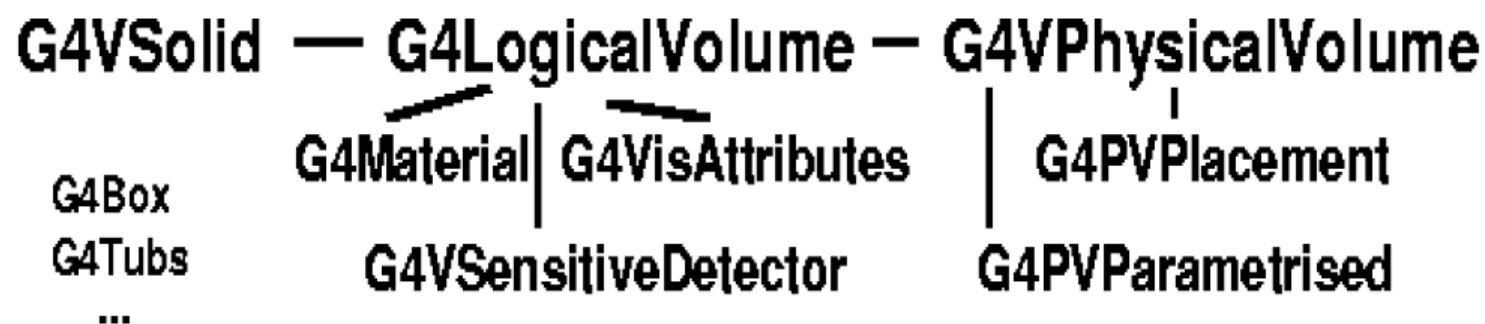


## Объем описывается в три этапа

- форма (*G4VSolid*)
- логический объем (*G4LogicalVolume*)
- физический объем (*G4VPhysicalVolume*)

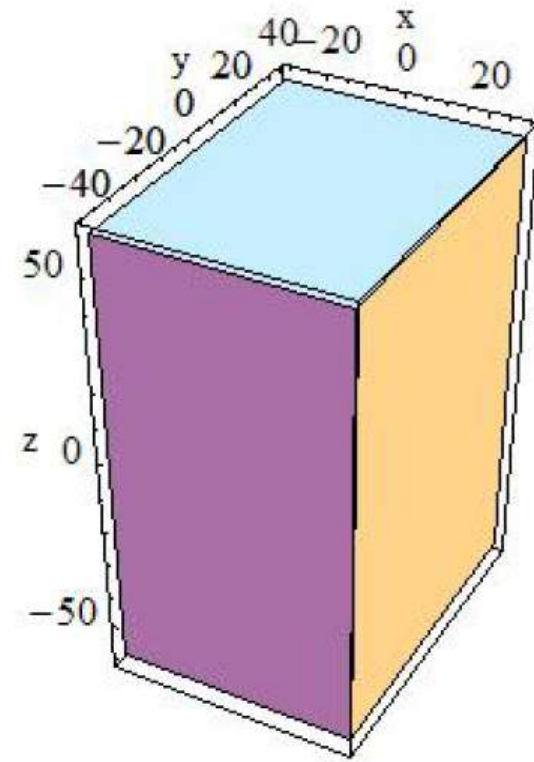


## Параллелепипед

```
G4VSolid* scint_solid = new  
    G4Box(const G4String& pName,  
          G4double pX,  
          G4double pY,  
          G4double pZ);
```

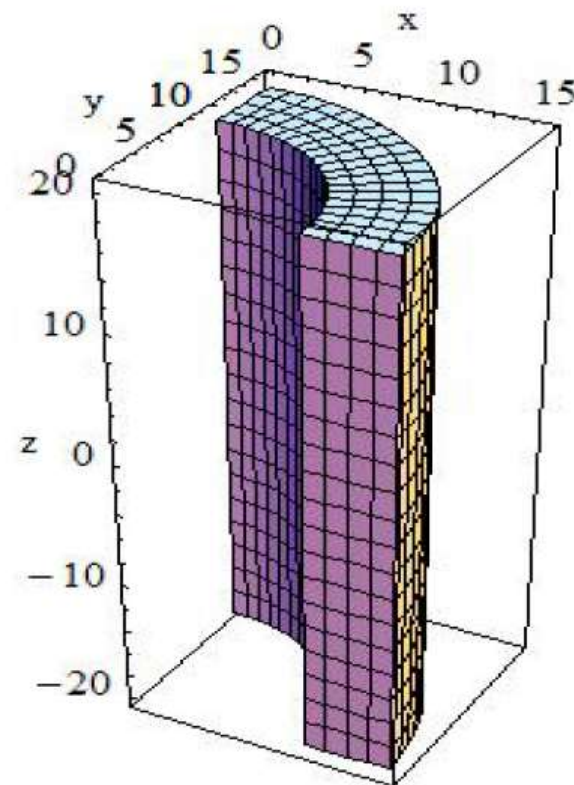
Пример:

```
G4Box* aBox = new  
G4Box("BoxA", 20.0*cm, 40.0*cm, 60.0*cm);
```



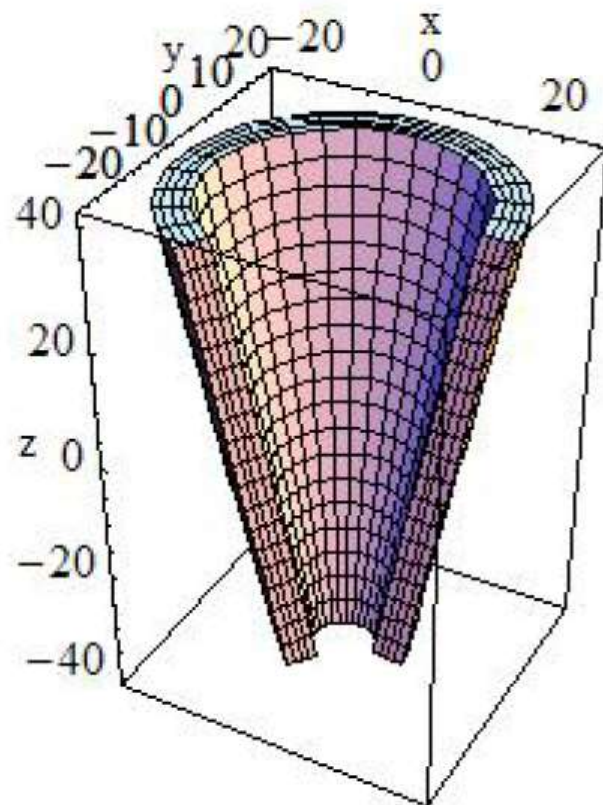
## Цилиндр

```
G4VSolid* calor_solid = new  
  G4Tubs(const G4String& pName,  
          G4double pRMin,  
          G4double pRMax,  
          G4double pDz, - полувысота  
          G4double pSPhi,  
          G4double pDPhi)
```



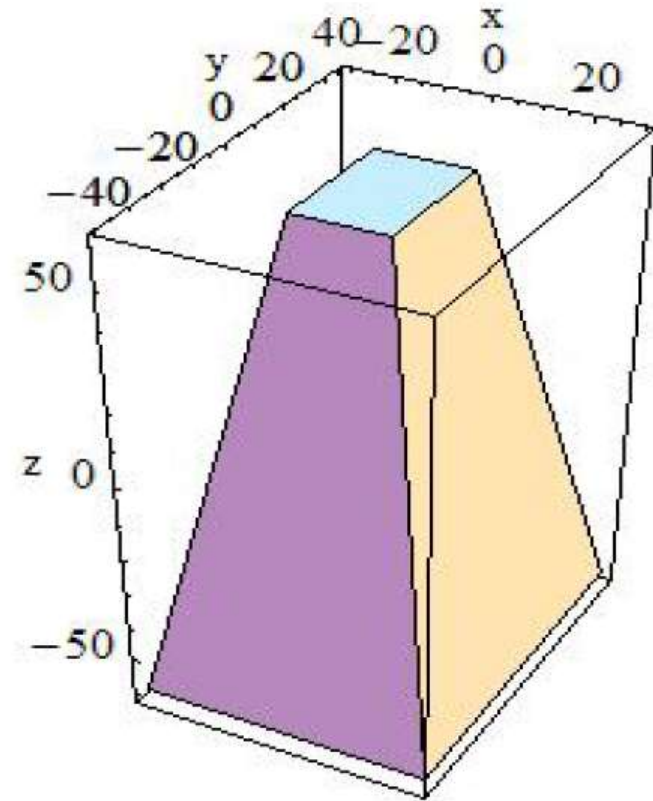
## Конус

```
G4VSolid* scint_solid = new  
G4Cons(const G4String& pName,  
        G4double pRmin1,  
        G4double pRmax1,  
        G4double pRmin2,  
        G4double pRmax2,  
        G4double pDz,  
        G4double pSPhi,  
        G4double pDPhi)
```



