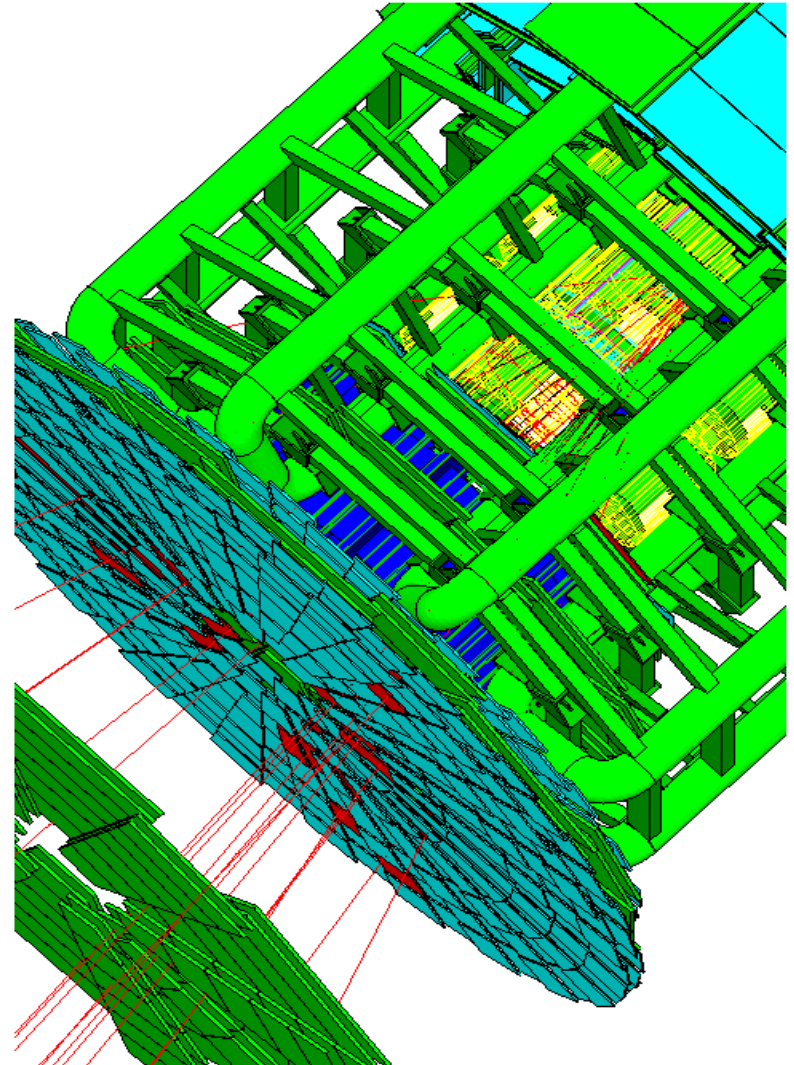


Geometry III

Makoto Asai (SLAC)
Geant4 Tutorial Course

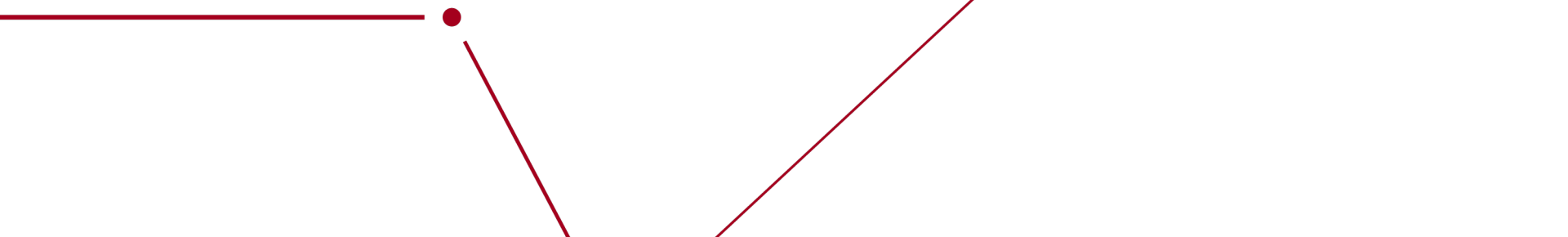
- Magnetic field
- Field integration and other types of field
- GDML/CAD interfaces
- Geometry checking tools
- Geometry optimization
- Parallel geometry
- Moving objects



Geant 4

Version 10.0-p01

Defining a magnetic field



- Create your Magnetic field class. It must be instantiated in ConstructSDandField() method of your DetectorConstruction

- Uniform field :

- Use an object of the G4UniformMagField class

```
G4MagneticField* magField =
```

```
    new G4UniformMagField(G4ThreeVector(1.*Tesla,0.,0.));
```

- Non-uniform field :

- Create your own concrete class derived from G4MagneticField and implement GetFieldValue method.

```
void MyField::GetFieldValue(
```

```
    const double Point[4], double *field) const
```

- Point[0..2] are **position in global coordinate system**, Point[3] is **time**
- field[0..2] are returning magnetic field

- One field manager is associated with the 'world' and it is set in G4TransportationManager
- Other volumes can override this
 - An alternative field manager can be associated with any logical volume
 - The field must accept **position in global coordinates** and return **field in global coordinates**
 - By default this is propagated to all its daughter volumes

```
G4FieldManager* localFieldMgr
```

```
    = new G4FieldManager(magField) ;
```

```
logVolume->setFieldManager(localFieldMgr, true) ;
```

where 'true' makes it push the field to all the volumes it contains, unless a daughter has its own field manager.

- Customizing the field propagation classes
 - Choosing an appropriate stepper for your field
 - Setting precision parameters

