ジオメトリの定義

Geant4 10.3.P3準拠

Geant4　HEP/Space/Medicine 講習会資料

KEK 大学共同利用機関法人 高エネルギー加速器研究機構

本資料に関する注意

- 本資料の知的所有権は、高エネルギー加速器研究機構およびGeant4 collaborationが有します
- 以下のすべての条件を満たす場合に限り無料で利用することを許諾します
  - 学校、大学、公的研究機関等における教育および非軍事目的の研究開発のための利用であること
    - Geant4の開発者はいかなる軍事関連目的へのGeant4の利用を拒否します
  - このページを含むすべてのページをオリジナルのまま利用すること
    - 一部を抜き出して配布したり利用してはいけません
  - 誤字や間違いと疑われる点があれば報告する義務を負うこと
- 商業的な目的での利用、出版、電子ファイルの公開は許可なく行えません
- 本資料の最新版は以下からダウンロード可能です
  - http://geant4.kek.jp/lecture/
- 本資料に関する問い合わせ先は以下です
  - Email: lecture-feedback@geant4.kek.jp

目次

ジオメトリ定義の要点

言葉の定義

- ジオメトリ(*geometry*)
  - シミュレーションの対象となるシステム(例:実験室内の測定器システム、宇宙空間におかれた観測システム、など)の幾何学的な情報をさす
  - システムの幾何学情報は以下のボリューム(*volume*)を単位として記述される
- 物体あるいはボリューム(*volume*)
  - システムを構成する物体のエレメント(例:シリコン検出器センサ)を一般的にvolumeとよぶ
  - volumeはその幾何学形状、構成物質、配置(据付)位置などの情報をもつ
- ワールド・ボリューム(*World volume*)
  - システムを構成するすべてのvolumeを包含するvolume
  - システム内のボリューム配置情報の基本座標系となる

    [注] Geant4は右手座標系を採用している
- 親(*mother*)および子(*daughter*) volume
  - volume内に他のvolumeを配置することができ、前者を*mother volume*、後者を*daughter volume*とよぶ
- 立体(*solid*)
  - volumeの形と寸法の情報を持つ (物質の情報はもたない)
- 物質(*material*)
  - volumeを構成する物質情報 (化学組成、ガス、個体等の物理状態など)

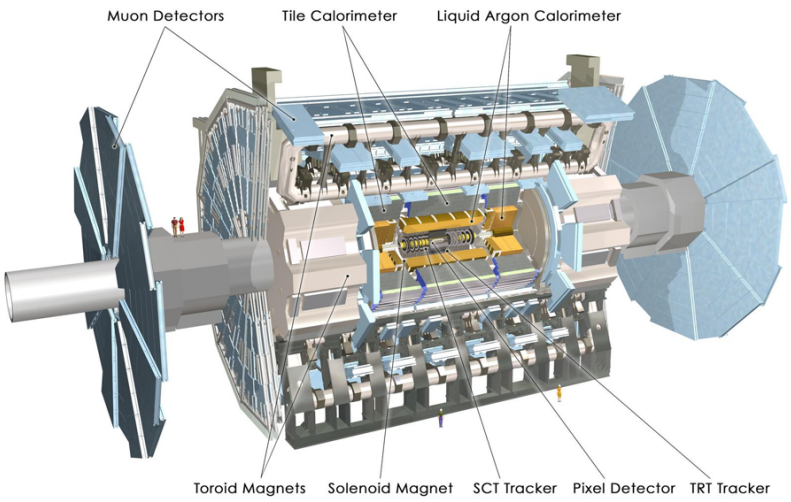
Geant4のジオメトリ定義方法を理解する鍵：*Logical Volume*

■ 複雑なシステムのシミュレーションでのジオメトリ定義の問題

- シミュレーション対象のシステムの構成エレメント数は場合によっては膨大となる
  - 例： LHC実験での測定器はエレメント数が~10^8のオーダー
- それぞれのエレメントを独立なvolumeで記述すると、コンピュタ・メモリのサイズおよびアクセスがボトルネックとなる可能性がある
- 従ってメモリを効率利用できるジオメトリ定義方法が重要

■ Geant4ジオメトリ定義法の基本設計思想

- 構成エレメント数が膨大なシステムでも多くの場合、同じエレメントが繰り返し使われる
  - 例： Atlas/LHC実験のPixel検出器
- 同一のエレメントが多数ある場合、エレメントを一つのvolumeとして定義し、それを共有することでメモリ消費を軽減できる
- Geant4ではこの共有volumeを*Logical Volume*とよび、ジオメトリ定義の基本となる



Geant4ジオメトリ定義では対象エレメントに対して、必ずLogical Volumeを作り、それをWorld volumeあるいは他のlogical volumeに配置する手法をとる

ジオメトリ定義の流れ

ジオメトリ定義にはGeant4が提供するジオメトリ・クラスを用いる

← 基本的な定義ステップは以下のとおり:

■ ステップ　1:　Solid定義

- Geant4が用意している立体(G4VSolid)の中から適当な形状を選択し、その寸法を設定

  [例]　直方体:2 x 3 x 5 m^3;　円筒:直径=1m、長さ=3m;　など

■ ステップ　2:　Logical Volume定義

- 上の立体に物質情報、有感情報、電磁場情報などを付加し、ボリューム(Volume)を作る
- このボリュームはシステムのステップ3でで繰り返し使うことができるもので Logical Volume (G4LogicalVolume)とよばれる
- Logical Volumeは空間配置されていないので、位置情報は持たない

  [例]　直方体のシリコン素子

■ ステップ　3:　Physical Volume定義

- 上で作られたLogical Volumeを物理的に空間に配置する
- 配置されたボリュームはPhysical Volume(G4PhysicalVolume)とよばれる

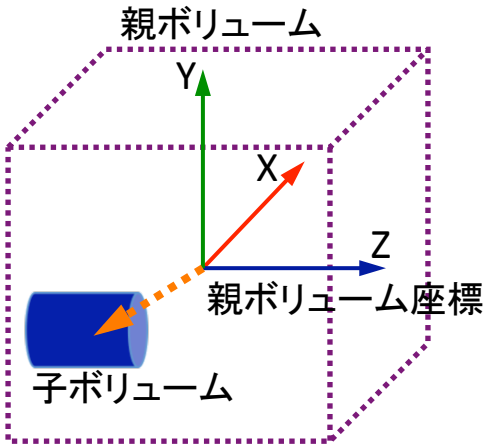
  [例]　実験室の中心に配置された直方体シリコン素子

ジオメトリ・クラスの使用例

■ ジオメトリ・クラスを使ったジオメトリ定義の具体例 ― 手続きの詳細は後述

Step1. Solid定義: 形状およびサイズを指定

```
G4VSolid* pBoxSolid =
  new G4Box("aBoxSolid", 1.*m, 2.*m, 3.*m);
```



Step 2. Logical Volume定義: solidに物質情報などを付加

```
G4LogicalVolume* pBoxLog =
  new G4LogicalVolume( pBoxSolid,
      pBoxMaterial, "aBoxLog", 0, 0, 0);
```

Step 3. Physical Volume定義: logical volumeに位置、回転情報などを付加

```
G4VPhysicalVolume* aBoxPhys =
  new G4PVPlacement( pTrans3D, pBoxLog, "aBoxPhys", pMotherLog, 0, copyNo);
```