## Трапезоид

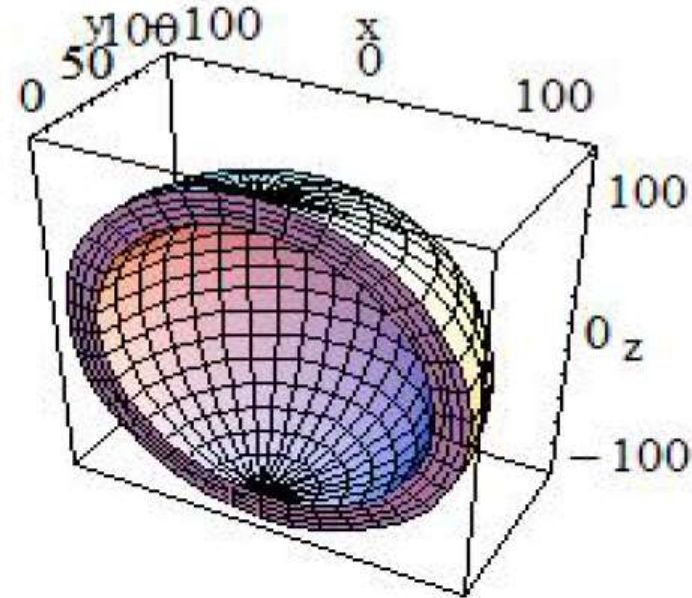
```
G4VSolid* aSolid = new  
G4Trd(const G4String& pName,  
        G4double dx1,  
        G4double dx2,  
        G4double dy1,  
        G4double dy2,  
        G4double dz)
```





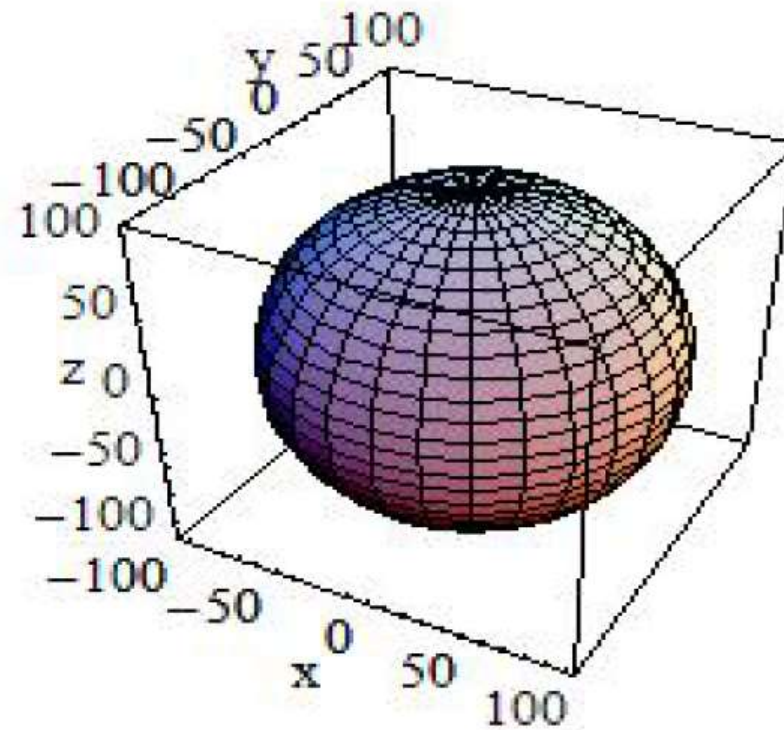
## Сфера

```
G4VSolid* aSolid = new  
  G4Sphere(const G4String& pName,  
            G4double  pRmin,  
            G4double  pRmax,  
            G4double  pSPhi,  
            G4double  pDPhi,  
            G4double  pSTheta,  
            G4double  pDTheta )
```



## Шар

```
G4VSolid* aSolid = new  
  G4Orb(const G4String& pName,  
        G4double pRmax)
```



# Логический объем

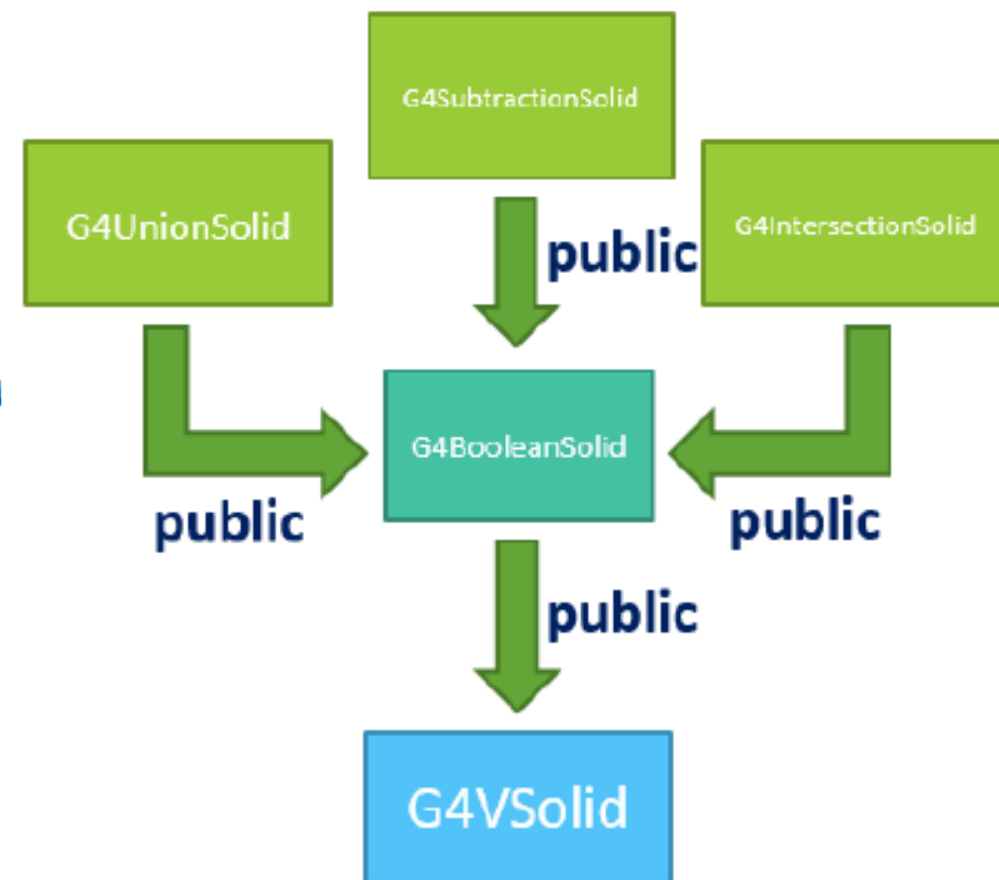
```
Pb_box = new G4Box("Pb_cal", 10. * cm, 10. * cm, 1. * cm);  
Pb_mat=nist->FindOrBuildMaterial("G4_Pb");  
Pb_log = new G4LogicalVolume(Pb_box, Pb_mat,"Pb_cal_log");  
  
G4VisAttributes* CGreen = new G4VisAttributes(G4Colour(0.0,1.0,0.0));  
Pb_log->SetVisAttributes(CGreen);
```

- Кроме геометрических параметров, содержит описание материала, заполняющего объем, свойства визуализации объема и описание детектирующей способности



## Булевы формы

- Объединение двух форм при помощи логической операции
- *G4UnionSolid*, *G4SubtractionSolid*, *G4IntersectionSolid*
- При описании положение второй формы описывается в координатной системе первой
- Не следует злоупотреблять булевыми формами, т.к. усложняется трекинг и увеличивается время моделирования события. По возможности лучше использовать обычные вложенные объемы



```
G4ThreeVector Transport(0, 0, -3. * cm);
G4RotationMatrix* RM = new G4RotationMatrix(0, 0, 0);
G4Tubes* cyl = new G4Tubes("Cylinder", 0, 50*mm, 50*mm, 0, twopi);
G4Box* box = new G4Box("Box", 20*mm, 30*mm, 40*mm);

G4UnionSolid* union = new G4UnionSolid("Box+CylinderMoved", box, cyl, RM, Transport);
G4IntersectionSolid* intersection = new G4IntersectionSolid("Box*CylinderMoved", box, cyl, RM, Transport);
G4SubtractionSolid* subtraction = new G4SubtractionSolid("Box-CylinderMoved", box, cyl, RM, Transport);
```

