Version 11.0

(Electro) Magnetic Fields

John Apostolakis (CERN)

Geant4 Advanced Course at CERN

10-14 October 2022

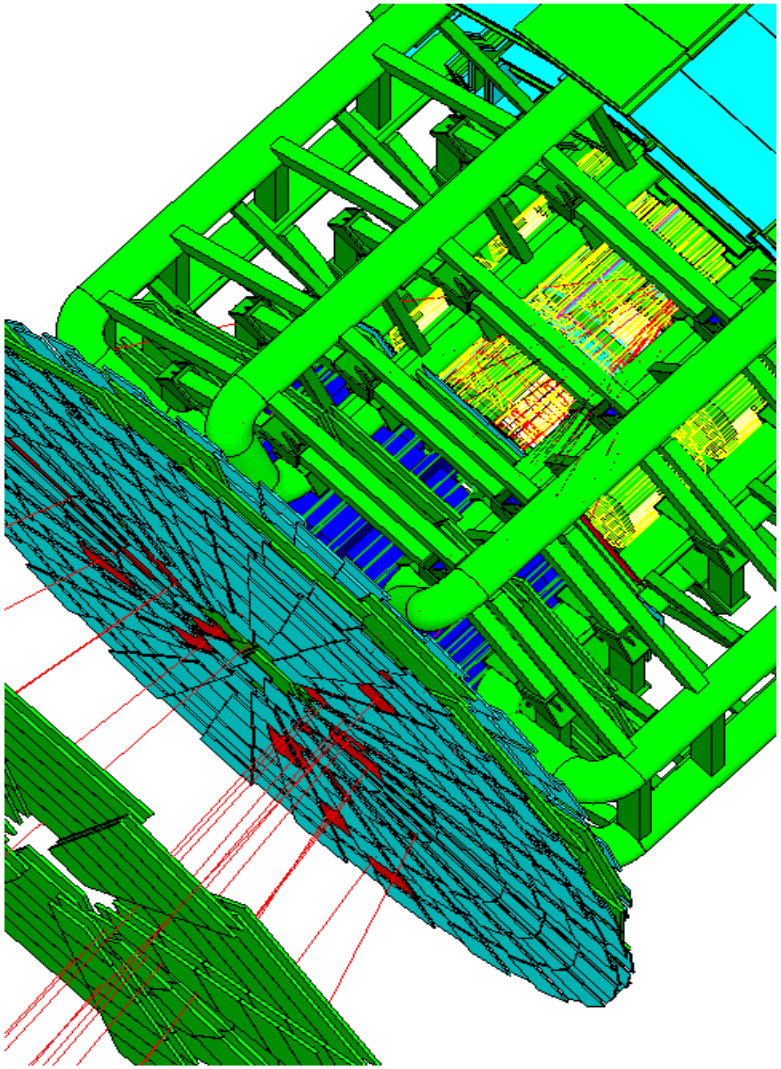
Contents

Creating a setup

- Magnetic field
- Integration of trajectories in field
- Other types of field

Taking control of integration

- Tuning accuracy
- Advanced integration methods



DEFINING A MAGNETIC FIELD

Basics 1: Defining a Magnetic field

- How to create a (magnetic) field ? Instantiate it in the *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 x,y,z position in global coordinates, Point[3] is time
    - field[0..2] are output x,y,z components of magnetic field (in G4 units)

Basics 2: How to assign a field to the whole detector

- The *global field manager* is the one associated with the 'world' volume
  - it already exists, before G4VUserDetectorConstruction is called,
  - it is created / set in G4TransportationManager.
- To associate your field with the world, you must obtain that global field manager:

```
auto tm = G4TransportationManager::GetTransportationManager();

G4Fieldmanager* globalFieldManager = tm->GetFieldManager();
```

- Then you assign the field to it

```
G4Field* field= new G4UniformField(…); // B/E or other field
globalFieldManager->SetDetectorField(field);
```

- The global field manager can also be assigned directly to the world volume if obtain in a different way. E.g.

```
G4VPhysicalVolume* fMyWorld; // In class declaration, e.g. MyDetectorConstruction.hh
fMyWorld = new … ; // In the Construct() method
G4Fieldmanager* globalfieldManager = fMyWorld->GetFieldManager();
```

Note: In an (advanced) use case with parallel worlds only the primary geometry (the 'mass' geometry in which nearly all materials are assigned) matters - the *global field manager* and all others are associated only with this geometry.

Refinement: Region and local fields

- Other volumes can override a global field
  - An alternative field manager can be associated with any logical volume – or a 'region'
    - The field must accept position in global coordinates and return the value of the field in global coordinates

```
auto localFieldManager = new G4FieldManager(myField);
Region->setFieldManager(localFieldManager);
```

The region Field Manager overrides the global one (if any.)

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

The logical Volume's Field Manager overrides the region and the global (if any.)

Note that the assignment affects also sub-volumes contained in **`logVolume`**:

- By default only sub-volumes that do not yet have a field manager.
- Using '**`true`**' for the second argument asks it to push the field to all the sub-volumes, even if a daughter volume has its own field manager.

How to Nullify the field in a volume

Let's clarify this with an example.

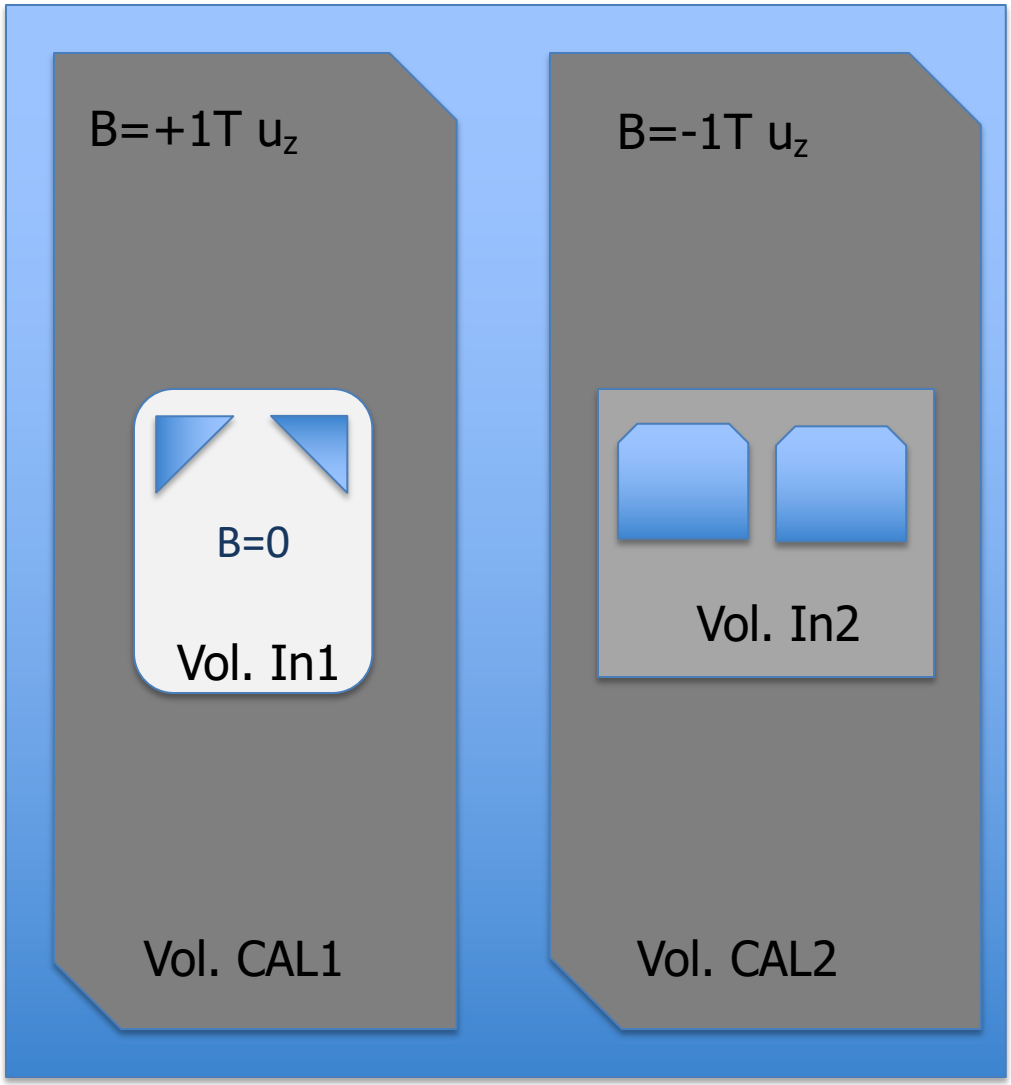
We created a geometry with 2 Calorimeters in the world volume, CAL1 and CAL2.

