Основы геометрии

ЛЕКЦИЯ 2

Содержание

- Связь геометрии и ядра Geant4
- **G4VUserDetectorConstruction**
- Структура геометрического объема
- Формы
- Логические объемы и их методы
- Физические объемы и иерархия
- Реализация простейшей геометрии

G4RunManager

Связь с ядром Geant4 осуществляет объекта класса управления G4RunManager (или G4MTRunManger).

```
11  #ifdef G4MULTITHREADED
12      runManager = new G4MTRunManager;    //многопоточный режим
13  #else
14      runManager = new G4RunManager;      //однопоточный режим
15  #endif
```

Данный класс содержит метод **G4RunManger**::**SetUserInitialization**(), который позволяет инициализировать и связывать с ядром разные части процесса моделирования (*геометрия, описание физических процессов или классы действий*) .

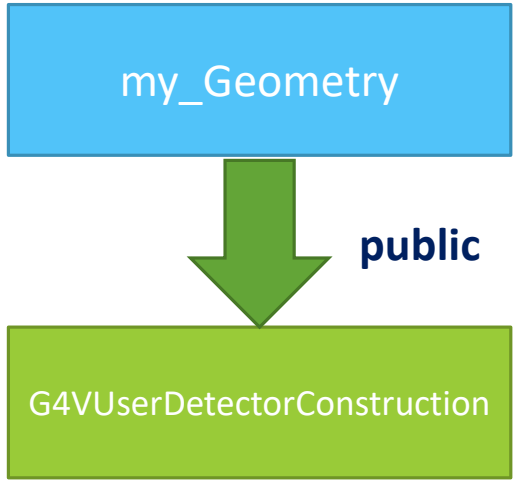
```
runManager->SetUserInitialization(new Geometry());
```
(*где Geometry - пользовательская геометрия*)

Кроме того данный класс обладает группой статических методов, позволяющих из любой части проекта получить указатель на любой из объектов моделирования.

```
Geometry* my_geometry = static_cast<Geometry*>(
        G4RunManager::GetRunManager()->GetUserDetectorConstruction());
```

G4VUserDetectorConstruction

Это базовый абстрактный класс для любой пользовательской геометрии. Реализация потомка данного класса являет обязательным этапом в каждом проекте моделирования.



Данный класс обладает единственным чисто виртуальным методом **Construct**().

```
virtual G4VPhysicalVolume* Construct() = 0;
```

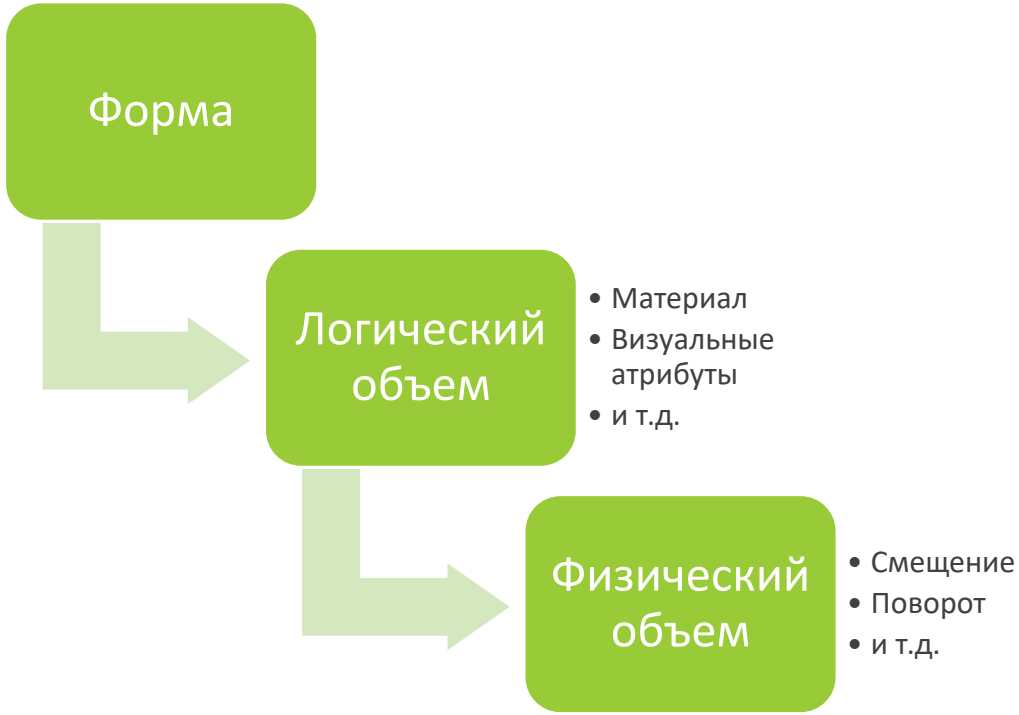
Данный метод вызывается объектом класса **G4RunManager** во время инициализации

