Some tips for geometries of medical applications

Makoto Asai (SLAC)
Most efficient way of simulating DICOM-like 3D voxels with material parameterization
Options you can take - 1

- There is no silver bullet. You can try some/all of these options combined.
- Huge number of cells
  - If 3D parameterized volume with material parameterization is used,
    - Compact memory size but slow in case of 1D optimization
    - Fast but huge memory size in case of 3D optimization
  - Use replica for the first and second axes slices and 1-dimensional parameterization for the third axis. Use G4NestedParameterisation to parameterize the material.
- Material map
  - Though number of materials appear is quite limited, each cell must at least have a pointer to a material. I.e. you have to have a huge material map which has entries of the number of cells.
  - Split your whole voxel geometry into reasonable number of regions, and assign a dedicated stack to each region. For example 5*5*5 = 125 regions.
  - Load material map (from your file on the disk) only for one region. If a track reaches to the boundary of the region you are currently simulating, suspend the track.
  - Simulate all the tracks in one region. Once a region becomes empty, load material map for another region and simulate all tracks in that region.
  - Note that some tracks may come back to a region you have already simulated.
Options you can take - 2

- Event biasing
  - In particular, geometrical importance biasing, and secondary particle splitting must be good options to take.
  - You must validate results of your biasing options with full simulation.

- Shower parameterization
  - In stead of having a full EM shower, you may want to consider the shower parameterization in particular for the core part of the shower.

- Dedicated navigator
  - Given the geometry is perfectly regular, you may want to consider implementing a dedicated navigator that is absolutely simple-minded to handle just regular pattern of boxes of same size, thus quite fast.
  - Your navigator should be set to G4Transportation process.

- Parallelization
  - Allocate good number of CPUs...
  - Details on Takashi’s talk.
Nested parameterization

- Suppose your geometry has three-dimensional regular reputation of same shape and size of volumes without gap between volumes. And material of such volumes are changing according to the position.
  - E.g. voxels made by CT Scan data (DICOM)
- Instead of direct three-dimensional parameterized volume, use replicas for the first and second axes sequentially, and then use one-dimensional parameterization along the third axis.
- It requires much less memory for geometry optimization and gives much faster navigation for ultra-large number of voxels.
Nested parameterization

- Given geometry is defined as two sequential replicas and then one-dimensional parameterization,
  - Material of a voxel must be parameterized not only by the copy number of the voxel, but also by the copy numbers of ancestors.
  - Material is indexed by three indices.
- G4VNestedParameterisation is a special parameterization class derived from G4VPVParameterisation base class.
  - 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 derived from G4VPVParameterization.
- G4VNestedParameterisation class has three pure virtual methods you have to implement,
  - in addition to ComputeTransformation() method, which is mandatory for all G4VPVParameterization classes.

virtual G4Material* ComputeMaterial(G4VPhysicalVolume *currentVol, const G4int repNo, const G4VTouchable *parentTouch=0)=0;
- Return a material pointer w.r.t. copy numbers of itself and ancestors.
- Must cope with parentTouch=0 for navigator's sake. Typically, return a default material if parentTouch=0.

virtual G4int GetNumberOfMaterials() const=0;
- Return 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

- G4VNestedParameterisation is a kind of G4VPVParameterization.
  - It can be used as an argument of G4PVParameterised.
  - All other arguments of G4PVParameterised are unaffected.
- Nested parameterization of placement volume is **not** supported.
  - All levels used as indices of material must be **repeated volume**. There cannot be a level of placement volume in between.

- If you are interested in, you can download a sample program
  - It is still a beta-version. It will be replaced with current examples/extended/runAndEvent/RE02 at 9.0 release.
  - In case you utilize this sample program and write a paper before the official release of this example, please acknowledge the authors explicitly.
    - Makoto Asai (SLAC)
    - Tsukasa Aso (Toyama N.C.M.T.)
    - Akinori Kimura (Ashikaga I.T.)
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 volume
  - In case speed of the moving volume is slow enough compared to speed of elementary particles, so that you can assume the position of moving volume is still within one event.
- Two tips to simulate moving objects:
  1. Use parameterized volume to represent the moving volume.
  2. Do not optimize (voxelize) the mother volume of the moving volume(s).
Moving objects - tip 1

- Use parameterized volume to represent the moving volume.
  - Use event number as a time stamp and calculate position/rotation of the volume as a function of event number.

```cpp
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(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.

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

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

- Position and rotation are set as the function of event number.
Moving objects - tip 2

- Do not optimize (voxelize) the mother volume of the moving volume(s).
  - If moving volume gets out of the original optimized voxel, the navigator gets lost.

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

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

- If you are interested in, you can download a sample program