Montecarlo Methods for Medical Physics

Francesco Longo
(francesco.longo@ts.infn.it)

Summary of the Course

- Part1 (Monday)
  - General (and brief) introduction to Monte Carlo methods
  - Montecarlo methods in Medical Physics
- Part2 (Monday)
  - Introduction to the Geant4 toolkit
- Part3
  - Fundamentals of a Geant4 application (Tuesday)
  - Geometry, Physics, Particle Flux, Scoring needs (Today - …)
- Laboratory (Next weeks)
  - Realisation of an example relevant to Medical Physics

Part 2

Introduction to Geant4

Strategic vision

OO technology

- Openness to **extension** and **evolution**

new implementations can be added w/o changing the existing code

- Robustness and ease of **maintenance**

protocols and well defined dependencies minimize coupling

Toolkit

A set of compatible components

- each component is **specialised** for a specific functionality
- each component can be **refined** independently to a great detail
- components can be **integrated** at any degree of complexity
- it is easy to provide (and use) **alternative** components
- the user application can be **customised** as needed

Outline of Part2

- General Introduction to G4
  - What is G4 ?
  - Review of user documentation
  - Geant4 as a toolkit
- Basics of OO programming
- Geant4 Kernel and basics of the toolkit
  - Run, Event, Step
  - Particle and Physics processes
  - User classes

Simulation basics

Geant4 simulation toolkit

- Modeling the experimental set-up
- Tracking particles through matter
- Interaction of particles with matter
- Modeling the detector response
- Run and event control
- Accessory utilities *(random number generators, PDG particle information, physical constants, system of units etc.)*

- User interface
- Interface to event generators
- Visualisation *(of the set-up, tracks, hits etc.)*
- Persistency
- Analysis

Part 2

Main ingredients of a G4 application

To use Geant4, you have to…

- Geant4 is a toolkit. You have to build an application.
- To make an application, you have to
  - Define your geometrical setup
    - Material, volume
  - Define physics to get involved
    - Particles, physics processes/models
    - Production thresholds
  - Define how an event starts
    - Primary track generation
  - Extract information useful to you
- You may also want to
  - Visualize geometry, trajectories and physics output
  - Utilize (Graphical) User Interface
  - Define your own UI commands
  - etc.

Run in Geant4

- As an analogy of the real experiment, a run of Geant4 starts with “Beam On”.
- Within a run, the user cannot change
  - detector setup
  - settings of physics processes
- Conceptually, a run is a collection of events which share the same detector and physics conditions.
  - A run consists of one event loop.
- At the beginning of a run, geometry is optimized for navigation and cross-section tables are calculated according to materials appear in the geometry and the cut-off values defined.
- G4RunManager class manages processing a run, a run is represented by G4Run class or a user-defined class derived from G4Run.
  - A run class may have a summary results of the run.
- G4UserRunAction is the optional user hook.

