L05: GEOMETRY DESCRIPTION

Tomohiro Yamashita
Kobe Proton Center

Contents

Introduction

INTRODUCTION

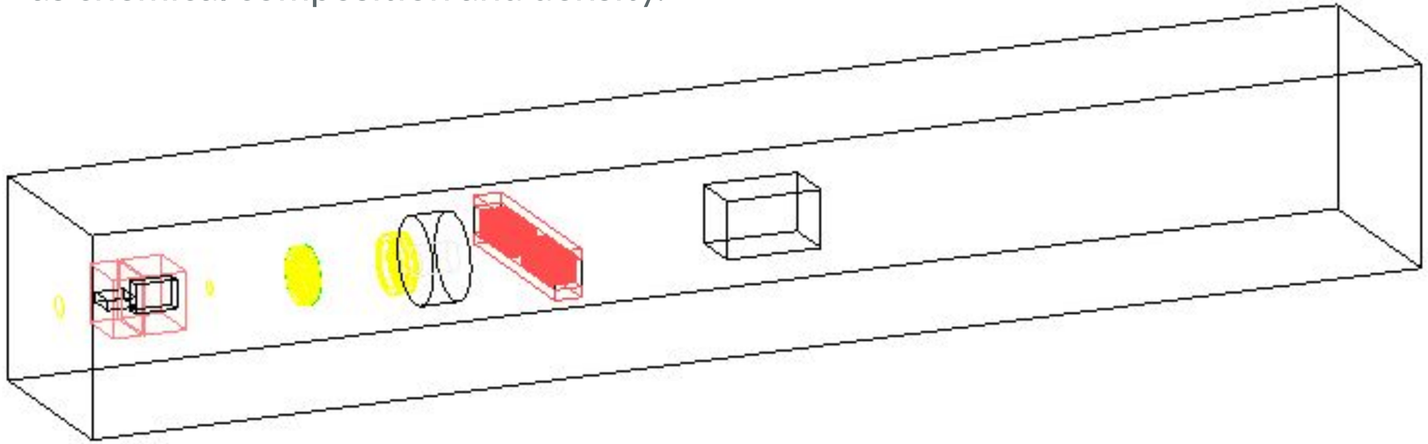
- To run Geant4 simulation, users must give 3 mandatory things to Geant4,
  - Geometry
    - The Focus of this Lecture.
    - A description of the system, in which your simulation will take place.
    - Multi-Leaf Collimator, Bolus, Water Phantom, Patient and etc.
  - Physics List
    - A list of all the particles and all the physics processes to be simulated.
  - Initial condition
    - A way to start the simulation.
    - Energy, position and direction of protons.
    - Energy spectrum of X-rays.
- To make the simulation meaningful, you have to tell Geant4
  - A way to retrieve informations from the simulation
    - E.g. Dose distribution in a water phantom.

Concepts in Geometry

- Geometry
  - The complete description of the system, including shapes, positions, and material information, is referred to as the geometry.
  - Geometry defines the spatial layout of objects in Geant4 simulations.
- Volumes as Building Blocks
  - In the Geant4, elements such as MLC, Water Phantom, and patients are represented as volumes.
  - Each volume contains information about its geometric shape, constituent materials, and position.
- Hierarchy of Volumes
  - Volumes are placed within another, with the inner volumes referred to as daughter volumes and the outer volumes as mother volumes.
- Three Layers: Solid, Logical Volume, Physical Volume
  - Solid: Represents the shape and size of an object, defining its geometry.
  - Logical Volume: Combines a solid with material properties, serving as an abstract entity.
  - Physical Volume: An instance of a logical volume placed within a mother volume, defining its position and orientation.

Concepts in Geometry

- The World volume
  - The largest volume in the system is called as the World volume, encompassing all other volumes.
  - In the context of radiation therapy simulations, the treatment room corresponds to the World volume.
  - The world volume defines the global coordinate system. The origin of the global coordinate system is at the center of the world volume.
- Material
  - Volumes also store information about the materials they are composed of, including such as chemical composition and density.



Steps of geometry definition

Steps of geometry definition

1. Prepare all materials you want to use in your simulation.
2. Define shapes (Solid).
3. Define volumes by adding properties such as material to the Solid.
    - This volume is called "Logical Volume" (G4LogicalVolume).
4. Position the Logical Volumes created in the previous step.
    - This positioned volume is called “Physical Volume”.
5. Associate a field to geometry if nescesary.
6. Define how you will retrieve information.
    - E.g. Dose distribution in a water phantom.
7. Customize the visual representation of your geometry if nescesary.
8. Assign specific production cuts and etc. to parts of your geometry if nescesary.

Geometry definition class

1. Inherit from the G4VUserDetectorConstruction class to create your own geometry definition class.
2. Describe your geometry within the "construct" function.
3. Create an object of the geometry definition class and register it with G4RunManager.

Header file

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// 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
```

Your own geometry definition class

The base class provided by Geant4 for geometry definition

Write your geometry description in this function

Materials

Materials in Geant4

- Three classes to describe materials; G4Isotope, G4Element and G4Material
- G4Isotope: Represents a single isotope.
    - Describes the properties of atoms: atomic number, number of nucleons, mass per mole, etc.
- G4Element: Represents a single element.
    - Describes the properties of elements: effective atomic number, effective number of nucleons, effective mass per mole, etc.
    - Also includes parameters like radiation length and ionization parameters.
- G4Material: Represents the chemical and physical properties of a material.
    - Describes the macroscopic properties of matter: density, state, temperature, pressure.
    - Also includes radiation length, mean free path, dE/dx, etc.
- When defining geometry, only G4Material is used.
    - G4Element/G4Isotope is used only to create G4Material.

NIST Material Database

- While users can define their own elements and materials, it is generally recommended to use the standard NIST (National Institute of Standards and Technology, U.S.) Material Database provided by Geant4

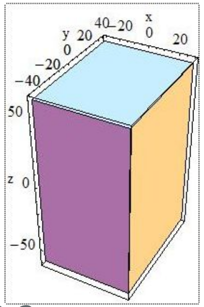
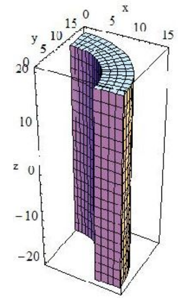
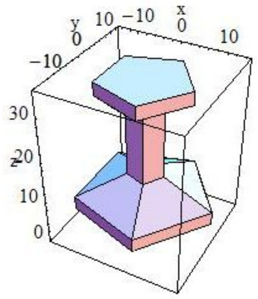
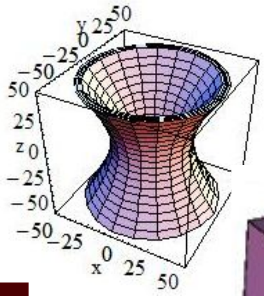
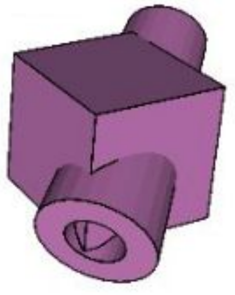
```
G4NistManager* manager = G4NistManager::GetPointer();