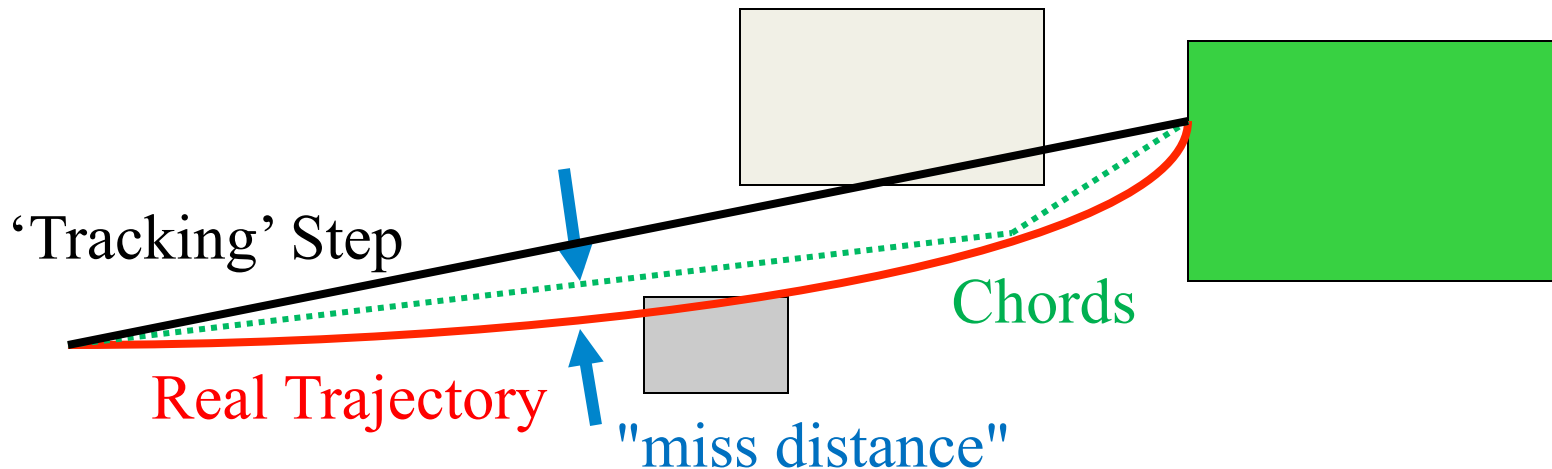
```
MyField* fMagneticField = new MyField();  
G4Fieldmanager* fFieldMgr = new G4FieldManager();  
fFieldMgr->SetDetectorField(fMagneticField);  
fFieldMgr->CreateChordFinder(fMagneticField);  
G4bool forceToAllDaughters = true;  
fMagneticLogical->SetFieldManager(fFieldMgr,  
                                   forceToAllDaughters);  
// Register the field and its manager for deleting  
G4AutoDelete::Register(fMagneticField);  
G4AutoDelete::Register(fFieldMgr);
```

- /example/basic/B5 is a good starting point

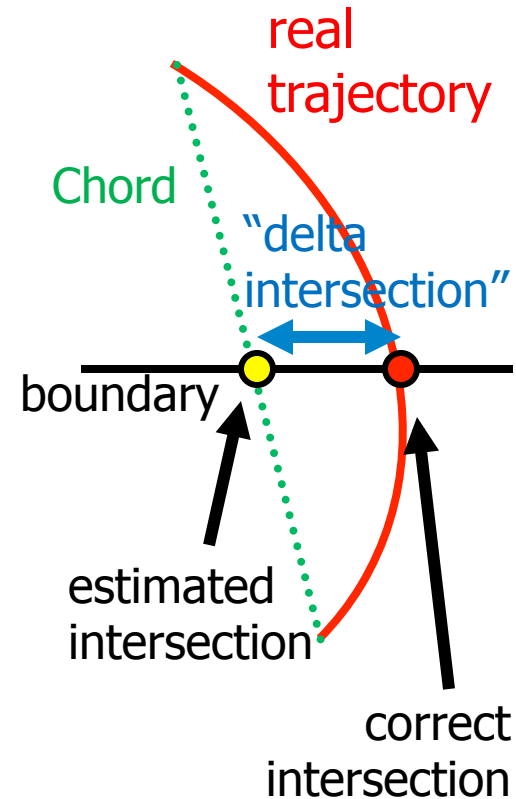
- In order to propagate a particle inside a field (e.g. magnetic, electric or both), we solve **the equation of motion** of the particle in the field.
- We use a Runge-Kutta method for the integration of the ordinary differential equations of motion.
 - Several Runge-Kutta ‘steppers’ are available.
- In specific cases other solvers can also be used:
 - In a uniform field, using the analytical solution.
 - In a smooth but varying field, with RK+helix.
- Using the method to calculate the track's motion in a field, Geant4 breaks up this curved path into linear chord segments.
 - We determine the chord segments so that they closely approximate the curved path.



- We use the chords to interrogate the **G4Navigator**, to see whether the track has crossed a volume boundary.
- One physics/tracking step can create several chords.
 - In some cases, one step consists of several helix turns.
- User can set the accuracy of the volume intersection,
 - By setting a parameter called the **“miss distance”**
 - It is a measure of the error in whether the approximate track intersects a volume.
 - It is quite expensive in CPU performance to set too small “miss distance”.



- In addition to the “miss distance” there are two more parameters which the user can set in order to adjust the accuracy (and performance) of tracking in a field.
 - These parameters govern the accuracy of the intersection with a volume boundary and the accuracy of the integration of other steps.
- The “delta intersection” parameter is the accuracy to which an intersection with a volume boundary is calculated. This parameter is especially important because it is used to limit a bias that our algorithm (for boundary crossing in a field) exhibits. The intersection point is always on the 'inside' of the curve. By setting a value for this parameter that is much smaller than some acceptable error, the user can limit the effect of this bias.



- The “**delta one step**” parameter is the accuracy for the endpoint of 'ordinary' integration steps, those which do not intersect a volume boundary. This parameter is a limit on the estimation error of the endpoint of each physics step.
- “**delta intersection**” and “**delta one step**” are strongly coupled. These values must be reasonably close to each other.
 - At most within one order of magnitude
- These tunable parameters can be set by

```
theChordFinder->SetDeltaChord( miss_distance );
```

```
theFieldManager->SetDeltaIntersection( delta_intersection );
```

```
theFieldManager->SetDeltaOneStep( delta_one_step );
```

- Further details are described in **Section 4.3 (Electromagnetic Field) of the Application Developers Manual**.

Geant 4

Version 10.0-p01

Field integration
and
Other types of field

- Runge-Kutta integration is used to compute the motion of a charged track in a general field. There are many general steppers from which to choose, of low and high order, and specialized steppers for pure magnetic fields.
- By default, Geant4 uses the classical fourth-order Runge-Kutta stepper (**G4ClassicalRK4**), which is general purpose and robust.
 - If the field is known to have specific properties, lower or higher order steppers can be used to obtain the results of same quality using fewer computing cycles.
- In particular, if the field is calculated from a field map, a lower order stepper is recommended. The less smooth the field is, the lower the order of the stepper that should be used.
 - The choice of lower order steppers includes the third order stepper **G4SimpleHeum**, the second order **G4ImplicitEuler** and **G4SimpleRunge**, and the first order **G4ExplicitEuler**. A first order stepper would be useful only for very rough fields.
 - For somewhat smooth fields (intermediate), the choice between second and third order steppers should be made by trial and error.