Event in Geant4

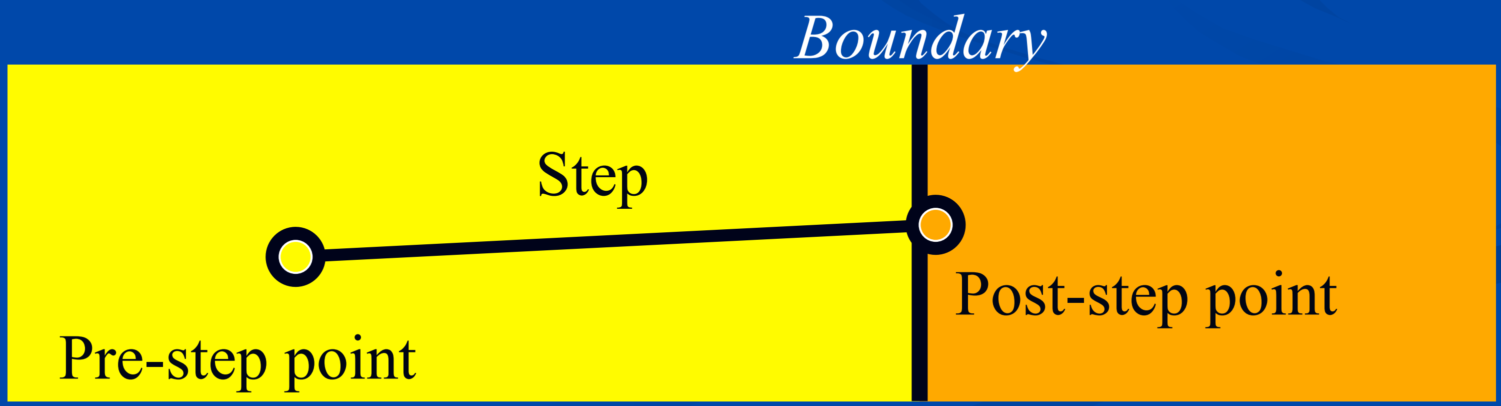
- An event is the basic unit of simulation in Geant4.
- At beginning of processing, primary tracks are generated. These primary tracks are pushed into a stack.
- A track is popped up from the stack one by one and "tracked". Resulting secondary tracks are pushed into the stack.
  - This "tracking" lasts as long as the stack has a track.
- When the stack becomes empty, processing of one event is over.
- G4Event class represents an event. It has following objects at the end of its (successful) processing.
  - List of primary vertices and particles (as input)
  - Hits and Trajectory collections (as output)
- G4EventManager class manages processing an event. G4UserEventAction is the optional user hook.

Track in Geant4

- Track is a snapshot of a particle.
  - It has physical quantities of current instance only. It does not record previous quantities.
  - Step is a "delta" information to a track. Track is not a collection of steps. Instead, a track is being updated by steps.
- Track object is deleted when
  - it goes out of the world volume,
  - it disappears (by e.g. decay, inelastic scattering),
  - it goes down to zero kinetic energy and no "AtRest" additional process is required, or
  - the user decides to kill it artificially.
- No track object persists at the end of event.
  - For the record of tracks, use trajectory class objects.
- G4TrackingManager manages processing a track, a track is represented by G4Track class.
- G4UserTrackingAction is the optional user hook.

Step in Geant4

- Step has two points and also "delta" information of a particle (energy loss on the step, time-of-flight spent by the step, etc.).
- Each point knows the volume (and material). In case a step is limited by a volume boundary, the end point physically stands on the boundary, and it logically belongs to the next volume.
  - Because one step knows materials of two volumes, boundary processes such as transition radiation or refraction could be simulated.
- G4SteppingManager class manages processing a step, a step is represented by G4Step class.
- G4UserSteppingAction is the optional user hook.



Particle in Geant4

- A particle in Geant4 is represented by three layers of classes.
- G4Track
  - Position, geometrical information, etc.
  - This is a class representing a particle to be tracked.
- G4DynamicParticle
  - "Dynamic" physical properties of a particle, such as momentum, energy, spin, etc.
  - Each G4Track object has its own and unique G4DynamicParticle object.
  - This is a class representing an individual particle.
- G4ParticleDefinition
  - "Static" properties of a particle, such as charge, mass, life time, decay channels, etc.
  - G4ProcessManager which describes processes involving to the particle
  - All G4DynamicParticle objects of same kind of particle share the same G4ParticleDefinition.

