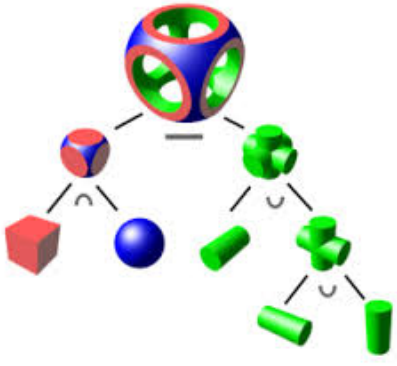
Примеры G4VSolid



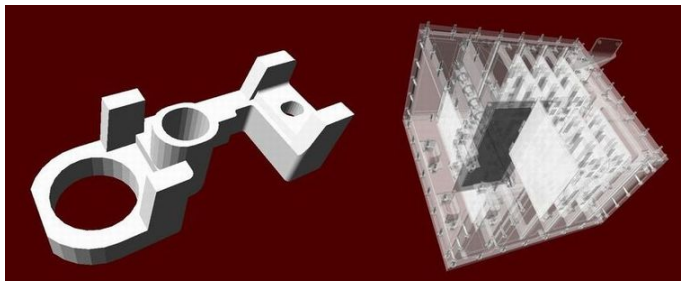

```
G4Box("solidBox", 30*mm, 40*mm
    ↪ , 60*mm)
```




```
G4Orb("solidOrb", 100*mm)
```




```
G4SubtractionSolid("boolSolid"
    ↪ , &box, &cylinder);
```



```
G4Orb("solidOrb", 100*mm)
```

logical Volume и материал

```
G4LogicalVolume( G4VSolid*                pSolid,
                 G4Material*              pMaterial,
                 const G4String&          Name,
                 G4FieldManager*          pFieldMgr=0,
                 G4VSensitiveDetector*    pSDetector=0,
                 G4UserLimits*            pULimits=0,
                 G4bool                   Optimise=true )
```

pSolid - разобрались

pULimits - разберемся чуть позже

Optimize - просто оставить true

pMaterial - Создание материала:

```
//NIST
G4NistManager* man = G4NistManager::Instance();
G4Material* pbMat = man->FindOrBuildMaterial("G4_Pb");
G4Material* pbMat2 = man->BuildMaterialWithNewDensity("
    ↪ DensPb", "G4_Pb", 1.0*(g/cm3));

//Silicon
new G4Material("PbMat" , 82., 207.211*g/mole, 11.3415*g/cm3)
```

Sensitive detector

Класс для создания активных областей: `G4VSensitiveDetector`
Создается в G4VUserDetectorConstruction::ConstructSDandField()

- `ProcessHits()` - вызывается каждый шаг, прошедший в объеме; передается шаг
- `Initialize()` - в начале события, передается набор хитов
- `EndOfEvent()` - в конце события, аналогично

Хиты (Hits) - шаги, затронувшие активные области; хранится вся необходимая информация о частице и ее энерговыделение

```
G4SDManager* SDMan = G4SDManager::GetSDMpointer();
SDMan->AddNewDetector(sensdet);
SDDet.push_back(SDInfo(logicdet,sensdet));

logicdet->SetSensitiveDetector(sensdet);
//Or (inside Construct())
SetSensitiveDetector("LogVolName", sensdet);
```

Physical placement

Задает конкретную трансляцию и поворот объема в пространстве

- Простейший случай: просто расположить объем в пространстве

```
G4PVPlacement(          G4RotationMatrix*   pRot,
              const     G4ThreeVector&      tlate,
                        G4LogicalVolume*    pCurrentLogical,
              const     G4String&           pName,
                        G4LogicalVolume*    pMotherLogical,
                        G4bool              pMany,
                        G4int               pCopyNo,
                        G4bool              pSurfChk=false )
```

- Реплики: `G4PVReplica`
- Параметрическая реплика: `G4PVParameterised`

G4PVReplica

```
G4PVReplica( const G4String&           pName,
             G4LogicalVolume*          pCurrentLogical,
             G4LogicalVolume*          pMotherLogical,
             const EAxis               pAxis,
             const G4int               nReplicas,
             const G4double            width,
             const G4double            offset=0 )
//Example
new G4PVReplica("physiPairs", logicPair, logicCal, kXaxis,
    ↪ 30, 2.3*mm, 0);
```

Ограничения:

- Реплика должна быть *единственным* объемом в родительском
- Реплики должны целиком заполнять родительский объем; никаких пустот