- Trying a few different types of steppers for a particular field or application is suggested if maximum performance is a goal.
- Specialized steppers for pure magnetic fields are also available. They take into account the fact that a local trajectory in a slowly varying field will not vary significantly from a helix.
 - Combining this in with a variation, the Runge-Kutta method can provide higher accuracy at lower computational cost when large steps are possible.
- To change the stepper
theChordFinder
 - >`GetIntegrationDriver()`
 - >`RenewStepperAndAdjust(newStepper);`
- Further details are described in [Section 4.3 \(Electromagnetic Field\) of the Application Developers Manual](#).

Other types of field

- The user can create their own type of field, inheriting from **G4VField**, and an associated **Equation of Motion** class (inheriting from **G4EqRhs**) to simulate other types of fields. Field can be time-dependent.
- For pure electric field, Geant4 has **G4ElectricField** and **G4UniformElectricField** classes. For combined electromagnetic field, Geant4 has **G4ElectroMagneticField** class.
- Equation of Motion class for electromagnetic field is **G4MagElectricField**.

```
G4ElectricField* fEMfield
```

```
    = new G4UniformElectricField( G4ThreeVector(0., 100000.*kilovolt/cm, 0.) );
```

```
G4EqMagElectricField* fEquation = new G4EqMagElectricField(fEMfield);
```

```
G4MagIntegratorStepper* fStepper = new G4ClassicalRK4( fEquation, nvar );
```

```
G4FieldManager* fFieldMgr
```

```
    = G4TransportationManager::GetTransportationManager()-> GetFieldManager();
```

```
fFieldMgr->SetDetectorField( fEMfield );
```

```
G4MagInt_Driver* fIntgrDriver
```

```
    = new G4MagInt_Driver(fMinStep, fStepper,  
                          fStepper->GetNumberOfVariables() );
```

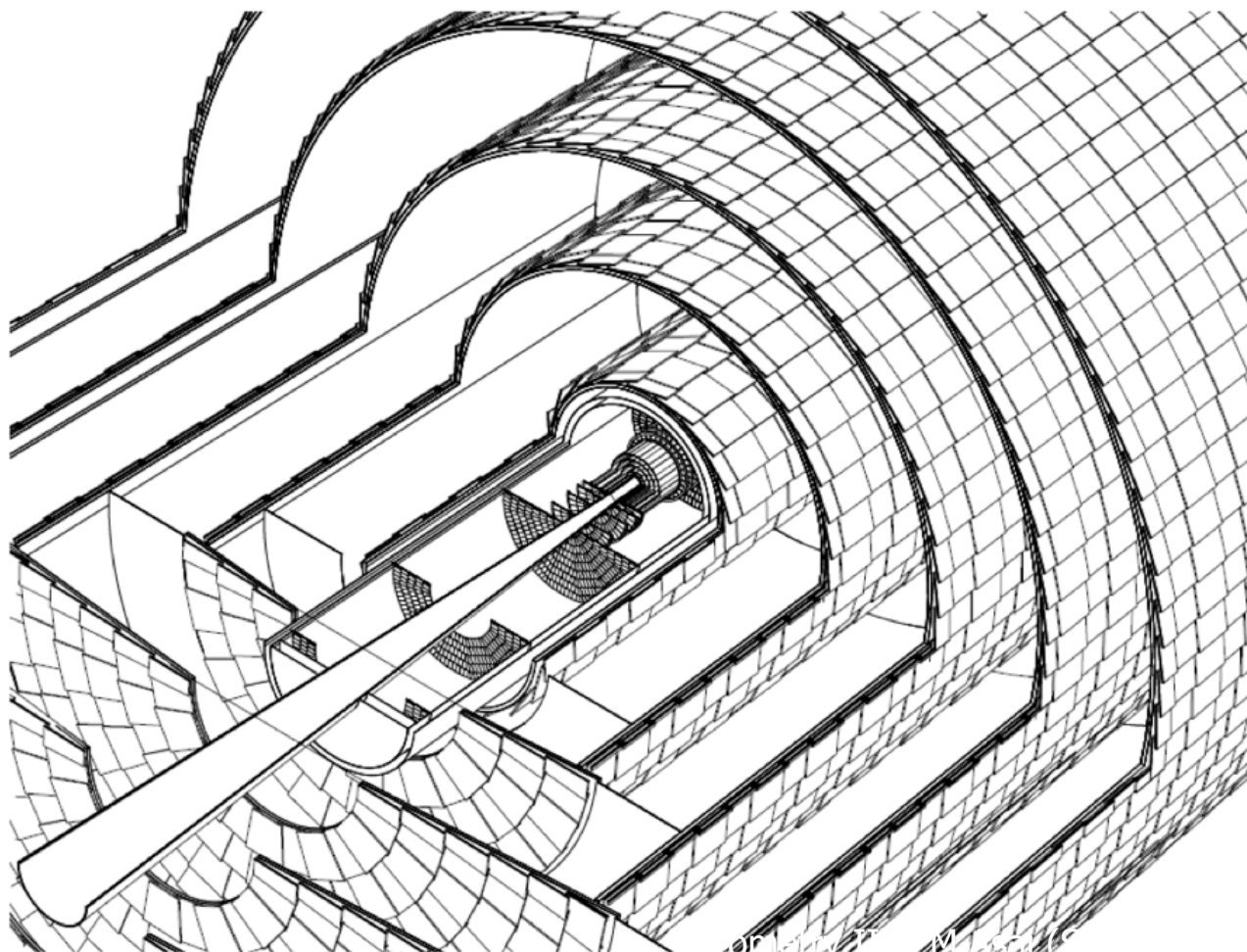
```
G4ChordFinder* fChordFinder = new G4ChordFinder(fIntgrDriver);
```

Geant 4

Version 10.0-p01

GDML/CAD interfaces

- Up to now, the course has shown how to define materials and volumes from C++.
- This part of slides shows some alternate ways to define geometry at runtime by providing a file-based detector description.



