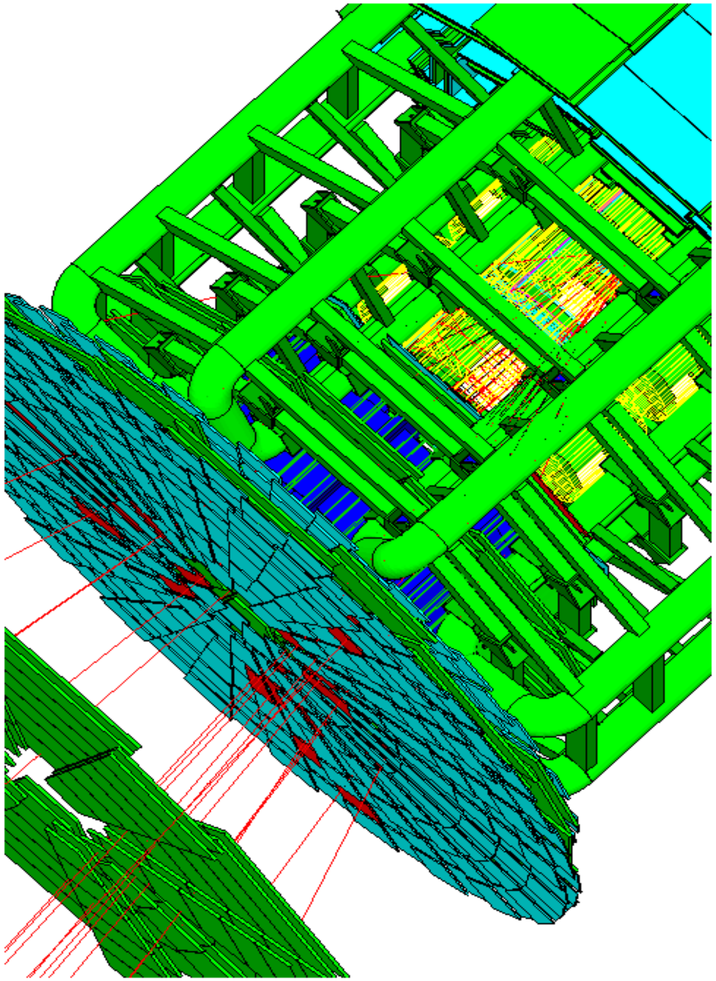
Geometry and Fields:



Further and advanced features

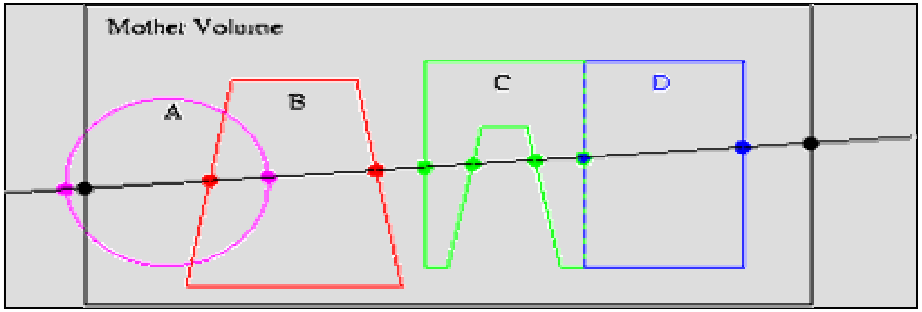
J. Apostolakis & G. Cosmo

PART I

Detector Description
Advanced features

- *Debugging tools*
- *Creating geometry in simpler way*
  - *The Geant4 Geometrical Editor (tabular)*
  - *Grouping volumes*
  - *Reflections of volumes and hierarchies*
- *User defined solids*
- *Interface to CAD systems*

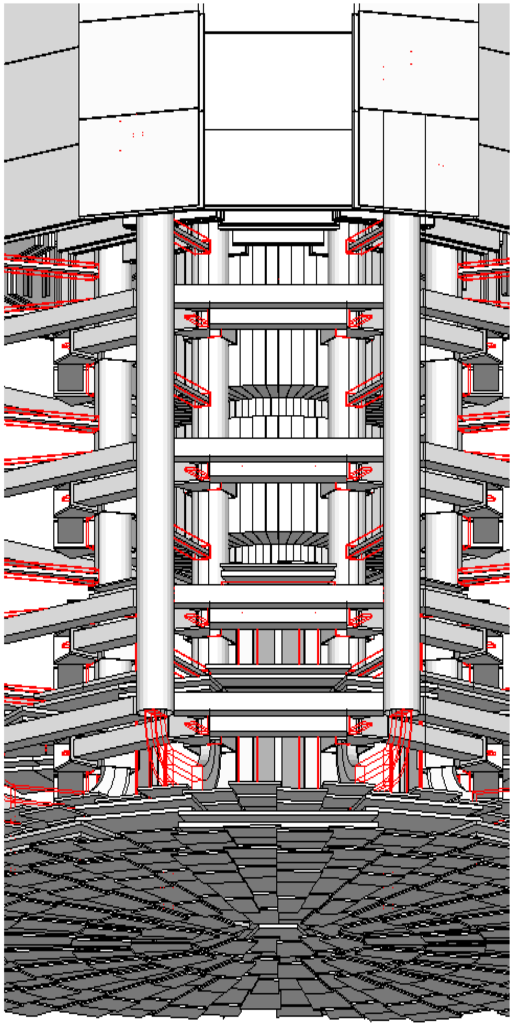
Debugging geometries



- An *overlapping* volume is a contained 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
- The problem of detecting overlaps between volumes is bounded by the complexity of the solid models description
- Utilities are provided for detecting wrong positioning
  - Graphical tools (DAVID & OLAP)
  - Kernel run-time commands

Debugging tools: DAVID

- DAVID is a graphical debugging tool for detecting potential intersections of volumes
  - It intersects volumes directly, using their graphical representations.
- 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/GEANT4/vis/DAWN/About_DAVID.html



Debugging run-time commands

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

`geometry/test/run` or `geometry/test/grid_test`

- to start verification of geometry for overlapping regions based on a standard grid setup, limited to the first depth level

`geometry/test/recursive_test`

- applies the grid test to all depth levels (may require lots of CPU time!)