# Физический объем

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

- Строится на основе логического объема
- Описывает положение объема в пространстве

```
G4RotationMatrix* ZERO_RM = new G4RotationMatrix(0, 0, 0);
```

```
G4ThreeVector Pb_vect = G4ThreeVector(0, 0, 0);
```

```
Pb_box = new G4Box("Pb_cal", 10. * cm, 10. * cm, 1. * cm);
```

```
Pb_log = new G4LogicalVolume(Pb_box, Pb_mat, "Pb_cal_log");
```



```
Pb_pvpl = new G4PVPlacement(ZERO_RM, Pb_vect, Pb_log, "Pb_cal_pvpl", world_log, 0, false, 0);
```



## Вложенность объемов

- Все объемы должны быть вложены один в другой  
Перекрытие объемов не допускается!
- В любой модели существует только один “самый верхний объем” (экспериментальный зал), в который “вкладываются” все остальные
- Дочерний объем позиционируется в локальной системе координат, связанной с родительским объемом. Положение любого объекта (объема, частицы и т.д.) одновременно вычисляется как в глобальной координатной системе, связанной с экспериментальным залом, так и в локальной, связанной с объемом, в котором объект в данный момент находится

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

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

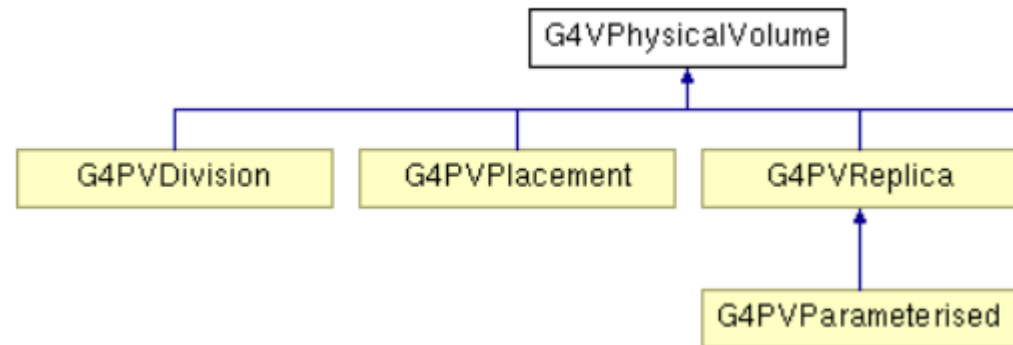
единственный необходимый для реализации геометрии метод: `virtual G4VPhysicalVolume* Construct() = 0;`

Этот метод является чисто виртуальным в наследуемом классе `G4VUserDetectorConstruction` и возвращает указатель на физический объём. Этот указатель должен указывать на материнский объём с привязанными к нему всеми дочерними объёмами, используемыми в геометрии.

```
G4VPhysicalVolume* Geometry::Construct()
{
...
...
...
return new G4PVPlacement(0, G4ThreeVector(), world_log, "world_pvpl", 0, false, 0);
}
```

# Физический объем

- Строится на основе логического объема
- Описывает положение объема в пространстве
- Позволяет одновременно описать серию одинаковых объемов или параметризовать свойства объема в зависимости от номера копии





**G4PVPlacement** = один объем

Единственная копия данного объема размещается в материнском объеме

**G4PVReplica** = одновременное описание нескольких объемов

Материнский объем заполняется одинаковыми дочерними объемами

**G4AssemblyVolume**

одновременно размещается несколько не вложенных друг в друга объемов, которые ведут себя как единое целое при геометрических преобразованиях (поворотах и т.д.)

**G4PVParameterised** = одновременное описание нескольких объемов

Возможна параметризация формы, размеров, материала, положения и поворотов в пространстве в зависимости от номера копии

Специальный тип G4PVParameterised: **G4PVDivision**

# G4PVReplica (копирование)

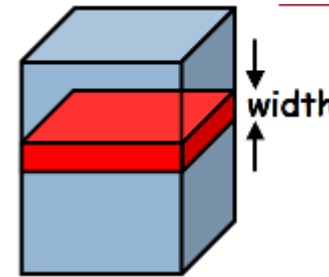
Конструктор физического  
объёма с копиями

```
G4PVReplica( const G4String&      pName,  
              G4LogicalVolume*    pCurrentLogical,  
              G4LogicalVolume*    pMotherLogical,  
              const EAxis          pAxis,  
              const G4int          nReplicas,  
              const G4double       width,  
              const G4double       offset=0 )
```

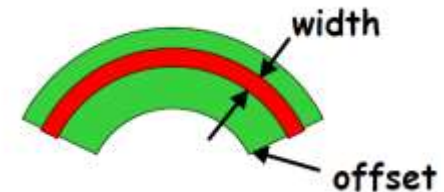
Материнский объём полностью заполняется копиями дочернего объёма той же формы.

Разбиение материнского объёма на копии дочернего объёма может реализовываться:

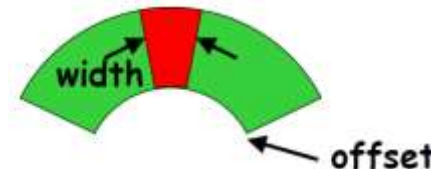
1. Вдоль одной из координатных осей X, Y, Z.



2. Вдоль радиального направления ().



3. По азимутальному углу.



# G4PVReplica

## Особенности и ограничения

- ☐ Смещение «offset» реализуется только для заполнения цилиндра (G4Tubs) и конуса (G4Cons).
- ☐ Внутри любой копии может быть реализовано своё разбиение.
- ☐ Внутри любой копии могут располагаться обычные объёмы при условии отсутствия пересечений с материнским объёмом или другими копиями.
- ☐ При радиальном разбиении внутри копии не могут располагаться другие объёмы.
- ☐ Внутри любой копии не могут располагаться параметризованные объёмы.

# проект Detector\_Replica

Detector\_Replica.cc

DataWriter.cc (DataWriter.hh)

Loader.cc (DataWriter.hh)

Geometry.cc (Geometry.hh)

Action.cc (Action.hh)

PrimaryPart.cc (PrimaryPart.hh)

RunAct.cc (RunAct.hh)

EventAct.cc (EventAct.hh)

~~TrackAct.cc (TrackAct.hh)~~

~~StackAct.cc (StackAct.hh)~~

StepAct.cc (StepAct.hh)

# G4PVReplica (проект Detector\_Replica)

## 1. Разбиение вдоль одной из координатных осей X, Y, Z (kXAxis, kYAxis, kZAxis).

Локальная система координат располагается в центре каждой копии.

Смещение «offset» **НЕ РЕАЛИЗУЕТСЯ**.

Расположение центра каждой копии:

`-width* (nReplicas-1) *0.5+n*width`

`nReplicas` - число копий;    `n` - номер копии;

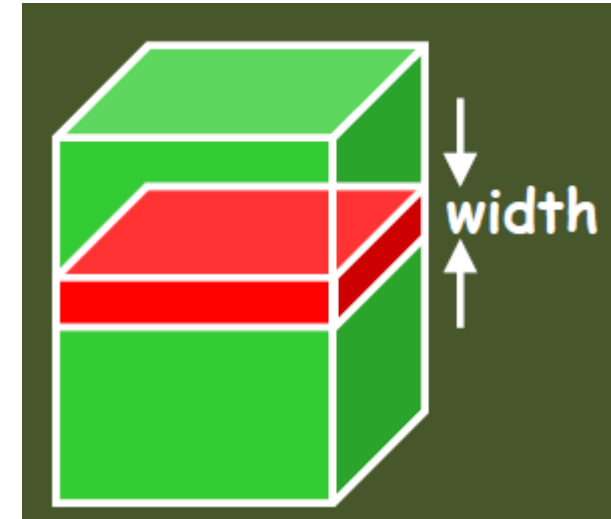
Пример определения материнского объёма (Geometry.cc)

```
Si_box = new G4Box("Si_vol", 10. * cm, 1. * cm, 1. * cm);
```

```
Si_log = new G4LogicalVolume(Si_box, world_mat,"Si_vol_log");
```

```
Si_vect = G4ThreeVector(0, 0, 2.*cm);
```

```
Si_pvpl = new G4PVPlacement(ZERO_RM, Si_vect, Si_log, "Si_vol_pvpl", world_log, 0, false, 0);
```



