Version 10.5

Geometry II

Gabriele Cosmo (CERN)

Geant4 Advanced Course

Adapted/extended from original material by M.Asai (SLAC)

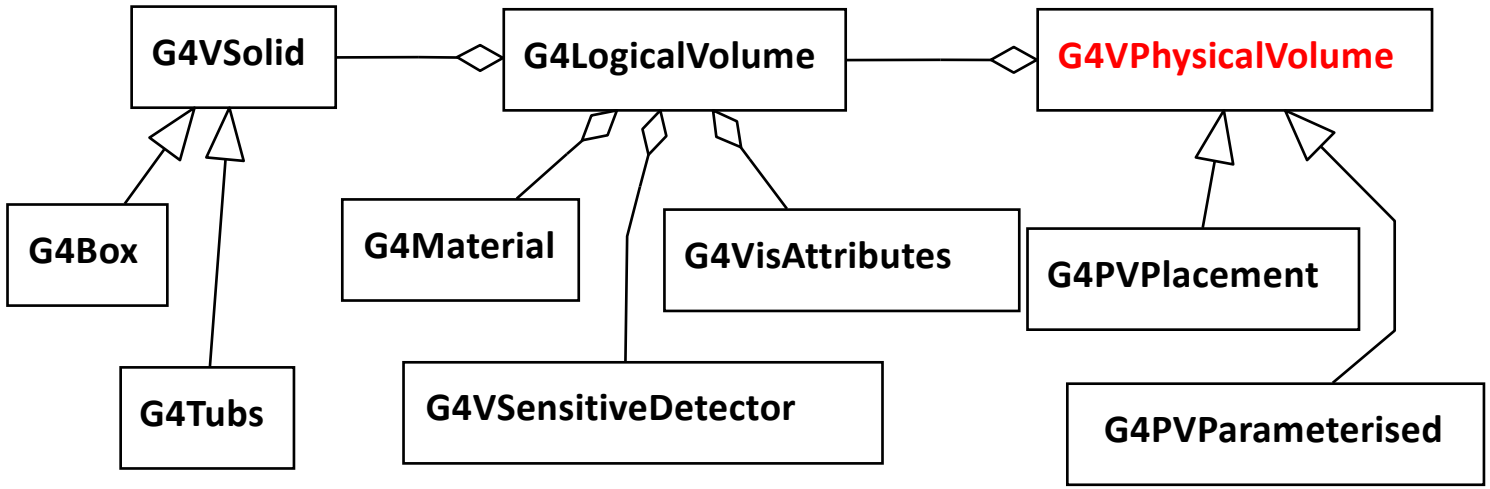
Contents

- Placements, Parameterised and Replicated volumes - Introduction
- Divided volumes
- Nested parameterisations
- Geometrical regions
- Assembly volumes
- Reflected volumes
- Geometry optimisation
- Parallel geometries
- Touchables
- CAD interface

Introduction

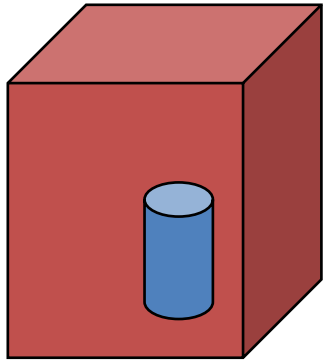
Detector geometry

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

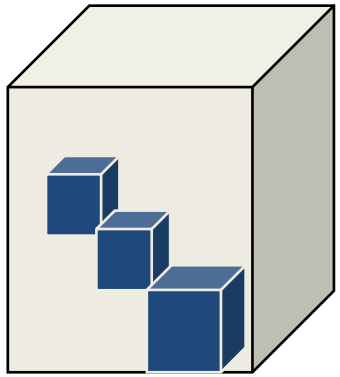


Physical Volumes – placements/repeated

- 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 represents any number of “real” volumes
  - reduces use of memory
  - *Parameterised*
    - Shape, size, material, sensitivity, vis attributes, position and rotation can be parameterised by the 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 Volumes - 1

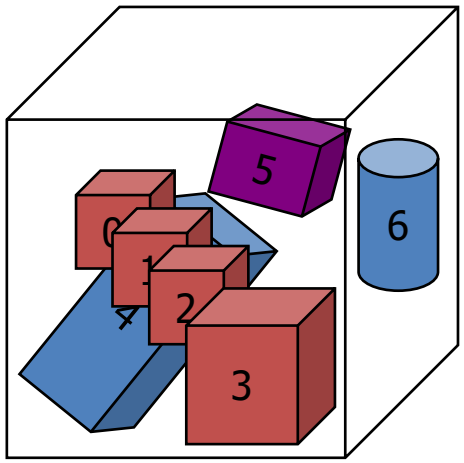
- G4PVPlacement 1 Placement = One Placement Volume
  - A volume instance positioned once in its mother volume
- G4PVParameterised 1 Parameterized = Many Repeated Volumes
  - Parameterised by the copy number
    - Shape, size, material, sensitivity, vis attributes, position and rotation can be parameterised by the copy number
    - You have to implement a concrete class of G4VPVParameterisation
  - Reduction of memory consumption
  - ➢ Daughters to parameterised volumes are allowed only if the volumes are identical in size & shape (so that granddaughters are safely fit inside)
  - ➢ Parameterisations of composed solids like Boolean, Reflected or Displaced solids are not recommended/supported
  - By implementing G4PVNestedParameterisation instead of G4VPVParameterisation, material, sensitivity and vis attributes can be parameterised by the copy numbers of the ancestors

Physical Volumes - 2

- 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

Parameterised Physical Volumes

- User should implement a class derived from G4VPVParameterisation abstract base class and define the following as a function of copy number:
  - where it is positioned (translation, rotation)
  - [optional] the size of the solid (dimensions)
  - [optional] 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
- Limitations/suggestions:
  - Applies to a limited set of solids only
    - Box, Tube, Trd, Simple Trapezoid, Cone, Sphere, Orb, Ellipsoid, Torus, Parallelepiped, Polycone, Polyhedron, Hyperboloid
  - Granddaughter volumes allowed only for special cases
  - Consider parameterised volumes as “leaf” volumes
- Typical use-cases
  - Complex detectors with large repetition of volumes, regular or irregular
  - Medical applications: the material in animal tissue measured as cubes with varying material



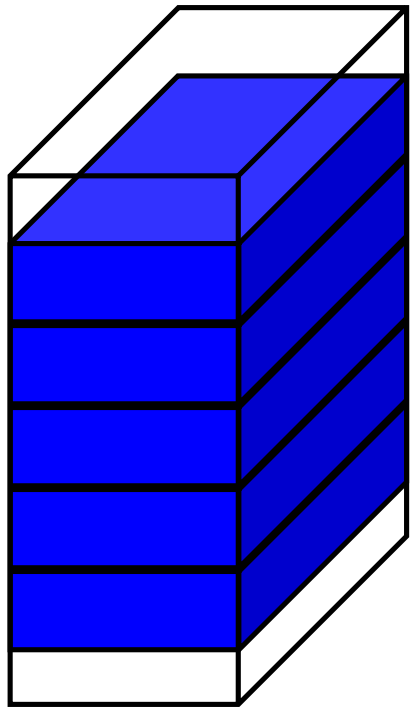
Divided Volumes

G4PVDivision

- A ”division” in Geant4 is a special kind of parameterised volume
  - G4VPVParameterisation is automatically generated according to the parameters given in G4PVDivision
- G4PVDivision is similar to G4PVReplica but
  - It allows gaps in between mother and daughter volumes
- The geometrical shape of all daughter volumes must be the same as for the mother volume.
  - G4VSolid (to be assigned to the daughter logical volume) must be a different object but of the same type
- Replication must be aligned along one axis

➢ For setups with no gaps, should use G4PVReplica instead

  - For identical geometry, navigation in G4PVReplica is faster

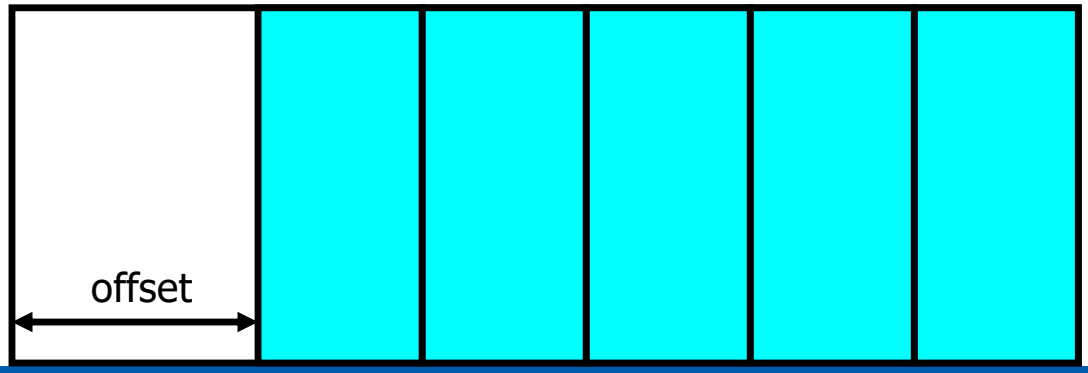


mother volume

G4PVDivision - # of divisions provided

```
G4PVDivision(const G4String& pName,
             G4LogicalVolume* pDaughterLogical,
             G4LogicalVolume* pMotherLogical,
             const EAxis pAxis,
             const G4int nDivisions,
             const G4double offset);
```

