Геометрия.

В основе геометрии Geant4 лежит концепция логических и физических объемов. Логический объем представляет собой элемент детектора определенной формы заданного материала, чувствительности и т. п. Кроме того, каждый логический объем может содержать внутри другие объемы. Физический объем представляет собой пространственное расположение логического объема относительно логического материнского объема (Подробнее ниже).

Для реализации геометрии в Geant4 необходимо унаследовать абстрактный базовый класс G4VUserDetectorConstruction. Если открыть описание шаблона данного класса, то можно заметить, что он содержит один

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

чисто виртуальный, т. е. обязательный для реализации, метод.

Чтобы понять, на что возвращается указатель в данном методе, рассмотрим, как реализуется геометрия в Geant4.

Формы

Любой геометрический объект строится в 3 этапа:

- На первом этапе создается основа будущего логического объема - его форма, включающая геометрические характеристики.
- На втором - описываются свойства формы, такие как: материал, сцинтилляционные свойства и т. п. Кроме того, на втором этапе форме можно установить различные визуальные атрибуты, упрощающие дальнейшую визуализацию моделирования.
- На третьем этапе осуществляется расположение модифицированной формы в пространстве, её поворот, смещение и т. п.

Начнем с рассмотрения примитива - прямоугольного параллелепипеда, или, проще говоря, коробки. За реализацию данной формы отвечает класс G4Box.

Чтобы создать простейшую форму – коробку, следует воспользоваться конструктором:

```
G4Box(const G4String& pName, G4double pX, G4double pY, G4double pZ);  // Конструктор коробки с именем, и половинами длин по pX,pY,pZ
```

Сразу отметим, что в качестве значений по X, Y, Z принимаются ПОЛОВИНЫ от реальной длины, ширины и высоты. Данное свойство характерно для большинства форм, используемых в G4.

Соответственно, реализация объекта простейшей формы будет выглядеть следующим образом:

```
G4Box *box = new G4Box("box", 5*cm, 5*cm, 5*cm);
```

Все геометрические формы в Geant4 реализованы в соответствии с концепцией Constructive Solid Geometry (CSG). Большинство комплексных форм описывается за счет их граничной поверхности, которая может быть первого, второго порядка, или B-spline поверхностью. Все это сделано, опираясь на ISO STEP стандарт CAD систем.

Возвращаясь к базовым формам, в Geant4 представлен целый набор примитивных форм, являющихся потомками абстрактного класса G4SCGSolid. К этому списку можно отнести: коробки, сферы, конусы, трубки и др.

Кроме того, для всех базовых форм предусмотрены методы для расчета занимаемого геометрического объема созданной ранее коробки:

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

а также её площади поверхности:

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

Логические объемы

Сама по себе форма не содержит никакой информации об объекте, кроме его геометрических размеров. Однако для моделирования взаимодействия излучения с веществом необходимо описать свойства этого самого вещества, в котором осуществляется моделирование. С этой целью в Geant4 предоставлен класс объектов, называемый логический объем. Одной из основных целей логического объема является связь формы с физическими свойствами объекта, как пример, материал, из которого она(форма) сделана.

Объекты логического объема относятся к классу G4LogicalVolume. Конструктор данного класса выглядит следующим образом:

```
G4LogicalVolume(G4VSolid* pSolid,                    //форма
                G4Material* pMaterial,               //материал
                const G4String& name,                //имя
                G4FieldManager* pFieldMgr=0,
                G4VSensitiveDetector* pSDetector=0,
                G4UserLimits* pULimits=0,
                G4bool optimise=true);
```

Стоит отметить, что поля «форма и материал» (pSolid and pMaterial) не должны быть равными **nullptr**. Поля: pFieldMgr, pSDetector, pULimits являются опциональными и в данной работе описаны не будут.

Рассматривая формы, следует обратить внимание на визуализацию геометрических объектов, например цвет или прозрачность, так как визуальные атрибуты являются частью именно логической, а не физической формы (см. ниже).