Пример определения дочернего объёма и размещения его 10 копий вдоль оси X в материнском объёме (Geometry.cc)

```
Si_det_box = new G4Box("Si_det_vol", 1. * cm, 1. * cm, 1. * cm);
```

```
Si_det_log = new G4LogicalVolume(Si_det_box, Si_mat,"Si_det_vol_log");
```

```
Si_det_pvpl= new G4PVReplica("Si_det_vol_pvpl", Si_det_log, Si_log, kXAxis, 10, 2.*cm);
```



# G4PVReplica

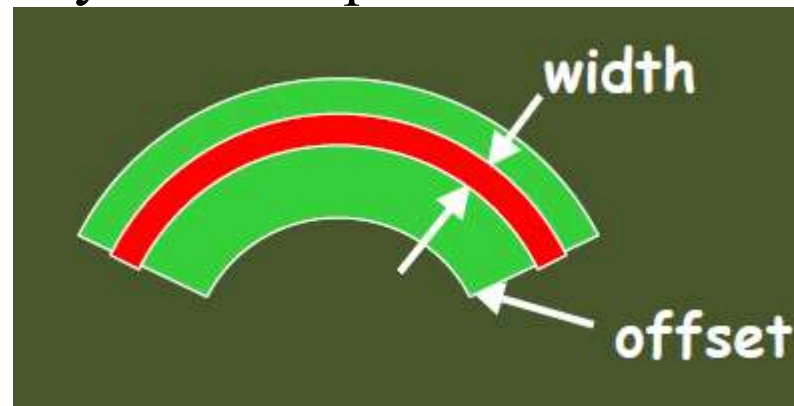
## 2. Разбиение вдоль радиального направления $kRAxis$ (секции цилиндра, конуса).

Локальная система координат совпадает с системой координат материнского объёма.

Смещение «offset» должно совпадать с внутренним радиусом материнского объёма.

Расположение (радиальное) центра каждой копии:

$$width * (n + 0.5) + offset \quad n - \text{номер копии};$$



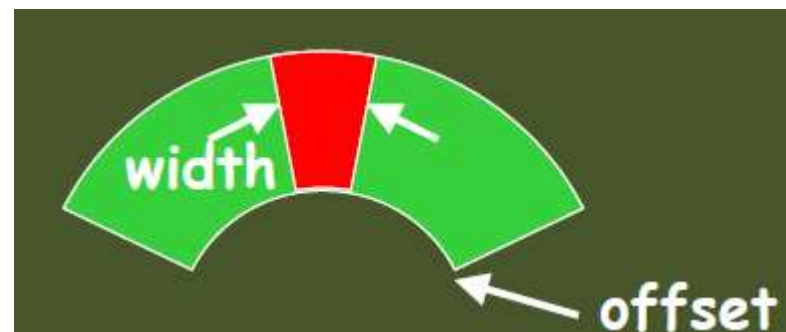
## 2. Разбиение по азимутальному углу $kPhi$ (угловые секции цилиндра, клиновидные секции конуса).

Локальная система координат вращается в соответствии с направлением биссектрисы угла, характеризующего угловой размер каждой копии.

Смещение «offset» должно совпадать с начальным азимутальным углом материнского объёма.

Расположение (угловое) центра каждой копии:

$$width * (n + 0.5) + offset \quad n - \text{номер копии};$$

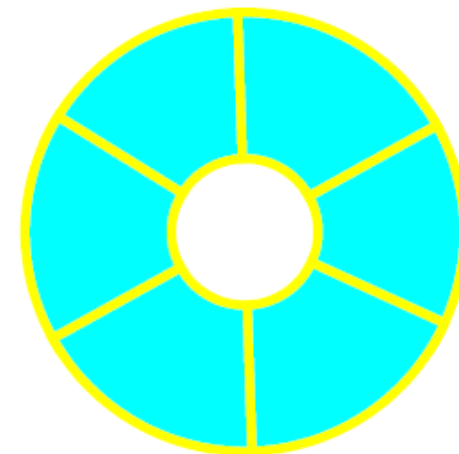


# G4PVReplica

Пример разбиения по азимутальному углу

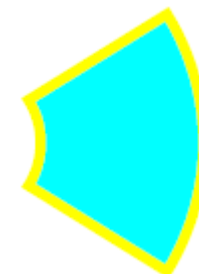
Описание материнского объёма

```
G4double tube_dPhi = 2.* M_PI * rad;  
G4VSolid* tube = new G4Tubs("tube",20*cm,50*cm,30*cm,0.,tube_dPhi);  
G4LogicalVolume* tube_log = new G4LogicalVolume(tube, Air, "tubeL", 0, 0, 0);  
G4VPhysicalVolume* tube_phys = new G4PVPlacement(0,G4ThreeVector(-200.*cm,0.,0.),  
"tubeP", tube_log, world_phys, false, 0);
```



Описание дочернего объёма и размещение его 6 копий в материнском объёме с использованием разбиения по азимутальному углу

```
G4double divided_tube_dPhi = tube_dPhi/6.;  
G4VSolid* div_tube = new G4Tubs("div_tube", 20*cm, 50*cm, 30*cm,  
-divided_tube_dPhi/2., divided_tube_dPhi);  
G4LogicalVolume* div_tube_log = new G4LogicalVolume(div_tube,Pb,"div_tubeL",0,0,0);  
G4VPhysicalVolume* div_tube_phys = new G4PVReplica("div_tube_phys", div_tube_log,  
tube_log, kPhi, 6, divided_tube_dPhi); 20
```



# G4AssemblyVolume (контейнеры для логических объёмов)

При использовании повторяющихся блоков той или иной конструкции можно сгруппировать такие объёмы в общий контейнер, представленный в Geant4 классом **G4AssemblyVolume**.

Контейнер позволяет группировать логические объёмы с целью многократного воспроизведения в мире.

Класс **G4AssemblyVolume** имеет два конструктора:

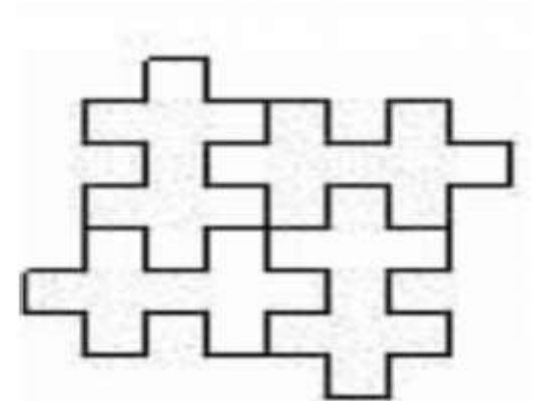
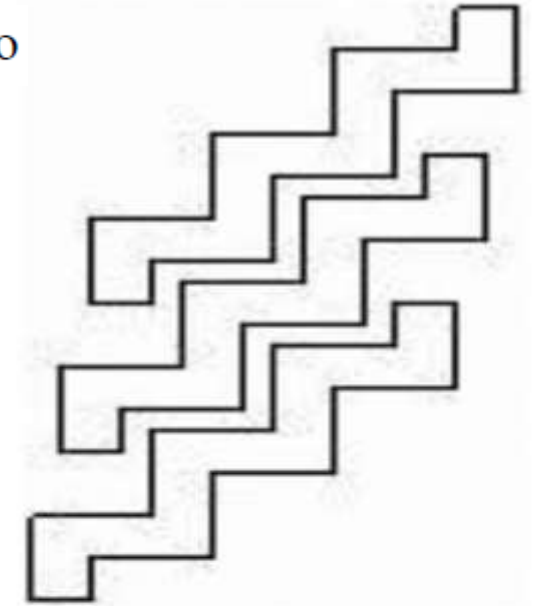
Параметризованный

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

И конструктор «по умолчанию» **G4AssemblyVolume()**

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

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



# G4AssemblyVolume

Существует метод на добавление логического объема в уже существующий контейнер.

