Geometry I

Makoto Asai (SLAC)
Geant4 Tutorial Course

Contents

- Introduction

- G4VUserDetectorConstruction class

- Solid and shape

- Logical volume
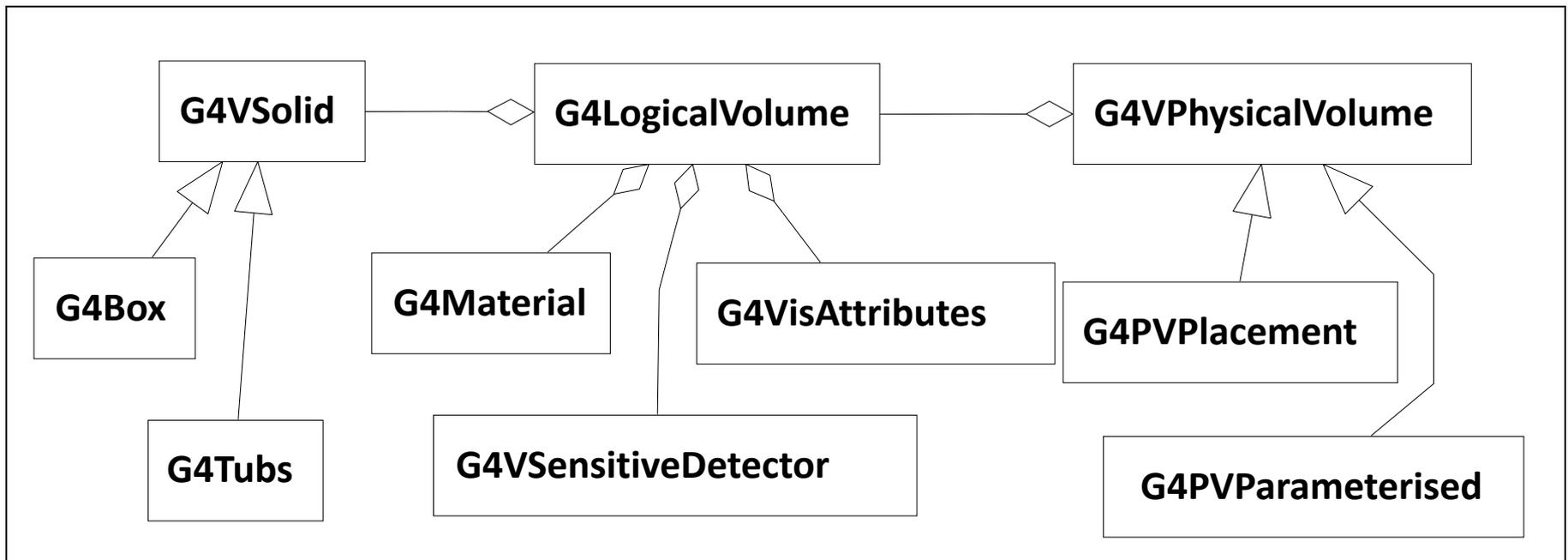
- Region

- Physical volume

- Placement

Introduction

Version 10.5

Detector geometry

- Three conceptual layers
  - G4VSolid -- *shape, size*
  - G4LogicalVolume -- *daughter physical volumes,*

    material, sensitivity, user limits, etc.
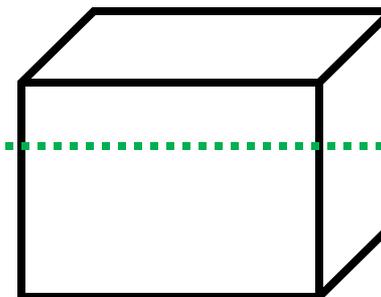 - G4VPhysicalVolume -- *position, rotation*

Define detector geometry

- Basic strategy

```
G4VSolid* pBoxSolid =

   new G4Box("aBoxSolid",

      1.*m, 2.*m, 3.*m);

G4LogicalVolume* pBoxLog =

   new G4LogicalVolume( pBoxSolid,

      pBoxMaterial, "aBoxLog", 0, 0, 0);

G4VPhysicalVolume* aBoxPhys =

   new G4PVPlacement( pRotation,

      G4ThreeVector(posX, posY, posZ),

      pBoxLog, "aBoxPhys", pMotherLog,

      0, copyNo);
```

Logical volume :
Solid : shape and size
+ material, sensitivity, etc.

Physical volume :
+ rotation and position

- A volume is placed in its mother volume. Position and rotation of the daughter volume is described with respect to the local coordinate system of the mother volume. The origin of mother volume's local coordinate system is at the center of the mother volume.

  - Daughter volume cannot protrude from mother volume.

Geometrical hierarchy

- One logical volume can be placed more than once. One or more volumes can be placed to a mother volume.

- Note that the mother-daughter relationship is an information of G4LogicalVolume.

  – If the mother volume is placed more than once, all daughters are by definition appear in all of mother physical volumes.

- The world volume must be a unique physical volume which fully contains all the other volumes.

  – The world volume defines the global coordinate system. The origin of the global coordinate system is at the center of the world volume.

  – Position of a track is given with respect to the global coordinate system.
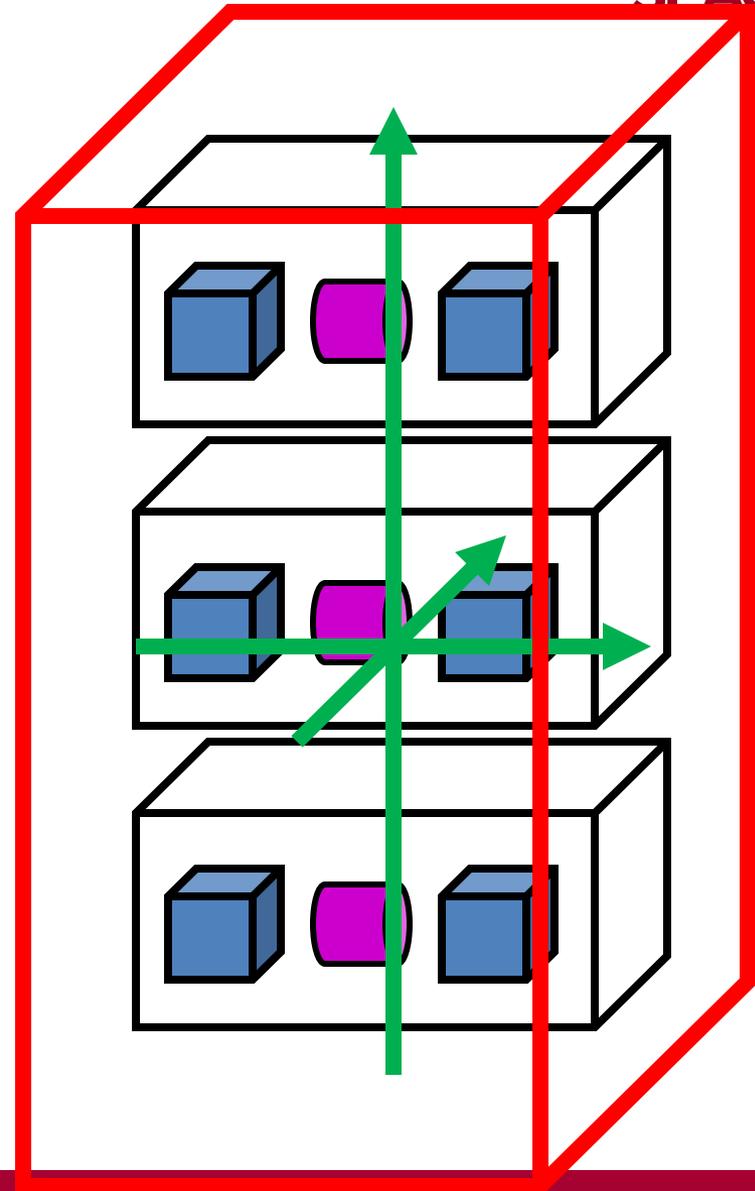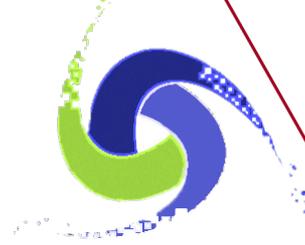
G4VUserDetectorConstruction

User classes

- main()
  - Geant4 does not provide *main().*

  Note : classes written in red are mandatory.

