Geant4 Geometry

南開大學
Nankai University

徐 音

2014-8-12

Outline

- Material
- Define detector geometry
- Debugging geometries
- Magnetic field
- Sensitive Detector

Email:
xuyin@nankai.edu.cn

System of Units

- System of units are defined in CLHEP, based on:
  - millimetre (**mm**), nanosecond (**ns**), Mega eV (**MeV**), positron charge (**eplus**) degree Kelvin (**kelvin**), the amount of substance (**mole**), radian (**radian**)…
- All other units are computed from the basic ones.
- Divide a variable by a unit you want to get.
  - G4cout << dE / MeV << " (MeV)" << G4endl;
- If no unit is specified, the *internal* G4 unit will be used
- To give a number, unit must be "multiplied" to the number.
  - for example :

```
G4double width = 12.5*m;
G4double density = 2.7*g/cm3;
```


Email:
xuyin@nankai.edu.cn

Definition of Materials

- Different kinds of materials can be described:
  - isotopes <-> G4Isotope
  - elements <-> G4Element
  - molecules, compounds and mixtures <-> G4Material

- **G4Isotope** and **G4Element** describe the properties of the *atoms*:
  - Atomic number, number of nucleons, mass of a mole, shell energies
  - Cross-sections per atoms, etc...
- **G4Material** describes the *macroscopic* properties of the matter:
  - temperature, pressure, state, density
  - Radiation length, absorption length, etc...

Email:
xuyin@nankai.edu.cn

Material: molecule

```
a = 1.01*g/mole;
G4Element* elH =
          new G4Element("Hydrogen",symbol="H",z=1.,a);
a = 16.00*g/mole;
G4Element* elO =
          new G4Element("Oxygen",symbol="O",z=8.,a);

density = 1.000*g/cm3;
G4Material* H2O =
          new G4Material("Water",density,ncomp=2);
G4int natoms;
H2O->AddElement(elH, natoms=2);
H2O->AddElement(elO, natoms=1);
```


Email:
xuyin@nankai.edu.cn

Material: mixture

```
G4Element* elC = …;
G4Material* SiO2 = ...;
G4Material* H2O = …;

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


Email:
xuyin@nankai.edu.cn

Material: Isotope

An element can be created according to user defined abundances

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

G4Element* elenrichedU =
      new G4Element("enriched U", symbol="U" , ncomponents=2);
elenrichedU->AddIsotope(isoU235, abundance=80.*perCent);
elenrichedU->AddIsotope(isoU235, abundance=20.*perCent);

G4Material* matenrichedU=
      new G4Material("Ufornuclear powergeneration",density=
      19.050*g/cm3 , ncomponents = 1 , kStateSolid );
matenrichedU>AddElement( elenrichedU , fractionmass = 1 );
```


Email:
xuyin@nankai.edu.cn

Material from NIST

NIST database:
http://physics.nist.gov/PhysRefData

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

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

Element/Material is retrieved from Geant4 material database by its name:

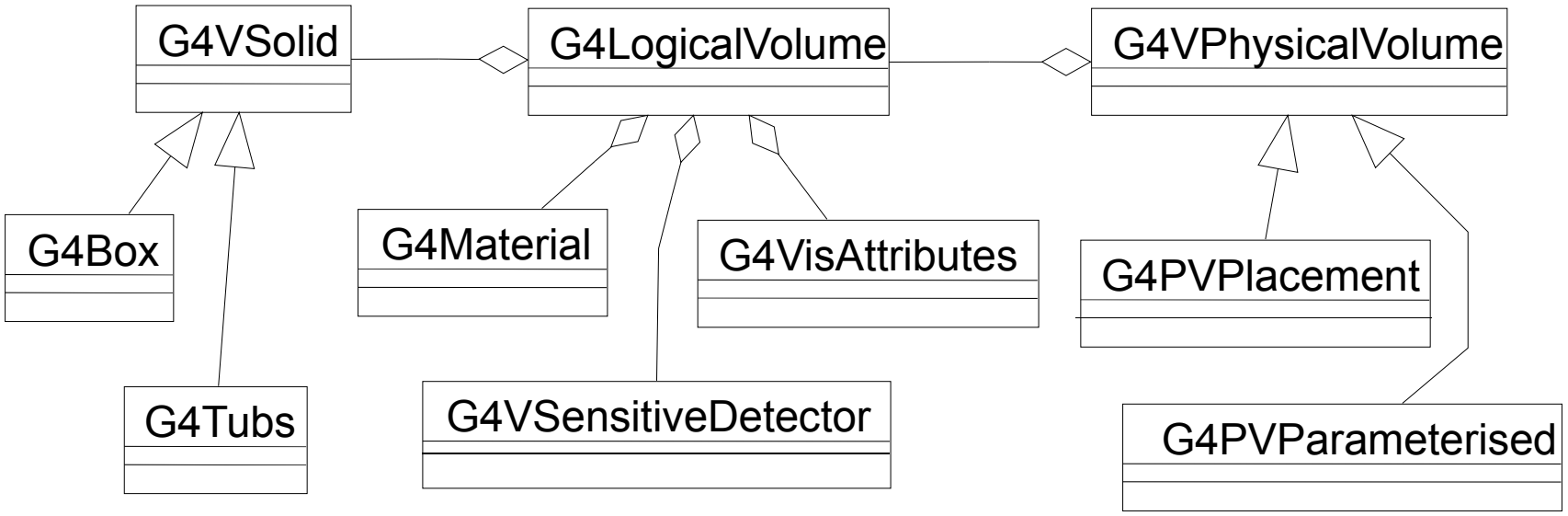
```
G4NistManager* manager = G4NistManager::Instance();
G4Element* elC
      = mananger->FindOrBuildElement("G4_C");
G4Material* matWater
      = mananger->FindOrBuildMaterial("G4_WATER");
```


Email:
xuyin@nankai.edu.cn

Detector geometry

Three conceptual layers

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



Email:
xuyin@nankai.edu.cn

Describe your detector

- Derive your own concrete class from G4VUserDetectorConstruction abstract base class.
- Implement the method Construct()
  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 volumes (optional)
  7) Define visualization attributes for the detector elements (optional)
  8) Define regions (optional)
- Set your construction class to G4RunManager

Email:
xuyin@nankai.edu.cn

main()

```
{
    // Construct the default run manager
    G4RunManager* runManager = new G4RunManager;

    // Set mandatory user initialization classes
    MyDetectorConstruction* detector = new MyDetectorConstruction;
    runManager->SetUserInitialization(detector);
    runManager->SetUserInitialization(new MyPhysicsList);

    // Set mandatory user action classes
    runManager->SetUserAction(new MyPrimaryGeneratorAction);

    // Set optional user action classes
    MyEventAction* eventAction = new MyEventAction();
    runManager->SetUserAction(eventAction);
    MyRunAction* runAction = new MyRunAction();
    runManager->SetUserAction(runAction);
}
```


Email:
xuyin@nankai.edu.cn

Your detector construction

```
#ifndef MyDetctorConstruction_h
#define MyDetctorConstruction_h 1
#include "G4VUserDetectorConstruction.hh"
class MyDetctorConstruction
        : public G4VUserDetectorConstruction
 {
public:
        G4VUserDetectorConstruction();
        virtual ~G4VUserDetectorConstruction();
        virtual G4VPhysicalVolume* Construct();
    //Construct() should return the pointer of the world physical volume.
    //The world physical volume represents all of your geometry setup.
public:
// set/get methods if needed
private:
// granular private methods if needed // data members if needed
};
 #endif
```


Email:
xuyin@nankai.edu.cn

Describe your detector

- Basic strategy

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

- A unique physical volume which represents the experimental area must exist and fully contains all other components
  - The world volume

Email:
xuyin@nankai.edu.cn

Solids

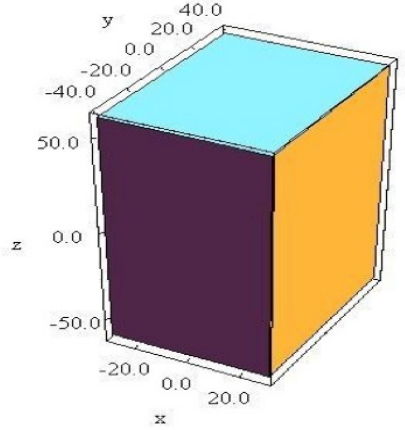
Solids defined in Geant4:

- CSG (Constructive Solid Geometry) solids
  - G4Box, G4Tubs, G4Cons, G4Trd, …
- Specific solids (CSG like)
  - G4Polycone, G4Polyhedra, G4Hype, ...
- BREP (Boundary Represented) solids
  - G4BREPSolidPolycone, G4BSplineSurface, ...
  - Any order surface
- Boolean solids
  - G4UnionSolid, G4SubtractionSolid, ...

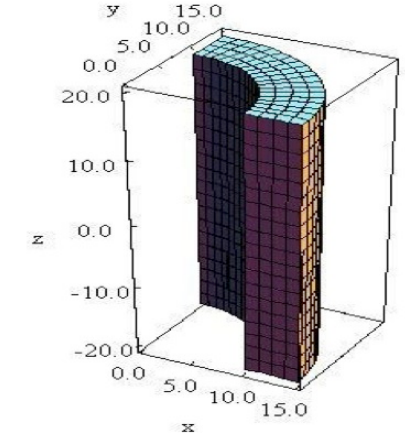
Email:
xuyin@nankai.edu.cn

CSG: G4Box, G4Tubs

```
G4Box(const G4String &pname,
                  G4double half_x,
                  G4double half_y,
                  G4double half_z);
```

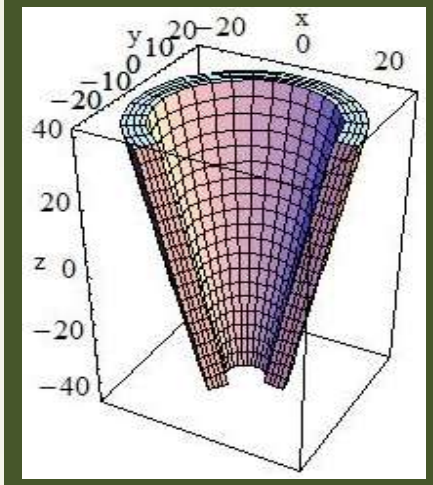



```
G4Tubs(const G4String &pname,
                  G4double pRmin,
                  G4double pRmax,
                  G4double pDz,
                  G4double pSphi,
                  G4double pDphi);
```