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

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

Для размещения контейнера в материнском мире следует воспользоваться методом:

```
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
```



# G4AssemblyVolume (проект AssemblyHit)

Класс

Метод

Geometry.cc → Geometry : public G4VUserDetectorConstruction → G4VPhysicalVolume\* Geometry::Construct()

Создание логического объёма для размещения в контейнере логических объёмов

```
Si_mat=nist->FindOrBuildMaterial("G4_Si");  
Si_det_box = new G4Box("Si_det_vol", 1. * cm, 1. * cm, 1. * cm);  
Si_det_log1 = new G4LogicalVolume(Si_det_box, Si_mat,"Si_det_vol_log1");
```

Создание пустого контейнера логических объёмов, его заполнение

```
assemblyDetector = new G4AssemblyVolume();  
G4RotationMatrix* assembly_RM = new G4RotationMatrix(0, 0, 0);  
G4ThreeVector assembly_vector = G4ThreeVector(0, 0, 0.*cm);  
assemblyDetector->AddPlacedVolume(Si_det_log1, assembly_vector, assembly_RM);  
G4RotationMatrix* assembly_RM1 = new G4RotationMatrix(0, 0, 0);  
assembly_RM1->rotateX(45.*deg); assembly_RM1->rotateY(45.*deg); assembly_RM1->rotateZ(45.*deg);  
assembly_vector.setZ(3.*cm);  
assemblyDetector->AddPlacedVolume(Si_det_log1, assembly_vector, assembly_RM1);  
G4RotationMatrix* assembly_RM2 = new G4RotationMatrix(0, 0, 0);  
assembly_RM2->rotateX(10.*deg); assembly_RM2->rotateY(20.*deg); assembly_RM2->rotateZ(30.*deg);  
assembly_vector.setZ(6.*cm);  
assemblyDetector->AddPlacedVolume(Si_det_log1, assembly_vector, assembly_RM2);
```



# G4AssemblyVolume (проект AssemblyHit)

## Размещение заполненного контейнера логических объёмов в материнском объёме

```
G4RotationMatrix* assembly_RM = new G4RotationMatrix(0, 0, 0);  
G4ThreeVector assembly_vector = G4ThreeVector(0, 0, 0.*cm);  
assembly_vector.setZ(5.*cm);  
assemblyDetector->MakeImprint(world_log, assembly_vector, assembly_RM);
```

Физические объёмы генерируются при вызове метода **MakeImprint**.  
Имена объёмов генерируются в следующем формате:

```
av_WWW_impr_XXX_YYY_ZZZ
```

WWW — номер контейнера

XXX — номер размещения контейнера

YYY — имя размещаемого логического объёма

ZZZ — индекс логического объёма в контейнере (порядковый номер заполнения контейнера)

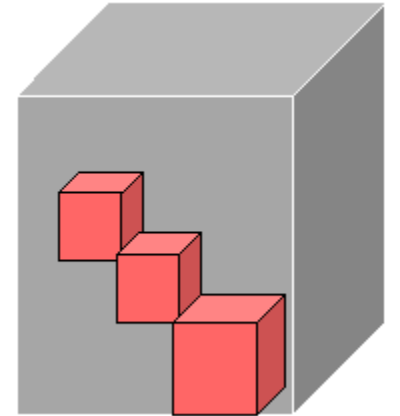
# Параметризация физического объема

Параметризованные объёмы – это возможность размещения копий объёмов, которые могут различаться по размерам, форме или материалам.

Пользователь должен определить материнский объём, в котором будут располагаться параметризованные объёмы.

Мир **не может** быть параметризован.

Материнский объём должен быть определён логическим или физическим.



Форма, размер, материал и положение копий в материнском объёме параметризуются как функция номера копии.

Пользователь должен написать свой класс – наследник `G4VPVParameterisation` и определить в нём необходимые свойства объёма.

Заданная пользователем параметризация используется при создании физических объёмов во время выполнения программы.

# Параметризация физического объема

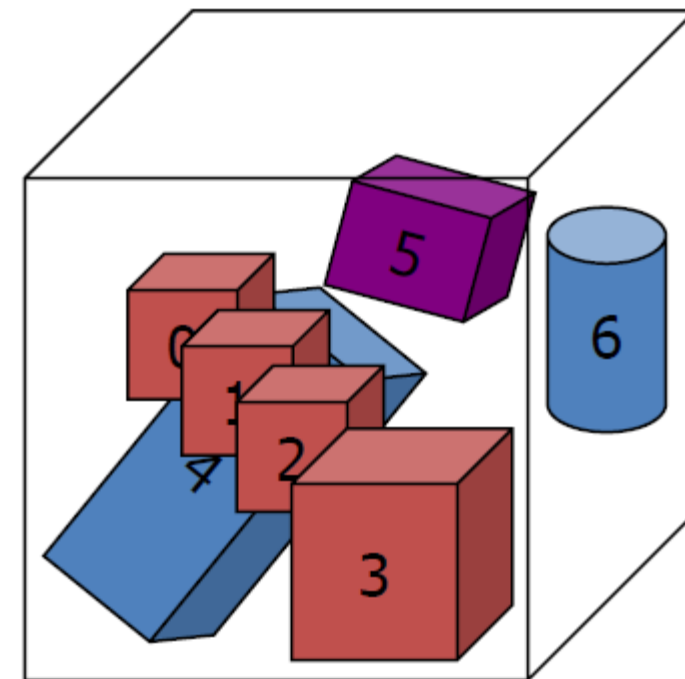
Конструктор физического объема с параметризованными копиями

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

Одномерная параметризация возможна вдоль одной из координатных осей X, Y, Z (kXAxis, kYAxis, kZAxis).

Трёхмерная параметризация задаётся при kUndefined.

Копии размещённых объёмов не должны пересекаться с материнским объёмом и друг с другом



# Параметризация физического объема

## Ограничения:

- Можно параметризовать только простые формы

Положение копии в материнском объёме (чисто виртуальный метод)

### Public Member Functions

	<code>G4VPVParameterisation ()</code>
virtual	<code>~G4VPVParameterisation ()</code>
virtual void	<code>ComputeTransformation (const G4int, G4VPhysicalVolume *) const =0</code>
virtual G4VSolid *	<code>ComputeSolid (const G4int, G4VPhysicalVolume *)</code>
virtual G4Material *	<code>ComputeMaterial (const G4int repNo, G4VPhysicalVolume *currentVol, const G4VTouchable *parentTouch=0)</code>
virtual G4bool	<code>IsNested () const</code>
virtual G4VVolumeMaterialScanner *	<code>GetMaterialScanner ()</code>
virtual void	<code>ComputeDimensions (G4Box &amp;, const G4int, const G4VPhysicalVolume *) const</code>
virtual void	<code>ComputeDimensions (G4Tubs &amp;, const G4int, const G4VPhysicalVolume *) const</code>
virtual void	<code>ComputeDimensions (G4Trd &amp;, const G4int, const G4VPhysicalVolume *) const</code>
virtual void	<code>ComputeDimensions (G4Trap &amp;, const G4int, const G4VPhysicalVolume *) const</code>
virtual void	<code>ComputeDimensions (G4Cons &amp;, const G4int, const G4VPhysicalVolume *) const</code>
virtual void	<code>ComputeDimensions (G4Sphere &amp;, const G4int, const G4VPhysicalVolume *) const</code>
virtual void	<code>ComputeDimensions (G4Orb &amp;, const G4int, const G4VPhysicalVolume *) const</code>
virtual void	<code>ComputeDimensions (G4Torus &amp;, const G4int, const G4VPhysicalVolume *) const</code>
virtual void	<code>ComputeDimensions (G4Para &amp;, const G4int, const G4VPhysicalVolume *) const</code>
virtual void	<code>ComputeDimensions (G4Polycone &amp;, const G4int, const G4VPhysicalVolume *) const</code>
virtual void	<code>ComputeDimensions (G4Polyhedra &amp;, const G4int, const G4VPhysicalVolume *) const</code>
virtual void	<code>ComputeDimensions (G4Hype &amp;, const G4int, const G4VPhysicalVolume *) const</code>

Форма

Материал

Размер

Константные функции не могут изменять объект, на котором вызывается, а также модифицировать нестатические элементы данных или вызывать какие-либо функции-члены, которые не являются константами.