Silicon Pixel & Microstrip
Tracker for Collider
Detector
N. Graf, J. McCormick,
LCDD, SLAC

- An XML-based language designed as an application independent persistent format for describing the geometries of detectors.
 - Implements “geometry trees” which correspond to the hierarchy of volumes a detector geometry can be composed of
 - Allows materials to be defined and solids to be positioned
- Because it is pure XML, GDML can be used universally
 - Not just for Geant4
 - Can be format for interchanging geometries among different applications.
 - Can be used to translate CAD geometries to Geant4
- XML is simple
 - Rigid set of rules, self-describing data validated against schema
- XML is extensible
 - Easy to add custom features, data types
- XML is Interoperable to OS's, languages, applications
- XML has hierarchical structure
 - Appropriate for Object-Oriented programming
 - Detector/sub-detector relationships

- Contains numerical values of constants, positions, rotations and scales that will be used later on in the geometry construction.
 - Uses CLHEP expressions
- Constants
 - `<constant name="length" value="6.25"/>`
- Variables
 - `<variable name="x" value="6"/>`
 - Once defined, can be used anywhere later, e.g.
 - `<variable name="y" value="x/2"/>`
 - `<box name="my_box" x="x" y="y" z="x+y"/>`
- Positions
 - `<position name="P1" x="25.0" y="50.0" z="75.0" unit="cm"/>`
- Rotations
 - `<rotation name="RotateZ" z="30" unit="deg"/>`
- Matrices
 - `<matrix name="m" coldim="3" values=" 0.4 9 126
8.5 7 21
34.6 7 9"/>`

- Simple Elements

```
<element Z="8" formula="O" name="Oxygen" >
  <atom value="16" />
</element>
```
- Material by number of atoms (“molecule”)

```
<material name="Water" formula="H2O">
  <D value="1.0" />
  <composite n="2" ref="Hydrogen" />
  <composite n="1" ref="Oxygen" />
</material>
```
- Material as a fractional mixture of elements or materials, (“compound”):

```
<material formula="air" name="Air" >
  <D value="0.00129" />
  <fraction n="0.7" ref="Nitrogen" />
  <fraction n="0.3" ref="Oxygen" />
</material>
```

- Box
- Cone Segment
- Ellipsoid
- Elliptical Tube
- Elliptical Cone
- Orb
- Paraboloid
- Parallelepiped
- Polycone
- Polyhedron
- Sphere
- Torus Segment
- Trapezoid (x&y vary along z)
- General Trapezoid
- Tube with Hyperbolic Profile
- Cut Tube
- Tube Segment
- Twisted Box
- Twisted Trapezoid
- Twisted General Trapezoid
- Twisted Tube Segment
- Extruded Solid
- Tessellated Solid
- Tetrahedron

- The Boolean operations union, subtraction and intersection are also supported, e.g.

```
<box name="box_1" x="1" y="5" z="20" />  
<box name="box_2" x="4" y="4.5" z="18" />  
<union name="union" >  
  <first ref="box_1" />  
  <second ref="box_2"/>  
  <positionref ref="union_position" />  
  <rotationref ref="union_rotation" />  
</union>
```

- Volumes are created from solids and materials that were previously defined in this or a linked GDML file
- Both logical and physical volumes are defined in one

```
<volume name="World">  
  <materialref ref="Air"/>  
  <solidref ref="WorldBox"/>  
  <physvol>  
    <volumeref ref="vol0"/>  
      <positionref ref="center"/>  
      <rotationref ref="identity"/>  
    </physvol>  
</volume>
```

- GDML files can be directly imported into Geant4 geometry, using the GDML plug-in facility:

```
#include "G4GDMLParser.hh"
```
- Generally you will want to put the following lines into your DetectorConstruction class:

```
G4GDMLParser parser;  
parser.Read("geometryFile.gdml");  
G4VphysicalVolume* W=parser.GetWorldVolume();
```
- To include the Geant4 module for GDML,
 - Install the XercesC parser (version 2.8.0 or 3.0.0)
<http://xerces.apache.org/xerces-c/download.cgi>
 - Set appropriate environment variables when G4 libraries are built
- Examples available in:

```
$G4INSTALL/examples/extended/persistency/gdml
```

- Users with 3D engineering drawings may want to incorporate these into their Geant4 simulation as directly as possible
- Difficulties include:
 - Proprietary, undocumented or changing CAD formats
 - Usually no connection between geometry and materials
 - Mismatch in level of detail required to machine a part and that required to transport particles in that part
- CAD is never as easy as you might think (if the geometry is complex enough to require CAD in the first place)
- **CADMesh** is a direct CAD model import interface for GEANT4 optionally leveraging VCGLIB, and ASSIMP by default. Currently 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.
 - <https://code.google.com/p/cadmesh/>
- One output format most CAD programs do support is STEP
 - Not a complete solution, in particular does not contain material information
 - There are movements under way to get new formats that contain additional information, but none yet widely adopted.

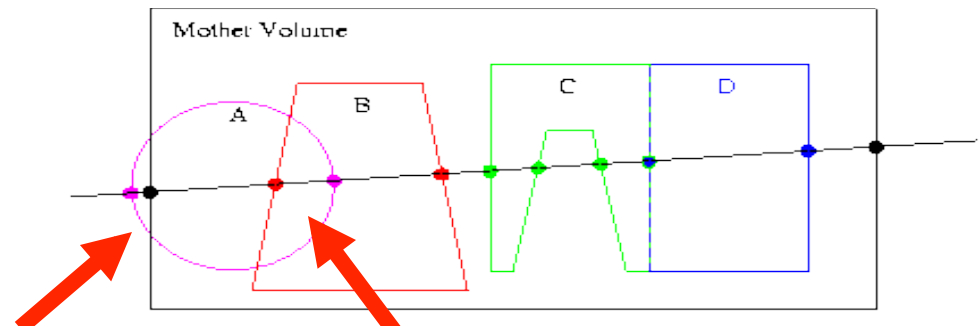
- Imperfect, but still helpful solutions are tools to convert STEP to GDML and provide the user a way to add materials information
- There are two cases where existing CAD programs have added GDML export features. Since these CAD programs can also read in STEP, they can be used as STEP to GDML converters.
 - Neither option is free, neither option works perfectly
 - ST-Viewer
 - <http://www.steptools.com>
 - FastRad
 - GDML export extension was funded by European Space Agency
 - Not free except for limited, trial mode that can handle only a small number of volumes
 - <http://www.fastrad.net/>
- Discussion of these solutions takes place in the Geant4 Persistency forum:
<http://hypernews.slac.stanford.edu/HyperNews/geant4/get/persistency.html>
- Useful technical note:
 - Linking computer-aided design (CAD) to Geant4-based Monte Carlo simulations for precise implementation of complex treatment head geometries., Constantin et. al., Phys Med Biol. 2010 Apr 21;55(8):N211-20. Epub 2010 Mar 26

Geant 4

Version 10.0-p01

Geometry checking tools

- 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**.
- Geant4 **does not allow** for malformed geometries, **neither protruding nor overlapping**.
 - The behavior of navigation is unpredictable for such cases.
- 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 (DAVID, OLAP)



protruding

- M.Asai (

overlapping

- 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.