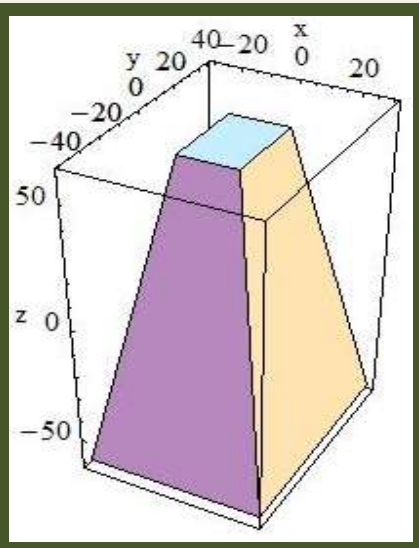


Email:
xuyin@nankai.edu.cn

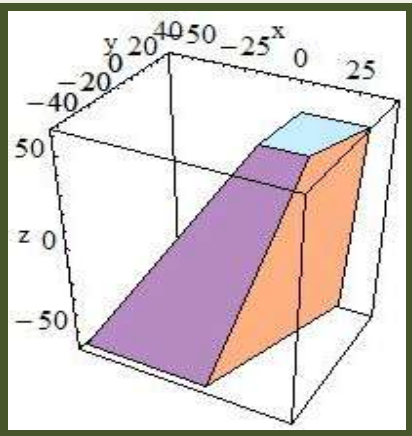
Other CSG solids



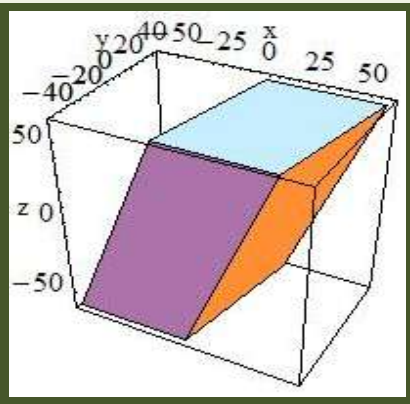
G4Cons



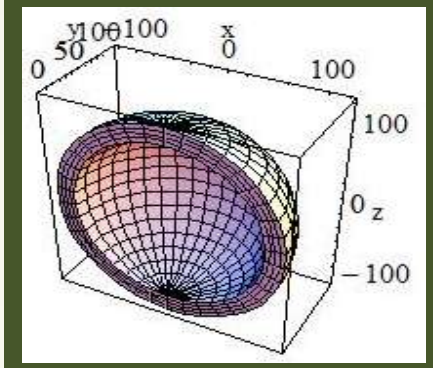
G4Trd



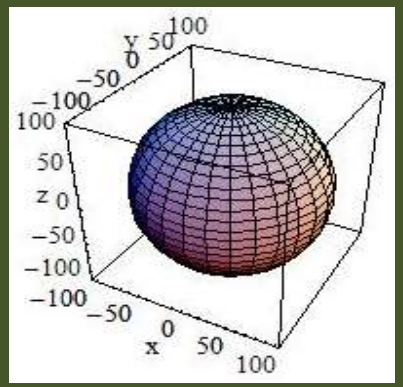
G4Trap



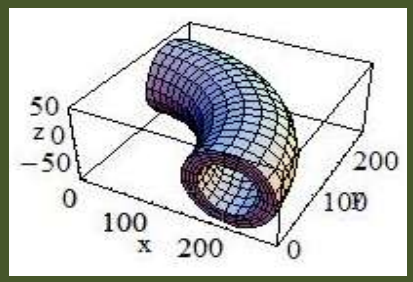
G4Para
(parallelepiped)



G4Sphere



G4Orb

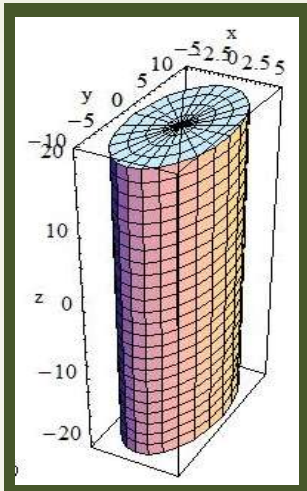


G4Torus

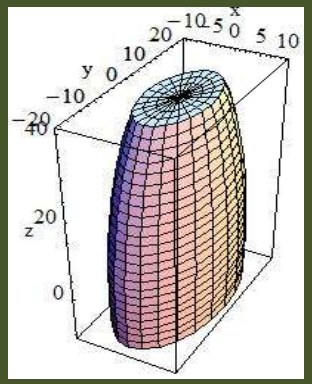
Consult to
Section 4.1.2 of Geant4 Application
for all available shapes.

Email:
xuyin@nankai.edu.cn

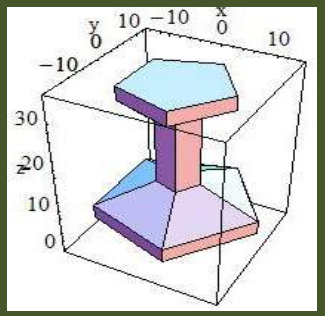
Specific CSG solids



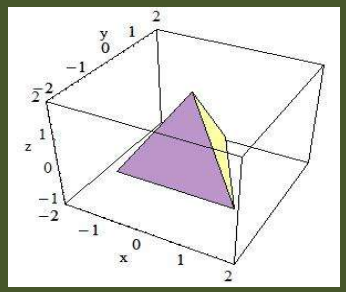
G4EllipticalTube



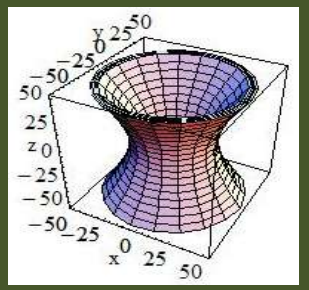
G4Ellipsoid



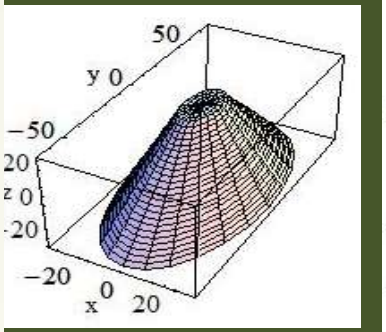
G4Polyhedra



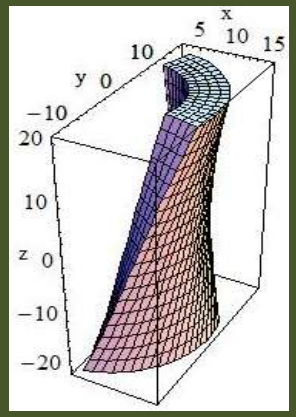
G4Tet
(tetrahedra)



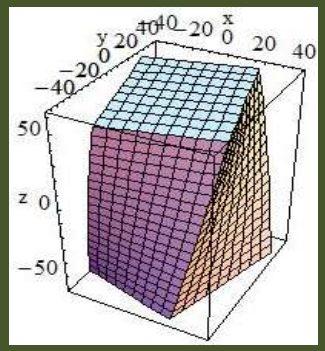
G4Hype



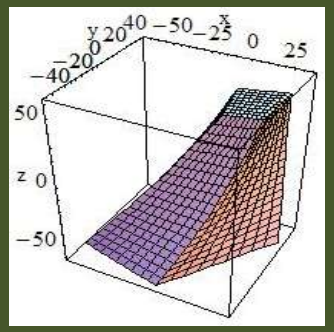
G4EllipticalCone



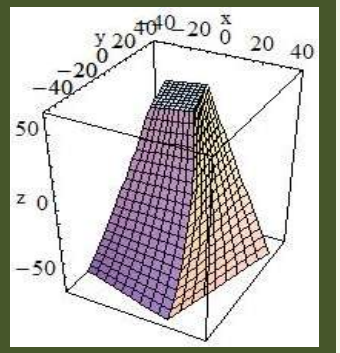
G4TwistedTubs



G4TwistedBox



G4TwistedTrap



G4TwistedTrd

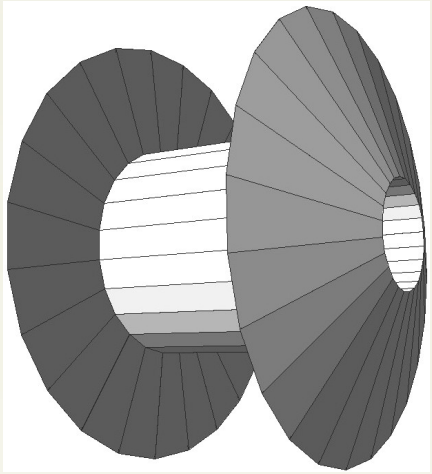
Consult to
Section 4.1.2 of Geant4 Application Deve
for all available shapes.

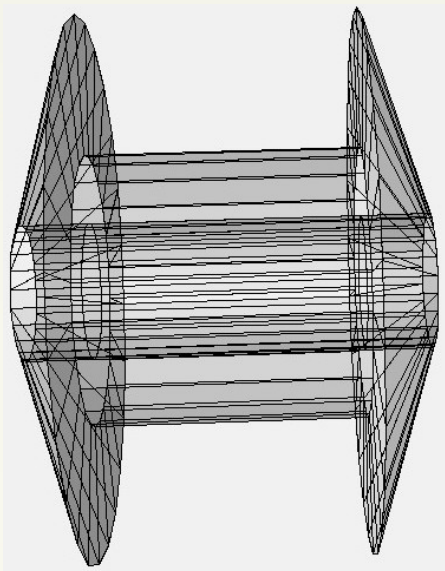
Email:
xuyin@nankai.edu.cn

BREP Solids

BREP = Boundary REPresented Solid

- Listing all its surfaces specifies a solid
  - e.g. 6 planes for a cube
- Surfaces can be
  - planar, 2nd or higher order
  - Splines, B-Splines
- Advanced BREPS built through CAD systems