- Initialization classes
  - Use G4RunManager::SetUserInitialization() to define.
  - Invoked at the initialization
    - G4VUserDetectorConstruction ⬅
    - G4VUserPhysicsList
    - G4VUserActionInitialization

- Action classes
  - Instantiated in G4VUserActionInitialization.
  - Invoked during an event loop
    - G4VUserPrimaryGeneratorAction
    - G4UserRunAction
    - G4UserEventAction
    - G4UserStackingAction
    - G4UserTrackingAction
    - G4UserSteppingAction

G4VUserDetectorConstruction

```
class G4VUserDetectorConstruction
{
 public:
   G4VUserDetectorConstruction();
   virtual ~G4VUserDetectorConstruction();
 public:
   virtual G4VPhysicalVolume* Construct() = 0;
   virtual void ConstructSDandField();
 public:
   void RegisterParallelWorld(G4VUserParallelWorld*);
```

Construct() should return the pointer of the world physical volume. The world physical volume represents all of your geometry setup.

Sensitive detector and field should be instantiated and set to logical volumes in ConstructSDandField() method.

Your detector construction

```cpp
#ifndef MyDetctorConstruction_h
#define MyDetctorConstruction_h 1
#include "G4VUserDetectorConstruction.hh"
class MyDetctorConstruction
      : public G4VUserDetectorConstruction
{
 public:
  G4VUserDetectorConstruction();
  virtual ~G4VUserDetectorConstruction();
  virtual G4VPhysicalVolume* Construct();
  virtual void ConstructSDandField();
 public:
  // set/get methods if needed
 private:
  // granular private methods if needed
  // data members if needed
};
#endif
```

Describe your detector

- Derive your own concrete class from G4VUserDetectorConstruction abstract base class.

- Implement Construct() and ConstructSDandField() methods
    1) Construct all necessary materials
    2) Define shapes/solids
    3) Define logical volumes
    4) Place volumes of your detector geometry
    5) Associate (magnetic) field to geometry *(optional)*
    6) Instantiate sensitive detectors / scorers and set them to corresponding logical volumes *(optional)*
    7) Define visualization attributes for the detector elements *(optional)*
    8) Define regions *(optional)*
- Set your construction class to G4RunManager or G4MTRunManager
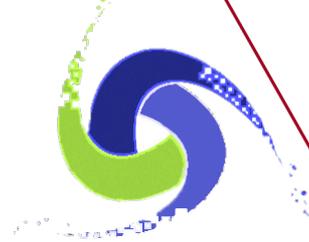
Solid and shape

G4VSolid

- Abstract class. All solids in Geant4 are derived from it.

- It defines but does not implement all functions required to:
  - compute distances between the shape and a given point
  - check whether a point is inside the shape
  - compute the extent of the shape
  - compute the surface normal to the shape at a given point

- User can create his/her own solid class.
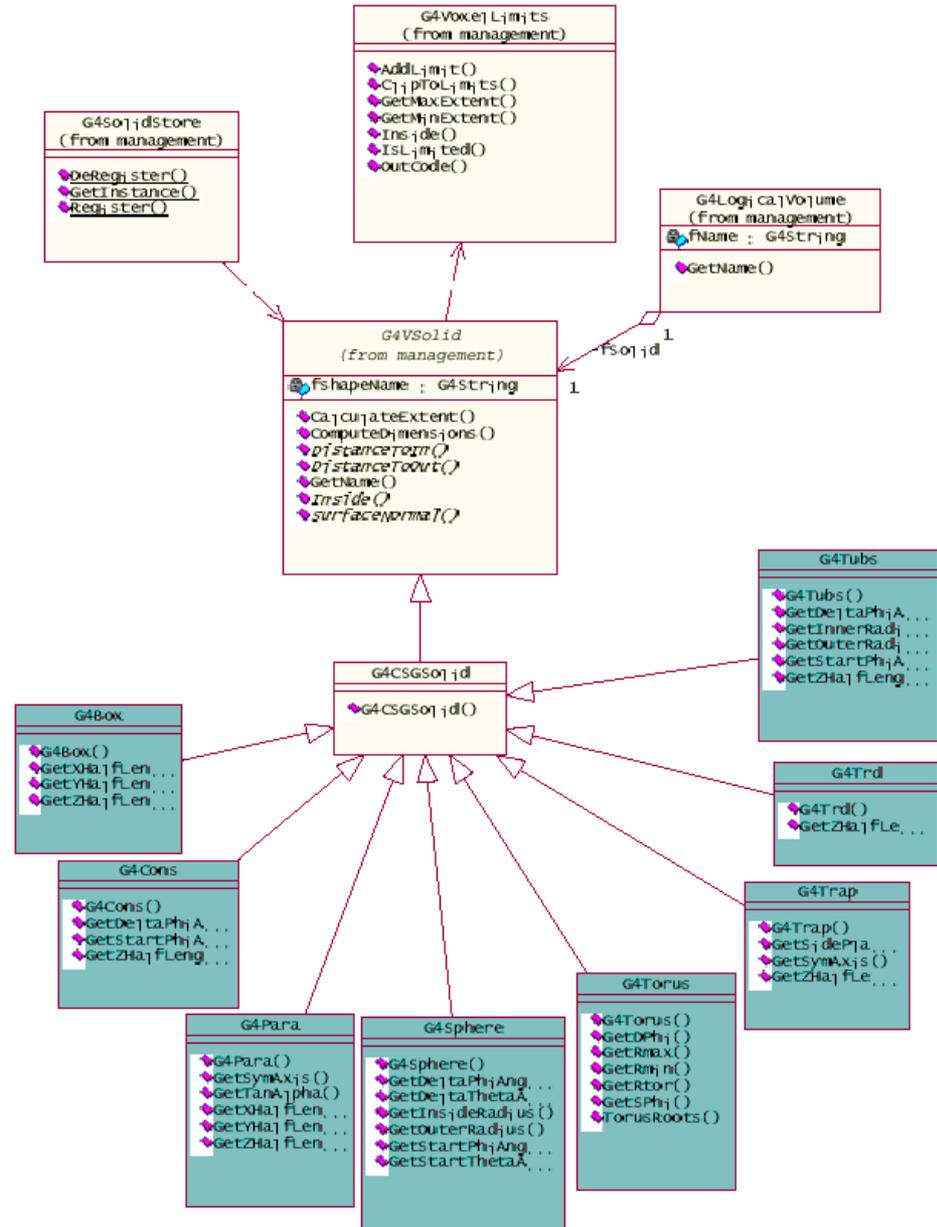
Solids

- Solids defined in Geant4:

    - CSG (Constructed Solid Geometry) solids

        - G4Box, G4Tubs, G4Cons, G4Trd, …

            - Analogous to simple GEANT3 CSG solids

    - Specific solids (CSG like)

