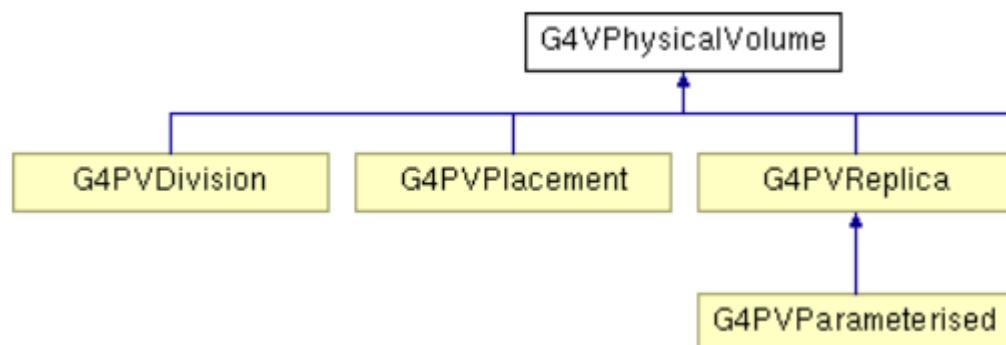


# Физический объем

- 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

**G4PVR** = 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: **G4PVD**

# G4PVReplica (копирование)

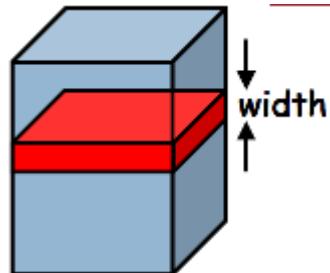
```
G4PVReplica( const G4String& pName,  
              G4LogicalVolume* pCurrentLogical,  
              G4LogicalVolume* pMotherLogical,  
              const EAxis pAxis,  
              const G4int nReplicas,  
              const G4double width,  
              const G4double offset=0 )
```

Конструктор физического объёма с копиями

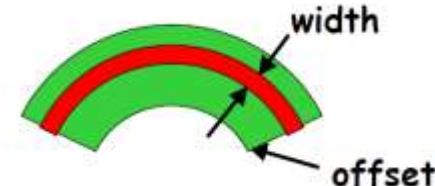
Материнский объём полностью заполняется копиями дочернего объёма той же формы.

Разбиение материнского объёма на копии дочернего объёма может реализовываться:

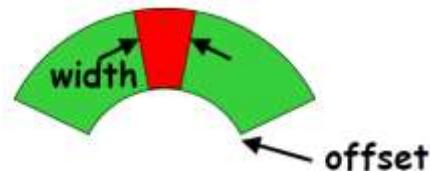
**1. Вдоль одной из координатных осей X, Y, Z.**