G4Element*  elm = manager->FindOrBuildElement(”atomic-symbol", G4bool iso);
G4Element*  elm = manager->FindOrBuildElement(G4int iZ, G4bool iso);
G4Material* mat = manager->FindOrBuildMaterial("name", G4bool iso);
G4Material* mat = manager->FindOrBuildMaterial("G4_WATER");
G4double isotopeMass = manager->GetMass(G4int iZ, G4int N);
```

  - G4bool iso = true ; utilizes natural isotope abundances from the environment.
  - G4bool iso = false ; does not consider isotopes.

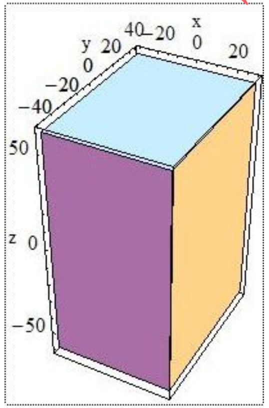
Solid

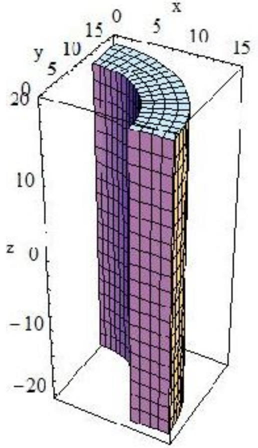
Solid definition

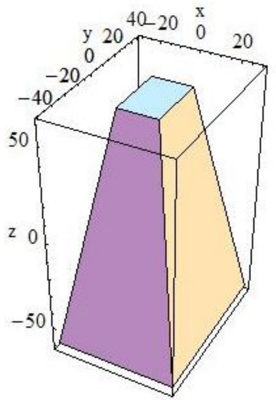
- Geant4 provide many classes of shape for use inherited from G4VSolid
- 4 types solids defined in Geant4
  - CSG (Constructed Solid Geometry) solids
    - G4Box, G4Tubs, G4Cons (cone, or conical section), G4Trd (trapezoid), …
  - Specific solids (similar to CSG)
    - G4Polycone, G4Polyhedra, G4Hype (tube with a hyperbolic profile), …
  - Boolean solids
    - Creating a solid by boolean operation of solids
  - Tessellated solids
    - For complex geometrical shapes imported from CAD systems