- ボリューム (*Volume*)は親ボリューム(*Mother Volume*)の中に配置される
- 配置を表す回転、位置情報は親物体の固有座標系を用いて表現される
- 親物体の固有座標系は親物体を構成するsolidが保持する座標系に一致
- 配置された物体は子物体(*Daughter Volume*)ともよばれる

[配置原則] – 物体は親物体から飛び出して配置しない
– 物体同士を交差させて配置しない

ジオメトリ定義の作成

- 基本的な手続きは以下のようになる:
  1） G4VUserDetectorConstructionクラスを継承しユーザのジオメトリ定義クラスを作成
  2） ジオメトリ定義クラスのオブジェクトを作成してG4RunManagerに直接登録

ヘッダーファイル

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Geometry.hh
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#ifndef Geometry_h
#define Geometry_h 1

#include "G4VUserDetectorConstruction.hh"
class G4VPhysicalVolume;

//-------------------------------------------------------------------
  class Geometry : public G4VUserDetectorConstruction
//-------------------------------------------------------------------
{
  public:
    Geometry();
   ~Geometry();

    G4VPhysicalVolume* Construct();

  private:
    G4VPhysicalVolume* ConstructDetector();
};
#endif
```

ユーザのジオメトリ定義クラス（名前は任意）

Geant4が提供するジオメトリ定義用の基底クラス

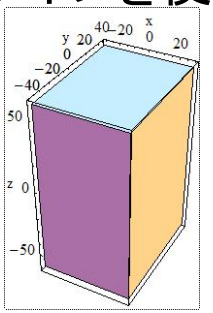
ジオメトリ定義の記述はこの関数に書く

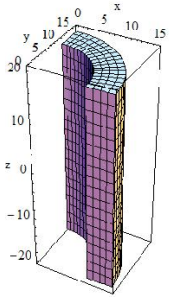
実装の詳細はHands-Onで学ぶ

Step 1: Solids（立体）の使い方

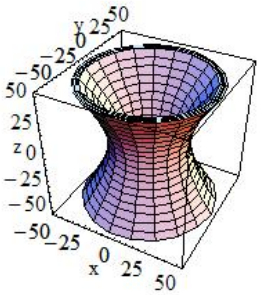
Step 1: Solidの定義 – Geant4が提供する立体

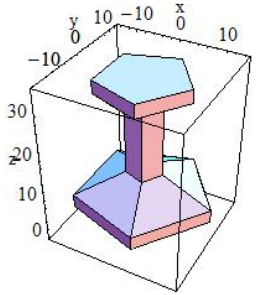
- Geant4では立体(*solid*)を表現するのに以下のタイプを使用する:
  - CSG(Constructive Solid Geometry)立体
    - G4Box, G4Tubs, G4Cons, G4Trd,…
    - 単純な形状の立体表現



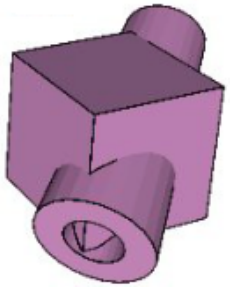


  - 特殊形状立体(CSGと同じ性質)
    - G4Polycone, G4Polyhedra, G4Hype, ...

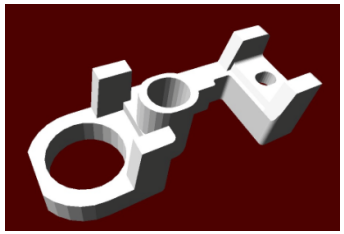




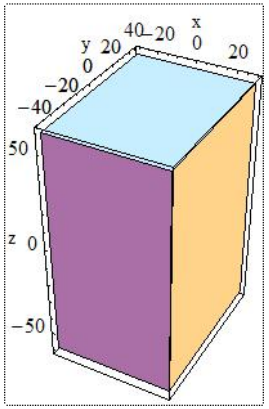
  - Boolean(ブール演算)で表現出来る立体
    - G4UnionSolid, G4IntersectionSolid, G4SubtractionSolid



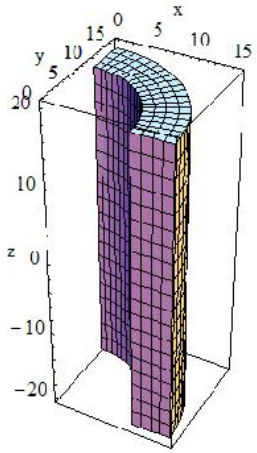
  - 多数の切子面で出来たCAD向け形状
    - G4TesselatedSolid



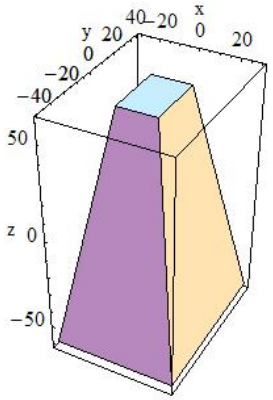
Step 1: Solidの定義 – CSG立体の種類



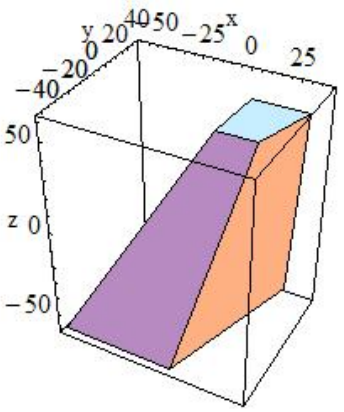
G4Box



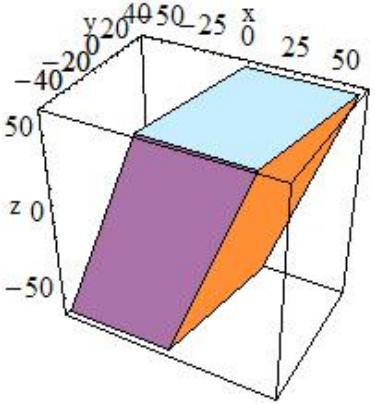
G4Tubs



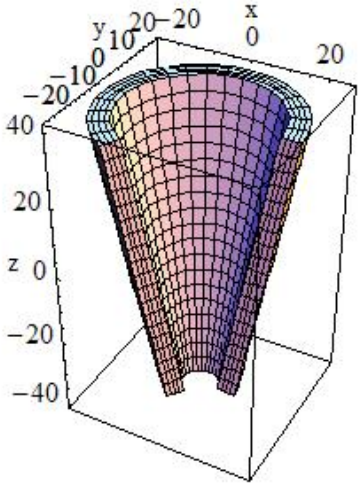
G4Trd(trapezoid)



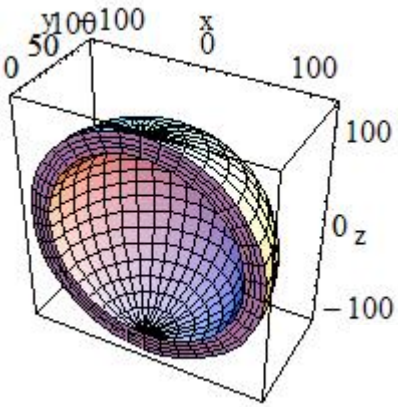
G4Trap (generic trapezoid)



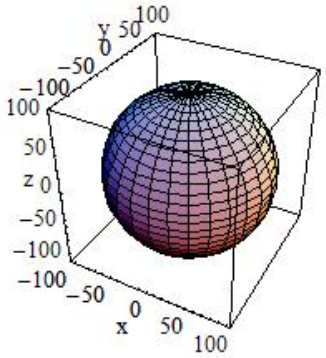
G4Para (parallelepiped)



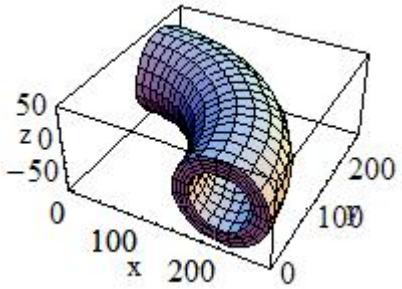
G4Cons



G4Sphere



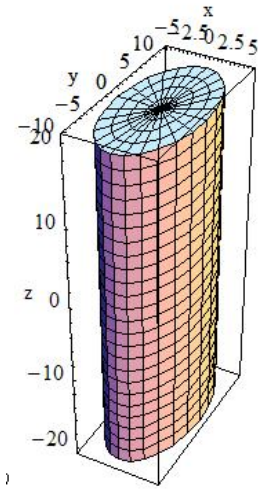
G4Orb（中空でない球体）



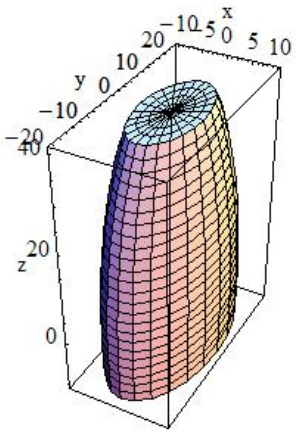
G4Torus

ユーザマニュアルのGeant4 Application Developers Guide 4.1.2章に全てのCSGがまとめられている

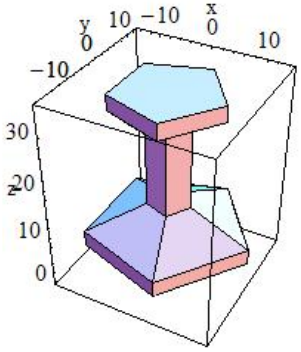
Step 1: Solidの定義 – 特殊形状CSG立体



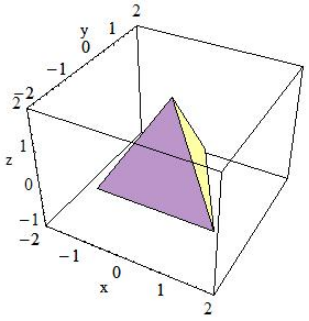
G4EllipticalTube



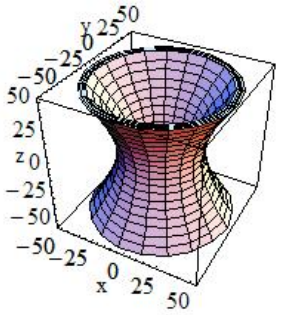
G4Ellipsoid



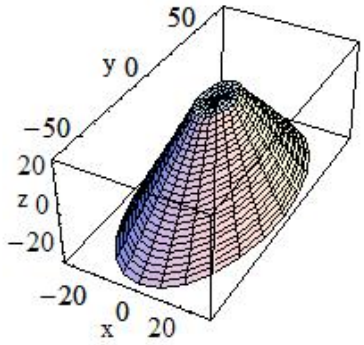
G4Polyhedra



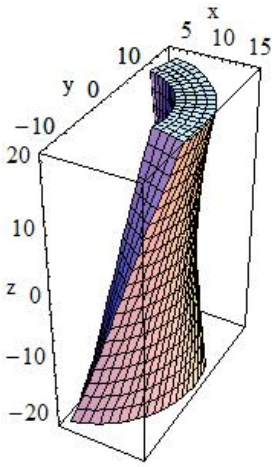
G4Tet
(tetrahedra)



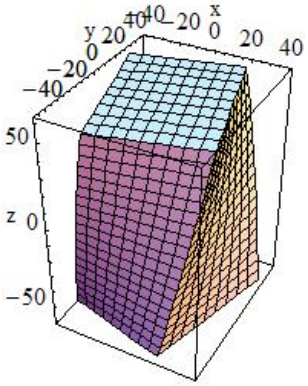
G4Hype



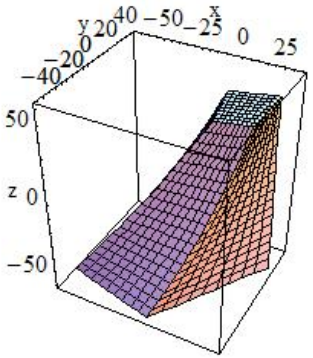