(*т.е. во время вызова метода* **G4RunManager**::**Initialize**())

Структура геометрического объекта

Метод **Construct**() возвращает указатель на **G4VPhysicalVolume** – физический объем.

Любой геометрический объем в Geant4 строится по следующему принципу:

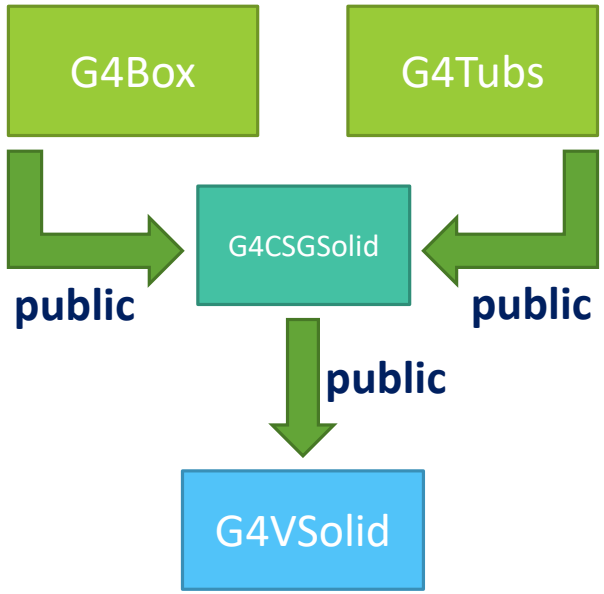


1. Создается **форма** геометрического объекта: (куб, сфера и т.п.)

2. Создается **логический объем**. Используется ранее созданная форма, которой присваиваются различные свойства: материал, цвет и т.п.

3. Создается **физический объем**. Используется ранее созданный физический объем, и указывается как он должен быть расположен в пространстве/другом объеме.

Формы



Создание любого геометрического объекта в Geant4 начинается с **формы**.

В Geant4 существует целый набор примитивных форм, являющихся потомками абстрактного класса **G4CSGSolid** (*Constructive Solid Geometry*)

Внимание: Вся память под формы должна выделяться с использованием **new**, после чего формы регистрируются в **G4SolidStore**, в одну из задач которого входит освобождение памяти по завершении работы.

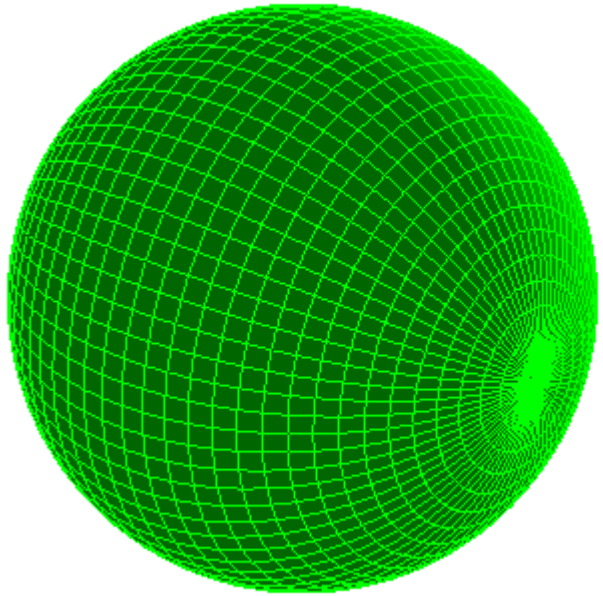
Кроме того для всех форм предусмотрены методы на расчет занимаемого геометрического объема:

```
G4cout << "Volume = "        << box->GetCubicVolume() << '\n';
```