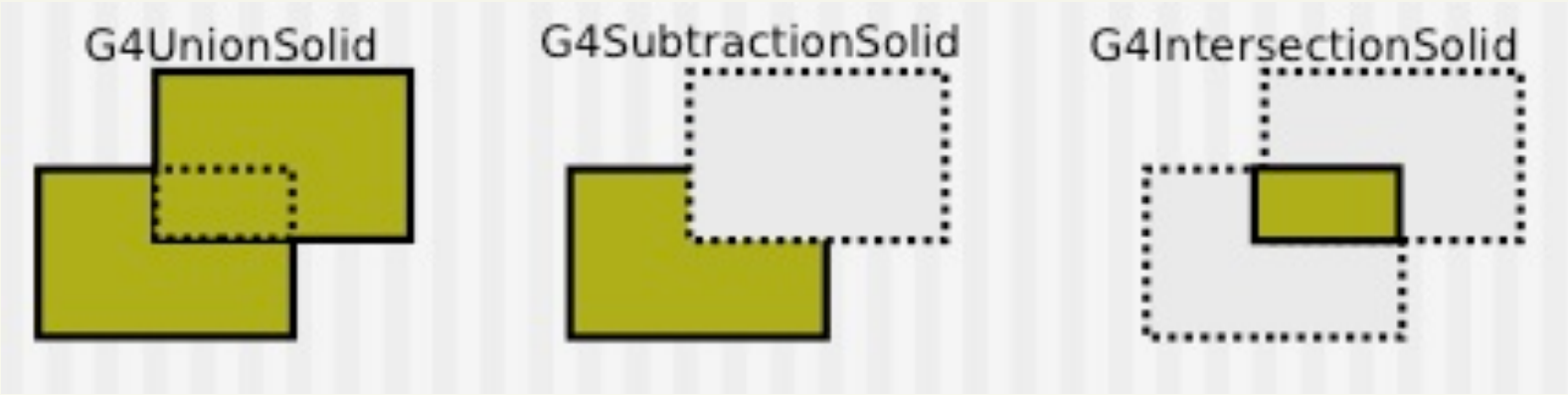




Email:
xuyin@nankai.edu.cn

Boolean Solids

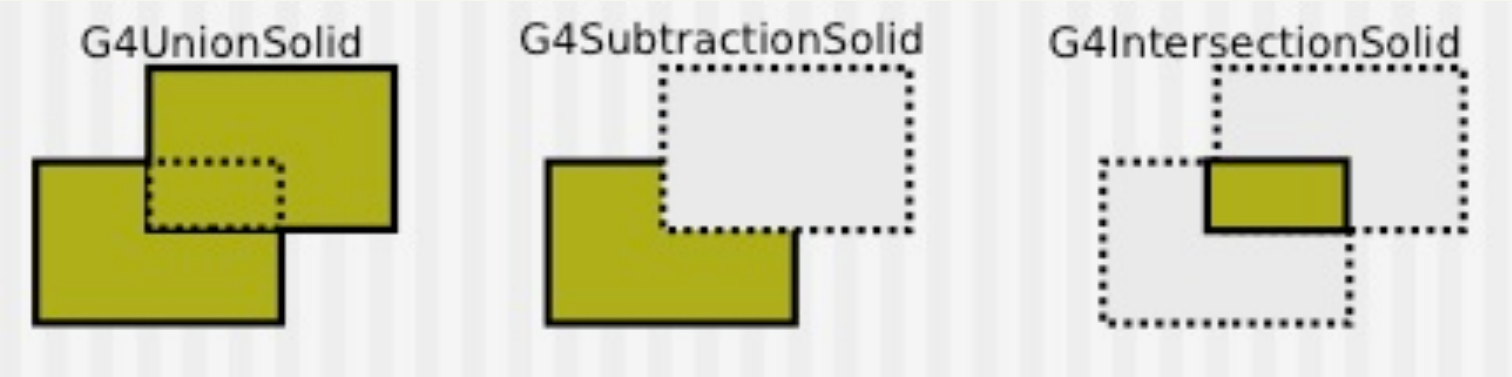
- Solids can be combined using boolean operations:
  - G4UnionSolid, G4SubtractionSolid, G4IntersectionSolid
- Solids can be either CSG or other Boolean solids
- Note: tracking cost for the navigation in a complex Boolean solid is proportional to the number of constituent solids



Email:
xuyin@nankai.edu.cn

Boolean Solids - Example

```
G4Box* box = new G4Box(“Box", 20, 30, 40);
G4Tubs* cylinder
        = new G4Tubs(“Cylinder”, 0, 50, 50, 0, 2*M_PI);
G4UnionSolid* union
        = new G4UnionSolid("Box+Cylinder", box, cylinder));
G4IntersectionSolid* intersect
        = new G4IntersectionSolid("Box*Cylinder", box, cylinder, 0,
                G4ThreeVector(30, 20, 0));
G4SubtractionSolid* subtract
        = new G4SubtractionSolid("Box-Cylinder", box, cylinder,
                0, G4ThreeVector(30, 20, 0));
```



Email:
xuyin@nankai.edu.cn

Computing volumes and masses

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

  G4double **GetCubicVolume**();

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

- Overall mass of a geometry setup (subdetector) 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.

Email:
xuyin@nankai.edu.cn

G4LogicalVolume

- Contains all information of volume except position:
  - Shape and dimension (G4VSolid)
  - Material, sensitivity, visualization attributes
  - Position of daughter volumes
  - Magnetic field, User limits
  - Shower parameterization
  - Region

- Physical volumes of same type can share a logical volume

Email:
xuyin@nankai.edu.cn

G4LogicalVolume

To create a logical volume for a given material and solid, the user must instantiate G4LogicalVolume:

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

Email:
xuyin@nankai.edu.cn

Geometrical hierarchy

Mother and daughter volumes

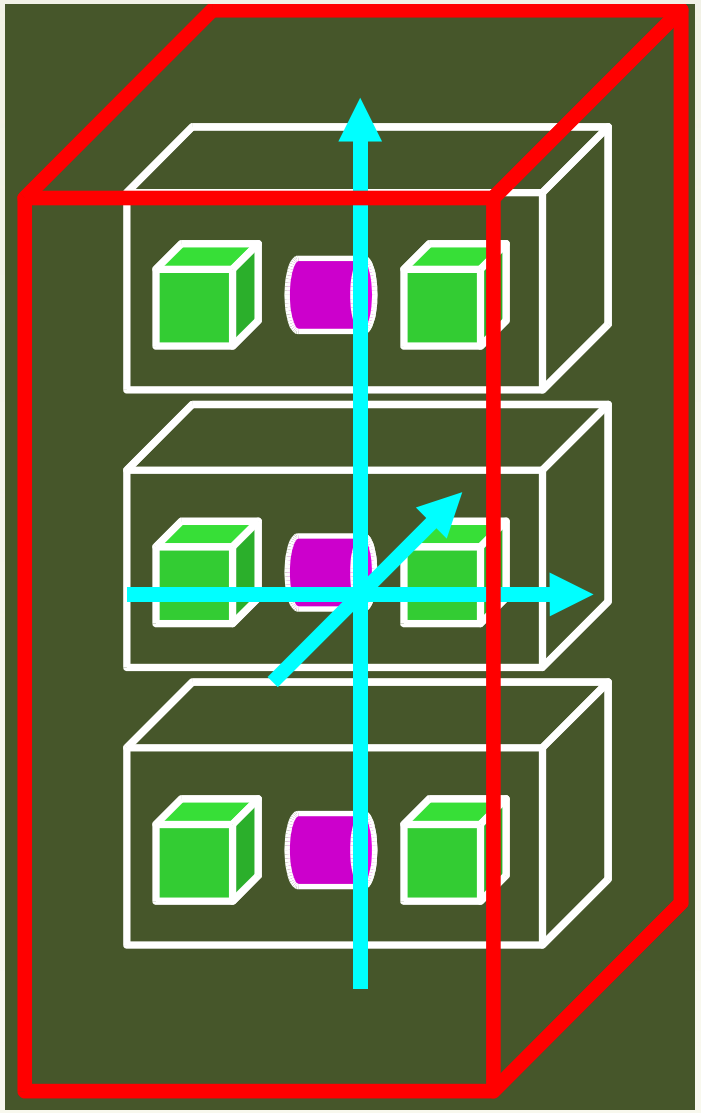
- A volume is placed in its mother volume
  - Position and rotation of the daughter volume is described with respect to the local coordinate system of the mother volume
  - The origin of the mother's local coordinate system is at the center of the mother volume
  - Daughter volumes cannot protrude from the mother volume
  - Daughter volumes cannot overlap
- One or more volumes can be placed to mother volume
- If the logical volume of the mother is placed more than once, all daughters appear by definition in all these physical instances of the mother

Email:
xuyin@nankai.edu.cn

Geometrical hierarchy

World volume = root volume of the hierarchy

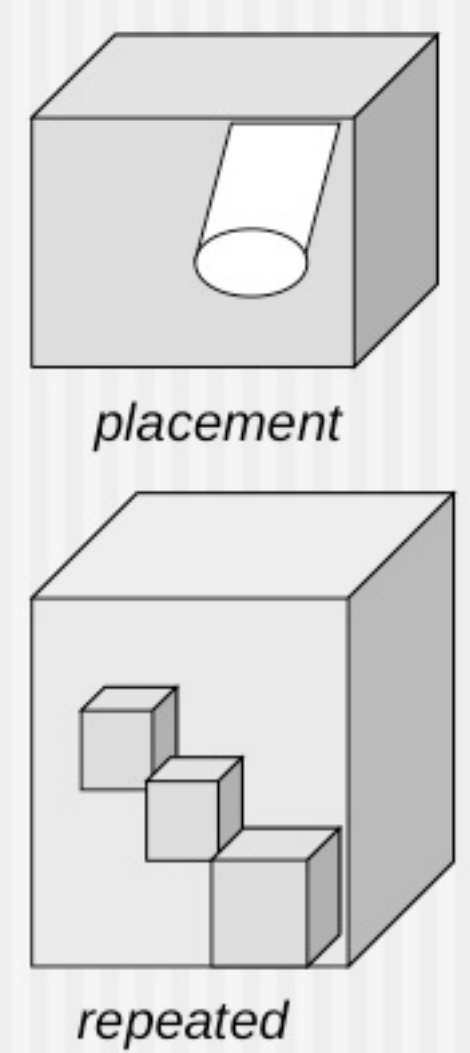
- The world volume must be a unique physical volume which fully contains all other volumes
  - The world defines the global coordinate system
  - The origin of the global coordinate system is at the center of the world volume
  - Should not share any surface with contained geometry



Email:
xuyin@nankai.edu.cn

Physical Volumes

- Placement: it is one positioned volume
  - Represents one “real” volume
- Repeated: a volume placed many times
  - can represent any number of volumes
  - Replica and Division
    - simple repetition along one axis
  - Parameterized
    - Repetition with respect to copy number
- A **mother** volume can contain **either**
  - **many placement** volumes **OR**
  - **one repeated** volume



Email:
xuyin@nankai.edu.cn

G4VPhysicalVolume

- G4VPhysicalVolume is the base class of physical volumes
  - Like G4VSolid, it as an abstract class
  - Use the inherited classes to place your logical volumes