G4EllipticalCone



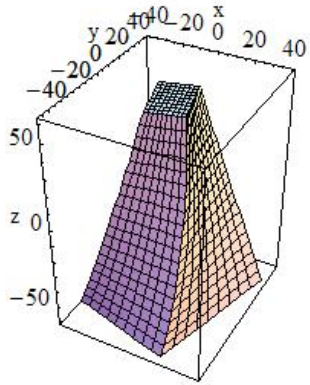
G4TwistedTubs



G4TwistedBox



G4TwistedTrap



G4TwistedTrd

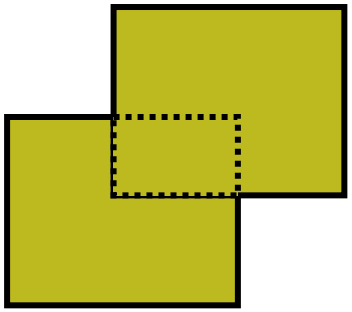
ユーザマニュアルのGeant4 Application Developers Guide 4.1.2章に全てのCSGがまとめられている

Step 1: Solidの定義 – ブール表現

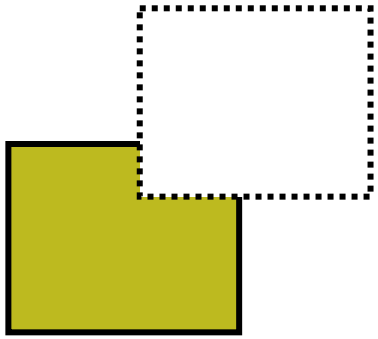
- 立体をブール演算で表現する事ができる
  - **G4UnionSolid, G4SubtractionSolid, G4IntersectionSolid**
  - 表現に必要なものは2つの立体とそれに対するブール演算
  - 2番目の立体は1番目の立体の空間座標系に対して用いて配置される
  - ブール演算の結果得られるたものは一つの立体(ブール立体)である。従って必要に応じて3番目の立体を演算する事が出来る
- 演算に使える立体は基本的にCSGかあるいはブール立体である

  [注] ブール立体はCSG立体に比べて粒子trackingの時間が長くなることに注意

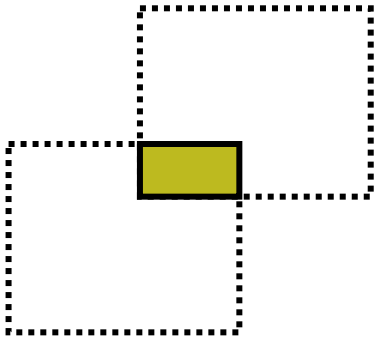
G4UnionSolid



G4SubtractionSolid



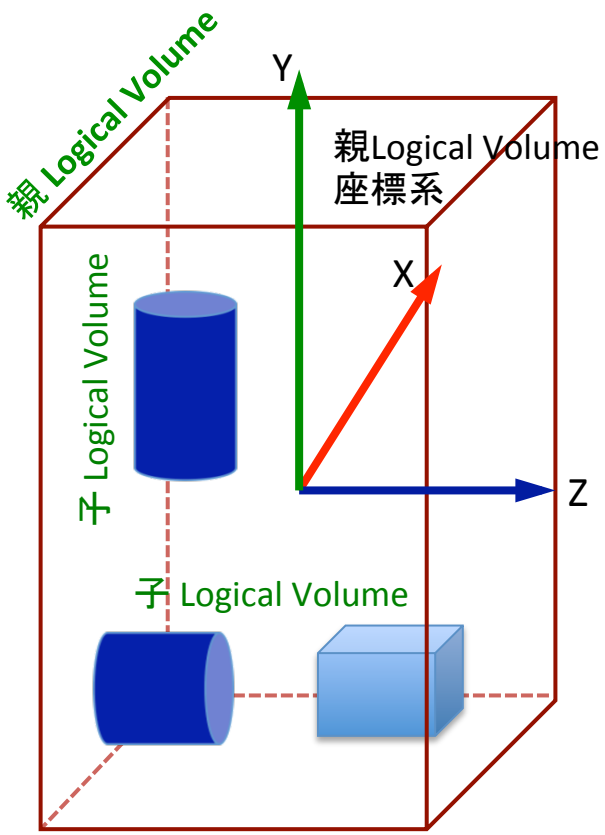
G4IntersectionSolid



Step 2: Logical Volumeの使い方

Step 2: Logical Volumeの定義 – Volumeの階層性

- *Logical Volume*で定義された物体は幾度も使う事が可能
- 一つのlogical volume（親物体）の中に複数のlogical volumes（子物体）を配置できる
- 親物体は子物体の*G4LogicalVolume*へのポインタを保持
  - しかし、子物体は親物体の情報(ポインタ)を持っていない
  - 親物体を他の物体の中に配置すると、親物体に含まれている物体（子物体）も同時に配置される
- ***ワールド*(*World Volume*)**は親子関係の最上位にあり、他の全ての物体はその中に含まれる形で配置される
  - ‘*ワールド*’ が保持する座標系は*global coordinate system*(グローバル座標系)とよばれ、ジオメトリの全体の基本座標系である
- 粒子の位置情報は*global coordinate system*で表現される
  - 位置情報を子物体の座標系に変換する手法は用意されている



[注]
子volumeが配置されることで、
それが占める親volumeの部分は
子volumeに置き換わる

Step 2: Logical Volumeの定義 – G4LogicalVolumeクラス

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

■ G4LogicalVolumeオブジェクトが保持する情報

- 形状とサイズ (G4VSolid)　: 設定は必須
- 物質情報 (G4Material)　: 設定は必須 （volume中で粒子をtrackingさせない場合を除く）
- 名前 (G4String)
- 電磁場情報 (G4FieldManager)
- 有感度設定 (G4VSensitiveDetector)
- ユーザ設定条件 (G4UserLimits)
- ジオメトリの最適化手法のon/off (G4bool)
- 可視化情報(Visualization)　: これは関数SetVisAttributes()で与える

[注]

- このクラスは基底クラスとして使用は不可 （仮想クラスではない）

Step 2: Logical Volumeの定義 – 物質表現クラスの基礎

- Logical volumeを定義する際には物質を表現するクラスが必要となる
- G4Isotope: 一つのアイソトープを表現するクラス
  - 原子番号、質量数、モル質量、など
- G4Element: 一つの元素を表現するクラス
  - (有効)原子番号、(有効)質量数、(有効)原子量、シェル数、など
  - 含有アイソトープの情報(存在比: 天然存在度等)
  - Radiation length、イオン化パラメター、など
- G4Material: 物質の化学的・物理的性質を表現するクラス
  - 化学式、構成元素の数、質量比、など
  - 密度、状態(固体、液体、ガス)、温度、など
  - Radiation length, nuclear interaction length,など

[注]

シミュレーションの幾何学形状の定義で必要となるのはG4Materialのみ

- G4Element/G4IsotopeはG4Materialを作るときだけに使用される

Step 2: Logical Volumeの定義 – Geant4 NISTの使用

- ユーザが元素、物質等を自分で定義できるが、Geant4で標準で提供する *NIST 物質データベース*(National Institute of Standards and Technology, U.S.) を使うことを奨励する

- NISTデータベースの使い方

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

G4Element*  elm = manager->FindOrBuildElement("symb", G4bool iso);
G4Element*  elm = manager->FindOrBuildElement(G4int iZ, G4bool iso);
G4Material* mat = manager->FindOrBuildMaterial("name", G4bool iso);
G4double isotopeMass = manager->GetMass(G4int iZ, G4int N);
```