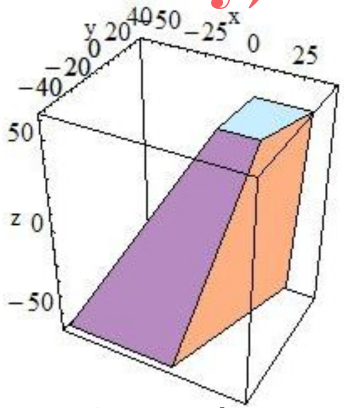






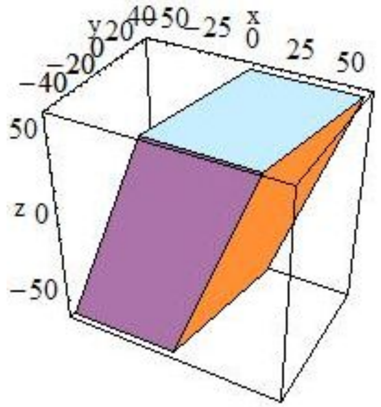
CSG (Constructed Solid Geometry) solids

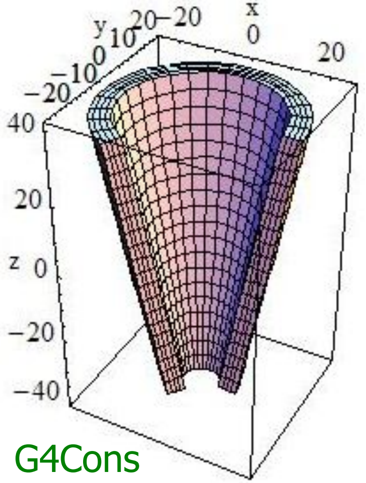

G4Box

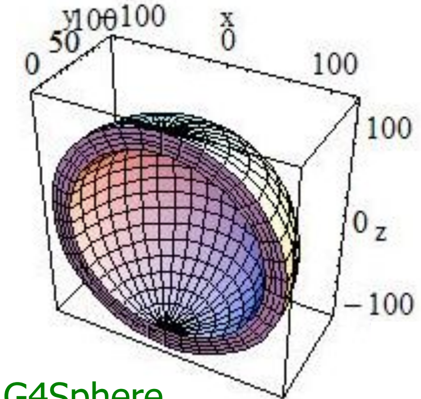

G4Tubs

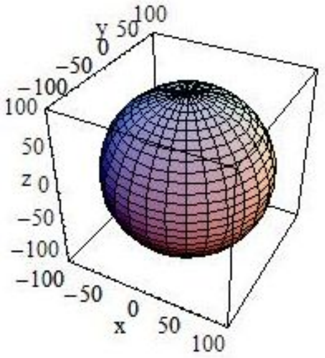

G4Trd(trapezoid)

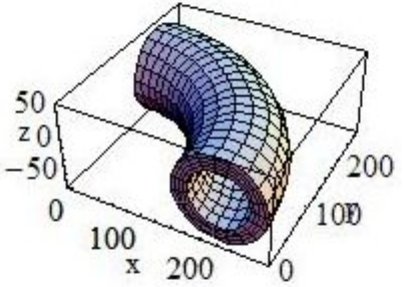

G4Trap (generic trapezoid)


G4Para (parallelepiped)

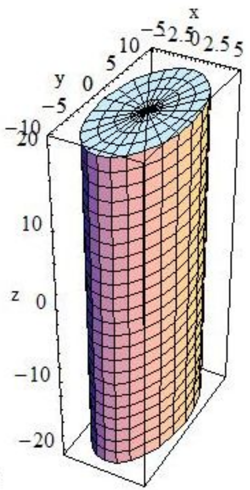

G4Cons

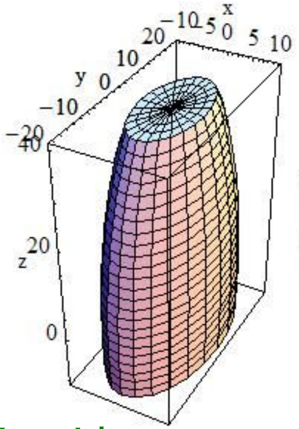

G4Sphere

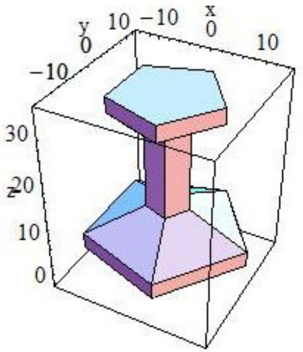

G4Orb(full solid sphere)

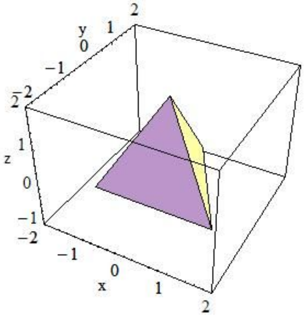

G4Torus

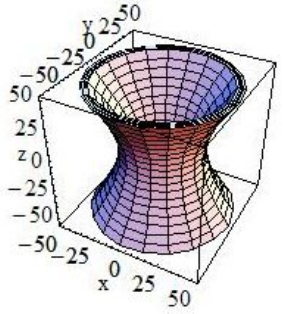
Specific solids

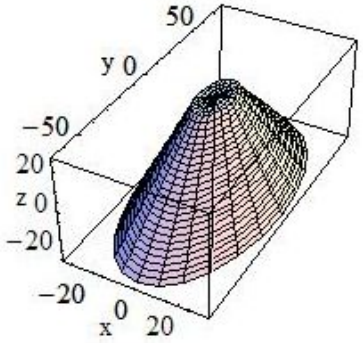

G4EllipticalTube

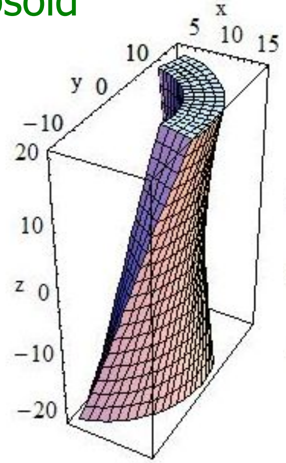

G4Ellipsoid

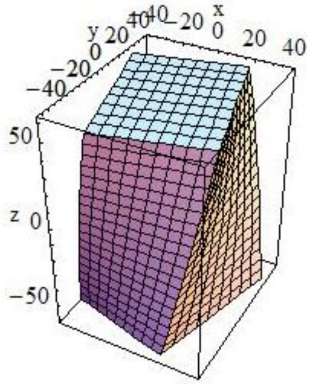

G4Polyhedra

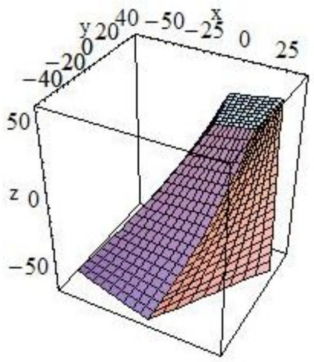

G4Tet
(tetrahedra)

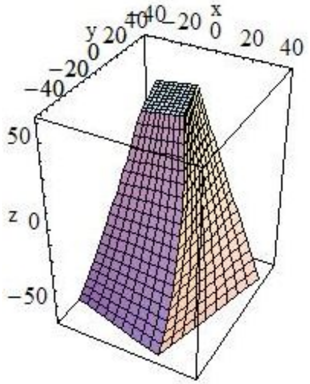

G4Hype


G4EllipticalCone


G4TwistedTubs

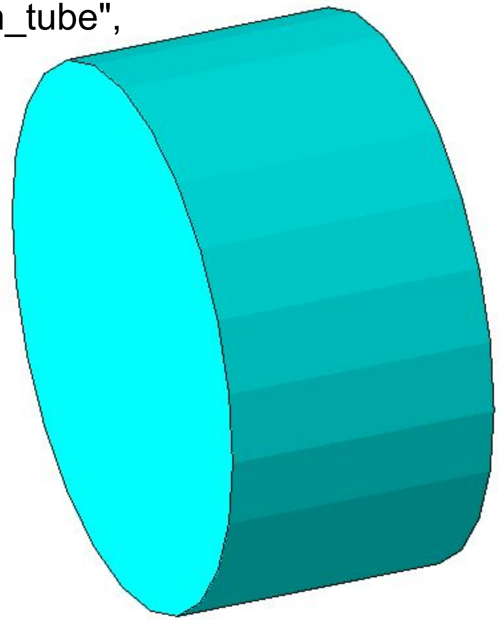

G4TwistedBox


G4TwistedTrap

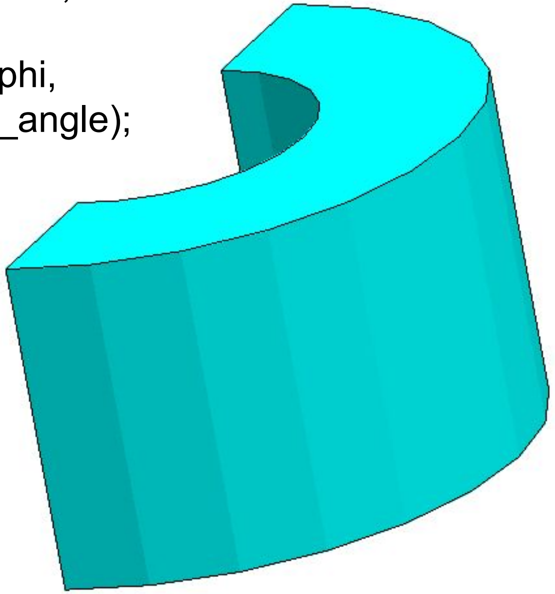

G4TwistedTrd

Examples