        - G4Polycone, G4Polyhedra, G4Hype, …

    - Tessellated solid

        - Solid made by facets

    - Boolean solids

        - G4UnionSolid, G4SubtractionSolid, …
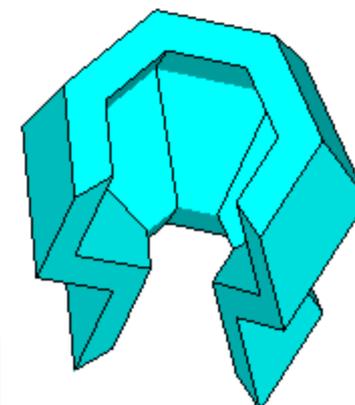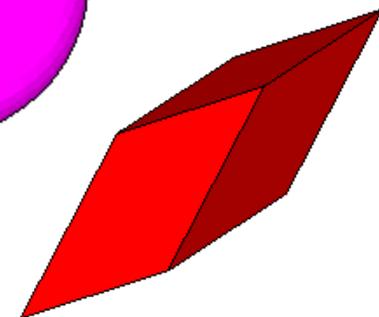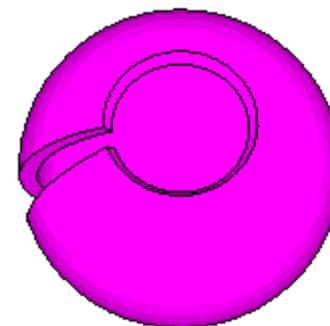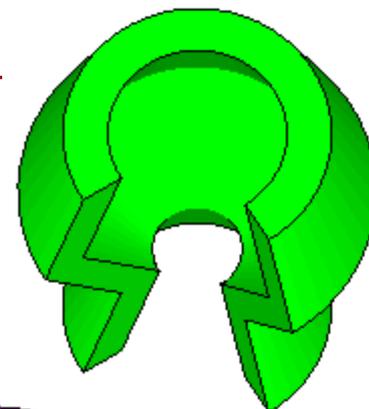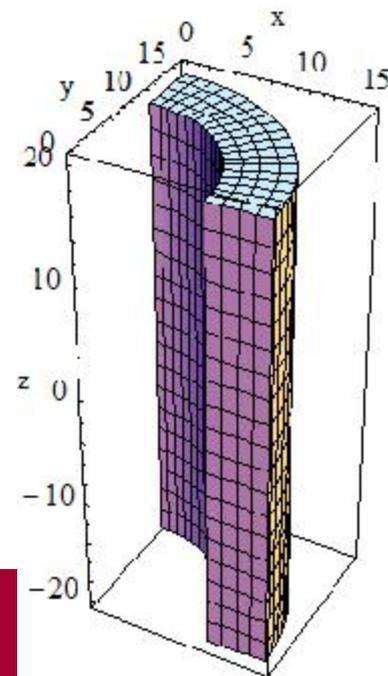
CSG: G4Box, G4Tubs
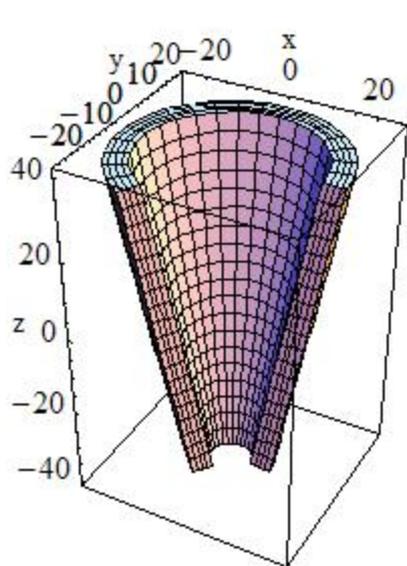
```
G4Box(const G4String &pname,   // name

          G4double half_x,   // X half size

          G4double half_y,   // Y half size

          G4double half_z);  // Z half size
```
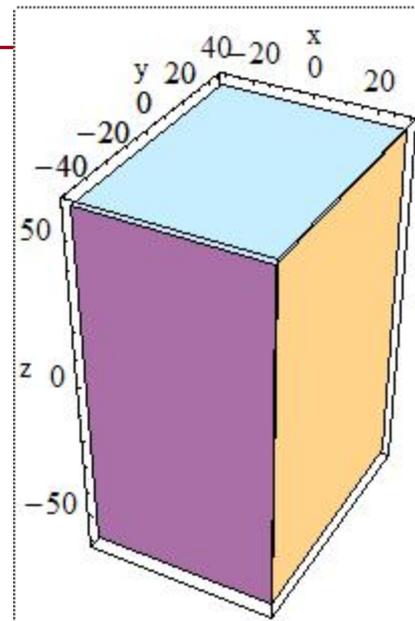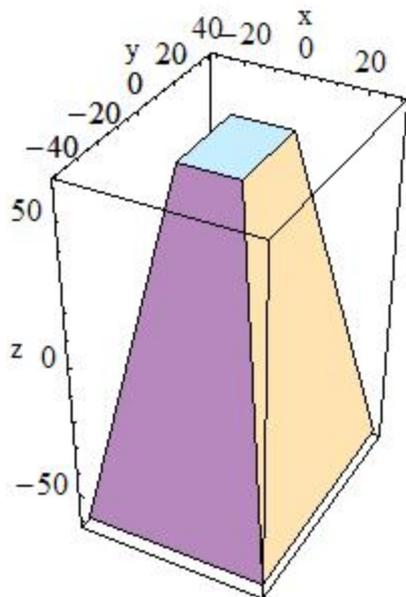


```
G4Tubs(const G4String &pname,   // name

          G4double  pRmin,   // inner radius

          G4double  pRmax,   // outer radius

          G4double  pDz,     // Z half length

          G4double  pSphi,   // starting Phi

          G4double  pDphi);  // segment angle
```
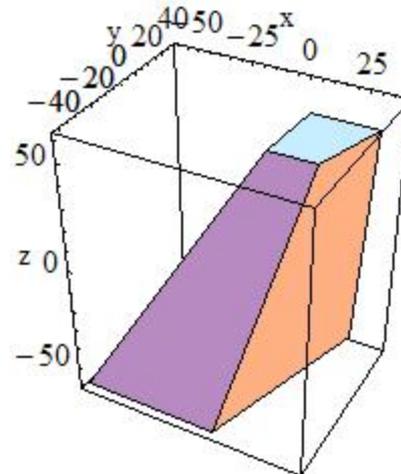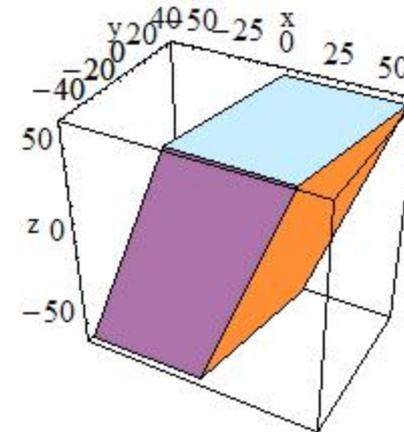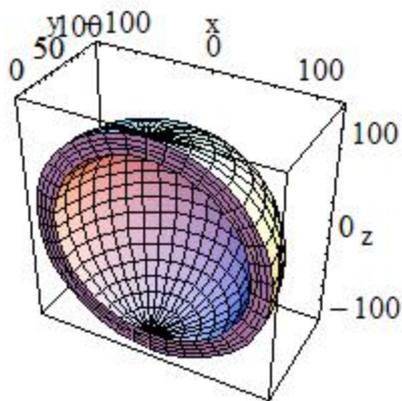
Other CSG solids

G4Cons

G4Trd

G4Trap

G4Para
(parallelepiped)

G4Torus
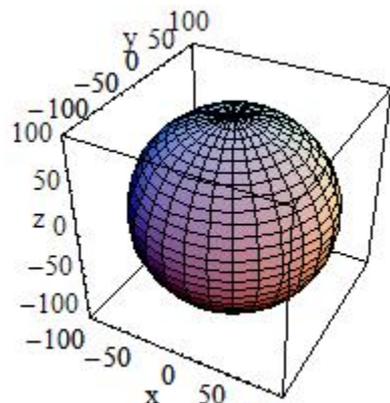
G4Sphere

G4Orb
(full solid sphere)
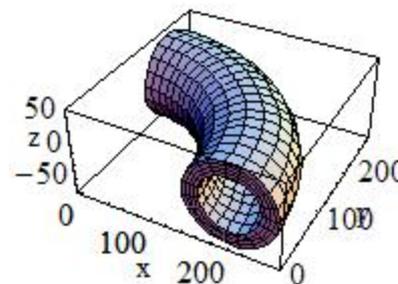
Consult to Section 4.1.2 of Geant4 Application Developers Guide for all available shapes.

```
G4Polycone(const G4String& pName,

            G4double phiStart,

            G4double phiTotal,

            G4int numRZ,

            const G4double r[],

            const G4double z[]);
```