To create a "B=0" region or volume inside the volume "IN1" in "CAL1", we must created a

```
G4MagneticField* Bfield= nullptr;
G4FieldManager zeroFieldMgr= new
G4FieldManager( Bfield );
```

Then we use this to assign a zero field to Volume "In1":

```
volCal1->SetFieldManager(
zeroFieldMgr );
```



FIELD PROPAGATION

Choosing an appropriate integration method for your field

Setting precision parameters

Basics 3: Ensuring the Magnetic field deflects charged particles

In the DetectorConstruction's *ConstructSDandField*() method, after creating a field

```
G4MagneticField* fMyField = new MyMagneticField();
G4Fieldmanager* fieldManager = new G4FieldManager();
fieldManager->SetDetectorField(fMyField);
```

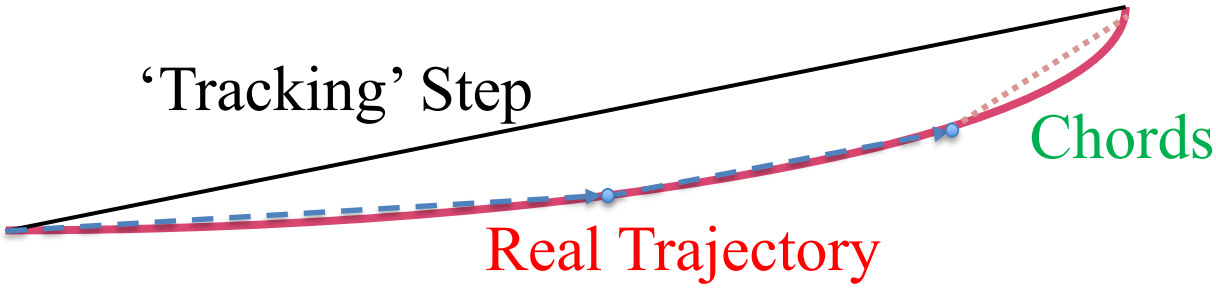
a user must create an integration method. There is a very easy way – only for pure magnetic fields:

```
fieldManager->CreateChordFinder(fMyField);//Use default method
G4bool pushToContained = true;
myLogicalVol->SetFieldManager(fieldManager, pushToContained);
// This overwrites existing field managers in daughter volumes
// Register the field and its manager for deleting at the end
G4AutoDelete::Register(fMField);
G4AutoDelete::Register(fieldManager);
```

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

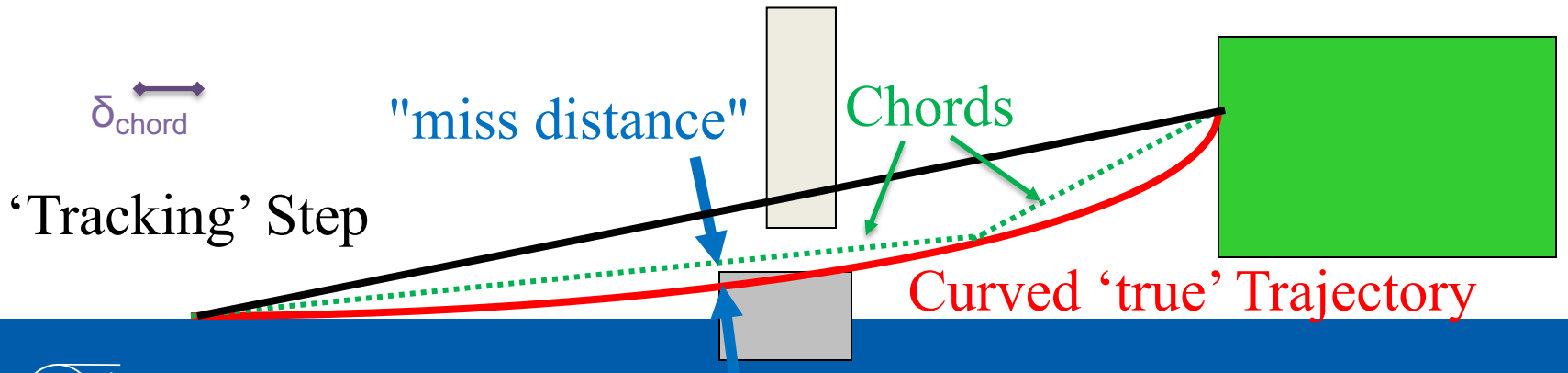
Field integration: overview of Geant4 approach

- To propagate a particle in an external field (magnetic, electric, both or other), we integrate numerically its equation of motion.
- We use Runge-Kutta integration method for the ordinary differential equations
  - Several Runge-Kutta ‘steppers’ are available.
- In specific cases other integration methods can also be used:
  - In a uniform field, using the analytical solution – a helix (*G4ExactHelix*).
  - In a smooth but varying field, with RK+helix.
  - An alternative multi-step integration method, Bulirsch-Stoer.
- As it calculates the track's motion in a field, Geant4 breaks up its curved path into linear chord segments.
  - Choosing chord segments so that they approximate the curved path within a given tolerance.

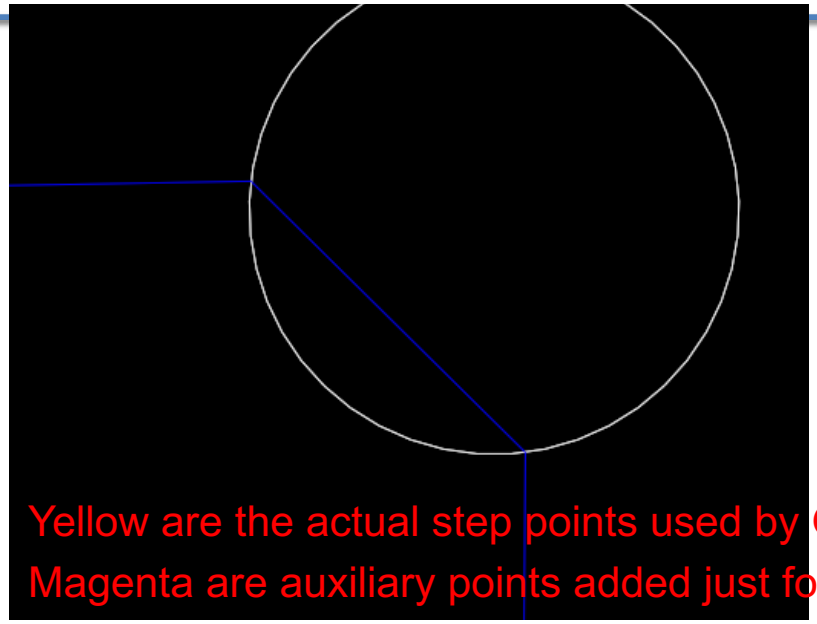


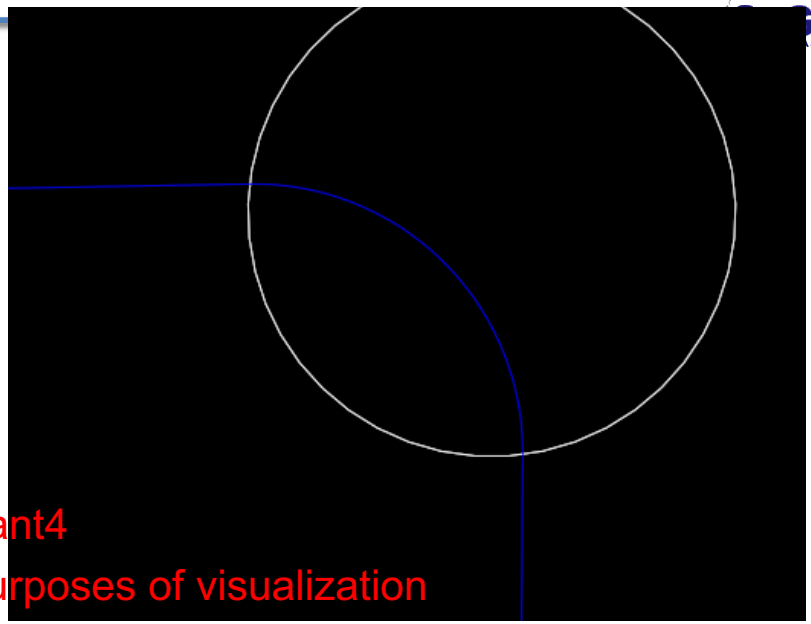
Tracking in field

- The chords are used to interrogate the G4Navigator, to check whether/where a track has crossed a volume boundary.
- One physics/tracking step can create several chords.
  - In gases or vacuum, a ‘physics’ step can span several helix turns.
- The user controls the accuracy of the volume intersection,
  - By setting a parameter δ_{chord} to limit the “miss distance” It is the
    - maximum acceptable error in approximating the curved track by chords,
    - maximum depth inside a volume that a curved track could enter and yet the volume is still missed (not crossed by the series of chords.)
  - It is quite expensive in CPU performance to set too small a “miss distance”.