```
G4double inner_radius = 0.0;
G4double outer_radius = 10.0*CLHEP::cm;
G4double length = 5.0*CLHEP::cm;
G4double starting_phi = 0.0;
G4double segment_angle = 2*M_PI*CLHEP::rad;
G4Tubs* phantom_solid
   = new G4Tubs("phantom_tube",
            inner_radius,
            outer_radius,
            length,
            starting_phi,
            segment_angle);
```

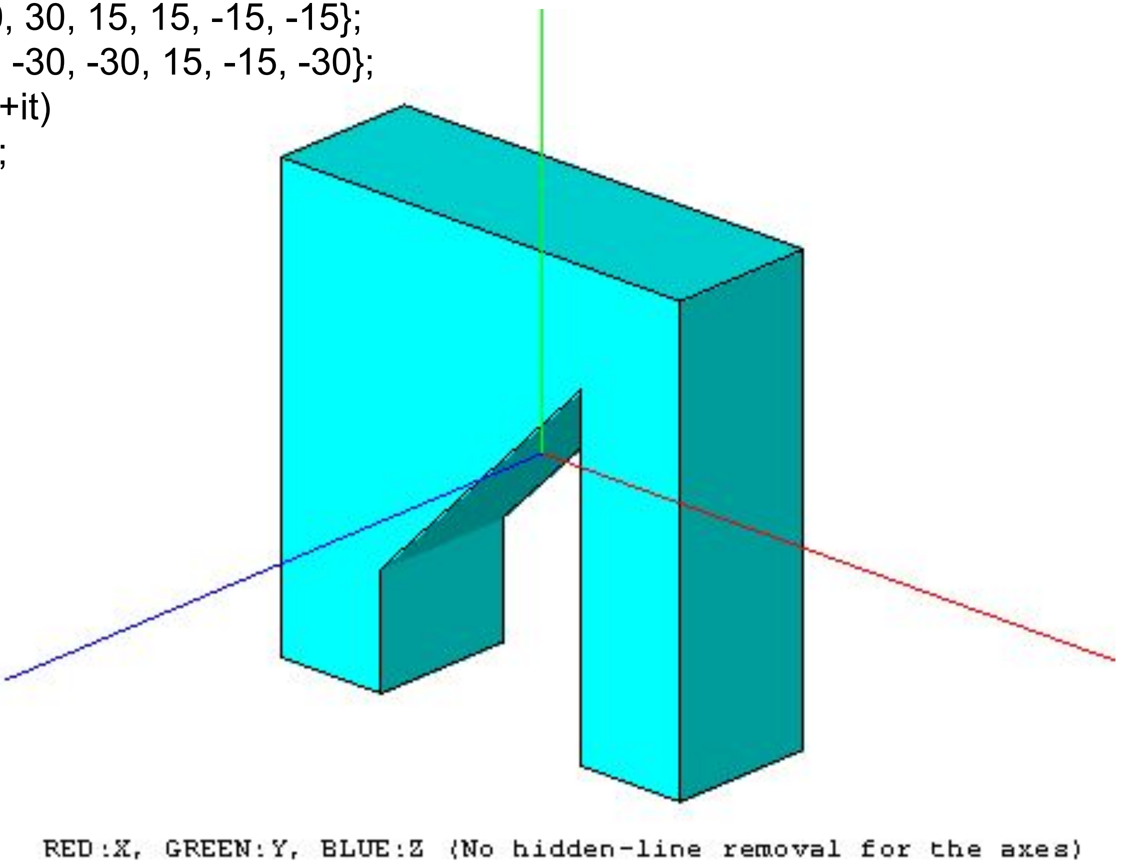