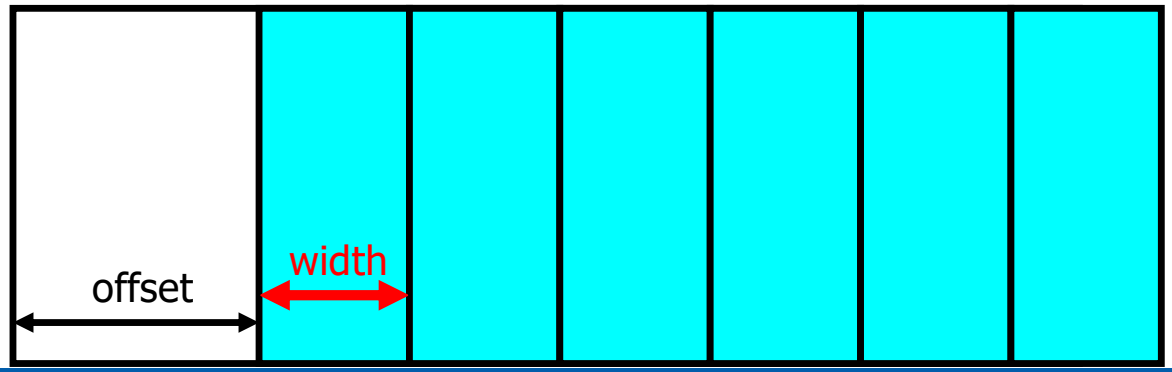
- The size (width) of the daughter volume is calculated as
  ((size of mother) - offset) / nDivisions



G4PVDivision – width of daughter slice provided

```
G4PVDivision(const G4String& pName,
             G4LogicalVolume* pDaughterLogical,
             G4LogicalVolume* pMotherLogical,
             const EAxis pAxis,
             const G4double width,
             const G4double offset);
```

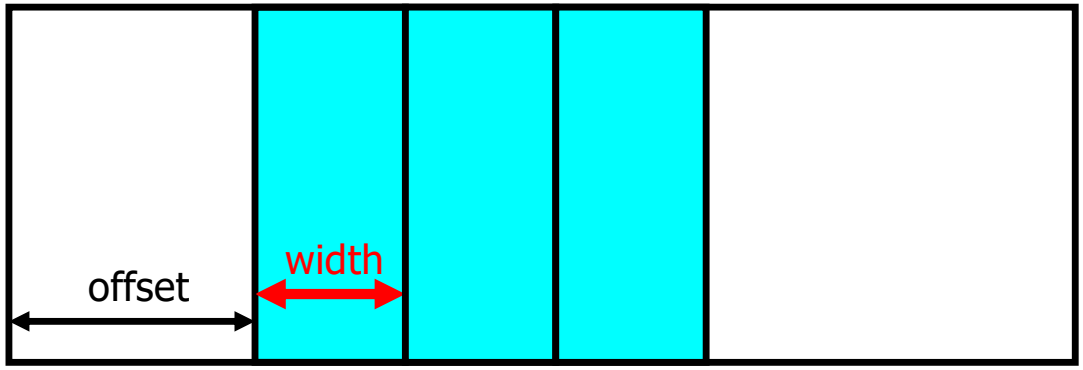
- The number of daughter volumes is calculated as
  `int( ( (size of mother) - offset ) / width )`
  As many daughters as width and offset allow



G4PVDivision – both # of divisions and width of slice provided

```
G4PVDivision(const G4String& pName,
             G4LogicalVolume* pDaughterLogical,
             G4LogicalVolume* pMotherLogical,
             const EAxis pAxis,
             const G4int nDivisions,
             const G4double width,
             const G4double offset);
```

- *nDivisions* daughters of *width* thickness



G4PVDivision – Supported cases

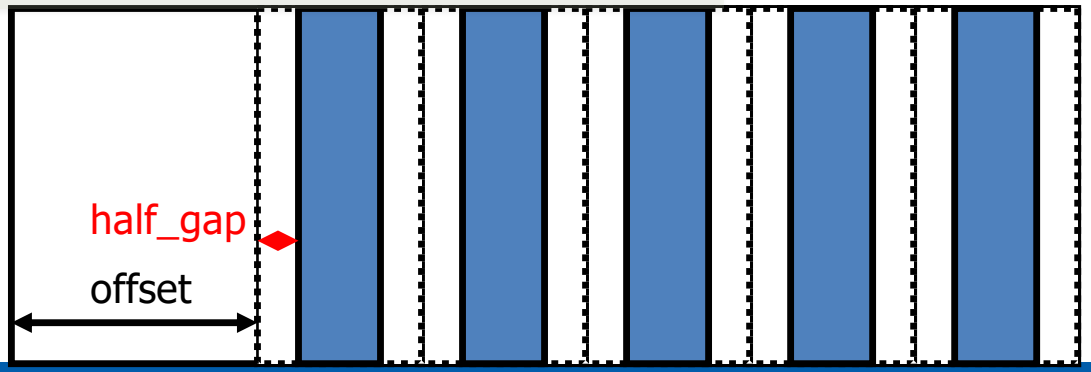
G4PVDivision currently supports following shapes / axes:

- G4Box : *kXAxis, kYAxis, kZAxis*
- G4Tubs : *kRho, kPhi, kZAxis*
- G4Cons : *kRho, kPhi, kZAxis*
- G4Trd : *kXAxis, kYAxis, kZAxis*
- G4Para : *kXAxis, kYAxis, kZAxis*
- G4Polycone : *kRho, kPhi, kZAxis*
  - *kZAxis* - the number of divisions has to be the same as solid sections (i.e. numZPlanes-1), the width will not be taken into account
- G4Polyhedra : *kRho, kPhi, kZAxis*
  - *kPhi* - the number of divisions has to be the same as solid sides (i.e. numSides), the width will not be taken into account
  - *kZAxis* - the number of divisions has to be the same as solid sections (i.e. numZPlanes-1), the width will not be taken into account

G4ReplicatedSlice

- Extension of G4PVDivision allowing gaps in between divided volumes:

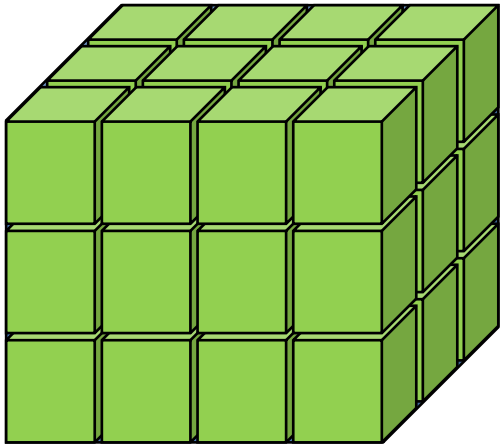
```
G4PVDivision(const G4String& pName, G4LogicalVolume* pDaughterLogical,
             G4LogicalVolume* pMotherLogical,
             const EAxis pAxis, const G4int nDivisions,
             const G4double half_gap, const G4double offset);
G4PVDivision(const G4String& pName,  G4LogicalVolume* pDaughterLogical,
             G4LogicalVolume* pMotherLogical,
             const EAxis pAxis, const G4double width,
             const G4double half_gap,  const G4double offset);
G4PVDivision(const G4String& pName, G4LogicalVolume* pDaughterLogical,
             G4LogicalVolume* pMotherLogical,
             const EAxis pAxis, const G4int nDivisions, const G4double width,
             const G4double half_gap, const G4double offset);
```

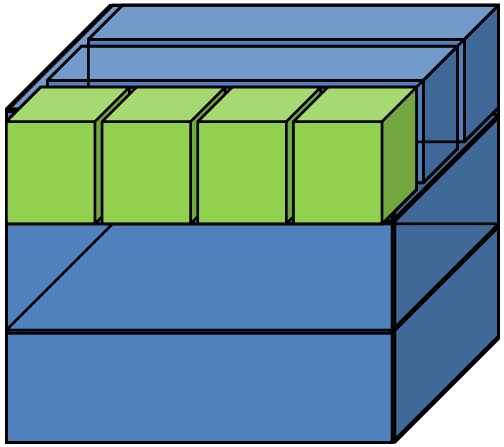


Nested Parameterisations

Nested Parameterisation

- Suppose a geometry with three-dimensional regular repetition of the same shape and size of volumes without gap between volumes. Material of such volumes are changing according to the position
  - E.g. voxels made by CT Scan data (DICOM)
- Instead of a full three-dimensional parameterised volume:
  - Use replicas for the first and second axes sequentially, and then use one-dimensional parameterisation along the third axis

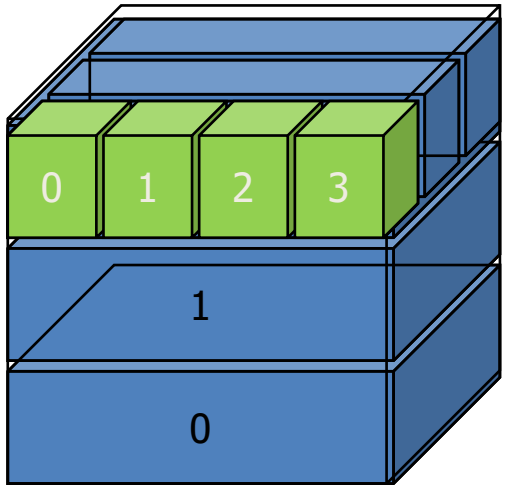




Much less memory required for geometry optimization and much faster navigation for ultra-large number of voxels

Nested Parameterisation

- Given a geometry defined as two sequential replicas and then one-dimensional parameterisation,
  - Material of a single voxel must be parameterised not only by the copy number of the voxel, but also by the copy numbers of the ancestors
  - Material is indexed by three indices



