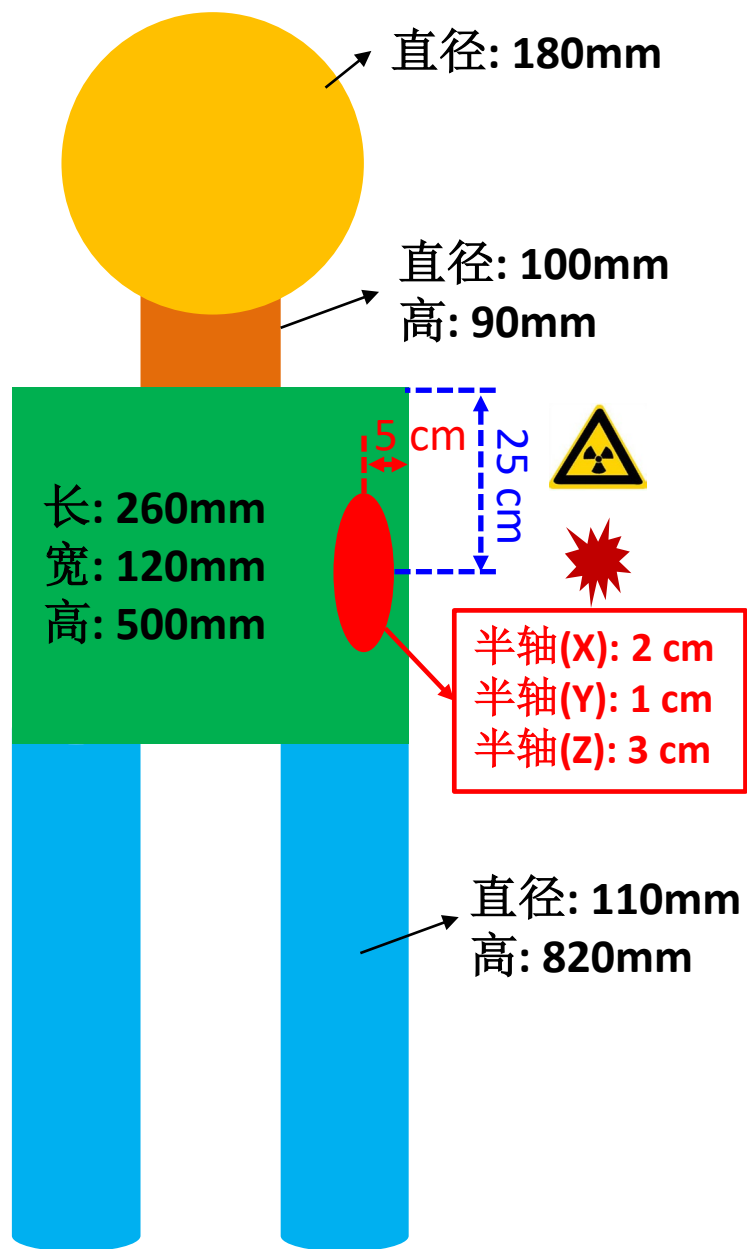


探测器模拟与数据分析

--探测器模拟基础

曹国富

中国科学院高能物理研究所



- ❖ 放疗是肿瘤治疗的一种常用手段，射线种类有伽马/质子/中子/离子等。
- ❖ 请设计并实现一个探测器模拟程序，从能量沉积的大小、尺度和位置几方面，对伽马和质子射线能量进行优化，对比使用这两种射线进行放疗的优劣。
- ❖ 硼中子俘获疗法(BNCT)是一种新的疗法，2020年全球首个BNCT设备在日本获批上市。假如使用0.5 eV的热中子束流，请研究预期治疗效果与肿瘤区内¹⁰B原子浓度的关系，以及束流截面大小的影响。最后，与其它射线的治疗效果进行对比。
- ❖ 肿瘤大小和位置如图红色区域所示。

- ❄ 开启虚拟机，并打开Terminal。
- ❄ Geant4, ROOT和cmake的环境变量都已经自动设置完成。

```
bash$ vim ~/.cshrc
```

- ❄ 从geant4目录下拷贝B1例子到自己的工作目录。

```
bash$ cd ~/
bash$ mkdir g4work
bash$ cp /usr/local/geant4.10.07/share/Geant4-10.7.0/examples/basic/B1 . -r
```

- ❄ 编译并运行B1。

```
bash$ cd ~/g4work/B1
bash$ mkdir build <建立一个编译目录>
bash$ cd build
bash$ cmake .. <生成B1的makefile>
bash$ make <编译B1>
bash$ ./exampleB1 <运行B1>
geant4$ /run/beamOn 10
```

□ B1例子中有3种几何形状:

长方体 (*G4Box*)

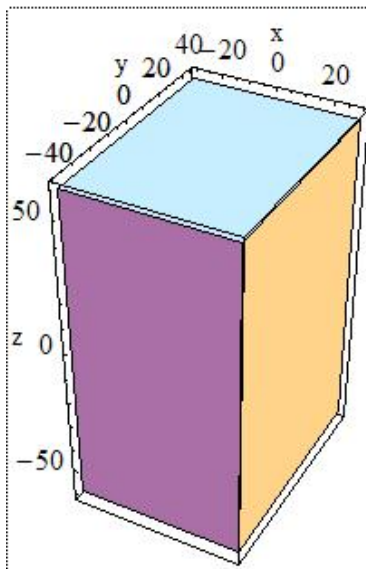
圆锥 (*G4Cons*)

梯形 (*G4Trd*)

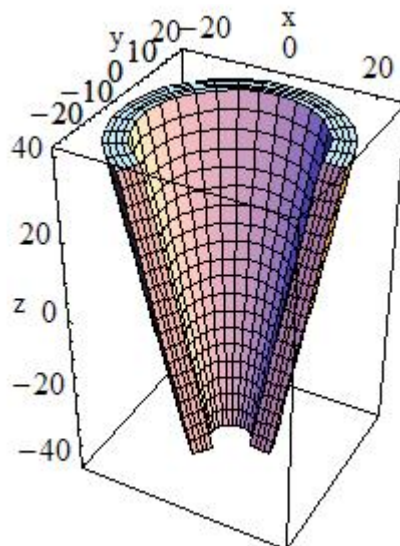
□ B1例子中使用了4种物质:

空气、水、组织、骨骼

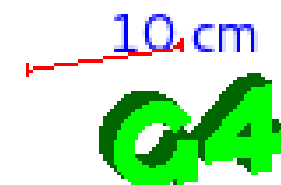
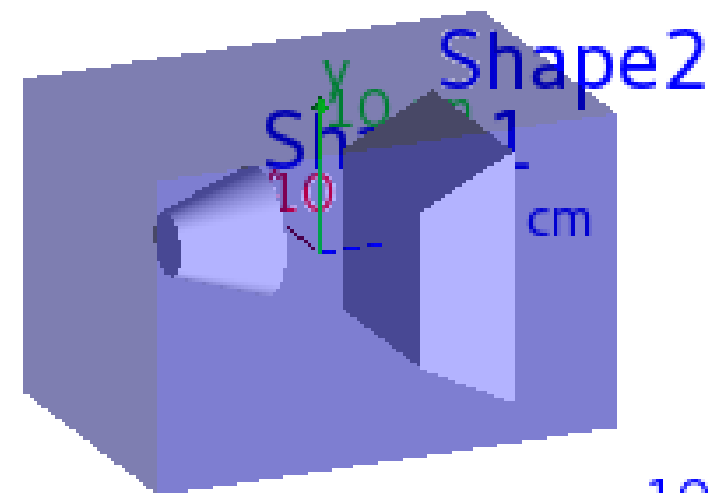
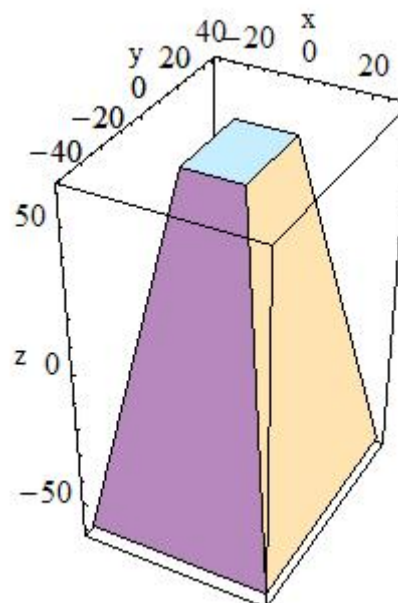
G4Box