**2. Вдоль радиального направления () .**



**3. По азимутальному углу.**

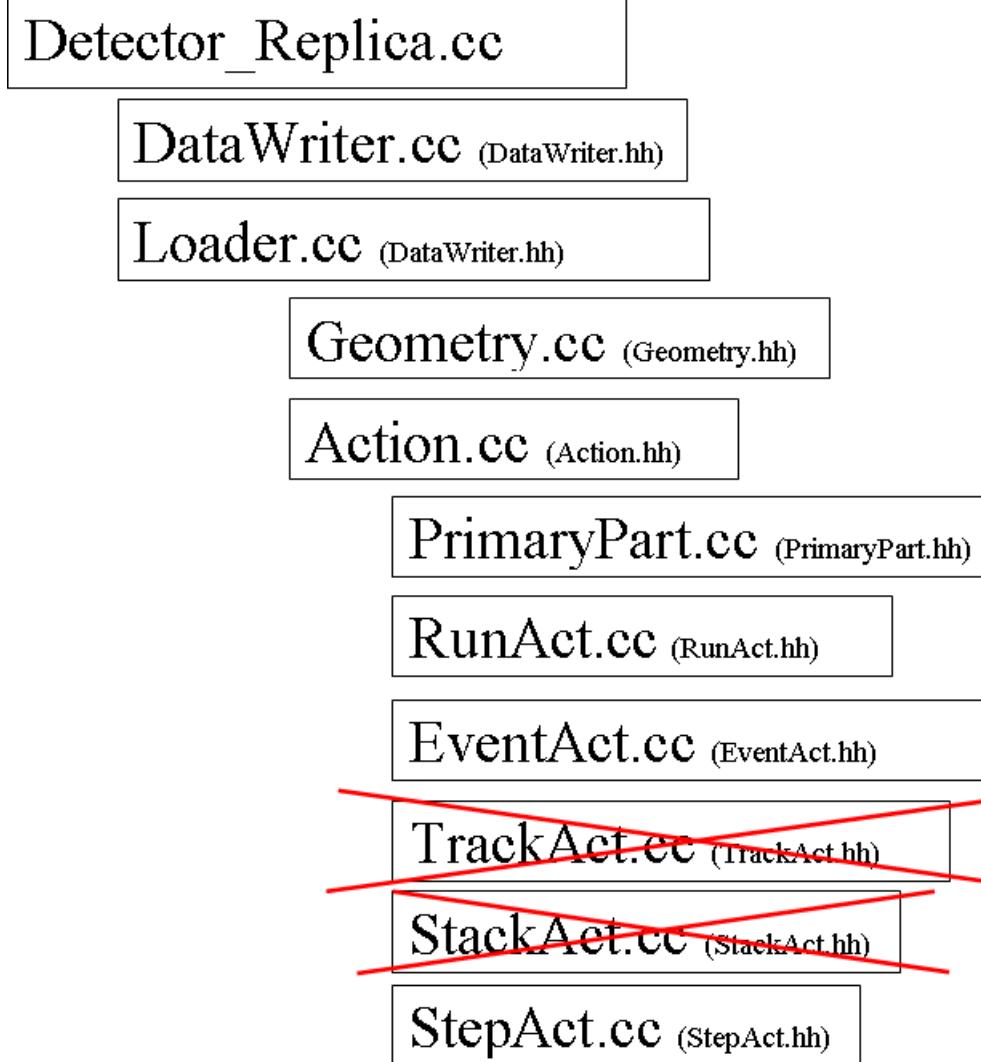


# G4PVReplica

## 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



# 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.

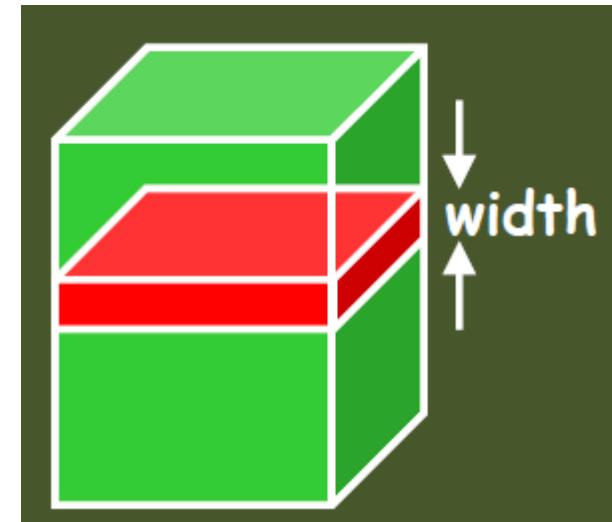
Расположение центра каждой копии:

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

**nReplicas** - число копий; **n** - номер копии; **n**=0, ..., nReplicas-1

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).

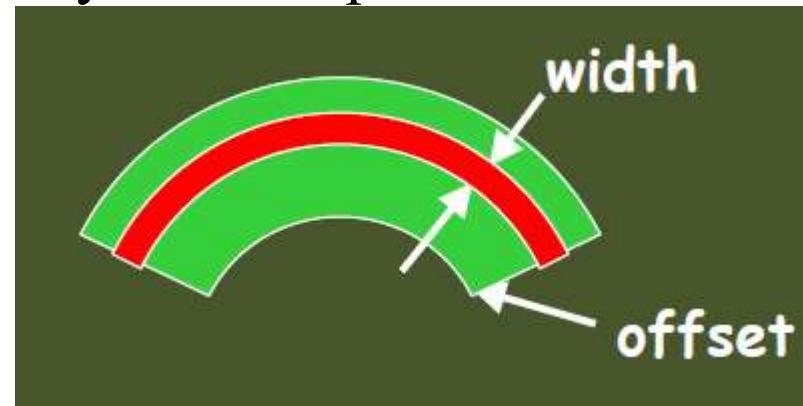
Локальная система координат совпадает с системой координат материнского объёма.

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

Расположение (радиальное) центра каждой копии:

$width * (n + 0.5) + offset$

n - номер копии;



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

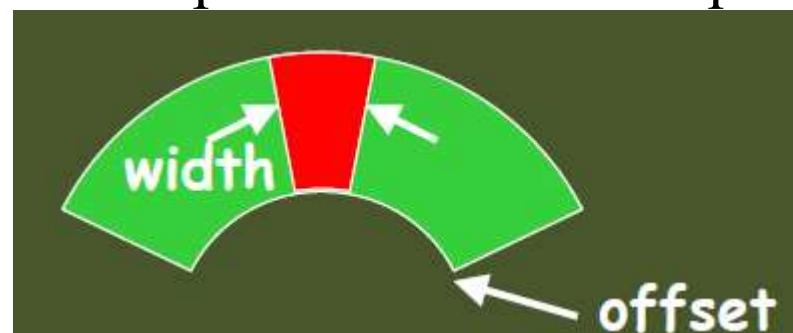
Локальная система координат вращается в соответствии с направлением биссектрисы угла, характеризующего угловой размер каждой копии.