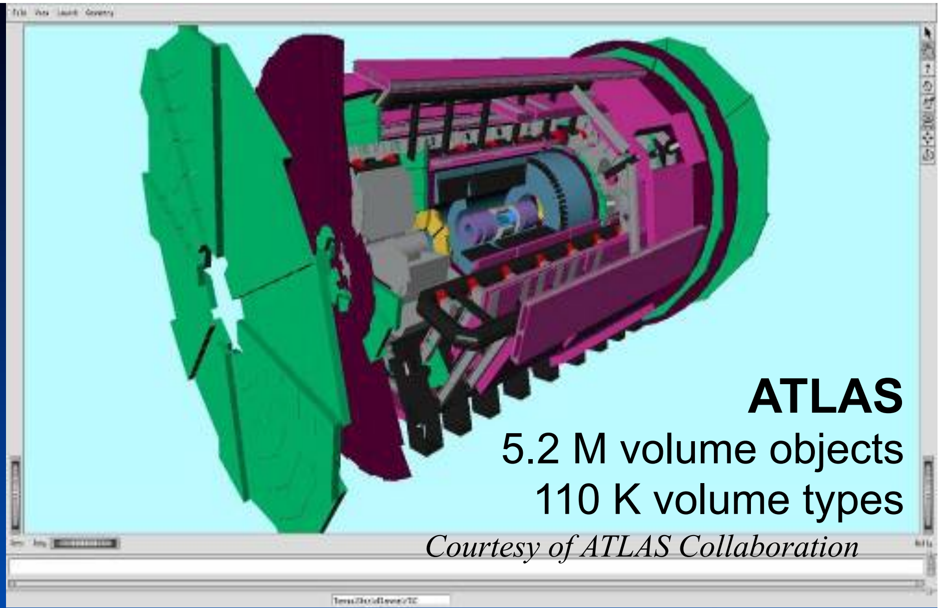
Extract useful information

- Given geometry, physics and primary track generation, Geant4 does proper physics simulation "silently".
  - You have to add a bit of code to extract information useful to you.
- There are two ways:
  - Use user hooks (G4UserTrackingAction, G4UserSteppingAction, etc.)
    - You have an access to almost all information
    - Straight-forward, but do-it-yourself
  - Use Geant4 scoring functionality
    - Assign G4VSensitiveDetector to a volume
    - Hits collection is automatically stored in G4Event object, and automatically accumulated if user-defined Run object is used.
  - Use Geant4 native scorers to get specified quantities (dose, energy release, flux, path length, etc.)