`geometry/test/cylinder_test`

- shoots lines according to a cylindrical pattern

`geometry/test/line_test`

- to shoot a line along a specified direction and position

`geometry/test/position`

- to specify position for the `line_test`

`geometry/test/direction`

- to specify direction for the `line_test`

Debugging run-time commands - 2

- Example layout:

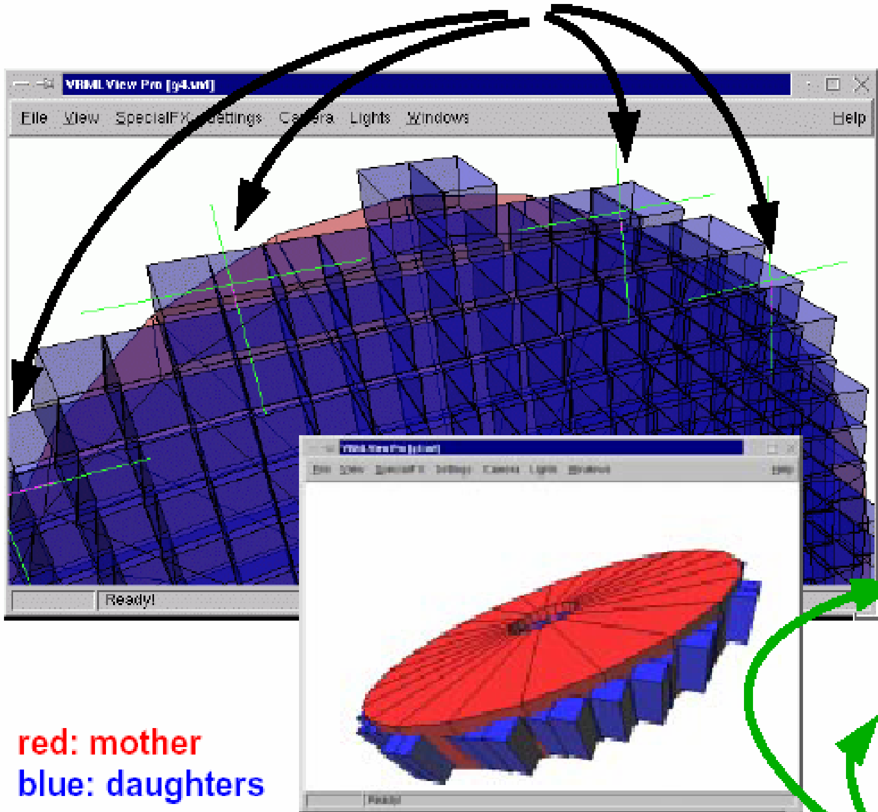
```
GeomTest: no daughter volume extending outside mother detected.
GeomTest Error: Overlapping daughter volumes
    The volumes Tracker[0] and Overlap[0],
    both daughters of volume World[0],
    appear to overlap at the following points in global coordinates: (list truncated)
  length (cm)    -----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

Debugging tools: OLAP

- Uses tracking of neutral particles to verify boundary crossing in opposite directions
- Stand-alone batch application
  - Provided as extended example
  - Can be combined with a graphical environment and GUI (ex. Qt library)
  - Integrated in the CMS Iguana Framework

Debugging tools: OLAP

graphical indication of detected overlaps



red: mother
blue: daughters

daughters are protruding their mother

Geant4 Macro:

```
/vis/scene/create
/vis/sceneHandler/create VRML2FILE
/vis/viewer/create
/olap/goto ECalEnd
/olap/grid 7 7 7
/olap/trigger
/vis/viewer/update
```

Output:

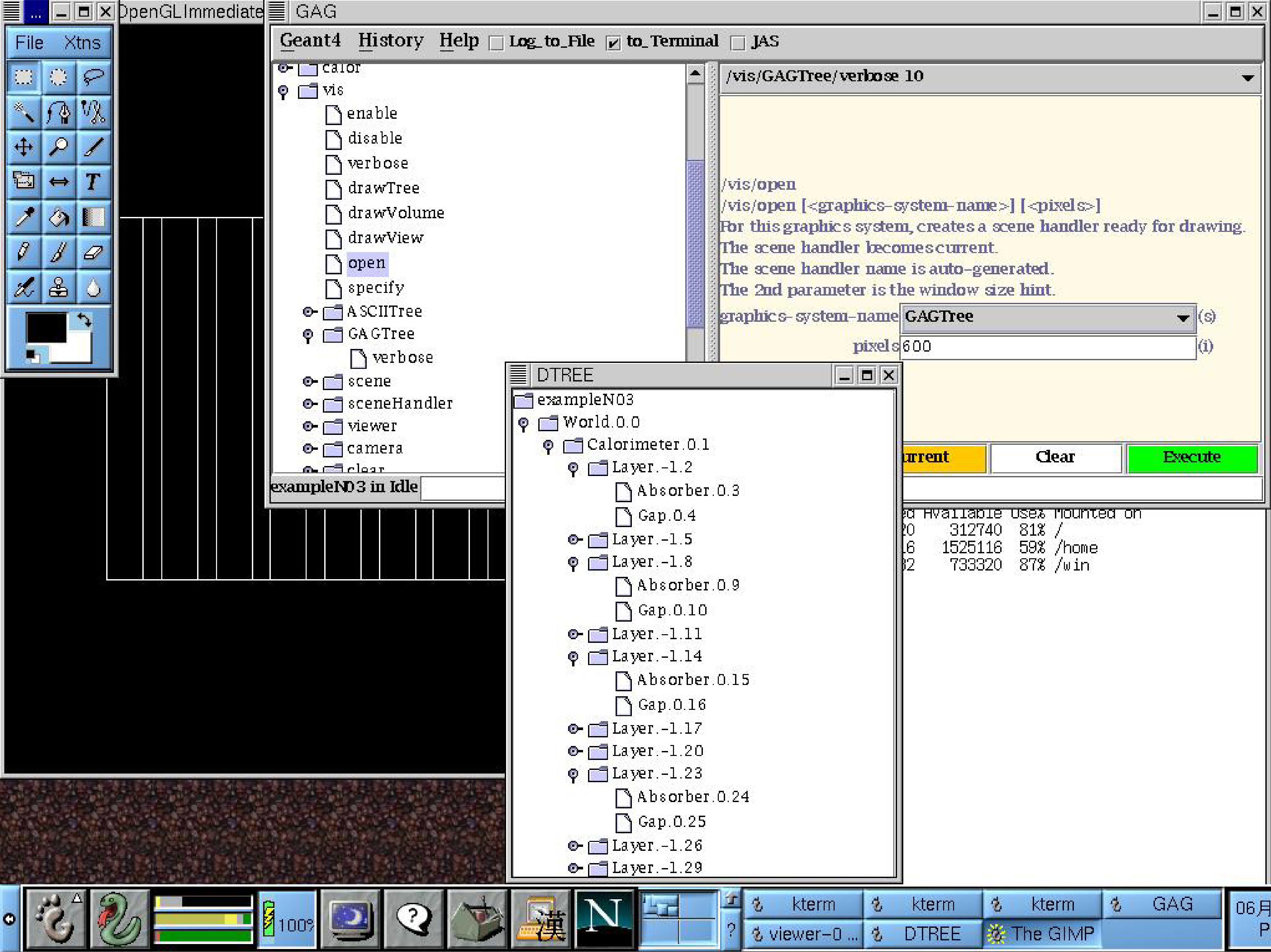
```
delta=59.3416
vol 1: point=(560.513,1503.21,-141.4)
vol 2: point=(560.513,1443.86,-141.4)
A -> B:
[0]:  ins=[2]   PVName=[NewWorld:0] Type=[N] ...
[1]:  ins=[0]   PVName=[ECalEndcap:0] Type=[N] ..
[2]:  ins=[1]   PVName=[ECalEndcap07:38] Type=[N]