Пример с репликой

```
//Size definition
double dx = 12*cm; dy = dx;
double PbThick = 0.8*mm; double SciThick = 1.5*mm;
int nPairs = 100;
double pairThick = PbThick + SciThick;
double fullThick = nPairs*pairThick;

//main ECal volume
EcalSolid = new G4Box("SolidEcal", dx/2.0, dy/2.0, fullThick/2.0);
logicEcal = new G4LogicalVolume(EcalSolid, defMat, "LogicEcal");
new G4PVPlacement(0, G4ThreeVector(0, 0, 0), logicEcal, "PhysiEcal",
    ↪  logicWorld, false, 0);

PairSolid = new G4Box("SolidPair", dx/2.0, dy/2.0, pairThick/2.0);
logicPair = new G4LogicalVolume(PairSolid, defMat, "LogicPair");

PbSolid = new G4Box("SolidPb", dx/2.0, dy/2.0, PbThick/2.0);
logicPb = new G4LogicalVolume(PbSolid, PbMat, "LogicPb");
ScintSolid = new G4Box("SolidScint", dx/2.0, dy/2.0, SciThick/2.0);
logicScint = new G4LogicalVolume(ScintSolid, PbMat, "LogicScint");

new G4PVPlacement(0, G4ThreeVector(0, 0, -pairThick/2.0 + PbThick
    ↪ /2.0), logicPb, "PhysiPb", logicPair, false, 0);
new G4PVPlacement(0, G4ThreeVector(0, 0, pairThick/2.0 - SciThick /
    ↪ 2.0), logicScint, "PhysiScint", logicPair, false, 0);

new G4PVReplica("PhysiPairReplica", logicPair, logicEcal, kZAxis,
    ↪ nPairs, pairThick, 0);
```

Иерархия: logicEcal -> logicPair в Replica -> logicPb+logicScint

G4PVParameterised

```
G4PVParameterised( const G4String&                 pName,
                   G4LogicalVolume*                pCurrent,
                   G4LogicalVolume*                pMother,
                   const EAxis                     pAxis,
                   const G4int                     nReplicas,
                   G4VPVParameterisation*          pParam,
                   G4bool                          pSurfChk=
                       ↪ false )
```

`G4VPVParameterisation` - наследуемый пользователем и затем передаваемый класс, реализующий методы:

- `ComputeTransformation(copyNo, placement)` - задает, где находятся реплики
- `ComputeDimensions(G4VSolid&, copyNo)` - меняет размер реплик
- Опционально: `ComputeSolid` - меняет вид Solid
- Опционально: `ComputeMaterial` - меняет материал

Создание physicsList и некоторые особенности моделирования

Процедура шага симуляции для одного трека

1. Вычисление сечений и среднего свободного пробега каждого определенного процесса
2. Каждый процесс предлагает "свой"размер шага основываясь на них (и иногда других факторах), выбирается минимальный
3. Размер шага обрезается в соответствии с границами материала
4. Применяются все непрерывные процессы, пересчитывается энергия, позиция, время, если были вторичные частицы - добавляются в список, не умер ли трек
5. Применяются дискретные процессы: все вторичные частицы сохраняются в списке, не умер ли трек
6. Следующий шаг

Physics List

- Конкретный набор физических моделей (набор `G4VProcess`), используемых при различных энергиях для различных частиц
- Выбор Physics List - обязанность ученого и зависит от конкретной задачи (энергии, частицы), желаемой точности и производительности
- GEANT4 предоставляет несколько готовых Physics List, однако, можно (и это рекомендуется) собрать свой, основываясь на них
- Часто можно подсмотреть в примерах конкретную реализацию для конкретных случаев

Две части physics List:

- Электромагнитное взаимодействие
- Слабое и сильное - распады, ядерное взаимодействие и т. д.

Создание Physics List

1. Reference physics list

```
#include <QGSP_BERT.hh>
//...
runManager->SetUserInitialization(new QGSP_BERT(0));
```

Существующие: (FTFP, QGSP)_(BERT, BIC, INCLXX)_(, ATL, HP, TRV, AllHP), QBBC, QGS_BIC, Shielding...

2. G4PhysListFactory

```
#include <G4PhysListFactory.hh>
//...
G4PhysListFactory factory;
runManager->SetUserInitialization(factory.
    ↪ GetReferencePhysList("QGSP_BERT_EMV"));
```

Добавляет выбор электромагнитной части: EMV, EMX, *EMY*, EMZ, LIV, PEN, GS, LE, WVI, SS

Электромагнитные physics List

Opt	Название	Класс	Prec.
None	Standard	`G4EmStandardPhysics`	0
EMV	opt1	`G4EmStandardPhysics_option1`	- -
EMX	opt2	`G4EmStandardPhysics_option2`	-
EMY	opt3	`G4EmStandardPhysics_option3`	+
EMZ	opt4	`G4EmStandardPhysics_option4`	++
LIV	Livermore	`G4EmLivermorePhysics`	opt3 $\gamma, e-, e+$
PEN	Penelope	`G4EmPenelopePhysics`	opt3 $\gamma, e-, e+$
LE	Low Energy	`G4EmLowEPPhysics`	stand. + LE
Пожалуйста, забудьте:			
GS	Goudsmit-Sounderson	`G4EmStandardPhysicsGS`	stand. + GS
WVI	WVI (WentzelVI)	`G4EmStandardPhysicsWVI`	stand. + WVI
SS	Single scattering (SS)	`G4EmStandardPhysicsSS`	stand. + SS
–	DNA	`G4EmDNAPhysics_option(2,4,6)`	$<\approx 1eV$