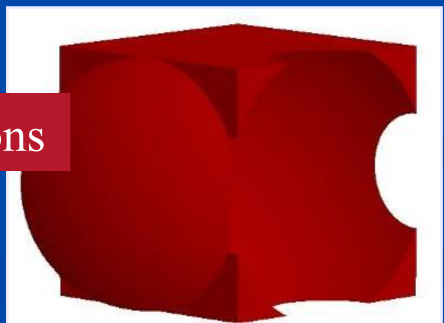
Geometry

Geometry

- Role
  - detailed detector description
  - efficient navigation

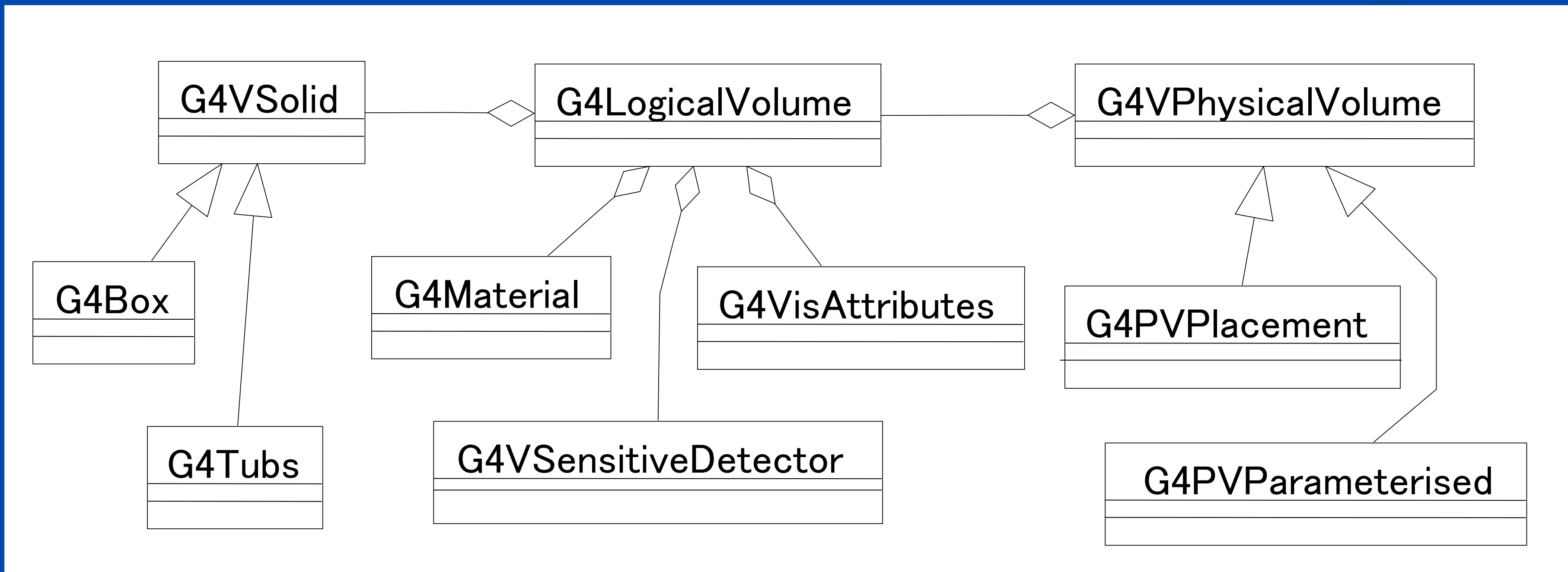


- Three conceptual layers
  - Solid: shape, size
  - LogicalVolume: material, sensitivity, daughter volumes, etc.
  - PhysicalVolume: position, rotation
- One can do fancy things with geometry…



Detector geometry

- Three conceptual layers
  - G4VSolid -- *shape, size*
  - G4LogicalVolume -- *daughter physical volumes,*
    material, sensitivity, user limits, etc.
  - G4VPhysicalVolume -- *position, rotation*



Geometry - Materials

Definition of Materials

- Different kinds of materials can be described:
  - isotopes <-> G4Isotope
  - elements <-> G4Element
  - molecules, compounds and mixtures <-> G4Material
- Attributes associated to G4Material:
  - temperature, pressure, state, density
- Prefer low-density material to vacuum

- Single element material

```
double density = 1.390*g/cm3;
double a = 39.95*g/mole;
G4Material* lAr =
 new G4Material("liquidArgon",z=18.,a,density);
```

First Homework

- Review G4 web pages
- Find Appropriate documentation
- Find relevant Medical Physics examples
- Define your preferred project
  - Simple geometry
  - Particle distributions
  - Scoring needs

Physics Lists

Strategic vision

OO technology

- Openness to **extension** and **evolution**

new implementations can be added w/o changing the existing code

- Robustness and ease of **maintenance**

protocols and well defined dependencies minimize coupling

Toolkit

A set of compatible components

- each component is **specialised** for a specific functionality
- each component can be **refined** independently to a great detail
- components can be **integrated** at any degree of complexity
- it is easy to provide (and use) **alternative** components
- the user application can be **customised** as needed

Physics

From the Minutes of LCB (LHCC Computing Board) meeting on 21 October, 1997:

"It was noted that experiments have requirements for **independent, alternative physics models**.
In Geant4 these models, differently from the concept of packages, allow the user to **understand** how the results are produced, and hence improve the **physics validation**. Geant4 is developed with a modular architecture and is the ideal framework where existing components are integrated and new models continue to be developed."