- 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`

- 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)	-----
. . .						

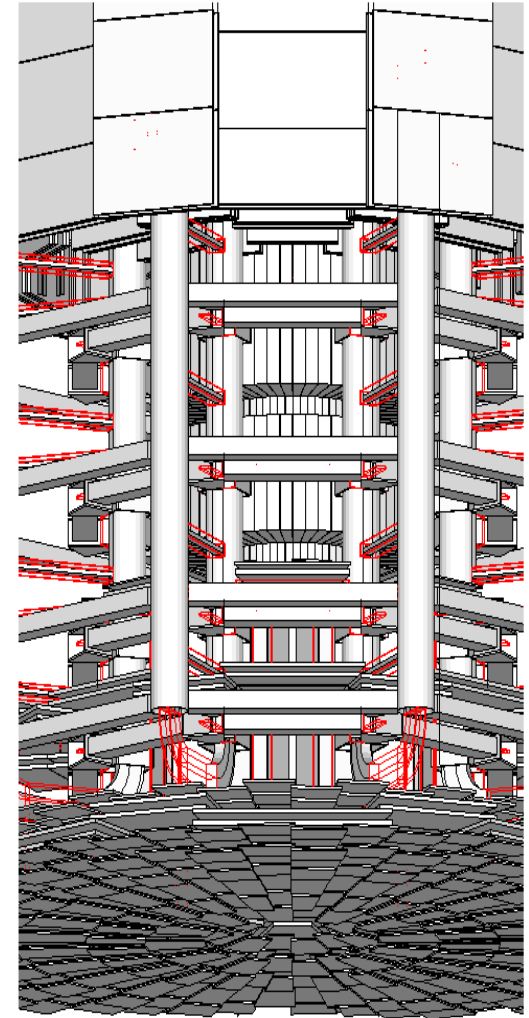
```
Which in the coordinate system of Tracker[0] are:
```

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)	-----
. . .						

- 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/>

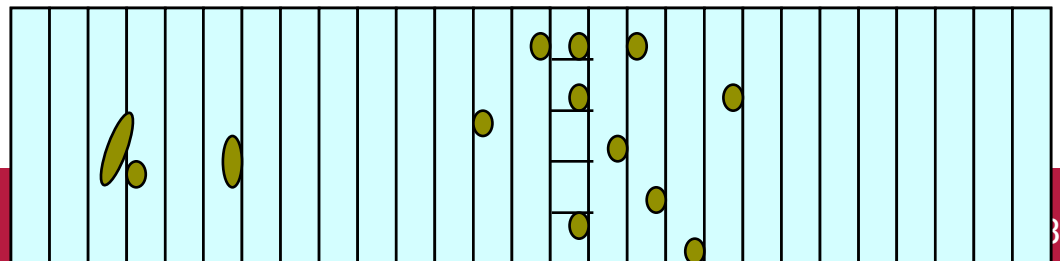


Geant 4

Version 10.0-p01

Geometry optimization
("voxelization")

- In case of Geant 3.21, the user had to carefully implement his/her geometry to maximize the performance of geometrical navigation.
- While in Geant4, user's geometry is automatically optimized to most suitable to the navigation. - "Voxelization"
 - 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 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

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

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

- Allows setting of visualization attributes for voxels

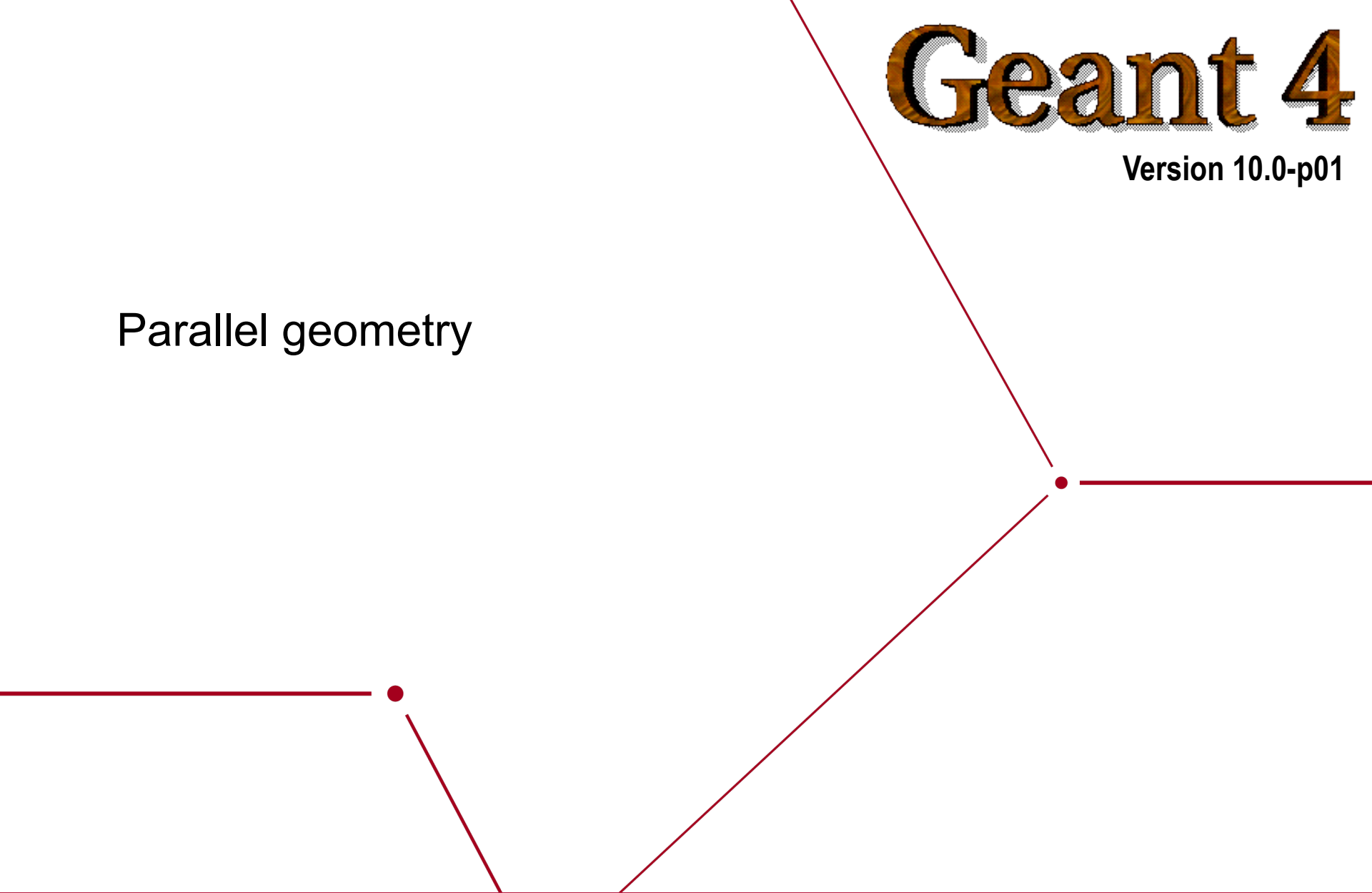
```
G4DrawVoxels::SetVoxelsVisAttributes(...)
```