  - G4bool iso = true : 元素組成は自然界のアイソトープ比を用いる
  - G4bool iso = false : アイソトープを用いない

Step 2: Logical Volumeの定義 – 磁場の表現

- 磁場の表現は*G4MagneticField*を継承したクラスを実装しておこなう
- 一様磁場の場合 ：

  Geant4が標準に持っているG4UniformMagFieldクラスを用いる

```
G4MagneticField* magField =
  new G4UniformMagField(G4ThreeVector(1.*Tesla,0.,0.);
```

- 非一様磁場の場合 ：

  *G4MagneticField*クラスを継承して独自の磁場クラスを定義し、そのGetFieldValueメソードを実装する

```
class MyField : G4MagneticField {……};
……
void MyField::GetFieldValue(
const G4double Point[4], G4double *field) const {
……
}
```

[注]

- **Point**[0..2] は空間点(*global coordinate system*)、**Point**[3] は時間を表す
- **field**[0..2] は空間点に対応する磁場の返り値

Step 2: Logical Volumeの定義 – 磁場の組込み

- 磁場を表現しているG4MagneticFieldオブジェクトを構造体定義に組み込むには以下の二つの手法がある:
  - 空間全体(World Volume)に対して組込む ― Global fieldの定義
  - 構造体中の特定の物体内部に組込む ― Local fieldsの定義

    [注] local fieldとglobal fieldは共存できる。共存させるとlocal fieldが優先

- Global Fieldの定義方法
  - G4TransportationManagerが独自に保持しているG4FieldManagerにG4MagneticFieldオブジェクトを渡す

```
G4FieldManager* globalFieldMgr =
   G4TransportationManager::GetTransportationManager()->GetFieldManager();
globalFieldMgr->SetDetectorField(magField);
globalFieldMgr->CreateChordFinder(magField);
```

使用例はHands-Onで学ぶ

Step 2: Logical Volumeの定義 – 磁場の組込み (つづき)

- Local Fieldの定義方法
  - 新たにG4FieldManagerを作る ー コンストラクタにlocal fieldを表現するG4MagneticFieldオブジェクトを渡す
  - 磁場を設定したいlogical volumeに上記のG4FieldManagerを渡す

```
G4FieldManager* localFieldMgr
  = new G4FieldManager(magField);

logVolume->setFieldManager(localFieldMgr, true);
```

[注]

- 構造物の中の任意の物体内に磁場を設定できる
- local fieldが設定されればglobal fieldがは無視される
- 上記のコードにある'setFieldManager()'の第二引数の意味
  - `true`: 無条件に子物体に対しても定義された磁場伝搬する
  - `false`: `G4FieldManager`を持っていない子物体のみに磁場が伝搬

