



GEANT4
A SIMULATION TOOLKIT

Version 10.7

Magnetic Fields

John Apostolakis (CERN)

Geant4 Advanced Course at CERN

4 October 2021 (v7.5 with corrections)



SLAC

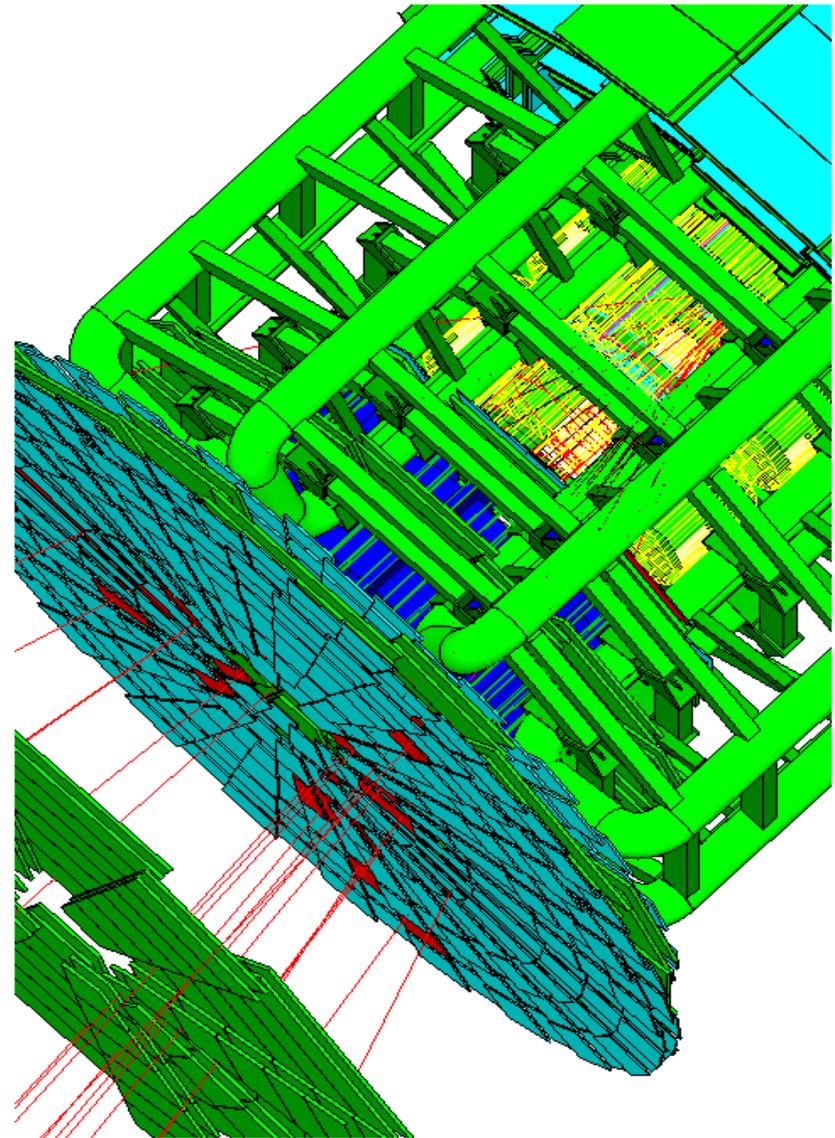
Contents

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

Taking control of integration

- Tuning accuracy
- Advanced integration methods

Many slides adapted from M. Asai (SLAC)





GEANT4
A SIMULATION TOOLKIT

Version 10.7

Defining a magnetic field



SLAC

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);
```

- Alternative: get the field manager from the world volume you created – if you kept a pointer to it as a data member in your G4VUserDetectorConstruction:

```
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 the *global field manager* is associated with the primary geometry (the ‘mass’ geometry in which nearly all materials are assigned.)



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.