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

Расположение (угловое) центра каждой копии:

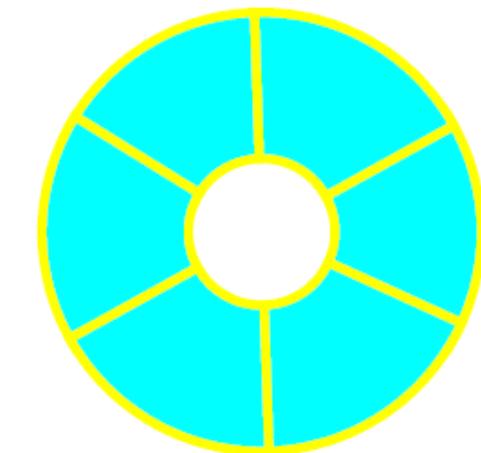
$width * (n + 0.5) + offset$

n - номер копии;



# G4PVReplica

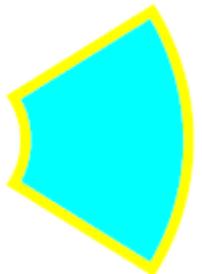
Пример разбиения по азимутальному углу



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);
```

# **G4AssemblyVolume** (контейнеры для логических объемов)

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

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

Класс **G4AssemblyVolume** имеет два конструктора:

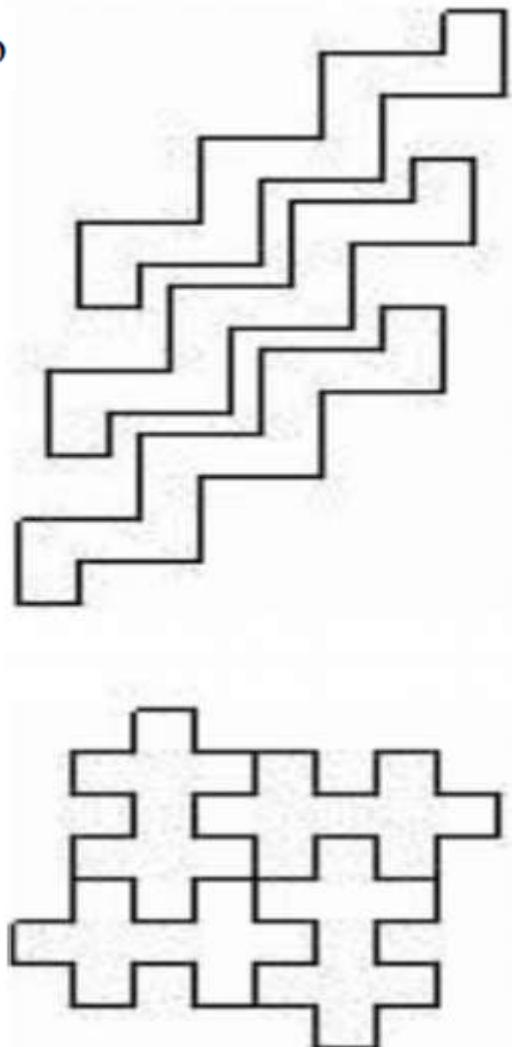
Параметризованный

```
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)

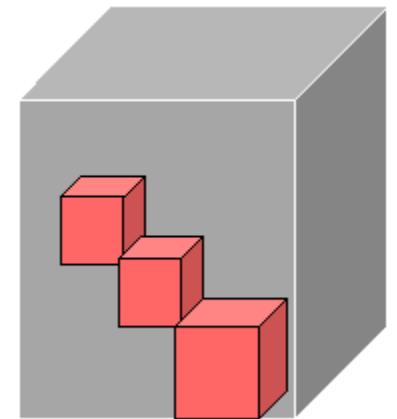
# Параметризация физического объема

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

Пользователь должен определить материнский объем, в котором будут располагаться параметризованные объемы.

Мир **не может** быть параметризован.

Материнский объем должен быть определён логическим или физическим.