B -> A:
[0]:  ins=[2]   PVName=[NewWorld:0] Type=[N] ...
```

NavigationHistories of points of overlap
(including: info about translation, rotation, solid specs)

Visualizing detector geometry tree

- Built-in commands defined to display the hierarchical geometry tree
  - As simple ASCII text structure
  - Graphical through GUI (combined with GAG)
  - As XML exportable format
- Implemented in the visualization module
  - As an additional graphics driver
- G3 DTREE capabilities provided and more



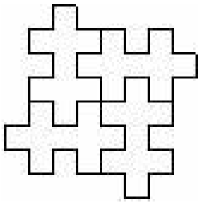
OpenGLImmediate
File Xtns
GAG
Geant4 History Help Log_to_File to_Terminal JAS
calor
vis
enable
disable
verbose
drawTree
drawVolume
drawView
open
specify
ASCIITree
GAGTree
verbose
scene
sceneHandler
viewer
camera
clear
exampleN03 in Idle
/vis/GAGTree/verbose 10
/vis/open
/vis/open [<graphics-system-name>] [<pixels>]
For this graphics system, creates a scene handler ready for drawing.
The scene handler becomes current.
The scene handler name is auto-generated.
The 2nd parameter is the window size hint.
graphics-system-name GAGTree (s)
pixels 600 (i)
Clear
Execute
DTREE
exampleN03
World.0.0
Calorimeter.0.1
Layer.-1.2
Absorber.0.3
Gap.0.4
Layer.-1.5
Layer.-1.8
Absorber.0.9
Gap.0.10
Layer.-1.11
Layer.-1.14
Absorber.0.15
Gap.0.16
Layer.-1.17
Layer.-1.20
Layer.-1.23
Absorber.0.24
Gap.0.25
Layer.-1.26
Layer.-1.29
Available Use% Mounted on
312740 81% /
1525116 59% /home
733320 87% /win
kterm
kterm
kterm
GAG
viewer-0 ...
DTREE
The GIMP

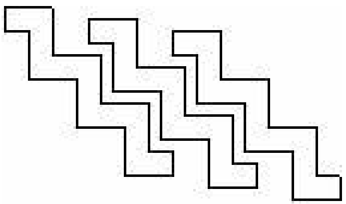
GGE (Graphical Geometry Editor)

- Implemented in JAVA, GGE is a graphical geometry editor compliant to Geant4. It allows to:
  - Describe a detector geometry including:
    - materials, solids, logical volumes, placements
  - Graphically visualize the detector geometry using a Geant4 supported visualization system, e.g. DAWN
  - Store persistently the detector description
  - Generate the C++ code according to the Geant4 specifications
- GGE can be downloaded from Web as a separate tool:
  - `http://erpc1.naruto-u.ac.jp/~geant4/`

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





- ***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

G4AssemblyVolume

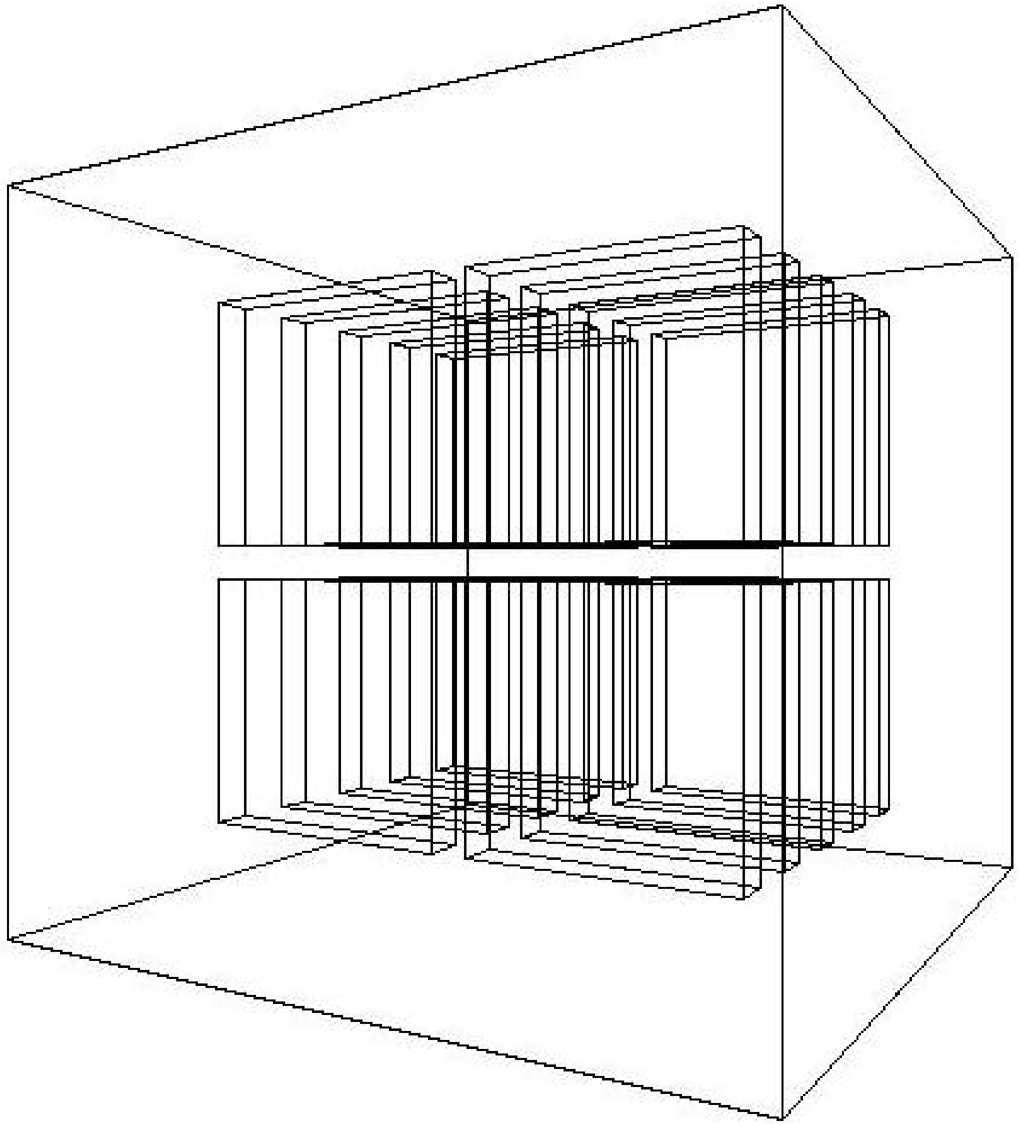
```
G4AssemblyVolume( G4LogicalVolume* volume,
                  G4ThreeVector& translation,
                  G4RotationMatrix* rotation);
```

- Helper class to combine logical volumes in arbitrary way
  - Participating logical volumes are treated as triplets
    - logical volume, translation, rotation
  - 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)

Assembly of volumes:
example -1