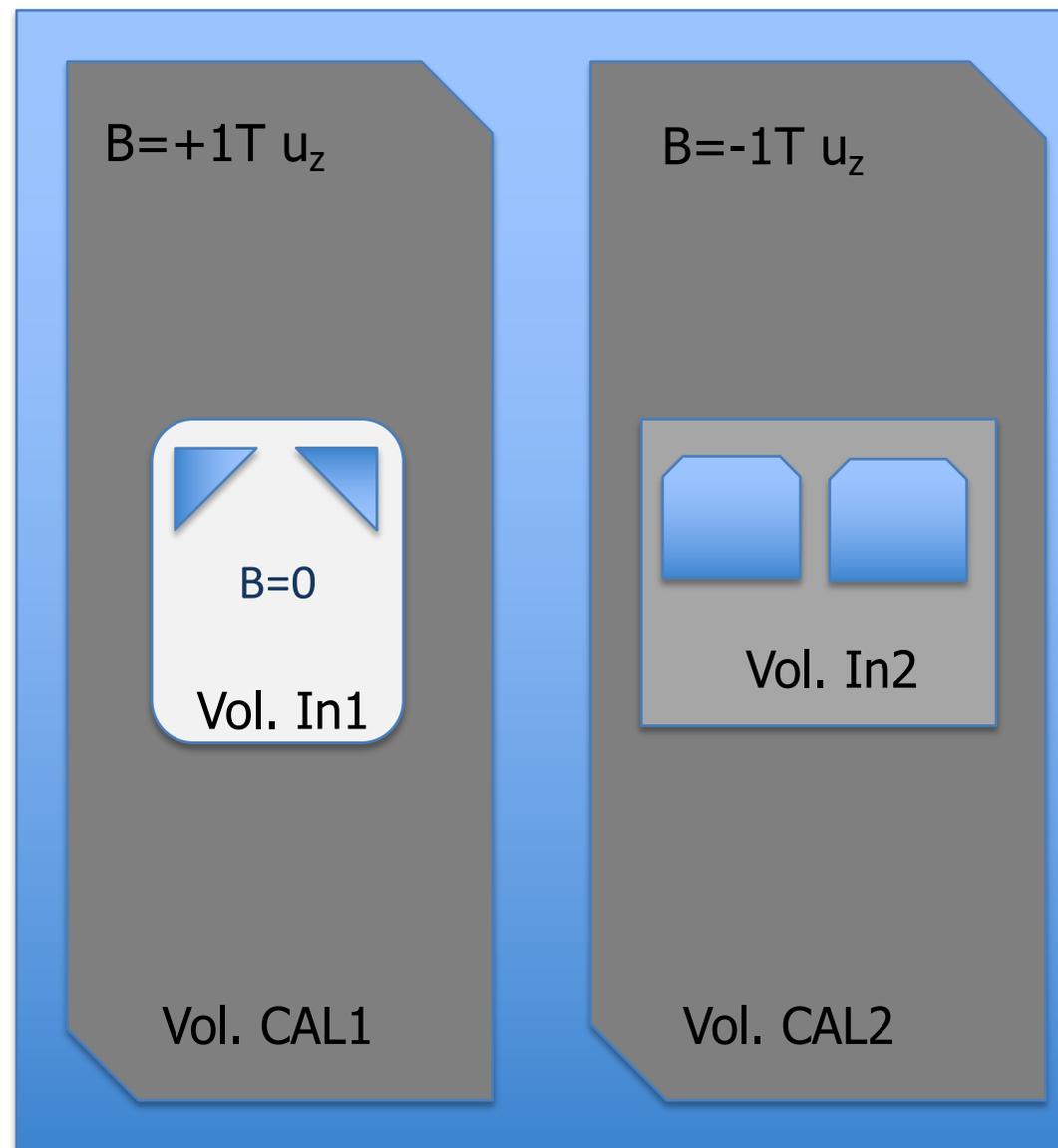
Let's clarify this with an example.

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 create a

```
G4MagneticField Bfield;
G4FieldManager nullFieldMgr= new
G4FieldManager( Bfield=nullptr; )
```

Then we assign a



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

a user must create an integration method. There is a very easy way:

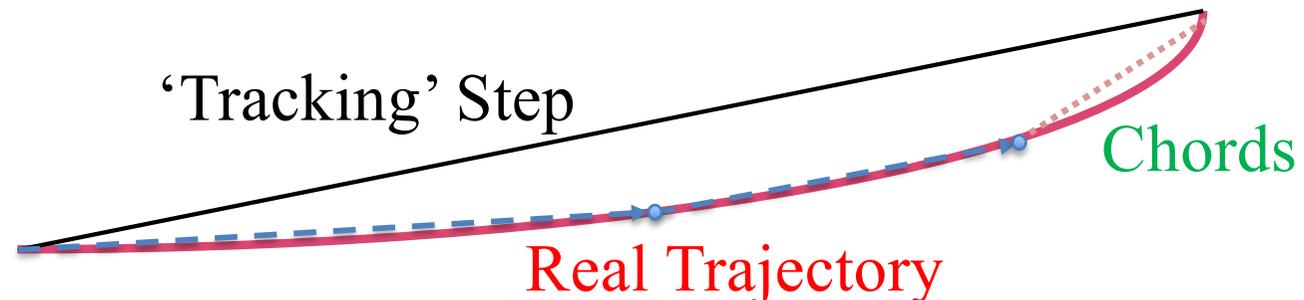
```
// Use the default method - only for magnetic fields  
fieldManager->CreateChordFinder(myMagField);
```

```
G4bool pushToContained = true; //overwrites an existing field managers  
myLogicalVol->SetFieldManager(fieldManager, pushToContained); // in progeny  
// Register the field and its manager for deleting at the end  
G4AutoDelete::Register(myMagField);  
G4AutoDelete::Register(fieldManager);
```

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

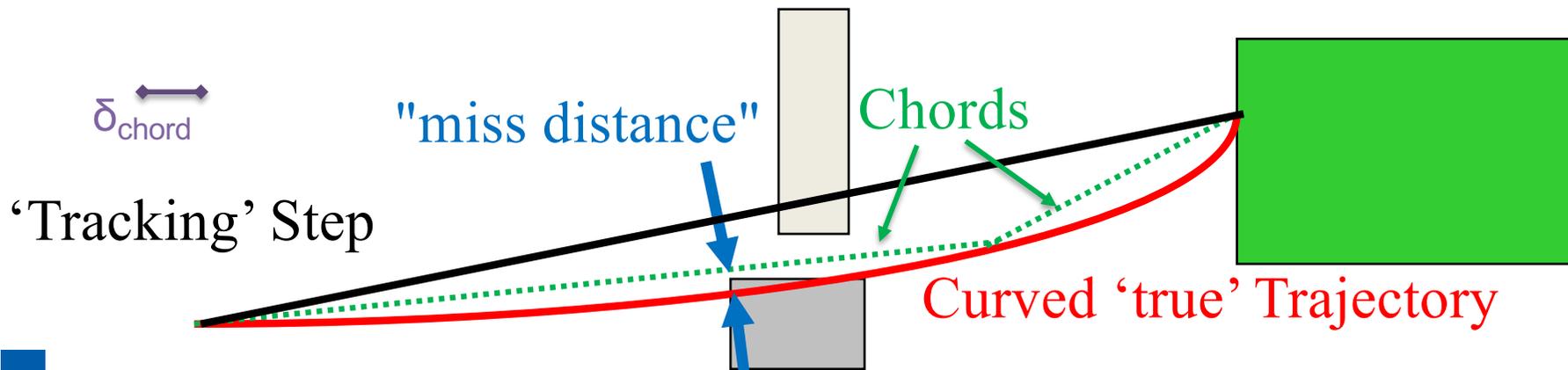
Field integration: overview of Geant4 approach

- To propagate a particle in an external field (magnetic, electric, both or other), we integrate numerically its **equation of motion**.
- Use Runge-Kutta integration method for ODEs (ordinary differential equations)
 - Several Runge-Kutta ‘steppers’ are available – used in most applications.
- 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.