```
G4double inner_radius = 5.0*CLHEP::cm;
G4double outer_radius = 10.0*CLHEP::cm;
G4double length = 5.0*CLHEP::cm;
G4double starting_phi = 0.0;
G4double segment_angle = M_PI*CLHEP::rad;
G4Tubs* phantom_solid
   = new G4Tubs("phantom_tube",
            inner_radius,
            outer_radius,
            length,
            starting_phi,
            segment_angle);
```



Examples

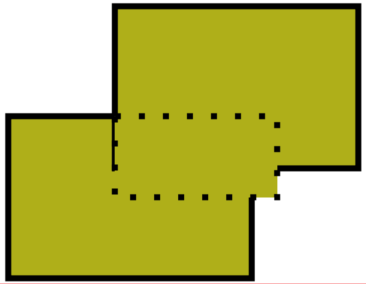
```
std::vector<G4double> xx{-30, -30, 30, 30, 15, 15, -15, -15};
std::vector<G4double> yy{-30, 30, 30, -30, -30, 15, -15, -30};
for (unsigned int it = 0; it < xx.size(); ++it)
    polygon.emplace_back(xx[it], yy[it]);
G4TwoVector off1{0.0, 0.0};
G4TwoVector off2{0.0, 0.0};
G4double scale1 = 1.0;
G4double scale2 = 1.0;
G4double halfZ = 10;
G4ExtrudedSolid* phantom_solid =
    new G4ExtrudedSolid(name,
    polygon,
    halfZ,
    off1, scale1,
    off2, scale2);
```


RED:X, GREEN:Y, BLUE:Z (No hidden-line removal for the axes)

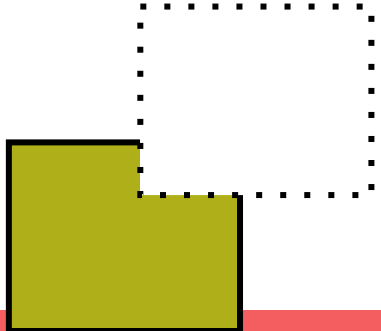
Boolean solid

- You can represent solids using boolean operations.
  - G4UnionSolid, G4SubtractionSolid, G4IntersectionSolid
  - To create these representations, you need two solids and a boolean operation that defines their interaction.
  - The second solid is positioned with respect to the spatial coordinates of the first solid.
  - The result of a boolean operation is a single solid, known as a "boolean solid," allowing you to perform additional operations if needed.
  - Solids usable for boolean operations are primarily based on Constructive Solid Geometry (CSG) or are themselves boolean solids.
  - Note that boolean solids may have longer times compared to CSG solids for tracking.

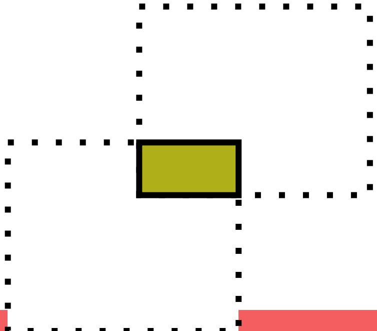
G4UnionSolid



G4SubtractionSolid



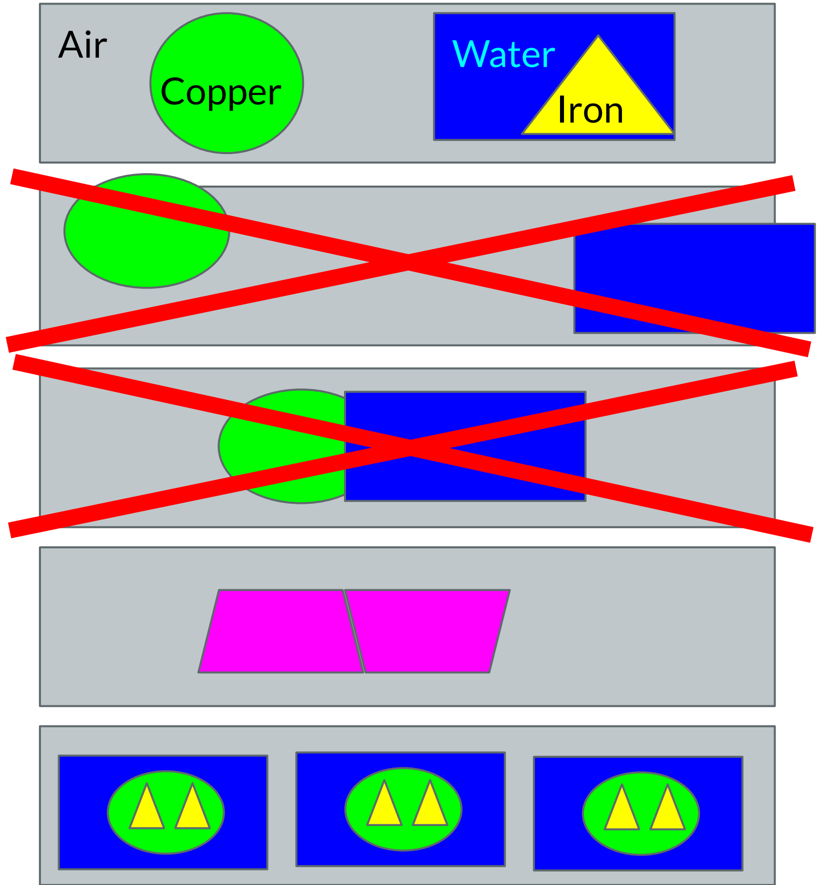
G4IntersectionSolid



Logical Volume

Logical Volume

- Logical Volumes are defined by adding properties, such as materials, to solids.
- When a daughter volume is placed within a mother volume, the volume information in the placed region is replaced by that of the daughter volume.
- Daughter volumes must always be logical volumes; mother volumes can be either logical or physical.
- One logical volume can be placed multiple times, with all daughters appearing in all instances of the mother physical volumes.
- Volumes can be placed in contact with each other, but they cannot extend beyond mother volumes, and intersecting volumes are not allowed.



Logical Volume

- Mother volumes hold pointers to daughter volumes, although daughter volumes do not have information (pointers) about the mother volume.
- The top of the hierarchy is the World Volume, containing all other volumes.
- The 'World' maintains the global coordinate system, which serves as the fundamental reference for the entire geometry.
- Particle positions are expressed in this global coordinate system, and methods are available to transform position information into the coordinate system of daughter volumes.

Defininig Logical Volume (G4LogicalVolume)

Constructor

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

- Information stored in a G4LogicalVolume object includes:
  - Shape and Size (G4VSolid): Mandatory.
  - Material Information (G4Material): Mandatory (unless not tracking particles within the volume).
  - Name (G4String): Not mandatory but recommended to assign a value.
  - Electromagnetic Field Information (G4FieldManager).
  - Sensitivity Configuration (G4VSensitiveDetector).
  - User-Defined Conditions (G4UserLimits).
  - On/Off for Geometry Optimization Methods (G4bool).

  } These arguments will be explained in later lectures. Logical Volumes can be defined without setting these arguments because default values are already set.
- The visibility of the volume in graphics can be set using the SetVisAttributes() function.
- This class cannot be used as a base class (not a virtual class).

Example of Logical Volume Definition

- Below is an example of code that defines the logical volume of a World Volume filled with air