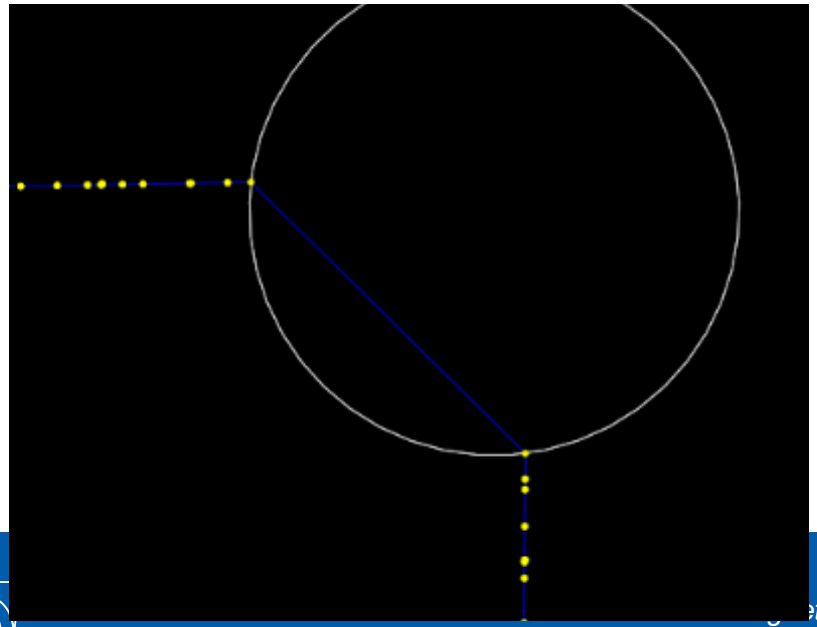
Regular versus Smooth Trajectory

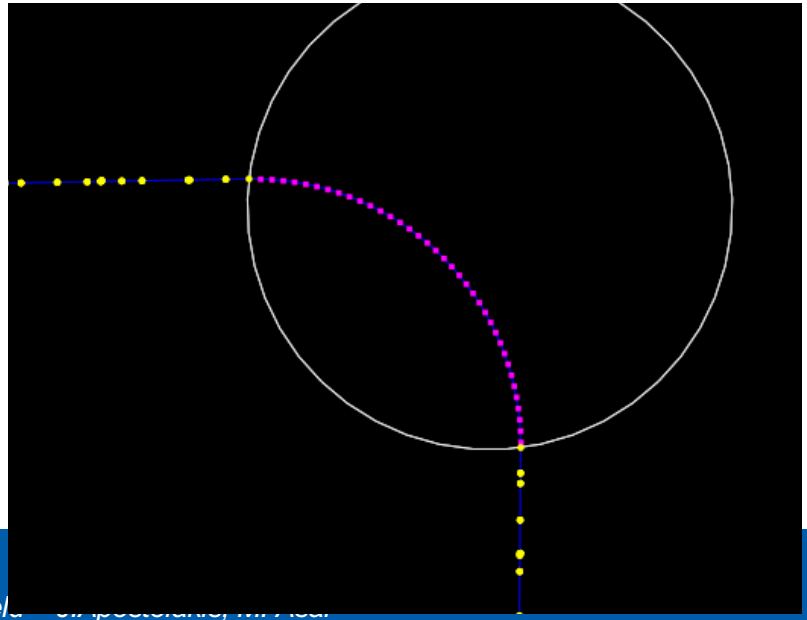




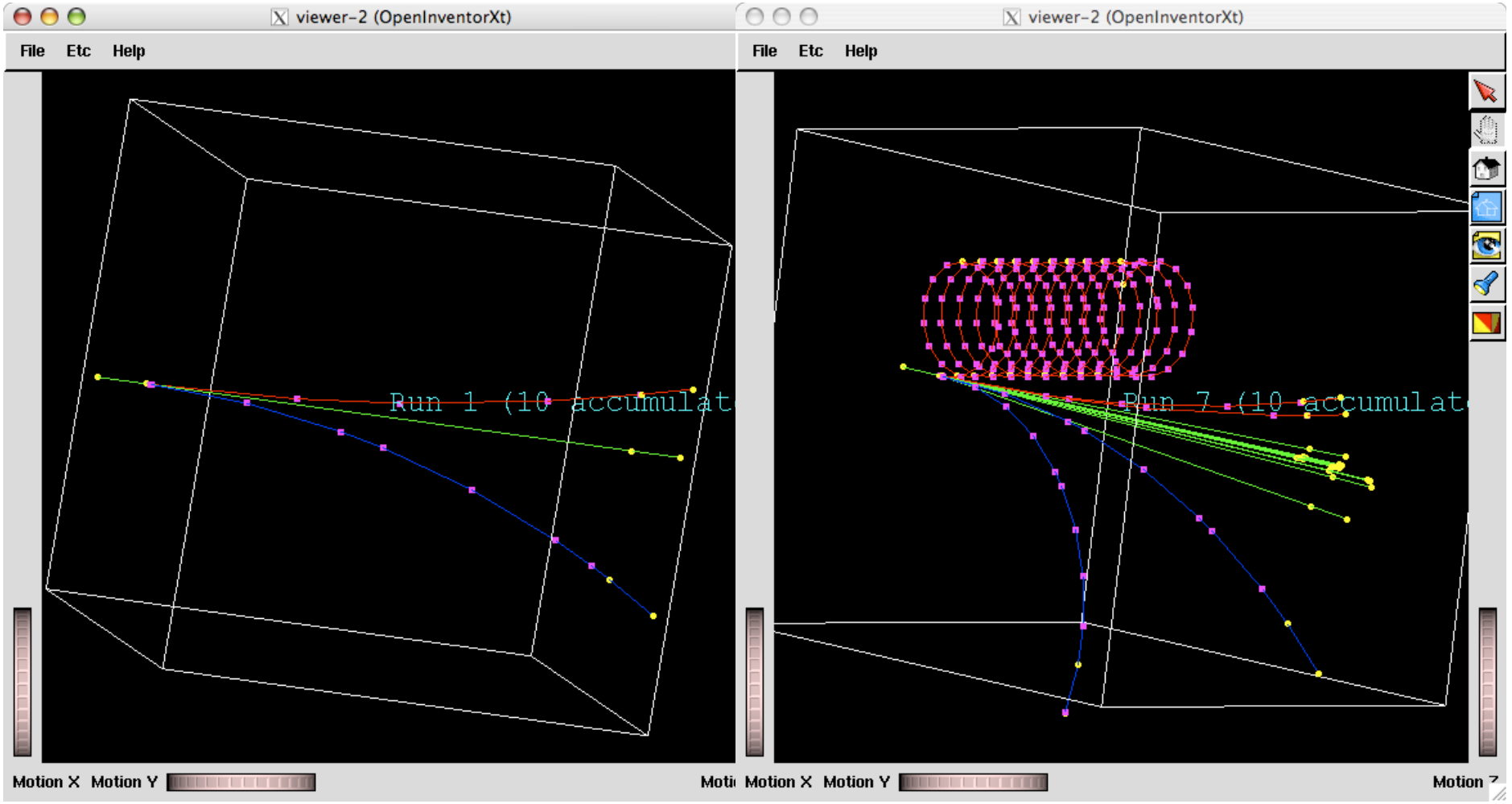
Yellow are the actual step points used by Geant4