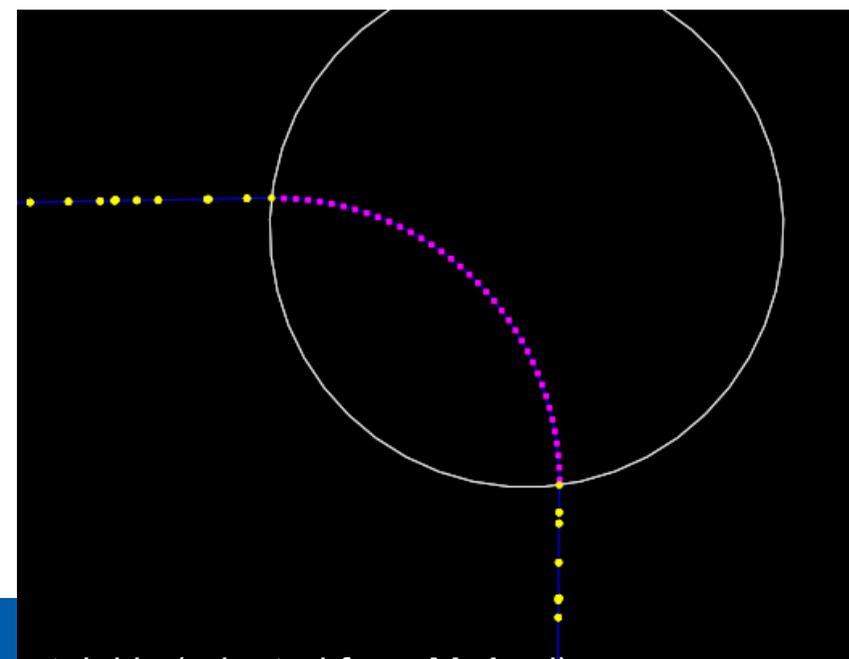
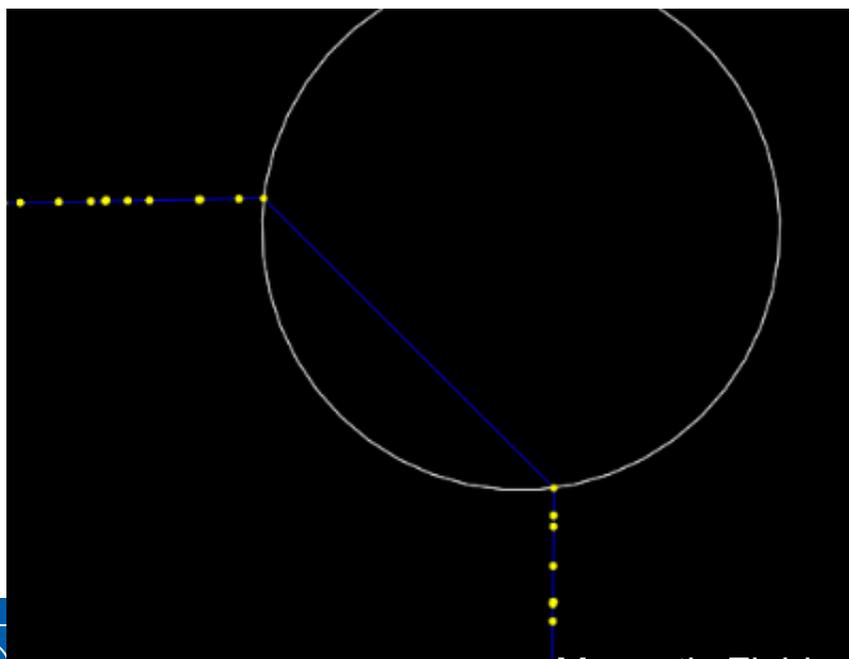
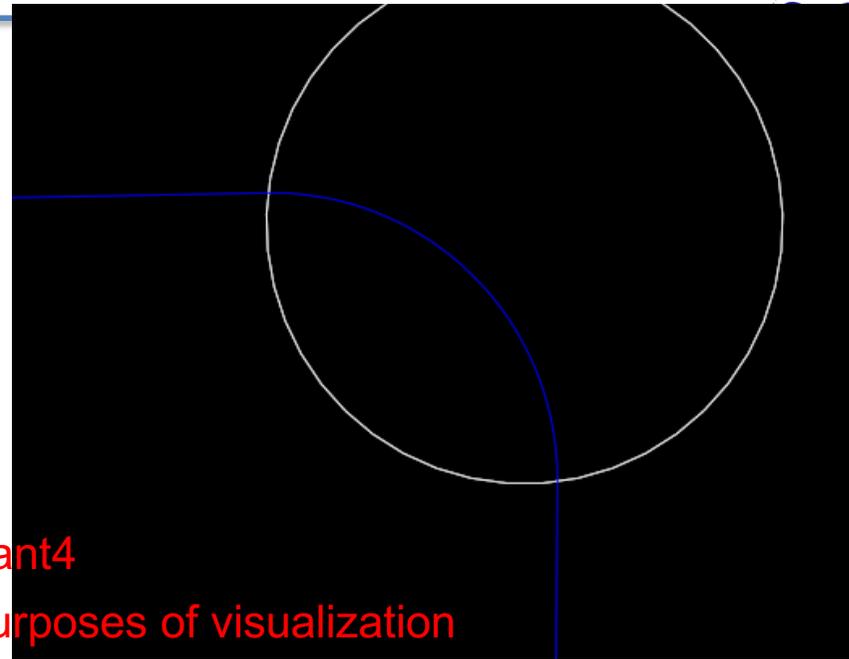
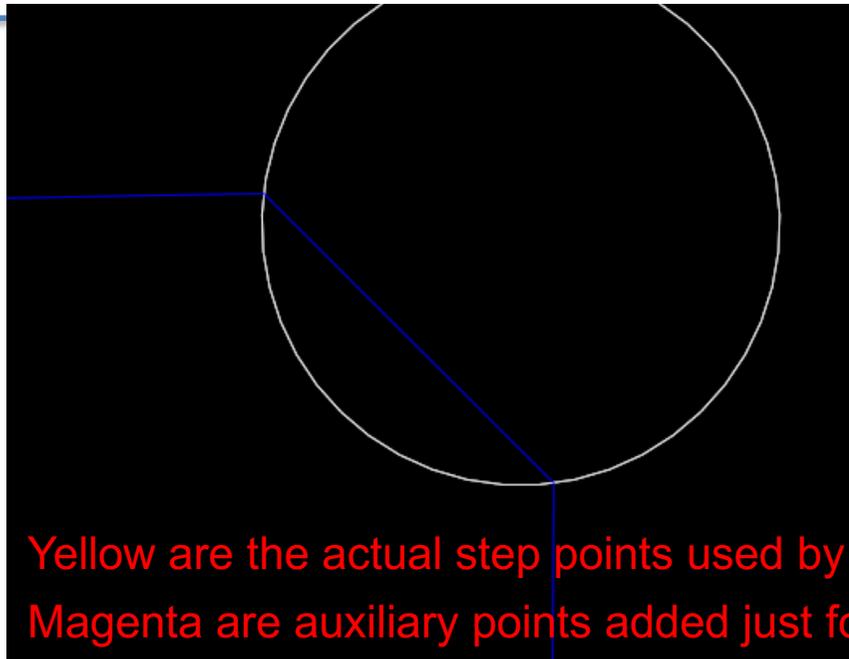