Для того, чтобы изменить цвет того или иного геометрического объекта, следует вызвать метод:

```
void  SetVisAttributes (const G4VisAttributes& VA);
```

который в качестве аргумента принимает интересующий нас параметр визуализации.

Рассмотрим простейший куб с длиной грани 10 см.

```
G4double box_size = 5 * cm;
G4Box *box = new G4Box("box", box_size, box_size, box_size);

auto box_log = new G4LogicalVolume(box, nist->FindOrBuildMaterial("G4_SODIUM_IODIDE"),
                                   "box_LOG");

box_log->SetVisAttributes(G4Colour::Blue());
```

В качестве формы используется «коробка». Объект log_box представляет собой логический объем, в качестве аргументов - принимающий форму и моделируемый материал. Кроме того, в рассматриваемом примере осуществляется передача логическому объему визуального атрибута «синий цвет». Стоит отметить, что теперь все физические объемы, которые будут использовать в качестве параметра данный логический объем, окрасятся синим цветом.

Отдельно стоит отметить имена форм и логических объемов. В рамках объектов своего класса эти имена должны быть уникальны. Для формы, логического объема и физического объема одного геометрического объекта можно использовать одно имя.

Как и для формы, для логического объема предусмотрено несколько полезных методов. Один из них позволяет рассчитать массу:

```
G4cout << "Mass = " << box_log->GetMass() / g << " g\n";
```

Для получения результата, к примеру в граммах, следует разделить получаемый за счет метода GetMass() результат на «грамм».

```
Mass = 3667 g
```

Физический объем. Материнский объем

Последним этапом построения геометрического объекта средствами Geant4 является реализация его физического объема или, иначе говоря, расположение в пространстве.

Geant4 предусматривает несколько способов осуществления данного процесса. Рассмотрим самый простейший среди них. В качестве инструмента создания физического объема воспользуемся конструктором G4PVPlacement (наследует G4VPhysicalVolume)

```
G4PVPlacement(G4RotationMatrix *pRot,                 //матрица поворота
              const G4ThreeVector &tlate,             //вектор смещения
              G4LogicalVolume *pCurrentLogical,       //логический объем
              const G4String& pName,                  //имя
              G4LogicalVolume *pMotherLogical,        //материнский объем
              G4bool pMany,                           //не используется
              G4int  pCopyNo,                         //номер копии
              G4bool pSurfChk=false);                 //проверка на
                                                      //пересечение с другими объемами
```

где pRot — матрица поворота, tlate — вектор смещения, pCurrentLogical — логический объем, pName — имя (идентификатор), pMotherLogical — материнский объем, pMany — данный параметр не реализован в текущей версии Geant4 (можно использовать false), pCopyNo - номер копии логического объема, для первого физического объема следует использовать значение 0, затем 1 и т.д.

Рассмотренный конструктор позволяет расположить в пространстве моделирования единичный логический объем, а также повернуть и сместить его относительно материнского объема.

Все геометрические объекты в Geant4 позиционируются внутри созданных ранее других объектов. Соответственно каждый создаваемый

физический объем смещается относительно центра логического объема, называемого материнским для данного физического объема.

Не трудно догадаться, что должен существовать один единственный материнский логический объем, который становится изначальным или главным для всех последующих объемов.

```
world_PV = new G4PVPlacement(nullptr, G4ThreeVector(), world_log, "world_PV", nullptr, false, 0);
```

Также такой объем называют миром.

В качестве указателя на материнский объем для этого физического объема следует использовать нулевой указатель.