```
// Define a plate
   G4VSolid* PlateBox = new G4Box( "PlateBox", plateX/2., plateY/2., plateZ/2. );
   G4LogicalVolume* plateLV = new G4LogicalVolume( PlateBox, Pb, "PlateLV", 0, 0, 0 );
 // Define one layer as one assembly volume
   G4AssemblyVolume* assemblyDetector = new G4AssemblyVolume();
 // Rotation and translation of a plate inside the assembly
   G4RotationMatrix Ra; G4ThreeVector Ta;
 // Rotation of the assembly inside the world
   G4RotationMatrix Rm;
 // Fill the assembly by the plates
   Ta.setX( caloX/4. ); Ta.setY( caloY/4. ); Ta.setZ( 0. );
   assemblyDetector->AddPlacedVolume( plateLV, G4Transform3D(Ra,Ta) );
   Ta.setX( -1*caloX/4. ); Ta.setY( caloY/4. ); Ta.setZ( 0. );
   assemblyDetector->AddPlacedVolume( plateLV, G4Transform3D(Ra,Ta) );
   Ta.setX( -1*caloX/4. ); Ta.setY( -1*caloY/4. ); Ta.setZ( 0. );
   assemblyDetector->AddPlacedVolume( plateLV, G4Transform3D(Ra,Ta) );
   Ta.setX( caloX/4. ); Ta.setY( -1*caloY/4. ); Ta.setZ( 0. );
   assemblyDetector->AddPlacedVolume( plateLV, G4Transform3D(Ra,Ta) );
 // Now instantiate the layers
   for( unsigned int i = 0; i < layers; i++ ) {
     // Translation of the assembly inside the world
     G4ThreeVector Tm( 0,0,i*(caloZ + caloCaloOffset) - firstCaloPos );
     assemblyDetector->MakeImprint( worldLV, G4Transform3D(Rm,Tm) );
   }
```

Assembly of volumes: example -2



Reflecting solids

- `G4ReflectedSolid`
  - utility class representing a solid shifted from its original reference frame to a new *reflected* 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
- Reflections can be applied to CSG and specific solids

Reflecting hierarchies of volumes - 1

`G4ReflectionFactory::Place(…)`

- Used for normal placements:
  1. Performs the transformation decomposition
  2. Generates a new reflected solid and logical volume
     - Retrieves it from a map if the reflected object is already created
  3. Transforms any daughter and places them in the given mother
  4. Returns a pair of physical volumes, the second being a placement in the reflected mother

```
G4PhysicalVolumesPair
Place(const G4Transform3D&   transform3D, // the transformation
      const G4String&        name,        // the actual name
            G4LogicalVolume* LV,          // the logical volume
            G4LogicalVolume* motherLV,    // the mother volume
            G4bool           noBool,      // currently unused
            G4int            copyNo)      // optional copy number
```

Reflecting hierarchies of volumes - 2

`G4ReflectionFactory::Replicate(…)`

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

```
G4PhysicalVolumesPair
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
```

User defined solids

- All solids should derive from **`G4VSolid`** and implement its abstract interface
  - will guarantee the solid is treated as any other solid predefined in the kernel
- Basic functionalities required for a solid
  - Compute distances to/from the shape
  - Detect if a point is inside the shape
  - Compute the surface normal to the shape at a given point
  - Compute the extent of the shape
  - Provide few visualization/graphics utilities

What a solid should reply to…- 1

```
EInside Inside(const G4ThreeVector& p) const;
```

- *Should return, considering a predefined tolerance:*
  - `kOutside` - *if the point at offset* `p` *is outside the shapes boundaries*
  - `kSurface` - *if the point is close less than* `Tolerance/2` *from the surface*
  - `kInside` - *if the point is inside the shape boundaries*

```
G4ThreeVector SurfaceNormal(const G4ThreeVector& p) const;
```

- *Should return the outwards pointing unit normal of the shape for the surface closest to the point at offset* `p`.