Tracking in field

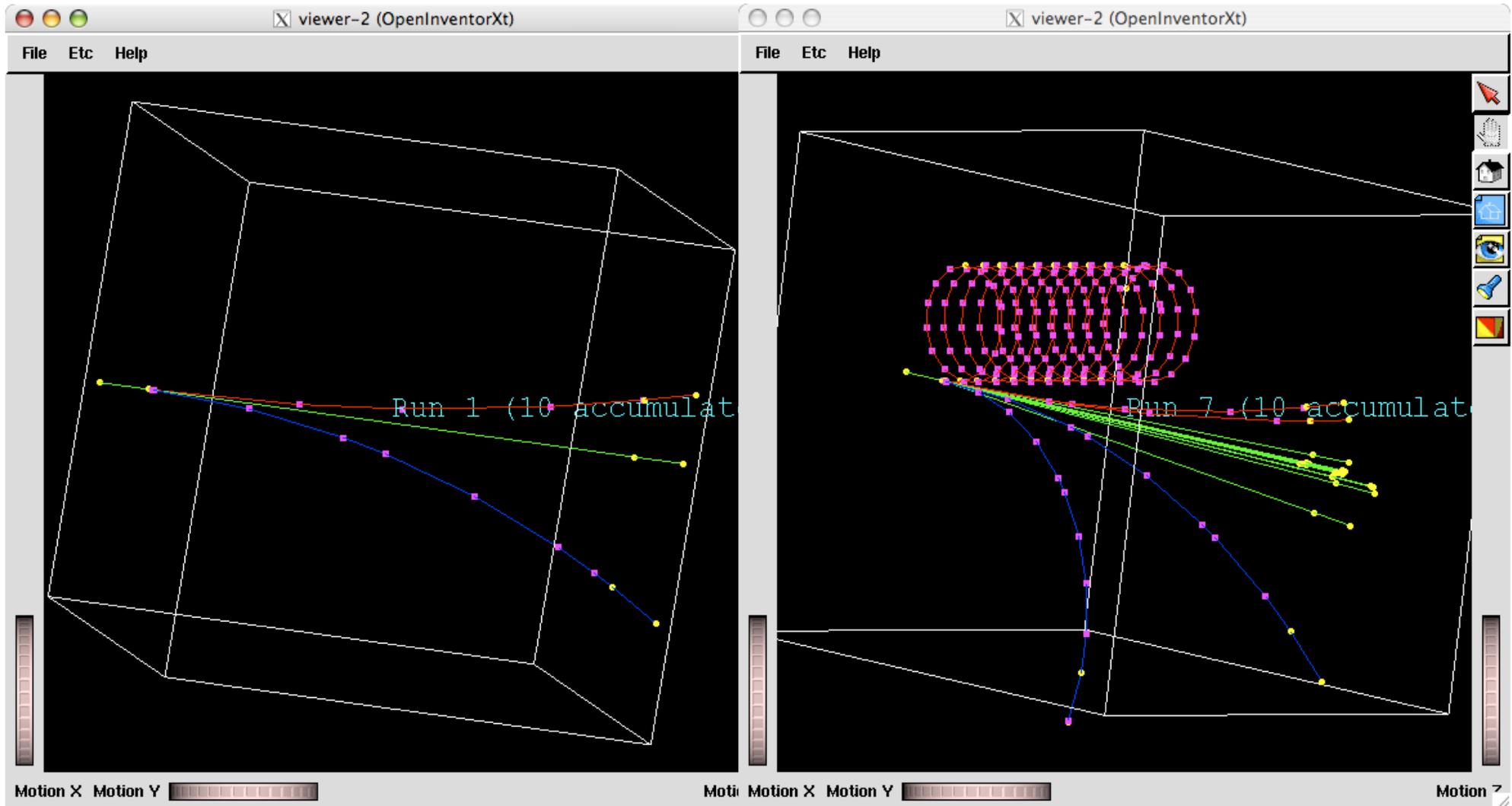
- The chords are used to interrogate the **G4Navigator**, to check whether/where a track has crossed a volume boundary.
- One physics/tracking step can create several chords.
 - In gases or vacuum, a ‘physics’ step can span several helix turns.
- The user controls the accuracy of the volume intersection,
 - 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