# проект Detector\_Parameterisation

Detector\_Replica.cc

DataWriter.cc (DataWriter.hh)

Loader.cc (DataWriter.hh)

Geometry.cc (Geometry.hh)

Action.cc (Action.hh)

PrimaryPart.cc (PrimaryPart.hh)

RunAct.cc (RunAct.hh)

EventAct.cc (EventAct.hh)

StepAct.cc (StepAct.hh)

Si\_det\_Parameterisation.cc  
(Si\_det\_Parameterisation.hh)





# Параметризация физического объема

ПРОЕКТ “Detector\_Parameterisation”.

## Материнский объём

```
Si_det_box2 = new G4Box("Si_det_box2", 1. * cm, 1. * cm, 10. * cm);  
Si_det_log2 = new G4LogicalVolume(Si_det_box2, Si_mat, "Si_det_log2");
```

## Параметризуемый объём

```
small_det_box = new G4Box("small_det_box", 1. * cm, 1. * cm, 1. * cm);  
small_det_log = new G4LogicalVolume(small_det_box, Si_mat, "small_det_log");
```

## Пользовательский класс с методами:

```
ComputeDimensions, ComputeTransformation, ComputeSolid, ComputeMaterial  
G4VPVParameterisation* Si_det = new Si_det_Parameterisation();
```

## Размещение параметризованных копий объёма в материнском объёме

```
small_det_pvpl= new G4PVParameterised("small_det_pvpl", small_det_log, Si_det_log2, kZAxis, 10, Si_det);
```

# Параметризация физического объема

Параметризация положения копий в материнском объёме.

```
void Si_det_Parameterisation::ComputeTransformation (const G4int copyNo, G4VPhysicalVolume* physVol) const
{
    G4double Xposition= 0.* cm;
    G4double Yposition= 0.* cm;
    G4double Zposition= 0.* cm;
    Zposition = -9.*cm + (2.*cm)*copyNo;
    G4ThreeVector origin(Xposition,Yposition,Zposition);
    physVol->SetTranslation(origin);
    physVol->SetRotation(0);
}
```

Параметризация размера.

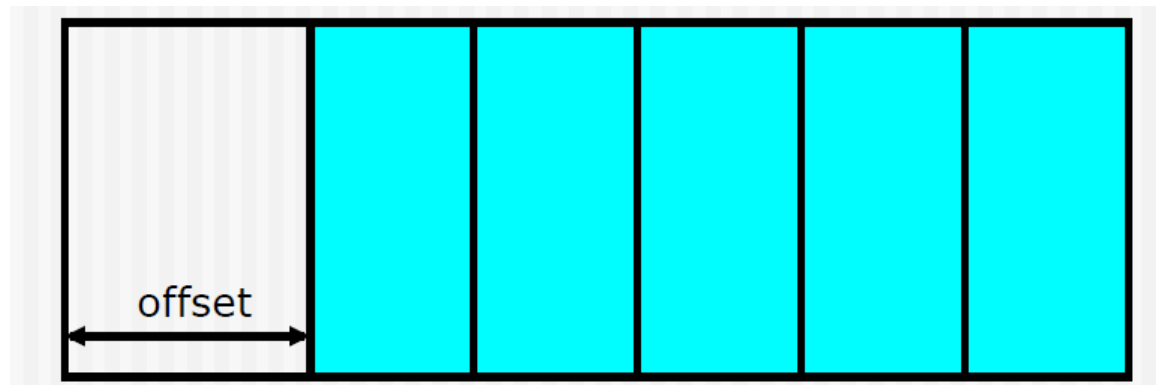
```
void Si_det_Parameterisation::ComputeDimensions (G4Box& Si_det_box, const G4int, const G4VPhysicalVolume*) const
{
    G4double XhalfLength = 1.* cm;
    G4double YhalfLength = 1 * cm;
    G4double ZhalfLength = 1.* cm;
    Si_det_box.SetXHalfLength(XhalfLength);
    Si_det_box.SetYHalfLength(YhalfLength);
    Si_det_box.SetZHalfLength(ZhalfLength);
}
```

# G4PVDivision

Вариант 1 конструктора  
физического объёма с  
копиями

```
G4PVDivision(const G4String& pName,  
             G4LogicalVolume* pDaughterLogical,  
             G4LogicalVolume* pMotherLogical,  
             const EAxis pAxis,  
             const G4int nDivisions,    // number of division is given  
             const G4double offset);
```

Размер размещаемых копий вычисляется как:  $\frac{\text{размер материнского объёма} - \text{смещение (offset)}}{\text{число копий}}$

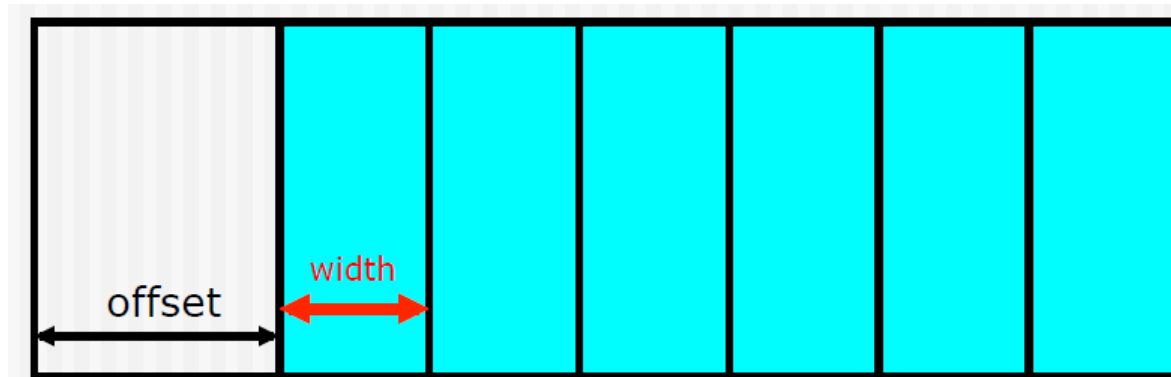


# G4PVDivision

Вариант 2 конструктора  
физического объёма с  
копиями

```
G4PVDivision(const G4String& pName,  
             G4LogicalVolume* pDaughterLogical,  
             G4LogicalVolume* pMotherLogical,  
             const EAxis pAxis,  
             const G4double width, // width of daughter volume is given  
             const G4double offset);
```

Число размещаемых копий вычисляется как:  $\text{int}(\frac{\text{размер материнского объёма} - \text{смещение (offset)}}{\text{число копий}})$



# G4PVDivision

```
G4PVDivision(const G4String& pName,  
             G4LogicalVolume* pDaughterLogical,  
             G4LogicalVolume* pMotherLogical,  
             const EAxis pAxis,  
             const G4int nDivisions,  
             const G4double width, // both number of division and width are given  
             const G4double offset);
```

Вариант 3 конструктора  
физического объёма с  
копиями

nDivisions копий с толщиной width





# G4PVDivision

Размещение копий в материнском объёме с использованием GPVDivision во многом совпадает с размещением через G4PVReplica.

- ✓ Копирование возможно только вдоль одной оси (**kXAxis**, **kYAxis**, **kZAxis**) или вдоль радиального направления (**kRho**) или по азимутальному углу (**kPhi**).
- ✓ Форма размещаемых копий должна совпадать с формой материнского объёма.

Отличия от использования G4PVReplica:

- Возможны зазоры между материнским и дочерними объёмами

При инициализации физического объёма через GPVDivision автоматически рассчитывается параметризация.

Поэтому размеры копии, заданные для соответствующего логического объёма, вычисляются снова в методе `G4VParameterisation::ComputeDimension()` в соответствии с параметрами конструктора.

# G4PVDivision

ПРОЕКТ “Division”.