Magenta are auxiliary points added just for purposes of visualization





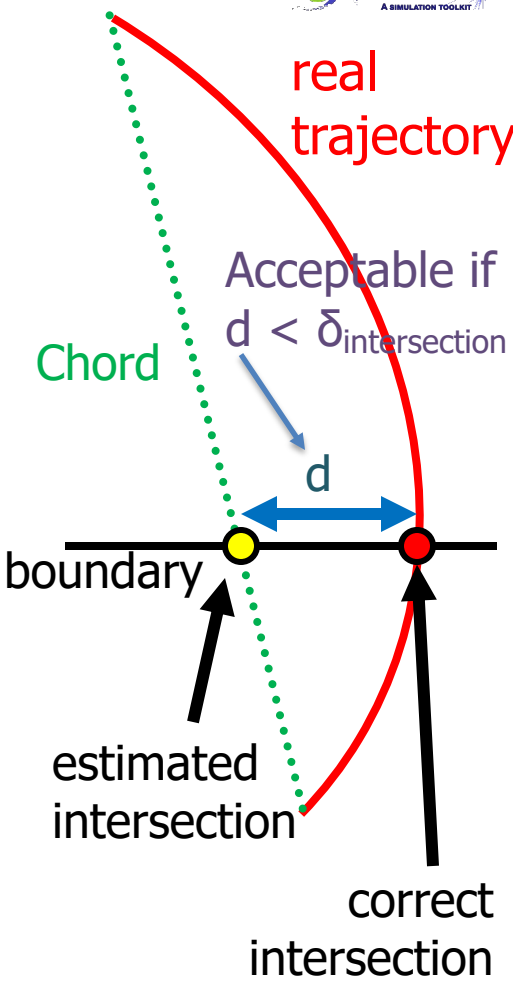
Smooth Trajectory Makes Big Difference for Trajectories that Loop in a Magnetic Field



- Yellow dots are the actual step points used by Geant4
- Magenta dots are auxiliary points added just for the purpose of visualization

Tunable parameters

- 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.
  - **Important**: it is used to **limit the bias** that our algorithm (for boundary crossing in a field) exhibits: the intersection point is always on the 'inside' of the curve.
  - Set its **value** much **smaller** than your acceptable error, to limit the cumulative effect of this bias (after the total number of volume crossings in the track's path.)



Tunable parameters

- The most important accuracy parameter is the maximum relative tolerance ε_{max} for the integration error
  - ε_{max} limits the estimated error for large steps: $|\Delta x| < \varepsilon_{max}\, s$ and $|\Delta p| < \varepsilon_{max}\, |p|$
- The "delta one step" parameter is accuracy for endpoint of integration steps that do **not intersect** a volume boundary.
  - It also limits on the estimated error of the endpoint of each physics step (essentially it is < 1000 $\delta_{1\ step}$.)
  - Values of $\delta_{intersection}$ and $\delta_{1\ step}$ should be within one order of magnitude.
- These tunable parameters can be set by

```
ptrChordFinder->SetDeltaChord( missDistance );
ptrFieldManager->SetDeltaIntersection( deltaIntersection );
ptrFieldManager->SetDeltaOneStep( deltaOneStep );
ptrFieldManager->SetEpsilonMax( epsilonMax );
ptrFieldManager->SetEpsilonMin( 0.1 * epsilonMax );
```

- Further details are described in Section 4.3 (Electromagnetic Field) of the Geant4 Application Developers Guide.

ADDITIONAL FIELDS & INTEGRATION METHODS

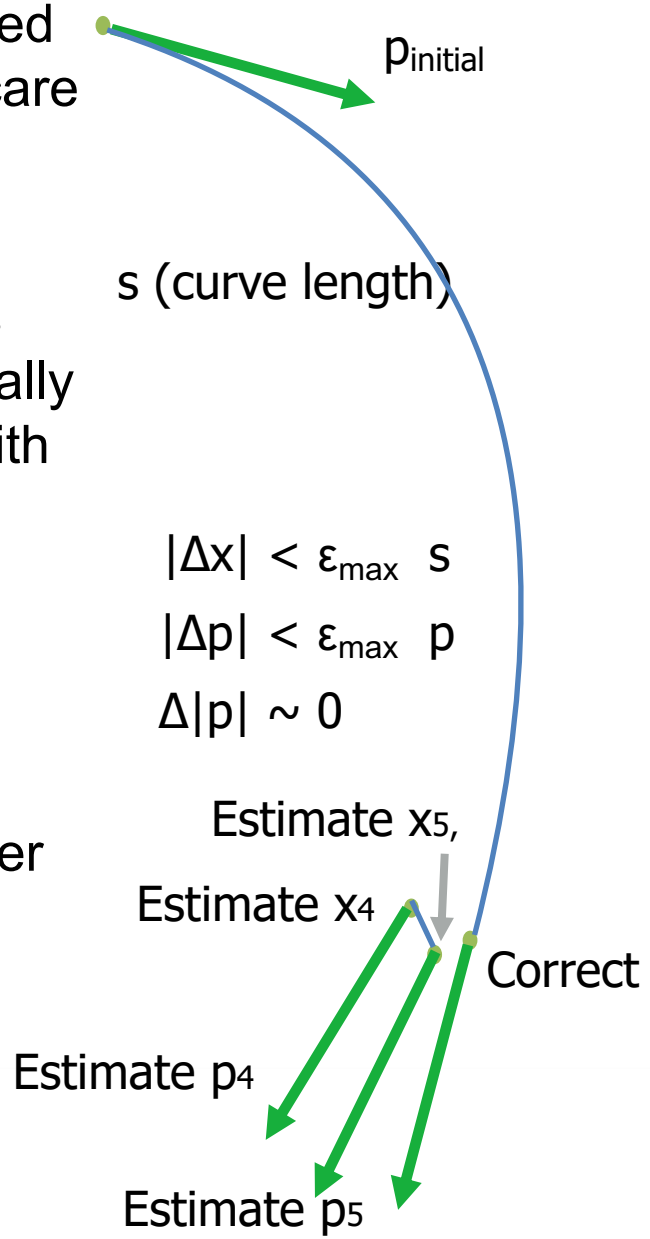
Field integration – usual and custom methods
Alternative types of field

Customizing field integration

- Runge-Kutta (RK) integration is used to compute the motion of a charged track in a any type of field: magnetic, electric, combined EM, gravitational or a mix.
- There are two kinds of RK steppers are available in Geant4:
  - General-purpose steppers, applicable to any field type, usable for any ODE.
  - Specialized steppers, applicable only to charged particle motion in **pure magnetic** fields.
- RK steppers related to the order of the Taylor expansion from which they derive:
  - low 1st, 2nd or 3rd order are considered 'low-order' steppers – with one field evaluation per stage (i.e. 1 initial + 2 other evaluations for 3rd order method).
  - 4th order (4 evals/step) & 5th order (6-7 evals/step) are the typical 'sweet' spot
  - higher order require many more stages (evaluations) per step – their increased accuracy is relevant for very specialized applications.
- An integration method must estimate both an end state (position, momentum, maybe polarisation) and an integration error for each state variable
  - Old (pre 1967) 'simple' methods estimated error by having the step

Integrating efficiently

- Given a detector's field B(x,y,z) [or B+E] we need to integrate the trajectory of each track, taking care
  - to stay within the relative accuracy ε_{max}
  - to be fast – in particular using as few calls as possible to the field evaluation method (typically expensive in arithmetic operations / time – with an interpolation or function evaluation)
- Typically choose Runge-Kutta methods
  - No memory / history needed – it "self starts"
  - Adjusts easily to change(s) of momentum after (frequent) collisions / interactions
  - Ability to adjust step size



Customizing field integration

- Runge-Kutta (RK) integration is used to compute the motion of a charged track in a any type of field: magnetic, electric, combined EM, gravitational or a mix. Geant4 offers
  - Many general-purpose steppers that can be applied for any equation / field.
  - Some specialized steppers, applicable only to pure magnetic fields.
- Default in G4 is the general purpose G4DormandPrince745 an embedded 4^{th}-5^{th} order RK stepper. (Embedded = compares 4^{th} & 5^{th} order to estimate error.)
  - It typically uses 6 field evaluation per integration, as it provides the derivative at the endpoint (avoids need to calculate it at the start of the next step.)
  - Earlier Geant4 versions (<10.4) had G4ClassicalRK4 as default – robust but needs 11 field evaluations per integration step.
- If the field is unusual, e.g. very rough or smooth, explore different lower or higher order steppers to seek results of same quality using fewer computing cycles.
  - High order (6+) steppers for high accuracy in very smooth fields
  - Low order (2-3) steppers for badly measured or poorly interpolated fields.