- G4VNestedParameterisation is a special parameterisation class derived from the base class G4VPVParameterisation
  - **`ComputeMaterial()`** method of G4VNestedParameterisation has a touchable object of the parent physical volume, in addition to the copy number of the voxel
    - Index of first axis = theTouchable->GetCopyNumber(1);
    - Index of second axis = theTouchable->GetCopyNumber(0);
    - Index of third axis = copy number

G4VNestedParameterisation

- G4VNestedParameterisation is a class derived from G4VPVParameterisation
- G4VNestedParameterisation class has three pure virtual methods to be implemented by the user in addition to **`ComputeTransformation()`**, which is a mandatory method for all G4VPVParameterisation classes:

```
virtual G4Material* ComputeMaterial(G4VPhysicalVolume *currentVol,
                                    const G4int repNo,
                                    const G4VTouchable *parentTouch=0)=0;
```

  - Returns a material pointer w.r.t. copy numbers of itself and ancestors
  - Must use **`parentTouch=0`** for navigator's sake. Typically, return a default material if **`parentTouch=0`**

```
virtual G4int GetNumberOfMaterials() const=0;
```

  - Returns total number of materials which may appear as the return value of **`ComputeMaterial()`** method

```
virtual G4Material* GetMaterial(G4int idx) const=0;
```

  - Return **`idx`**-th material
  - **`idx`** is not a copy number: **`idx=[0,nMaterial-1]`**

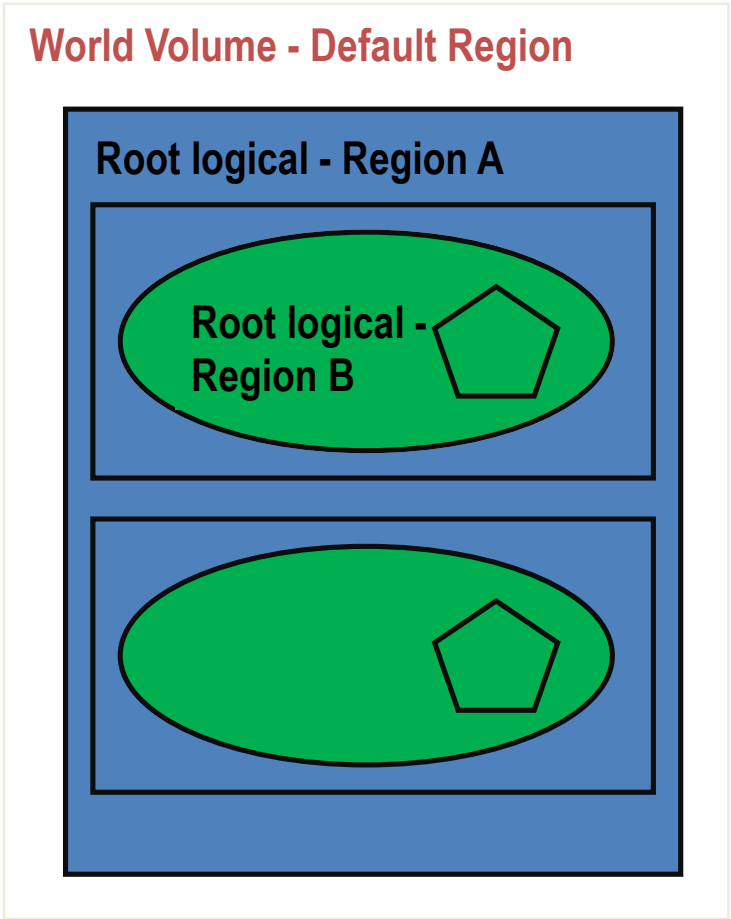
G4VNestedParameterisation - notes

- As sub-type of the G4VPVParameterisation class:
  - G4VNestedParameterisation can be used as an argument of G4PVParameterised
  - All other arguments of G4PVParameterised are unaffected
- Nested parameterisations of a placement volume are not supported
  - All levels used as indices of material must be a repeated volume
  - There cannot be a level of placement volume in between

Regions

Logical Volumes & Regions

- 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 is already set as root logical volume for another region
- Important restriction:
  - No logical volume can be shared by more than one regions, regardless if root volume or not



Geometrical Regions – Information which can be associated with…

- A *G4Region* 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.
    - User limits can be set directly to logical volume as well (if both logical volume and associated region have user limits, those associated to the logical volume take precedence)
  - 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
- NOTE:
  - The world logical volume is recognized as the default region
  - User is not allowed to define a region to the world logical volume

G4Region

- A region is instantiated and defined by:

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

  - A region propagates down the geometrical hierarchy until the bottom or another root logical volume in the hierarchy

- Production thresholds (cuts), for instance, can be assigned to a region by:

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

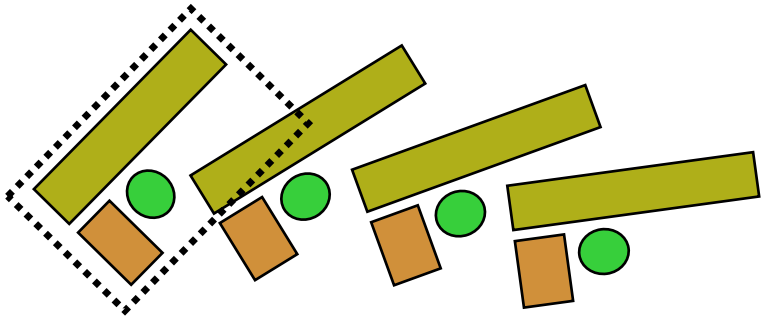
G4Region class

- G4Region class may take the 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);
- NOTES:
  - If any of the above properties are not set for a region
    - properties of the world volume (i.e. the default region) are used
  - Properties of the mother region do not propagate to daughter regions

Assembly Volumes

Grouping volumes

- To represent a regular pattern of positioned volumes, composing a more or less complex structure
  - structures which are hard to describe with simple replicas or parameterised volumes
  - structures which may consist of different shapes
  - too densely positioned to utilize a mother volume
- Assembly volume
  - acts as an *envelope* for its daughter volumes
  - its role is “over” once its logical volume has been placed
  - daughter physical volumes become independent copies in the final structure
- Participating daughter logical volumes are treated as triplets
  - logical volume
  - translation w.r.t. envelop
  - rotation w.r.t. envelop



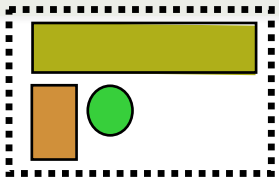
G4AssemblyVolume

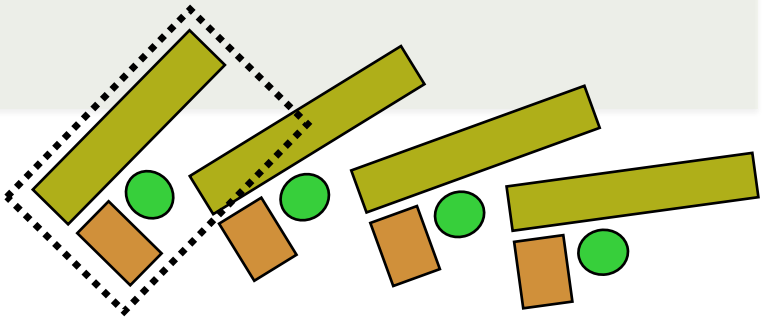
```
G4AssemblyVolume::AddPlacedVolume
                ( G4LogicalVolume* volume,
                  G4ThreeVector& translation,
                  G4RotationMatrix* rotation );
```

- Helper class to combine daughter logical volumes in arbitrary way
  - Imprints of the assembly volume are made inside a mother logical volume through **G4AssemblyVolume::MakeImprint(…)**
  - Each physical volume name is generated automatically
    - Format: **av**_WWW_**impr**_XXX_YYY_ZZZ
      - **WWW** – assembly volume instance number
      - **XXX** – assembly volume imprint number
      - **YYY** – name of the placed logical volume in the assembly
      - **ZZZ** – index of the associated logical volume
  - Generated physical volumes (and related transformations) are automatically managed (creation and destruction)

G4AssemblyVolume : example

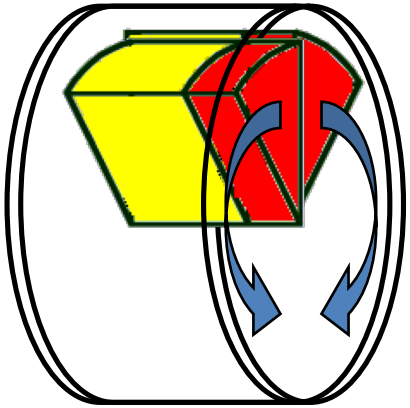
```
G4AssemblyVolume* assembly = new G4AssemblyVolume();
G4RotationMatrix Ra;
G4ThreeVector Ta;
Ta.setX(…); Ta.setY(…); Ta.setZ(…);
assembly->AddPlacedVolume( plateLV, Ta, Ra );
  … // repeat placement for each daughter
for( unsigned int i = 0; i < layers; i++ )
{
  G4RotationMatrix Rm(…);
  G4ThreeVector Tm(…);
  assembly->MakeImprint( worldLV, Tm, Rm );
}
```





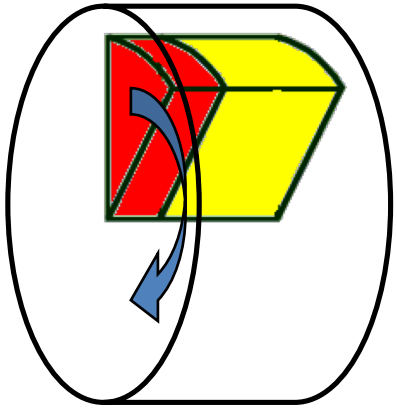
Reflected Volumes

Reflecting solids



- Let's take an example of a pair of mirror symmetric volumes
- Such geometry cannot be made by parallel transformation or 180 degree rotation