To use Geant4, you have to…

- Geant4 is a toolkit. You have to build an application.
- To make an application, you have to
  - Define your geometrical setup
    - Material, volume
  - Define physics to get involved
    - Particles, physics processes/models
    - Production thresholds
  - Define how an event starts
    - Primary track generation
  - Extract information useful to you
- You may also want to
  - Visualize geometry, trajectories and physics output
  - Utilize (Graphical) User Interface
  - Define your own UI commands
  - etc.

User classes

- main()
  - Geant4 does not provide *main().*

  Note : classes written in yellow are mandatory.
- Initialization classes
  - Use G4RunManager::SetUserInitialization() to define.
  - Invoked at the initialization
    - G4VUserDetectorConstruction
    - G4VUserPhysicsList
- Action classes
  - Use G4RunManager::SetUserAction() to define.
  - Invoked during an event loop
    - G4VUserPrimaryGeneratorAction
    - G4UserRunAction
    - G4UserEventAction
    - G4UserStackingAction
    - G4UserTrackingAction
    - G4UserSteppingAction

Physics in Geant4

- It is rather unrealistic to develop a uniform physics model to cover wide variety of particles and/or wide energy range.
- Much wider coverage of physics comes from mixture of theory-driven, parameterized, and empirical formulae. Thanks to polymorphism mechanism, both cross-sections and models (final state generation) can be combined in arbitrary manners into one particular process.
- Geant4 offers
  - EM processes
  - Hadronic processes
  - Photon/lepton-hadron processes
  - Optical photon processes
  - Decay processes
  - Shower parameterization
  - Event biasing techniques
  - And you can plug-in more

What is a Physics List?

- A class which collects all the particles, physics processes and production thresholds needed for your application
- It tells the run manager how and when to invoke physics

- It is a very flexible way to build a physics environment
  - user can pick the particles he wants
  - user can pick the physics to assign to each particle

- But, user must have a good understanding of the physics required
  - omission of particles or physics could cause errors or poor simulation

Why Do We Need a Physics List?

- Physics is physics – shouldn't Geant4 provide, as a default, a complete set of physics that everyone can use?
- No:
  - there are many different physics models and approximations
    - very much the case for hadronic physics
    - but also the case for electromagnetic physics
  - computation speed is an issue
    - a user may want a less-detailed, but faster approximation
  - no application requires all the physics and particles Geant4 has to offer
    - e.g., most medical applications do not want multi-GeV physics

Why Do We Need a Physics List?

- For this reason Geant4 takes an atomistic, rather than an integral approach to physics
  - provide many physics components (processes) which are de-coupled from one another
  - user selects these components in custom-designed physics lists in much the same way as a detector geometry is built

Physics Processes Provided by Geant4

- EM physics
  - "standard" processes valid from ~ 1 keV to ~ PeV
  - "low-energy" Livermore/ Penelope valid from 250 eV to ~ PeV
  - optical photons
- Weak physics
  - decay of subatomic particles
  - radioactive decay of nuclei
- Hadronic physics
  - pure hadronic processes valid from 0 to ~100 TeV
  - $\gamma-$, $\mu-$nuclear valid from 10 MeV to ~TeV
- Parameterized or "fast simulation" physics

Pre-packaged Physics Lists (1)

- Our example deals mainly with electromagnetic physics
- A complete and realistic set of EM physics lists are provided
  - add to it according to your needs
- Adding hadronic physics is more involved
  - for any one hadronic process, user may choose from several hadronic models to choose from
  - choosing the right models for your application requires care
  - to make things easier, hadronic physics lists are now provided according to some use cases

Pre-packaged Physics Lists (2)

- Originally referred to as “hadronic physics lists” but include electromagnetic physics already
- Can be found on the Geant4 web page at
  - PhysicsList Guide

- Caveats:
  - these lists are provided as a “best guess” of the physics needed in a given case
  - The user is responsible for validating the physics for his own application and adding (or subtracting) the appropriate physics
    - “Trust, but verify.”
  - they are intended as starting points or templates