```
G4double DistanceToIn(const G4ThreeVector& p,
                      const G4ThreeVector& v) const;
```

- *Should return the distance along the normalized vector* `v` *to the shape from the point at offset* `p`. *If there is no intersection, returns* `kInfinity`. *The first intersection resulting from 'leaving' a surface/volume is discarded. Hence, it is tolerant of points on the surface of the shape*

What a solid should reply to…- 2

```
G4double DistanceToIn(const G4ThreeVector& p) const;
```

- *Calculates the distance to the nearest surface of a shape from an outside point* `p`. *The distance can be an underestimate*

```
G4double DistanceToOut(const G4ThreeVector& p,
                       const G4ThreeVector& v,
                       const G4bool  calcNorm=false,
                             G4bool* validNorm=0,
                             G4ThreeVector* n=0) const;
```

- *Returns the distance along the normalised vector* `v` *to the shape, from a point at an offset* `p` *inside or on the surface of the shape. Intersections with surfaces, when the point is less than* `Tolerance/2` *from a surface must be ignored. If* `calcNorm` *is* `true`*, then it must also set* `validNorm` *to either:*
  - `True` - *if the solid lies entirely behind or on the exiting surface. Then it must set* `n` *to the outwards normal vector (the Magnitude of the vector is not defined)*
  - `False` - *if the solid does not lie entirely behind or on the exiting surface*

```
G4double DistanceToOut(const G4ThreeVector& p) const;
```

- *Calculates the distance to the nearest surface of a shape from an inside point* `p`. *The distance can be an underestimate*

Solid: more functions…

```
G4bool CalculateExtent(const EAxis pAxis,
                       const G4VoxelLimits& pVoxelLimit,
                       const G4AffineTransform& pTransform,
                             G4double& pMin, G4double& pMax) const;
```

- *Calculates the minimum and maximum extent of the solid, when under the specified transform, and within the specified limits. If the solid is not intersected by the region, return* `false`, *else return* `true`

<u>Member functions for the purpose of visualization</u>:

```
void DescribeYourselfTo (G4VGraphicsScene& scene) const;
```

- *"double dispatch" function which identifies the solid to the graphics scene*

```
G4VisExtent GetExtent () const;
```

- *Provides extent (bounding box) as possible hint to the graphics view*

Interface to CAD systems

- Models imported from CAD systems can describe the solid geometry of detectors made by large number of elements with the greatest accuracy and detail
  - A solid model contains the purely geometrical data representing the solids and their position in a given reference frame
  - Material information is generally missing
- Solid descriptions of detector models could be imported from CAD systems
  - e.g. Euclid & Pro/Engineer
    - using STEP AP203 compliant protocol
- Tracking in BREP solids created through CAD systems was supported
  - but since Geant4 5.2 the old NIST derived STEP reader can no longer be supported.

How to import CAD geometries

- Detector geometry description should be modularized
  - By sub-detector and sub-detector components
  - Each component in a separate STEP file
- `G4AssemblyCreator` and `G4Assembly` classes from the *STEPinterface* module should be used to read a STEP file generated by a CAD system and create the assembled geometry in Geant4
  - Geometry is generated and described through BREP shapes
  - Geometry modules for each component are assembled in the user code

Importing STEP models: example -1

```
G4AssemblyCreator MyAC("tracker.stp");
  // Associate a creator to a given STEP file.
MyAC.ReadStepFile();
  // Reads the STEP file.
STEPentity* ent=0;
  // No predefined STEP entity in this example.
  // A dummy pointer is used.
MyAC.CreateG4Geometry(*ent);
  // Generates GEANT4 geometry objects.

void *pl =  MyAC.GetCreatedObject();
  // Retrieve vector of placed entities.
G4Assembly* assembly = new G4Assembly();
  // An assembly is an aggregation of placed entities.
assembly->SetPlacedVector(*(G4PlacedVector*)pl);
  // Initialise the assembly.
```

Importing STEP models:
example - 2

```
G4int solids = assembly->GetNumberOfSolids();
   // Get the total number of solids among all entities.
for(G4int c=0; c<solids; c++)
   // Generate logical volumes and placements for each solid.
{
   ps = assembly->GetPlacedSolid(c);
   G4LogicalVolume* lv =
      new G4LogicalVolume(ps->GetSolid(), Lead, "STEPlog");
   G4RotationMatrix* hr = ps->GetRotation();
   G4ThreeVector* tr = ps->GetTranslation();
   G4VPhysicalVolume* pv =
      new G4PVPlacement(hr, *tr, ps->GetSolid()->GetName(),
                        lv, experimentalHall_phys, false, c);
}
```

PART II

Electromagnetic Fields

Field Contents

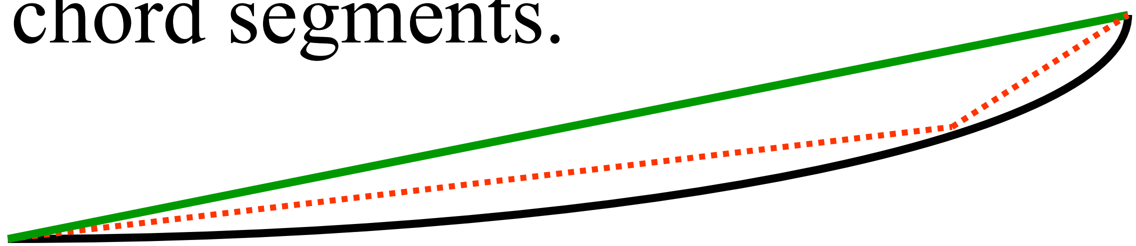
1. What is involved in propagating in a field
2. A first example
   - Defining a field in Geant4
3. More capabilities
4. Understanding and controlling the precision

Magnetic field: overview

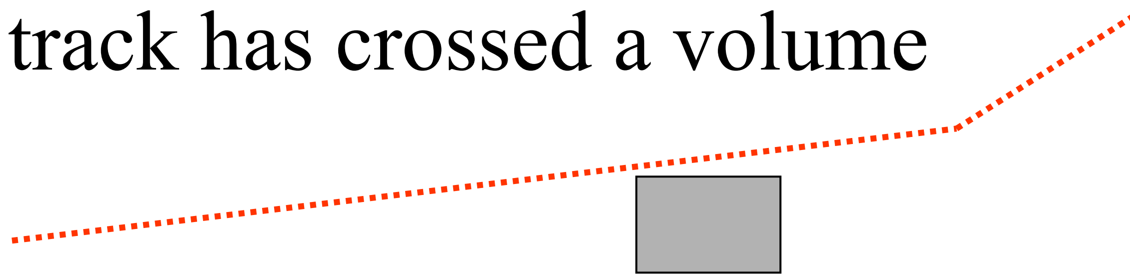
- In order to propagate a particle inside a field (e.g. magnetic, electric or both), we solve the equation of motion of the particle in the field.
- We use a Runge-Kutta method for the integration of the ordinary differential equations of motion.
  - Several Runge-Kutta 'steppers' are available.
- In specific cases other solvers can also be used:
  - In a uniform field, using the analytical solution.
  - In a nearly uniform field (BgsTransportation/future)
  - In a smooth but varying field, with new RK+helix.

Magnetic field: overview (cont)

- Using the method to calculate the track's motion in a field, Geant4 breaks up this curved path into linear chord segments.

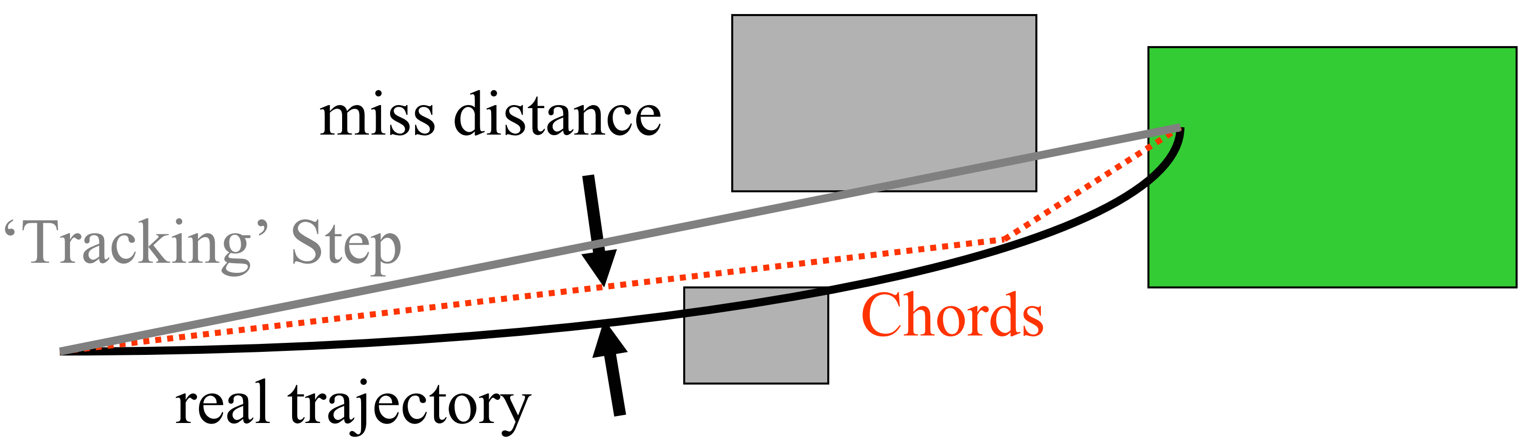


- We determine the chord segments so that they closely approximate the curved path.
- We use the chords to interrogate the Navigator, to see whether the track has crossed a volume boundary.



Stepping and accuracy

- You 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.
    - Default "miss distance" is 3 mm.
- One physics/tracking step can create several chords.
  - In some cases, one step consists of several helix turns.



Magnetic field: a first example

Part 1/2

Create your Magnetic field class

- Uniform field :
  - Use an object of the G4UniformMagField class

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

G4MagneticField* magField= new
  G4UniformMagField( G4ThreeVector(1.0*Tesla,
  0.0, 0.0 );
```