- G4VPhysicalVolume implementations:
  - G4PVPlacement
  - G4PVParameterised
  - G4PVReplica
  - G4PVDivision

Email:
xuyin@nankai.edu.cn

G4VPhysicalVolume

- G4PVPlacement 1 Placement = One Volume
  - A volume instance positioned once in a mother volume
- G4PVParameterised 1 Parameterized = Many Volumes
  - Parameterized by the copy number
    - Shape, size, material, position and rotation can be parameterized, by implementing a concrete class of G4VPVParameterisation.
  - Reduction of memory consumption
    - Currently: parameterization can be used only for volumes that either

      a) have no further daughters or b) are identical in size & shape.
- G4PVReplica, G4PVDivision 1 Replica = Many Volumes
  - Slicing a volume into smaller pieces (if it has a symmetry)

Email:
xuyin@nankai.edu.cn

G4PVPlacement

```
G4PVPlacement(G4RotationMatrix* pRot,                  // rotation const
              G4ThreeVector& tlate,                    // translation
              G4LogicalVolume* pCurrentLogical, // volume being placed
              const G4String& pName,            // phys. Volume name
              G4LogicalVolume* pMotherLogical, // mother logical volume
              G4bool pMany,                     // not used
              G4int pCopyNo,                    // position (copy) number
              G4bool pSurfChk=false);           // activate overlap checking
```

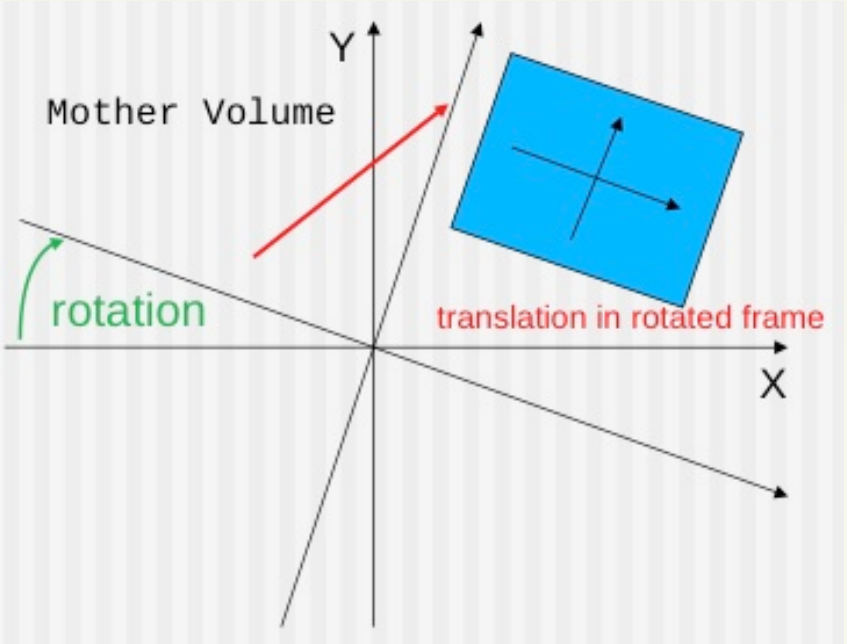
- *Single volume positioned relatively to the mother volume*
  - *In a frame rotated and translated relative to the coordinate system of the mother volume*

Email:
xuyin@nankai.edu.cn

G4PVPlacement Example

G4ThreeVector boxPosition(100.*cm, 80.*cm, 0.*cm);
G4RotationMatrix* boxRotation = new **G4RotationMatrix**;
boxRotation->**rotateZ**(30.*deg); *//Rotate around Z-axis*
//Other Rotations : rotateX(angle), rotateY(angle)

```
G4VPhysicalVolume* boxPhys
      = new G4PVPlacement(
                  boxRotation,
                  boxPosition,
                  boxLog,
                  "myBoxPhys",
                   motherLogical,
                  false,
                  1);
```



Email:
xuyin@nankai.edu.cn

G4PVParameterised

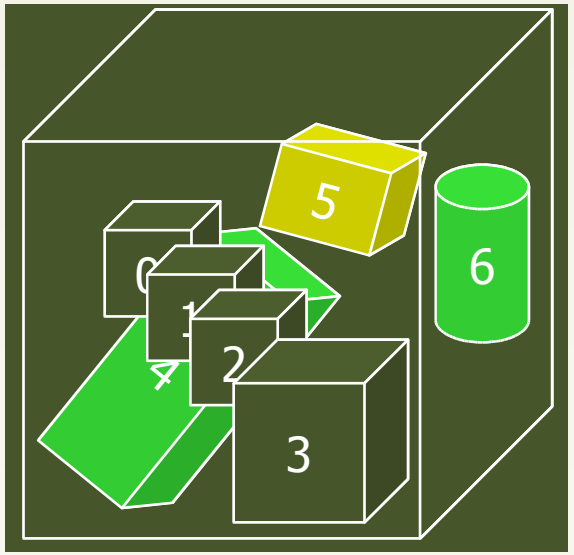
```
G4PVParameterised(const G4String& pName,
                  G4LogicalVolume* pLogical,
                  G4LogicalVolume* pMother,
                  const EAxis pAxis,
                  const G4int nReplicas,
                  G4VPVParameterisation *pParam
                  G4bool pSurfChk=false);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

- Replicates the volume **nReplicas** times using the parameterization **pParam**, within the mother volume **pMother**

Email:
xuyin@nankai.edu.cn

Parameterized Physical Volumes

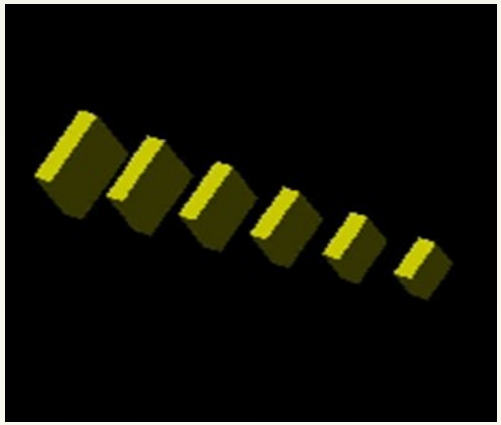
- User should implement a class derived from G4VPVParameterisation abstract base class and define following as a function of copy number
  - where it is positioned (transformation, rotation)
- Optional:
  - the size of the solid (dimensions)
  - the type of the solid, material, sensitivity, vis attributes
- All daughters must be fully contained in the mother.
- Daughters should not overlap to each other.



Email:
xuyin@nankai.edu.cn

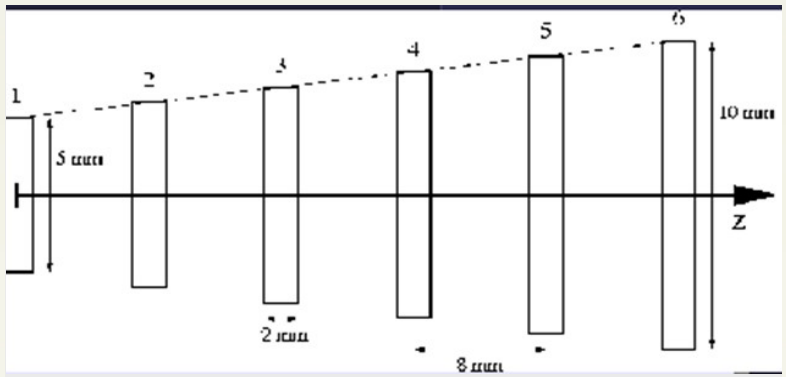
Parameterisation Example

```
G4VSolid* solidChamber = new G4Box("chamber", 100*cm, 100*cm, 10*cm);
G4LogicalVolume* logicChamber =
        new G4LogicalVolume(solidChamber, chamberMater, "Chamber", 0,
0, 0); ...
G4VPVParameterisation* chamberParam =
        new ChamberParameterisation(
                nofChambers, firstPositionZ, spacingZ,
                chamberWidth, firstLength, lastLength);
G4VPhysicalVolume* physChamber
        = new G4PVParameterised(
                "Chamber",
                logicChamber,
                logicTracker,
                kZAxis,
                NbOfChambers,
                chamberParam);
```



Email:
xuyin@nankai.edu.cn

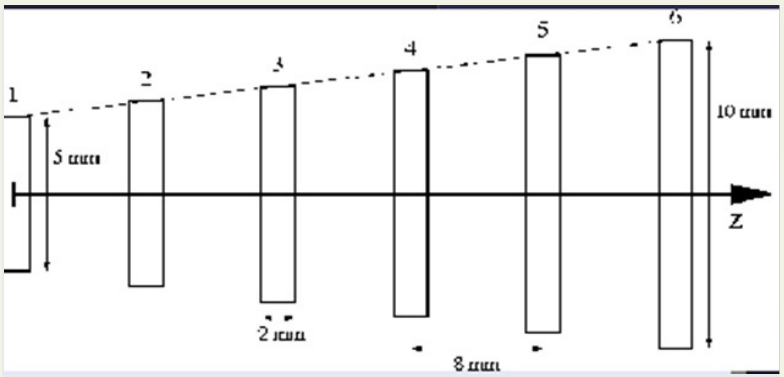
Parameterisation Example