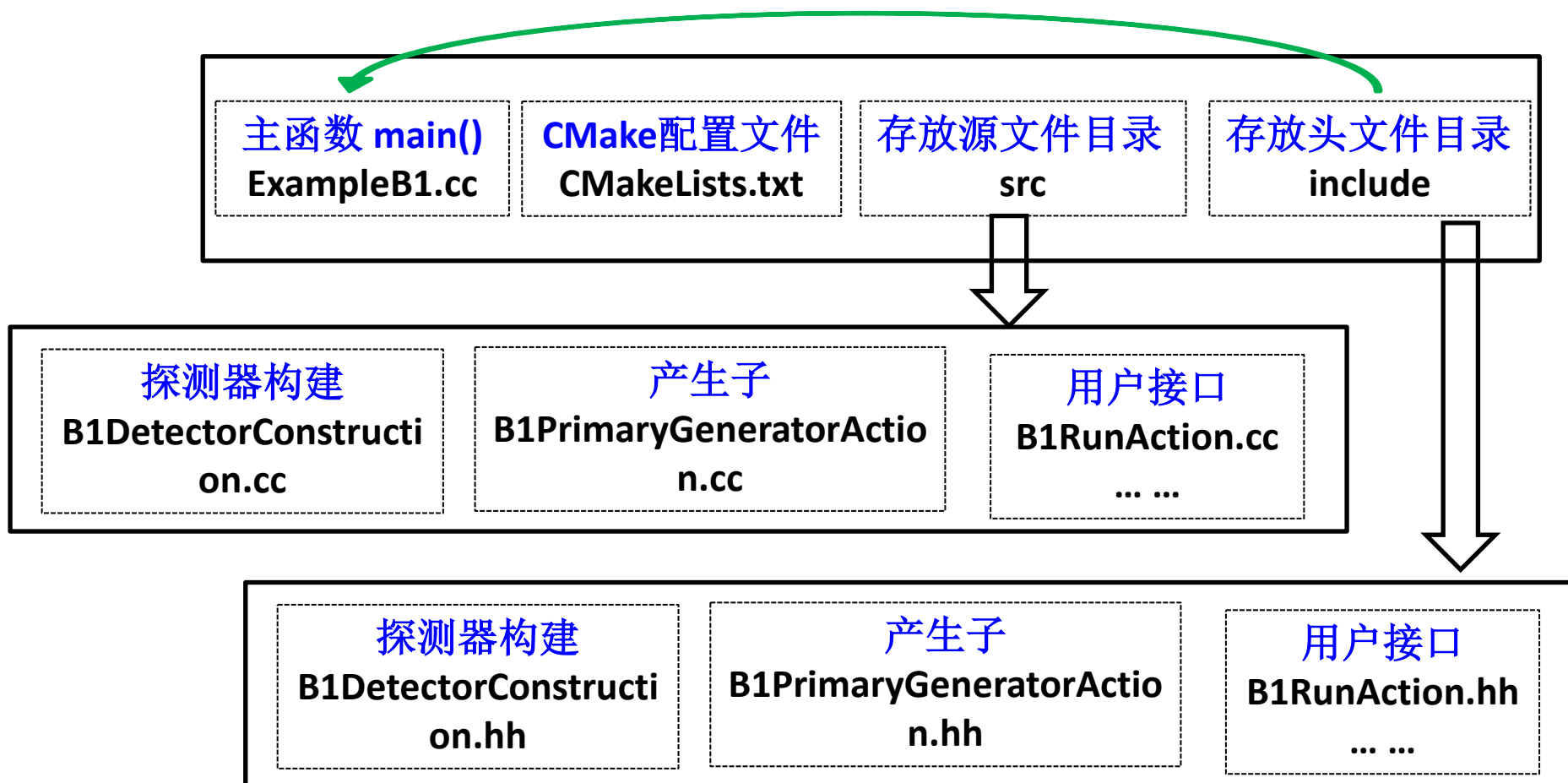
G4Cons



G4Trd



- ❄ Geant4只提供用户工具（库文件），没有主函数，也就没有可执行文件
- ❄ 用户在使用时，需要自己写主函数，例如B1例子



利用Geant4构建探测器 (物质+几何)

□ 如B1例子所示，用于构建探测器的类**必须**要继承其**基类**
(G4VUserDetectorConstruction)

```
49 class G4VUserDetectorConstruction
50 {
51     public:
52         G4VUserDetectorConstruction();
53         virtual ~G4VUserDetectorConstruction();
54
55     public:
56         virtual G4VPhysicalVolume* Construct() = 0;
57
58         virtual void ConstructSDandField();
59         //This method is used in multi-threaded applications to build
60         //per-worker non-shared objects: SensitiveDetectors and Field managers
61
62         virtual void CloneSD();
63         virtual void CloneF();
64
65     public:
66         void RegisterParallelWorld(G4VUserParallelWorld*);
67
68     public:
69         G4int ConstructParallelGeometries();
70         void ConstructParallelSD();
71 }
```

□ 用户需要具体实现探测器构建类中的**Construct()函数**，即，探测器的几何与物质定义需要在此函数中完成，并在最后返回“world physical volume”。

- ❄ **准备一个你自己的探测器构建类，需要从基类G4VUserDetectorConstruction继承。**
- ❄ **具体实现Construct()函数：**
 - 1) **构建所需的物质 (G4Material)**
 - 2) **定义探测器几何形状 (Solid volume)**
 - 3) **定义探测器几何属性 (Logical volume)**
 - 4) **完成探测器摆放 (Physical volume)**
 - 5) **定义探测器的显示属性(可选)**
 - 6) **定义regions (可选)**
- ❄ **具体实现ConstructSDandField()函数：**
 - 1) **定义探测器内的电场或磁场(可选)**
 - 2) **定义灵敏探测器(可选)**
- ❄ **把你构建好的探测器交给G4RunManager。**
- ❄ **当探测器结构较为复杂时，充分利用C++语言的特点，实现对探测器的模块化定义。**

❄ 物质(Material)由元素(Element)组成

➤ 化合物 (*molecules compounds*), 混合物 (*mixture*)

❄ 每种元素都包含若干个同位素(Isotopes)

Geant4给用户提供了3个类:

- | | | |
|-------------------------------------|---|------------|
| ➤ Isotope | ↔ | G4Isotope |
| ➤ Elements | ↔ | G4Element |
| ➤ molecules, compounds and mixtures | ↔ | G4Material |

G4Isotope and G4Element用来定义原子的属性。

- atomic number (Z), number of nucleons(N), molecular mass(A), shell structure, etc.
- 用在物质的定义中

G4Material

- 定义元素的组成
- 定义物质的属性: 温度(temperature), 压力(pressure), 状态(state), 密度(density)
- 用在探测器的几何定义中

□ 由单一元素组成的物质 (液氩)

➤ 通过 “名字, 密度, Z, A” 定义物质

$G4double\ density = 1.390 * g/cm^3;$

$G4double\ a = 39.95 * g/mole;$

$G4Material * LAr =$

$New\ G4Material("LiquidArgon", z=18., a, density);$

□ 化合物：由多个原子组成 (H_2O)

```
G4double a = 1.01 * g/mole;  
G4Element* elH = new G4Element("Hydrogen", symbol="H", z=1., a);  
a = 16.00 * g/mole;  
G4Element* elO = new G4Element("Oxygen", symbol="O", z=8., a);  
  
G4int ncomp;  
G4double density = 1.000 * g/cm3;  
G4Material* H2O = new G4Material("Water", density, ncomp=2);  
G4int natoms;  
H2O->AddElement(elH, natoms=2);  
H2O->AddElement(elO, natoms=1);
```

□ 混合物 (Air)

```
G4double a = 14.01 * g/mole;  
G4Element* eLN =  
new G4Element(name="Nitrogen", symbol="N", z= 7., a);  
a = 16.00 * g/mole;  
G4Element* eLO =  
new G4Element(name="Oxygen", symbol="O", z= 8., a);  
  
G4int ncomp;  
G4double density = 1.290 * mg/cm3;  
G4Material* Air =  
new G4Material(name="Air", density, ncomp=2);  
G4double fracMass;  
Air-> AddElement(eLN, fracMass=70.0*perCent);  
Air-> AddElement(eLO, fracMass=30.0*perCent);
```


□ 混合物：气凝胶 (Aerogel)

```
G4Element* eLC = ...;    // define "carbon" element
G4Material* SiO2 = ...;   // define "quartz" material
G4Material* H2O = ...;    // define "water" material

G4int ncomp;
G4double fractionmass;
G4double density = 0.200 * g/cm3;
G4Material* Aerog =
  new G4Material("Aerogel", density, ncomp=3);
Aerog->AddMaterial(SiO2, fractionmass=62.5*perCent);
Aerog->AddMaterial(H2O , fractionmass=37.4*perCent);
Aerog->AddElement (eLC , fractionmass= 0.1*perCent);
```

例如：为核反应定义富集的铀元素。

```
G4Isotope* isoU235 = new G4Isotope("U235", iz=92, ia=235, a=235.0439242*g/mole);
G4Isotope* isoU238 = new G4Isotope("U238", iz=92, ia=238, a=238.0507847*g/mole);

G4Element* elenrichedU = new G4Element("enriched U", symbol="U", ncomponents=2);
elenrichedU-> AddIsotope(isoU235, abundance=90.*perCent);
elenrichedU-> AddIsotope(isoU238, abundance=10.*perCent);

G4Material* matenrichedU=
new G4Material("U for nuclear power generation", density=19.050*g/cm3,
ncomponents=1, kStateSolid);
matenrichedU-> AddElement(elenrichedU, fractionmass=1.);
```

❄ 真空(vacuum)可以定义为低密度物质

➤ 使用 “*universe_mean_density* (=1.e-25*g/cm³)”

```
G4double z = 1.;  
G4double a = 1.008*g/mole;  
G4double density = 1.e-25*g/cm3;  
G4double temperature = 2.73*kelvin;  
G4double pressure = 3.e-18*pascal;  
G4Material* vacuum =  
  new G4Material("interGalactic", z, a, density, kStateGas, temperature, pressure);
```

❄ 对于强相互作用过程，正确定义探测器的物质对正确计算其反应截面是至关重要的。

➤ 避免使用 "*averaged material*".