Создание Physics List

3. G4VModularPhysList

```
MyPhysicsList::MyPhysicsList():G4VModularPhysicsList()
{
  G4DataQuestionaire it(photon, neutron, neutronxs);
  defaultCutValue = 0.7*mm;
  SetVerboseLevel(1);

  // EM Physics
  RegisterPhysics( new G4EmStandardPhysics(ver) );

  // Synchroton Radiation & GN Physics
  RegisterPhysics( new G4EmExtraPhysics(ver) );
  // Decays
  RegisterPhysics( new G4DecayPhysics(ver) );

  // Hadron physics
  RegisterPhysics( new G4HadronElasticPhysicsXS(ver) );
  RegisterPhysics( new G4QStoppingPhysics(ver) );
  RegisterPhysics( new G4IonBinaryCascadePhysics(ver) );
  RegisterPhysics( new G4HadronInelasticQBBC(ver));

  // Neutron tracking cut
  RegisterPhysics( new G4NeutronTrackingCut(ver) );
}
```

Создание Physics List

Пример: Убираем адронную физику

```
MyPhysicsList::MyPhysicsList():G4VModularPhysicsList()
{
  G4DataQuestionaire it(photon, neutron, neutronxs);
  defaultCutValue = 0.7*mm;
  SetVerboseLevel(1);

  // EM Physics
  RegisterPhysics( new G4EmStandardPhysics(ver) );

  // Synchroton Radiation & GN Physics
  RegisterPhysics( new G4EmExtraPhysics(ver) );
  // Decays
  RegisterPhysics( new G4DecayPhysics(ver) );

  // Hadron physics
  if(enableHadron) {
    RegisterPhysics( new G4HadronElasticPhysicsXS(ver) );
    RegisterPhysics( new G4QStoppingPhysics(ver) );
    RegisterPhysics( new G4IonBinaryCascadePhysics(ver) );
    RegisterPhysics( new G4HadronInelasticQBBC(ver));
  }
  // Neutron tracking cut
  RegisterPhysics( new G4NeutronTrackingCut(ver) );
}
```

Создание Physics List

4. С нуля: G4VUserPhysicsList

Пожалуйста, не надо

Хотя, в принципе, можно посмотреть, как построены стандартные physics Lists, но они все на деле модульные

Создание двух Physics List

Ни в коем случае!

```
//Wrong!
G4VUserPhysicsList* qgsp = new QGSP_BERT(0);
G4VUserPhysicsList* ftfp = new FTFP_BERT(0);

if(use_QGSP) {runManager->SetUserInitialization(qgsp);}
else {runManager->SetUserInitialization(ftfp);}

//Instead:
if(use_QGSP) {runManager->SetUserInitialization(new
    ↪ QGSP_BERT(0));}
else {runManager->SetUserInitialization(new FTFP_BERT(0));}

//Or:
G4VUserPhysicsList* pList;
if(use_QGSP) {pList = new QGSP_BERT(0);}
else {pList = new FTFP_BERT(0);}
runManager->SetUserInitialization(pList);
```

Production cuts и Step Limiter

Production cuts (или "пороги")

- Минимальная энергия вторичной частицы (ниже ее вторичная частица не создается и сразу убивается в точке создания)
- Наиболее влияет на энерговыделение эти ограничения на электроны
- Задается в размерности длины пробега

Step Limiter (или "длина шага")

- Задает минимальный размер шага (процесс, который постоянно "выкрикивает"определенный размер шага)
- На энерговыделение в большинстве случаев влияет не сильно, влияет на траекторию

Production cuts и Step Limiter

Production cuts (или "пороги")

```
//Inside G4VModularPhysicsList::SetCuts()
this->SetCutsWithDefault();
this->SetCutValue(0, "proton");
this->SetCutValue(DefOpt.cutVal*CLHEP::mm, "e-");
this->SetCutValue(DefOpt.cutVal*CLHEP::mm, "e+");
this->SetCutValue(DefOpt.cutVal*CLHEP::mm, "gamma");
G4VUserPhysicsList::DumpCutValuesTable();

//Or
G4UImanager* UI = G4UImanager::GetUIpointer();
UImanager->ApplyCommand("/run/setCut␣0.002␣mm");
```

Step Limiter (или "длина шага")

```
//In main()
G4VModularPhysicsList* physicsList = new FTFP_BERT;
physicsList->RegisterPhysics(new G4StepLimiterPhysics());
runManager->SetUserInitialization(physicsList);

//In detCnst
logicDet->SetUserLimits(new G4UserLimits(usLimit));
//Or...
G4LogicalVolume( ..., new G4UserLimits(usLimit), ...)
```

Прочие пороги

```
//In main()
G4VModularPhysicsList* physicsList = new FTFP_BERT;
physicsList->RegisterPhysics(new G4StepLimiterPhysics());
physicsList->RegisterPhysics(new SpecialCutsBuilder());
runManager->SetUserInitialization(physicsList);
```

```
G4UserLimits(G4double uStepMax = DBL_MAX,
             G4double uTrakMax = DBL_MAX,
             G4double uTimeMax = DBL_MAX,
             G4double uEkinMin = 0.,
             G4double uRangMin = 0. );
```

- Максимальная длина шага
- Максимальная полная длина трека
- Максимальное время жизни
- Минимальная кин. энергия
- Минимальный оставшийся пробег

Создание User Action

User Action

User Action определяют код, исполняемый Geant4 в определенных местах, создается наследованием от определенных классов