- Non-uniform field :
  - Create your own concrete class derived from G4MagneticField

Magnetic field: a first example

Tell Geant4 to use your field

Part 2/2

- Find the global Field Manager

```
G4FieldManager* globalFieldMgr=
 G4TransportationManager::
   GetTransportationManager()
                    ->GetFieldManager();
```

- Set the field for this FieldManager,

```
globalFieldMgr->SetDetectorField(magField);
```

- and create a Chord Finder.

```
globalFieldMgr->CreateChordFinder(magField);
```

In practice: exampleN04

From geant4/examples/novice/N04/src/ExN04DetectorConstruction.cc

```
G4VPhysicalVolume* ExN04DetectorConst
{
  //------------------------------------
  // Magnetic field
  //------------------------------------

  static G4bool fieldIsInitialized =
  if(!fieldIsInitialized)
  {
    ExN04Field* myField = new ExN04Fi
    G4FieldManager* fieldMgr
      = G4TransportationManager::GetT
        ->GetFieldManager();
    fieldMgr->SetDetectorField(myFiel
    fieldMgr->CreateChordFinder(myFie
    fieldIsInitialized = true;
  }
```

Beyond your first field

- Create your own field class
  - To describe your setup's EM field
- Global field and local fields
  - The world or detector field manager
  - An alternative field manager can be associated with any logical volume
    - Currently the field must accept position global coordinates and return field in global coordinates
- Customizing the field propagation classes
  - Choosing an appropriate stepper for your field
  - Setting precision parameters

Creating your own field

Create a class, with one key method – that calculates the value of the field at a Point

Point [0..2]	position
Point[3]	time

```
void ExN04Field::GetFieldValue(
  const double Point[4],
    double *field) const
{
  field[0] = 0.;
  field[1] = 0.;
  if(abs(Point[2])<zmax &&
   (sqr(Point[0])+sqr(Point[1]))<rmax_sq)
  { field[2] = Bz; }
  else
  { field[2] = 0.; }
}
```

Global and local fields

- One field manager is associated with the ‘world’
  - Set in G4TransportationManager
- Other volumes can override this
  - By associating a field manager with any logical volume
    - 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.

Precision parameters

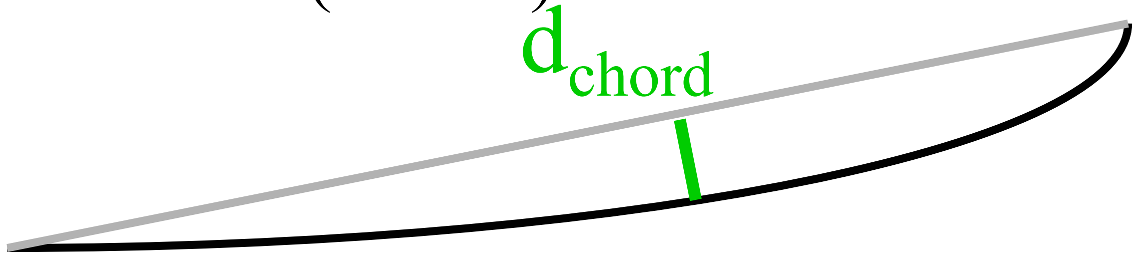
- Errors come from
  - Break-up of curved trajectory into linear chords
  - Numerical integration of equation of motion
    - or potential approximation of the path,
  - Intersection of path with volume boundary.
- Precision parameters enable the user to limit these errors and control performance.
  - The following slides attempt to explain these parameters and their effects.