Step 2: Logical Volumeの定義 – Sensitive Detector

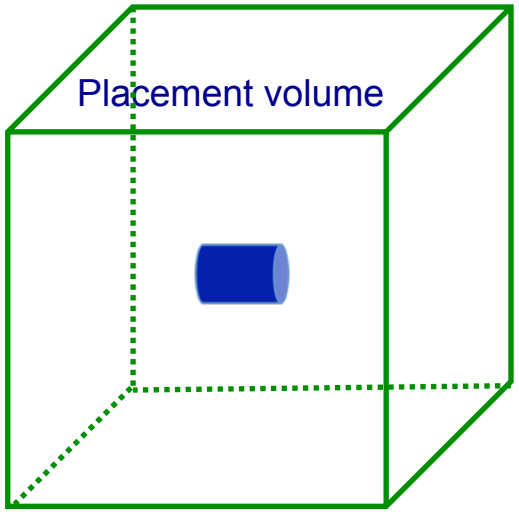
- Sensitive Detectorとは
  - 一般的にジオメトリ定義で作るボリュームは以下の二種類に分類できる
    - 検出器のように粒子が入ると何らかの情報・信号を返すもの ➔ active volume
    - 粒子が入っても情報・信号を返さないもの ➔ passive volume
  - Sensitive Detector(*G4VSensitiveDetector*)はactive volumeを表現するために用いる
    - logical volumeにsensitive detectorを表現するオブジェクトへのポインターを設定する
    - 粒子がこのボリュームに入射すると、sensitive detectorオブジェクトが持つ関数がよばれ、その関数で定義されている処理をおこなう
- G4VSensitiveDetectorクラス
  - 基本的にはG4UserSteppingActionと同じで、ユーザがtrackingの各ステップごとに何らかの処理をしたい場合に使う
    - G4SteppingActionとの違い： 粒子がsensitive detector内にいる時だけよばれる

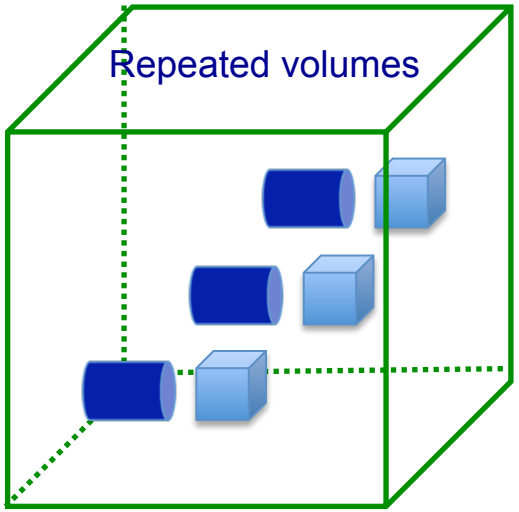
使用例はHands-Onで学ぶ

Step 3: Physical Volumeの使い方

Physical Volumeとは

- Physical Volumeとは親logical volume内に配置された子logical volumeをさす
  - 子volumeが置かれた部分のvolume情報（物質など）は親volumeでなく、子volumeのものに置きかわる
  - 親volumeは子volumeの位置、回転の情報を保持
- 配置には以下の制限がある：
  - 子物体は親物体からはみ出してはいけない
  - 子物体同士は互いに重なり合ってはいけない
- Physical Volumeを作る：　大きく分けて二つの手法
  - Placement volume：親物体空間に一度だけ物体を置く
    - 一つのphysical volumeオブジェクトは、親物体空間に配置された一つの物体に対応している
  - Repeated volumes：親物体空間に複数回、物体を置く
    - 一つのphysical volumeオブジェクトが親物体空間に置かれた任意の個数のvolumesに対応する
    - 以下の手法がある：関数表現をすることで柔軟に物体配置ができる
      - レプリカ
      - パラメター表現 (*parameterization*)
    - この表現を用いることでメモリーを節約できる





G4PVPlacement

■ G4PVPlacementのコンストラクターは以下のとおり

```
G4PVPlacement(
  const G4Transform3D &Transform3D,        // 子物体配置の回転・並行情報を表すオブジェクト
  G4LogicalVolume *pLogical,               // 子物体のlogical volumeへのポンタ
  const G4String  &pName,                  // 配置された後の子物体の名前(配置の名前)
  G4LogicalVolume *pMotherLogical,         // 親物体(配置先)logical volumeへのポインター
  G4bool pMany,                            // この引数は現在、使われていない
  G4int  pCopyNo,                          // コピー番号(下記[注 1]参照)
  G4bool pSurfChk=false);                  // 物体が重なって配置されているかのチェック・フラグ
```

[注 1] コピー番号について

- 整数値を設定することで、G4PVPlacementによるlogical volume配置にコピー番号(配置したlogical volumeの*ID*)を付加できる ← 使わないなら全てゼロとしてOK
- コピー番号は重複を許すが、配置IDとして使用したいならば異なった値にする

[注 2] G4PVPlacementのコンストラクタの種類

- 上に示したコンストラクタ以外に幾つかのシグネチャーの異なるコンストラクタがある
- 子物体を親物体のphysical volumeに配置したい場合、以下のコンストラクタを使う (ワールドへの配置等)

```
G4PVPlacement(
  const G4Transform3D &Transform3D,        // 子物体配置の回転・並行情報を表すオブジェクト
  const G4String    &pName,                // 配置された後の子物体の名前(配置の名前)
  G4LogicalVolume   *pLogical,             // 子物体のlogical volumeへのポンタ
  G4VPhysicalVolume *pMotherPhys,          // 親物体(配置先)physical volumeへのポインター
  G4bool pMany,                            // この引数は現在、使われていない
  G4int  pCopyNo,                          // コピー番号(下記[注 1]参照)
  G4bool pSurfChk=false);                  // 物体が重なって配置されているかのチェック・フラグ
```