❄ **在Geant4中，可以直接使用NIST物质数据库。**

➤ <https://www.nist.gov/pml/atomic-weights-and-isotopic-compositions-relative-atomic-masses>

❄ **对一些主要参数的描述都是非常准确的。**

- 密度
- 平均激发势
- 化学键
- 元素组成
- 同位素组成

❄ **用户可以使用UI命令来打印物质和元素列表，如**

- `/material/nist/printElement` (打印定义好的元素)
- `/material/nist/listMaterials` (打印定义好的物质)

❄ **更加详细的信息，请参考Geant4用户手册附录6。**

Z	A	m	error	(%)	Aeff
=====					
14	Si	22	22.03453	(22)	28.0855(3)
		23	23.02552	(21)	
		24	24.011546	(21)	
		25	25.004107	(11)	
		26	25.992330	(3)	
		27	26.98670476	(17)	
		28	27.9769265327	(20)	92.2297(7)
		29	28.97649472	(3)	4.6832 (5)
		30	29.97377022	(5)	3.0872 (5)
		31	30.97536327	(7)	
		32	31.9741481	(23)	
		33	32.978001	(17)	
		34	33.978576	(15)	
		35	34.984580	(40)	
		36	35.98669	(11)	
		37	36.99300	(13)	
		38	37.99598	(29)	
		39	39.00230	(43)	
		40	40.00580	(54)	
		41	41.01270	(64)	
		42	42.01610	(75)	

同位素使用自然丰度，一共有3000多种同位素可供使用。

Elementary Materials from the NIST Data Base				
Z	Name	ChFormula	density(g/cm ³)	I(eV)
1	G4_H	H_2	8.3748e-05	19.2
2	G4_He		0.000166322	41.8
3	G4_Li		0.534	40
4	G4_Be		1.848	63.7
5	G4_B		2.37	76
6	G4_C		2	81
7	G4_N	N_2	0.0011652	82
8	G4_O	O_2	0.00133151	95
9	G4_F		0.00158029	115
10	G4_Ne		0.000838505	137
11	G4_Na		0.971	149
12	G4_Mg		1.74	156
13	G4_Al		2.6989	166
14	G4_Si		2.33	173

✧ NIST中的基本材料:

- H to Cf (Z=1 to 98)

✧ NIST中的混合物:

- G4_ADIPOSE_TISSUE

✧ 高能物理中常用的物质:

- liquid Ar, PbWO₄, etc

Compound Materials from the NIST Data Base				
N	Name	ChFormula	density(g/cm ³)	I(eV)
13	G4_Adipose_Tissue		0.92	63.2
	1	0.119477		
	6	0.63724		
	7	0.00797		
	8	0.232333		
	11	0.0005		
	12	2e-05		
	15	0.00016		
	16	0.00073		
	17	0.00119		
	19	0.00032		
	20	2e-05		
	26	2e-05		
	30	2e-05		
4	G4_Air		0.00120479	85.7
	6	0.000124		
	7	0.755268		
	8	0.231781		
	18	0.012827		
2	G4_CsI		4.51	553.1
	53	0.47692		
	55	0.52308		

不需要提前定义元素和物质。

```
G4NistManager* manager = G4NistManager::GetPointer();

G4Element* elm = manager->FindOrBuildElement("symb", G4bool iso);

G4Element* elm = manager->FindOrBuildElement(G4int Z, G4bool iso);

G4Material* mat = manager->FindOrBuildMaterial("name", G4bool iso);
// iso is omitable (true by default)

G4Material* mat = manager->ConstructNewMaterial("name",
                                                const std::vector<G4int>& Z,
                                                const std::vector<G4double>& weight,
                                                G4double density, G4bool iso);

G4double isotopeMass = manager->GetMass(G4int Z, G4int N);
```

□ 参考B1例子，与物质的定义一样，探测器几何的定义仍在探测器构建类中的 **Construct()** 函数中实现，并在最后返回“world physical volume”。

□ 探测器几何的构建方法

❄ **Introduction**

❄ **Solid and shape**

❄ **Logical volume**

❄ **Various ways of placement**

- Simple placement volume
- Parameterized volume
- Replicated volume
- Divided volume

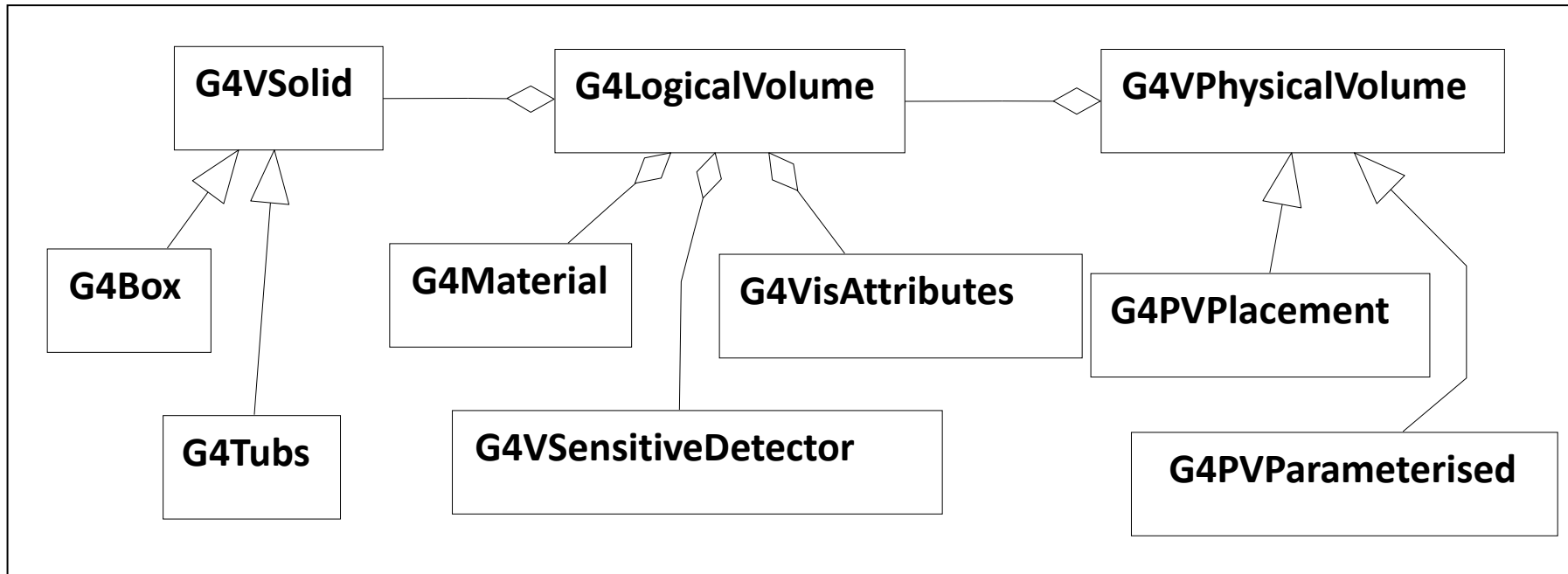
几何形状<长方形，球形，
柱形，>

几何属性<物质，显示属
性，灵敏探测器，>

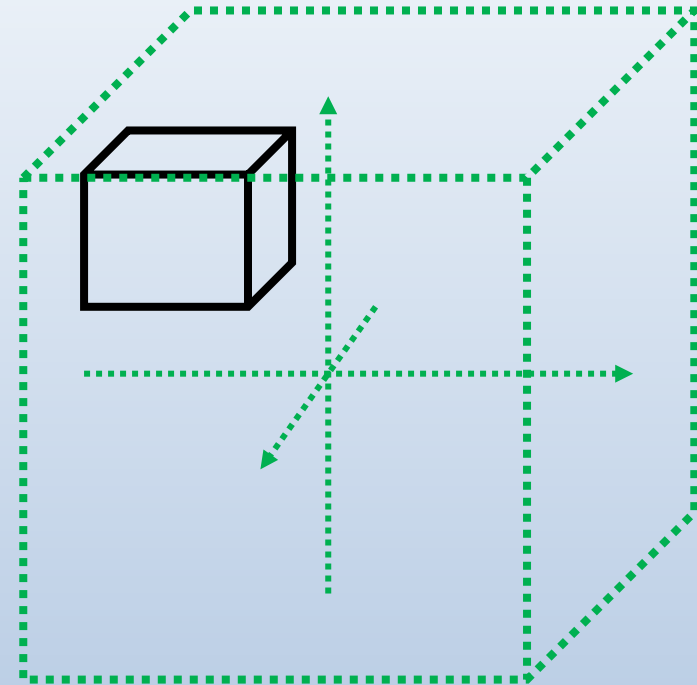
几何的摆放<位置，旋
转，>

❄ 三步构建几何

- **G4VSolid** -- *shape, size*
- **G4LogicalVolume** -- *daughter physical volumes, material, sensitivity, user limits, etc.*
- **G4VPhysicalVolume** -- *position, rotation*



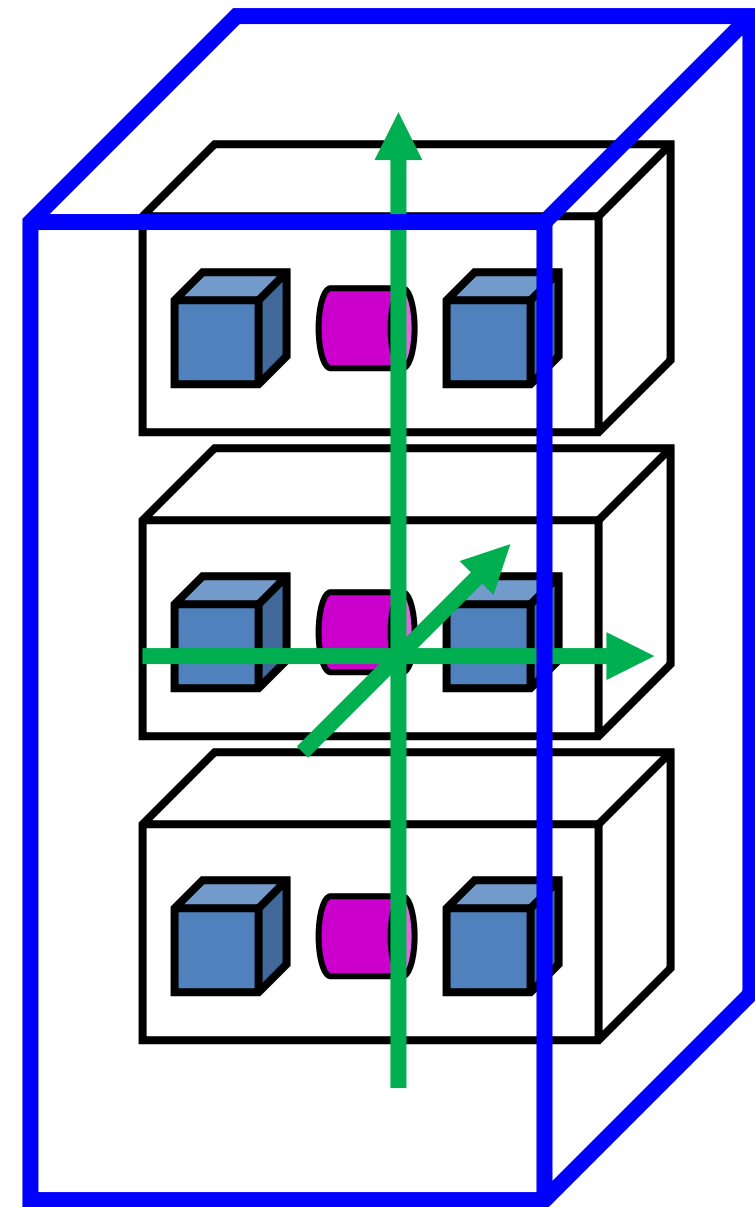
```
G4VSolid* pBoxSolid =  
new G4Box("aBoxSolid", 1.*m, 2.*m, 3.*m);  
  
G4LogicalVolume* pBoxLog =  
new G4LogicalVolume( pBoxSolid, pBoxMaterial, "aBoxLog", 0, 0, 0);  
  
G4RotationMatrix* rm = new G4RotationMatrix();  
rm->RotateY(90*deg);  
G4VPhysicalVolume* aBoxPhys =  
new G4PVPlacement( rm, G4ThreeVector(posX, posY, posZ), pBoxLog,  
"aBoxPhys", pMotherLog, 0, copyNo);
```