Помним, что единственный необходимый для реализации геометрии метод:

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

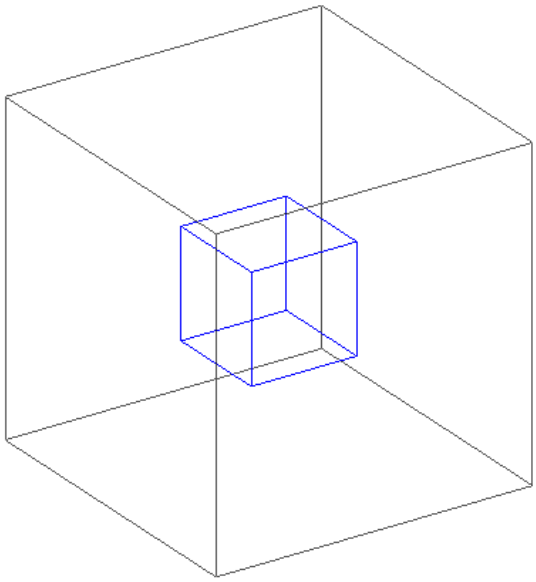
возвращает указатель на физический объем. Объем, на который возвращается указатель, является главным материнским объемом или представленным выше миром, с привязанными к нему всеми дочерними объемами, используемыми в геометрии.

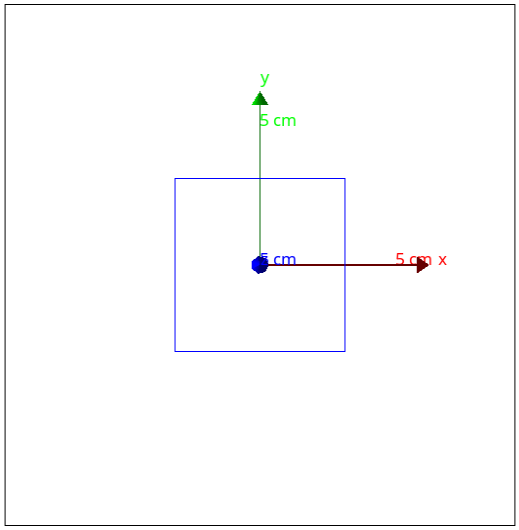
Рассмотрим пример реализации метода Construct() для размещения представленной выше коробки в мире.

```
G4VPhysicalVolume* Geometry::Construct() {
    auto size = 15 * cm;
    auto world = new G4Box("world", size / 2., size / 2., size / 2.);
    auto world_log = new G4LogicalVolume(world, world_mat, "world_log");
    world_log->SetVisAttributes(G4VisAttributes::Invisible);
    auto world_PV = new G4PVPlacement(nullptr, G4ThreeVector(), world_log, "world_PV",
                                      nullptr, false, 0);
    G4double box_size = 2.5 * cm;
    auto *box = new G4Box("box", box_size, box_size, box_size);
    auto box_log = new G4LogicalVolume(box, nist->FindOrBuildMaterial("G4_SODIUM_IODIDE"),
                                       "box_LOG");
    box_log->SetVisAttributes(G4Colour::Blue());
    new G4PVPlacement(nullptr, G4ThreeVector(0, 0, 0), box_log, "box_PV", world_log, false, 0);
    return world_PV;
}
```

В данном примере мир представляет собой коробку 15x15x15 см. С нулевым смещением и без поворота внутри данного мира размещается реализованная

выше синяя коробка 5x5x5 см. Если визуализировать данную геометрию средствами G4, будет получен следующий результат:

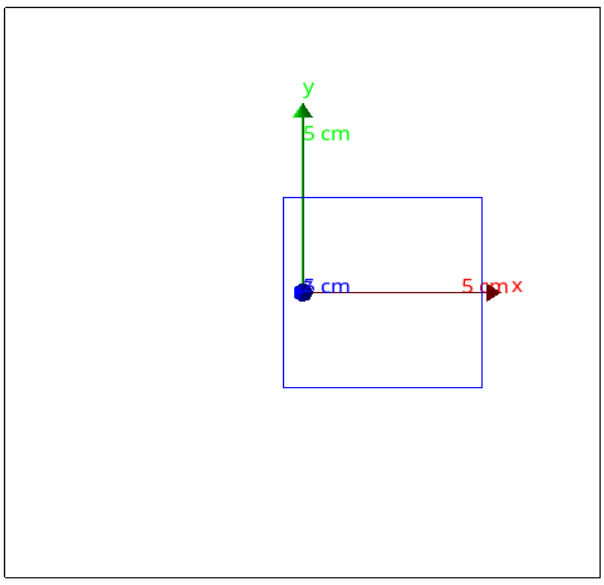




Попробуем сместить коробку относительно центра мира на 2 см по оси X. Для этого в качестве аргументов вектора смещения следует передать значения (2*cm,0,0)