- **numRZ** - numbers of corners in the **r,z** space

- **r, z** - coordinates of corners
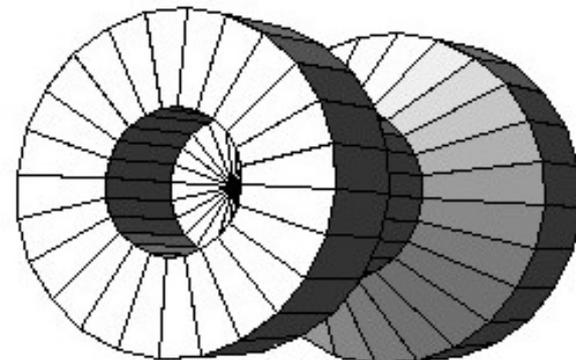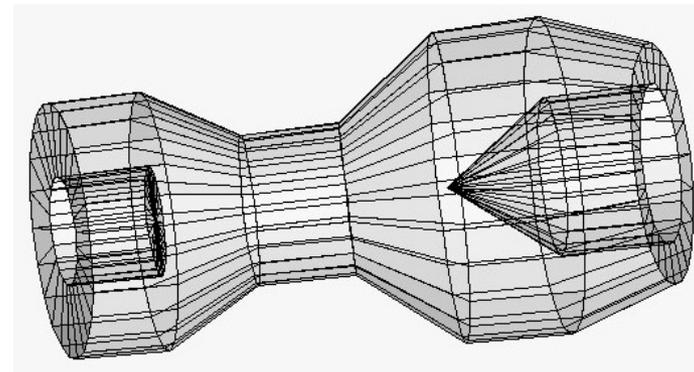
Other Specific CSG solids
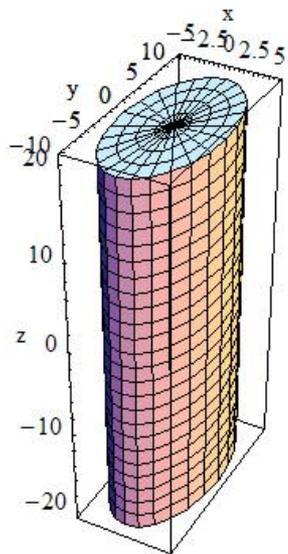
G4EllipticalTube
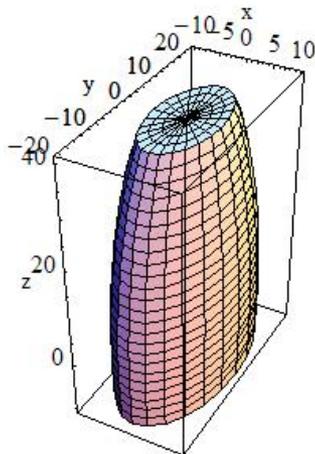
G4Ellipsoid
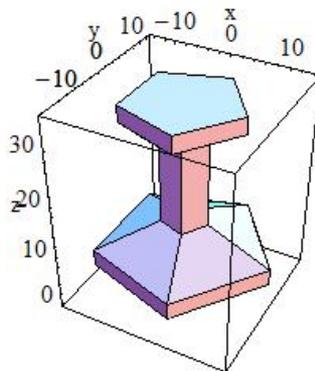
G4Polyhedra
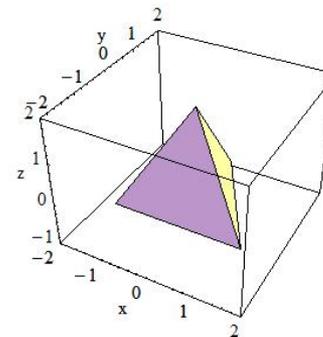
G4Tet
(tetrahedra)
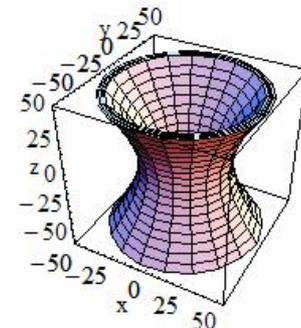
G4Hype
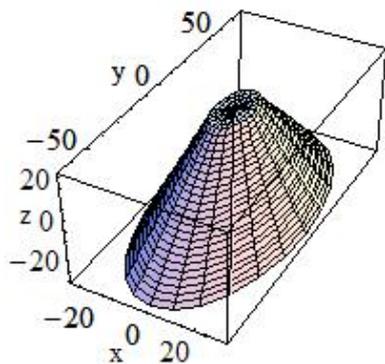
G4EllipticalCone
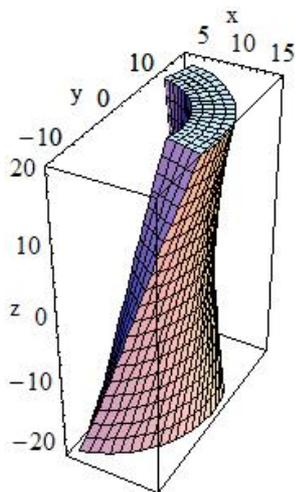
G4TwistedTubs
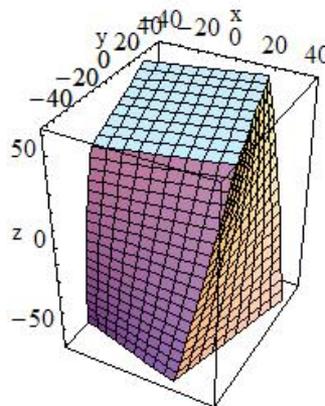
G4TwistedBox
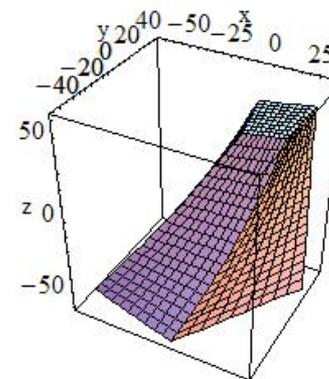
G4TwistedTrap
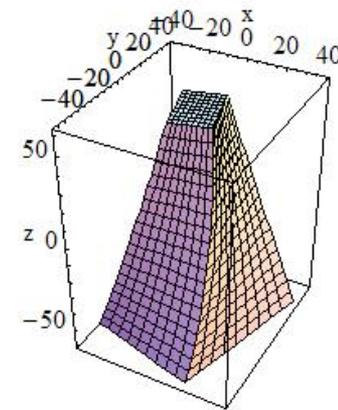
G4TwistedTrd

Consult to Section 4.1.2 of Geant4 Application Developers Guide for all available shapes.