- useful for debugging purposes

Geant 4

Version 10.0-p01

Parallel geometry

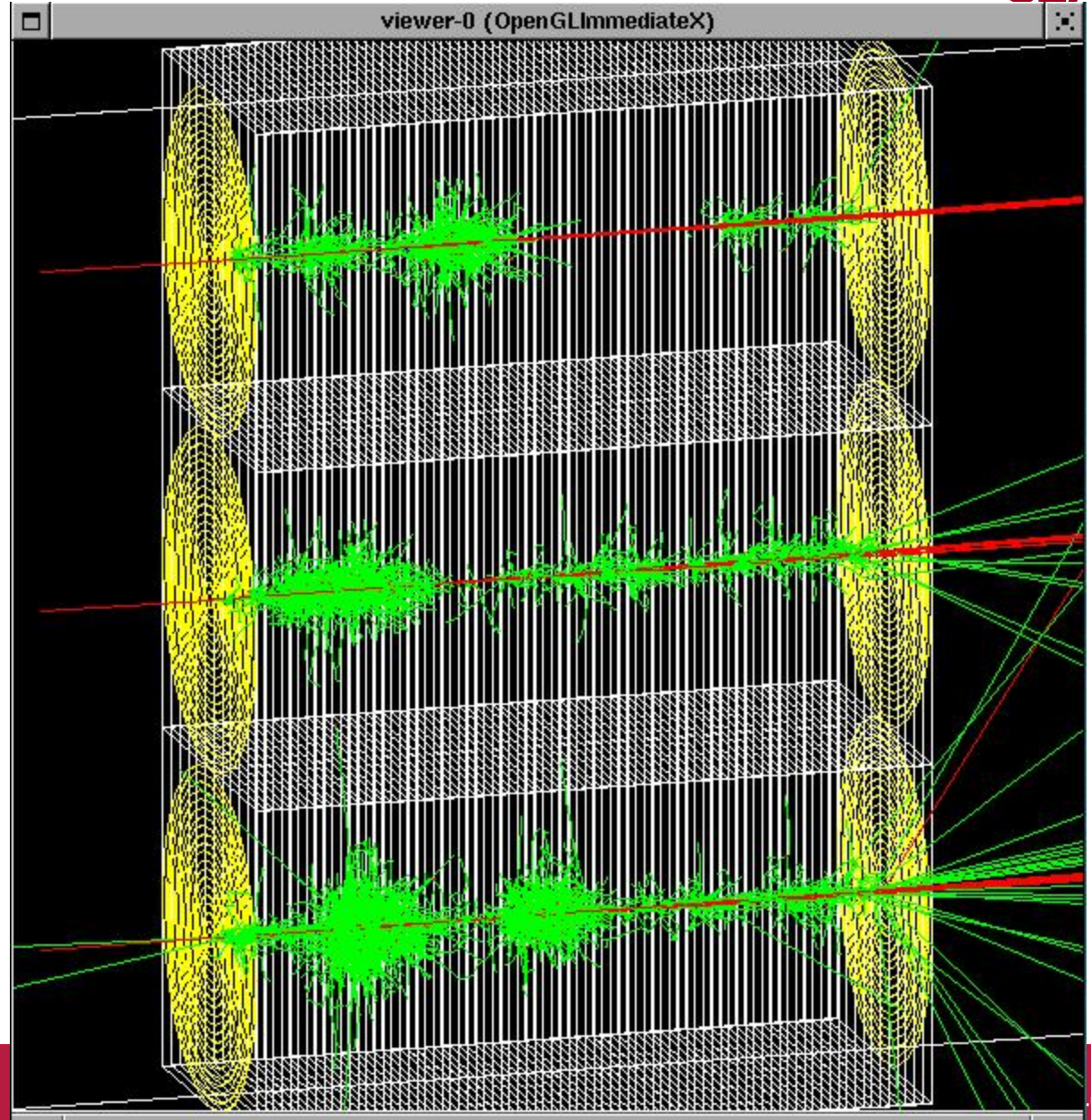


- In the previous versions, we have already had several ways of utilizing a concept of parallel world. But the usages are quite different to each other.
 - Ghost volume for shower parameterization assigned to G4GlobalFastSimulationManager
 - Readout geometry assigned to G4VSensitiveDetector
 - Importance field geometry for geometry importance biasing assigned to importance biasing process
 - Scoring geometry assigned to scoring process
- We merge all of them into common parallel world scheme.
 - Readout geometry for sensitive detector will be kept for backward compatibility.
 - Other current “parallel world schemes” became obsolete.

- Occasionally, it is not straightforward to define sensitivity, importance or envelope to be assigned to volumes in the mass geometry.
 - Typically a geometry built machinery by CAD, GDML, DICOM, etc. has this difficulty.
- New parallel navigation functionality allows the user to define more than one worlds simultaneously.
 - New G4Transportation process sees all worlds simultaneously.
 - A step is limited not only by the boundary of the mass geometry but also by the boundaries of parallel geometries.
 - 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.

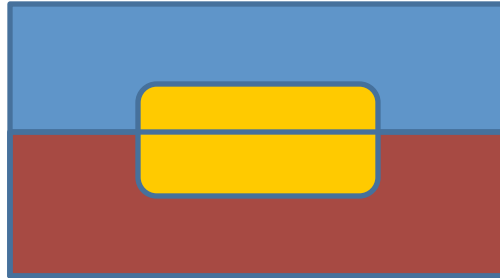
- **G4VUserParallelWorld** is the new base class where the user implements a parallel world.
 - The world physical volume of the parallel world is provided by G4RunManager as a clone of the mass geometry.
 - All UserParallelWorlds must be registered to UserDetectorConstruction.
 - Each parallel world has its dedicated G4Navigator object, that is automatically assigned when it is constructed.
- Though all worlds will be comprehensively taken care by G4Transportation process for their navigations, each parallel world must have its own process to achieve its purpose.
 - For 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.