❄ 当把一个体积被放入其母体积时，其位置与旋转都是相对于其母体积的局域坐标来说的。母体积局域坐标系的原点在几何中心。

➤ 子体积的大小不能超出其母体积。

- ❄ 一个逻辑体积可以被多次摆放，一个母体积内可以放置一个或多个子体积
- ❄ 体积之间的母女关系是通过G4LogicalVolume来建立的
 - 如果一个母体积被多次摆放，那么这个母体积内的子体积会跟随母体积一起被多次摆放。
- ❄ 世界只能有一个！(world volume必须是唯一的，它包含所有其它体积)
 - World volume定义了全局坐标系，坐标原点在world的中心
 - 在模拟过程中，径迹的位置都是相对于全局坐标。



❄ Geant4中已经定义好的几何形状

➤ CSG (Constructed Solid Geometry) solids

- G4Box, G4Tubs, G4Cons, G4Trd, ...
- Analogous to simple GEANT3 CSG solids

➤ Specific solids (CSG like)

- G4Polycone, G4Polyhedra, G4Hype, ...

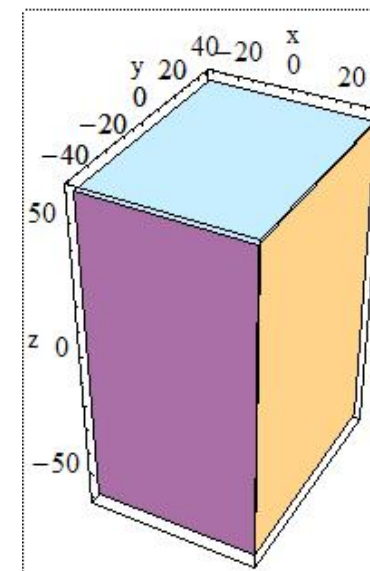
➤ Tessellated Solids

- 定义几何体表面(G4VFacet)
- 适用从CAD转换的复杂几何

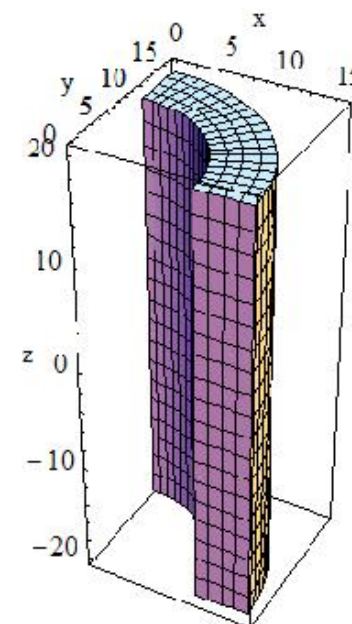
➤ Boolean solids

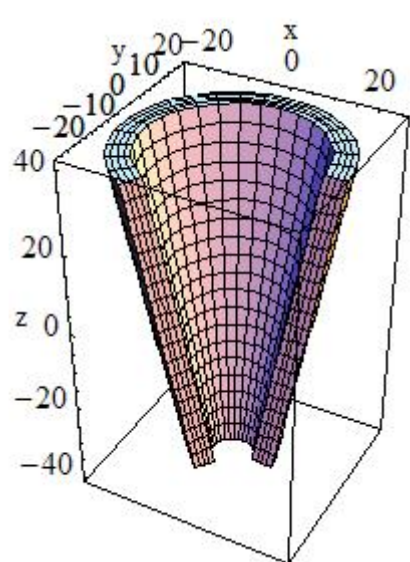
- G4UnionSolid, G4SubtractionSolid, ...

```
G4Box(const G4String &pname,    // name
      G4double half_x,          // X half size
      G4double half_y,          // Y half size
      G4double half_z);         // Z half size
```

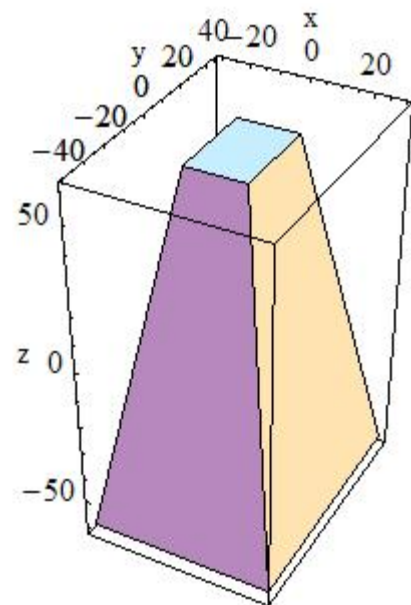


```
G4Tubs(const G4String &pname,    // name
      G4double pRmin,             // inner radius
      G4double pRmax,             // outer radius
      G4double pDz,               // Z half length
      G4double pSphi,             // starting Phi
      G4double pDphi);            // segment angle
```

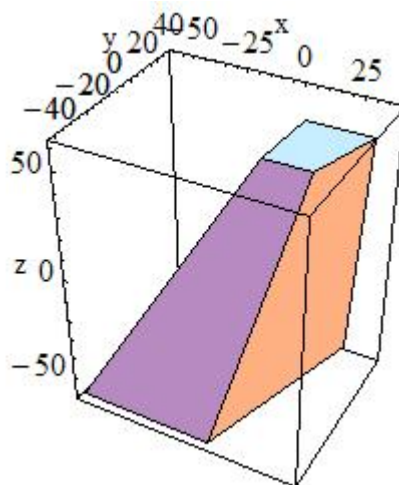




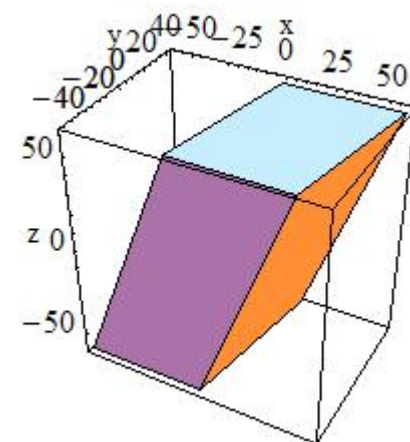
G4Cons



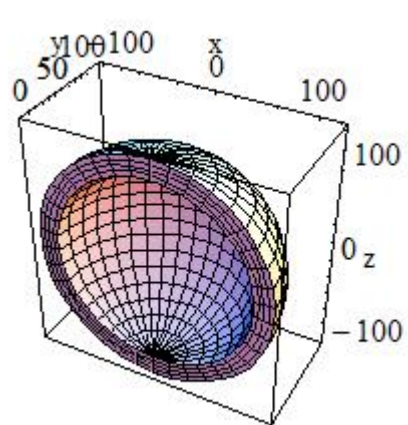
G4Trd



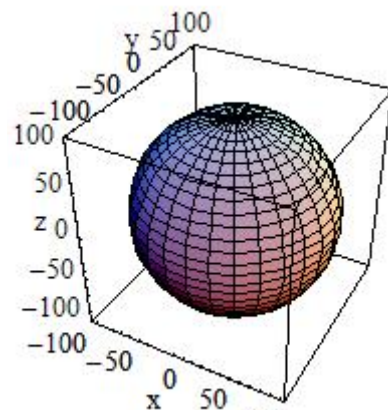
G4Trap



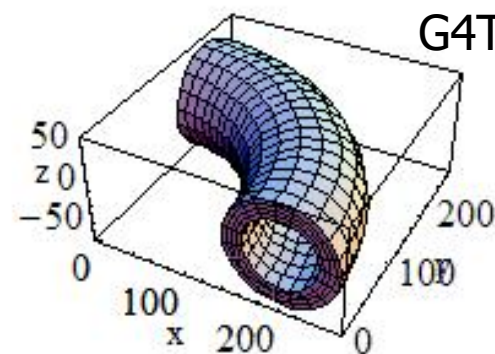
G4Para
(parallelepiped)