Volume miss error

Due to the approximation of the curved path by linear sections (chords)

$d_{chord} < \delta_{chord}$

Parameter δ_{chord} value



- Parameter δ_{chord}
- Effect of this parameter as $\delta_{chord} \rightarrow 0$

$$s_{1step}^{propagator} \simeq (8\, \delta_{chord}\, R_{curv})^{1/2}$$

so long as $s^{propagator} < s^{phys}$ and $s^{propagator} > d_{min}^{integr}$

Integration error

Due to error in the numerical integration (of equations of motion)

Parameter(s): $\varepsilon_{integration}$

$$\max(\| \vec{\Delta r} \| / s_{step} \;,\; \|\vec{\Delta p}\| / \|\vec{p}\|) < \varepsilon_{integration}$$

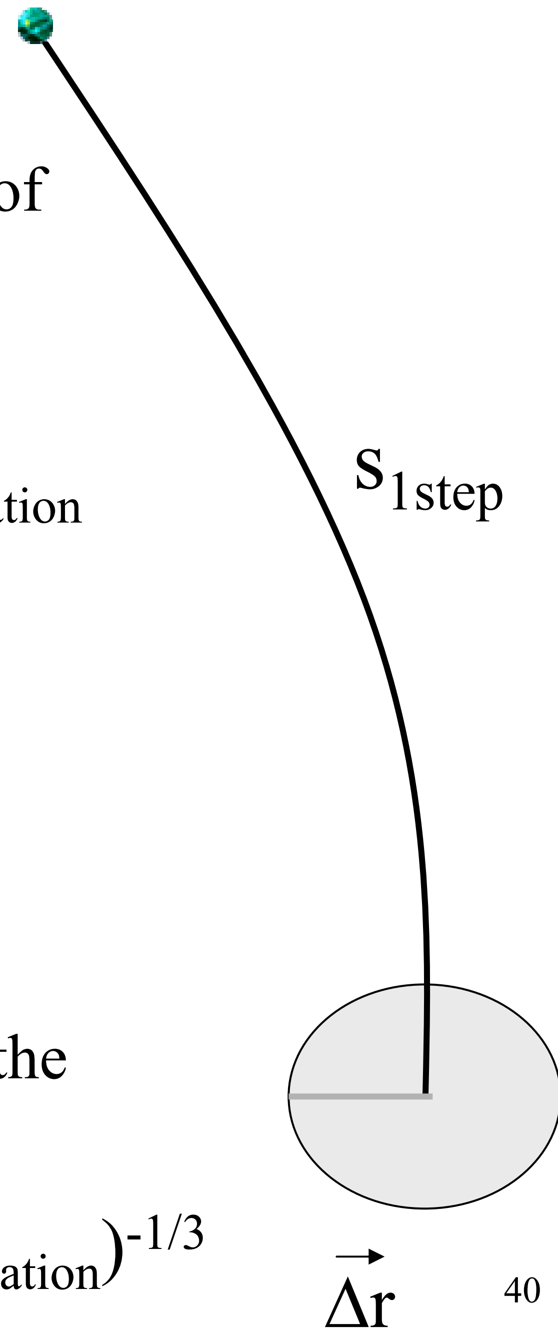
- It limits the size of the integration step.
- For ClassicalRK4 Stepper

$$s_{1step}^{integration} \sim (\varepsilon_{integration})^{1/3}$$

for small enough $\varepsilon_{integration}$

- The integration error should be influenced by the precision of the knowledge of the field (measurement or modeling).

$$N_{steps} \sim (\varepsilon_{integration})^{-1/3}$$



Integration errors (cont.)

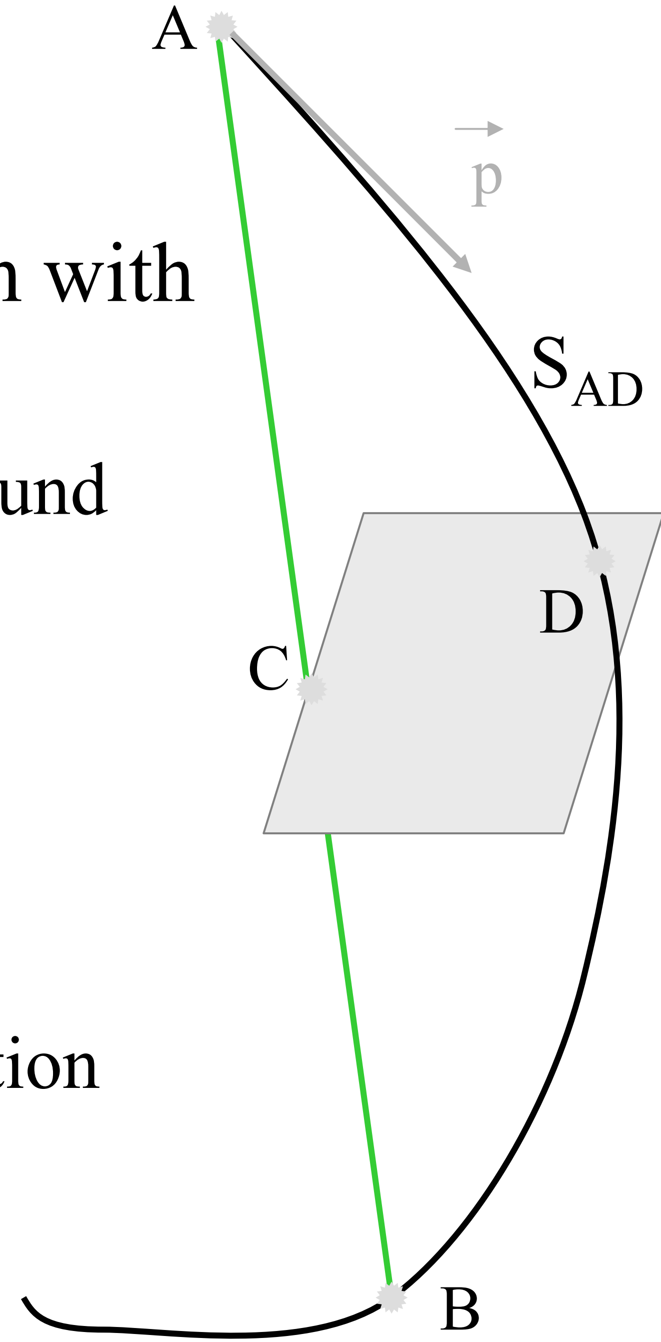
- In practice
  - $\varepsilon_{integration}$ is currently represented by 3 parameters — Defaults
  - epsilonMin, a minimum value (used for big steps) — $0.5*10^{-7}$
  - epsilonMax, a maximum value (used for small steps) — 0.05
  - DeltaOneStep, a distance error (for intermediate steps) — 0.25 mm
    - $\varepsilon_{integration} = \delta_{one\ step} / s_{physics}$
- Determining a reasonable value
  - I suggest it should be the minimum of the ratio (accuracy/distance) between sensitive components, ..
- Another parameter — Default
  - d_{min} is the minimum step of integration — 0.01 mm
    - (newly enforced in Geant4 4.0)