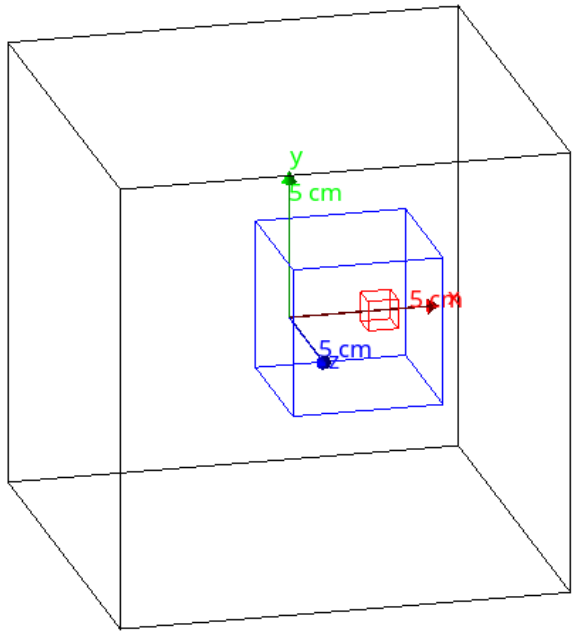
```
new G4PVPlacement(nullptr, G4ThreeVector(2 * cm, 0, 0), box_log, "box_PV", world_log, false, 0);
```

В результате:



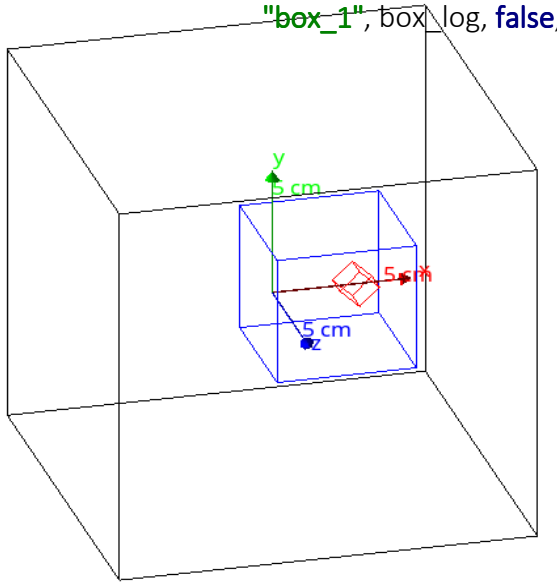
```
auto box_1 = new G4Box("box_1", box_size_1, box_size_1, box_size_1);
auto box_1_log = new G4LogicalVolume(box_1, nist->FindOrBuildMaterial("G4_BGO"), "box_1");
box_log->SetVisAttributes(G4Colour::Red());
new G4PVPlacement(nullptr, G4ThreeVector(1 * cm, 0, 0), box_1_log, "box_1", box_log, false, 0);
```

Усложним задачу и добавим внутри синего куба еще один, дочерний для него. Соответственно, в качестве материнского объема теперь выступает не **world_log** а **box_log**. Кроме того, осуществим смещение нового кубика относительно материнского на 1*см по оси X. В результате:



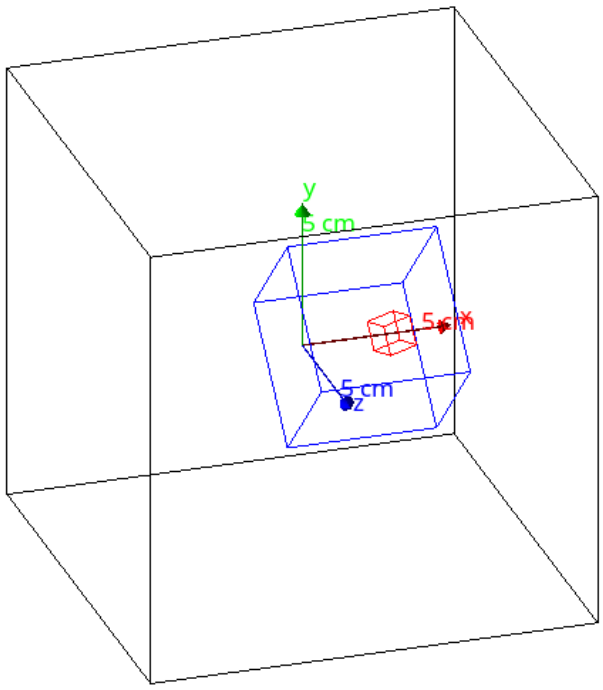
Можно заметить, что смещение красного кубика произошло не в глобальной системе координат мира, а в системе координат синего кубика. Для большей наглядности осуществим поворот сначала красного кубика на 45 градусов за счет угла phi,

```
new G4PVPlacement(new G4RotationMatrix(45 * deg, 0, 0), G4ThreeVector(1 * cm, 0, 0), box_1_log,
                  "box_1", box_log, false, 0);
```



а затем синего на 45 градусов с помощью угла teta

```
new G4PVPlacement(new G4RotationMatrix(0, 45 * deg, 0), G4ThreeVector(2 * cm, 0, 0), box_log,
                  "box_PV", world_log, false, 0);
```



Как можно видеть, поворот осуществляется за счет передачи указателя на объект класса матрицы поворота, где

первое поле — phi,

второе поле — teta

третье поле — psi

Контейнеры-хранилища

Все создаваемые в процессе моделирования формы, логические объемы и физические объемы помещаются в специальные контейнеры-хранилища для удобного доступа к элементам геометрии из других частей проекта.

- Для форм — G4SolidStore
- Для логических объемов — G4LogicalStore
- Для физических объемов — G4PhysicalVolumeStore.

Рассмотрим содержимое хранилища. Для получения доступа к контейнеру необходимо вызвать статический метод GetInstance(), возвращающий указатель на выбранное хранилище. Так как по своей природе хранилище является контейнером, то с ним можно работать с помощью итераторов. Например, распечатаем имена всех созданных в проекте форм:

```
for (auto item : *G4SolidStore::GetInstance())
    G4cout << item->GetName() << '\n';
```

```
world
box
box_1
```

в результате в консоль выведется следующая информация:

Аналогичную информацию можно получить для хранилищ логических и физических объемов.

Если мы добавим объем с уже существующим именем-идентификатором, то он успешно добавится в общий список хранилища. К примеру, добавив дополнительный физический объем, используя уже существующий «красный кубик» с тем же именем, что было задано ранее:

```
new G4PVPlacement(new G4RotationMatrix(45 * deg, 0, 0), G4ThreeVector(1 * cm, -3 * cm, 0),
                  box_1_log, "box_1", box_log, false, 1);
```

и распечатав содержимое хранилища физических форм:

```
for (auto item : *G4PhysicalVolumeStore::GetInstance())
    G4cout << item->GetName() << '\n';
```

в консоли будет выведено:

```
world_PV
box_PV
box_1
box_1
```

Однако, данная ситуация может вызвать проблемы в некоторых задачах моделирования. Например, если возникнет необходимость идентификации конкретного детектирующего элемента в массиве.