```
// Get pointer to 'Material Manager‘
   G4NistManager* materi_Man = G4NistManager::Instance();

// Define the shape of solid
   G4double leng_X_World = 2.0*m;          // X-full-length of world
   G4double leng_Y_World = 2.0*m;          // Y-full-length of world
   G4double leng_Z_World = 2.0*m;          // Z-full-length of world
   auto solid_World =
     new G4Box{ "Solid_World", leng_X_World/2.0, leng_Y_World/2.0, leng_Z_World/2.0 };

// Define logical volume
   G4Material* materi_World = materi_Man->FindOrBuildMaterial( "G4_AIR" );
   auto logVol_World = new G4LogicalVolume{ solid_World, materi_World, "LogVol_World" };
```

Creating the NIST Material Manager

Defining the solid shape of the World.

Retrieving air material from NIST.

Creating the logical volume of the World.

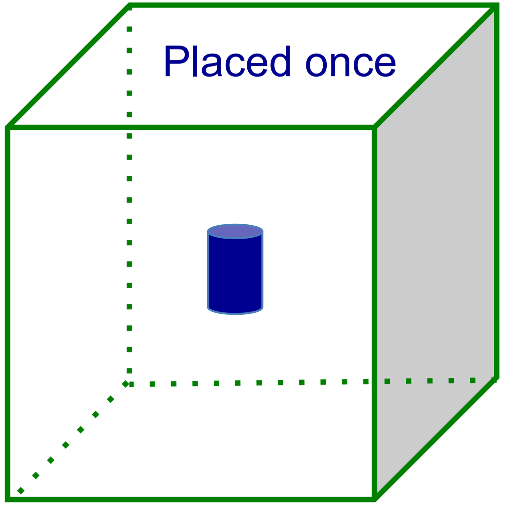
Physical Volume

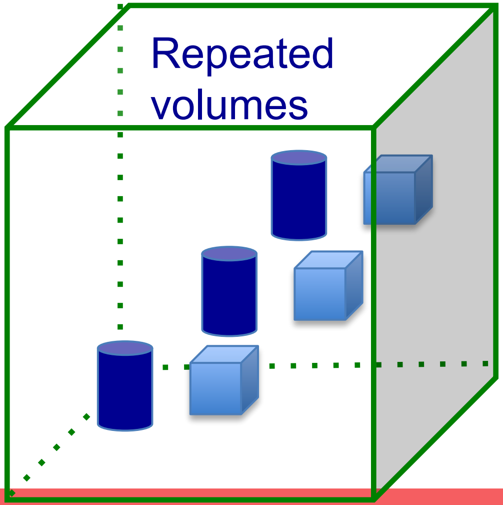
Physical Volume

- A Physical Volume refers to a daughter Logical Volume placed within a mother volume.
  - The volume information (e.g., material) in the region where the daughter volume is placed belongs to the daughter volume, not the mother volume.
  - The mother volume holds information about the position and orientation of the daughter volume.
  - The mother volume can be either a logical volume or a physical volume.
- There are constraints on placement:
  - Daughter volume must not extend beyond the boundaries of the mother volume.
  - Daughter volume must not overlap with each other.

Creating Physical Volumes

- There are two main methods.
    - Places an object once within the mother volume..
        - Geant4 provides G4PVPlacement
        - One physical volume object corresponds to a single object placed within the mother volume.
    - Repeated volumes: Places daughter volumes multiple times within the mother volume.
        - One physical volume object corresponds to any number of volumes placed within the mother volume.
        - Techniques include function representation for flexible object placement:
            - Replicas
            - Parameterization
        - This representation helps conserve memory.





G4PVPlacement

G4PVPlacement constructor

```
G4PVPlacement(
  const G4Transform3D &Transform3D,  // Object representing the rotation and translation of the child.
  G4LogicalVolume *pLogical,         // Pointer to the child object's logical volume.
  const G4String &pName,             // Name of the child object after placement (placement name).
  G4LogicalVolume *pMotherLogical,   // Pointer to the mother logical volume (placement destination).
  G4bool pMany,                      // This argument is currently not in use.
  G4int pCopyNo,                     // Copy number.
  G4bool pSurfChk = false);          // Collision Check.
```

- Copy number:
  - By setting an integer value, you can add a copy number (an ID for placed logical volumes) to G4PVPlacement. If not needed, you can set all copy numbers to zero.
  - Copy numbers can be duplicated, but if you want to use them as placement IDs, they should be different.
- Collision Check for Overlapping Objects:
  - Overlapping objects are not allowed during placement.
  - Setting the flag to 'true' automatically checks for overlap (setting to 'true' is recommended).

G4PVPlacement

- To place a daughter volume in the mother physical volume

```
G4PVPlacement(
    const G4Transform3D &Transform3D,  // Object representing the rotation and translation of the child.
    const G4String &pName,             // Name of the child object after placement (placement name).
    G4LogicalVolume *pLogical,         // Pointer to the logical volume of the child object.
    G4VPhysicalVolume *pMotherPhys,    // Pointer to the mother physical volume (placement destination).
    G4bool pMany,                      // This argument is currently not in use.
    G4int pCopyNo,                     // Copy number.
    G4bool pSurfChk = false);          // Collision Check.
```

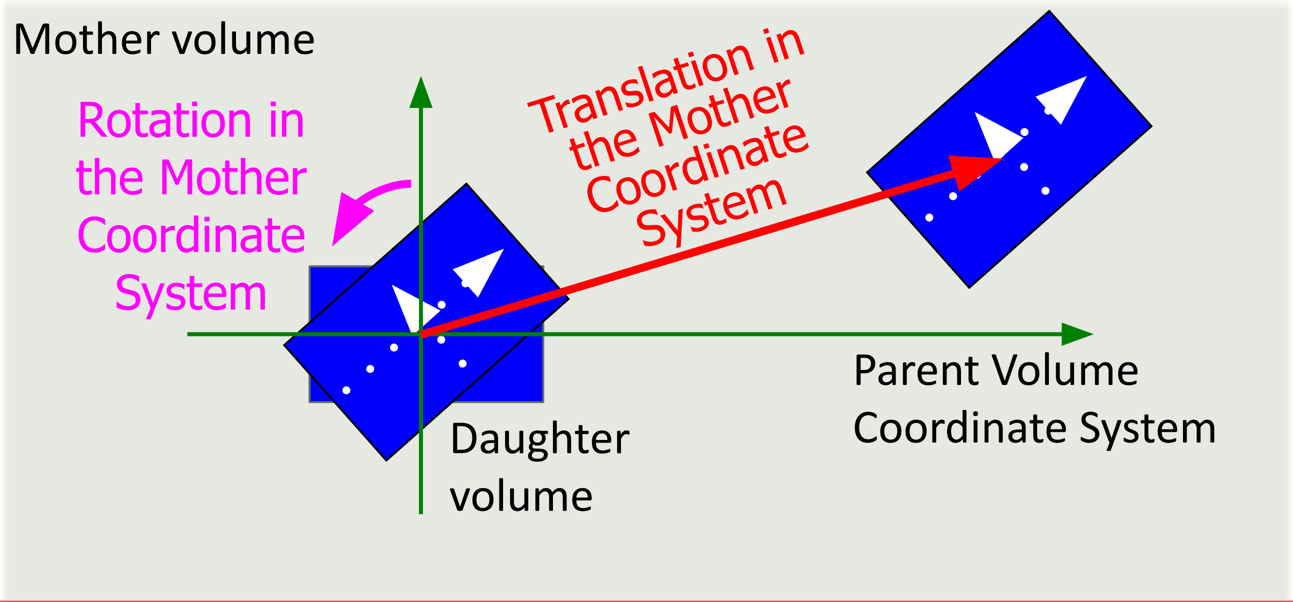
- Another constructor

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

This constructor can be confusing when dealing with rotated placements.
Therefore, we recommend using the constructor mentioned above.

Volume Placement with G4PVPlacement

1. Rotate the daughter volume within the mother coordinate system.
2. Translate the daughter volume within the mother volume's coordinate system.
3. [Note] Rotation and translation are independent, so the order of operations doesn't need to be considered.


Mother volume
Rotation in the Mother Coordinate System
Translation in the Mother Coordinate System
Parent Volume Coordinate System
Daughter volume

Translation/Rotation

- G4Transform3D: Represents 3D rotations and translations in the object's coordinate system.
  - Rotation is expressed using the G4RotationMatrix class.
  - Translation is expressed using the G4ThreeVector class.
  - A 'typedef' of CLHEP's Transform3D class, which is defined in the following location in Geant4 distribution code:
  - Geant4/externals/clhep/include/CLHEP/Geometry/Transform3D.h
- G4RotationMatrix: Represents 3D rotations in the object's coordinate system.
  - A 'typedef' of CLHEP's HepRotation class, which is defined in the following location in Geant4 distribution code:
  - Geant4/externals/clhep/include/CLHEP/Vector/Rotation.h
- G4ThreeVector: Represents 3D vectors.
  - A 'typedef' of CLHEP's Hep3Vector class, which is defined in the following location in Geant4 distribution code:
  - Geant4/externals/clhep/include/CLHEP/Vector/ThreeVector.h

Example of G4Transform3D / G4RotationMatrix / G4ThreeVector

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;
```