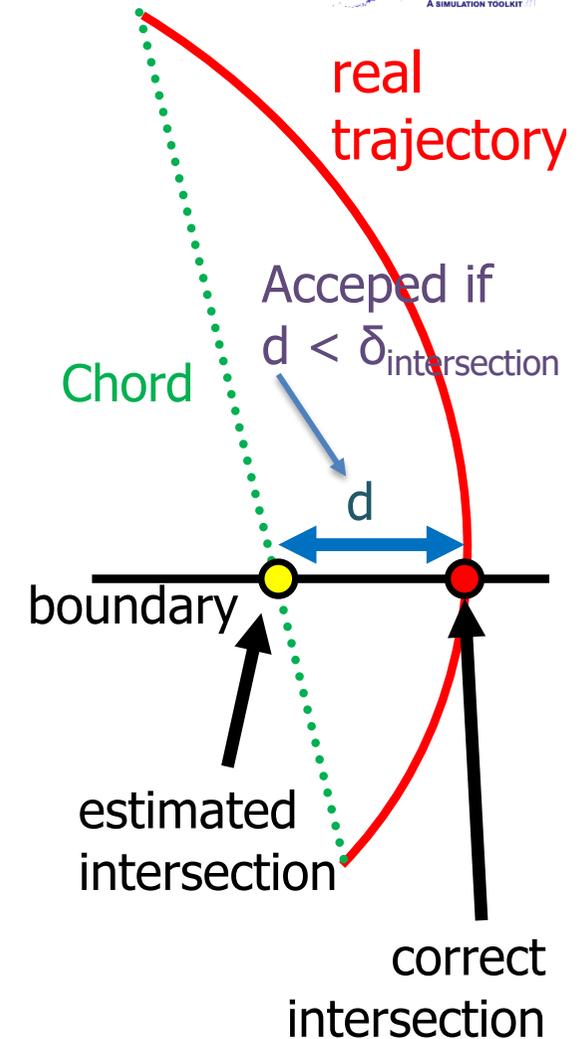
Smooth Trajectory Makes Big Difference for Trajectories that Loop in a Magnetic Field



- 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: $|\Delta x| < \epsilon_{\max} s$ and $|\Delta p| < \epsilon_{\max} |p|$
- 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 $< 1000 \delta_{1 \text{ step}}$)
 - Values of $\delta_{\text{intersection}}$ and $\delta_{1 \text{ step}}$ 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**.