1. G4VUserPrimaryGeneratorAction
Обязательно!

- `GeneratePrimaries(G4Event*)` - в начале события при создании первичных частиц

2. G4UserRunAction

- `G4Run* GenerateRun()` - в начале рана *до* определения физики
- `BeginOfRunAction()` - сразу до входа в цикл событий (*после* определения физики)
- `EndOfRunAction()` - в конце рана

User Action

3. G4UserEventAction

- `beginOfEventAction()` - до создания первичных частиц
- `endOfEventAction()` - в конце события

4. G4UserStackingAction

ExampleN04

- `G4ClassificationOfNewTrack ClassifyNewTrack(const G4Track*)` - определение вида трека: срочный, может подождать, перенесен в следующее событие, убить
- `NewStage()` - как только нет больше срочных треков
- `PrepareNewEvent()` - в начале события, обрабатываем перенесенные

User Action

5. G4UserTrackingAction

Задает действия до и после обработки трека

- `PreUserTrackingAction(const G4Track*)`
- `PostUserTrackingAction(const G4Track*)`

6. G4UserSteppingAction

- `UserSteppingAction(const G4Step*)` - сразу после шага

Инициализация UserAction

1. Хотим только то, что требуется

```
//In main()
runManager->SetUserInitialization(new myPrimaryGenerator());
```

2. Хотим что-то еще (почти наверняка)

Класс G4VUserActionInitialization

```
class myUserAction : public G4VUserActionInitialization {
  public:
  void Build() const;
  void BuildForMaster() const;
}
void UserActionInitialization::BuildForMaster() const {
  SetUserAction(new RunAction);
}
void UserActionInitialization::Build() const {
  SetUserAction(new RunAction);
  SetUserAction(new PrimaryGeneratorAction());
  SetUserAction(new EventAction);
  SetUserAction(new SteppingAction);
}

//In main()
runManager->SetUserInitialization(new myUserAction);
```

Организация вывода используя userAction

Один из методов (как это делаю я)

1. Создать класс data, хранящий в себе все необходимые значения для вывода (например, полное энерговыделение всех детекторов)
2. Унаследовать G4Run или G4Event (легче и оптимальнее - Run), сделать его членом data
3. BeginOfRunAction: создать файл, записать хеадер
4. EndOfRunAction: закрыть файл
5. BeginOfEventAction: вывод в консоль номера события (G4Event::getEventID())
6. G4Run::RecordEvent(G4Event*) - простановка флагов, вывод в файл, очистка data (внимание! мультипоточность)
7. В SensetiveDetector в ProcessHits записываем в data все необходимые

Создание первичных частиц:
G4ParticleGun, General Particle Source и макросы

Запуск частиц

- Работаем с `G4VUserPrimaryGeneratorAction`: наследуем, определяем конструктор, деструктор, `GeneratePrimaries(G4Event*)`
- По сути, мы создаем `G4PrimaryVertex` и засовываем их в принимаемый G4Event
- Делаем мы это через `G4VPrimaryGenerator`: создаем, задаем параметры, в `GeneratePrimaries` - вызываем метод `GeneratePrimaryVertex(G4Event*)`

Geant4 уже предоставляет две реализации `G4VPrimaryGenerator`:

- G4ParticleGun
- G4GeneralParticleSource

G4ParticleGun

- Пользователь задает частицу, энергию, направление импульса, положение
- Если требуется некоторая случайность - пользователь сам генерирует все необходимые распределения, варьируя начальные параметры в каждом событии

```
void SetParticleDefinition(G4ParticleDefinition*)
void SetParticleMomentum(G4ParticleMomentum)
void SetParticleMomentumDirection(G4ThreeVector)
void SetParticleEnergy(G4double)
void SetParticleTime(G4double)
void SetParticlePosition(G4ThreeVector)
void SetParticlePolarization(G4ThreeVector)
void SetNumberOfParticles(G4int)
```

Пример с G4ParticleGun

```
#include <G4VUserPrimaryGeneratorAction.hh>
#include <G4ParticleGun.hh>
#include <G4ThreeVector.hh>

class myPriGen : public G4VUserPrimaryGeneratorAction {
  public:
    myPriGen();
    ~myPriGen();
    void GeneratePrimaries(G4Event*);
  private:
    G4ParticleGun*  fParticleGun;
}

myPriGen::myPriGen():G4VUserPrimaryGeneratorAction() {
  fParticleGun  = new G4ParticleGun(nofParticles);
  G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
  G4ParticleDefinition* particle = particleTable->FindParticle("proton");
  fParticleGun->SetParticleDefinition(particle);
  fParticleGun->SetParticleMomentumDirection(G4ThreeVector(0,0,1));
  fParticleGun->SetParticlePosition(G4TreeVector(0,0,0));
}

myPriGen::GeneratePrimaries(G4Event* evt) {
  fParticleGun->SetParticleEnergy((0.5+1.0*double(rand())/double(RAND_MAX))*
      ↪ GeV);
  fParticleGun->GeneratePrimaryVertex(evt);
}

myPriGen::~myPriGen() {
  delete fParticleGun;
}
```