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

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

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

# Параметризация физического объема

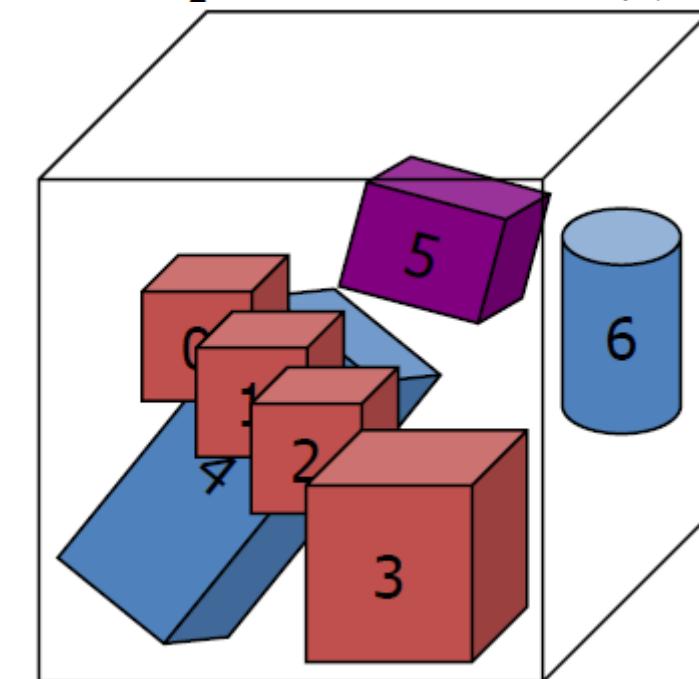
```
G4PVParameterised(const G4String& pName,  
G4LogicalVolume* pLogical,  
G4LogicalVolume* pMother,  
const EAxis pAxis,  
const G4int nReplicas,  
G4VPVParameterisation* pParam  
G4bool pSurfChk=false);
```

Конструктор физического объема с параметризованными копиями

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

Трёхмерная параметризация задаётся при kUndefined.

Копии размещённых объемов не должны пересекаться с материнским объемом и друг с другом



# Параметризация физического объема

## Ограничения:

- Можно параметризовать только простые формы

### Public Member Functions

Положение копии в материнском объёме (чисто виртуальный метод)

```
virtual G4VPParameterisation ()  
virtual ~G4VPParameterisation ()  
virtual void ComputeTransformation (const G4int G4VPhysicalVolume *) const =0  
virtual G4VSolid * ComputeSolid (const G4int, G4VPhysicalVolume *)  
virtual G4Material * ComputeMaterial (const G4int repNo, G4VPhysicalVolume *currentVol, const G4VTouchable *parentTouch=0)  
virtual G4bool IsNested () const  
virtual G4VVolumeMaterialScanner * GetMaterialScanner ()  
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
```

The diagram consists of three red arrows pointing from text labels on the right to specific function names in the code. The first arrow points from 'Форма' to 'ComputeSolid'. The second arrow points from 'Материал' to 'ComputeMaterial'. The third arrow points from 'Размер' to 'ComputeDimensions'.

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

# проект Detector\_Parameterisation

Detector\_Parameterisation.cc

DataWriter.cc (DataWriter.hh)

Loader.cc (DataWriter.hh)

Geometry.cc (Geometry.hh)

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

Action.cc (Action.hh)

PrimaryPart.cc (PrimaryPart.hh)

RunAct.cc (RunAct.hh)

EventAct.cc (EventAct.hh)

StepAct.cc (StepAct.hh)

# Параметризация физического объема

ПРОЕКТ “Detector\_Parameterisation”.

## Материнский объём

```
Si_box = new G4Box("Si_box", 10. * cm, 1. * cm, 1. * cm);  
Si_log = new G4LogicalVolume(Si_box, world_mat,"Si_log");
```

## Параметризуемый объём

```
Si_det_box = new G4Box("Si_det_box", 1. * cm, 1. * cm, 1. * cm);  
Si_det_log = new G4LogicalVolume(Si_det_box, Si_mat, "Si_det_log");
```

## Пользовательский класс с методами:

ComputeDimensions, ComputeTransformation, ComputeSolid, ComputeMaterial

```
G4VPVParameterisation* Si_det = new Si_det_Parameterisation();
```

## Размещение параметризованных копий объема в материнском объеме

```
Si_det_pvpl= new G4PVParameterised("Si_det_pvpl", Si_det_log, Si_log, kXAxis, 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;
    Xposition = -9.*cm + (2.*cm)*copyNo;
    G4ThreeVector origin(Xposition,Yposition,Zposition);
    physVol->SetTranslation(origin);
    physVol->SetRotation(0);
}
```