GEANT4
A SIMULATION TOOLKIT

Version 10.7

Additional Fields & Integration Methods

Field integration – usual and custom methods
Alternative types of field



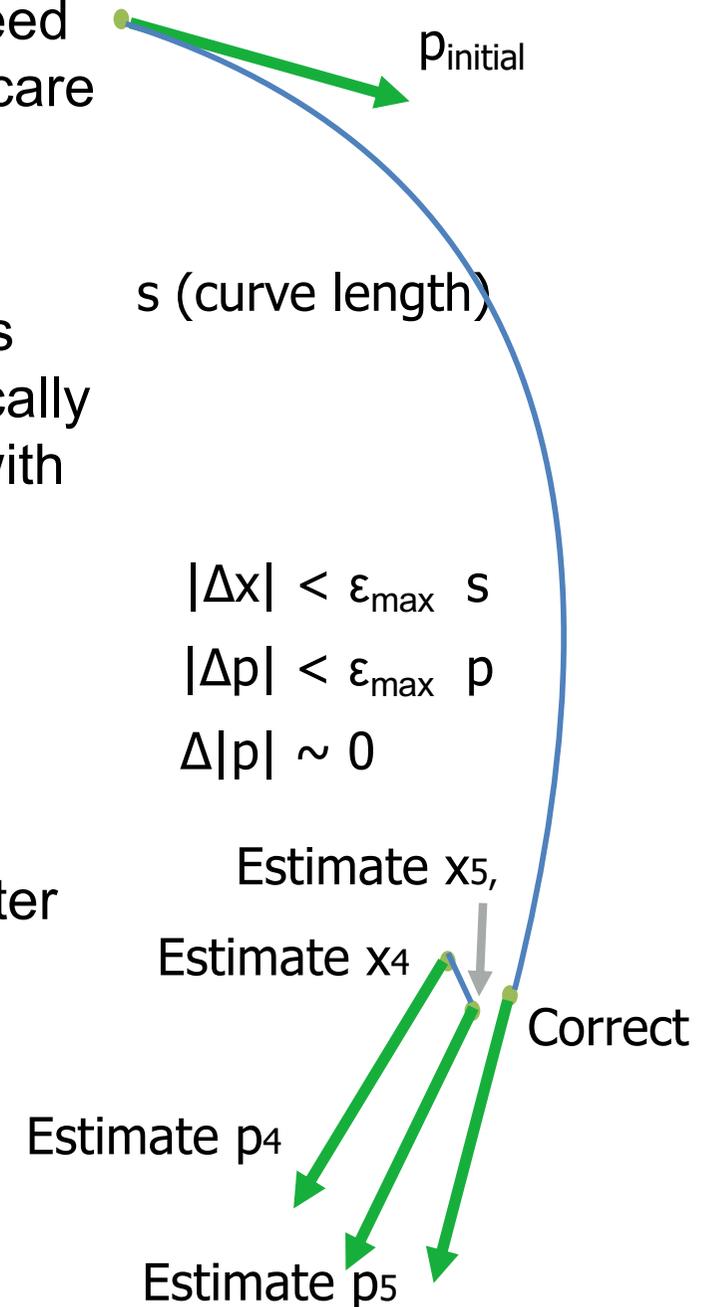
SLAC

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 4th-5th order RK stepper. (Embedded = compares 4th & 5th 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 $F(x, y)$
 - $f_i = F(x_0 + a_i h, y_0 + h \sum_{j<i} b_{ij} f_j)$
 - $y_{\text{estim}}(x_0 + h) = \sum_i c_i f_i$
- Each method has a ‘Butcher 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 $\sim h^N$
 - 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	0	b_{ij}
	$\frac{1}{2}$	$\frac{1}{2}$
	$\frac{3}{4}$	0 $\frac{3}{4}$
c_j	1	$\frac{2}{9}$ $\frac{1}{3}$ $\frac{4}{9}$

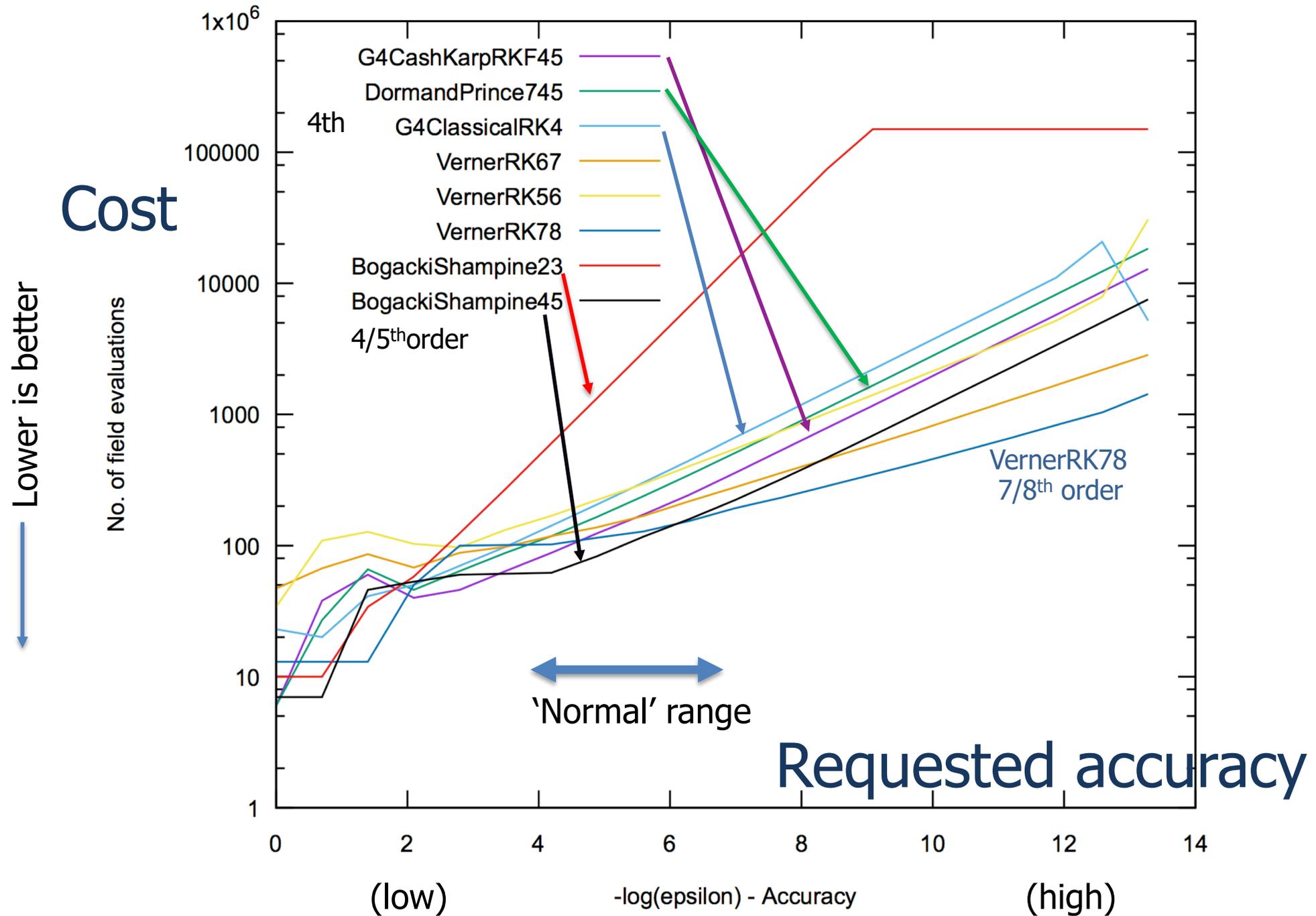
$$y_{\text{RBS3}} = 2f_1/9 + f_2/3 + 4f_3/9$$