A reflection is a "special" transformation which should be applied to the geometrical shape concerned



- G4ReflectedSolid (derived from G4VSolid)
  - Utility class representing a solid shifted from its original reference frame to a new mirror symmetric one
  - The reflection (G4Reflect[X/Y/Z]3D) is applied as a decomposition into rotation and translation
- G4ReflectionFactory
  - Singleton object using G4ReflectedSolid for generating placements of reflected volumes

Reflecting hierarchies of volumes - placements

```
G4PhysicalVolumesPair G4ReflectionFactory::Place
 (const G4Transform3D& transform3D, // the transformation
  const G4String&  name,            // the name
  G4LogicalVolume* LV,              // the logical volume
  G4LogicalVolume* motherLV,        // the mother volume
  G4bool           noBool,          // not used
  G4int            copyNo)          // optional copy number
```

- Used for normal placements:
  i. Performs the transformation decomposition
  ii. Generates a new reflected solid and logical volume
    - Retrieves it from a map if the reflected object is already created
  iii. Transforms any daughter and places them in the given mother
  iv. Returns a pair of physical volumes, the second being a placement in the reflected mother
- **`G4PhysicalVolumesPair`** is an: **`std::map<G4VPhysicalVolume*,G4VPhysicalVolume*>`**

Reflecting hierarchies of volumes - replicas

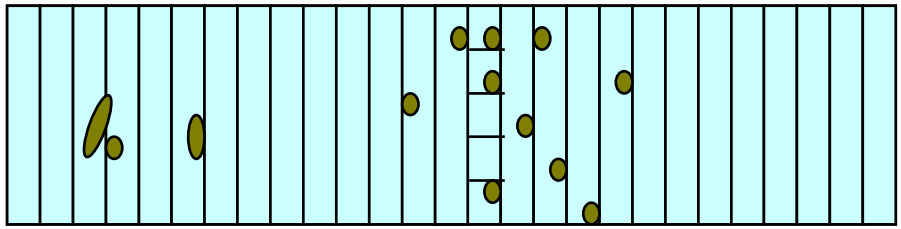
```
G4PhysicalVolumesPair G4ReflectionFactory::Replicate
  (const G4String&  name,       // the actual name
   G4LogicalVolume* LV,         // the logical volume
   G4LogicalVolume* motherLV,   // the mother volume
   Eaxis            axis        // axis of replication
   G4int            replicaNo   // number of replicas
   G4int            width,      // width of single replica
   G4int           offset=0)    // optional mother offset
```

- Creates replicas in the given mother volume
- Returns a pair of physical volumes
  - the second being a replica in the reflected mother

Geometry Optimisation

Smart Voxelisation

- In past versions of Geant, the user had to carefully implement his/her geometry to maximize the performance of the geometrical navigation
- In Geant4, the user's geometry is automatically optimised to be most suitable for the navigation, thanks to a 3D "Voxelization" technique
  - For each mother volume, one-dimensional virtual division is performed
  - Subdivisions (slices) containing same volumes are gathered into one
  - Additional division again using second and/or third Cartesian axes, if needed
- *"Smart voxels"* are computed at initialisation time
  - When the detector geometry is *closed*
  - Does not require large memory or computing resources
  - At tracking time, searching is done in a hierarchy of virtual divisions



Detector Description Tuning

- Some geometry topologies may require ‘special’ tuning for ideal and efficient optimisation
  - For example: a dense nucleus of volumes included in a very large mother volume
- Granularity of voxelisation can be explicitly set
  - Methods **`Set/GetSmartless()`** from **`G4LogicalVolume`**
- Critical regions for optimisation can be detected
  - Helper class **`G4SmartVoxelStat`** for monitoring time spent in detector geometry optimisation
    - Automatically activated if **`/run/verbose`** greater than **`1`**

```
Percent     Memory     Heads     Nodes    Pointers    Total CPU    Volume
-------     ------     -----     -----    --------    ---------    -----------
  91.70         1k         1        50          50         0.00    Calorimeter
   8.30         0k         1         3           4         0.00    Layer
```

Visualising Voxel Structure

- The computed voxel structure can be visualised with the final detector geometry
  - Helper class **`G4DrawVoxels`**
  - Visualize voxels given a logical volume

```
G4DrawVoxels::DrawVoxels(const G4LogicalVolume*)
```

  - Allows setting of visualisation attributes for voxels

```
G4DrawVoxels::SetVoxelsVisAttributes(…)
```

  - Useful for debugging purposes

Parallel Geometries

Parallel navigation

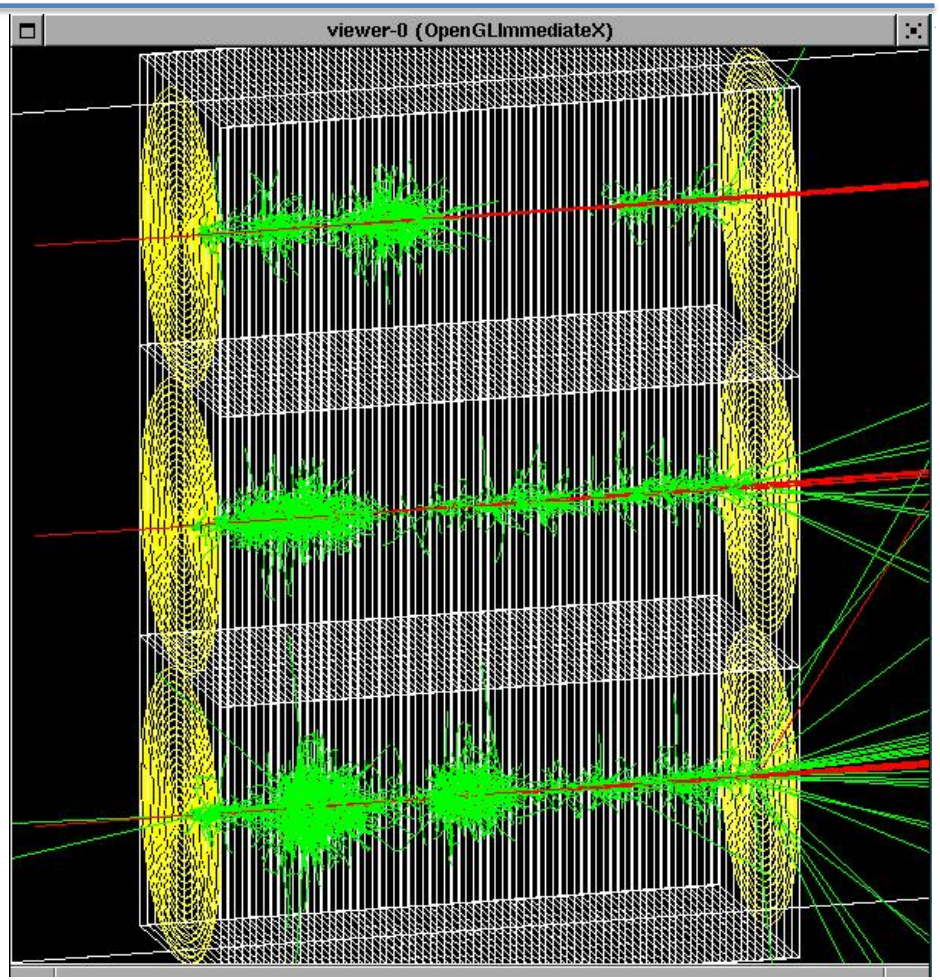
- Occasionally, it may not be straightforward to define attributes, like sensitivity, importance or envelope to be assigned to volumes in the mass geometry
  - Typically applicable to geometries imported from CAD, GDML, DICOM, etc.
- Parallel navigation functionality allows to define more than one overlapping geometry setups (worlds) simultaneously
  - The **`G4Transportation`** process can act on all worlds simultaneously
  - A step is limited not only by the boundary of the original mass geometry but also by the boundaries of each parallel geometry
  - Materials, production thresholds and EM field are used only from the mass geometry
  - In a parallel world, the user can define volumes in arbitrary manner with sensitivity, regions with shower parameterization, and/or importance field for biasing
  - Volumes in different worlds may overlap

G4VUserParallelWorld & G4ParallelWorldProcess

- G4VUserParallelWorld is the base class where to implement a parallel geometry
  - The world physical volume of the parallel world is provided by the **`G4RunManager`** as a clone of the mass geometry
  - Each parallel world has its dedicated **`G4Navigator`** object, that is automatically assigned at construction
  - All parallel geometries defined by the user must be registered in the UserDetectorConstruction
- Though all worlds will be comprehensively taken care by **`G4Transportation`** for their navigations, each parallel world must have its own process to achieve its purpose
  - Example: in case the user defines a sensitive detector to a parallel world, a process dedicated to this world is responsible to invoke this detector. **`G4SteppingManager`** sees only the detectors in the mass geometry. The user has to have G4ParallelWorldProcess in his physics list

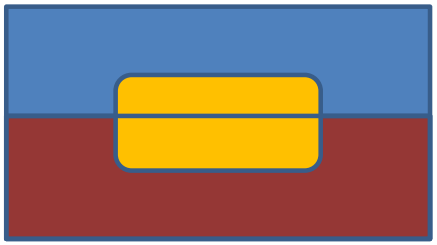
Example - extended/runAndEvent/RE06

- Mass geometry
  - sandwich of rectangular absorbers and scintillators
- Parallel scoring geometry
  - Cylindrical layers

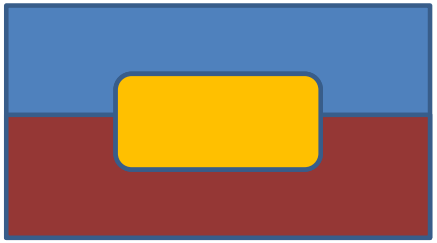


Layered mass geometries in parallel world

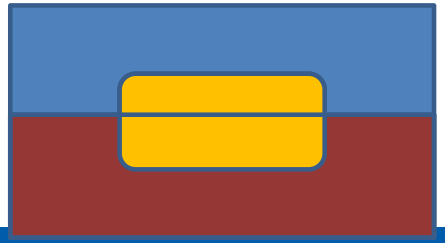
- Suppose to implement a wooden brick floating on the water.