Intersection error

- In intersecting approximate path with volume boundary
  - In trial step AB, intersection is found with a volume at C
  - Step is broken up, choosing D, so

    $$S_{AD} = S_{AB} * |AC| \; / \; |AB|$$

  - If $|CD| < \delta_{intersection}$
    - Then C is accepted as intersection point.
  - So δ_{int} is a position error/bias

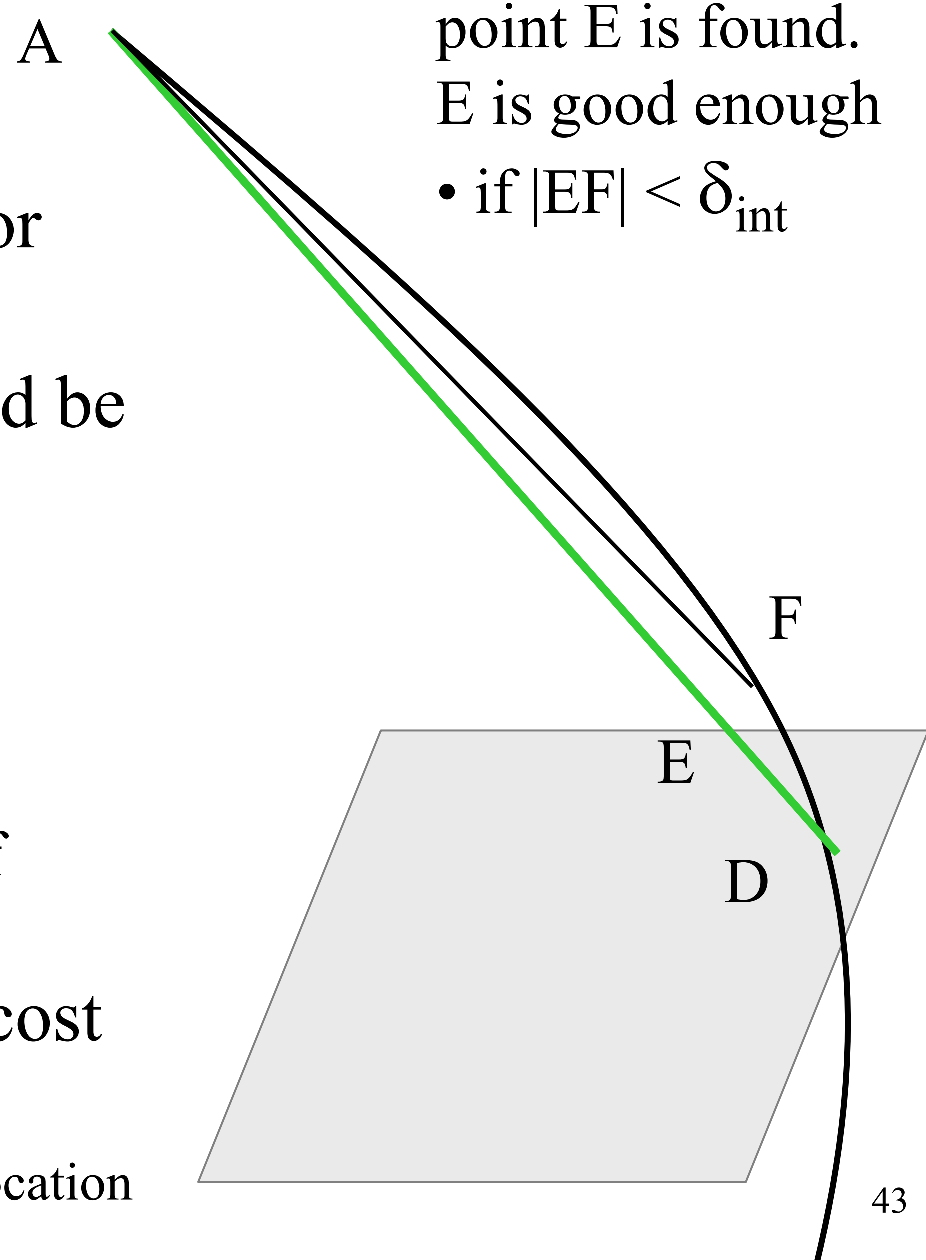


Intersection error (cont)

- So δ_{int} must be small
  - compared to tracker hit error
  - Its effect on reconstructed momentum estimates should be calculated
    - And limited to be acceptable
- Cost of small δ_{int} is less
  - than making δ_{chord} small
  - Is proportional to the number of boundary crossings – not steps.
- Quicker convergence / lower cost
  - Possible with optimization
    - adding std algorithm, as in BgsLocation

If C is rejected, a new intersection point E is found. E is good enough

- if $|EF| < \delta_{int}$



The ‘driving force’

- Distinguish cases according to the factor driving the tracking step length
  - ‘physics’, eg in dense materials
  - fine-grain geometry
- Distinguish the factor driving the propagator step length (if different)
  - Need for accuracy in ‘seeing’ volume
  - Integration inaccuracy
    - Strongly varying field

Potential Influence

G4 Safety improvement

Other Steppers, tuning d_{min}

Where to find the parameters

Parameter	**Name**	**Class**	**Default**
δ_{miss}	DeltaChord	ChordFinder	3.0 mm
d_{min}	stepMinimum	ChordFinder	0.01 mm
$\delta_{intersection}$	DeltaIntersection	FieldManager	0.10 mm
$\delta_{one\ step}$	`DeltaOneStep`	FieldManager	0.25 mm
	epsilonMin	PropagatorInField	$5\ 10^{-7}$
	epsilonMax	PropagatorInField	0.05

What if time does not change much?

- If adjusting these parameters (together) by a significant factor (10 to 100) does not produce results,
  - Then it is likely that the field propagation is not the dominant (most CPU intensive) part of your program.
  - Look into alternative measures
    - modifying the physics 'cuts' – ie production thresholds
      - To create fewer secondaries, and so track fewer particles
    - determining the number of steps of neutral vs charged particles,
      - To find whether neutrons, gammas 'dominate'
    - profiling your application
      - You can compile using G4PROFILE=yes, run your program and then use "gprof" to get an execution profile.