```
class ChamberParameterisation :
public G4VPVParameterisation
{
    public:
            ChamberParameterisation(..);
            ~ChamberParameterisation();
            void ComputeTransformation(
                  G4int copyNo,
                  G4VPhysicalVolume* physVol) const;
            void ComputeDimensions (
                  G4Box& trackerLayer,
                  G4int copyNo,
                  Const G4VPhysicalVolume* physVol) const;
    private:
            G4int fNoChambers; G4double fStartZ; G4double fSpacingZ; G4double
            fHalfWidth; G4double fHalfLengthFirst; G4double fHalfLengthIncr;
};
```


Email:
xuyin@nankai.edu.cn

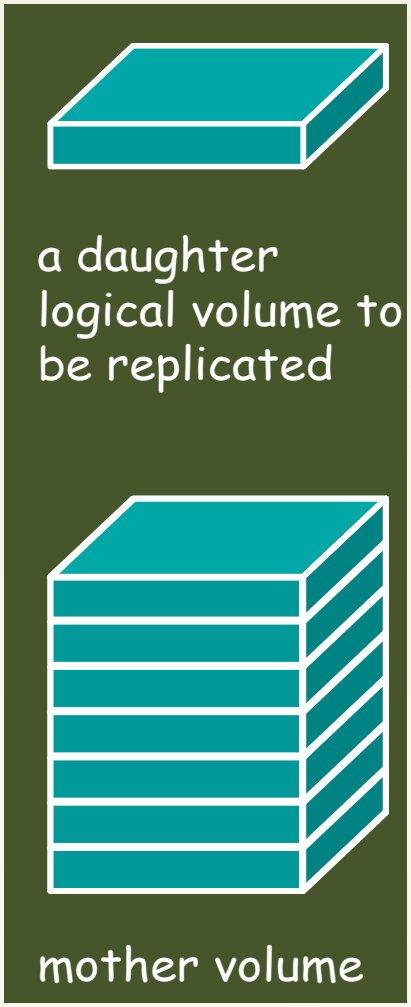
Parameterisation Example

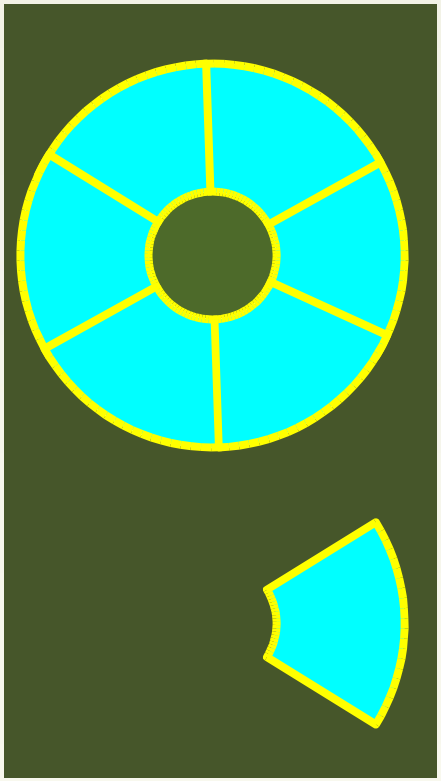


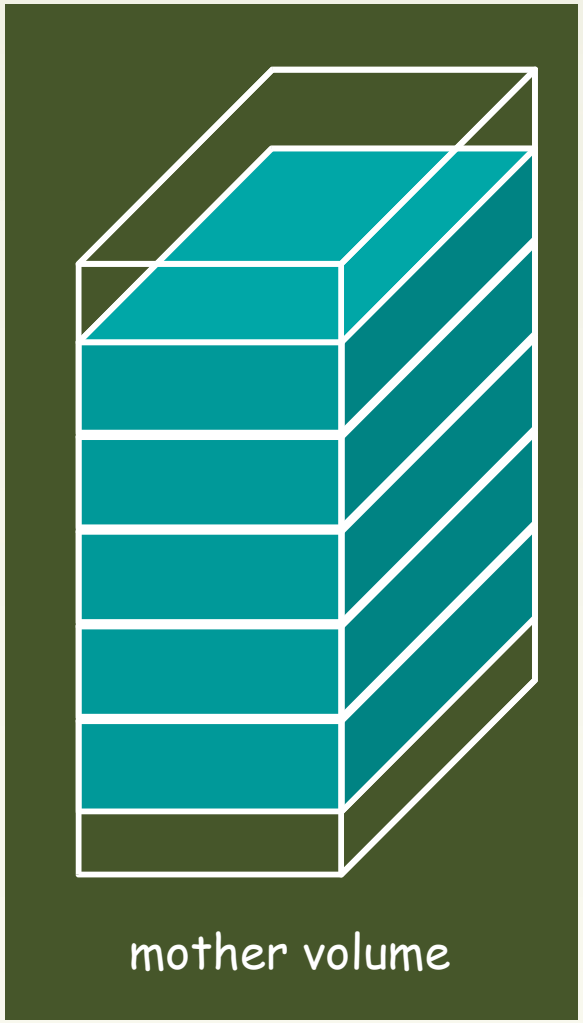

```
void ChamberParameterisation
::ComputeTransformation (
            G4int copyNo,
            G4VPhysicalVolume* physVol) const
{
      G4double zPosition= fStartZ + (copyNo+1) * fSpacingZ;
      physVol->SetTranslation(G4ThreeVector(0, 0, zPosition));
      physVol->SetRotation(0);
}
void ChamberParameterisation::ComputeDimensions (
            G4Box& trackerChamber,
            G4int copyNo,
            const G4VPhysicalVolume* physVol) const
{
      G4double halfLength = fHalfLengthFirst + copyNo * fHalfLengthIncr;
      trackerChamber.SetXHalfLength(halfLength);
      trackerChamber.SetYHalfLength(halfLength);
      trackerChamber.SetZHalfLength(fHalfWidth);
}
```


Email:
xuyin@nankai.edu.cn

Replica and Division

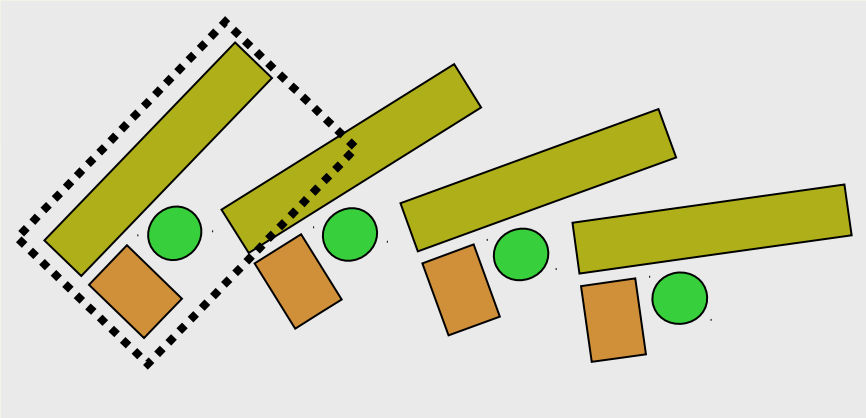


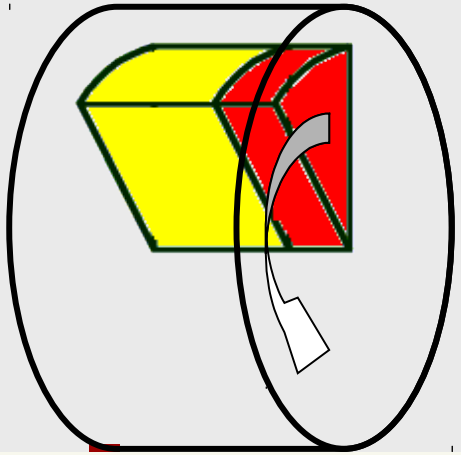


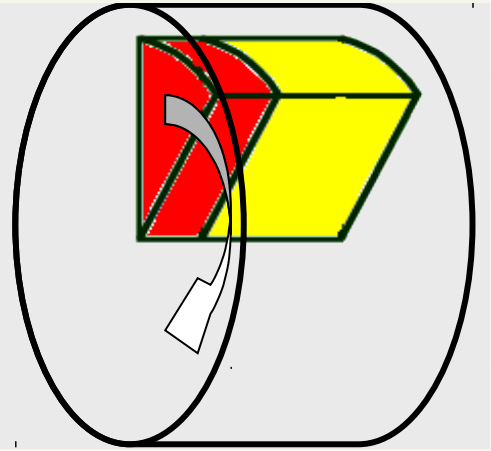


Email:
xuyin@nankai.edu.cn

Grouping(Assembly) and Reflecting



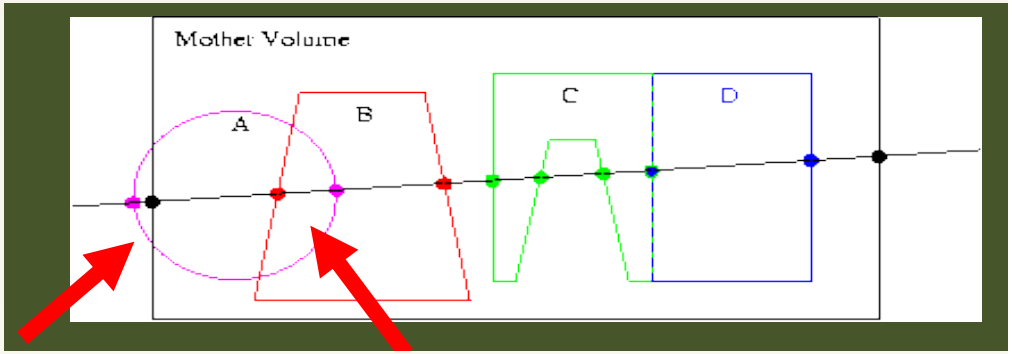




Email:
xuyin@nankai.edu.cn

Debugging geometries

- An protruding volume is a contained daughter volume which actually protrudes from its mother volume.
- Volumes are also often positioned in a same volume with the intent of not provoking intersections between themselves. When volumes in a common mother actually intersect themselves are defined as overlapping.
- The problem of detecting overlaps between volumes is bounded by the complexity of the solid models description.
- Utilities are provided for detecting wrong positioning
  - Optional checks at construction
  - Kernel run-time commands
  - Graphical tools



protruding

overlapping

Email:
xuyin@nankai.edu.cn

Optional checks at construction

- Constructors of G4PVPlacement and G4PVParameterised have an optional argument “pSurfChk”.