- Geant4の公式examplesでは以下のコンストラクタがよく使われる

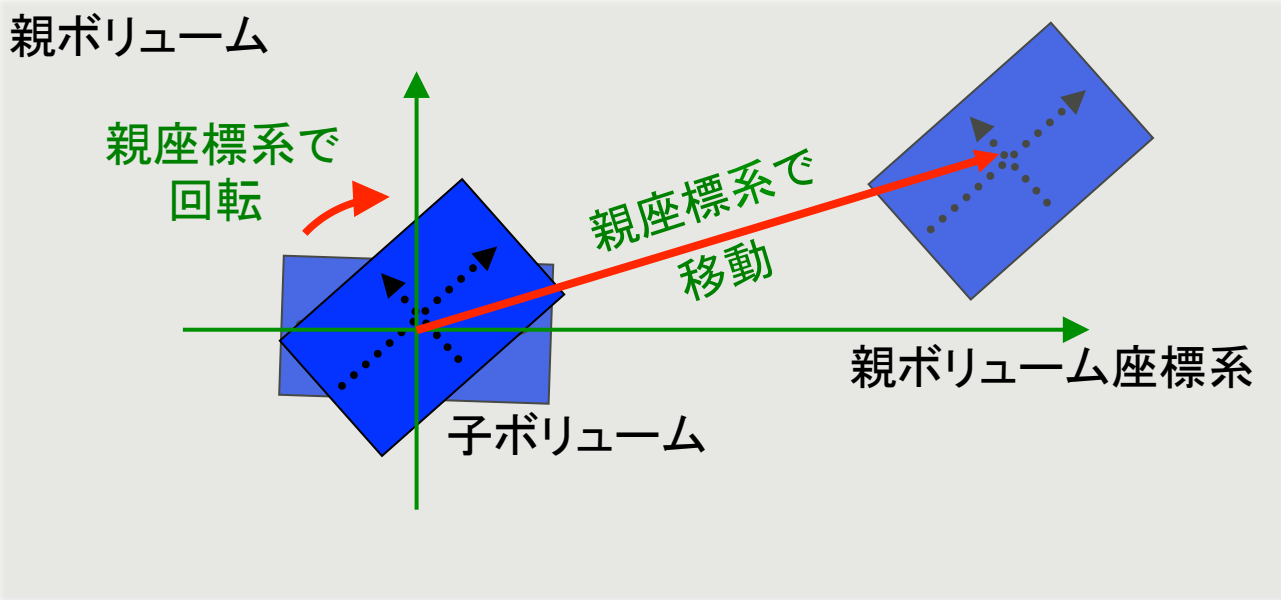
```
G4PVPlacement(G4RotationMatrix *pRot, const G4ThreeVector &tlate, G4LogicalVolume……)
```

しかし、このコンストラクタは回転配置の表現で混乱を招く恐れがある ➔ 上のコンストラクタ使用を推奨

Physical Volumeの作成

- G4PVPlacement（前スライドのコンストラクタ）を使った場合のvolume配置表現
  - 子物体を親座標系で回転
  - 親物体の座標系で子物体を空間移動

    ［注］ 回転と移動は独立であるから操作順序は考えなくて良い



G4Transform3Dについて

- *G4Transform3D*は物体の3次元座標系での回転と並行移動を表現するクラス
  - 回転は*G4RotationMatrix*クラスを用いて表現
  - 並行移動は*G4ThreeVector*クラスを用いて表現
  - CLHEP（CERNが開発したHEP用のC++ライブラリ）の*Transform3D*クラスの'typedef'

```
typedef HepGeom::Transform3D G4Transform3D;
```

    - Transform3Dクラスの定義ファイルはGeant4配布コードの以下にある：
      Geant4/externals/clhep/include/CLHEP/Geometry/Transform3D.h
- *G4RotationMatrix*は物体の3次元座標系での回転を表現するクラス
  - CLHEPの*HepRotation*クラスの'typedef'

```
typedef CLHEP::HepRotation G4RotationMatrix;
```

    - *HepRotation*クラスの定義ファイルはGeant4配布コードの以下にある：
      Geant4/externals/clhep/include/CLHEP/Vector/Rotation.h
- *G4ThreeVector*は3次元ベクトルを表現するクラス
  - CLHEPの*Hep3Vector*クラスの' typedef'

```
typedef CLHEP::Hep3Vector G4ThreeVector;
```

    - *Hep3Vector*クラスの定義ファイルはGeant4配布コードの以下にある：
      Geant4/externals/clhep/include/CLHEP/Vector/ThreeVector.h

[注] これらのクラスの使用例を次に示すが、使い方の詳細は上に示した定義ファイルを参照

G4Transform3D・G4RotationMatrix・G4ThreeVectorの使用例

■ *G4ThreeVector* (#include "G4ThreeVector.hh")

```
G4ThreeVector v1(1.0, 2.0, 3.0);
G4cout << v1.x() << ", " << v1.y() << ", " << v1.z() << G4endl;
G4cout << v1.theta() << ", " << v1.phi() << ", " << v1.mag() << G4endl;
G4ThreeVector v2;
v2.setX(10.); v2.setY(20.); v2.setZ(30.);
G4cout << v2.x() << ", " << v2.y() << ", " << v2.z() << G4endl;
```

コンストラクタ
x，y，z値
theta，phi値と絶対サイズ
コンストラクタ
x，y，z値　設定

■ *G4RotationMatrix* (#include "G4RotationMatrix.hh")

```
G4RotationMatrix rot1;
rot1.rotateY(10.0*deg);    // rotation in Y: 10.0 deg
rot1.rotateY(20.0*deg);    // rotation in Y: 20.0 deg (additional)

G4RotationMatrix rot2;
rot2.rotateY(10.0*deg);    // rotation in Y: 10.0 deg (first in Y-axis)
rot2.rotateZ(20.0*deg);    // rotation in Z: 20.0 deg (then  in Z-axis)

G4RotationMatrix rot3;
rot3= rot1 * rot2;         // multiply
rot3 = G4RotationMatrix(); // reset 'rot3'

G4cout << rot1.xx() << ", " << rot1.xy() << ", " << rot1.xz() << G4endl;
G4cout << rot1.yx() << ", " << rot1.yy() << ", " << rot1.yz() << G4endl;
G4cout << rot1.zx() << ", " << rot1.zy() << ", " << rot1.zz() << G4endl;
```

コンストラクタ
y軸で連続2回転する
マトリックス
コンストラクタ
y軸の次にZ軸で回転する
マトリックス
コンストラクタ
二つのマトリックスの積
rot3マトリックスの値をリセット
rot1マトリックスの各成分

■ *G4Transform3D* (#include "G4Transform3D.hh")

```
G4Transform3D trans(rot1, v1);  // transform3D using rot1 and v1
```

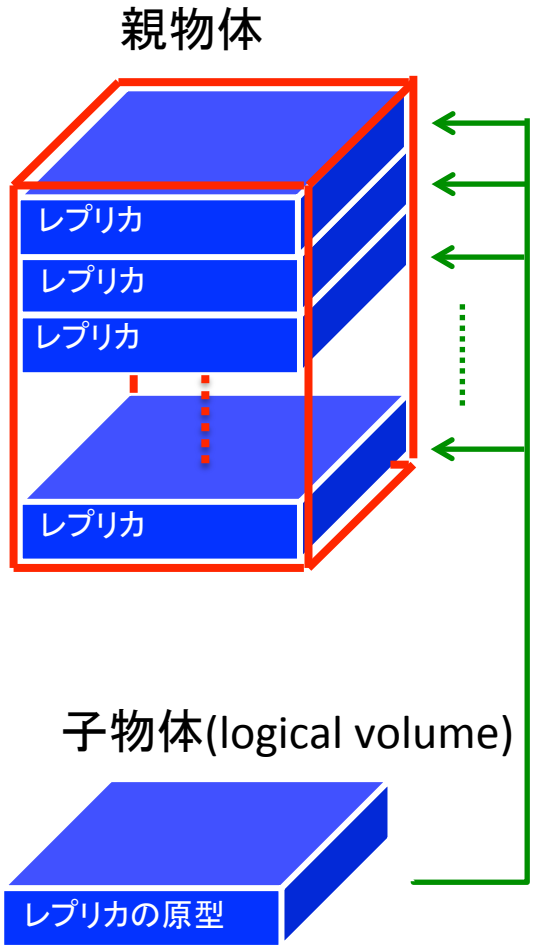
コンストラクタ

レプリカ(Replicated Volume)とは

- 親物体内部を子物体(レプリカ)によって完全に埋め尽くしたい時に使う。全てのレプリカは同一の大きさと形状を持っている
- 子物体を*G4PVPlacement*で一つずつ配置していくのと比べて、メモリー節約ができる
  - 特に大規模な分割では節約量は大きい