## Параметризация размера.

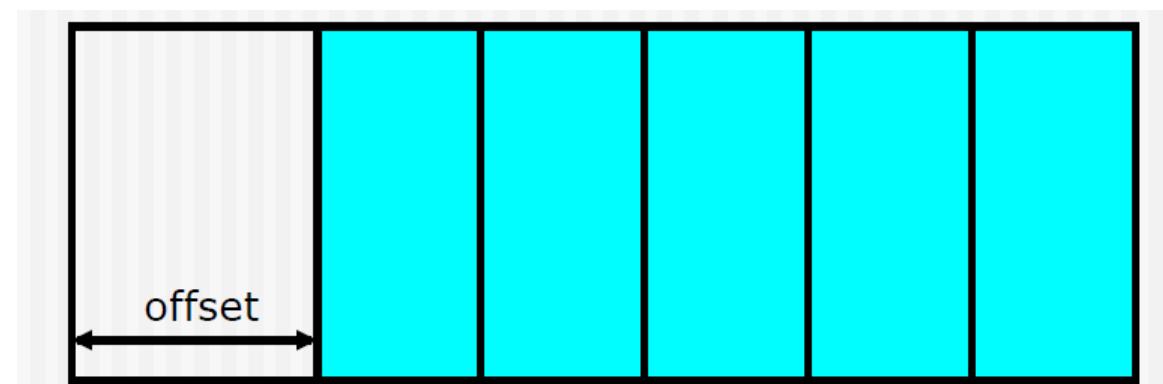
```
void Si_det_Parameterisation::ComputeDimensions (G4Box& Si_det_box, const G4int, const G4VPhysicalVolume*) const
{
    G4double XhalfLength = 0.5* cm;
    G4double YhalfLength = 1 * cm;
    G4double ZhalfLength = 1.* cm;
    Si_det_box.SetXHalfLength(XhalfLength);
    Si_det_box.SetYHalfLength(YhalfLength);
    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:  $\frac{\text{размер материнского объёма} - \text{sмещение} (offset)}{\text{число копий}}$

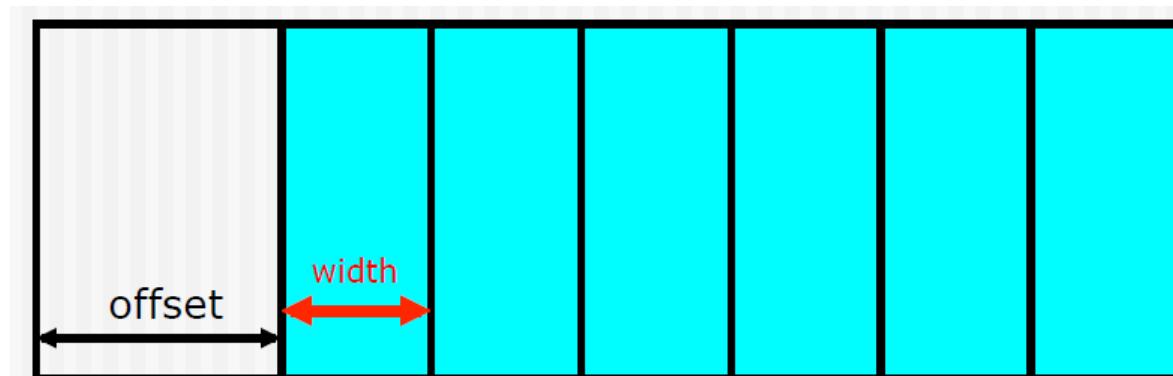


# G4PVDivision

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:  $\text{int}(\frac{\text{размер материнского объёма} - \text{смещение} (\text{offset})}{\text{width}})$

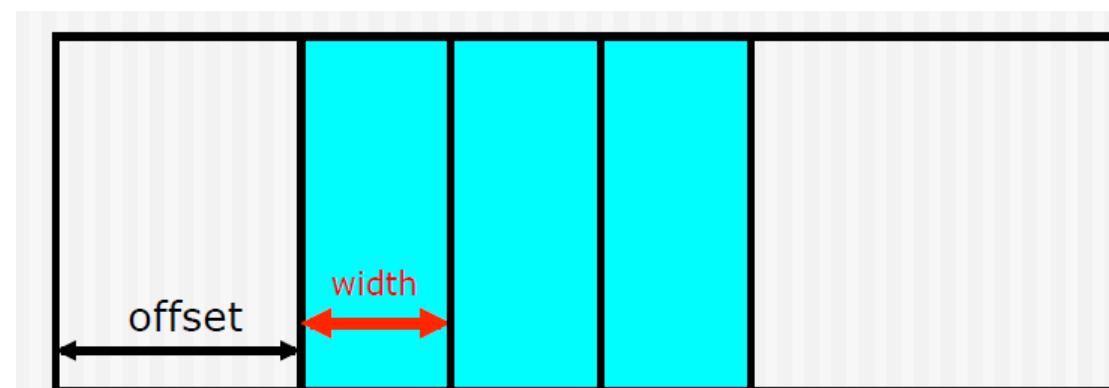


# G4PVDivision

Вариант 3 конструктора физического объёма с копиями

```
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);
```

nDivisions копий с толщиной width



# G4PVDivision

Размещение копий в материнском объёме с использованием G4PVDivision 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 G4PVDivision 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 3
//Si_det_pvpl= new G4PVDivision("Si_det_vol_pvpl", Si_det_log, Si_log, kXAxis, 5, 1.*cm, 3.*cm); // Вариант 3
```