```
G4PVPlacement(G4RotationMatrix* pRot,
              const G4ThreeVector &tlate,
              G4LogicalVolume *pDaughterLogical,
              const G4String &pName,
              G4LogicalVolume *pMotherLogical,
              G4bool pMany, G4int pCopyNo,
              G4bool pSurfChk=false);
```

- If this flag is true, overlap check is done at the construction.
  - Some number of points are randomly sampled on the surface of creating volume.
  - Each of these points are examined
    - If it is outside of the mother volume, or
    - If it is inside of already existing other volumes in the same mother volume.
- This check requires lots of CPU time, but it is worth to try at least once when you implement your geometry of some complexity.

Email:
xuyin@nankai.edu.cn

Debugging run-time commands

Built-in run-time commands to activate verification tests for the user geometry are defined

- to start verification of geometry for overlapping regions based on a standard grid setup, limited to the first depth level
  geometry/test/run or **geometry/test/grid_test**
- applies the grid test to all depth levels (may require lots of CPU time!)
  geometry/test/recursive_test
- shoots lines according to a cylindrical pattern
  geometry/test/cylinder_test
- to shoot a line along a specified direction and position
  geometry/test/line_test
- to specify position for the **line_test**
  geometry/test/position
- to specify direction for the **line_test**
  geometry/test/direction

Email:
xuyin@nankai.edu.cn

Debugging run-time commands

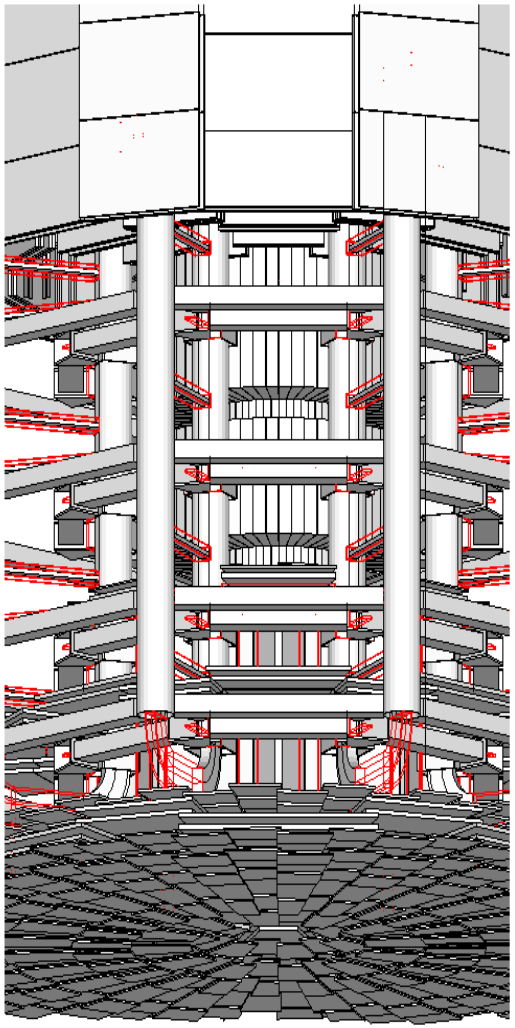
Example layout:

GeomTest: no daughter volume extending outside mother detected. GeomTest Error: Overlapping daughter volumes
The volumes Tracker[0] and Overlap[0],
both daughters of volume World[0],
appear to overlap at the following points in global coordinates: (list truncated)
length (cm) ----- start position (cm) ----- ----- end position (cm) -----
240 -240 -145.5 -145.5 0 -145.5 -145.5
Which in the mother coordinate system are:
length (cm) ----- start position (cm) ----- ----- end position (cm) ----- .. .
Which in the coordinate system of Tracker[0] are:
length (cm) ----- start position (cm) ----- ----- end position (cm) ----- .. .
Which in the coordinate system of Overlap[0] are:
length (cm) ----- start position (cm) ----- ----- end position (cm) ----- .. .

Email:
xuyin@nankai.edu.cn

Debugging run-time commands

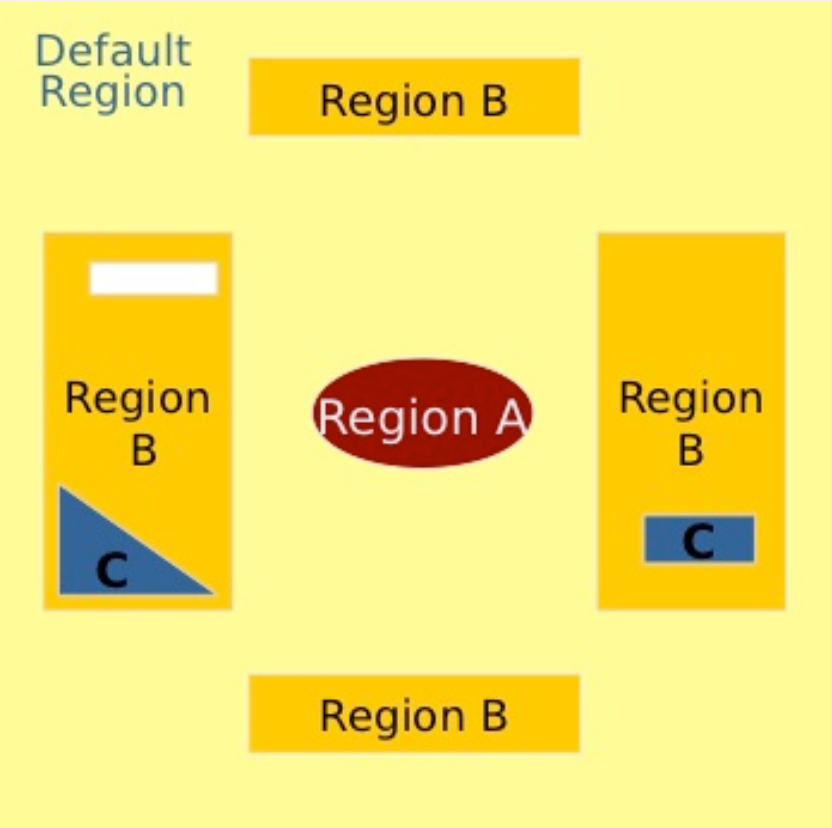
- DAVID is a graphical debugging tool for detecting potential intersections of volumes
- Accuracy of the graphical representation can be tuned to the exact geometrical description.
  - physical-volume surfaces are automatically decomposed into 3D polygons
  - intersections of the generated polygons are parsed.
  - If a polygon intersects with another one, the physical volumes associated to these polygons are highlighted in color(red is the default).
- DAVID can be downloaded from the Web as external tool for Geant4
- http://geant4.kek.jp/~tanaka/



Email:
xuyin@nankai.edu.cn

Detector Region

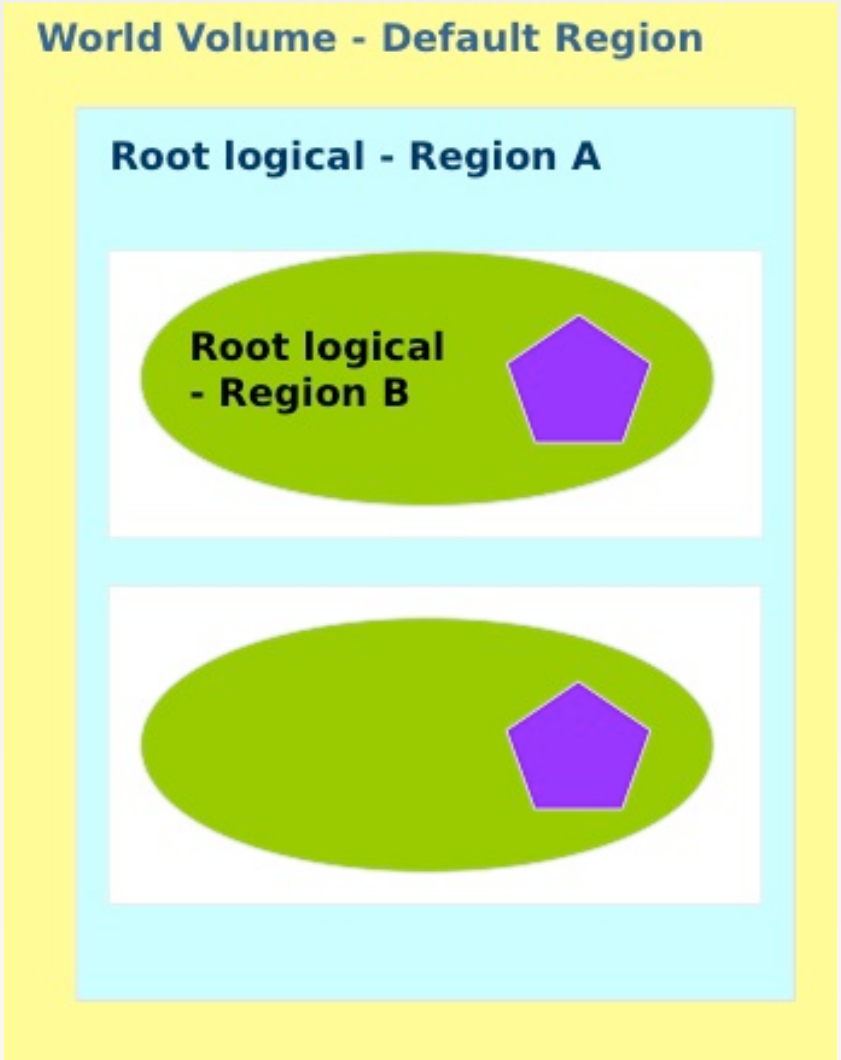
- Concept of **region**:
  - Set of geometry volumes, typically of a sub-system
    - barrel + end-caps of the calorimeter;
  - Or any group of volumes
- A region may have its unique
  - Production thresholds (cuts)
  - User limits
- Artificial limits affecting to the tracking, e.g. max step length, 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.



Email:
xuyin@nankai.edu.cn

Root Region

- **World** volume is recognized as the **default region**. (The default cuts defined in Physics list are used for it.)