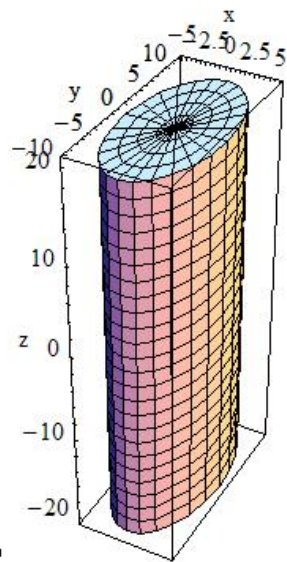
G4Sphere



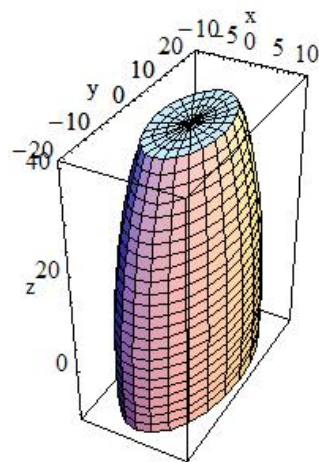
G4Orb
(full solid sphere)



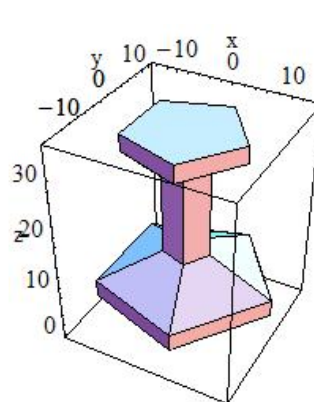
G4Torus



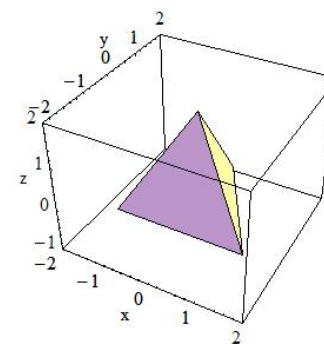
G4EllipticalTube



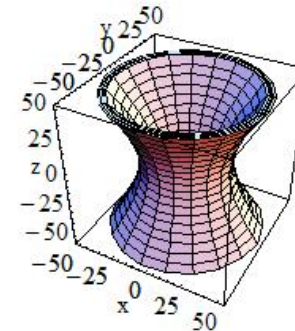
G4Ellipsoid



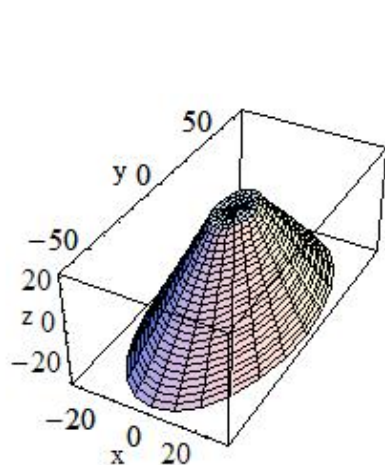
G4Polyhedra



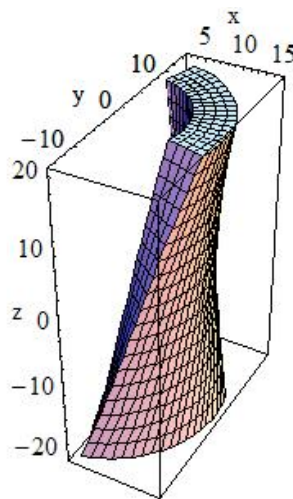
G4Tet
(tetrahedra)



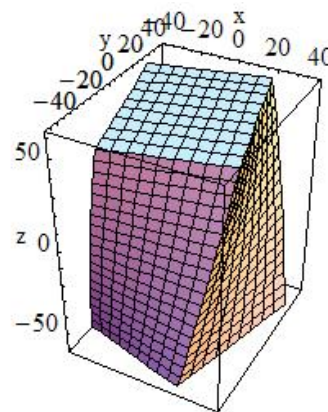
G4Hype



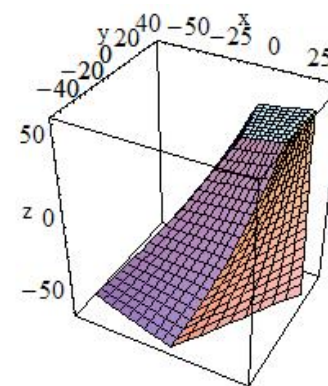
G4EllipticalCone



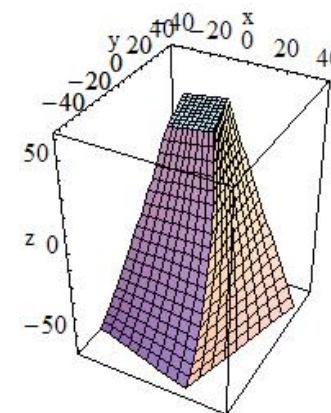
G4TwistedTubs



G4TwistedBox



G4TwistedTrap



G4TwistedTrd

更多的形状，请参考 [Section 4.1.2 of Geant4 Application Developers Guide](#)

```
// First declare a tessellated solid
G4TessellatedSolid* solidTarget = new G4TessellatedSolid("Solid_name");
G4double targetSize = 10*cm ;
G4TriangularFacet *facet1 = new G4TriangularFacet (G4ThreeVector(-targetSize, -targetSize, 0.0),
                                                    G4ThreeVector(+targetSize, -targetSize, 0.0),
                                                    G4ThreeVector( 0.0, 0.0, +targetSize),
                                                    ABSOLUTE);

G4TriangularFacet *facet2 = new G4TriangularFacet (G4ThreeVector(+targetSize, -targetSize, 0.0),
                                                    G4ThreeVector(+targetSize, +targetSize, 0.0),
                                                    G4ThreeVector( 0.0, 0.0, +targetSize),
                                                    ABSOLUTE);
```

```
G4TriangularFacet ( const G4ThreeVector Pt0,
                    const G4ThreeVector vt1,
                    const G4ThreeVector vt2,
                    G4FacetVertexType fType )
```

i.e., it takes 4 parameters to define the three vertices:

G4FacetVertexType	ABSOLUTE in which case Pt0, vt1 and vt2 are the three vertices in anti-clockwise order looking from the outside.
G4FacetVertexType	RELATIVE in which case the first vertex is Pt0, the second vertex is Pt0+vt1 and the third vertex is Pt0+vt2, all in anti-clockwise order when looking from the outside.



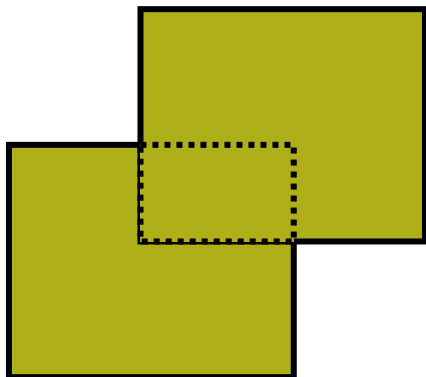
❄ 几何体间可以通过布尔操作进行合并：

- G4UnionSolid, G4SubtractionSolid, G4IntersectionSolid.
- 要求：2个几何体, 1个布尔运算, 第2个几何的平移与转动（可选）。
- 第2个几何的位置是相对于第一个几何的坐标系而言的。
- 2个几何通过布尔操作后，变成一个几何，因此，它还可以和第3个几何通过布尔操作进行合并。

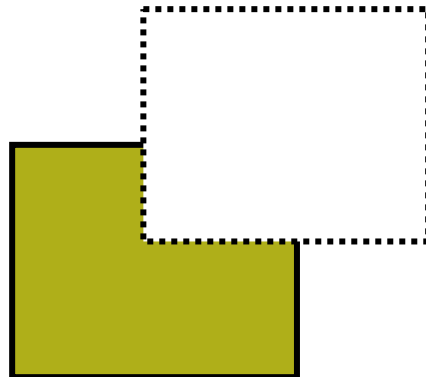
❄ CSG型几何或者布尔型几何都可以通过此种方式进行合并。

❄ 注意：在构建探测器几何时，应避免使用过多的布尔型几何，否则，运行速度会降低。

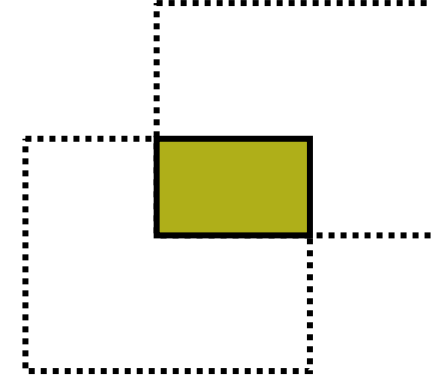
G4UnionSolid



G4SubtractionSolid



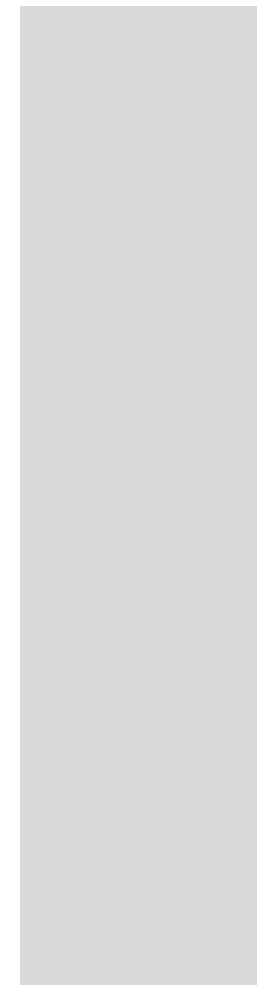
G4IntersectionSolid



```
G4VSolid* box = new G4Box("Box",50*cm,60*cm,40*cm) ;
G4VSolid* cylinder
    = new G4Tubs("Cylinder",0.,50.*cm,50.*cm,0.,2*M_PI*rad) ;
G4VSolid* union
    = new G4UnionSolid("Box+Cylinder", box, cylinder) ;
G4VSolid* subtract
    = new G4SubtractionSolid("Box-Cylinder", box, cylinder,
        0, G4ThreeVector(30.*cm,0.,0.)) ;
G4RotationMatrix* rm = new G4RotationMatrix() ;
rm->RotateX(30.*deg) ;
G4VSolid* intersect
    = new G4IntersectionSolid("Box&&Cylinder",
        box, cylinder, rm, G4ThreeVector(0.,0.,0.)) ;
```