Пара слов о случайности

- double - формат данных, хранящий числа с плавающей точкой в 64 битах
- Между 0 и 1 находится $2^{10} \times 2^{52}$ значений
- rand() генерирует случайное **целое** число от 0 до `RAND_MAX`
- Гарантируется лишь, что `RAND_MAX` > 32767 (на деле обычно 2^{32})
- В старом добром `double(rand())/double(RAND_MAX)` мы не только теряем точность и скорость из-за деления, но и просто и из-за ограничений `RAND_MAX`

Решение: я обожаю C++11

```
#include <random>
int main() {
  std::random_device rd; std::mt19937_64 gen(rd());
  std::normal_distribution<> d(5,2);
  std::cout << d(gen);
}
```

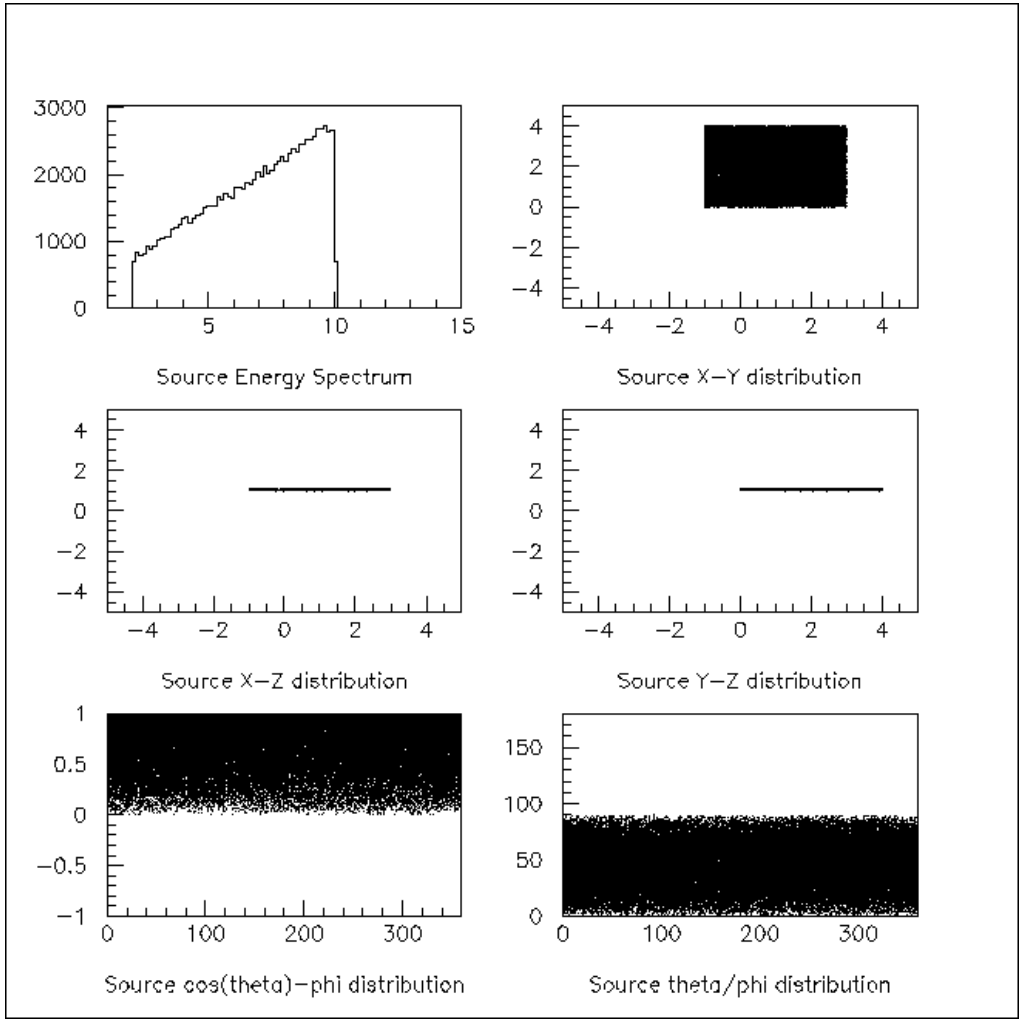
General Particle Source

- Крайне гибкий источник частиц, настраиваемый командами UI
- Самостоятельно управляет случайностью и распределениями
- Внутри `G4VUserPrimaryGeneratorAction` имеет предельно короткую интеграцию

```
#include <G4VUserPrimaryGeneratorAction.hh>
#include <G4GeneralParticleSource.hh>
#include <G4ThreeVector.hh>

class myPriGen : public G4VUserPrimaryGeneratorAction {
  public:
    myPriGen();
    ~myPriGen();
    void GeneratePrimaries(G4Event*);
  private:
    G4GeneralParticleSource*  fParticleGun;
}

myPriGen::myPriGen():G4VUserPrimaryGeneratorAction() {
  fParticleGun  = new G4GeneralParticleSource();
}
myPriGen::GeneratePrimaries(G4Event* evt) {
  fParticleGun->GeneratePrimaryVertex(evt);
}
myPriGen::~myPriGen() {
  delete fParticleGun;
}
```