  - User is not allowed to define a region to the world volume
- A logical volume becomes a root logical volume once it is assigned to a region.
  - All daughter volumes belonging to the root logical volume share the same region (and properties), unless a daughter volume itself becomes to another root logical volume
- Important restriction :
  - No logical volume can be shared by more than one regions, regardless of root volume or not



Email:
xuyin@nankai.edu.cn

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);

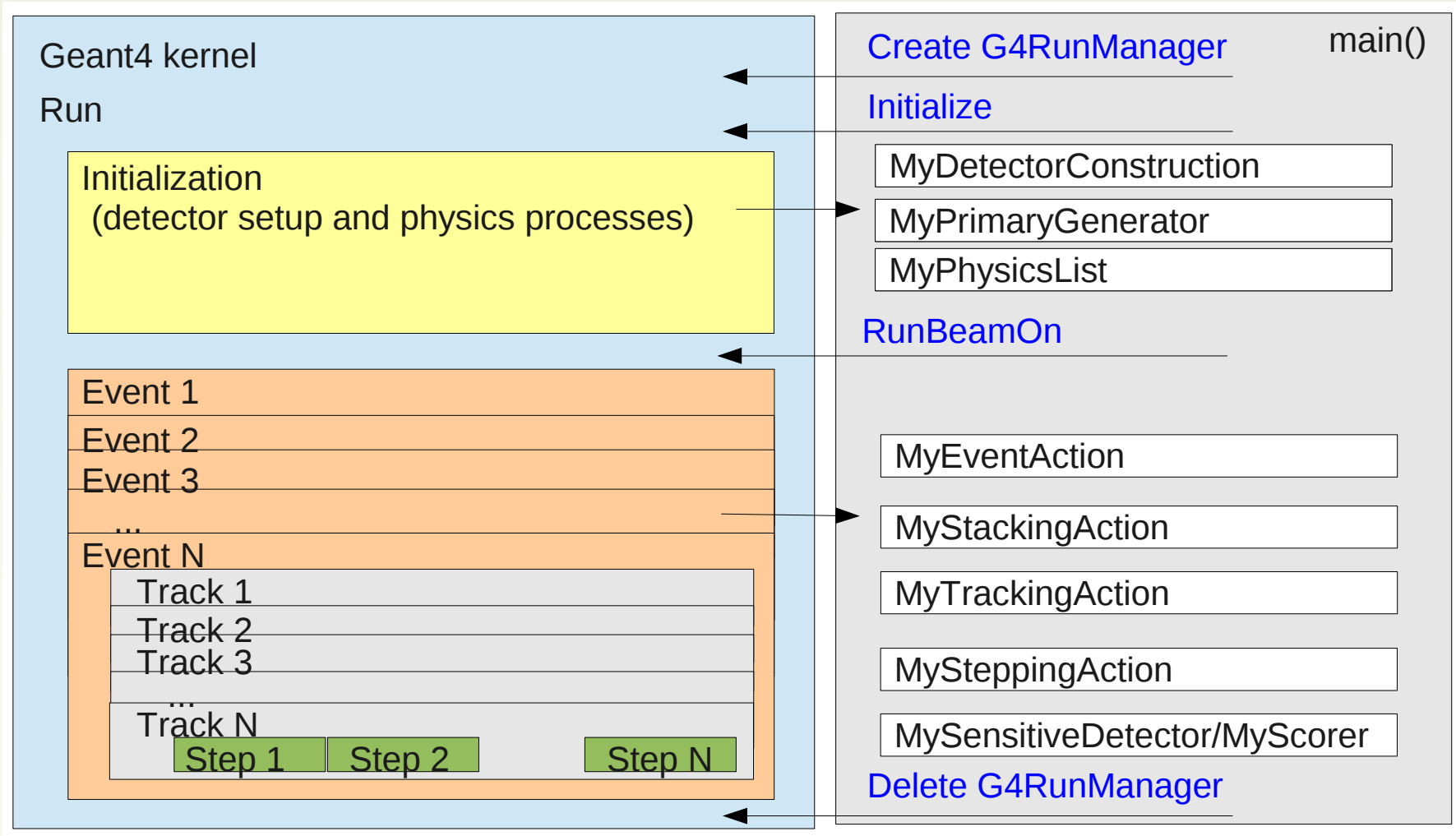
Email:
xuyin@nankai.edu.cn

Extract useful information

- Given geometry, physics and primary track generation, Geant4 does proper physics simulation “silently”.
  - You have to add a bit of code to extract information useful to you.
- There are two ways:
  - Use user hooks (G4UserTrackingAction, G4UserSteppingAction, etc.)
    - You have full access to almost all information
    - Straight-forward, but do-it-yourself
  - Use Geant4 scoring functionality
    - Assign G4VSensitiveDetector to a volume
    - Hit is a snapshot of the physical interaction of a track or an accumulation of interactions of tracks in the sensitive (or interested) part of your detector.
    - Hits collection is automatically stored in G4Event object, and automatically accumulated if user-defined Run object is used.
    - Use user hooks (G4UserEventAction, G4UserRunAction) to get event / run summary

Email:
xuyin@nankai.edu.cn

User Application and Geant4 Kernel



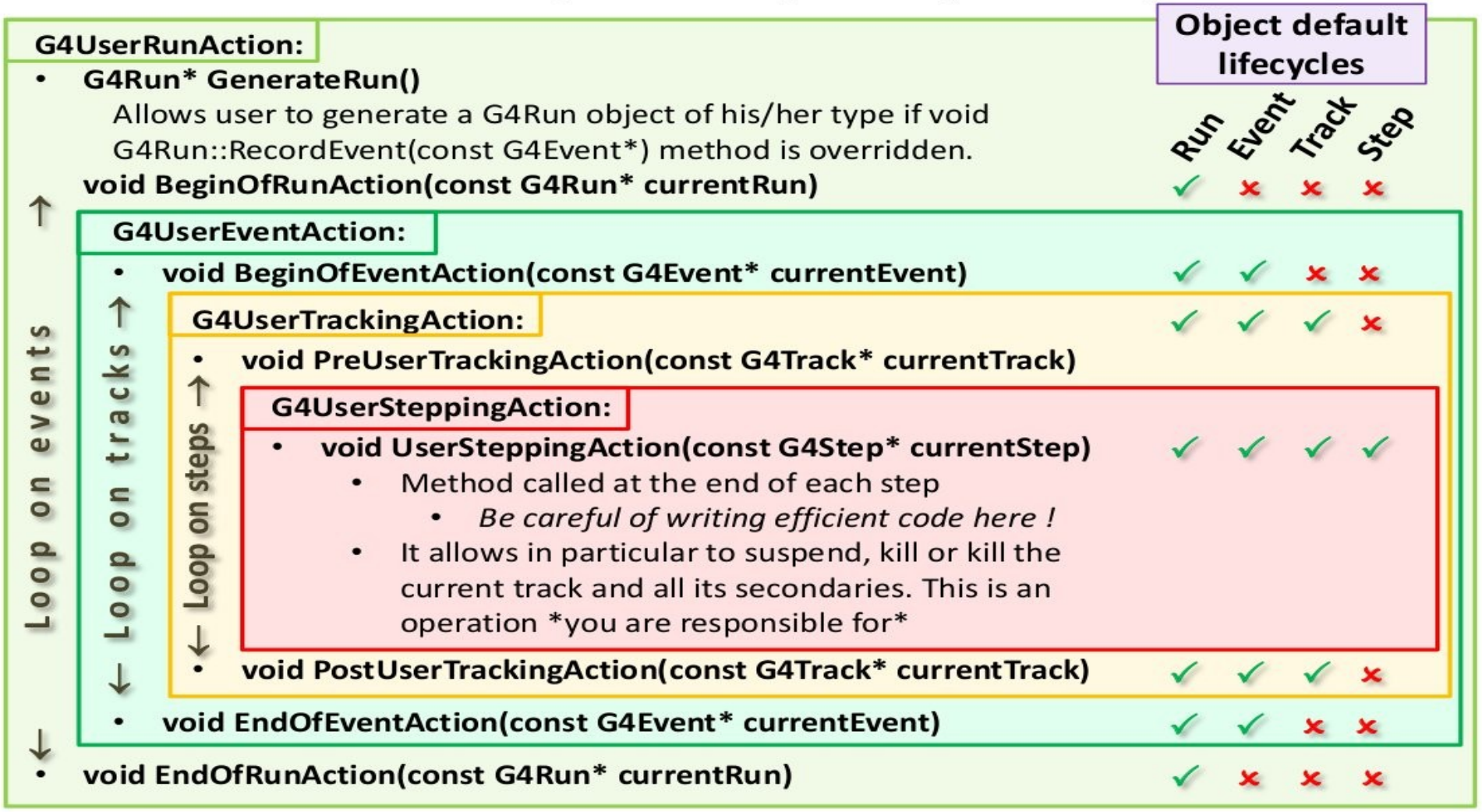
Email:
xuyin@nankai.edu.cn

Getting Information from Geant4 Objects

- At each phase of run processing user can access the corresponding Geant4 objects:
  - G4Run, G4Event, G4Track, G4Step, G4StepPoint
  - Note that the objects are provided via constant pointer and so they cannot be modified in the user functions

Email:
xuyin@nankai.edu.cn

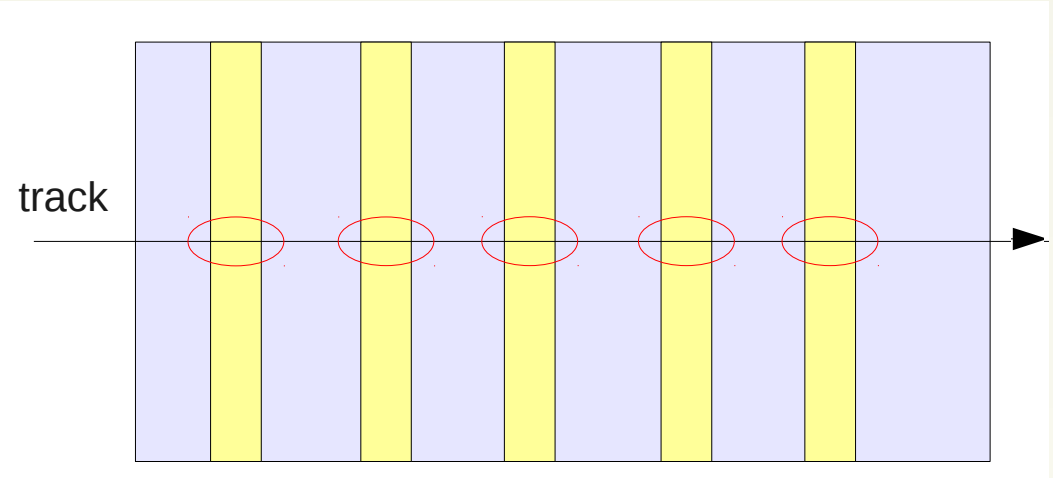
Sequence invocation of users actions
and related (default) lifecycle objects



Email:
xuyin@nankai.edu.cn

Sensitive Detector

- A sensitive detector (G4VSensitiveDetector) can be assigned to a logical volume (G4LogicalVolume).
  - The sensitive detectors are invoked only when a step takes place in a logical volume that they are assigned to
    - Users can implement their own sensitive detector classes



The sensitive detector will be invoked in all steps inside yellow layers

An example of an calorimeter with a sensitive detector assigned to yellow layers

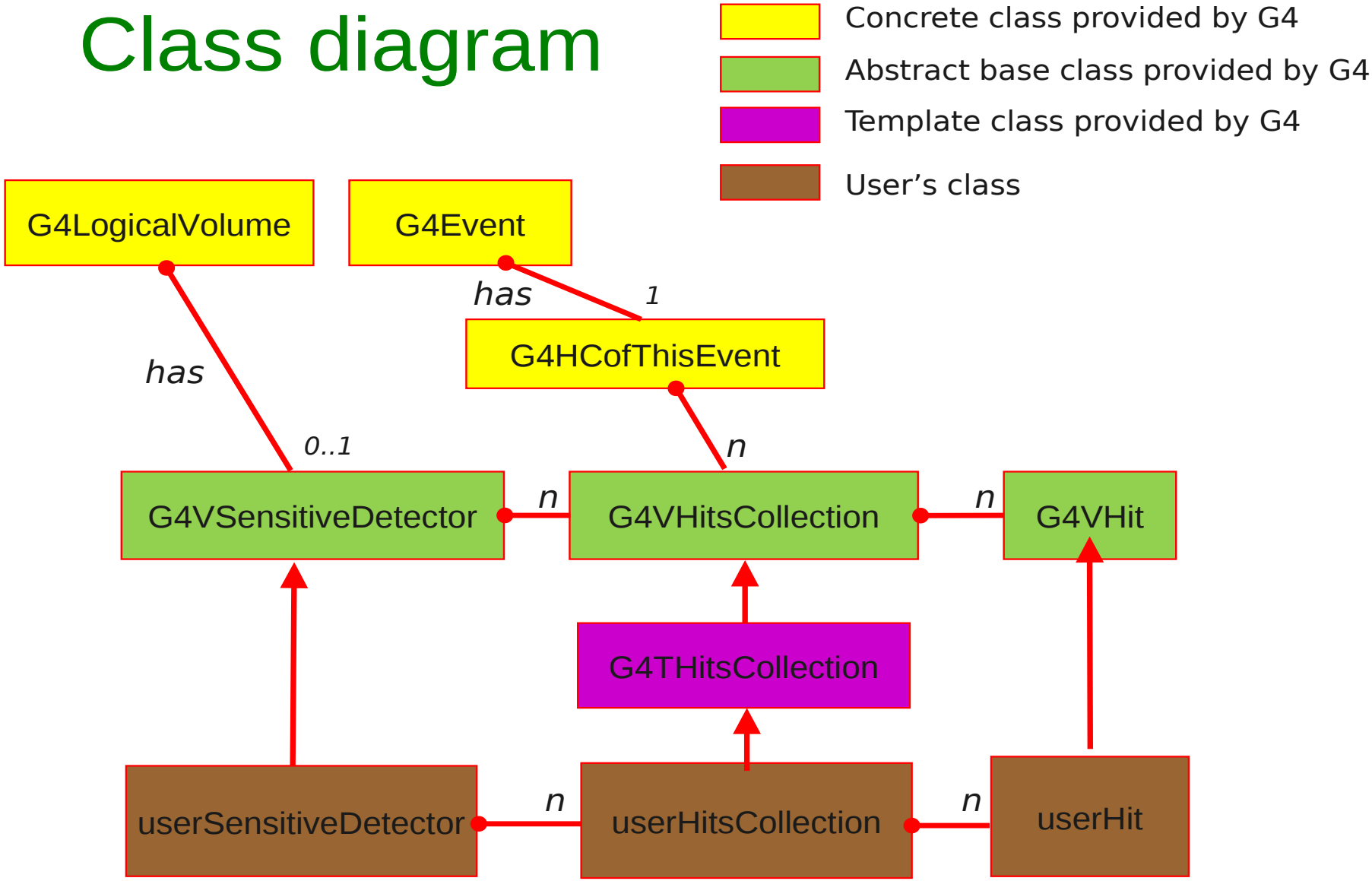
Email:
xuyin@nankai.edu.cn

Hit and Hits Collection

- When a step takes place in a sensitive logical volume, a user sensitive detector function ProcessHits(..) is invoked wher user can access to G4Step object and register **hits**
- Hit is a snapshot of the physical interaction of a track or an accumulation of interactions of tracks in the sensitive region of your detector
  - You can store various types information: position and time of the step, momentum and energy of the track, energy deposition of the step, geometrical information, ...
- Hit objects (G4VHit) must be stored in a dedicated collection (G4THitsCollection) which is available in the sensitive detector object.
- The collection will be the associated to a G4Event object and can be accessed
  - through G4Event at the end of event, to be used for analyzing an event
  - through G4SDManager during processing an event, to be used for event filtering.

Email:
xuyin@nankai.edu.cn

Class diagram



Email:
xuyin@nankai.edu.cn

Hit Class