- Dig a hole in water…

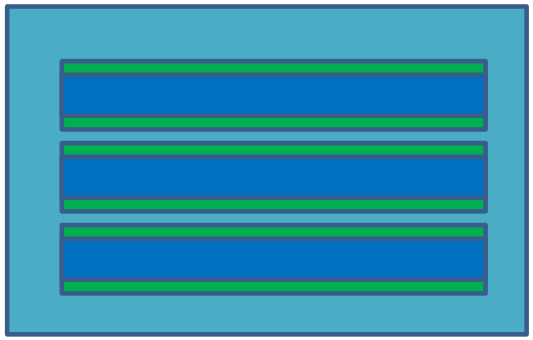


- Or, chop a brick into two and place them separately…

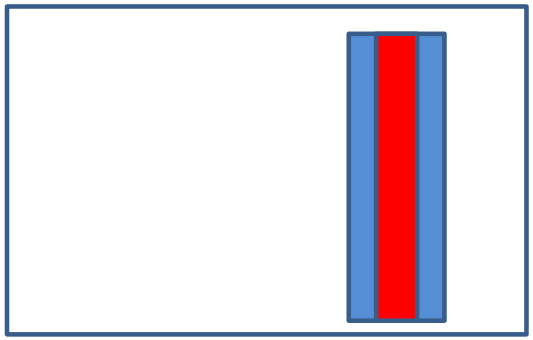


Layered mass geometries in parallel worlds

- A parallel geometry may be stacked on top of the mass geometry or other parallel world geometries, allowing a user to define more than one worlds with materials (and region/cuts)
  - Track will see the material of top-layer
  - Alternative way of implementing a complicated geometry
    - Rapid prototyping
    - Safer, more flexible and powerful extension of the concept of "many" in Geant-3



Mass world

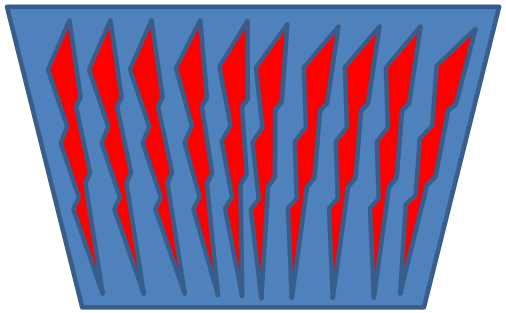


Parallel world

Layered mass geometries in parallel worlds – HEP use case

- A parallel world may be associated only to some limited types of particles
  - May define geometries of different levels of detail for different particle types
  - Example for sampling calorimeter: the mass world defines only the crude geometry with averaged material, while a parallel world with all the detailed geometry. Real materials in detailed parallel world geometry are associated with all particle types except e+, e- and gamma
    - e+, e- and gamma do not see volume boundaries defined in the parallel world, i.e. their steps won't be limited
  - Shower parameterisations may have their own geometry

