

Сложные геометрические формы

ЛЕКЦИЯ 3





Содержание

- ☐ Логические формы и их виды
- ☐ Контейнеры
- ☐ Использование контейнера при построении геометрии

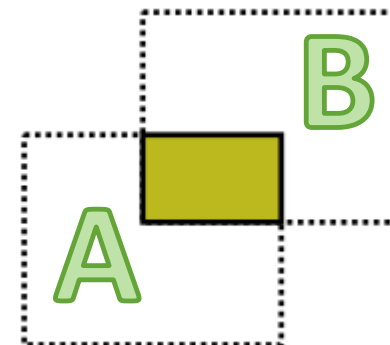
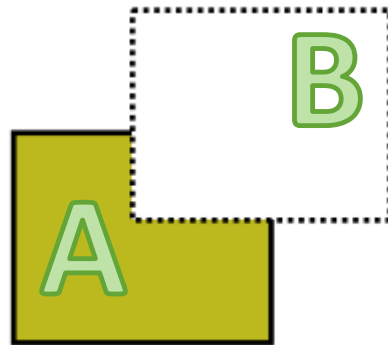
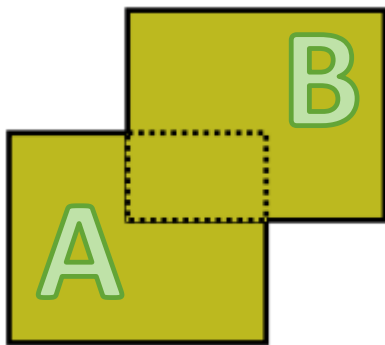


Логические формы

В Geant4, кроме инструментов для создания простейших геометрических форм, предоставлен набор логических операций, позволяющих реализовывать сложные пространственные объекты.

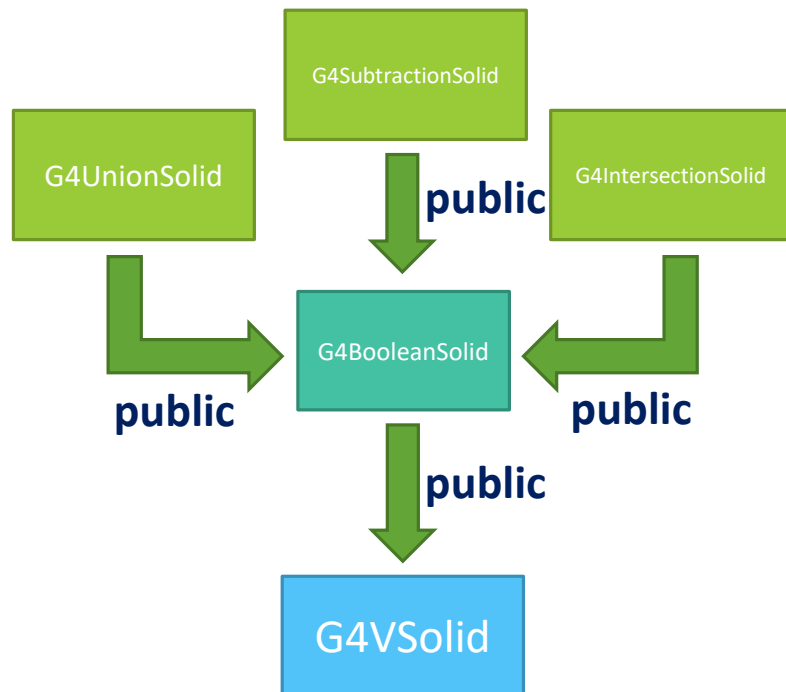
К этим операциям относятся:

- ❑ **G4UnionSolid** - объединение двух геометрических объектов **A** и **B** в один
- ❑ **G4SubtractionSolid** – вычитание из объекта **A** объектом **B**
- ❑ **G4IntersectionSolid** – общая область для объектов **A** и **B**