Constructor

Matrix that rotates twice continuously about the y-axis

Matrix that rotates about the Z-axis following the y-axis.

The product of two matrices

Reset the values of the rot3 matrix

Each component of the rot1 matrix

Example of G4Transform3D / G4RotationMatrix / G4ThreeVector

G4ThreeVector (#include "G4ThreeVector.hh”)

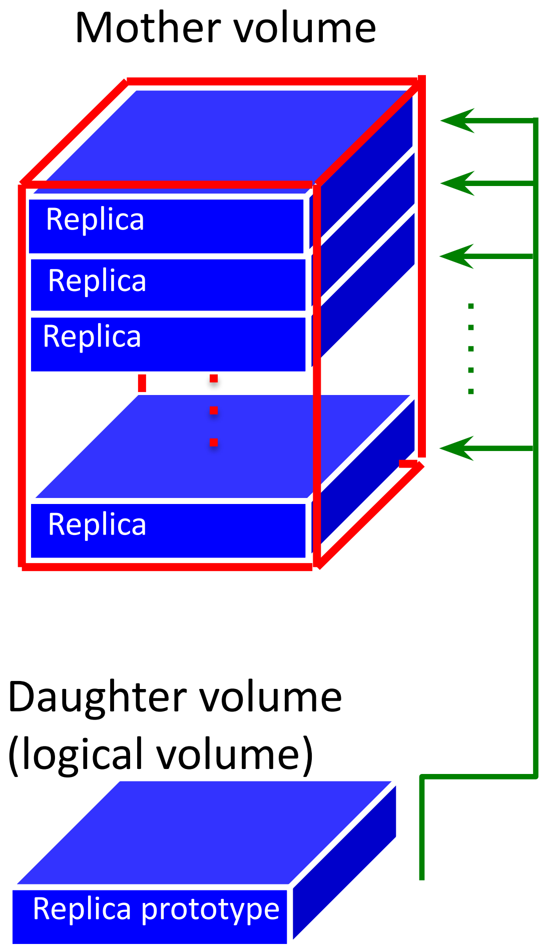
```
G4ThreeVector v1{1.0, 2.0, 3.0};
G4cout << v1.x() << ", " << v1.y() << ", " << v1.z() << G4endl;
G4cout << v1.theta() << ", " << v1.phi() << ", " << v1.mag() << G4endl;
G4ThreeVector v2{};
v2.setX(10.); v2.setY(20.); v2.setZ(30.);
G4cout << v2.x() << ", " << v2.y() << ", " << v2.z() << G4endl;
```

G4Transform3D (#include "G4Transform3D.hh”)

```
G4Transform3D trans(rot1, v1);  // transform3D using rot1 and v1
G4VPhysicalVolume* phantom_phys;
phantom_phys
        = new G4PVPlacement(trans,                  // G4Transform3D
                            phantom_log,            // G4LogicalVolume
                            "phantom",              // name
                            experimentalHall_log,   // pMotherLogical
                            false,                  // pMany
                            0);
```

Replicated Volume

- Similar to "G4PVPlacement," "G4PVReplica" (Replica) is a method for placing daughter volumes within a mother volume.
- While "G4PVPlacement" places volumes one by one, Replica allows you to place multiple daughter volumes at once.
- It is used when you want to completely fill the interior of the mother volume with daughter objects of the same size and shape.
- When dealing with a large number of placements, using Replica can significantly save memory.
- [Note]
    - The mother volume of a Replica can be either a logical volume or a physical volume.
    - You can place volumes using G4PVPlacement inside the daughter volume of a Replica. However, these volumes should not intersect or overlap with the mother volume or other Replicas.
    - Placing parameterized volumes (discussed later) within the daughter volume of a Replica is not allowed.



Replicated Volume

Constructor

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

Daguhter volume (Replica)

Mother volume (Colud be G4VPhysicalVolume*)

Arrangement direction

Number of Replicated Volumes

Thickness of Replicated Volume Arrangement Direction

Thickness of Replicated Volume Arrangement Direction

- Placement of the replica volume along the X-axis direction.