How does an explicit Runge-Kutta method work?

- "Integrate" $dy/dx = F(x, y)$ from x_0 to x_0+h
- Uses evaluations of $F(x, y)$
  - $f_i = F(x_0 + a_i h , y_0 + h \Sigma_{j<i} b_{ij} f_j)$
  - $y_{estim}(x_0 + h) = \Sigma_i c_i f_i$
- Each method has a 'Buther tableau' made up of the coefficients a_i, b_{ij}, c_i, c'_i
- Key Parameters of an RK method:
  - Number of 'stages' = number of evaluations of the derivative F()
  - 'Order' N: the expected scaling of the errors ~ h^N
  - Embedded method = 2nd 'line' to estimate error

$f_1 = F(x_0 , y_0)$

$f_2 = F(x_0 + a_2 h , y_0 + h b_{21} f_1)$

$f_3 = F(x_0 + a_3 h , y_0 + h b_{31} f_1 + h b_{32} f_2)$

a_i | b_{ij}

0			
$\frac{1}{2}$	$\frac{1}{2}$		
$\frac{3}{4}$	0	$\frac{3}{4}$	
c_j 1	$\frac{2}{9}$	$\frac{1}{3}$	$\frac{4}{9}$

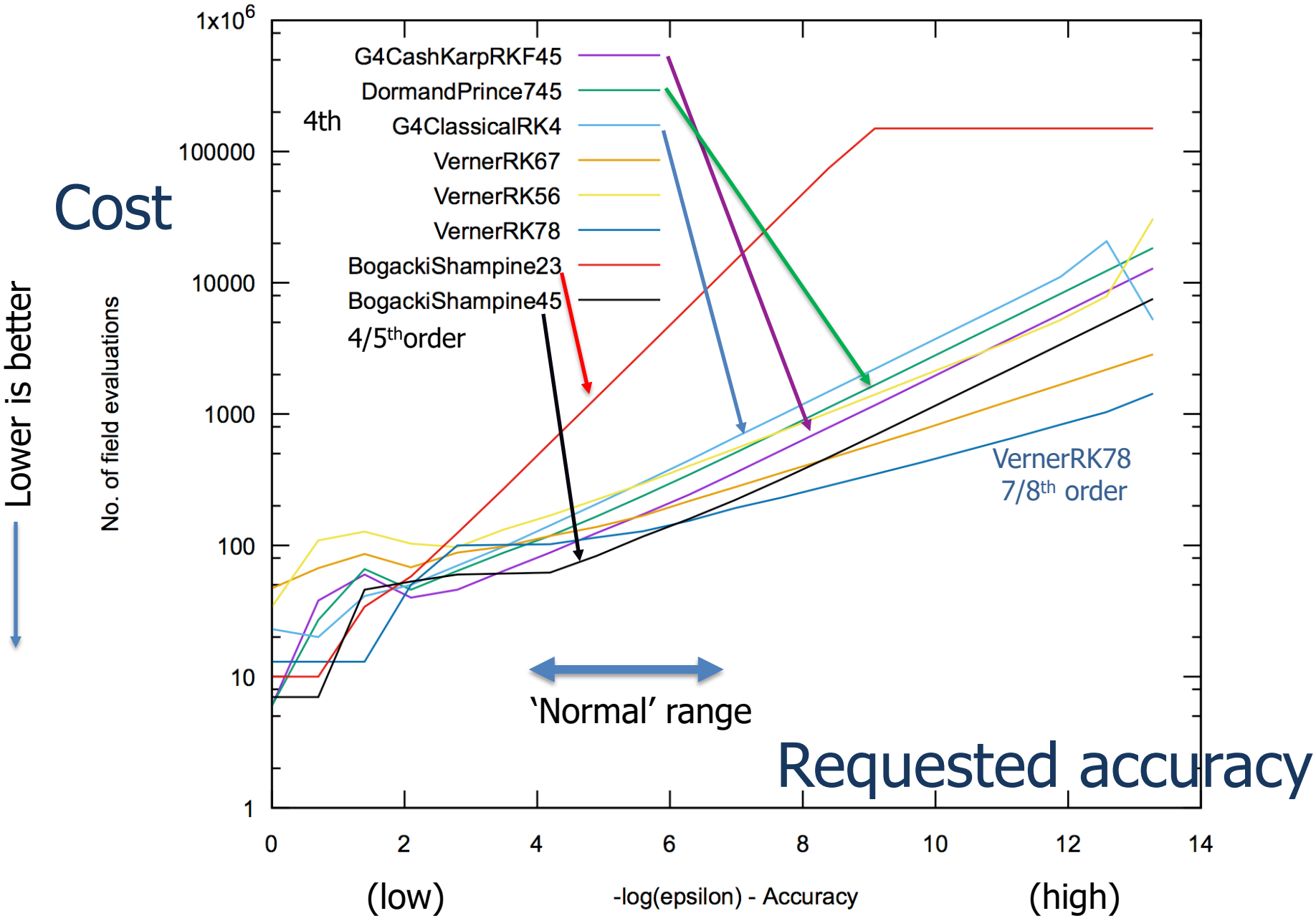
$y_{RBS3}=2f_1/9 + f_2/3 + 4f_3/9$

c'_j | $\frac{7}{24}$ | $\frac{1}{4}$ | $\frac{1}{3}$ | $\frac{1}{8}$ |

$y'(x_0 +h) = \Sigma_i c'_i f_i$

$\Delta y = \Sigma_i (c'_i - c_i) f_i$

Thanks to Somanth Banerjee (Google Summer of Code student 2015)



Alternative Runge-Kutta steppers – some context

- Embedded methods (invented by Felhberg) provide built-in error estimates by comparing estimators of 2 different orders (that share evaluation points!)
  - G4 Dormand Prince 745 (DoPri5), is the most widely used method provides stability and performance. It's the default in MATLAB, GNU Octave ("ode45"), and many other packages/applications.
    - It evaluates the field values and derivatives at the final point (with p_{final}), called FSAL, ready for use in next step (→ 1 less evaluation)
  - Cash-Karp (1990) – early method with six stages = 6 evaluations of derivative/field *(and could abort early – not used.) . Not FSAL, no interpolation*
  - Alternatives 5th order FSAL methods (with interpolation formulas) to consider:
    - G4Bogacki-Shampine45
    - G4TsitourasRK45

FSAL = First Same As Last

Specialised Runge-Kutta and 'hybrid' steppers

- Specialised steppers for pure magnetic fields:
  - Helix (***G4ExactHelix***) - for constant field
  - AtlasRK4/NystromRK4: 3 field evaluations + evaluation of error using numerical estimate of 4th derivative
  - Experimental hybrid Helix / RK methods that use helix as baseline (G4HelixSimpleRunge)
- 'Classical RK4' = was default in
  - It was the original 4th order method
  - Needed 11 evaluations – its error estimate comes from breaking step in two => 1(initial)+10 extra evaluations per step
  - General, robust, and expensive in CPU cycles.
- Lower order RK methods for short steps, and/or lower accuracy

FSAL = First Same As Last

Choices of Runge-Kutta methods (steppers)

- The default is the general purpose G4DormandPrinceRKF45 an embedded 4^{th}-5^{th} order RK stepper. (Embedded = compares 4^{th} & 5^{th} order to estimate error.)
  - If the field is very smooth, you may consider higher order steppers
  - of most potential interest in large volumes filed with gas or vacuum.
- If the field is rough, 3^{rd} order steppers could obtain the results of same quality using fewer computing cycles
  - 3^{rd} order stepper(s): G4SimpleHeum, G4BogackiShampine23 (FSAL)
- For reasonably smooth (or not very rough) fields, the choice between 3^{rd} , 4^{th} or 5^{th} order steppers should be made by trial and error.
- For the 'roughest' field map, consider a robust 2^{nd} order RK (G4SimpleRunga)
- The less smooth the field is, we would expect that a lower the order of the stepper would be more robust (but no lower than 2^{nd} order.)
- However the relative performance depends on many factors, and benchmarking is recommended to identify the best performing stepper.