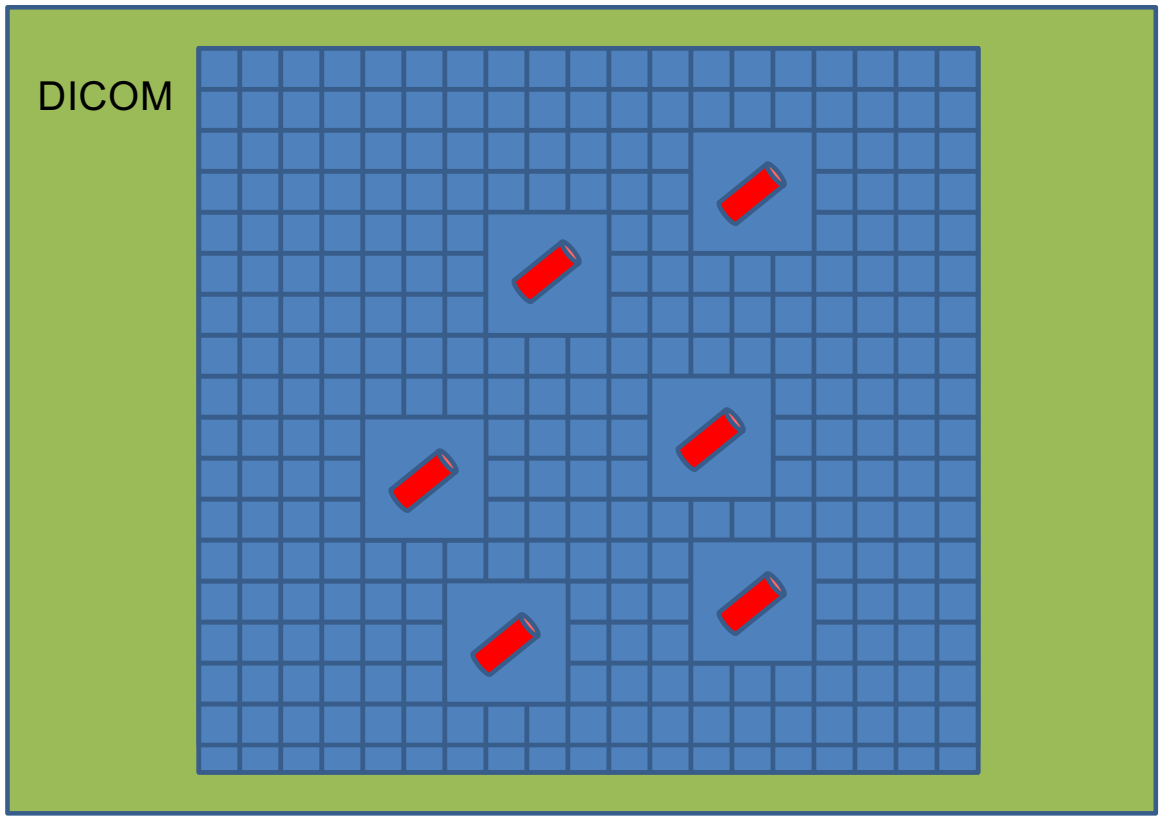
Geometry seen by e+, e-, γ



Geometry seen by other particles

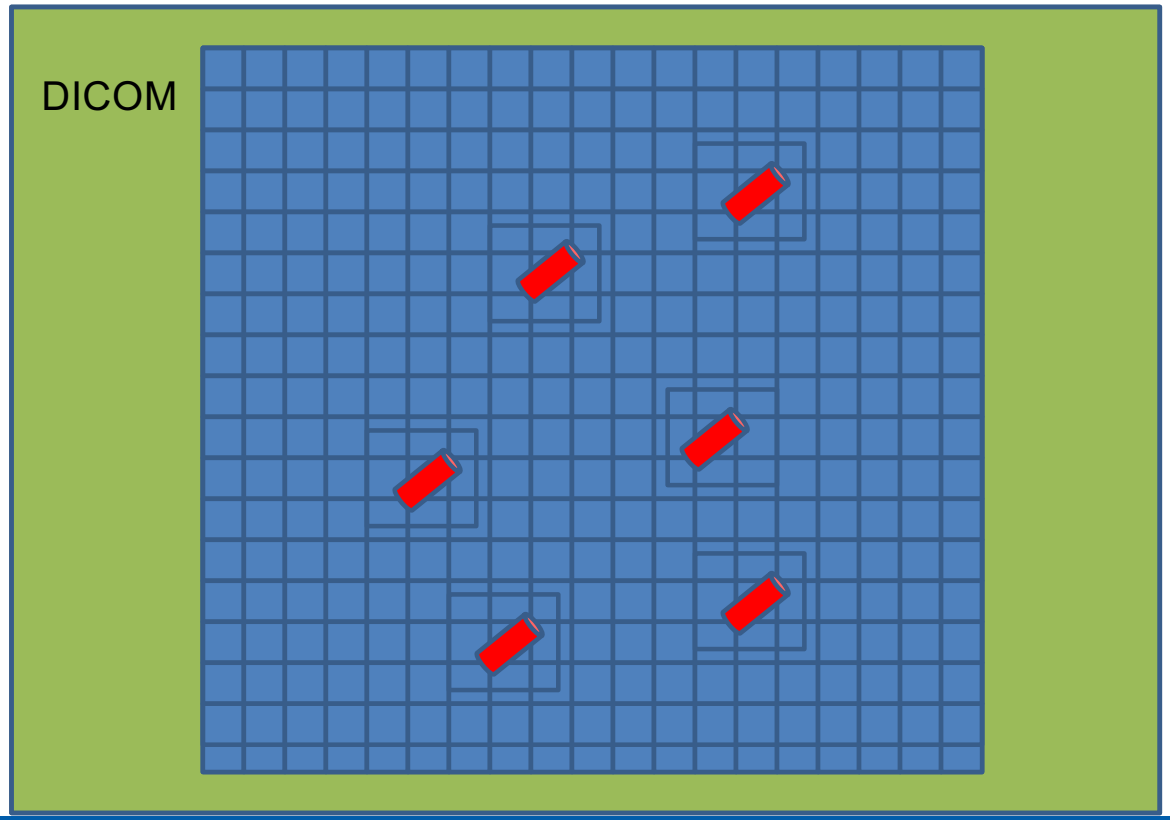
A medical use case

- Brachytherapy treatment for prostate cancer



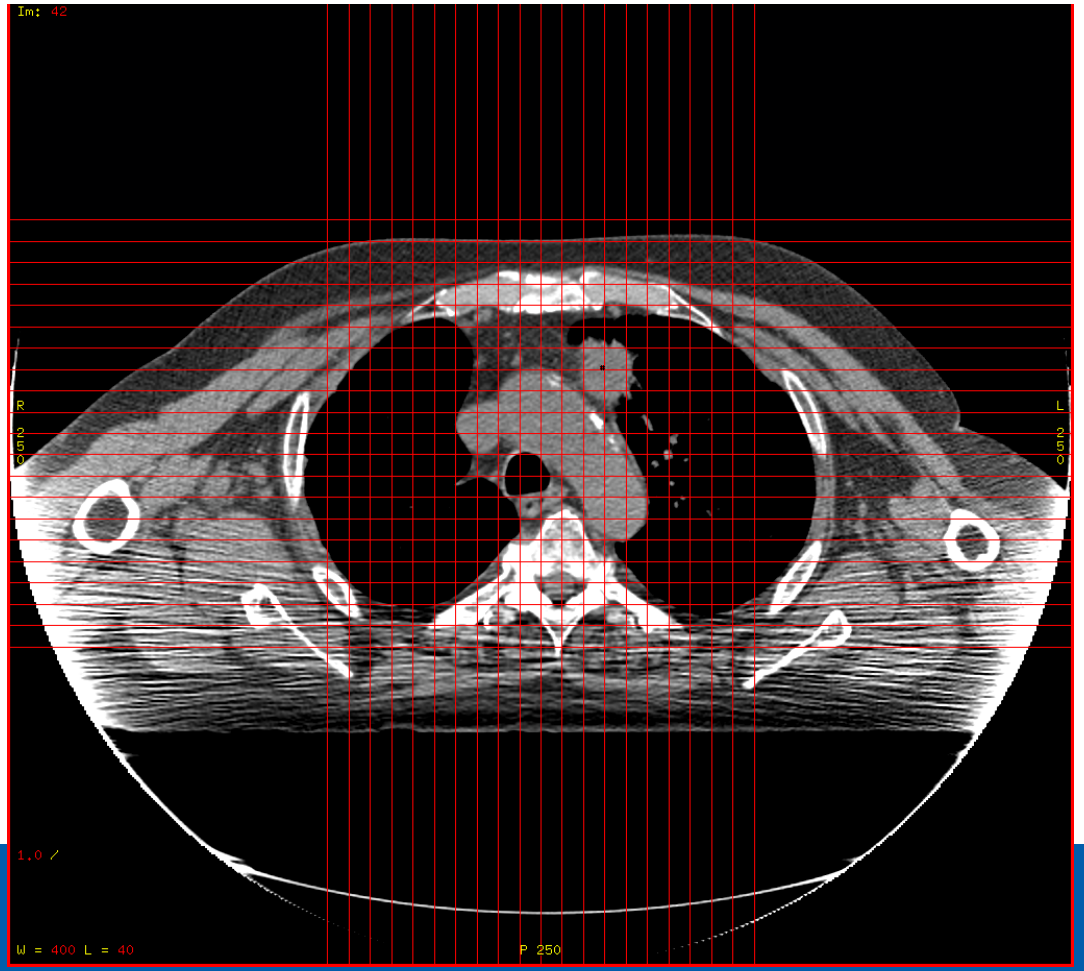
A medical use case

- Alternatively, seeds could be implemented in an empty parallel world
  - Seeds in the parallel world would be encapsulated in empty boxes for faster navigation



Another important use case in medicine

- DICOM data contain void air region outside of the patient, while the treatment head should be placed as close as patient's body

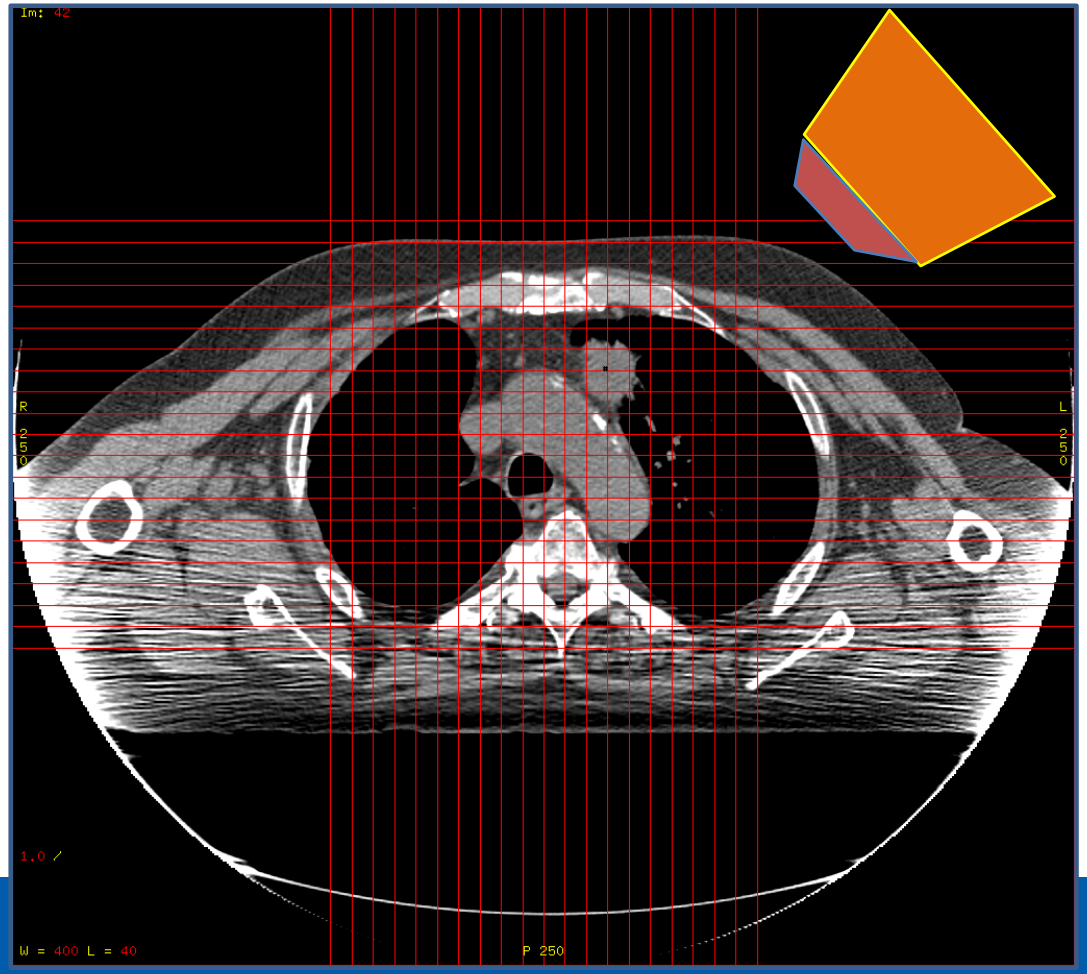


Another important use case in medicine

- Implement the treatment head in a parallel world:

On more use case in medicine

- And overlay:



Defining a parallel world with layered mass geometry

main() (RE04.cc)

```
G4String paraWorldName = "ParallelWorld";
G4VUserDetectorConstruction* realWorld = new RE04DetectorConstruction;
G4VUserParallelWorldConstruction* parallelWorld
  = new RE04ParallelWorldConstruction(paraWorldName);
realWorld->RegisterParallelWorld(parallelWorld);
runManager->SetUserInitialization(realWorld);

G4VModularPhysicsList* physicsList = new FTFP_BERT;
physicsList->RegisterPhysics
      (new G4ParallelWorldPhysics(paraWorldName,true));
runManager->SetUserInitialization(physicsList);
```

Switch of layered mass geometry

– The name defined in the G4VUserParallelWorld constructor is used as the physical volume name of the parallel world, and must be given to **`G4ParallelWorldPhysics`**

Defining a parallel world

```
void RE04ParallelWorldConstruction::Construct()
{
  //
  // World
  G4VPhysicalVolume* ghostWorld = GetWorld();
  G4LogicalVolume* worldLogical = ghostWorld->GetLogicalVolume();
  //
  // material defined in the mass world
  G4Material* water = G4Material::GetMaterial("G4_WATER");
  //
  // parallel world placement box
  G4VSolid* paraBox = new G4Box("paraBox",5.0*cm,30.0*cm,5.0*cm);
  G4LogicalVolume* paraBoxLogical
      = new G4LogicalVolume(paraBox, water, "paraBox");
  new G4PVPlacement(0,G4ThreeVector(-25.0*cm,0.,0.),paraBoxLogical,
                    "paraBox",worldLogical,false,0);
```

- The world physical volume of the parallel world is provided as a clone of the world volume of the mass geometry automatically
- One can fill volumes regardless of the volumes in the mass geometry
- Logical volumes in a parallel world may not have a material assigned