При добавлении нового физического объема из уже существующего ранее логического объема, дублирования данного логического объема не возникает:

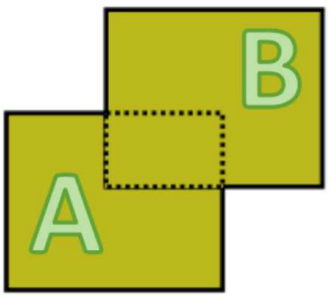
```
for (auto item : *G4LogicalVolumeStore::GetInstance())
      G4cout << item->GetName() << '\n';
```

в консоли:

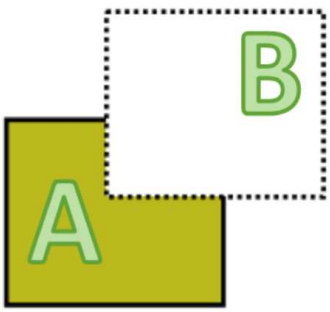
```
world_log
box_LOG
box_1
```

Логические операции над формами

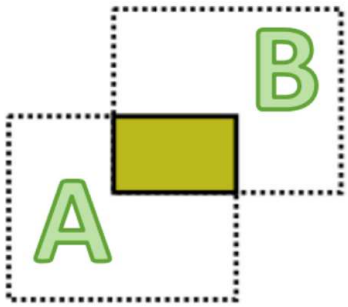
Кроме инструментов для создания простейших геометрических форм в Geant4 предоставлен набор логических операций, позволяющих реализовывать сложные пространственные объекты.
Данные операции принято называть логическими. К ним относятся



G4UnionSolid — объединение двух геометрических форм A и B в одну;



G4SubtractionSolid — вычитание из одной формы A другой формы B;



G4IntersectionSolid — Общая область пересечения формы A и формы B.

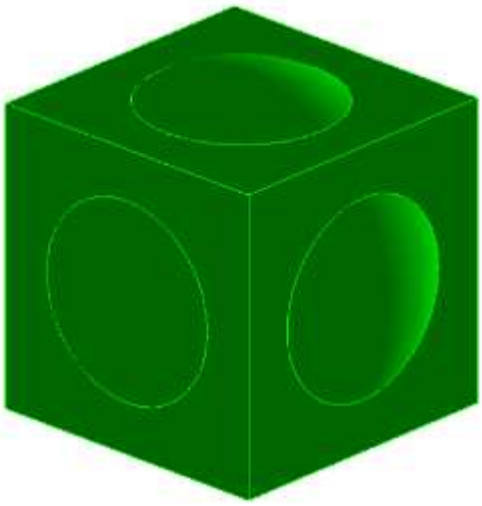
Все логические формы наследуют класс G4BooleanSolid, который, в свою очередь, наследует класс G4VSolid. Отсюда следует, что любую логическую форму можно использовать как обычную форму для построения логического объема.

Рассмотрим примеры построения геометрических объемов, формы которых созданы за счет логических операций.

Допустим дан куб (G4Box) с длинной грани 10*см и шар (G4Orb) радиуса 6*см. Тогда их пересечение можно описать в виде:

```
G4double box_len = 10 * cm;
G4double orb_r = 6 * cm

;auto box = new G4Box("box", box_len / 2., box_len / 2., box_len / 2.);
auto orb = new G4Orb("orb", orb_r);

auto my_union = new G4UnionSolid("union", box, orb);
auto my_union_log = new G4LogicalVolume(my_union, nist->FindOrBuildMaterial("G4_BGO"),
                                        "union");

my_union_log->SetVisAttributes(G4Color::Green());

new G4PVPlacement(nullptr, G4ThreeVector(), my_union_log, "union", world_log, false, 0);
```