Tessellated solids

- **G4TessellatedSolid** (since 8.1)
  - Generic solid defined by a number of facets (`G4VFacet`)
    - Facets can be triangular (`G4TriangularFacet`) or quadrangular (`G4QuadrangularFacet`)
  - Constructs especially important for conversion of complex geometrical shapes imported from CAD systems
  - But can also be explicitly defined:
    - By providing the vertices of the facets in *anti-clock wise* order, in *absolute* or *relative* reference frame
  - GDML binding
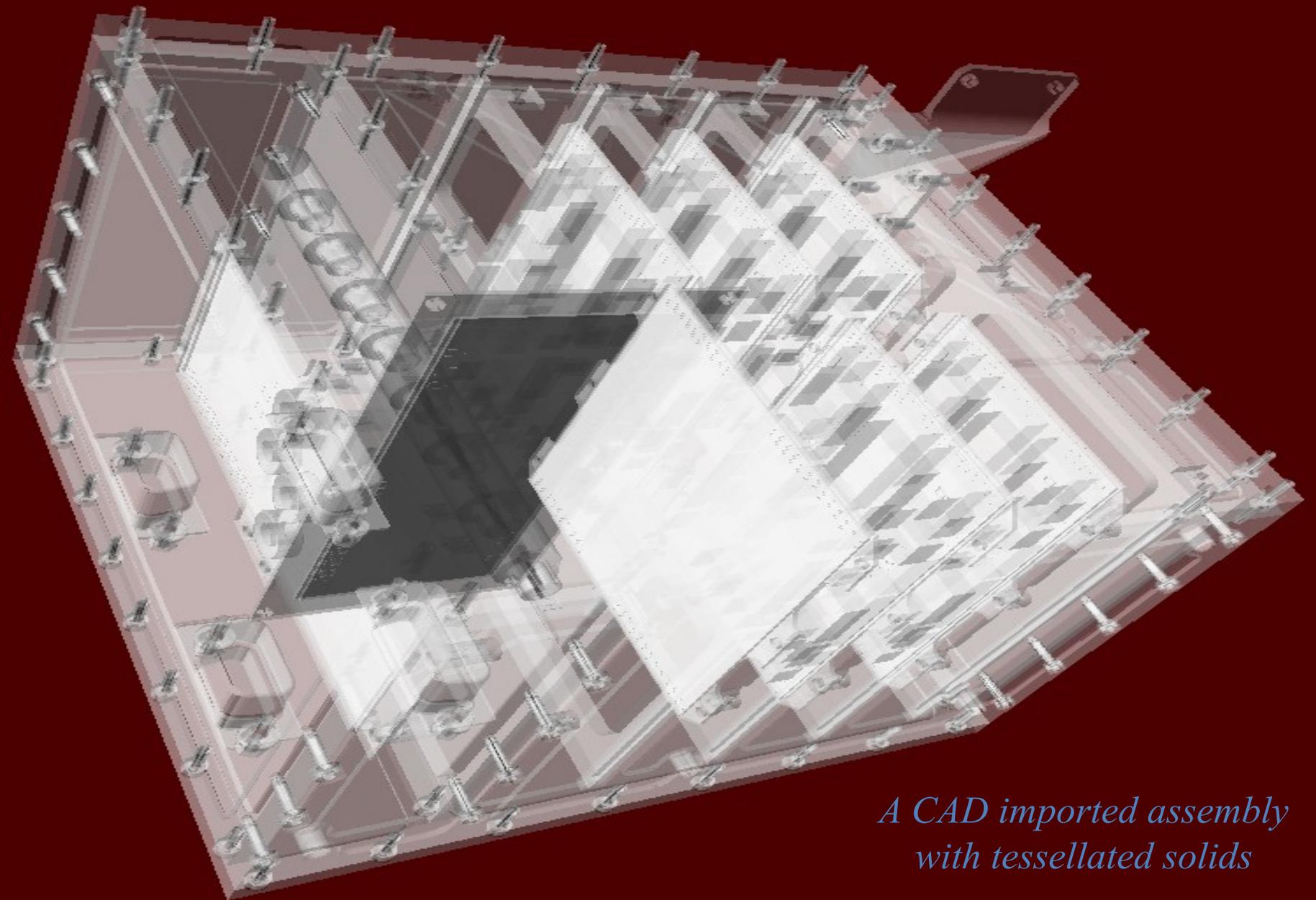
*A CAD imported assembly
with tessellated solids*

New solid library

- An important effort was begun in the last few years to write a new solid library, reviewing at the algorithmic level most of the primitives and provides an enhanced, optimized and well-tested implementation to be shared among software packages.

- In most cases considerable performance improvement was achieved.

  - For example, the time required to compute intersections with the tessellated solid was dramatically reduced with the adoption of spatial partitioning for composing facets into a 3D grid of voxels.

- Such techniques allow speedup factors of a few thousand for relatively complex structures having of order 100k to millions of facets, which is typical for geometry descriptions imported from CAD drawings.

  - Consequently, it is now possible to use tessellated geometries for tuning the precision in simulation by increasing the mesh resolution, something that was not possible before.

New in v10.4
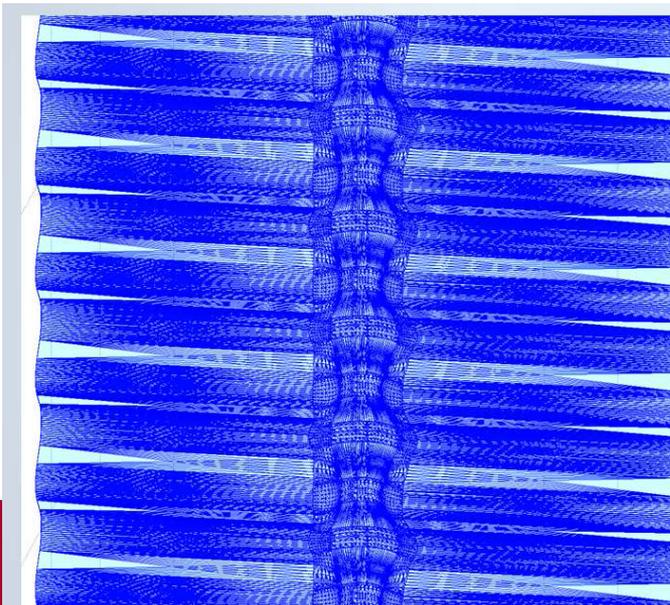
Method	Speedup
Inside	2423x
DistanceToIn	1334x
DistanceToOut	1976x

Information	Value
Number of facets	164.149
Number of voxels	100.000
Memory saved compared with original Geant4	22% (51MB)

Boolean Solids

▸ Solids can be combined using boolean operations:

  ▸ **G4UnionSolid, G4SubtractionSolid, G4IntersectionSolid**

  ▸ Requires: 2 solids, 1 boolean operation, and an (optional) transformation for the 2nd solid

  ▸ 2nd solid is positioned relative to the coordinate system of the 1st solid

  ▸ Result of boolean operation becomes a solid. Thus the third solid can be combined to the resulting solid of first operation.