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

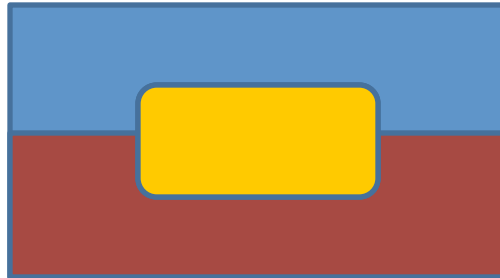


Layered mass geometries in parallel world

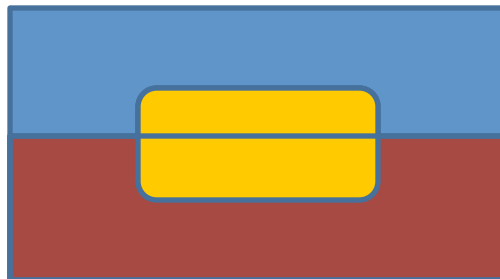
- Suppose you implement a wooden brick floating on the water.



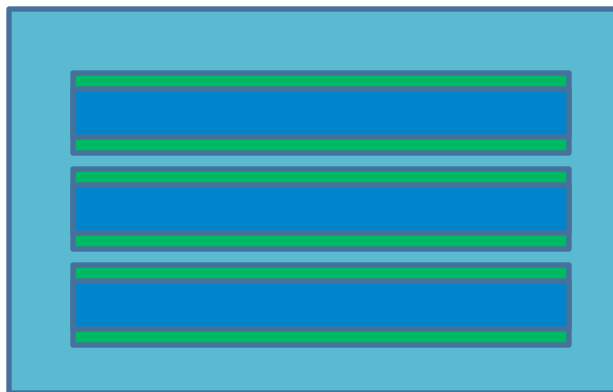
- Dig a hole in water...



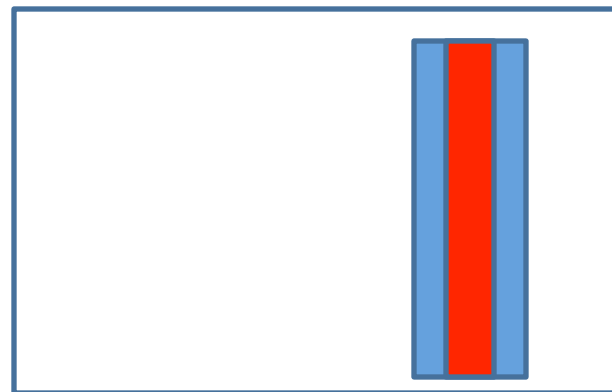
- Or, chop a brick into two and place them separately...



- Parallel geometry may be stacked on top of mass geometry or other parallel world geometry, allowing a user to define more than one worlds with materials (and region/cuts).
 - Track will see the material of top-layer, if it is null, then one layer beneath.
 - Alternative way of implementing a complicated geometry
 - Rapid prototyping
 - Safer, more flexible and powerful extension of the concept of “many” in Geant3



Mass world

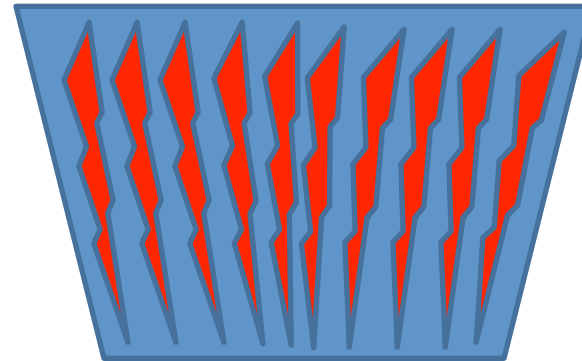


Parallel world

- 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 parameterization such as GFLASH may have its own geometry



Geometry seen by e^+ , e^- , γ



Geometry seen by other particles

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.