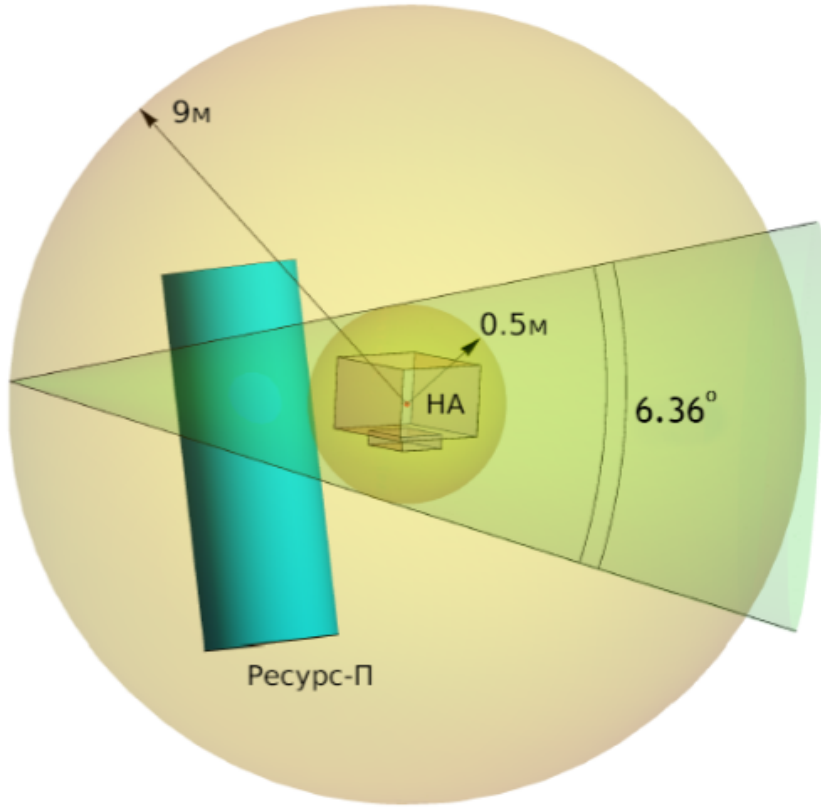
Пример настройки gps

```
/run/initialize
/run/setCut 0.01 mm

/gps/particle gamma
/gps/pos/type Plane
/gps/pos/shape Square
/gps/pos/centre 1 2 1 cm
/gps/pos/halfx 2 cm
/gps/pos/halfy 2 cm
/gps/ang/type cos
/gps/ene/type Lin
/gps/ene/min 2 MeV
/gps/ene/max 10 MeV
/gps/ene/gradient 1
/gps/ene/intercept 1
/run/beamOn 10000
```



Изотропный поток

```
/gps/pos/type Surface
/gps/pos/centre 0. 0. 0. mm
/gps/pos/shape Sphere
/gps/pos/radius 9 m

/gps/ang/type cos
/gps/ang/surfnorm true
/gps/ang/maxtheta 3.18 deg
```



$G = G_{source}\frac{N}{N_0}$, где $G_{source} = 4\pi R_0^2 \sin^2(\alpha/2) = 4\pi R_i^2$

Источник КЛ

```
/run/initialize
/run/setCut 0.01 mm
# Изотропный поток
/gps/particle ion
/gps/ion 2 4 2
/gps/ene/type Pow
/gps/ene/alpha -2.64
/gps/ene/min 10 GeV
/gps/ene/max 100 TeV
/gps/source/intensity 4749.37

/gps/source/add 11776.4
# Снова изотропный поток...
/gps/particle proton
/gps/ene/type Pow
/gps/ene/alpha -2.71
/gps/ene/min 10 GeV
/gps/ene/max 100 TeV
/gps/source/intensity 11776.4

/run beamOn 1000
```