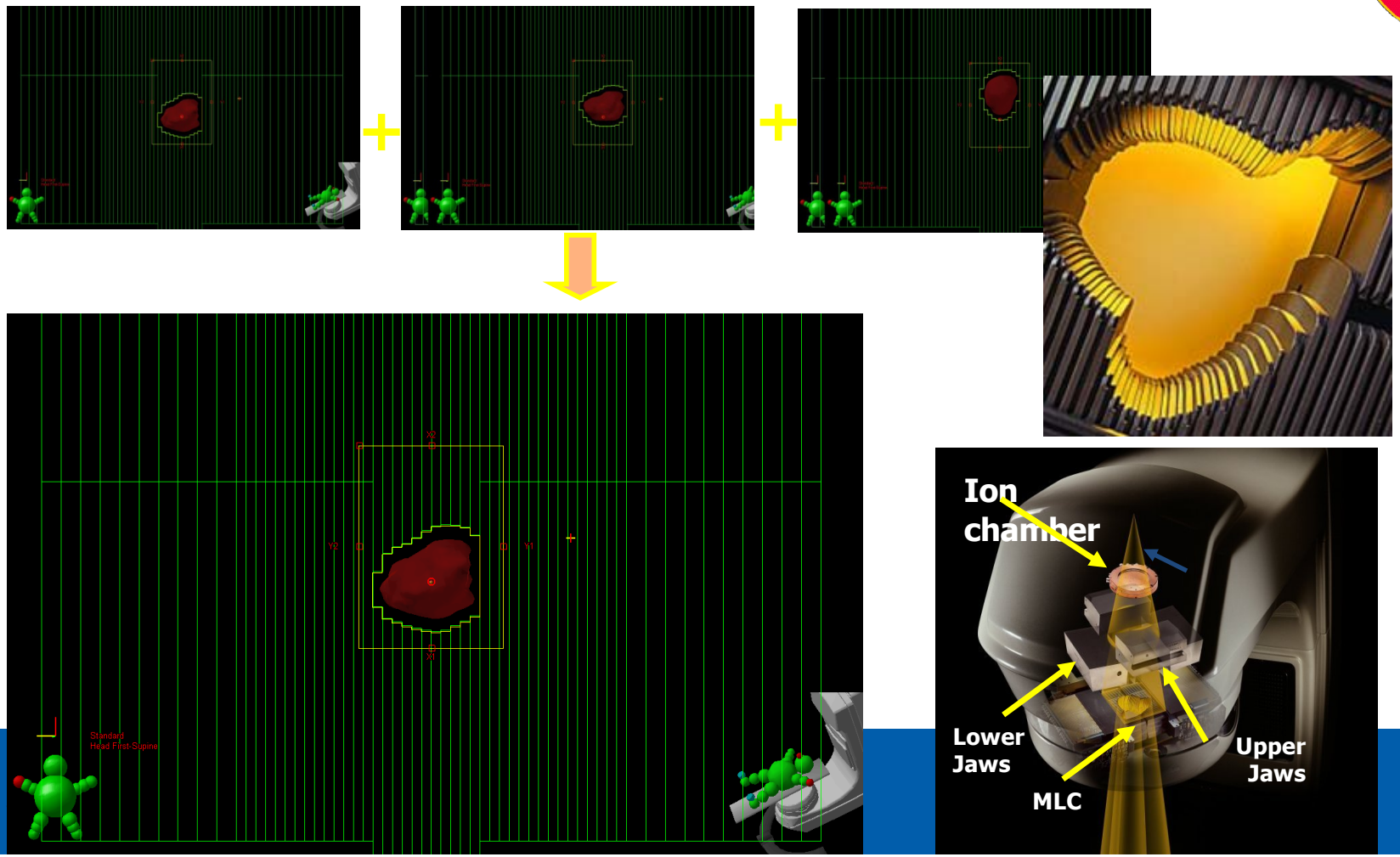
Moving Objects

Moving objects

- In some applications, it is essential to simulate the movement of some volumes
  - E.g. particle therapy simulation
- Geant4 can deal with moving volumes
  - In case speed of the moving volume is slow enough compared to speed of elementary particles, so that you can assume the position of the moving volume is still within one event
- Two tips to simulate moving objects:
  1. Use a parameterised volume to represent the moving volume
  2. Do not optimise (voxelise) the mother volume of the moving volume(s)

4D RT Treatment Plan

Source: Lei Xing, Stanford University



Moving objects – Use of parameterised volumes

- Use parameterised volume to represent the moving volume
  - Event number used as a time stamp
  - Calculate position/rotation of the volume as a function of the event number

```
void MyMovingVolumeParameterisation::ComputeTransformation
     (const G4int copyNo, G4VPhysicalVolume *physVol) const
{
  static G4RotationMatrix rMat;
  G4int eID = 0;
  const G4Event* evt =
    G4RunManager::GetRunManager()->GetCurrentEvent();
  if(evt) eID = evt->GetEventID();
  G4double t = 0.1*s*eID;
  G4double r = rotSpeed*t;
  G4double z = velocity*t+orig;
  while(z>0.*m) {z-=8.*m;}
  rMat.set(CLHEP::HepRotationX(-r));
  physVol->SetTranslation(G4ThreeVector(0.,0.,z));
  physVol->SetRotation(&rMat0);
}
```

Null pointer must be protected. This method is also invoked while geometry is being closed at the beginning of run, i.e. event loop has not yet began

Here, event number is converted to time. (0.1 sec/event)

You are responsible not to make the moving volume get out of (protrude from) the mother volume

Position and rotation are set as the function of event number

Moving objects – Avoid voxelisation

- Do not optimize (voxelise) the mother volume of the moving volume(s)
  - If moving volume gets out of the original optimised voxel, navigation gets invalidated

```
motherLogical -> SetSmartless( number_of_daughters );
```

  - With this method invocation, the one-and-only optimised voxel has all daughter volumes
  - For the best performance, use a hierarchal geometry so that each mother volume has the least number of daughters

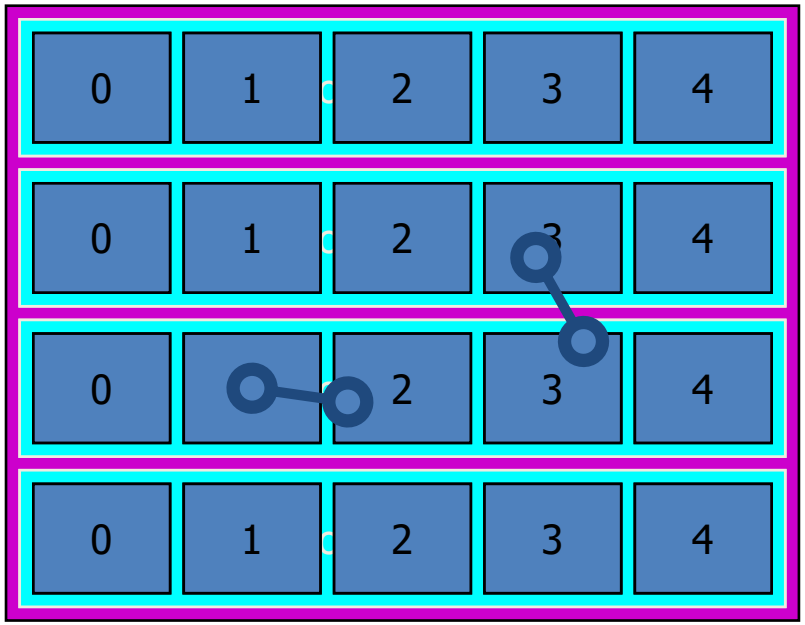
Touchables

Step Point & Touchable

- A track in Geant4 is composed of steps; **`G4Step`** has two **`G4StepPoint`** objects as its starting and ending points. All the geometrical information of the particular step should be taken from the "PreStepPoint"
  - The geometrical information associated with G4Track is identical to the "PostStepPoint"
- Each **`G4StepPoint`** object provides:
  - Position in world coordinate system
  - Global and local time
  - Material
  - **`G4TouchableHistory`** for geometrical information
- The **`G4TouchableHistory`** object is a vector of information for each geometrical hierarchy, including:
  - copy number
  - transformation / rotation to its mother
- *Handles* (or *smart-pointers*) to touchables are intrinsically used in Geant4
  - Touchables are reference counted objects

Copy numbers

- Suppose a calorimeter is made of 4x5 cells
  - and it is implemented by two levels of replica
- In reality, there is only one physical volume object for each level. Its position is parameterised by its copy number
- To get the copy number of each level, suppose what happens if a step belongs to two cells



- The geometrical information in **`G4Track`** is identical to "PostStepPoint"
  - One cannot get the correct copy number for the "PreStepPoint" by directly accessing to the physical volume
- Use touchable to get the proper copy number, transform matrix, etc.

How to get information from a touchable?

G4TouchableHistory has information on the geometrical hierarchy of the point

```
G4Step* aStep;
G4StepPoint* preStepPoint = aStep->GetPreStepPoint();
G4TouchableHistory* theTouchable =
    (G4TouchableHistory*)(preStepPoint->GetTouchable());
G4int copyNo = theTouchable->GetVolume()->GetCopyNo();
G4int motherCopyNo
           = theTouchable->GetVolume(1)->GetCopyNo();
G4int grandMotherCopyNo
           = theTouchable->GetVolume(2)->GetCopyNo();
G4ThreeVector worldPos = preStepPoint->GetPosition();
G4ThreeVector localPos = theTouchable->GetHistory()
    ->GetTopTransform().TransformPoint(worldPos);
```

CAD Interface

Importing CAD geometries

- Users with 3D engineering drawings may want to incorporate these into simulation as directly as possible
- Difficulties include:
  - Proprietary, undocumented or changing CAD formats
  - Usually no connection between geometrical parts and materials
  - Mismatch in the level of detail required for engineering and that required for transporting particles for simulation in the geometry
- CAD is never as easy as you might think
  - If the geometry is complex enough to require CAD in the first place
- Most CAD programs do support the STEP format
  - Not a complete solution, in particular as it does not contain material information
  - There are developments under way to define extensions for treatment of additional information, although none yet widely adopted

Solutions for CAD models conversion – One example: CADMesh

CADMesh is a direct CAD model import interface for Geant4 optionally based on VCGLIB, and ASSIM:

https://code.google.com/p/cadmesh/

http://arxiv.org/pdf/1105.0963.pdf

It supports the import of triangular facet surface meshes defined in formats such as STL and PLY.
A **`G4TessellatedSolid`** is returned and can be included in a standard user detector constructor