и площади поверхности:

```
G4cout << "Surface area = " << box->GetSurfaceArea() << '\n';
```

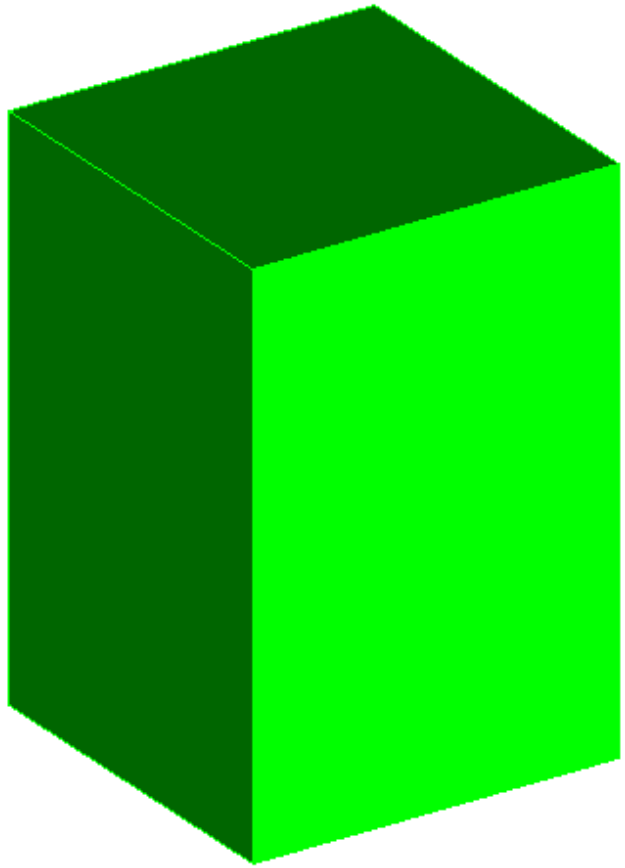
Простейшие CSG формы



G4Orb

```
G4Orb(const G4String &pName,      //имя формы
                G4double pRmax)   //максимальный радиус
```

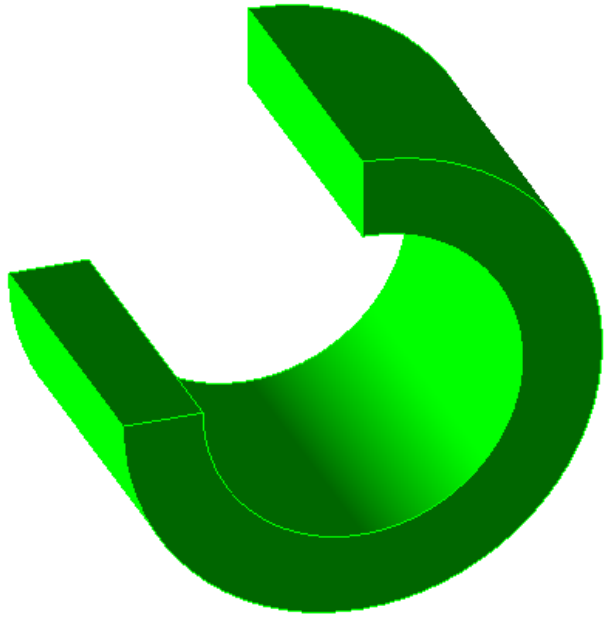
Простейшие CSG формы



G4Box

```
G4Box(const G4String& pName,     //имя формы
            G4double pX,         //половина длинны по X
            G4double pY,         //половина длинны по Y
            G4double pZ)         //половина длинны по Z
```

Простейшие CSG формы



G4Tubs

```
G4Tubs( const    G4String& pName,      //имя формы
                 G4double pRMin,       //внутренний радиус
                 G4double pRMax,       //внешний радиус
                 G4double pDz,         //половина длинны по Z
                 G4double pSPhi,       //стартовый угол от которого начинается трубка
                 G4double pDPhi )      //финальный угол на до которого откладывается трубка
```