▸ Solids to be combined can be either CSG or other Boolean solids.
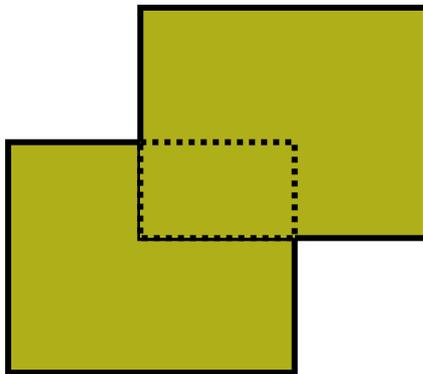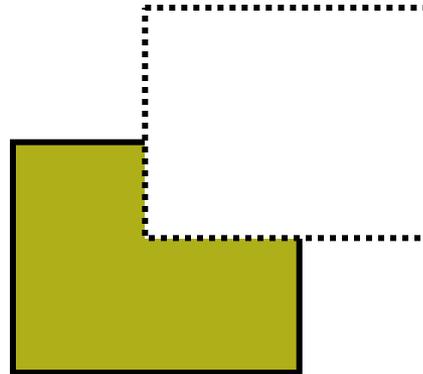
G4UnionSolid   G4SubtractionSolid   G4IntersectionSolid
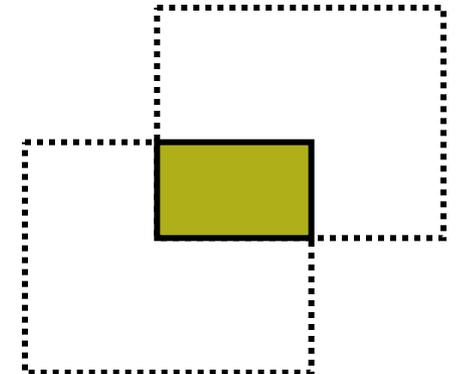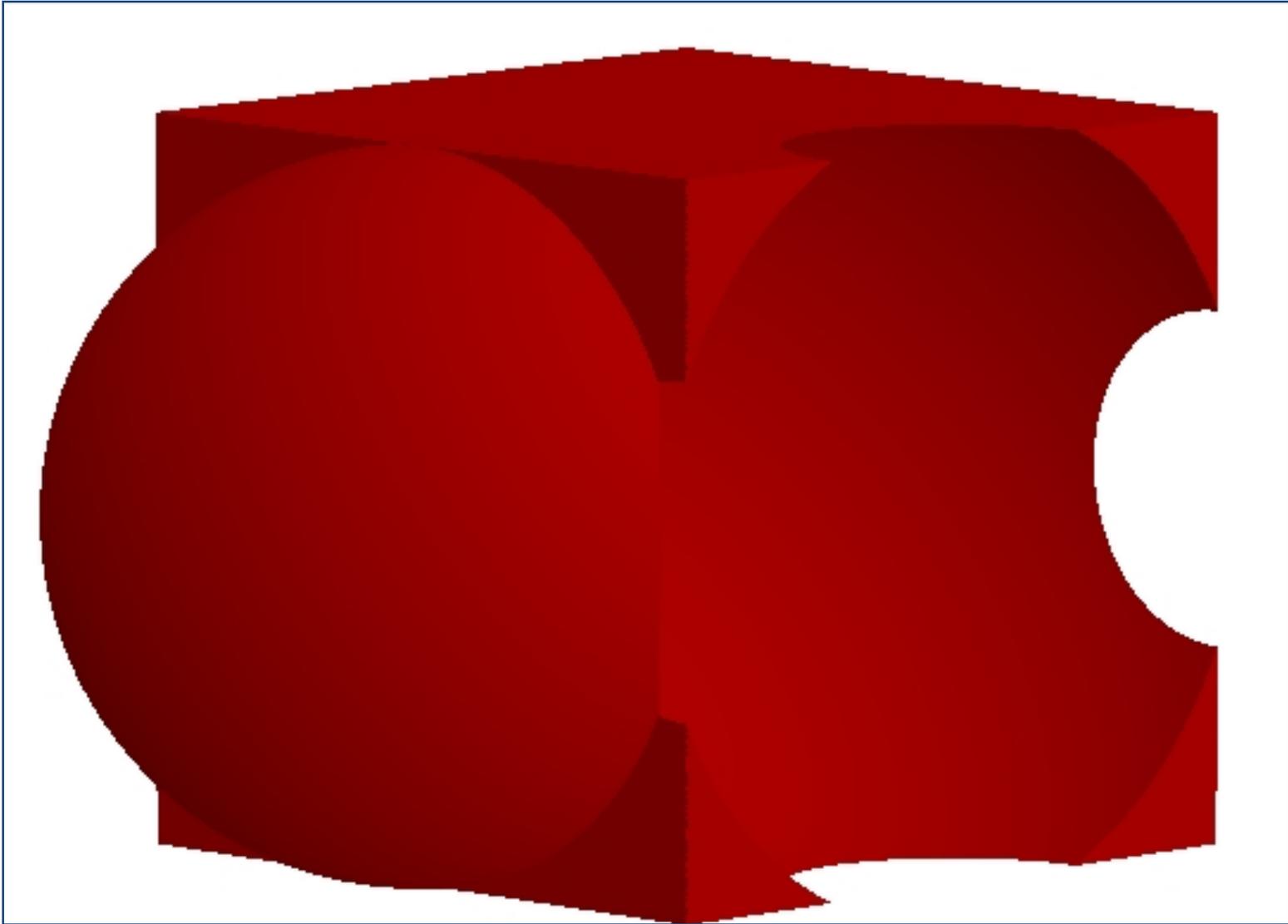
Boolean solid

```
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.));
```

‣ The origin and the coordinates of the combined solid are the same as those of the first solid.

New "multi-union" solid

- In addition to a full set of highly optimized primitives and a tessellated solid, the library includes a new "multi-union" structure implementing a composite set of many solids to be placed in 3D space.

- This differs from the simple technique based on Boolean unions, with the aim of providing excellent scalability on the number of constituent solids.

- The multi-union adopts a similar voxelization technique to partition 3D space, allowing dramatically improved speed and scalability over the original implementation based on Boolean unions.

New in v10.4

Scaling of Multi-Union inside method with boxes
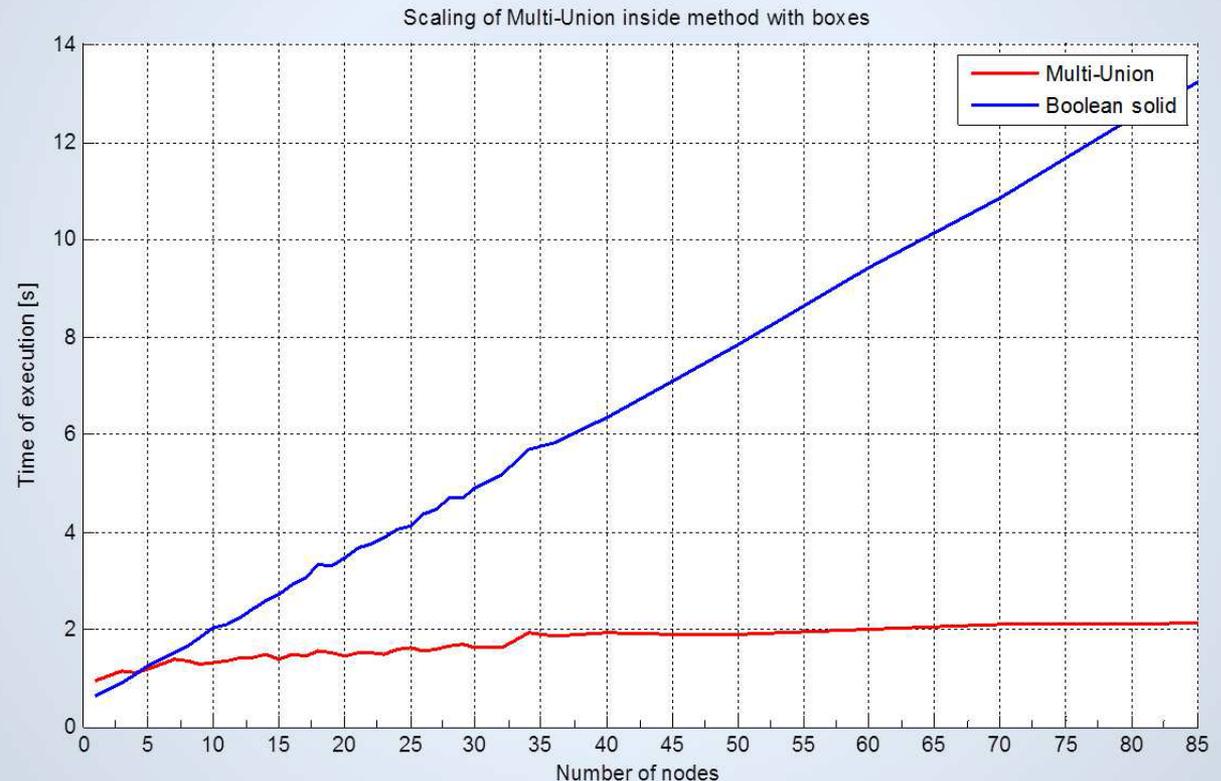
G4MultiUnion

G4MultiUnion* munion_solid = new G4MultiUnion("UnitedBoxes");

for(int i=0 ; I < nNode ; i++)
{
  G4Box* aBox = new G4Box(…);
  G4ThreeVector pos = G4ThreeVector(…);
  G4RotationMatrix rot = G4ThreeVector(…);
  G4Transform3D tr = G4Transform3D(rot, pos);
  munion_solid -> AddNode(*aBox, tr);
}

munion_solid -> Voxelize();
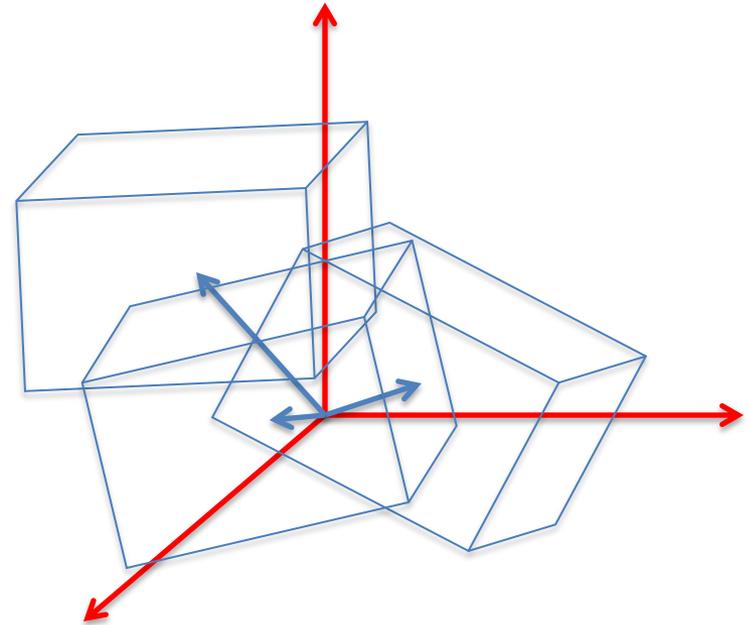
Note : G4MultiUnion is a solid. Use it to create a logical volume.