```
#include "G4VHit.hh"
class MyHit : public G4VHit
{
  public:
     MyHit(some_arguments);
     virtual ~MyHit();
     virtual void Draw();
     virtual void Print();
     // some set/get methods; eg.
     void         SetEdep (G4double edep)  { fEdep = edep; };
     G4double GetEdep() const              { return fEdep; };
  private:
     // some data members; eg.
     G4double fEdep;  // energy deposit
};
#include "G4THitsCollection.hh"
typedef G4THitsCollection<MyHit> MyHitsCollection;
```

- Hit is a user-defined class derived from G4VHit.
- You can store various types information by implementing your own concrete Hit class.
  - In this example we store the energy deposition of the step
- Hit objects of a concrete hit class must be stored in a dedicated collection which is instantiated from G4THitsCollection template class.

Email:
xuyin@nankai.edu.cn

Defining a Sensitive Detector

- Sensitive detector are defined and assigned to logical volumes in user detector construction class
- Each sensitive detector object must have a unique name. One detector object can be shared by more logical volumes.

MyDetectorConstruction.cc

- One logical volume cannot have more than one detector objects. But, one detector object can generate more than one kinds of hits
  - e.g. a double-sided silicon micro-strip detector can generate hits for each side separately

```
// defined previously
G4LogicalVolume* myLV = ...;
G4VSensitiveDetector* mySD
  = new MySensitiveDetector("/mydet");
G4SDManager* sdManager
  = G4SDManager::GetSDMpointer();
sdManager->AddNewDetector(mySD);
myLV->SetSensitiveDetector(mySD);
```


Email:
xuyin@nankai.edu.cn

Defining a Sensitive Detector

```
#include "G4VSensitiveDetector.hh"
#include "MyHit.hh"
class G4Step;
class G4HCofThisEvent;

class MySensitiveDetector : public G4VSensitiveDetector
 public:
    MySensitiveDetector(G4String name);
    virtual ~MySensitiveDetector();

    virtual void Initialize(G4HCofThisEvent* hce);
    virtual G4bool ProcessHits(
                              G4Step* step,
                              G4TouchableHistory* ROhistory);
    virtual void EndOfEvent(G4HCofThisEvent* hce);
 private:
    MyHitsCollection * fHitsCollection;
    G4int fCollectionID;
};
```

- Sensitive detector is a user-defined class derived from G4VSensitiveDetector

Email:
xuyin@nankai.edu.cn

SD Implementation: Constructor

```
void MySensitiveDetector::MySensitiveDetector(G4String name)
  : G4VSensitiveDetector(name),
    fHitsCollection(0),
    fCollectionID(-1)
{
  collectionName.insert("collectionName");
}
```

- In the constructor, define *the name of the hits collection* which is handled by this sensitive detector
- In case your sensitive detector generates more than one kinds of hits (e.g. anode and cathode hits separately), define all collection names.

Email:
xuyin@nankai.edu.cn

SD Implementation: Initialize()

```
void MySensitiveDetector::Initialize(G4HCofThisEvent* hce)
{
  if ( fCollectionID <0 ) fCollectionID = GetCollectionID(0);
  fHitsCollection
     = new MyHitsCollection (SensitiveDetectorName, collectionName[0]);
  hce->AddHitsCollection(collectionID,hitsCollection);
}
```

- This method is invoked at the beginning of each event.
- Get *the unique ID number for this collection*.
  - GetCollectionID() is available after this sensitive detector object is constructed and registered to G4SDManager.
- *Instantiate hits collection(s)* and attach it/them to G4HCofThisEvent object given in the argument.

Email:
xuyin@nankai.edu.cn

SD Implementation: ProcessHits()

```
G4bool MySensitiveDetector::ProcessHits(G4Step* step,
                                        G4TouchableHistory* /*ROhistory*/)
{
  MyHit* hit = new MyHit();
  // get some properties from G4Step and set them to the hit
  fHitsCollection->insert(hit);
  return true;
}
```

- This method is invoked for every steps in the volume(s) where this sensitive detector is assigned.
- In this method, *generate a hit corresponding to the current step*
- Currently, returning boolean value is not used.

Email:
xuyin@nankai.edu.cn

SD Implementation: EndOfEvent()

```
void MySensitiveDetector::EndOfEvent(G4HCofThisEvent* /*hce*/)
{
  if ( verboseLevel>1 ) {
    G4int nofHits = fHitsCollection->entries();
    G4cout << "\n-------->Hits Collection: in this event they are " << nofHits
           << " hits " << G4endl;
    for ( G4int i=0; i<nofHits; ++i ) (*fHitsCollection)[i]->Print();
  }
}
```

- This method is invoked at the end of processing an event.
  - It is invoked even if the event is aborted
  - It is invoked before UserEventAction::EndOfEventAction

Email:
xuyin@nankai.edu.cn

Tracker and Calorimeter Detector Types

- *A tracker detector* typically generates a hit for every single step of every single (charged) track, it typically contains
  - Position and time
  - Energy deposition of the step
  - Track ID
- *A calorimeter detector* typically generates a hit for every cell, and accumulates energy deposition in each cell for all steps of all tracks, a calorimeter hit typically contains
  - Sum of deposited energy
  - Cell ID

Email:
xuyin@nankai.edu.cn

Magnetic field

- Magnetic field class
  - Uniform field :
    G4UniformMagField class object:
    G4MagneticField* magField
    = new **G4UniformMagField**(G4ThreeVector(0, 0, 1.*Tesla));
  - Non-uniform field :
    User has to define his own concrete class derived from
    G4MagneticField, which implements the method
    GetFieldValue(..)

Email:
xuyin@nankai.edu.cn

Magnetic field Example

B4DetectorConstruction.cc

```
#include "G4UniformMagField.hh"
#include "G4FieldManager.hh"
#include "G4TransportationManager.hh"
#include "G4GenericMessenger.hh"

void B4DetectorConstruction::SetMagField(G4double fieldValue)
{
  // Apply a global uniform magnetic field along X axis
  G4FieldManager* fieldManager
    = G4TransportationManager::GetTransportationManager()->GetFieldManager();

  // Delete the existing magnetic field
  if ( fMagField )  delete fMagField;

  if ( fieldValue != 0. ) {
    // create a new one if not null
    fMagField
      = new G4UniformMagField(G4ThreeVector(fieldValue, 0., 0.));

    fieldManager->SetDetectorField(fMagField);
    fieldManager->CreateChordFinder(fMagField);
  }
  else {
    fMagField = 0;
    fieldManager->SetDetectorField(fMagField);
  }
}
```


Email:
xuyin@nankai.edu.cn

Thanks

南開大學
Nankai University

徐 音
2014-8-12