Другие формы

Полный список доступных форм представлен в Приложении D

Логические объемы

Логический объем (*или экземпляр класса* **G4LogicalVolume**) представляет собой форму со (*к примеру физическими или визуальными*) свойствами.

```
G4LogicalVolume(G4VSolid*              pSolid,        //указатель на форму
                G4Material*            pMaterial,     //указатель на материал
                const G4String&        name,          //имя логического объема
                G4FieldManager*        pFieldMgr=0,   //указатель на поле (магнитное, электрическое и т.п.)
                G4VSensitiveDetector*  pSDetector=0,  //указатель на детектор
                G4UserLimits*          pULimits=0,    //указатель на пользовательские ограничения
                G4bool                 optimise=true) //по умолчанию - оптимизировать
```

- Указатели на материла и форму не могут быть нулевыми
- Поле, детектор и пользовательские ограничения – опциональны
- Для каждого элемента доступны методы типа Set/Get на изменение и получение значений
- Данный класс не может быть использован в качестве базового при наследовании

Физические объемы и иерархия объемов

Физический объем – это логический объем с позиционными характеристиками относительно:

- начала координат. Такой объем называется «мировым» Примечание: *в данном случае pMotherLogical = 0*;
- другого логического объема, к которому он принадлежит

Наиболее часто используемый способ размещения единичного логического объема:

```
G4PVPlacement(G4RotationMatrix*     pRot,           //матрица поворота
              const G4ThreeVector   &tlate,         //вектор смещения
              G4LogicalVolume*      pCurrentLogical,//используемый логический объем
              const G4String&       pName,          //имя физического объема
              G4LogicalVolume*      pMotherLogical, //материнский логический объем
              G4bool                pMany,          //не используется в Geant4
              G4int                 pCopyNo,        //номер физического объема, т.е.
                                                    //(сколько раз использовался
                                                    //данный логический объем)
              G4bool                pSurfChk=false) //проверка на пересечение
```

Иерархия объемов: методы G4LogicalVolume

Определить количество дочерних объемов из логического можно с помощью:

```
inline G4int GetNoDaughters() const;
```

Получить указатель на дочерний объем с номером **i** (нумерация начинается от 0)

```
inline G4VPhysicalVolume* GetDaughter(const G4int i) const;
```

Определить **ОБЩЕЕ** количество объемов в иерархии (*включая дочерние у дочерних и т.д.*)

```
G4int TotalVolumeEntities() const;
```

Расчёт массы для логического объема

```
G4double GetMass(G4bool forced=false, G4bool propagate=true,
                 G4Material* parMaterial=0);
```

Метод **GetMass**() возвращает массу дерева логических объемов исходя из свойств материалов. Учитывает как основной объем, так и все вложенные дочерние *(по умолчанию)*. Если значение параметра *propagate* = **false**, то учитывается только материнский объем.

Примечание: Данный метод является крайне ресурсоемким, поэтому при его повторном вызове значение берется из кэша. Однако если параметры геометрии изменились, то данное значение следует пересчитать, для чего можно вызвать метод **ResetMass**()

Реализация простейшей геометрии: Шаг 1

- Унаследуем класс **G4VUserDetectorConstruction**

```
15  class Geometry: public G4VUserDetectorConstruction {
16  public:
17      virtual G4VPhysicalVolume *Construct();
18  };
```

Примечание: нам не понадобиться какой-либо особый конструктор или деструктор, так что их реализовывать не будем

Реализация простейшей геометрии: Шаг 2

Реализуем метод **Construct**(). Первое что нужно сделать – это задать материнский мир, для этого:

- Зададим размер и форму мира:

```
G4double size = 50 * m;
G4Box *world = new G4Box("world", size / 2., size / 2., size / 2.);
```

Зададим логический объем: нам нужно указать форму (*уже есть*) и материал.