Reference Physics Lists

- Reference physics lists attempt to cover a wide range of use cases
  - Extensive validation by LHC experiments for simulation hadronic showers
  - Comparison experinents for neutron production and transport demonstrates good agreement
    - QGSP_BIC_HP, QGSP_BERT_HP
  - user feedback, e.g. vi hypernews, is welcome
- Users responsible for validating results

- Documentation available from G4 Physics List manual
- Physics Lists User forum for questions and feedback

G4 home page

- https://geant4.web.cern.ch/

Geant4

User Support

1. Getting started
2. Training courses and materials
3. Source code
   a. Download page
   b. LXR code browser
   c. doxygen documentation
   d. GitHub
   e. GitLab @ CERN
4. Frequently Asked Questions (FAQ)
5. Bug reports and fixes
6. User requirements tracker
7. User Forum
8. Documentation
   a. Introduction to Geant4 [pdf]
   b. Installation Guide: [pdf]
   c. Application Developers [pdf]
   d. Toolkit Developers Guide [pdf]
   e. Physics Reference Manual [pdf]
   f. Physics List Guide [pdf]
9. Examples
10. User Aids
    a. Tips for improving CPU performance
11. Contact Coordinators & Contact Persons

Hands On

Work on Medical Physics Example

- Check the Example documentation or the source code.
  - Find the geometrical info
  - Find the physics list
  - Find the particle source mechanism
  - Find the particle scoring mechanism
- Start designing your application …

Laboratory

Procedure

1) Copiate l’esempio basic B3 nella vostra directory
2) Entrate nella vostra directory B3
3) Costruite la sottodirectory dove compilerete
4) Entrate nella directory B3/build
5) Eseguite il comando per creare i makefiles
6) Compilate l’esempio
7) Entrate nella directory B3/build/B3a
8) Lanciate l’esempio exampleB3a

Procedure

1) cp -r /gpfs/glast/Geant4/G4_10.5.p01/geant4.10.05.p01-install/share/Geant4-10.5.1/examples/basic/B3 .
2) cd B3
3) mkdir build
4) cd build
5) cmake ../
6) make
7) cd B3a
8) ./exampleB3a

Particle Generation

To use Geant4, you have to…

- Geant4 is a toolkit. You have to build an application.
- To make an application, you have to
  - Define your geometrical setup
    - Material, volume
  - Define physics to get involved
    - Particles, physics processes/models
    - Production thresholds
  - Define how an event starts
    - Primary track generation
  - Extract information useful to you
- You may also want to
  - Visualize geometry, trajectories and physics output
  - Utilize (Graphical) User Interface
  - Define your own UI commands
  - etc.

User classes

- main()
  - Geant4 does not provide *main().*

  Note : classes written in yellow are mandatory.
- Initialization classes
  - Use G4RunManager::SetUserInitialization() to define.
  - Invoked at the initialization
    - G4VUserDetectorConstruction
    - G4VUserPhysicsList
- Action classes
  - Use G4RunManager::SetUserAction() to define.
  - Invoked during an event loop
    - G4VUserPrimaryGeneratorAction ←
    - G4UserRunAction
    - G4UserEventAction
    - G4UserStackingAction
    - G4UserTrackingAction
    - G4UserSteppingAction

Primary particle generation

G4VUserPrimaryGeneratorAction

- This class is one of mandatory user classes to control the generation of primaries.
  - This class itself should NOT generate primaries but invoke `GeneratePrimaryVertex()` method of primary generator(s) to make primaries.
- Constructor
  - Instantiate primary generator(s)
  - Set default values to it(them)
- GeneratePrimaries() method
  - Randomize particle-by-particle value(s)
  - Set these values to primary generator(s)
    - Never use hard-coded UI commands
  - Invoke `GeneratePrimaryVertex()` method of primary generator(s)

Built-in primary particle generators

G4ParticleGun

- Concrete implementations of G4VPrimaryGenerator
  - A good example for experiment-specific primary generator implementation
- It shoots one primary particle of a certain energy from a certain point at a certain time to a certain direction.
  - Various set methods are available
  - Intercoms commands are also available for setting initial values
- One of most frequently asked questions is :

  I want "particle shotgun", "particle machinegun", etc.