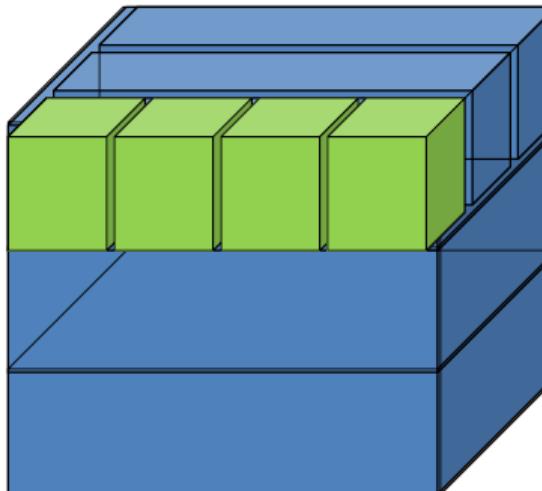
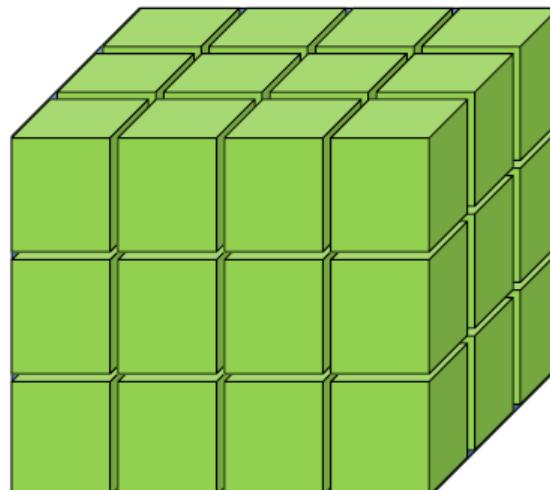
# Nested parameterization (вложенная параметризация)

При моделировании экспериментальных установок ячеистой структуры, состоящих из одинаковых детекторов без промежутков между ними, можно вместо трёхмерной параметризации G4PVParameterised использовать специальную параметризацию **G4VNestParameterisation**.

Класс **G4VNestParameterisation** 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 возрастании числа детекторов.



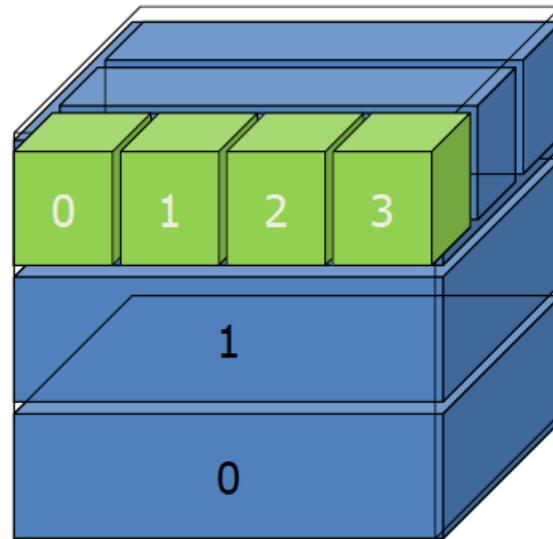
# Nested parameterization

## 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



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

Индекс копии вдоль первой оси определяется как: parentTouch → GetCopyNumber(1);  
вдоль второй оси: parentTouch → GetCopyNumber(0);  
вдоль третьей оси: repNo.

GetNumberOfMaterial возвращает значение, которое интерпретируется как полное число используемых материалов nMaterial.

GetMaterial может возвращать индекс материала idx в диапазоне [0, nMaterial-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);
```

Память выделена, объём не размещён

## 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);
```

# 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

Полное число материалов (заглушка).