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

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

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

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 (\rightarrow 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 4th-5th order RK stepper. (Embedded = compares 4th & 5th order to estimate error.)
 - If the field is very smooth, you may consider higher order steppers
 - of most potential interest in large volumes filled with gas or vacuum.
- If the field is rough, 3rd order steppers could obtain the results of same quality using fewer computing cycles
 - 3rd order stepper(s): **G4SimpleHeum**, **G4BogackiShampine23** (FSAL)
- For reasonably smooth (or not very rough) fields, the choice between 3rd , 4th or 5th order steppers should be made by trial and error.
- For the 'roughest' field map, consider a robust 2nd 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 2nd 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, you must now create a new stepper.

```
G4int nvar= 6; // Integrate only position[3] + momentum[3]
G4HelixSimpleHeum* stepper= new G4HelixSimpleHeum(equation, nvar);
delete theChordFinder->GetIntegrationDriver();
auto integrationDriver= new
    G4IntegrationDriver<G4HelixSimpleHeum>( fMinStep, stepper,
                                           stepper->GetNumberOfVariables() );

theChordFinder->SetIntegrationDriver(integrationDriver);
```

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

Other types of 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:

```
G4int nvar= 8; // Integrate position[3] + momentum[3] + E + t
G4ElectricField* EMfield = new
    G4UniformElectricField( G4ThreeVector(0., 1.0e5*kilovolt/cm, 0.) );
auto equation = new G4EqMagElectricField(EMfield);
auto stepper = new G4DormandPrince745( equation, nvar );
G4FieldManager* fieldManager =
    G4TransportationManager::GetTransportationManager()-> GetFieldManager();
fieldManager->SetDetectorField( EMfield );
auto integrDriver= new
    G4IntegrationDriver<G4DormandPrince745>( fMinStep, stepper,
                                             stepper-
>GetNumberOfVariables() );
G4ChordFinder* fChordFinder = new G4ChordFinder(integrDriver);
```

- Notes:

1. A user-defined field type can be time-dependent (its value can change with time).
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**)

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

NEWEST METHODS

Alternative to RK, and ‘improved’ RK methods with Interpolation



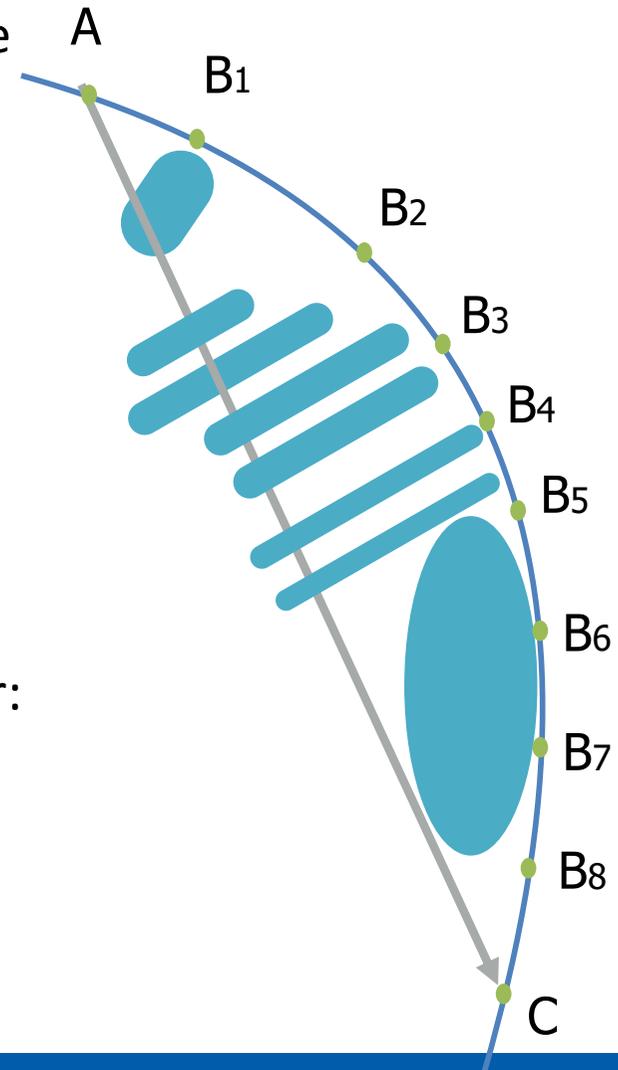
The Bulirsch Stoer method is a multi-step method, alternative to Runge-Kutta:

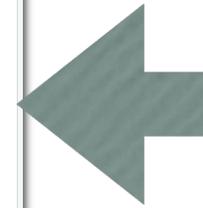
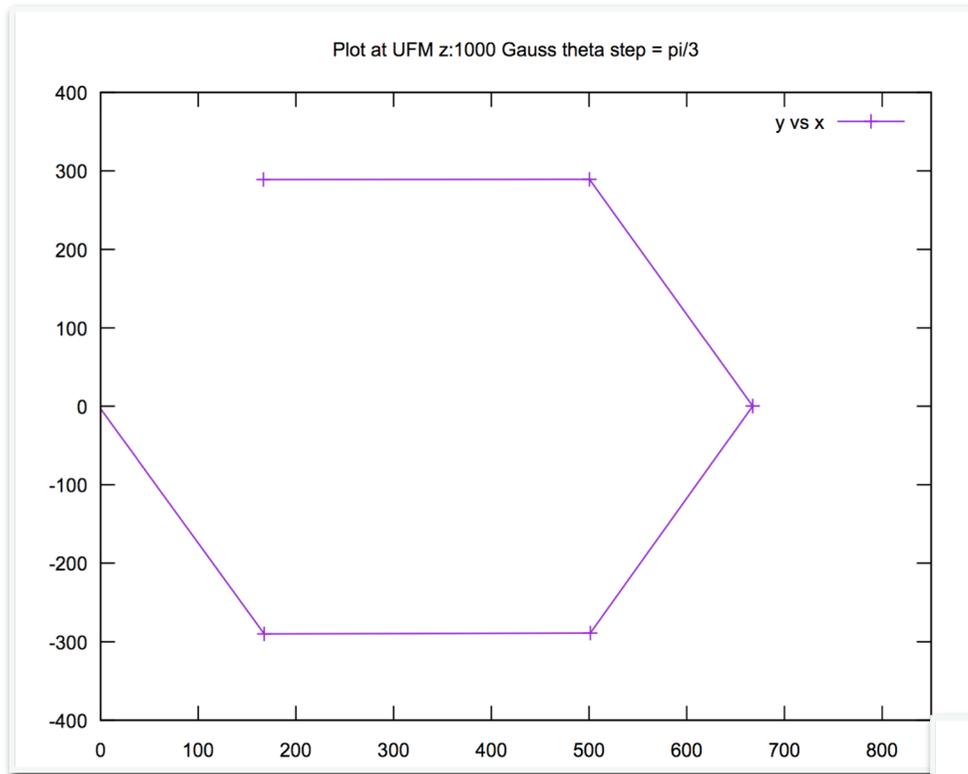
```
G4BulirschStoer* pBSstepper =  
    new G4BulirschStoer( fEquation, nVar, epsilon );  
auto pDriver = new G4IntegrationDriver<G4BulirschStoer>( stepMinimum,  
                                                         pBSstepper, nVar );
```