G4LogicalVolume

G4LogicalVolume

```
G4LogicalVolume(G4VSolid* pSolid,

                G4Material* pMaterial,

                const G4String &name,

                G4FieldManager* pFieldMgr=0,

                G4VSensitiveDetector* pSDetector=0,

                G4UserLimits* pULimits=0);
```

- Contains all information of volume except position and rotation
  - Shape and dimension (G4VSolid)
  - Material, sensitivity, visualization attributes
  - Position of daughter volumes
  - Magnetic field, User limits, Region
- Physical volumes of same type can share the common logical volume object.
- The pointers to solid must NOT be null.
- The pointers to material must NOT be null for tracking geometry.
- It is not meant to act as a base class.

Computing volumes and weights

- Geometrical volume of a generic solid or boolean composition can be computed from the solid:

    G4double GetCubicVolume();

    – Exact volume is determinatively calculated for most of CSG solids, while estimation based on Monte Carlo integration is given for other solids.

- Overall weight of a geometry setup (sub-geometry) can be computed from the logical volume:

    G4double GetMass(G4bool forced=false,

    G4bool propagate=true, G4Material* pMaterial=0);

    – The computation may require a considerable amount of time, depending on the complexity of the geometry.

    – The return value is cached and reused until *forced*=true.

    – Daughter volumes will be neglected if *propagate*=false.
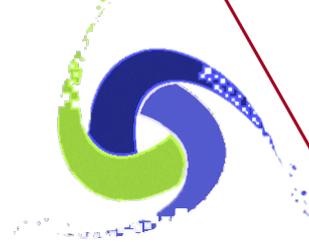
Region

Region

- A region may have its unique
  - Production thresholds (cuts)
    - If a region in the mass geometry does not have its own production thresholds, those of the default region are used (i.e., may not be those of the parent region).
  - User limits
    - Artificial limits affecting to the tracking, e.g. max step length, max number of steps, min kinetic energy left, etc.
    - You can set user limits directly to logical volume as well. If both logical volume and associated region have user limits, those of logical volume wins.
  - User region information
    - E.g. to implement a fast Boolean method to identify the nature of the region.
  - Fast simulation manager
  - Regional user stepping action
  - Field manager
- Please note :
  - World logical volume is recognized as the default region. User is not allowed to define a region to the world logical volume.

Root logical volume

- A logical volume can be a region. More than one logical volumes may belong to a region.

- A region is a part of the geometrical hierarchy, i.e. a set of geometry volumes, typically of a sub-system.

- A logical volume becomes a root logical volume once a region is assigned to it.

  – All daughter volumes belonging to the root logical volume share the same region, unless a daughter volume itself becomes to another root.

- Important restriction :

  – No logical volume can be shared by more than one regions, regardless of root volume or not.

World Volume - Default Region

Root logical - Region A

Root logical - Region B
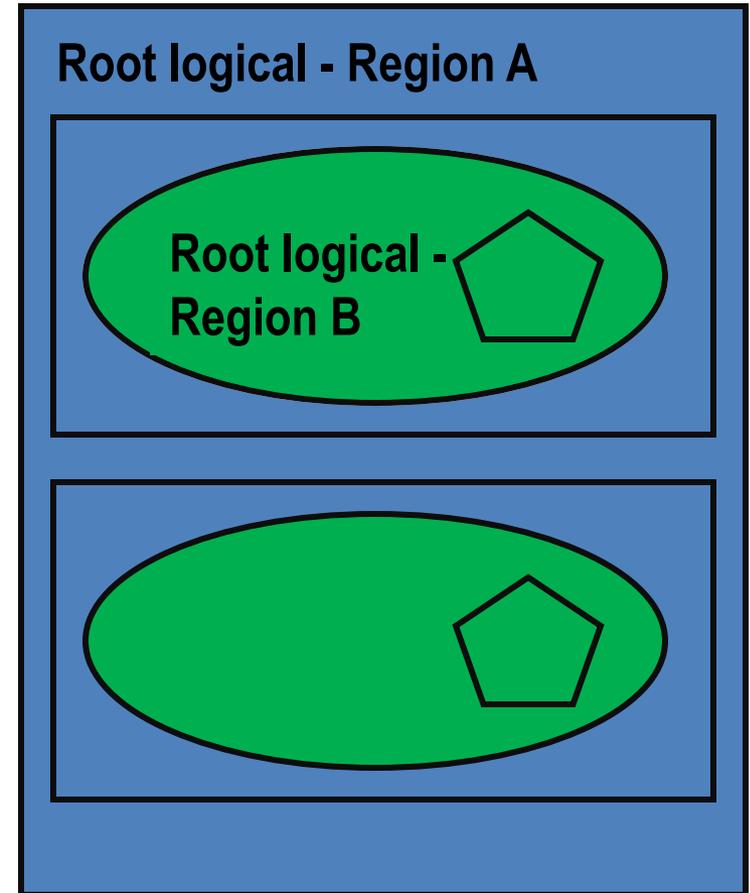
G4Region

- A region is instantiated and defined by

  ```
  G4Region* aRegion = new G4Region("region_name");

  aRegion->AddRootLogicalVolume(aLogicalVolume);
  ```

  - Region propagates down to all geometrical hierarchy until the bottom or another root logical volume.

- Production thresholds (cuts) can be assigned to a region by

  ```
  G4Region* aRegion
    = G4RegionStore::GetInstance()->GetRegion("region_name");

  G4ProductionCuts* cuts = new G4ProductionCuts;

  cuts->SetProductionCut(cutValue);

  aRegion->SetProductionCuts(cuts);
  ```

G4Region class

- 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.

Physical volume

Version 10.5

Detector geometry

- Three conceptual layers

    – G4VSolid -- *shape, size*

    – G4LogicalVolume -- *daughter physical volumes,*

             material, sensitivity, user limits, etc.

    – G4VPhysicalVolume -- *position, rotation*

Define detector geometry
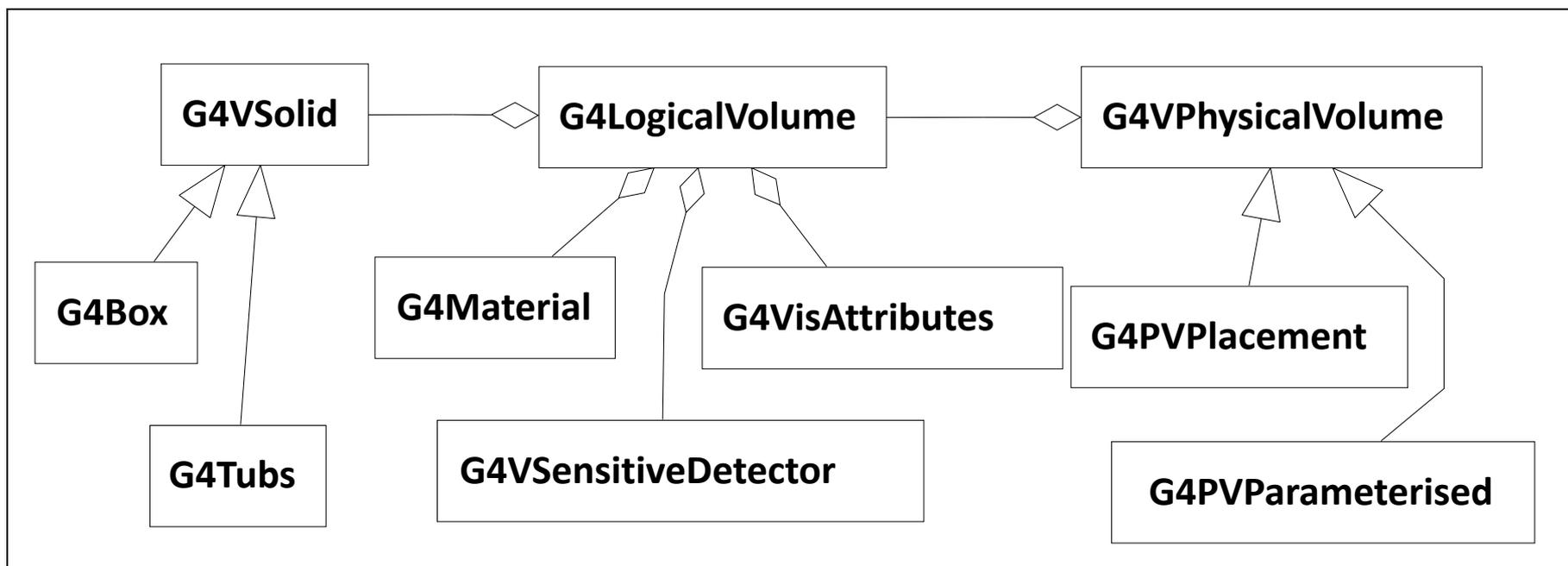
- Basic strategy

```
G4VSolid* pBoxSolid =

    new G4Box("aBoxSolid", 1.*m, 2.*m, 3.*m);

G4LogicalVolume* pBoxLog =

    new G4LogicalVolume( pBoxSolid, pBoxMaterial,

                              "aBoxLog", 0, 0, 0);

G4VPhysicalVolume* aBoxPhys =

    new G4PVPlacement( pRotation,

        G4ThreeVector(posX, posY, posZ), pBoxLog,

        "aBoxPhys", pMotherLog, 0, copyNo);
```