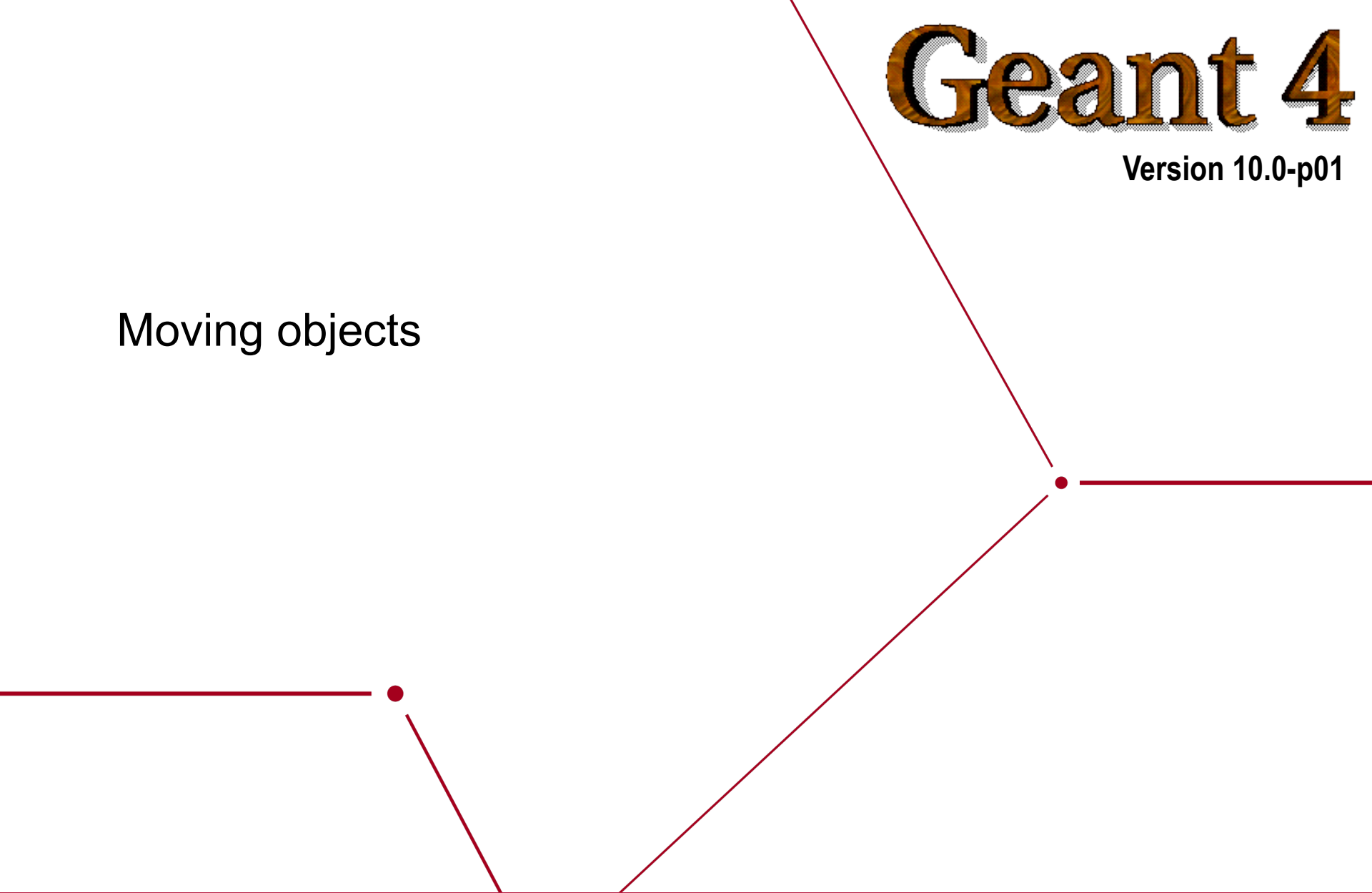
```
void RE04ParallelWorldConstruction::Construct()
{
  //
  // World
  G4VPhysicalVolume* ghostWorld = GetWorld();
  G4LogicalVolume* worldLogical = ghostWorld->GetLogicalVolume();
  //
  // material defined in the mass world
  G4Material* water = G4Material::GetMaterial("G4_WATER");
  //
  // parallel 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 is provided as a clone of the world volume of the mass geometry. The user cannot create it.
- You can fill contents regardless of the volumes in the mass geometry.
- Logical volumes in a parallel world needs not to have a material.

Geant 4

Version 10.0-p01

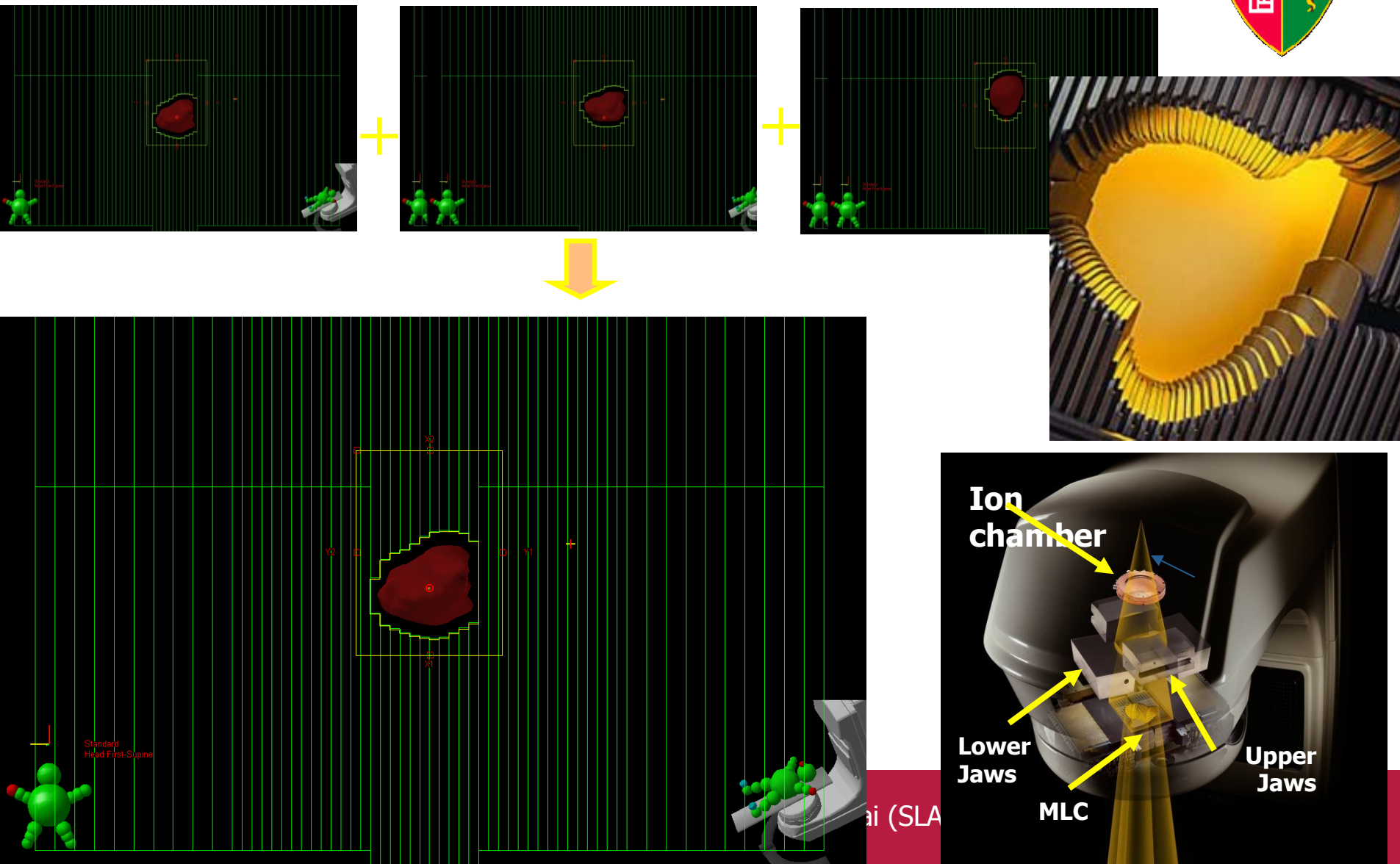
Moving objects



4D RT Treatment Plan



Source: Lei Xing, Stanford University



- 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.

```
void MyMovingVolumeParameterisation::ComputeTransformation
```

```
(const G4int copyNo, G4VPhysicalVolume *physVol) const
```

```
{
```

```
static G4RotationMatrix rMat;
```

```
G4int eID = 0;
```

```
const G4Event* evt = G4RunManager::G4EventManager::GetEvent();
```

```
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(z,0,0));
```

```
physVol->SetRotation(&rMat0);
```

```
}
```

Null pointer must be protected.

This method is also invoked while

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) t

Position and rotation are set as the function of event number.

- 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.

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.