Customizing field integration

- Trying a few different types of steppers for a particular field or application is suggested, if maximum performance is a goal.
  - What is the most performant option may also be different in different regions – e.g. depending on whether the field is varying greatly.
- Specialized steppers for pure magnetic fields are available. Some assume that a local trajectory in a slowly varying field will resemble 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
  - Suggested are G4HelixSimpleRunge and G4HelixSimpleHeum
- To change the stepper reliably you must create a new integration driver for it

```
auto driver= new G4IntegrationDriver(…);
theChordFinder->SetIntegrationDriver(driver);
```

  For the full code see the next slide (new)

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

Creating a different Stepper - and support classes

- **To exercise full control** you can choose to create your chosen stepper and driver
  - This 'chain' of classes also must be created when using fields which are not pure magnetic fields (next slide)
- First the header files and the field

```
#include "G4UniformMagneticField.hh"
#include "G4DormandPrince745.hh"
#include " G4IntegrationDriver.hh"
#include "G4ChordFinder.hh"
// #include  "G4MagIntegratorDriver.hh"  // - for G4MagInt_Driver
G4MagneticField* Bfield= new G4UniformMagneticField( G4ThreeVector(0., 0., 1.0e5*kilogauss) );
```

- Next the *equation of motion* – and the class for G4MagneticField is **G4Mag_UsualEqRhs**

```
auto equation = new G4Mag_UsualEqRhs(Bfield);
```

- The stepper:

```
const int nvar=6;
auto stepper = new G4DormandPrince745( equation, nvar );  // the default is 6 already: x,y,z, px,py,pz
```

- Now we create the integration driver, which manages the error control:

```
auto integrDriver= new G4IntegrationDriver( fMinStep, stepper, nvar );  // New way
// = new G4MagInt_Driver( fMinStep, stepper, stepper->GetNumberOfVariables() );  // Old way
```

- Finally, we can create the chord finder, and set the field manager

```
G4FieldManager* fieldManager = G4TransportationManager::GetTransportationManager()-> GetFieldManager();  // field
    manager for 'world' volume
fieldManager->SetDetectorField( Bfield );
fieldManager->SetChordFinder( new G4ChordFinder(integrDriver) );
```

- *Note: A user-defined field type can be time-dependent (its value can change with time). In GetFieldValue(G4double posTim[4], G4double fieldVal[]) the parameter posTim[3] is time*

Other types of field – using an Electric or combined E-B field

- For pure electric field, Geant4 has **G4ElectricField** (base) and the simple **G4UniformElectricField** (concrete) classes.
- **G4ElectroMagneticField** is the base class for combined electro-magnetic fields.
  - the *equation of motion* class for it is **G4EqMagElectricField**
- An example:

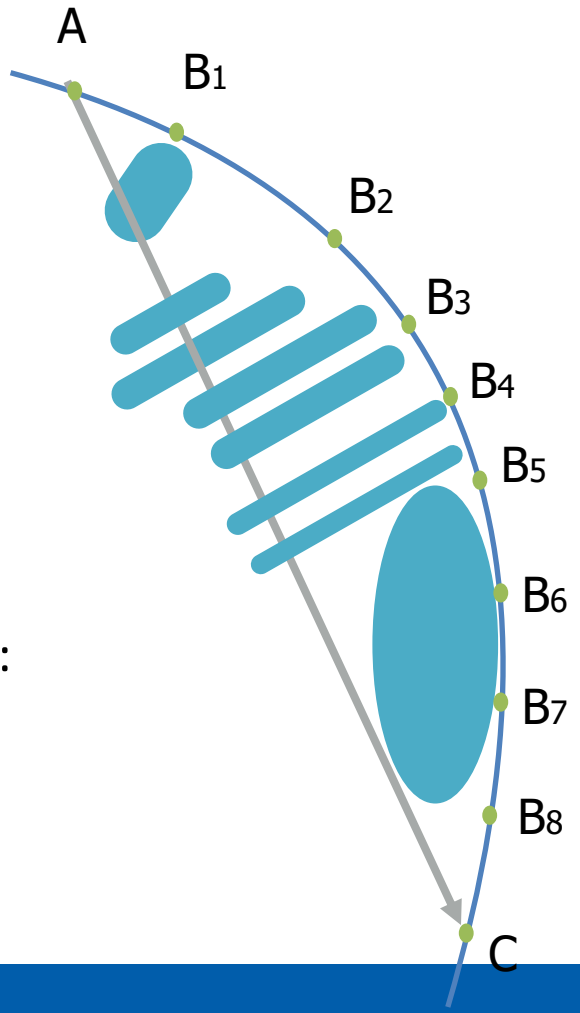
```
G4ElectricField* EMfield
    = new G4UniformElectricField( G4ThreeVector(0., 1.0e5*kilovolt/cm, 0.) );
auto equation = new G4EqMagElectricField(EMfield);
const int nvar=6;
auto stepper = new G4DormandPrince745( equation, nvar );
G4FieldManager* fieldManager = G4TransportationManager::GetTransportationManager()->
    GetFieldManager();
fieldManager->SetDetectorField( EMfield );
auto integrDriver= new G4IntegrationDriver( fMinStep, stepper, nvar );  // New way
// = new G4MagInt_Driver( fMinStep, stepper, stepper->GetNumberOfVariables() );  // Old way
G4ChordFinder* chordFinder = new G4ChordFinder(integrDriver);
fieldManager->SetChordFinder( chordFinder );
```

- *Notes:*
  1. *In a combined EM field the return values of the GetFieldValue following the convention that fieldVal[0] to [2] are B_x, B_y, B_z and fieldVal[3] to [5] are E_x, E_y, E_z*
  2. *A user can create their own type of field, inheriting from G4VField, and must create a corresponding Equation of Motion class (that inherits from G4EqRhs)*

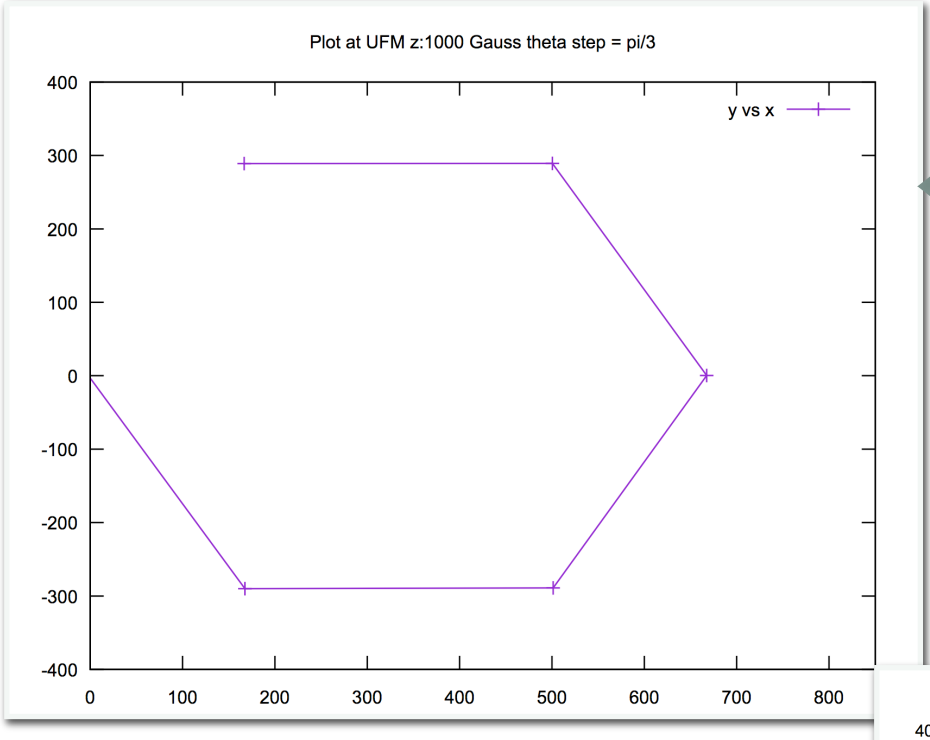
NEWEST METHODS

New RK methods - Interpolation

- Selected RK methods offer capability of estimating any intermediate point given its 'distance' along the curve
  - One-time cost of a few extra field evaluations
- Reduced cost of evaluating intermediate points (vs new integration)
  - Enable faster location of intersection point with surface boundary
- Enabled using a new type of (G4V)IntegrationDriver: **G4InterpolationDriver**