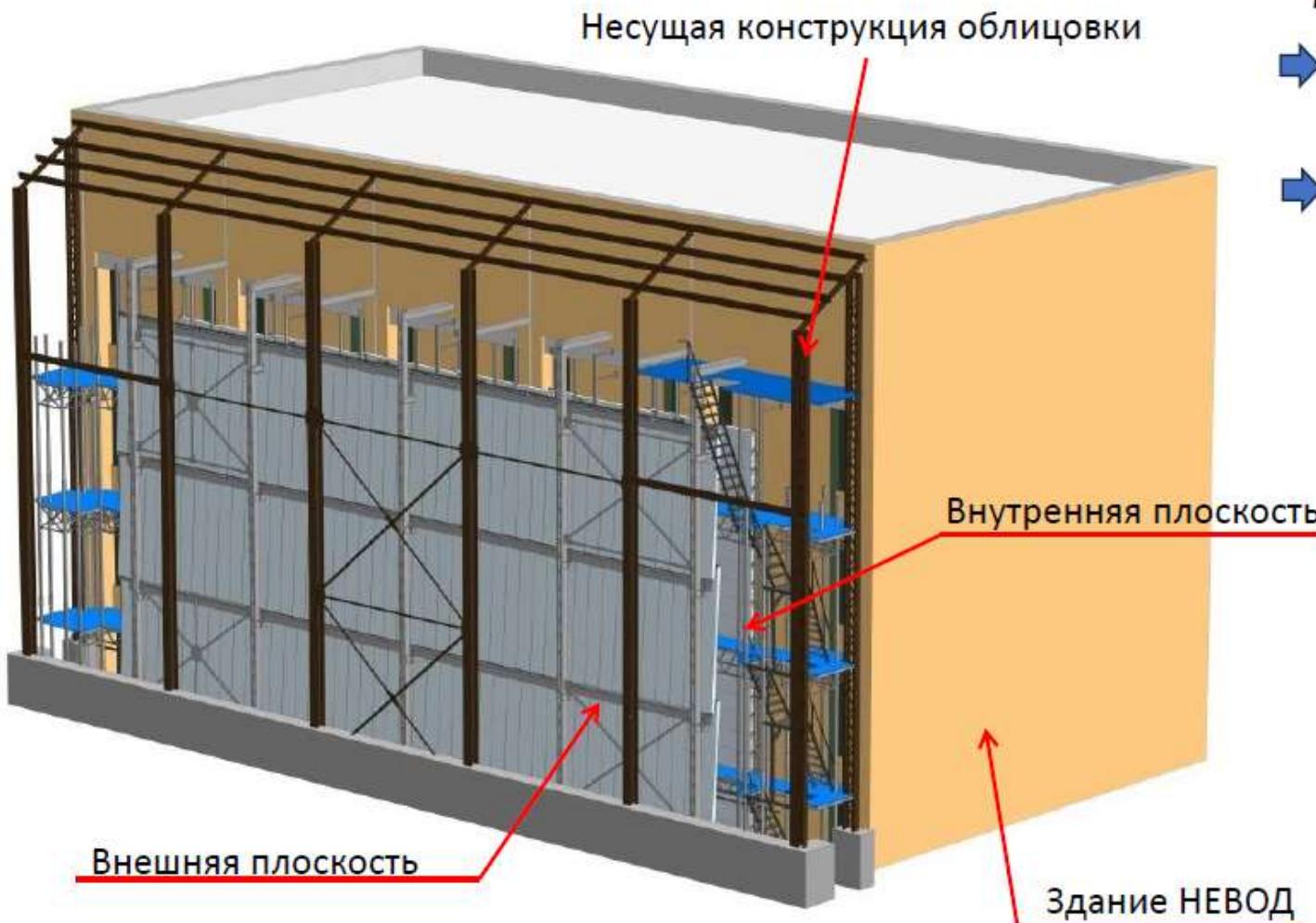
```
G4int Si_det_Parameterisation::GetNumberOfMaterials() const
{
    return 1;
}
```

Индекс материала (заглушка).

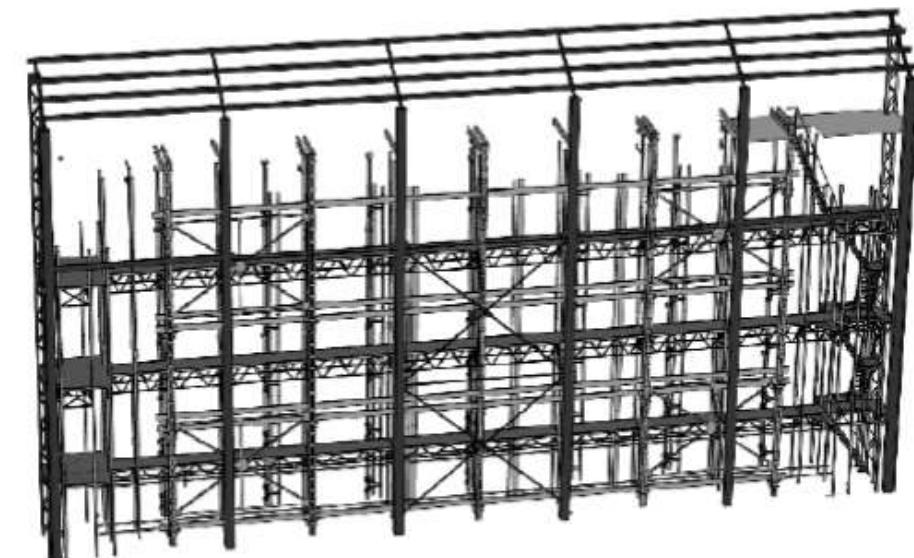
```
G4Material* Si_det_Parameterisation::GetMaterial(G4int num) const
{
    G4Material* mat;
    G4NistManager* nist; nist = G4NistManager::Instance();
    mat=nist->FindOrBuildMaterial("G4_Si");
    return mat;
}
```