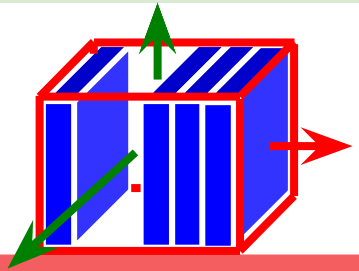
```
G4PVReplica repX("Linear Array”,pRepLogical, pContainingMother, kXAxis, 5, 10*mm);
```

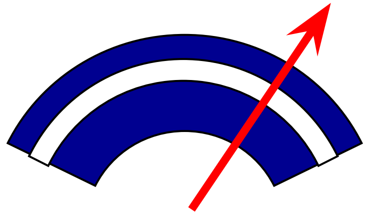
- Placement of the replica volume along the Y-axis direction.

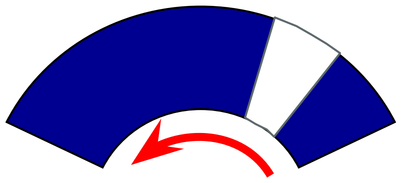
```
G4PVReplica repR("RSlices”, pRepRLogical, pContainingMother, kRho, 5, 10*mm, 0);
```

- Placement of the replica volume along the Y-axis direction.

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

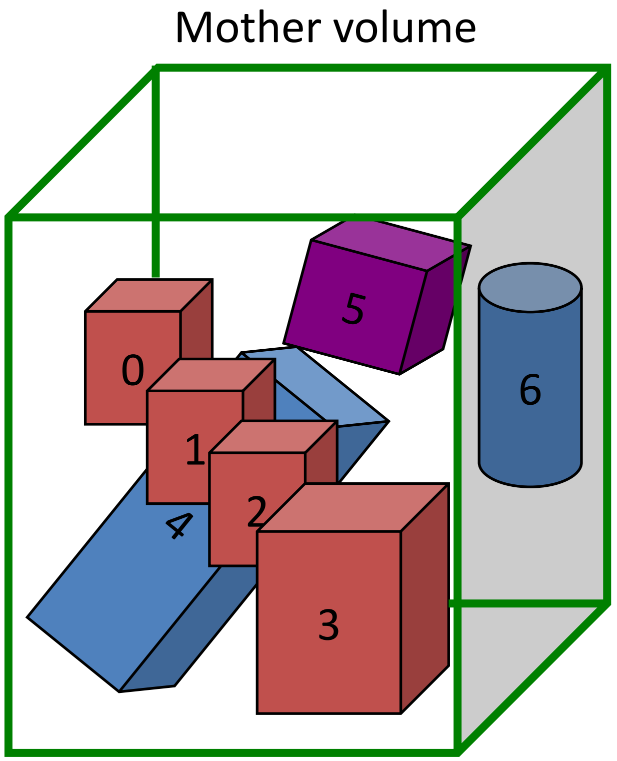






Parameterized Volume

- While replicated volumes come with restrictions,
- In contrast, parameterized volume offers greater flexibility:
  - Objects can be positioned by adjusting their sizes.
  - Objects can be arranged by modifying their shapes.
  - Objects can be placed with varying materials.
  - And more.
- Typical use cases encompass:
  - Complex-shaped geometry.
    - Regular or irregular arrangements of identical solids.
  - Medical applications
    - Representing the Multi-Leaf Collimator (MLC) as an assembly of rectangular solid leaves with varying positions.
- To implement parameterized volumes, you need to create your own custom class, inheriting from G4VPVParameterisation, to define how the variations should be applied within the geometry.



Objects with number are daughter volumes

G4PVParameterised class

Constructor

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

Daguhter volume (Replica)

Mother volume

Arrangement direction

ArrangemYour own class inheriting from G4VPVParameterisationent direction

- Within the parent volume, replicate daughter objects 'nReplicas' times according to the parameterized placement description in your class inheriting from G4VPVParameterisation (pParam).
- Each replicated object is assigned a 'copy number' starting from zero.
- pAxis specifies the primary direction along which replicated daughter objects are predominantly arranged. This information is used internally for simulation acceleration.
  - kXAxis, kYAxis, kZAxis: Arranged along the X, Y, or Z coordinate axes, respectively.
  - kUndefined: Not aligned with any specific coordinate axis.

Patient geometry

- Collection of voxels with same sizes without gaps, with varying materials from CT values
- Use nested parameterization
    - Replicated volumes for the first and second axes sequentially
    - One dimensional parameterization along the third axis
    - Requires less memory for geometry optimization and gives faster navigation
    - More detail will be covered in a later lecture.

Other topics related to Geometry

Data Retrieval in Geant4

- Customized Data Retrieval
    - Every Geant4 simulation is unique, and the information you need to retrieve varies.
    - You have to define your data retrieval approach based on your specific needs.
- Multiple methods
    - Geant4 offers multiple methods to define how you retrieve your simulation data.
    - These methods include: Built-in scoring commands, Use scorers in the tracking volume, Assign G4VSensitiveDetector to a volume
- Detailed explanations will be presented in the later lecture.

Enhancing Simulation Efficiency with Production Cuts

- To optimize simulation speed, specific production thresholds ("cuts") can be assigned to different parts of the geometry.
- Geant4 introduces the concept of Regions to address this need.
- You can set different production cuts for each geometrical region in your simulation.
- The world volume is treated as a region by default, and you are not allowed to define a region for the world logical volume.

Visualization in Geant4

- Geant4 provides a visualization system for viewing your geometry, trajectories, and more.
- You can set Visualization Attributes to your logical volumes
  - Visibility settings (Visible/Invisible)
  - Color
  - Wireframe/Solid rendering
  - etc

Uniquely Identifying a Volume

- Geant4 represents geometry as a hierarchy of volumes.
- This hierarchy can be complex, with volumes containing other volumes.
- To navigate this hierarchy effectively, Geant4 provides a valuable tool: **Touchables**.
- Touchables serve the purpose of providing a unique identification for each geometry element.
- They allow you to retrieve information about the position of a volume during particle tracking.

Summary: Steps of geometry definition

1. Prepare all materials.
2. Define Solids.
3. Define Logical Volume.
4. Define Physical Volume.
5. Associate a field to geometry if nescesary.
6. Define how you will retrieve information.
7. Customize the visual representation of your geometry.
8. Assign specific production cuts and etc..