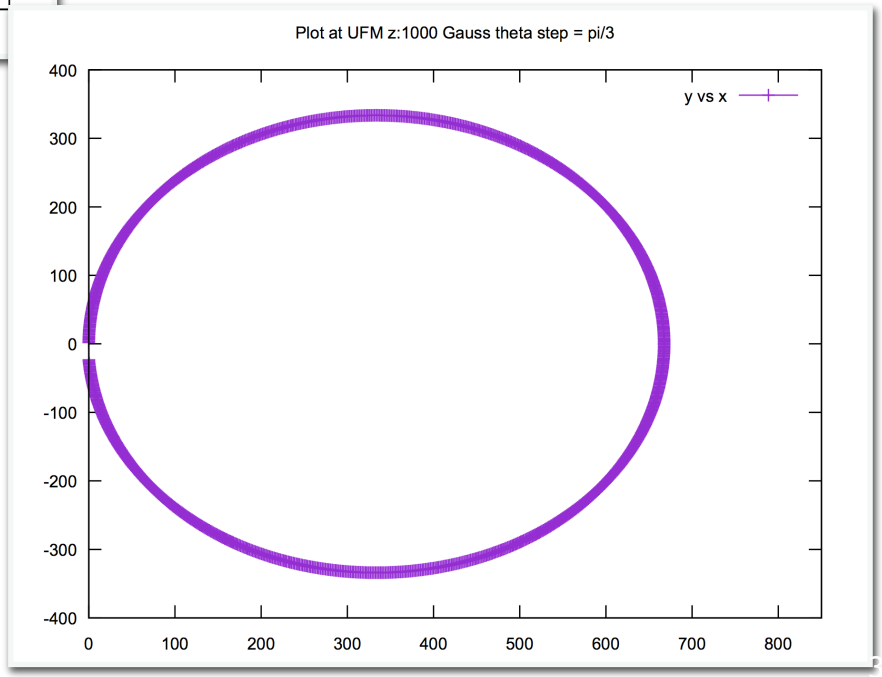
Somanth Banerjee (GSoC 2015)



6 computed points in a circular trajectory

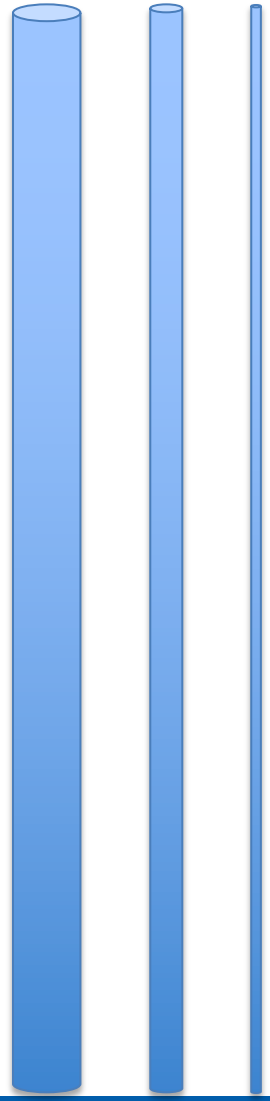
Calling 100 times between each pair of points, to give "Dense" output

Interpolated result



Dealing with looping particles (1/3)

- In volumes with magnetic field and vacuum or gas as material some tracks can need a large (>10^4) number of integration steps
  - This can be a major sink of CPU time
- So within a physics step there is a limit of 1,000 integration steps (tunable.)
  - When a track reaches this limit, it is marked as 'looping'. It is now a candidate for being killed.
- Geant4 will kill particles found to be looping:
  - If $E < E_{warning}$ a track is killed immediately without warning
  - If $E_{warning} < E < E_{important}$ the track is killed, and a small warning is printed (to cout in Geant4 versions < 10.5)
  - If $E > E_{important}$ the track is given an extra number of chances (by default 10) before being killed.
- Their values can be changed using methods of G4Transportation. Default values are (chosen for collider HEP experiments):
  - $E_{warning}$ = 100 MeV
  - $E_{important}$ =250 MeV



Dealing with looping particles (2/3)

- These values can be changed using G4Transportation's methods:

```
SetThresholdWarningEnergy( G4double );
SetThresholdImportantEnergy( G4double );
SetThresholdTrials( G4int maxTrials );
```

First you must find the G4Transportation process

```
#include "G4Proton.hh"
#include "G4ParticleDefinition.hh"
#include "G4Transportation.hh"

G4ParticleDefinition particleDef= G4Proton::G4Proton();
G4VProcess procTr = particleDef->GetProcessManager()
                    ->GetProcess("Transportation");
G4Transportation* protonTransport =
   dynamic_cast<G4Transportation*>(procTr);
```

Then you can change its properties:

```
if( protonTransport )
  protonTransport->SetThresholdWarningEnergy(10.0*CLHEP::keV);
```

Dealing with looping particles (3/3)

- In Geant4 10.5 several changes were implemented:
  - only stable particles are killed
  - each particle with energy above the warning energy which is killed **generates a detailed warning** (using G4Exception) with location, volume, material, particle momentum and energy.
  - for the first 5 tracks killed a detailed description is printed that describes the criteria and parameters used to decide what tracks are killed, and guidance.
- Guidance regarding how to 'save' tracks:
  - by changing the values of thresholds or
  - by adopting different integration methods.

Examples that demonstrate these

- examples/basic/B2
  - Use G4GlobalMagFieldMessenger to create a global, uniform magnetic field
- examples/basic/B5
  - Creating a custom magnetic field & assigning it to a field
- examples/extended/field
  - field01: exploring integration methods
  - field02: a combined E+B field : Electric+Magnetic
  - field03: local field defined in a volume
  - field04: overlapping field elements
  - field05: tracking of polarization and spin-frozen condition
  - field06: ultra cold neutrons and gravity field
  - Blinetracer: visualize B-field lines

Takeaways

- Runge-Kutta (RK) integration is used to compute the motion of a charged track in a any type of field: magnetic, electric, combined EM, gravitational or a mix.
  - Many general steppers are available applicable to any equation / field
  - A few specialised steppers can be used only for pure magnetic fields.
- Default is the general purpose G4DormandPrinceRKF45 stepper
  - is an embedded 4^{th}-5^{th} order & uses 6 (extra) field evaluations per step
  - it provides the end derivative ('FSAL') and provides interpolation.
- If the field has very rough or smooth, consider lower or higher order steppers
  - Expect same quality using fewer computing cycles,
  - Try a different stepper (or two) to see whether it improves CPU time.
- RK steppers with interpolation will reduce the number of field calls for each intersection boundary
  - Currently optional, with plans to introduce them as default in 2019.
- Different types of fields available, and user can create their own
  - A field must be accompanied by its corresponding equation of motion.

FURTHER INFORMATION

Alternative to RK, and 'improved' RK methods with Interpolation

The Bulirsch Stoer method is a multi-step method, alternative to Runge-Kutta:

```
G4BulirschStoer* pBSstepper =
        new G4BulirschStoer( fEquation, nVar, epsilon );
auto pDriver = new G4IntegrationDriver<G4BulirschStoer>( stepMinimum,
                                                         pBSstepper, nVar );
```

A variety of promising Runge-Kutta methods, seeking improvements in

- efficiency - accuracy of error estimation (fewer 'bad' steps)
- accuracy of solution - extending it further (larger steps)
- speed - reduce number of evaluations of derivative/field, e.g. reuse derivatives
- **interpolation**: obtain the state (x,p,..) at any intermediate point

Using the latest developments – FSAL and Interpolation

These promising Runge-Kutta methods can be explicitly created as follows:

- Interpolation via the G4InterpolationDriver class and a compatible stepper:

```
#include "G4InterpolationDriver.hh"
…
auto stepper= new G4DormandPrince745( equation, nvar );
auto interpolationDriver= new G4InterpolationDriver(fMinStep, stepper, nvar );
```

  Note that this is now used default for pure magnetic fields.