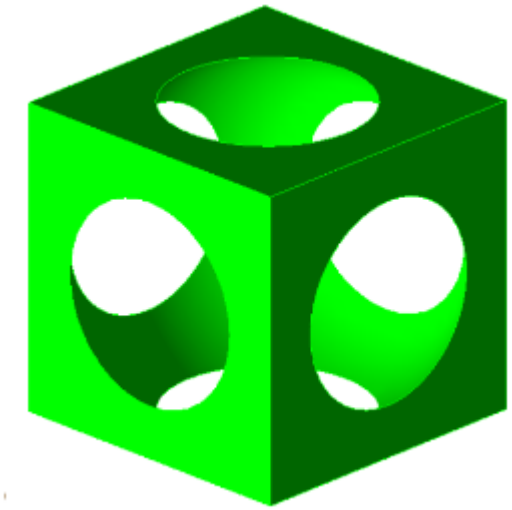
Полученная конструкция будет иметь вид, представленный на рисунке



Аналогично, для тех же начальных данных G4SubtractionSolid можно использовать следующим образом:

```
auto box = new G4Box("box", box_len / 2., box_len / 2., box_len / 2.);
auto orb = new G4Orb("orb", orb_r);

auto my_subtraction = new G4SubtractionSolid("subtraction", box, orb);
auto my_subtraction_log = new G4LogicalVolume(my_subtraction, nist->FindOrBuildMaterial("G4_BGO"),
                                              "subtraction");

my_subtraction_log->SetVisAttributes(G4Color::Green());

new G4PVPlacement(nullptr, G4ThreeVector(), my_subtraction_log, "subtraction", world_log, false, 0);
```

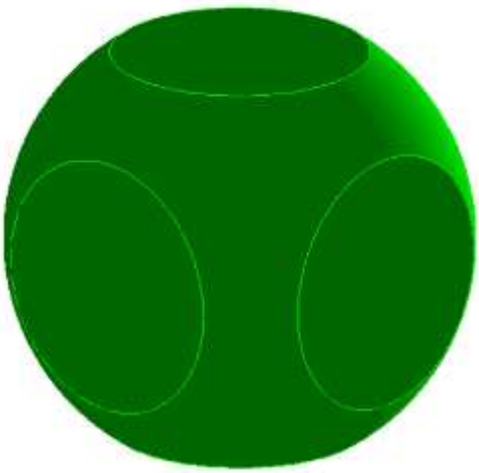
с результатом, представленным на рисунке.



И наконец, для операции пересечения

```
auto box = new G4Box("box", box_len / 2., box_len / 2., box_len / 2.);
auto orb = new G4Orb("orb", orb_r);

auto my_intersection = new G4IntersectionSolid("intersection", box, orb);
auto my_intersection_log = new G4LogicalVolume(my_intersection, nist->FindOrBuildMaterial("G4_BGO"),
                                               "intersection");

my_intersection_log->SetVisAttributes(G4Color::Green());

new G4PVPlacement(nullptr, G4ThreeVector(), my_intersection_log, "intersection", world_log, false, 0);
```

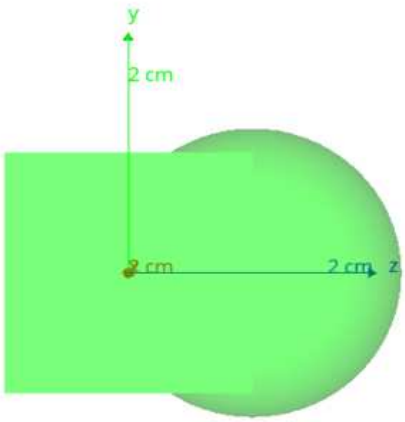
в результате получится следующее изображение



При создании логических форм можно использовать как смещение, так и поворот. Всегда следует иметь ввиду, что все операции осуществляются относительно центра первой в методе формы.

```
auto my_union = new G4UnionSolid("union", box, orb, nullptr, G4ThreeVector(0, 0, 10. * cm));
```

Центром полученной таким образом логической фигуры всегда является центр первой фигуры. Результат полученного выше объединения, расположенного в начале координат, можно увидеть на рисунке.



Контейнеры для логических объемов

Представим, что в рамках задачи в геометрии многократно используется один и тот же сложный массив объектов. С целью упрощения построения таких геометрий в Geant4 реализованы специальные контейнеры. Они позволяют группировать логические объемы с целью многократного воспроизведения в мире.