组合以后几何的坐标系及其原点与用于构建布尔型几何的第一个几何体相同

```
G4VSolid  
G4VSolid  
= new G4VSolid  
G4VSolid  
= new G4VSolid  
G4VSolid  
= new G4VSolid  
  
G4Rotat  
rm->Rot  
G4VSolid  
= new
```



组合体

本相同

```
#include "G4MultiUnion.hh"

// Define two -G4Box- shapes
G4Box* box1 = new G4Box("Box1", 5.*mm, 5.*mm, 10.*mm);
G4Box* box2 = new G4Box("Box2", 5.*mm, 5.*mm, 10.*mm);

// Define displacements for the shapes
G4RotationMatrix rotm = G4RotationMatrix();
G4ThreeVector position1 = G4ThreeVector(0.,0.,1.);
G4ThreeVector position2 = G4ThreeVector(0.,0.,2.);
G4Transform3D tr1 = G4Transform3D(rotm,position1);
G4Transform3D tr2 = G4Transform3D(rotm,position2);

// Initialise a MultiUnion structure
G4MultiUnion* munion_solid = new G4MultiUnion("Boxes_Union");

// Add the shapes to the structure
munion_solid->AddNode(*box1,tr1);
munion_solid->AddNode(*box2,tr2);

// Finally close the structure
munion_solid->Voxelize();

// Associate it to a logical volume as a normal solid
G4LogicalVolume* lvol = new G4LogicalVolume(munion_solid,  munion_mat,  "Boxes_Union_LV");
```

Since release 10.4, the possibility to define multi-union structures is part of the standard set of constructs in Geant4. A G4MultiUnion structure allows for the description of a Boolean union of many displaced solids at once, therefore representing volumes with the same associated material.


```
G4LogicalVolume(G4VSolid* pSolid,  
                G4Material* pMaterial,  
                const G4String &name,  
                G4FieldManager* pFieldMgr=0,  
                G4VSensitiveDetector* pSDetector=0,  
                G4UserLimits* pULimits=0);
```

❄ 包含几何体除位置和转动外的所有信息

- 形状和尺寸 (G4VSolid)
- 物质, 灵敏探测器, 可视化属性
- 子体积的位置
- 电磁场, 用户限制 (User limits) , 区域 (Region)

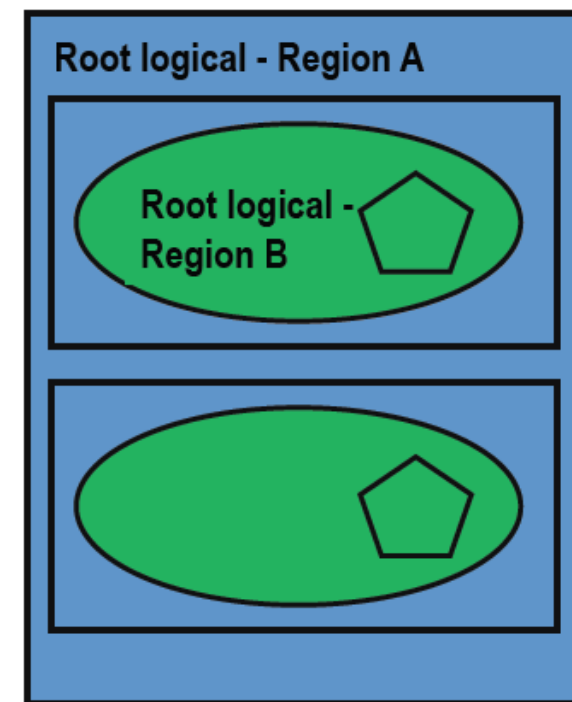
❄ 相同类型的物理几何体(Physical volumes)可以共用同一个逻辑几何。

❄ 几何形状和尺寸(solid)的指针**不能为空**。

❄ 物质的指针**不能为空**。

- ❖ 一个逻辑几何体可以被定义为一个区域，多个逻辑几何可以属于同一区域。
- ❖ 区域(region)是几何层次关系中的一个组成部分。例如，一系列的几何体组成一个集合，可以定义为一个区域，形成一个子系统。
- ❖ 一个逻辑几何一旦被指定为一个区域，那么它就成了一个根逻辑几何(root logical volume)。
 - 所有根逻辑几何的子几何体共享同一个区域，除非某一个子几何成为另一个根逻辑几何。
- ❖ 一个逻辑几何只能属于一个区域，不能被共享。
- ❖ World logical volume 已经被默认指定为一个区域，因此，用户不能再为world指定一个区域。

World Volume - Default Region



❄ 可以用如下方法定义一个区域:

```
G4Region* aRegion = new G4Region("region_name");  
aRegion->AddRootLogicalVolume(aLogicalVolume);
```

❄ 给一个区域指定粒子产生阈(Production cuts)

```
G4Region* aRegion = G4RegionStore::GetInstance()-  
>GetRegion("region_name");  
G4ProductionCuts* cuts = new G4ProductionCuts;  
cuts->SetProductionCut();  
aRegion->SetProductionCuts(cuts);
```

G4Region class may take following quantities.

- void SetProductionCuts(G4ProductionCuts* cut);
- void SetUserInformation(G4VUserRegionInformation* uri);
- void SetUserLimits(G4UserLimits* ul);
- void SetFastSimulationManager(G4FastSimulationManager* fsm);
- void SetRegionalSteppingAction(G4UserSteppingAction* rusa);
- void SetFieldManager(G4FieldManager* fm);

Please note:

- If any of the above properties are not set for a region, properties of the world volume (i.e. default region) are used. Properties of mother region do **not** propagate to daughter region.

- ❄ 几何可视化属性由类**G4VisAttributes**进行管理。
- ❄ 用户可以通过该类对某一个几何体的可视化属性进行设置，如：

```
myLogical->SetVisAttributes(G4VisAttributes::Invisible);  
  
G4VisAttributes* yourLogVis = new G4VisAttributes(G4Colour(0.,1.,1.));  
yourLogical->SetVisAttributes(yourLogVis);
```

- ❄ 尝试对B1例子中的几何设置不同的可视化属性。

G4Colour white	()	;	// white
G4Colour white	(1.,1.,1.)	;	// white
G4Colour gray	(.5,.5,.5)	;	// gray
G4Colour black	(0.,0.,0.)	;	// black
G4Colour red	(1.,0.,0.)	;	// red
G4Colour green	(0.,1.,0.)	;	// green
G4Colour blue	(0.,0.,1.)	;	// blue
G4Colour cyan	(0.,1.,1.)	;	// cyan
G4Colour magenta	(1.,0.,1.)	;	// magenta
G4Colour yellow	(1.,1.,0.)	;	// yellow

❄ Placement volume: 摆放一次

- 一个物理几何体代表着一个真实的探测器单元。

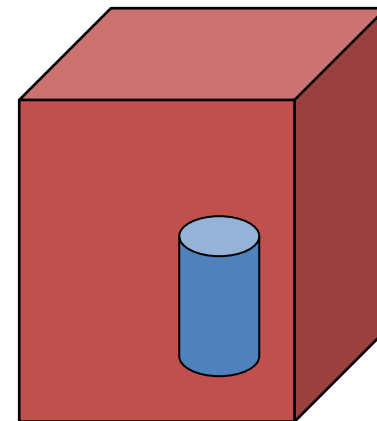
❄ Repeated volume: 重复摆放

- 一个物理几何体代表着任意多个真实的探测器单元。
- 可以有效降低内存的使用。
- Parameterised (参数化)
 - 根据几何体编号(copy number)进行参数化, 并重复摆放。
- Replica and Division
 - 沿着一个坐标轴重复摆放。

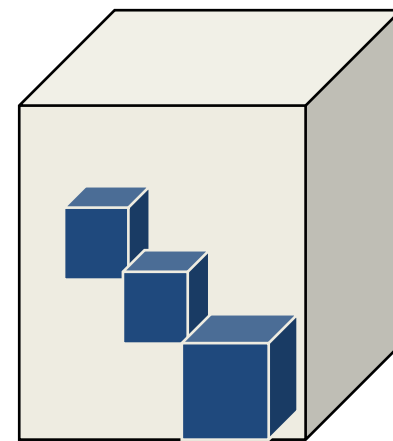
❄ 一个母体积内可以包含:

- 多个一次摆放的几何
- 或者是, 一个重复摆放的几何

❄ 两个几何体之间发生重叠(overlap)是绝对不允许的。



placement



repeated

❄ G4PVPlacement 1 Placement = One Placement Volume

- 几何体在其母体积内摆放一次。

❄ G4PVParameterised 1 Parameterized = Many Repeated Volumes

- 根据几何体编号(copy number)进行参数化摆放。
 - 根据**编号**, 可以对几何体的形状, 大小, 材料, 灵敏探测器, 可视化属性, 位置和旋转实现参数化。
 - 用户必须实现一个具体的类, 继承于**G4VPVParameterisation**。
- 降低内存消耗。
- 目前, 参数化摆放几何的方法仅适用于下面的几何体中
 - a) 几何体内没有子体积, 或者
 - b) 几何体有相同的尺寸和形状, 以确保没有overlap的情况。
- 如果使用**G4PVNestedParameterisation**作为基类进行参数化, 那么, 物质、灵敏探测器和可视化属性可以通过原初几何体的编号进行参数化。