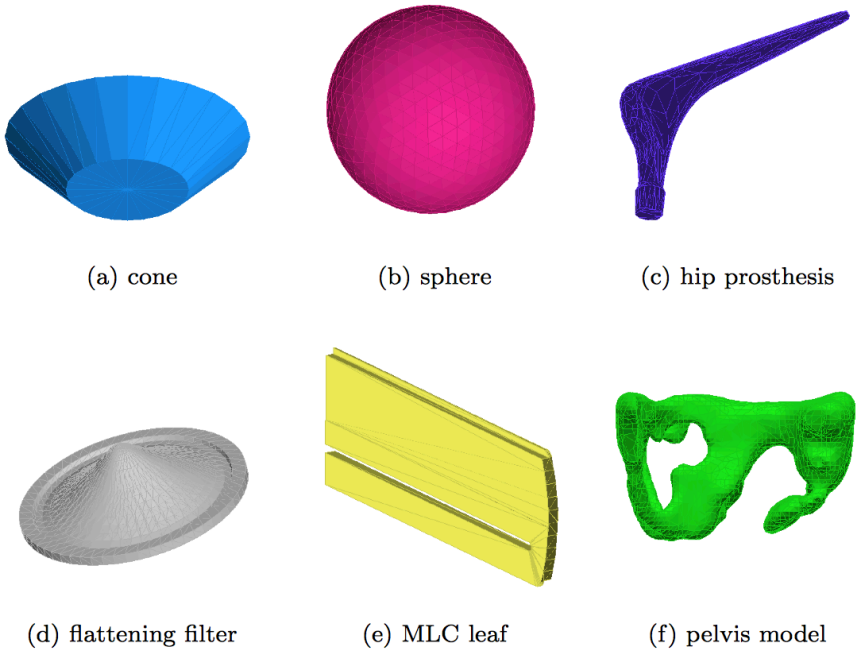


Fig. 3 Six test geometries loaded directly into GEANT4 using the proposed CAD interface and visualised using the GEANT4 OpenGL viewer.

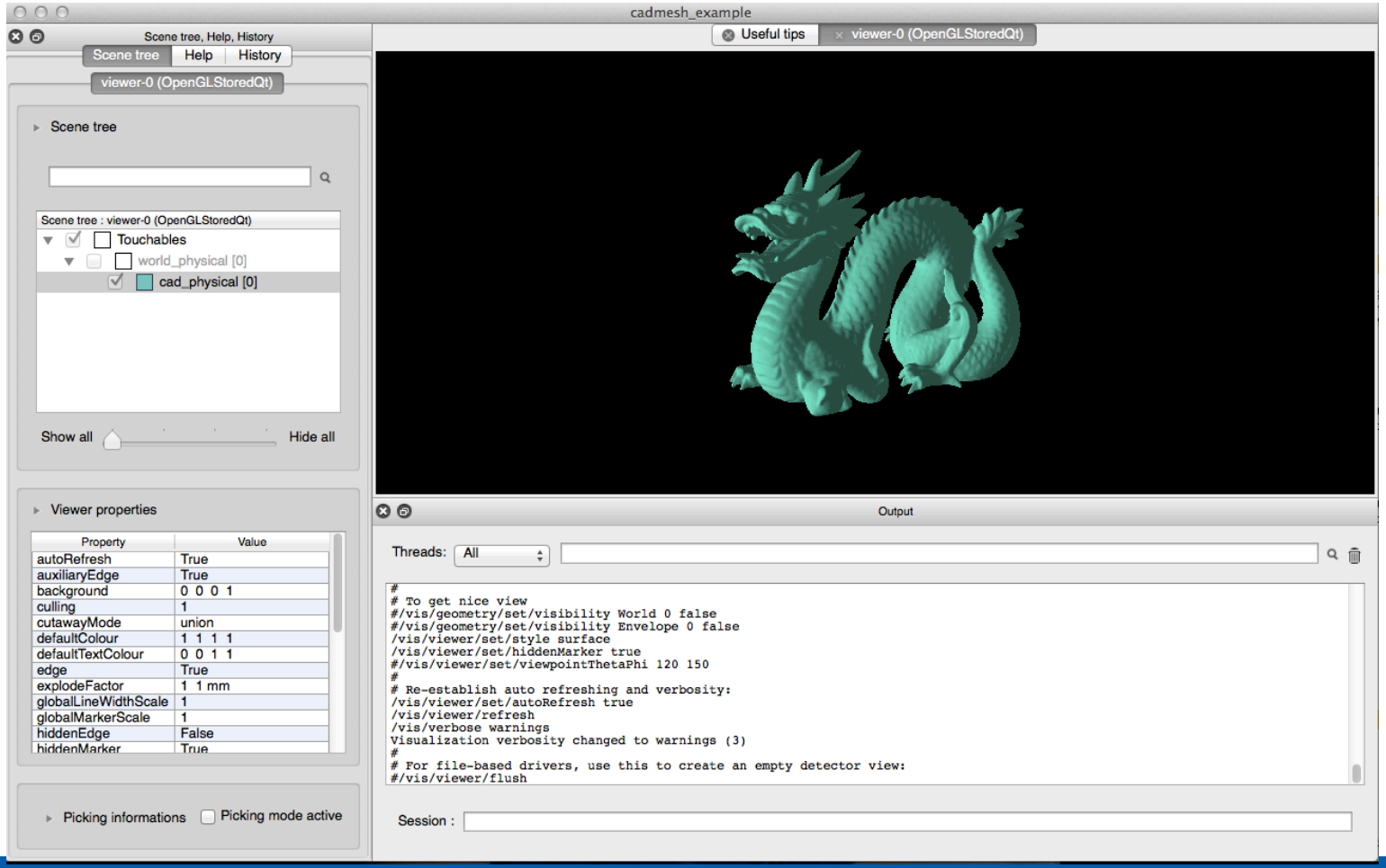
Example: loading a CADMesh solid

- CADMesh provides a **`G4TessellatedSolid`** object

```
#include “CADMesh.hh”
G4VPhysicalVolume* MyDetectorConstruction::Construct()
{
  …
  CADMesh cadMesh;
  G4VSolid* aSolid = cadMesh.LoadMesh( file_name, file_type );
  G4LogicalVolume* aLV = new G4LogicalVolume( aSolid, material, “name” );
  G4VPhysicalVolume* aPV = new G4PVPlacement( 0,
      G4ThreeVector(x,y,z), aLV, “name”, motheLV, 0, 0 );
  …
```

- See also developed STEP to GDML conversion procedures:
  - http://csrsrv1.fynu.ucl.ac.be/csr_web/geant/step-gdml.php

CADMesh


cadmesh_example
Scene tree, Help, History
Scene tree
Help
History
viewer-0 (OpenGLStoredQt)
Scene tree
Scene tree : viewer-0 (OpenGLStoredQt)
Touchables
world_physical [0]
cad_physical [0]
Show all
Hide all
Viewer properties
Property
Value
autoRefresh
True
auxiliaryEdge
True
background
0 0 0 1
culling
1
cutawayMode
union
defaultColour
1 1 1 1
defaultTextColour
0 0 1 1
edge
True
explodeFactor
1 1 mm
globalLineWidthScale
1
globalMarkerScale
1
hiddenEdge
False
hiddenMarker
True
Picking informations
Picking mode active
Useful tips
viewer-0 (OpenGLStoredQt)
Output
Threads:
All
#
To get nice view
#/vis/geometry/set/visibility World 0 false
#/vis/geometry/set/visibility Envelope 0 false
/vis/viewer/set/style surface
/vis/viewer/set/hiddenMarker true
#/vis/viewer/set/viewpointThetaPhi 120 150
#
Re-establish auto refreshing and verbosity:
/vis/viewer/set/autoRefresh true
/vis/viewer/refresh
/vis/verbose warnings
Visualization verbosity changed to warnings (3)
#
For file-based drivers, use this to create an empty detector view:
#/vis/viewer/flush
Session :

Importing CAD geometries: more solutions …

1. InStep, supporting import/export of different format, including STL, STEP and GDML
   https://www.solveering.com/InStep/instep.aspx
2. SALOME, can import STEP BREP/IGES/STEP/ACIS, mesh it, export to STL than use STL2GDML to export to GDML
   http://www.salome-platform.org
3. ESABASE2, space environment analysis CAD, basic modules are free for academic non-commercial use.
   Exports to GDML shapes or complete geometries. Imports STEP
   https://esabase2.net
4. Blender GDML exporter, GDML plugin for the Blender tool
   http://projects.blender.org/tracker/index.php?func=detail&aid=30578&group_id=153&atid=467
5. FASTRAD, 3D tool for radiation shielding analysis; exports meshes to GDML
   http://www.fastrad.net
6. STEP Solutions, commercial, exports meshes to then import as GDML
   http://www.steptools.com/products/stdev/
7. Cogenda TCAD, for case of 3D meshes. Module Gds2Mesh exports to GDML
   http://www.cogenda.com/article/products#VTCAD
8. SW2GDML convert SolidWorks descriptions (using its API) to real primitives in GDML, including materials
   https://github.com/cvuosalo/SW2GDMLconverter
9. CadMC, tool to convert FreeCAD geometries to Geant4 (tessellated and CSG shapes)
   http://polar.psi.ch/cadmc/
10. McCAD tool 'integrated approach' by KIT group, using half-space solids extensions to Geant4
    http://indico.cern.ch/event/400576/contributions/1841503/attachments/802327/1099598/CERN_Visit_YQIU_V0.2.pdf