Physical Volumes

- Placement volume : it is one positioned volume

  – One physical volume object represents one "real" volume.

- Repeated volume : a volume placed many times
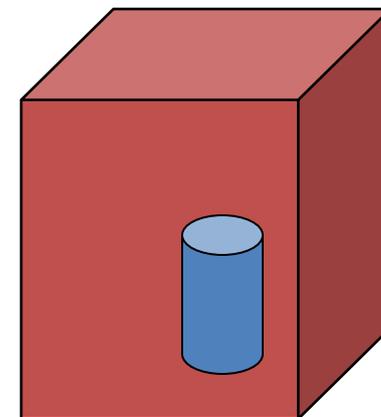
  – One physical volume object <u>represents</u> any number of "real" volumes.

  – reduces use of memory.

  – Parameterised

    - repetition w.r.t. copy number

  – Replica and Division

    - simple repetition along one axis

- A mother volume can contain either

  – many placement volumes

  – or, one repeated volume

placement

repeated

Physical volume
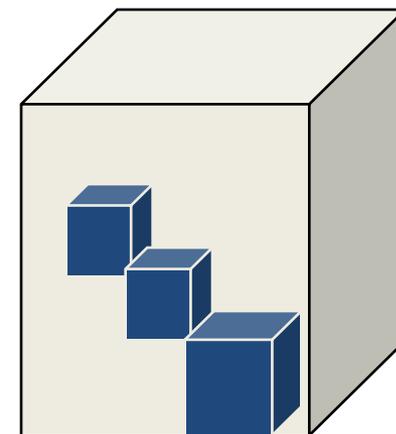
- G4PVPlacement   1 Placement = One Placement Volume
  - A volume instance positioned once in its mother volume

- G4PVParameterised   1 Parameterized = Many Repeated Volumes
  - Parameterized by the copy number
    - Shape, size, material, sensitivity, vis attributes, position and rotation can be parameterized by the copy number.
    - You have to implement a concrete class of G4VPVParameterisation.
  - Reduction of memory consumption
  - Currently: parameterization can be used only for volumes that either

    a) have no further daughters, <u>or</u>

    b) are identical in size & shape (so that grand-daughters are safely fit inside).
  - By implementing G4PVNestedParameterisation instead of G4VPVParameterisation, material, sensitivity and vis attributes can be parameterized by the copy numbers of ancestors.

Physical volume

- **G4PVReplica**   1 Replica = Many Repeated Volumes
  - Daughters of same shape are aligned along one axis
  - Daughters fill the mother completely without gap in between.
- **G4PVDivision**   1 Division = Many Repeated Volumes
  - Daughters of same shape are aligned along one axis and fill the mother.
  - There can be gaps between mother wall and outmost daughters.
  - No gap in between daughters.
- **G4ReflectionFactory**   1 Placement = a pair of Placement volumes
  - generating placements of a volume and its reflected volume
  - Useful typically for end-cap calorimeter
- **G4AssemblyVolume**   1 Placement = a set of Placement volumes
  - Position a group of volumes

G4PVPlacement

G4PVPlacement

```
G4PVPlacement(

   G4Transform3D(G4RotationMatrix &pRot, // rotation of daughter volume

             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
```
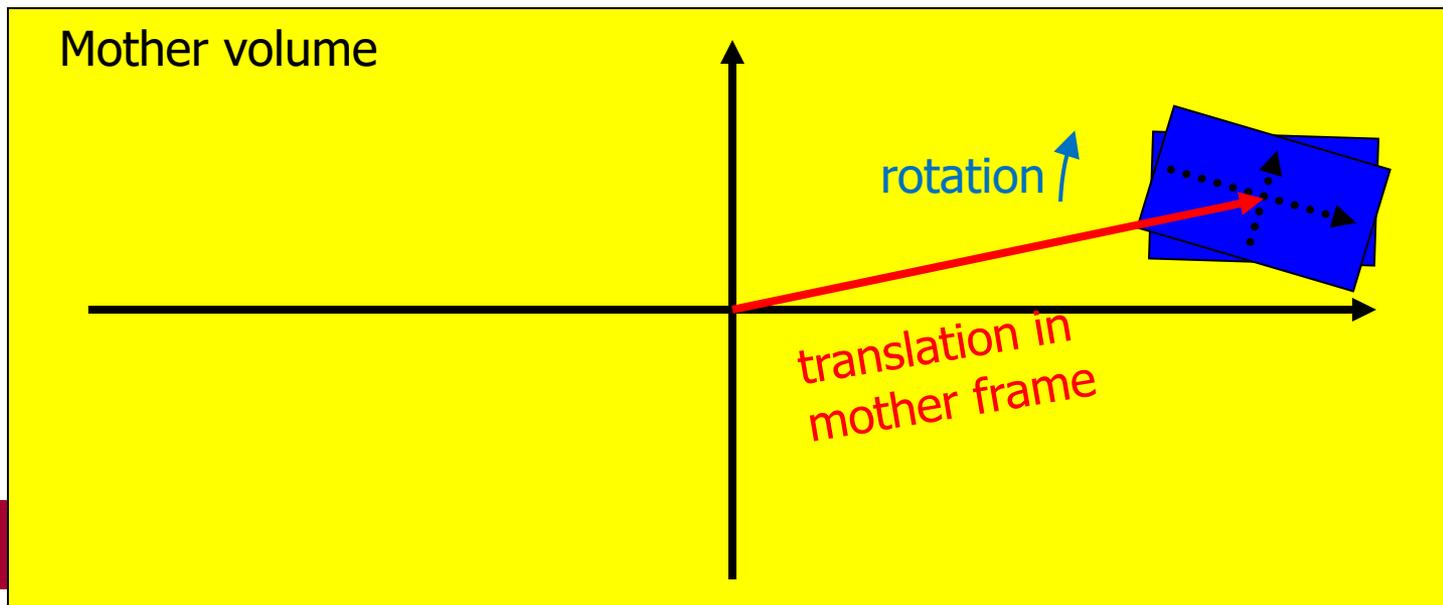
- Single volume positioned relatively to the mother volume.

Alternative G4PVPlacement

```
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…
```

Note:

- This G4PVPlacement is identical to the previous one if there is no rotation.

  - Previous one is much easier to understand.

- The advantage of this second constructor is setting the pointer of the rotation

  matrix rather than providing the values of the matrix.

  - You may change the matrix without accessing to the physical volume.

  - This is for power-users, though.