- The FSAL property of a stepper signifies that the tableau/method calculates the derivative at the final point – avoiding the need to do this at the start of the next step. It is available in a few additional steppers for use via an alternative 'Driver' class:

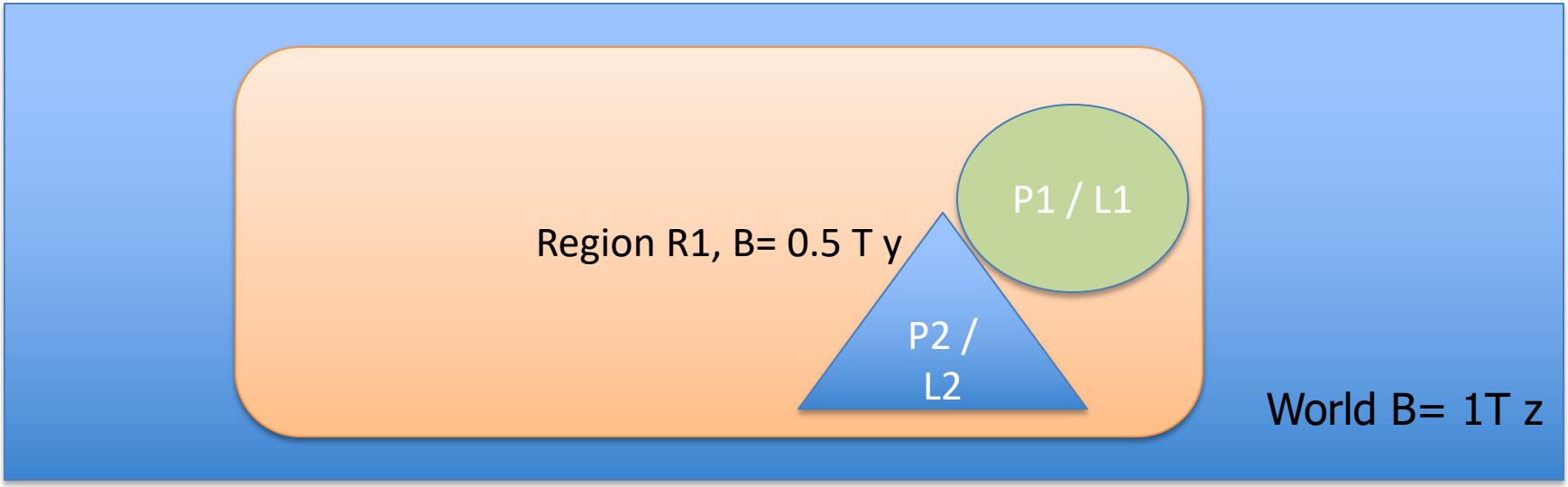
```
#include "G4RK547FEq1.hh"
#include "G4FSALIntegrationDriver.hh"
..
auto stepper1 = new G4RK547FEq1( equation );
auto fsalDriver = new G4FSALIntegrationDriver<G4RK547FEq1>( fMinStep, stepper1 );
```

Dmitry Sorokin, 2017-2020

EXERCISES (HOMEWORK)

Homework Exercise 1 – Theoretical

- A setup contains three magnetic field:
  - the global field manager contains a field B=1T z,
  - the region R1's field manager B= 0.5 T y,
  - the current physical volume P2's logical L2 volume B= 0.1 T z.
- What value will a tracks see if it is:
  i. Inside a physical volume P2 whose logical volume is L2;
  ii. Inside a physical volume P1 that is contained in R1, but whose logical volume is not L2;
  iii. At the boundary between volumes P1 and P2;



Homework Exercise 2

1. Examine examples/basic/B5
   A. How does it create its magnetic field – what method & code ?
   B. Where is its magnetic field class defined ?
   C. What method creates the classes needed to integrate the trajectory (Stepper, Driver, ChordFinder) ?
2. Copy it to another directory (e.g. B5-variant)

3. Set stringent **integration accuracy** requirement
   1. Change the maximum 'one step' integration error to 0.01 mm
   2. Change the maximum relative integration error to 1.0e-04
   3. Change the minimum relative integration error to 1.0e-05

   Hint: Examine the G4FieldManager class to find relevant methods.

4. Examine the 'extended' field example in examples/field/field01
   1. Look for code that creates an Equation of Motion (G4UsualEq_Rhs)
   2. Look for the code which creates a G4DormandPrince745 stepper
   3. Look for the code that creates an Integration Driver
   4. Find where a G4ChordFinder is created

Homework Exercise 3

- Continuing with 'B5-Variant'. Now change the integration method
  A. Try to comment out and replace the 'easy' way to define the integration method

```
fFieldMgr->CreateChordFinder(fMagneticField);
```

  B. First create an equation of motion for a magnetic field – G4UsualEq_Rhs
  C. Next create a Stepper – an embedded Runge-Kutta 'stepper' class that does an integration and returns its result and estimated integration error
    A. Create the (default) G4DormandPrinceRK745 stepper – 7 stages, 4/5th order
  D. Experiment with using a low(er) order method instead
    A. Create a low(er) embedded Runge-Kutta method for integration (e.g. G4BogackiShampine23
  E. Experiment with alternative methods of similar or high order
    A. A different 4th/5th order embedded Runge-Kutta method such as G4TsitourasRK45
    B. You could also use a high order embedded Runge-Kutta method for integration such as G4TsitourasRK45 (4th/5th order) or G4DormandPrinceRK56 (5th/6th order).

Homework Exercise 4 – alternative

A. Examine examples/basic/B2/B2a
   i. How does it creates its magnetic field – what method & code in DetectorConstruction.cc ?
   ii. Check the relevant class in the documentation – where ? (hint: UGAD)
B. Set stringent **integration accuracy** requirement
   i. Change the maximum 'one step' integration error to 0.01 mm
   ii. Change the maximum relative integration error to 1.0e-04
   iii. Change the minimum relative integration error to 1.0e-05
   iv. Hint: Examine the G4FieldManager class to find relevant methods

HINTS & BACKGROUND

Global Field Manager

```
class G4GlobalMagFieldMessenger : public G4UImessenger
{
  public:  // with description

    G4GlobalMagFieldMessenger(const G4ThreeVector& value = G4ThreeVector());
    virtual ~G4GlobalMagFieldMessenger();

    virtual void SetNewValue(G4UIcommand*, G4String);

    void  SetFieldValue(const G4ThreeVector& value);
    G4ThreeVector GetFieldValue() const;

    inline void  SetVerboseLevel(G4int verboseLevel);
    inline G4int GetVerboseLevel() const;
```

Global Field Manager

```
//_____________________________________________________________________

G4GlobalMagFieldMessenger::G4GlobalMagFieldMessenger(const G4ThreeVector& value)
 : G4UImessenger()
{
  fDirectory = new G4UIdirectory("/globalField/");
  fDirectory->SetGuidance("Global uniform magnetic field UI commands");

  fSetValueCmd = new G4UIcmdWith3VectorAndUnit("/globalField/setValue",this);
  fSetValueCmd->SetGuidance("Set uniform magnetic field value.");
  fSetValueCmd->SetParameterName("Bx", "By", "By", false);
  fSetValueCmd->SetUnitCategory("Magnetic flux density");
  fSetValueCmd->AvailableForStates(G4State_PreInit,G4State_Idle);

  fSetVerboseCmd = new G4UIcmdWithAnInteger("/globalField/verbose",this);
  fSetVerboseCmd->SetGuidance("Set verbose level: ");
  fSetVerboseCmd->SetGuidance("  0: no output");
  fSetVerboseCmd->SetGuidance("  1: printing new field value");
  fSetVerboseCmd->SetParameterName("globalFieldVerbose", false);
  fSetVerboseCmd->SetRange("globalFieldVerbose>=0");
  fSetVerboseCmd->AvailableForStates(G4State_PreInit,G4State_Idle);

  // Create field
  fMagField = new G4UniformMagField(value);

  // Set field value (the field is not activated if value is zero)
  SetField(value, "G4GlobalMagFieldMessenger::G4GlobalMagFieldMessenger");
}
```

Hint: G4ChordFinder constructors

```
class G4ChordFinder
{
   public:  // with description

      explicit G4ChordFinder( G4VIntegrationDriver* pIntegrationDriver );
        // The most flexible constructor, which allows the user to specify
        // any type of field, equation, stepper and integration driver.

      G4ChordFinder( G4MagneticField* itsMagField,
                     G4double         stepMinimum = 1.0e-2, // * mm
                     G4MagIntegratorStepper* pItsStepper = nullptr,
                     // G4bool           useHigherEfficiencyStepper = true,
                     G4int            stepperDriverChoice = 2  );
        // A constructor that creates defaults for all "children" classes.
        //
        // The type of equation of motion is fixed.
        // A default type of stepper (Dormand Prince since release 10.4) is used,
        // and the corresponding integration driver.
        // Except if 'useFSAL' is set (true), which provides a FSAL stepper
        // and its corresponding specialised (templated) driver.
```