Класс, отвечающий за данные контейнеры, называется G4AssemblyVolume. Данный класс имеет два конструктора:

```
G4AssemblyVolume();
G4AssemblyVolume( G4LogicalVolume* volume,        //логический объем
                  G4ThreeVector& translation,     //смещение объема
                  G4RotationMatrix* rotation);    //поворот объем
```

В случае применения параметризованного конструктора смещение и поворот осуществляются относительно центра контейнера.

Существует метод на добавление логического объема в уже существующий контейнер.

```
void AddPlacedVolume( G4LogicalVolume* pPlacedVolume,      //размещаемый объем
                      G4ThreeVector& translation,          //смещение
                      G4RotationMatrix* rotation);         //поворот
```

В контейнер можно добавить другой, ранее созданный контейнер.

Для размещения контейнера в материнском мире следует воспользоваться

```
void AddPlacedAssembly( G4AssemblyVolume* pAssembly,        //размещаемый контейнер
                        G4ThreeVector& translation,         //смещение
                        G4RotationMatrix* rotation);        //поворот
```

```
void MakeImprint( G4LogicalVolume* pMotherLV,               //материнский объем
                  G4ThreeVector& translationInMother,       //смещение в мат. объеме
                  G4RotationMatrix* pRotationInMother,      //поворот в мат. объеме
                  G4int copyNumBase = 0,                    //номер копии
                  G4bool surfCheck = false );               //проверка на пересечение
```

методом:

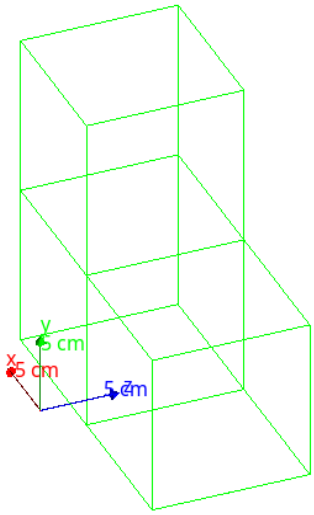
При многократном добавлении одного и того же логического объема в контейнер новые логические копии этого объема не появляются. Однако, все размещенные таким образом физические объемы будут иметь уникальные имена

Рассмотрим пример, в котором один и тот же куб трижды добавляется в контейнер. В каждой итерации осуществляется смещение куба по одной из осей на длину его грани.

Код данной задачи будет выглядеть следующим образом:

```
G4double box_len = 10 * cm;
auto box = new G4Box("box", box_len / 2., box_len / 2., box_len / 2.);
auto box_log = new G4LogicalVolume(box, nist->FindOrBuildMaterial("G4_BGO"),
                                   "box");

box_log->SetVisAttributes(G4Color::Green());

G4ThreeVector pos_av_vect(0, 0, 10 * cm);

auto AV = new G4AssemblyVolume(box_log, pos_av_vect, nullptr);

pos_av_vect.setX(10 * cm);
AV->AddPlacedVolume(box_log, pos_av_vect, nullptr);
pos_av_vect.setY(10 * cm);
AV->AddPlacedVolume(box_log, pos_av_vect, nullptr);

pos_av_vect.set(0, 0, 0);
AV->MakeImprint(world_log, pos_av_vect, nullptr);
```

Итоговая конфигурация будет соответствовать рисунку.



Если вывести имена физических объемов, доступные в данной задаче, то в консоли будет получен следующий результат.

```
world_PV
av_1_impr_1_box_pv_0
av_1_impr_1_box_pv_1
av_1_impr_1_box_pv_2
```

Также стоит отметить, что если расположить данный контейнер в мире еще один раз, финальный список физических объемов примет следующую форму:

```
world_PV
av_1_impr_1_box_pv_0
av_1_impr_1_box_pv_1
av_1_impr_1_box_pv_2
av_1_impr_2_box_pv_0
av_1_impr_2_box_pv_1
av_1_impr_2_box_pv_2
```