Логические формы

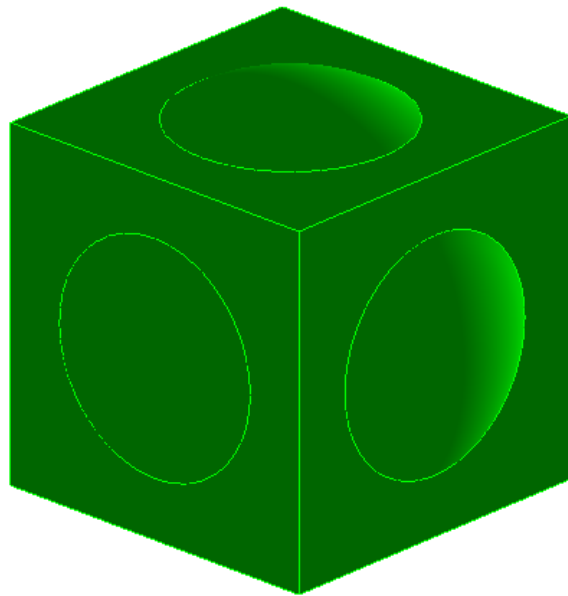


Все логические формы наследуют класс **G4BooleanSolid**, который наследует класс **G4VSolid**, следовательно, любую логическую форму можно использовать как обычную форму для построения логического объема.

Примечание: во всех примерах в качестве фигуры A используется куб с длинной стороны 20 мм, а для фигуры B используется шар радиуса 12 мм.



Виды логических форм

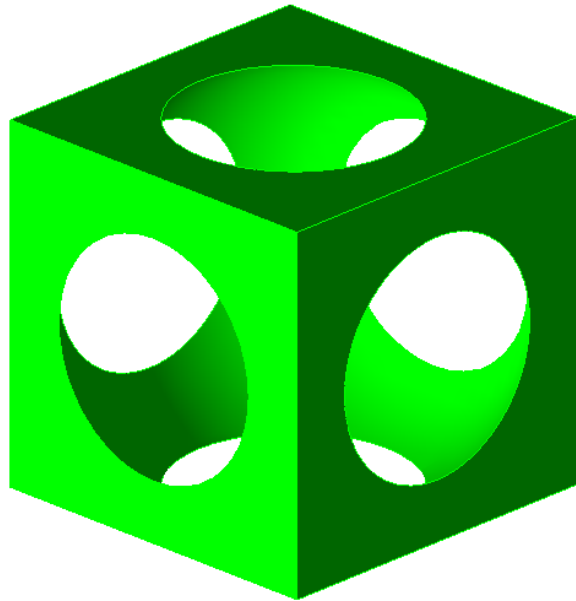


G4UnionSolid

```
24 G4Box *box = new G4Box("my_box", 10. * mm, 10. * mm, 10. * mm);
25 G4Orb *orb = new G4Orb("my_orb", 12. * mm);
26
27 G4UnionSolid* Union = new G4UnionSolid("my_union", box, orb);
28
29 G4LogicalVolume *Union_log = new G4LogicalVolume(Union, world_mat, "my_union_log");
30 new G4PVPlacement(0, G4ThreeVector(), Union_log, "my_union_PVP", world_log, false, 0);
```