[注]

- レプリカを別のレプリカ内に配置できる
- レプリカ内に別物体をG4PVPlacementで配置できる。ただし、別物体は親物体や他のレプリカと交差／重複して配置できない
- パラメータ表現立体(Parameterised volumes、後述)はレプリカ内に配置は不可



G4PVReplicaクラス

■ レプリカを表現するクラス

```
G4PVReplica(const G4String &pName,
            G4LogicalVolume *pLogical,
            G4LogicalVolume *pMother,
            const EAxis pAxis,
            const G4int nReplicas,
            const G4double width,
            const G4double offset=0.);
```

子物体（レプリカ）

親物体

配置方向

［使用例］

■ X軸方向への配置

```
G4PVReplica repX("Linear Array”,pRepLogical, pContainingMother, kXAxis, 5, 10*mm);
```

■ 動径方向への配置

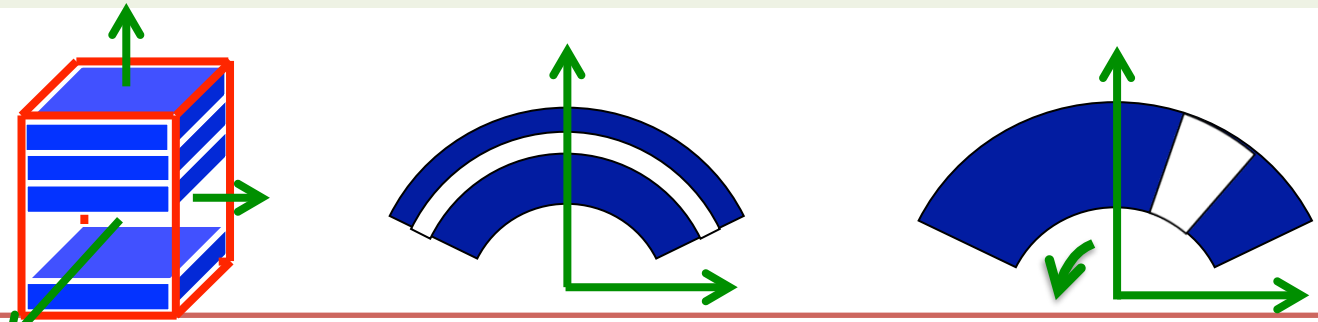
```
G4PVReplica repR("RSlices”, pRepRLogical, pContainingMother, kRho, 5, 10*mm, 0);
```

■ Z軸方向への配置

```
G4PVReplica repRZ("RZSlices”, pRepRZLogical, &repR, kZAxis, 5, 10*mm);
```

■ 極軸方向への配置

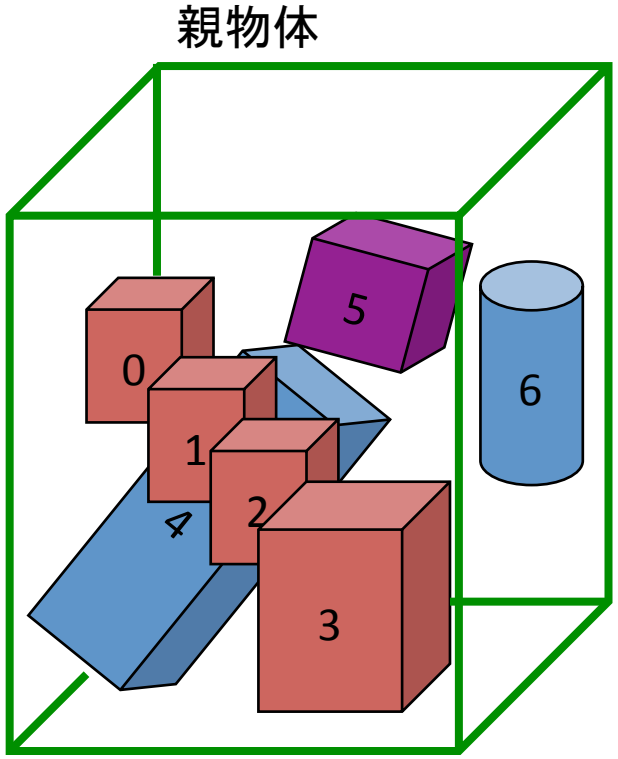
```
G4PVReplica repRZPhi("RZPhiSlices”, pRepRZPhiLogical, &repRZ, kPhi, 4, M_PI*0.5*rad, 0);
```



物体のパラメータ配置とは

- レプリカでは、繰り返し配置される物体には厳しい制限があった
- 一方、パラメータ配置では自由度の大きい配置が可能である
  - 物体のサイズを変化させて配置が可能
  - 物体の形状を変化させて配置が可能
  - 物体の物質を変化させて配置が可能
  - その他

- 典型的な応用例
  - 複雑な形状の測定器
    - 同一の立体が規則的／非規則的に並んでいるもの
  - 医学応用
    - 物質密度が異なる直方体として動物細胞を表現する



番号(CopyNumber)が
ふられているのが子物体

G4PVParameterisedクラス

- パラメータ配置物体を表すクラス

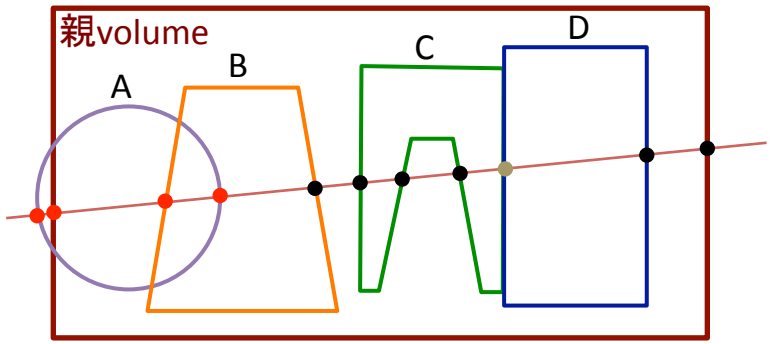
```
G4PVParameterised(const G4String& pName,
                  G4LogicalVolume* pLogical,
                  G4LogicalVolume* pMother,
                  const EAxis pAxis,
                  const G4int nReplicas,
                  G4VPVParameterisation* pParam);
```

子物体（レプリカ）
親物体
配置方向

- 親物体の中にG4VPVParameterisationクラス（pParam）で表現されるパラメータ配置に従って'nReplicas' 回、子物体を複製する
- 複製された一つ一つの物体にはゼロから始まる'*copy number*'が付加される
- G4VPVParameterisationクラスはユーザが実装する必要がある
  - 物体形状、サイズ、物質、空間配置は'*copy number*'をもとにしてパラメータ表現
- pAxisは複製子物体が主に配置される方向を与える。この情報はシミュレーションの高速化のために内部的に使用される
  - kXAxis, kYAxis, kZAxis ：それぞれX,Y,Z座標軸方向に並んでいる
  - kUndefined：特定の座標軸には並んでいない

Volumeの配置に関する制限事項とそのチェック

- G4PVPlacementあるいはレプリカ等を使ってvolumeを配置するときの制限事項:
  - 子volumeは親volumeからはみ出してはいけない
    - 右図でAは親volumeからはみ出している
  - 子物体同士は互いに重なり合ってはいけない
    - 右図でAとBは重なり合っている
    - CとDが重さるかどうかの判断は計算精度に依存