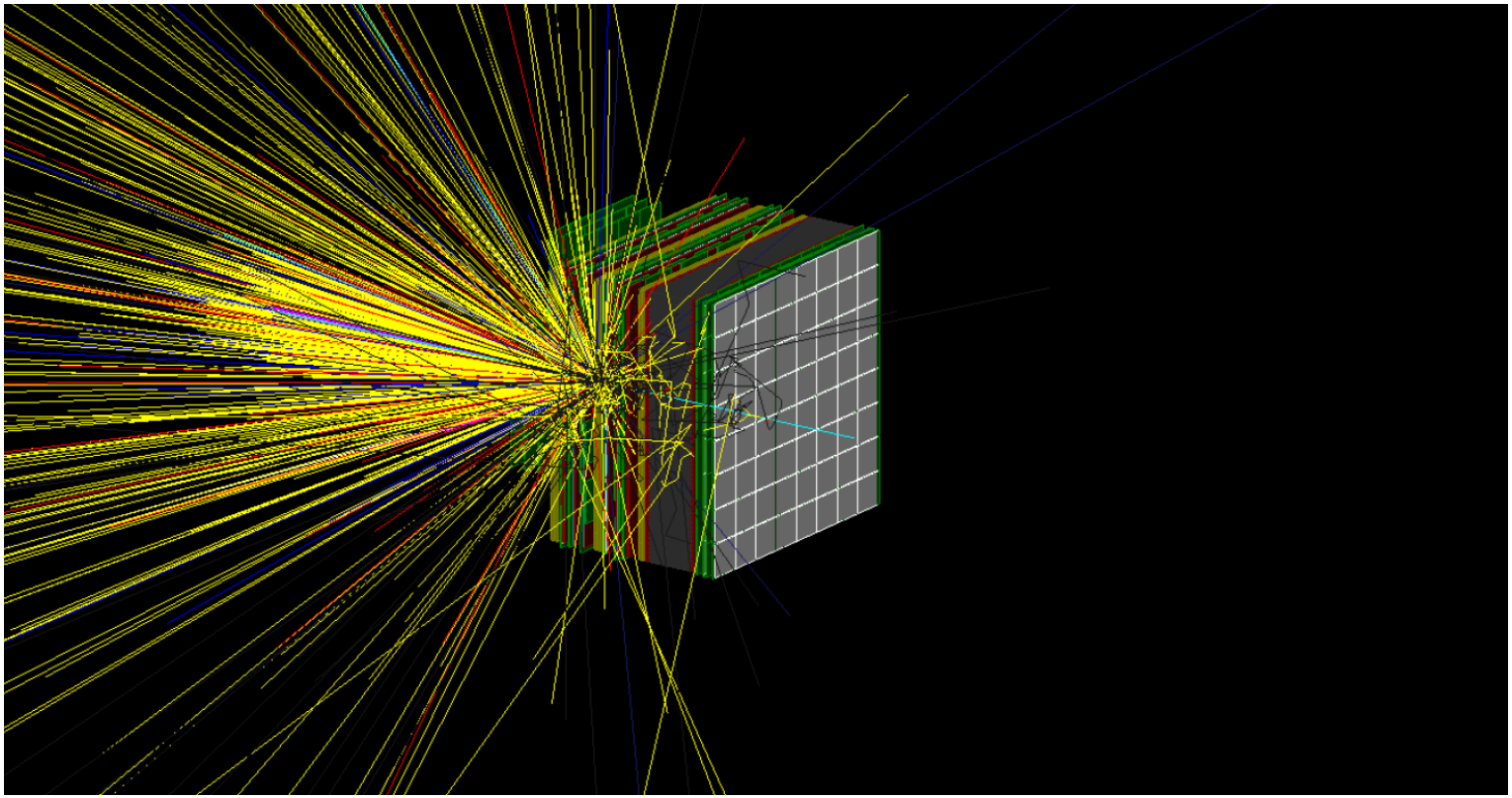
- Случайно выбирает частицу из названных (протон или гелий), так же определяются и все остальные ионы
- В принципе, можно задать не только степенной, но и вообще произвольный спектр
- Степенной спектр можно взять из классической статьи Horandel, 2002 `https://arxiv.org/abs/astro-ph/0210453`
- Экспериментальные точки: `https://lpsc.in2p3.fr/cosmic-rays-db/`

Как же запустить макрос?

```
G4UImanager* UI = G4UImanager::GetUIpointer();
UImanager->ApplyCommand("/control/execute␣macros.mac");
```

Визуализация

Зачем нужна визуализация



- Красивые картинки :)
- Визуальная проверка геометрии и работы physicsList
- Возможность реализации удобного GUI

Реализация визуализации

- Geant4 нужно собирать с поддержкой необходимой визуализации
- Слегка модифицировать main()
- Запустить для визуализации небольшой макрос

```
G4UImanager* UImanager = G4UImanager::GetUIpointer();

if (mfile!="") {
  G4String command = "/control/execute␣";
  UImanager->ApplyCommand(command+mfile);
} else {
  #ifdef G4UI_USE
    G4UIExecutive* ui = new G4UIExecutive(1, &argv[0]);
    #ifdef G4VIS_USE
      if(DefOpt.vis) {UImanager->ApplyCommand("/control/
          ↪ execute␣vis.mac");}
    #endif
    ui->SessionStart();
    delete ui;
  #endif
}
// Job termination
#ifdef G4VIS_USE
  delete visManager;
#endif
```

Макрос для визуализации

```
/vis/open OGLIQt
/vis/viewer/set/autoRefresh false
/vis/verbose errors

/vis/scene/add/trajectories smooth
/vis/modeling/trajectories/create/drawByCharge
/vis/modeling/trajectories/drawByCharge-0/default/setDrawStepPts true
/vis/modeling/trajectories/drawByCharge-0/default/setStepPtsSize 1

/vis/modeling/trajectories/create/drawByParticleID
/vis/modeling/trajectories/drawByParticleID-0/set gamma yellow
/vis/modeling/trajectories/drawByParticleID-0/set e- red
/vis/modeling/trajectories/drawByParticleID-0/set e+ blue

/vis/viewer/set/style surface
/vis/viewer/set/edge 1
/vis/viewer/set/viewpointThetaPhi 130 20
```

Мультипоточность

Как реализовать мультипоточность

https://twiki.cern.ch/twiki/bin/view/Geant4/QuickMigrationGuideForGeant4V10
Все примеры также переписаны на мультипоточность - имеет смысл посмотреть
От потока MasterThread ответвляется *n* потоков WorkerThread

Замена RunManager

```
#ifdef G4MULTITHREADED
  G4MTRunManager* runManager = new G4MTRunManager;
  runManager->SetNumberOfThreads(4);
#else
  G4RunManager* runManager = new G4RunManager;
#endif
```