## Материнский объём

```
world_mat = nist->FindOrBuildMaterial("G4_AIR");  
Si_box = new G4Box("Si_vol", 10. * cm, 1. * cm, 1. * cm);  
Si_log = new G4LogicalVolume(Si_box, world_mat, "Si_vol_log");  
Si_vect = G4ThreeVector(0, 0, 2.*cm);  
Si_pvpl = new G4PVPlacement(ZERO_RM, Si_vect, Si_log, "Si_vol_pvpl", world_log, 0, false, 0);
```

## Параметризуемый объём

```
Si_mat=nist->FindOrBuildMaterial("G4_Si");  
Si_det_box = new G4Box("Si_det_vol", 1. * cm, 1. * cm, 1. * cm);  
Si_det_log = new G4LogicalVolume(Si_det_box, Si_mat, "Si_det_vol_log");
```

## Размещение параметризованных копий объёма в материнском объёме

```
Si_det_pvpl= new G4PVDivision("Si_det_vol_pvpl", Si_det_log, Si_log, kXAxis, 5, 0); //Вариант 1  
//Si_det_pvpl= new G4PVDivision("Si_det_vol_pvpl", Si_det_log, Si_log, kXAxis, 5, 1.*cm, 0); //Вариант 2  
//Si_det_pvpl= new G4PVDivision("Si_det_vol_pvpl", Si_det_log, Si_log, kXAxis, 5, 1.*cm, 3.*cm); // Вариант 3
```

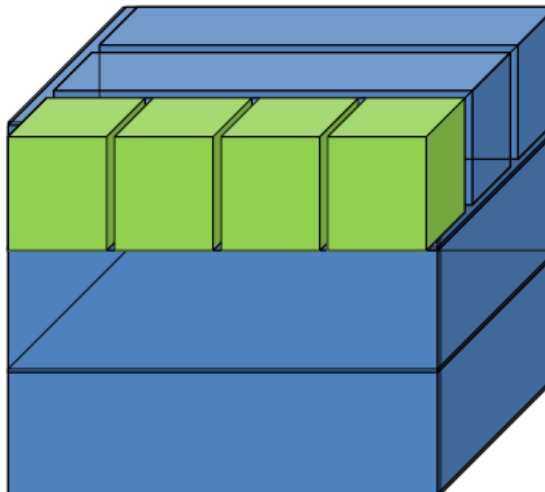
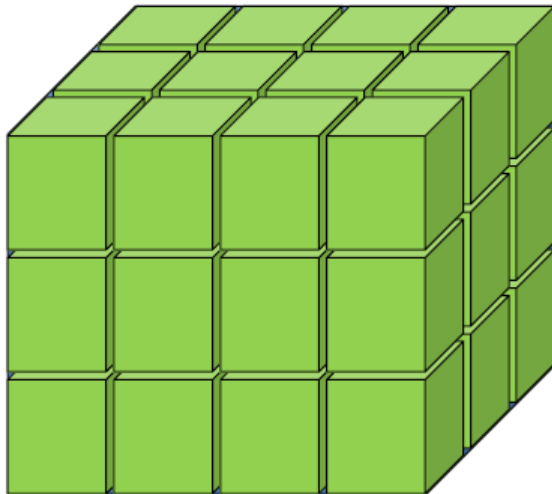
# Nested parameterization (вложенная параметризация)

При моделировании экспериментальных установок ячеистой структуры, состоящих из одинаковых детекторов без промежутков между ними, можно вместо трёхмерной параметризации G4PVParameterised использовать специальную параметризацию **G4VNestedParameterisation**.

Класс **G4VNestedParameterisation** является наследником класса G4VPVParameterization

При таком подходе вдоль двух координатных осей реализуется копирование с использованием G4PVReplica, а вдоль координатной третьей оси реализуется одномерная параметризация.

В результате оптимизируется использование памяти и осуществляется более быстрая навигация по ячейкам созданной структуры, что особенно заметно при возрастания числа детекторов.



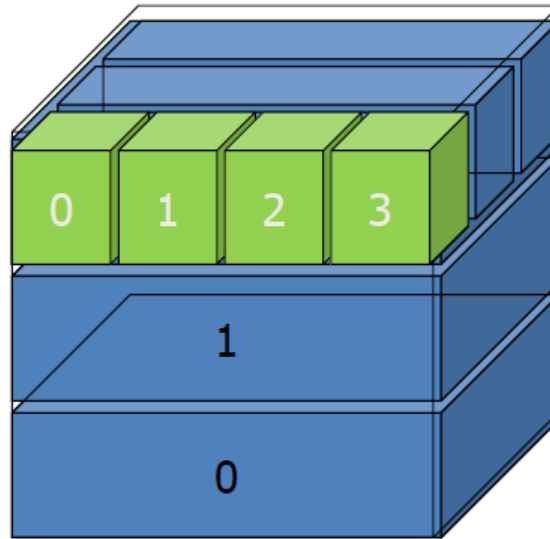
# Nested parameterization

## Public Member Functions

```
G4VNestedParameterisation ()
virtual ~G4VNestedParameterisation ()
virtual G4Material * ComputeMaterial (G4VPhysicalVolume *currentVol, const G4int repNo, const G4VTouchable *parentTouch=0)=0
virtual G4int GetNumberOfMaterials () const =0
virtual G4Material * GetMaterial (G4int idx) const =0
virtual void ComputeTransformation (const G4int no, G4VPhysicalVolume *currentPV) const =0
virtual G4VSolid * ComputeSolid (const G4int no, G4VPhysicalVolume *thisVol)
virtual void ComputeDimensions (G4Box &, const G4int, const G4VPhysicalVolume *) const
virtual void ComputeDimensions (G4Tubs &, const G4int, const G4VPhysicalVolume *) const
virtual void ComputeDimensions (G4Trd &, const G4int, const G4VPhysicalVolume *) const
virtual void ComputeDimensions (G4Trap &, const G4int, const G4VPhysicalVolume *) const
virtual void ComputeDimensions (G4Cons &, const G4int, const G4VPhysicalVolume *) const
virtual void ComputeDimensions (G4Sphere &, const G4int, const G4VPhysicalVolume *) const
virtual void ComputeDimensions (G4Orb &, const G4int, const G4VPhysicalVolume *) const
virtual void ComputeDimensions (G4Torus &, const G4int, const G4VPhysicalVolume *) const
virtual void ComputeDimensions (G4Para &, const G4int, const G4VPhysicalVolume *) const
virtual void ComputeDimensions (G4Polycone &, const G4int, const G4VPhysicalVolume *) const
virtual void ComputeDimensions (G4Polyhedra &, const G4int, const G4VPhysicalVolume *) const
virtual void ComputeDimensions (G4Hype &, const G4int, const G4VPhysicalVolume *) const
G4Material * ComputeMaterial (const G4int repNo, G4VPhysicalVolume *currentVol, const G4VTouchable *parentTouch=0)
virtual G4bool IsNested () const
virtual G4VVolumeMaterialScanner * GetMaterialScanner ()
```

Чисто виртуальные методы  
класса  
G4VNestedParamentrisation  
должны быть реализованы  
в классе - наследнике

# Nested parameterization



ComputeMaterials возвращает указатель на материал данного детектора, который идентифицируется своим номером копии и двумя номерами родительских копий.

Индекс копии вдоль первой оси определяется как:  $\text{parentTouch} \rightarrow \text{GetCopyNumber}(1)$ ;

вдоль второй оси:  $\text{parentTouch} \rightarrow \text{GetCopyNumber}(0)$ ;

вдоль третьей оси:  $\text{repNo}$ .

GetNumberOfMaterial возвращает значение, которое интерпретируется как полное число используемых материалов  $n\text{Material}$ .

GetMaterial может возвращать индекс материала  $\text{idx}$  в диапазоне  $[0, n\text{Material}-1]$ .



# Nested parameterization

Объект класса `G4VNestedParameterisation` может использоваться как аргумент конструктора `G4PVParameterised`.

Вложенная параметризация уже существующих физических объёмов не поддерживается.

При создании вложенной параметризации все используемые объёмы должны быть повторяющимися копиями вдоль соответствующего направления.

Все используемые объёмы **НЕ ДОЛЖНЫ** быть размещены как физические.

# Nested parameterization

ПРОЕКТ “Detector\_Parameterisation3D”.