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

- ▶ 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

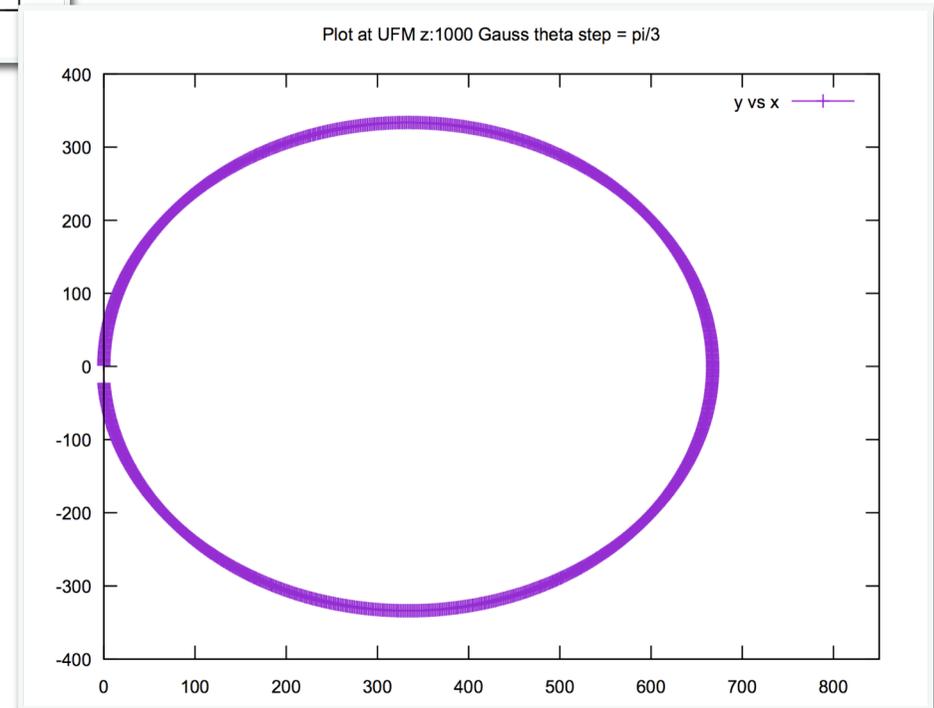




6 computed points in a circular trajectory

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

Interpolated result



Using the latest developments – FSAL and Interpolation



These promising Runge-Kutta methods are currently optional, and can be tried out 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 );
```

- ▶ 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"
```

```
..
```

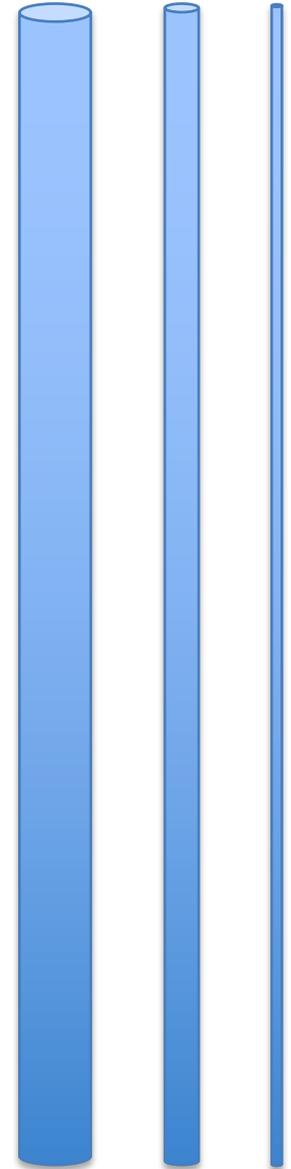
```
auto stepper1 = new G4RK547FEq1( equation );
```

```
auto fsalDriver = new G4FSALIntegrationDriver<G4RK547FEq1>( fMinStep, stepper1 );
```

Dmitry Sorokin, 2017-2020

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 $E < E_{\text{warning}}$ a track is killed immediately without warning
 - If $E_{\text{warning}} < E < E_{\text{important}}$ the track is killed, and a small warning is printed (to cout in Geant4 versions < 10.5)
 - If $E > E_{\text{important}}$ 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):
 - $E_{\text{warning}} = 100 \text{ MeV}$
 - $E_{\text{important}} = 250 \text{ MeV}$



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.

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 4th-5th 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.