- Т.к. тема «материалы» пока не изучена, то просто воспользуемся материалом «воздух» из базы NIST следующим образом:

```
G4NistManager *nist = G4NistManager::Instance();
G4Material *world_mat = nist->FindOrBuildMaterial("G4_AIR");
```

- И зададим логический объем используя форму и материал:

```
G4LogicalVolume *world_log = new G4LogicalVolume(world, world_mat, "world_log");
```

Реализация простейшей геометрии: Шаг 2

- Т.к. метод **Construct**() должен вернуть указатель на **G4VPhysicalVolume** реализуем физический объем следующим образом:

```
return new G4PVPlacement(0, G4ThreeVector(), world_log, "world_PVP", 0, false, 0);
```

где выделенный 0 означает что это «материнский мир»

- Таким образом мы получим:

```
10  G4VPhysicalVolume* Geometry::Construct() {
11
12      G4double size = 50 * m;
13      G4Box *world = new G4Box("world", size / 2., size / 2., size / 2.);
14
15      G4NistManager *nist = G4NistManager::Instance();
16      G4Material *world_mat = nist->FindOrBuildMaterial("G4_AIR");
17      G4LogicalVolume *world_log = new G4LogicalVolume(world, world_mat, "world_log");
18
19      return new G4PVPlacement(0, G4ThreeVector(), world_log, "world_PVP", 0, false, 0);
20  }
```

Реализация простейшей геометрии: Шаг 2 - результат

- Запустив, мы увидим кубик – «наш мир»:

Реализация простейшей геометрии: Шаг 3

Добавим в наш мир объект – шар.

- Зададим форму:

```
G4Orb *orb = new G4Orb("my_orb", 20. * mm);
```

- Логический объем (*воспользуемся ранее созданным материалом для мира*):

```
G4LogicalVolume *orb_log = new G4LogicalVolume(orb, world_mat, "my_orb_log");
```

- Физический объем, разместив его в логическом мире:

```
new G4PVPlacement(0, G4ThreeVector(), orb_log, "my_orb_PVP", world_log, false, 0);
```

Дочерний лог. объем

Материнский лог. объем

Примечание: *нам не нужна специальная переменная для хранения указателя на физический объем, т.к. в конструкторе уже сообщается к кому его надо привязать и куда передать.*

Реализация простейшей геометрии: Шаг 3

- В результате получилось:

```
12  G4VPhysicalVolume* Geometry::Construct() {
13
14      G4double size = 50 * m;
15      G4Box *world = new G4Box("world", size / 2., size / 2., size / 2.);
16
17      G4NistManager *nist = G4NistManager::Instance();
18      G4Material *world_mat = nist->FindOrBuildMaterial("G4_AIR");
19      G4LogicalVolume *world_log = new G4LogicalVolume(world, world_mat, "world_log");
20
21      G4Orb *orb = new G4Orb("my_orb", 20. * mm);
22      G4LogicalVolume *orb_log = new G4LogicalVolume(orb, world_mat, "my_orb_log");
23      new G4PVPlacement(0, G4ThreeVector(), orb_log, "my_orb_PVP", world_log, false, 0);
24
25      return new G4PVPlacement(0, G4ThreeVector(), world_log, "world_PVP", 0, false, 0);
26  }
```

Однако, при выводе на экран мы получим кубик, т.к. шар находится внутри куба.

Реализация простейшей геометрии: Шаг 4

- Установим визуальное свойство для материнского мира – «**невидимый**»:

```
world_log->SetVisAttributes(G4VisAttributes::Invisible);
```

В результате получим:

```
12  G4VPhysicalVolume* Geometry::Construct() {
13
14      G4double size = 50 * m;
15      G4Box *world = new G4Box("world", size / 2., size / 2., size / 2.);
16
17      G4NistManager *nist = G4NistManager::Instance();
18      G4Material *world_mat = nist->FindOrBuildMaterial("G4_AIR");
19      G4LogicalVolume *world_log = new G4LogicalVolume(world, world_mat, "world_log");
20      world_log->SetVisAttributes(G4VisAttributes::Invisible);
21
22      G4Orb *orb = new G4Orb("my_orb", 20. * mm);
23      G4LogicalVolume *orb_log = new G4LogicalVolume(orb, world_mat, "my_orb_log");
24      new G4PVPlacement(0, G4ThreeVector(), orb_log, "my_orb_PVP", world_log, false, 0);
25
26      return new G4PVPlacement(0, G4ThreeVector(), world_log, "world_PVP", 0, false, 0);
27  }
```

Реализация простейшей геометрии: Шаг 4 - финал