## Материнский объём

```
world_mat = nist->FindOrBuildMaterial("G4_AIR");  
Si_box = new G4Box("Si_vol", 10. * cm, 10. * cm, 10. * cm);  
Si_log = new G4LogicalVolume(Si_box, world_mat,"Si_vol_log");  
Si_vect = G4ThreeVector(0, 0, 10.*cm);  
new G4PVPlacement(ZERO_RM, Si_vect, Si_log, "Si_vol_pvpl", world_log, 0, false, 0);
```

## Создание копий вдоль оси X

```
Si_mat=nist->FindOrBuildMaterial("G4_Si");  
Si_det_box = new G4Box("Si_det_box", 1. * cm, 10. * cm, 10. * cm);  
Si_det_log = new G4LogicalVolume(Si_det_box, Si_mat,"Si_det_log");  
new G4PVReplica("Si_det_pvpl", Si_det_log, Si_log, kXAxis, 10, 2.*cm);
```

## Создание копий вдоль оси Y

```
Si_det_box2 = new G4Box("Si_det_box2", 1. * cm, 1. * cm, 10. * cm);  
Si_det_log2 = new G4LogicalVolume(Si_det_box2, Si_mat,"Si_det_log2");  
new G4PVReplica("Si_det_pvpl2", Si_det_log2, Si_det_log, kYAxis, 10, 2.*cm);
```

Память выделена, объём не размещён

# Nested parameterization

Создание объёма для копирования вдоль оси Z

```
small_det_box = new G4Box("small_det_box", 1. * cm, 1. * cm, 1. * cm);  
small_det_log = new G4LogicalVolume(small_det_box, Si_mat, "small_det_log");
```

Размещение параметризованных копий объёма в материнском объёме. Вложенная параметризация.

```
Si_det_Parameterisation* Si_det = new Si_det_Parameterisation();  
new G4PVParameterised("small_det_box", small_det_log, Si_det_log2, kZAxis, 10, Si_det);
```

Параметризация положения копий в материнском объёме.

```
void Si_det_Parameterisation::ComputeTransformation (const G4int copyNo, G4VPhysicalVolume* physVol) const  
{  
    G4double Xposition= 0.* cm;  
    G4double Yposition= 0.* cm;  
    G4double Zposition= 0.* cm;  
    Zposition = -9.*cm + (2.*cm)*copyNo; G4ThreeVector origin(Xposition,Yposition,Zposition);  
    physVol->SetTranslation(origin); physVol->SetRotation(0);  
}
```

# Nested parameterization

## Параметризация размера.

```
void Si_det_Parameterisation::ComputeDimensions (G4Box& Si_det_box, const G4int, const G4VPhysicalVolume*) const
{
    G4double XhalfLength = 1.* cm;
    G4double YhalfLength = 1 * cm;
    G4double ZhalfLength = 1.* cm;
    Si_det_box.SetXHalfLength(XhalfLength);
    Si_det_box.SetYHalfLength(YhalfLength);
    Si_det_box.SetZHalfLength(ZhalfLength);
}
```

## Параметризация материала ([заглушка](#)).

```
G4Material* Si_det_Parameterisation::ComputeMaterial (G4VPhysicalVolume* physVol, const G4int copyNo, const
G4VTouchable *parentTouch)
{
    G4Material* mat;
    G4NistManager* nist;
    nist = G4NistManager::Instance();
    mat=nist->FindOrBuildMaterial("G4_Si");
    return mat;
}
```

# Nested parameterization

Параметризация положения копий в материнском объёме.

```
void Si_det_Parameterisation::ComputeTransformation (const G4int copyNo, G4VPhysicalVolume* physVol) const
{
    G4double Xposition= 0.* cm;
    G4double Yposition= 0.* cm;
    G4double Zposition= 0.* cm;
    Zposition = -9.*cm + (2.*cm)*copyNo;
    G4ThreeVector origin(Xposition,Yposition,Zposition);
    physVol->SetTranslation(origin);
    physVol->SetRotation(0);
}
```

Полное число материалов (заглушка).

```
G4int Si_det_Parameterisation::GetNumberOfMaterials() const
{
    return 1;
}
```

Индекс материала (заглушка).

```
G4Material* Si_det_Parameterisation::GetMaterial(G4int num) const
{
    G4Material* mat;
    G4NistManager* nist; nist = G4NistManager::Instance();
    mat=nist->FindOrBuildMaterial("G4_Si");
    return mat;
}
```



# Визуализация детектора и событий

В Geant4 для упрощения процесса отладки, а также с целью большей наглядности процесса моделирования представлен визуальный режим работы.

## Объекты визуализации

### *Элементы модели*

Объемы детектора или отдельных подсистем

Траектории частиц

Срабатывания в чувствительных объемах

### *Элементы определяемые пользователем*

Линии, например оси координат

Маркеры

Текст

Масштабные линейки

# Визуализация детектора и событий

Своей графической системы в Geant4 нет.

Используется набор драйверов для установленных графических систем.

<i>Драйвер</i>	<i>Графическая система</i>	<i>Платформа</i>
OpenGL-Xlib	OpenGL	Linux, Mac + Xlib
OpenGL-Motif	OpenGL	Linux, Mac + Motif
OpenGL-Win32	OpenGL	Windows
Qt	OpenGL	Linux, Mac, Windows
OpenInventor-X	OpenInventor, OpenGL	Linux, Mac+Xlib или Motif
OpenInventor-Win32	OpenInventor, OpenGL	Windows
HepRep	WIRED (JAS)	Linux, Mac, Windows
DAWNFILE	Fukui Renderer DAWN	Linux, Mac, Windows
DAWN-Network	Fukui Renderer DAWN	Linux
VRMLFILE	просмотр VRML	Linux, Mac, Windows
VRML-Network	просмотр VRML по сети	Linux
RayTracer	просмотр JPEG	Linux, Mac, Windows
ASCIITree	нет	Linux, Mac, Windows
GAGTree	GAG	Linux, Mac, Windows
XMLTree	просмотр XML	Linux, Mac, Windows
gMocrenFile	gMocren	Windows only

# Визуализация детектора и событий

Если в рамках проекта необходимо визуальное представление, то следует создать объект класса G4VisExecutive, наследующий G4VisManager.

```
G4VisManager* visManager = new G4VisExecutive;
```

Для инициализации графической оболочки недостаточно создать объект класса, необходимо вызвать метод инициализации

```
visManager->Initialize();
```

не стоит считать, что сразу после вызова метода Initialize(), появится сцена, содержащая в себе все элементы моделирования.

Создание и настройка параметров сцены управляется командами в файле с расширением «mac» (vis.mac).  
Файл загружается из программного кода в интерактивном режиме взаимодействия пользователя с программой моделирования.

# ***Взаимодействие пользователя с программой моделирования***

Geant4 позволяет использовать команды, позволяющие управлять отдельными этапами моделирования во время работы проекта.

В Geant4 управление пользовательским интерфейсом , а так же обработку всех команд осуществляет класс **G4UImanager**. Пользователь **НЕ ДОЛЖЕН** осуществлять вызов конструктора данного класса или наследовать его.

Вместо это в процессе загрузки необходимо вызвать статический метод данного класса, возвращающий указатель на уже существующий объект данного класса.

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

# ***Взаимодействие пользователя с программой моделирования***

- Пакетный режим
- Пакетный режим, управляемый сценарием
- Интерактивный режим с командной строкой
- Интерактивный режим с графическим интерфейсом

# ***Взаимодействие пользователя с программой моделирования***

- **Пакетный режим**

```
// Initialize G4 kernel  
runManager->Initialize();  
  
// start a run  
int numberOfEvent = 1000;  
runManager->BeamOn(numberOfEvent);  
  
// job termination  
delete runManager;
```



# ***Взаимодействие пользователя с программой моделирования***

- Пакетный режим, управляемый сценарием**

```
runManager->Initialize();
G4UImanager *UImanager =G4UImanager::GetUIpointer();
if (argc != 1)
{
// batch mode
  G4String command = "/control/execute ";
  G4String fileName = argv[1];
  UImanager->ApplyCommand(command + fileName);
}
```

Запуск пакетного режима, управляемого сценарием в файле input.in  
./Detector\_Replica input.in

# ***Взаимодействие пользователя с программой моделирования***

- **Интерактивный режим с командной строкой**

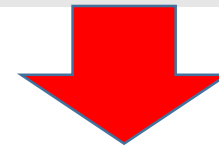
```
// Initialize G4 kernel
runManager->Initialize();

// Define UI terminal for interactive mode
G4UIsession * session = new G4UIterminal;
session->SessionStart();
delete session;

// job termination
delete runManager;
```

# Взаимодействие пользователя с программой моделирования

## • Интерактивный режим с графическим интерфейсом



```
runManager->Initialize();  
// Initialize visualization  
G4VisManager* visManager = new G4VisExecutive;  
visManager->Initialize();  
G4UImanager *UImanager = G4UImanager::GetUIpointer();  
  
if (argc != 1)  
{  
// batch mode  
G4String command = "/control/execute ";  
G4String fileName = argv[1];  
UImanager->ApplyCommand(command + fileName);  
}
```

```
else  
{  
// interactive mode : define UI session  
G4UIExecutive *ui = new G4UIExecutive(argc, argv, "qt");  
  
//Here we can redefine win32/qt interface  
UImanager->ApplyCommand("/control/execute vis.mac");  
  
ui->SessionStart();  
delete ui;  
}
```

Запуск интерактивного режима с графическим интерфейсом  
./Detector\_Replica