[注] volume同士が重なると、その重なり部分に属する空間点のvolume情報(例:物質情報)が確定できないので、シミュレーション結果が信頼できなくなる

- ユーザがジオメトリを作成したとき、上の制限事項に反したことしていないかをチェックするには以下の方法がある:
  - G4PVPlacementを用いてvolumeを配置するとき、コンストラクタの最後の引数である重なりチェック・フラグを'true'にする
    - アプリケーション開始時にチェックが行われ、制限事項に反していた場合メッセージを出力
  - アプリケーション開始後、以下のユーザコマンドを実行する:
    - `/geometry/test/run`
  - 以下の専用チェック・ツールを使用する:
    - DAVID
    - OLAP

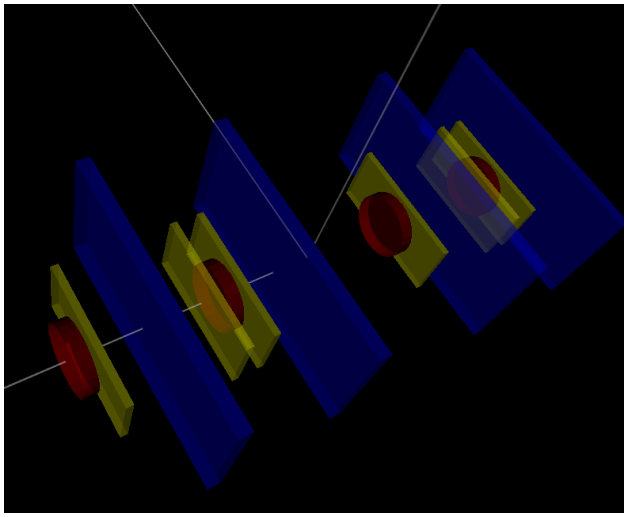
タッチャブル（Touchable）

タッチャブル(Touchable)とは

- 構造物の定義に於いて物体(logicalおよびphysical volume) は同一のもが再利用が可能:[注]参照
  - 物体が持っている固有情報(copy number等)だけでは、その物体が絶対空間(world volume)のどこに配置されているかは分からない
- 場所を特定するには最低限以下の情報が必要
  - 物体のworld volumeにいたる階層数
  - それぞれの階層での並進・回転マトリクス

[注]
- logical volume内に置かれたphysical volumeも再利用可能

- Touchable
  - 最低限、上記の情報を持つオブジェクトをGeant4では一般的にTouchableとよぶ
- G4VTouchableクラス
  - Touchableを表現するクラス
  - virtualクラスであるから、これを基底にして様々なTouchableクラスが作られる
    - 例: G4TouchableHistory (次頁参照)

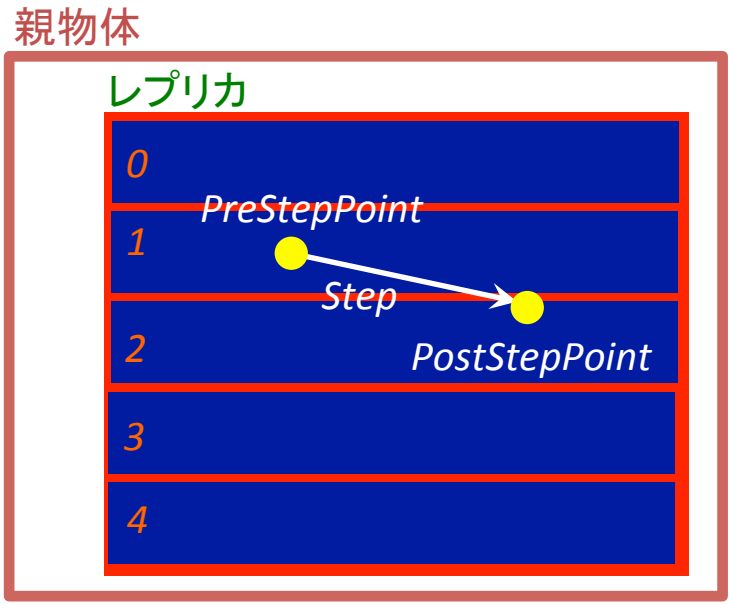


Touchable Historyとは

- ユーザが物体の場所を特定するのに通常用いることができるオブジェクトは Touchable Historyとよばれる
- このTouchable Historyは*G4TouchableHistory*クラスとして実装されている
- このオブジェクトはG4VTouchableクラスの派生クラスから作られており、以下の情報を保持している
  - 物体のworld volumeにいたる階層関係（volume history/hierarchy）
  - 各階層の物体へのポインター
  - 各階層での並進・回転マトリクス
  - 各階層の物体の配置手法（placement、parameterized 、replica、etc）
  - replica、parameterizationの詳細情報
- Touchable Historyはナビゲータ（後述）にリクエストを出すことで作られる
- Touchable Historyが保持する情報へのアクセスには、*Touchable Handle （スマートポインタ）*が用意されている
  - ユーザはこのHandleを通して情報アクセスを行う

Touchable Historyの使い方例

- 右図のような構造体の定義を考える
  - 親物体(box)の中に5分割のレプリカが配置
  - 親物体はworld volumeの中に配置
  - 親物体内にはただ一つの ’physical volume’(レプリカ)が存在するだけ
  - レプリカ内のboxはコピー番号を用いてパラメータ表現されている



- 上のレプリカ内の二つのboxにまたがる粒子のステップがあるとして、その始点と終点での幾何学関連の情報が入手したいとする:
  - この場合、正しい幾何情報を入手する手段は、“PreStepPoint”と“PostStepPoint”が保持している*Touchable History*に(handle通して)アクセスすること
  - 粒子のトラック情報にアクセスするだけでは“PreStepPoint”の情報を入手できない
- 具体的な手法は次のスライドを参照

Touchable History Handleの使い方例 ― つづき

■ 前述の構造定義で“PreStepPoint”と“PostStepPoint”の幾何学関連の情報を入手するコード例

```
G4StepPoint* point1 = step->GetPreStepPoint();          // point1はPreStepPoint
G4StepPoint* point2 = step->GetPostStepPoint();         // point2はPostStepPoint

G4TouchableHandle touch1 = point1->GetTouchableHandle();
G4TouchableHandle touch2 = point2->GetTouchableHandle();

G4VPhysicalVolume* volume = touch1->GetVolume();
G4VPhysicalVolume* volumeMt = touch1->GetVolume(1);    // 1階層上の物体(親物体)
G4VPhysicalVolume* volumeGM = touch1->GetVolume(2);    // 2階層上の物体
G4LogicalVolume* lVolume = volume->GetLogicalVolume();
G4Material* material = lVolume ->GetMaterial();

G4ThreeVector worldPosition = point1->GetPosition();  // point1の位置(絶対座標)
G4ThreeVector localPosition = touch1->GetHistory()->
   GetTopTransform().TransformPoint(worldPosition);    // point1の位置(ローカル座標)
```

[注]

- G4TouchableHandleはG4TouchableHistoryオブジェクトのhandle
- Touchable Historyを使った情報の入手方法の具体的なコード例は、ユーザドキュメント”Geant4 User‘s Guide for Application Developers” のFAQ.3 とFAQ.4を参照