# Работа с нетривиальной геометрией в Geant4

Пример: детектор ТРЕК (НЕВОД, НИЯУ МИФИ)



- ТРЕК регистрирует поток окологоризонтальных мюонов
  - ➡ Сильное влияние геометрии здания и металлоконструкций на характеристики потока частиц
  - ➡ Окружающая конструкция сильно *неоднородна*: невозможно свести геометрию к набору элементарных тел



- Существует 3D-модель металлического каркаса, выполненная в КОМПАС-3D

# Импорт 3D-моделей из программ CAD (САПР) в Geant4

## Пакет CADMesh (2)

C. M. Poole, I. Cornelius, J. V. Trapp, C. M. Langton.

- Позволяет транслировать CAD-модели в объекты класса G4Solid в Geant4.
- Поддерживаемые форматы: .PLY, .STL, .OBJ.
- Представляет собой единственный header-файл, который нужно скопировать в директорию /include проекта
- Не требует установки дополнительных проектов (но для дополнительных возможностей – да)

Пример:

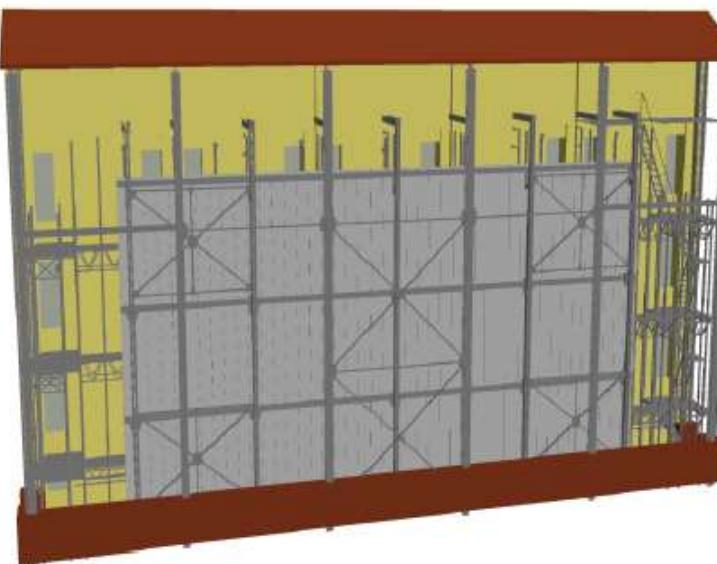
```
#include "CADMesh.hh"

G4VPhysicalVolume* DetectorConstruction::Construct(){
...
    // создаем объект G4TesselatedSolid
    auto mesh = CADMesh::TessellatedMesh::FromSTL("path/TREK_metal.stl");

    // в конструктор G4LogicalVolume передаем объект G4Solid, который
    // возвращается функцией GetSolid()
    auto mesh_logical = new G4LogicalVolume(mesh -> GetSolid(), material,
                                            "mesh_logical", 0, 0, 0);

    // создаем и размещаем объект G4PhysicalVolume обычным способом
    new G4PVPlacement(0, G4ThreeVector(), mesh_logical, "mesh_physical",
                      logicWorld, false, 0);
```

Результат трансляции геометрии в Geant4:



Импортированы модели:

- Металлоконструкций
- Здания
- Фундамента пристройки
- Облицовки и крыши

Полезные функции:

```
mesh -> SetScale(mm); // задать масштаб (mm = 1 в Geant4 !)

mesh -> SetOffset(x, y, z); // задать оффсет
mesh -> SetOffset(G4ThreeVector(x, y, z));

// заполнить сетку mesh тетраэдрами (требует подключения библиотеки tetgen)
auto tets = CADMesh::TetrahedralMesh::FromSTL("mesh.stl");
```

# 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*

*Визуализация напрямую из Geant4*

OpenGL

OpenInventor

Qt

*Создание файла + внешняя программа просмотра*

HepRep/WIRED

DAWN

VRML

gMocrenFile

*Только геометрия детектора*

RayTracer

AsciiTree

# 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 этого 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 numberEvent = 1000;  
runManager->BeamOn(numberEvent);  
  
// 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**

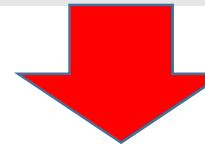
```
runManager->Initialize();
// Initialize visualization
G4VisManager* visManager = new G4VisExecutive;
visManager->Initialize();
G4UImanager *UImanager = G4UImanager::GetUIpointer();

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