❄ G4PVReplica 1 Replica = Many Repeated Volumes

- 具有相同形状的子体积沿着一个坐标轴整齐摆放。
- 子体积完全填充在母体积内，子母体积间没有缝隙。

❄ G4PVDivision 1 Division = Many Repeated Volumes

- 具有相同形状的子体积沿着一个坐标轴整齐摆放。
- 母体积与最外侧的子体积间有空隙。
- 子体积之间没有空隙。

❄ G4ReflectionFactory 1 Placement = a pair of Placement volumes

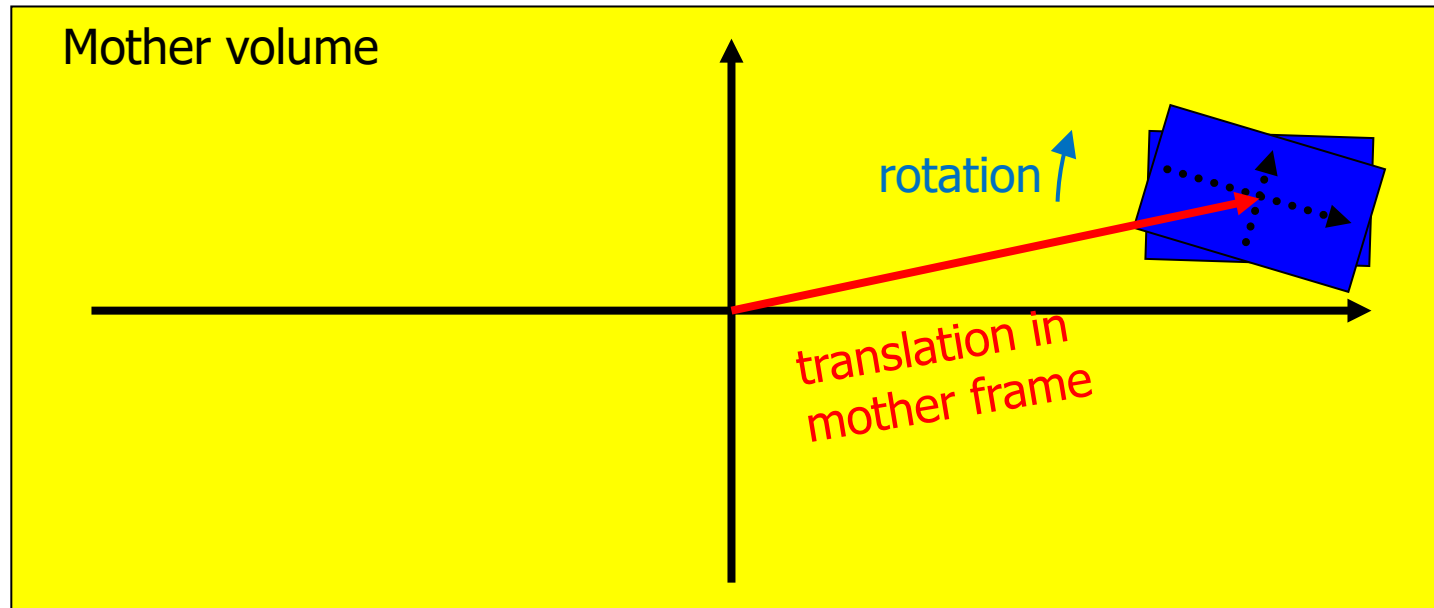
- 按照镜像对称的方式摆放几何。
- 常见于端盖量能器中。

❄ G4AssemblyVolume 1 Placement = a set of Placement volumes

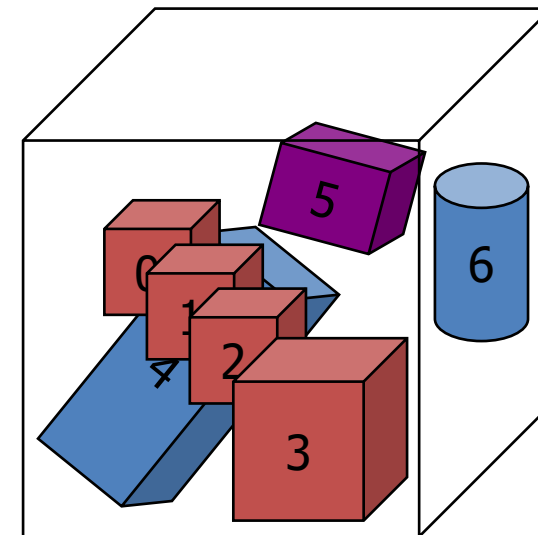
- 单次摆放的一个合集。

```
G4PVPlacement(G4RotationMatrix* pRot,    // rotation of mother frame
              const G4ThreeVector &tlate, // position in mother frame
              G4LogicalVolume *pDaughterLogical,
              const G4String &pName,
              G4LogicalVolume *pMotherLogical,
              G4bool pMany, // 'true' is not supported yet...
              G4int pCopyNo, // unique arbitrary integer
              G4bool pSurfChk=false); // optional boundary check
```

❄ Single volume positioned relatively to the mother volume.



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



- ❄ 在母体积pMother内，使用参数化方法pPar，重复摆放几何体nReplicas次。
- ❄ 用户需要实现一个自己的参数化几何类，以G4VPVParameterisation为基类，并实现下面的函数 (依赖于几何体编号):
 - where it is positioned (transformation, rotation)
 - the size of the solid (dimensions)
 - The type of the solid, material, sensitivity, vis attributes
- ❄ 对简单的CSG几何适用。

```
class G4VPVParameterisation
{
public:
    ... ..

    virtual void ComputeTransformation // position, rotation
        (const G4int copyNo, G4VPhysicalVolume* physVol) const;
    virtual void ComputeDimensions // size
        (G4Box& trackerLayer, const G4int copyNo,
         const G4VPhysicalVolume* physVol) const;

    ... ..

    virtual G4VSolid* ComputeSolid // shape
        (const G4int copyNo, G4VPhysicalVolume* physVol);
    virtual G4Material* ComputeMaterial // material, sensitivity, visAtt
        (const G4int copyNo, G4VPhysicalVolume* physVol,
         const G4VTouchable *parentTouch=0);
        // G4VTouchable should not be used for ordinary parameterization
};
```


B2bDetectorConstruction.cc

```

209  G4Tubs* chamberS
210      = new G4Tubs("tracker", 0, 100*cm, 100*cm, 0.*deg, 360.*deg);
211  fLogicChamber
212      = new G4LogicalVolume(chamberS, fChamberMaterial, "Chamber", 0, 0, 0);
213
214  G4double firstPosition = -trackerSize + chamberSpacing;
215  G4double firstLength   = trackerLength/10;
216  G4double lastLength    = trackerLength;
217
218  G4VPVParameterisation* chamberParam =
219      new B2bChamberParameterisation(
220          NbOfChambers,    // NoChambers
221          firstPosition,   // Z of center of first
222          chamberSpacing,  // Z spacing of centers
223          chamberWidth,    // chamber width
224          firstLength,     // initial length
225          lastLength);     // final length
226
227  // dummy value : kZAxis -- modified by parameterised volume
228
229  new G4PVParameterised("Chamber",    // their name
230                        fLogicChamber, // their logical volume
231                        trackerLV,     // Mother logical volume
232                        kZAxis,        // Are placed along this axis
233                        NbOfChambers,  // Number of chambers
234                        chamberParam,  // The parametrisation
235                        fCheckOverlaps); // checking overlaps

```

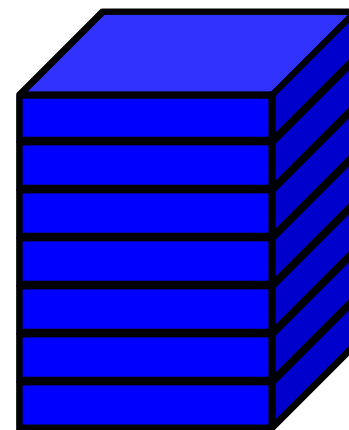
❄ 子几何体完全填满母体积，子体积具有相同的尺寸和形状。

❄ 重复摆放几何体时可以沿着下面的坐标轴进行：

- 笛卡尔坐标轴(X, Y, Z)
- 径向坐标轴(Rho) – 共心的圆柱/圆锥形状
- 方位角(Phi) – 扇形



a daughter
logical volume to
be replicated

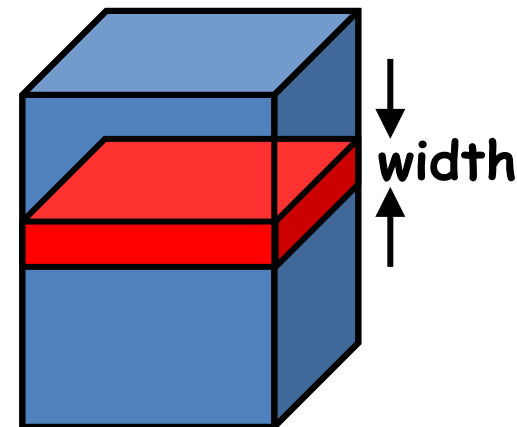


mother volume

❄ 笛卡尔坐标轴 - **kXaxis**, **kYaxis**, **kZaxis**

- 第n个子体积的中心在:

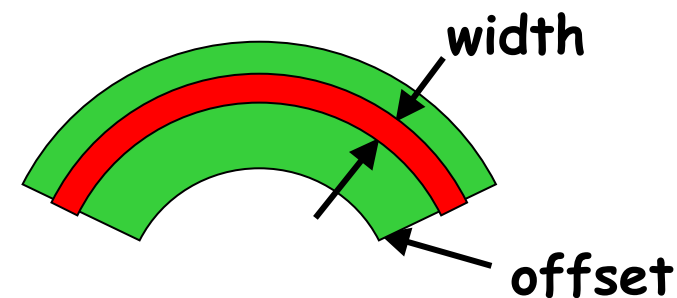
$$-width * (nReplicas - 1) * 0.5 + n * width$$
- Offset在这种情况下不起作用。



❄ 径向坐标轴 - **kRaxis**

- 第n个子体积的中心在:

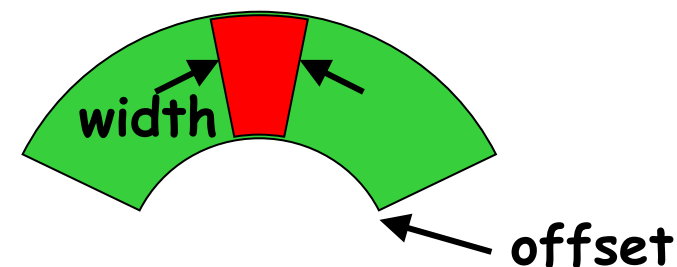
$$width * (n + 0.5) + offset$$
- Offset的值必须是母体积的内半径。



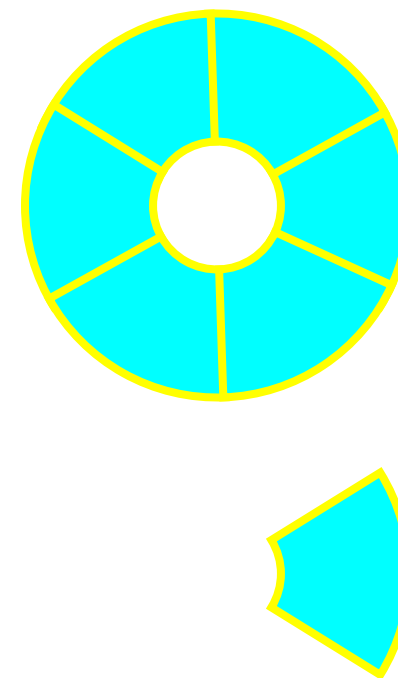
❄ Phi向坐标轴 - **kPhi**

- 第n个子体积的中心在:

$$width * (n + 0.5) + offset$$
- Offset必须是母体积的初始角度。



```
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);  
  
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/2.0);  
  
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);
```



利用N个微小的格子(G4Box)构建人体所在空间

⇒ G4PVReplica

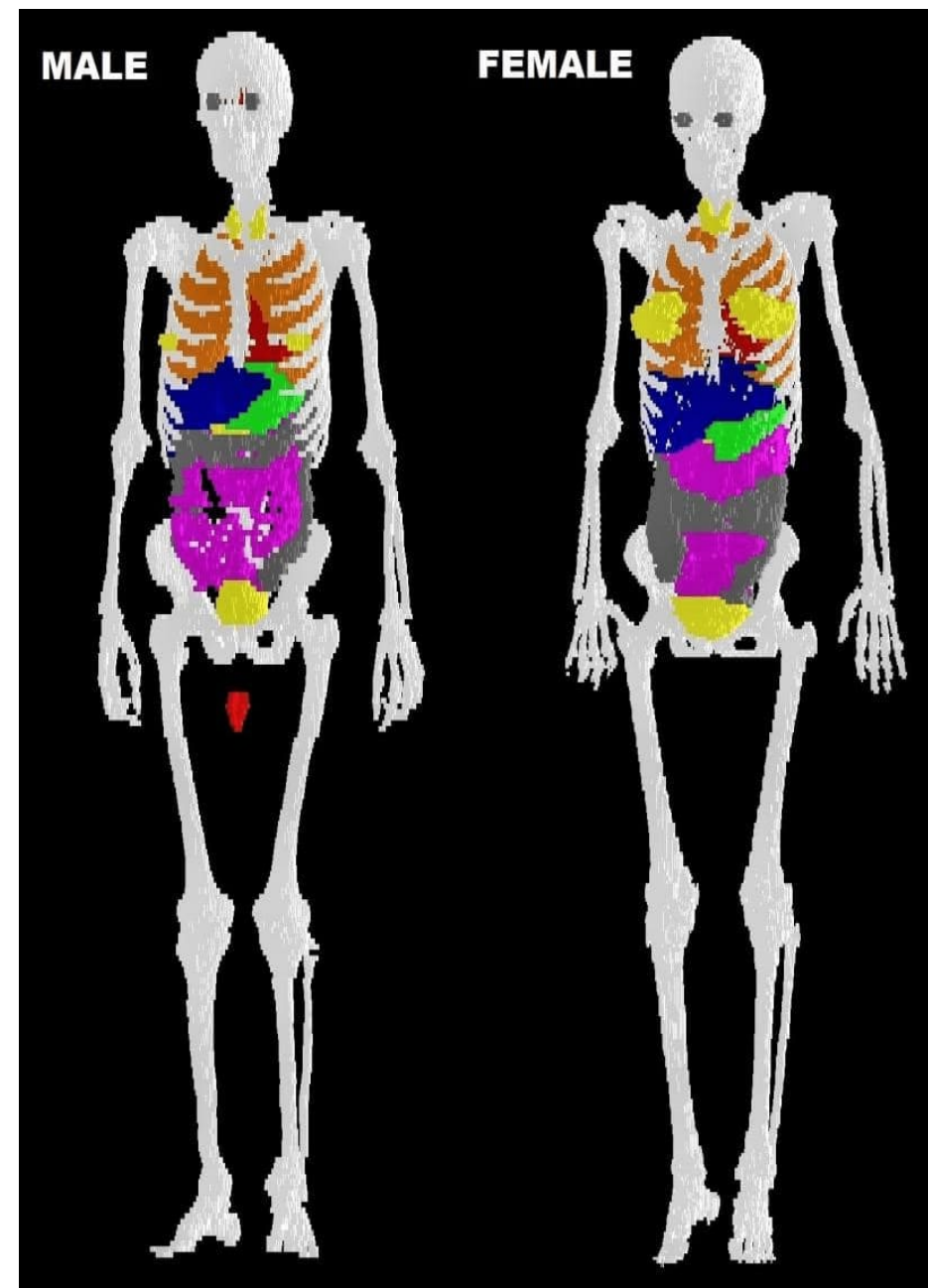
不同格子填充对应的物质

⇒ G4PVNestedParameterisation

需要实测数据

构建方法虽然简单，但占用内存空间较大

参考G4高级例子：[ICRP110_HumanPhantoms](#)



❄ 可以在ConstructSDandField()函数中实现

➤ World内均匀磁场

方式一:

```
G4ThreeVector field(0,1.*tesla,0);  
G4GlobalMagFieldMessenger* fMagFieldMessenger =  
new G4GlobalMagFieldMessenger(field)
```

方式二:

```
G4UniformMagField* magField = new G4UniformMagField(field);  
G4FieldManager* fieldMgr =  
G4TransportationManager::GetTransportationManager()  
->GetFieldManager();  
fieldMgr->SetDetectorField(magField);  
fieldMgr->CreateChordFinder(magField);
```

❄ 方式二中，用户可以设置Stepper的精度和数值积分所使用的方法

❄ 可以在ConstructSDandField()函数中实现

- 在World内定义非均匀磁场：创建用户的磁场类，从G4MagneticField类继承，实现GetFieldValue()函数

Geant4手册: <http://geant4-userdoc.web.cern.ch/geant4-userdoc/UsersGuides/ForApplicationDeveloper/html/Detector/electroMagneticField.html#an-overview-of-propagation-in-a-field>

Geant4例子: [examples/extended/field](#)

- 在某特定几何体内定义非均匀磁场:

```
MyField* myField = new MyField();  
G4FieldManager* localFieldMgr = new G4FieldManager(myField);  
G4bool allLocal = true;  
logicVolWithField ->SetFieldManager(localFieldMgr, allLocal);
```


需要在探测器构建类中实现!!!

```
G4VPhysicalVolume* volume1;
```

```
G4VPhysicalVolume* volume2;
```

```
G4OpticalSurface* OpSurface = new G4OpticalSurface("name");
```

```
G4LogicalBorderSurface* Surface = new G4LogicalBorderSurface("name",volume1,volume2,OpSurface);
```

```
G4double sigma_alpha = 0.1;
```

```
OpSurface -> SetType(dielectric_dielectric);
```

```
OpSurface -> SetModel(unified);
```

```
OpSurface -> SetFinish(groundbackpainted);
```

```
OpSurface -> SetSigmaAlpha(sigma_alpha);
```

```
const G4int NUM = 2;
```

```
G4double pp[NUM] = {2.038*eV, 4.144*eV};
```

```
G4double specularspike[NUM] = {1.0, 1.0};
```

```
G4double rindex[NUM] = {1.35, 1.40};
```

```
G4double reflectivity[NUM] = {0.3, 0.5};
```

```
G4double efficiency[NUM] = {0.8, 0.1};
```

```
G4MaterialPropertiesTable* SMPT = new G4MaterialPropertiesTable();
```

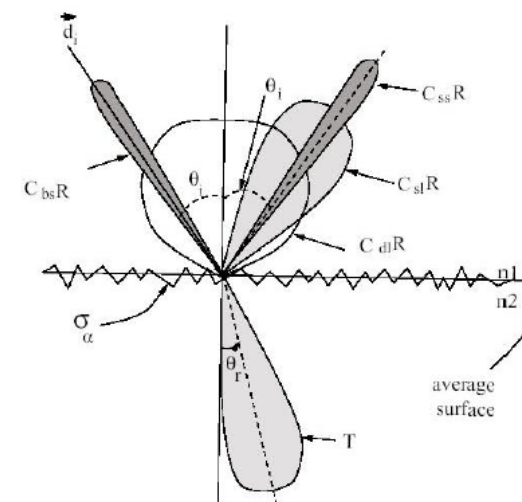
```
SMPT -> AddProperty("RINDEX",pp,rindex,NUM);
```

```
SMPT -> AddProperty("SPECULARSPIKECONSTANT",pp,specularlobe,NUM);
```

```
SMPT -> AddProperty("REFLECTIVITY",pp,reflectivity,NUM);
```

```
SMPT -> AddProperty("EFFICIENCY",pp,efficiency,NUM);
```

```
OpSurface -> SetMaterialPropertiesTable(SMPT);
```



Polar plot of the radiant intensity in the UNIFIED model

参考 “Section 5.2.5 of Geant4 Application Developers Guide ”