Виды логических форм

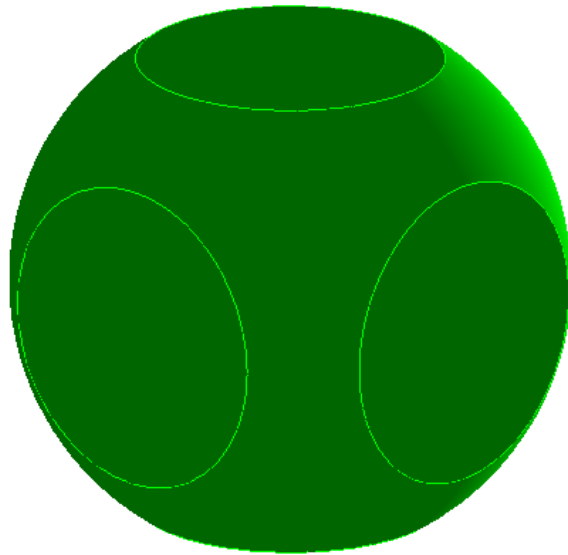


G4SubtractionSolid

```
26 G4Box *box = new G4Box("my_box", 10. * mm, 10. * mm, 10. * mm);  
27 G4Orb *orb = new G4Orb("my_orb", 12. * mm);  
28  
29 G4SubtractionSolid* Sub = new G4SubtractionSolid("my_Sub", box, orb);  
30  
31 G4LogicalVolume *Sub_log = new G4LogicalVolume(Sub, world_mat, "my_Sub_log");  
32 new G4PVPlacement(0, G4ThreeVector(), Sub_log, "my_Sub_PVP", world_log, false, 0);
```



Виды логических форм



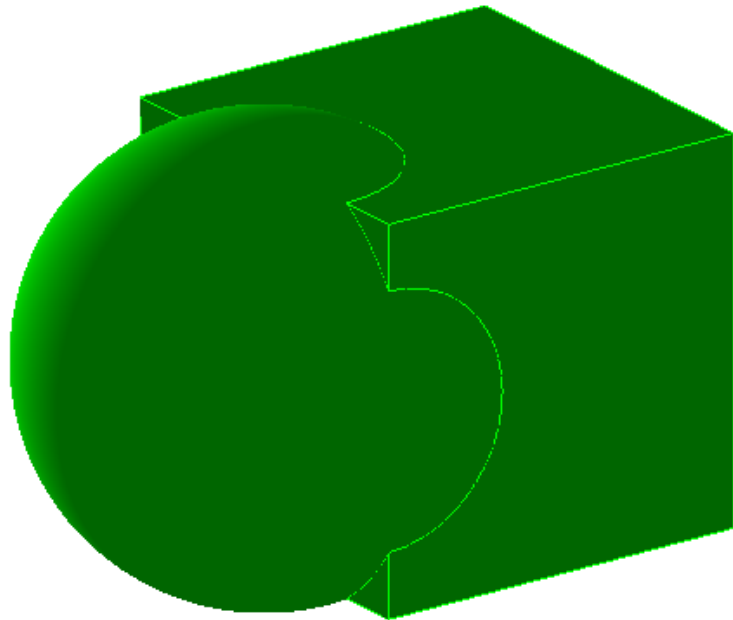
G4IntersectionSolid

```
26 G4Box *box = new G4Box("my_box", 10. * mm, 10. * mm, 10. * mm);
27 G4Orb *orb = new G4Orb("my_orb", 12. * mm);
28
29 G4IntersectionSolid* Inter = new G4IntersectionSolid("my_inter", box, orb);
30
31 G4LogicalVolume *Inter_log = new G4LogicalVolume(Inter, world_mat, "my_inter_log");
32 new G4PVPlacement(0, G4ThreeVector(), Inter_log, "my_inter_PVP", world_log, false, 0);
```

Использование смещения и поворота при создании логических форм



При создании логических форм можно использовать смещение поворот. В данном случае началом координат является фигура **A**. К примеру:



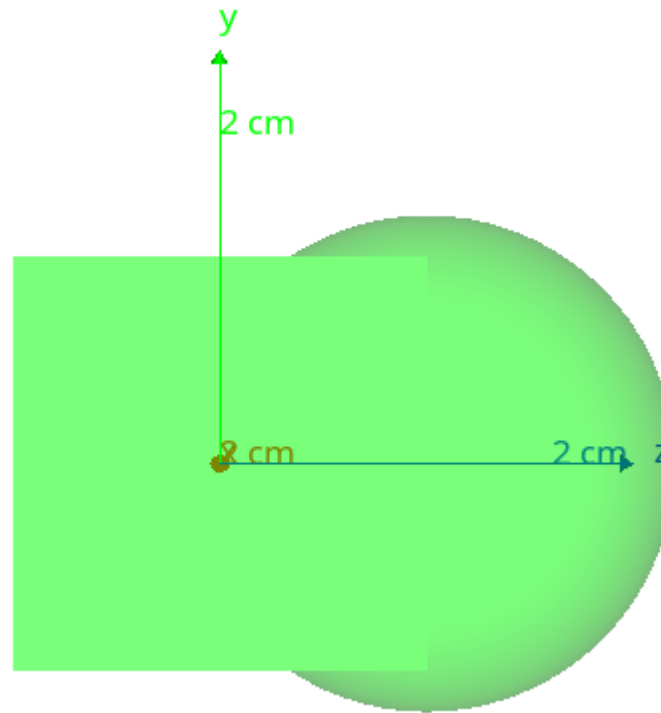
Примечание: на данной иллюстрации объектом A – куб, а объектом B - шар. Камера расположена «сзади» поэтому может показаться что шар сместился влево.

```
G4UnionSolid* Sub = new G4UnionSolid("my_Sub",box,orb,0,G4ThreeVector(0,0,10.*mm));
```


Использование смещения и поворота при создании логических форм



Центром логической фигуры всегда является центр фигуры **A**





Контейнеры в Geant4:

При использовании повторяющихся блоков той или иной конструкции можно сгруппировать такие объемы в общий контейнер, представленный в Geant4 как **G4AssemblyVolume**. Данный класс содержит два конструктора:

❑ Параметризованный:

```
G4AssemblyVolume( G4LogicalVolume* volume,      //Логический объем, центр которого
                  //станет началом координат
                  G4ThreeVector& translation,    //Вектор смещения
                  G4RotationMatrix* rotation)    //Матрица поворота
```

❑ конструктор «по умолчанию»:

```
G4AssemblyVolume();
```

В данном случае создается пустой контейнер, а после добавления первого логического объема, центр данного объема станет началом координат.



Использование контейнера при построении геометрии: Шаг 1

- ❑ Создадим пустой контейнер:

```
G4AssemblyVolume *aV = new G4AssemblyVolume();
```

- ❑ Создадим простейший логический объем формы : «коробка» и поместим в контейнер. Для добавления объемов используется метод **G4AssemblyVolume::AddPlacedVolume()**:

```
G4ThreeVector vect;  
G4LogicalVolume *box_log = new G4LogicalVolume(box, world_mat, "box_log");
```

```
aV->AddPlacedVolume(box_log, vect, 0);
```

Примечание: Обратите внимание что здесь создается нулевой вектор vect в котором в дальнейшем будет меняться поле X.

В контейнер так же можно добавить и другой контейнер с помощью метода **G4AssemblyVolume::AddPlacedAssembly()**



Использование контейнера при построении геометрии: Шаг 2

- Заполним контейнер объемами, расположенными на расстоянии 24 мм друг от друга:

```
for (int i = 0; i < 120; i += 24) {  
    vect.setX(i);  
    aV->AddPlacedVolume(box_log, vect, 0);  
}
```

- Разместим полученный контейнер в мире. Центром является центр первого добавленного логического объема. Для размещения используется метод **G4AssemblyVolume::MakeImprint()**:

```
vect.setX(0);  
aV->MakeImprint(world_log, vect, 0);
```

Примечание: В данном случае вектор уже представляет собой смещение относительно мира, поэтому выставляем значение X на 0



Использование контейнера при построении геометрии: Финал

□ Полный код выглядит следующим образом:

```
37  G4AssemblyVolume *aV = new G4AssemblyVolume();
38
39  G4ThreeVector vect;
40  G4LogicalVolume *box_log = new G4LogicalVolume(box, world_mat, "box_log");
41
42  aV->AddPlacedVolume(box_log, vect, 0);
43
44  for (int i = 0; i < 120; i += 24) {
45      vect.setX(i);
46      aV->AddPlacedVolume(box_log, vect, 0);
47  }
48
49  vect.setX(0);
50  aV->MakeImprint(world_log, vect, 0);
```

Использование контейнера при построении геометрии: Финал