Изменения в конструкции
SensetiveDetector *обязаны* быть отдельные на каждый поток.
Поэтому мы их и определяем в `ConstructSDandField()` вместо `Construct()`

Особенности вывода при реализации мультипоточности

1. Файл мы хотим создать только один раз, а не открывая каждый поток - либо определяем отдельный RunAction для Worker и Master, либо используем `if(IsMaster()){ ... }`.
2. Ровно то же самое и для закрытия файла
3. Run и Event каждые свои для своего потока - в них спокойно можно хранить текущее событие, другой поток его не тронет
4. При выводе в файл после каждого события **обязательно** надо ставить mutex:

```
#include "G4AutoLock.hh"
namespace {
        G4Mutex theMutex=G4MUTEX_INITIALIZER;
}
void OutputManager::Write(EventData od) {
        G4AutoLock l(&theMutex);
        int c = od.OutputData.write(buf);
        fwrite(buf, c, 1, OutputFile);
}
```

Фишки

Реализация консольного ввода

Простой консольный ввод:

```
myModel macros.mac output.dat 32 qgsp 0.002
```

```
int main (int argc, char** argv) {
//...
}
```

Типичный консольный ввод:

```
arss -i input.wav -o output.wav -tstep 0.1 -s
rm -rf *
model -m macros.mac -o output.dat -t 32 -qgsp -cutVal 0.002
```

В консольный ввод можно также встроить некоторые параметры модели, если, например, требуется какой-то перебор параметров и работать с sh-скриптами

Реализация консольного ввода

Самому реализовывать чтение флагов по всем стандартам непросто. Есть хорошая легковесная библиотека, ставшая сравнительно стандартной - TCLAP
http://tclap.sourceforge.net

```
int main(int argc, char** argv) {
  try {
    TCLAP::CmdLine cmd("My␣model", '␣', "ver.␣1.0");
    TCLAP::ValueArg<string> f_mName("m","mfile","Macros␣filename" ,false,"","
        ↪ filename", cmd);
    TCLAP::SwitchArg f_qgsp("","qgsp","Use␣QGSP", cmd, false);
    TCLAP::ValueArg<unsigned int> fThreads("t","","Number␣of␣threads", false,
        ↪ 1, "int", cmd);
    cmd.parse( argc, argv );

    mfile=f_mName.getValue();
    //...
  } catch(TCLAP::ArgException &e) {
    std::cout << "Error:␣" << e.error() << "␣for␣arg␣" << e.argId() << std::
        ↪ endl;
    return -1;
}
//Continue
}
```

Флаг наличия ядерного взаимодействия

```
void SteppingAction::UserSteppingAction(const G4Step* step) {
  G4VPhysicalVolume* vol = step->GetPreStepPoint()->GetTouchableHandle()->
      ↪ GetVolume();
  if(vol->GetName() != "PhysiDet") {return;}
  const G4VProcess* proc = step->GetPostStepPoint()->GetProcessDefinedStep();
  const G4Track* track = step->GetTrack();
  G4ProcessType process_type = proc->GetProcessType();
  if(process_type!=fElectromagnetic &&
     process_type!=fTransportation &&
     track->GetParentID() == 0 &&
     proc->GetProcessName() != "StepLimiter") {
               const MyRun* thisRun = dynamic_cast<const MyRun*>(G4RunManager
                   ↪ ::GetRunManager()->GetCurrentRun());
               if(thisRun==0) {cout << "Bad dynamic_cast." << endl; throw 5;}
               EventData* theData = &(thisRun->data);
               (theData->OutputData).mask |= N2RWC_FRAG;
     }
```

Geantino

Geantino - специфическая виртуальная частица в Geant4, пролетающая насквозь всего по прямой и не оставляющая ни в чем энерговыделение. Крайне полезна, например, при вычислении геомфактора методами Монте-Карло. Однако, несмотря на то, что энерговыделение она не оставляет, ее все еще можно увидеть в SensitiveDetector!

```
G4bool SensitiveGlobal::ProcessHits(G4Step *step)
{
  double energyDepos = step->GetTotalEnergyDeposit();
  G4Track *thisTrack = step->GetTrack();
  G4String pName = thisTrack->GetDefinition()->GetParticleName();
  if(pName == "geantino") {energyDepos = 1.0;}
  //Something here processes the eDep
  return true;
}
```

Заключительное слово

- Geant4 - замечательный инструмент, однако, как и любому моделированию, на 100% ему доверять нельзя
- Очень аккуратно подходите к выбору physicsList, порогов и шага
- Обязательно перепроверяйте геометрию, поначалу выводите всю информацию в консоль (куда встал детектор? какая плотность у кремния? какие пороги встали?)
- Верное программирование ведет в Geant4 и к оптимизации, а о ней очень надо заботиться - скорее всего, нужна будет большая статистика, да и просто сам по себе Geant4 требователен
- Крайне аккуратно относитесь к утечкам памяти
- Используйте бинарные выходные файлы, а не текстовые
- После моделирования удобно хранить выходные файлы либо в архивах, либо в Root Tree и использовать соответствующие библиотеки - быстрее загрузка

Спасибо за внимание!