- Instead of implementing such a fancy weapon, in your implementation of UserPrimaryGeneratorAction, you can
  - Shoot random numbers in arbitrary distribution
  - Use set methods of G4ParticleGun
  - Use G4ParticleGun as many times as you want
  - Use any other primary generators as many times as you want to make overlapping events

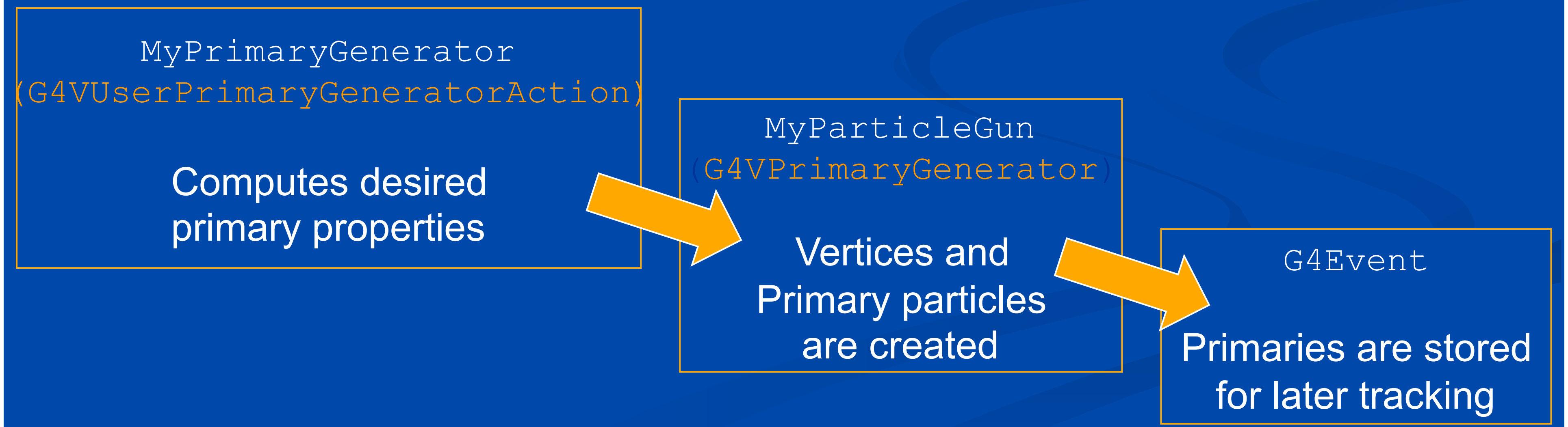
G4VUserPrimaryGeneratorAction

```
void T01PrimaryGeneratorAction::
          GeneratePrimaries(G4Event* anEvent)
{ G4ParticleDefinition* particle;
  G4int i = (int)(5.*G4UniformRand());
  switch(i)
  { case 0: particle = positron; break; ... }
  particleGun->SetParticleDefinition(particle);
  G4double pp =
    momentum+(G4UniformRand()-0.5)*sigmaMomentum;
  G4double mass = particle->GetPDGMass();
  G4double Ekin = sqrt(pp*pp+mass*mass)-mass;
  particleGun->SetParticleEnergy(Ekin);
  G4double angle = (G4UniformRand()-0.5)*sigmaAngle;
  particleGun->SetParticleMomentumDirection
           (G4ThreeVector(sin(angle),0.,cos(angle)));
  particleGun->GeneratePrimaryVertex(anEvent);
}
```

- You can repeat this for generating more than one primary particles.

Primary vertices and primary particles

- Primary vertices and primary particles are stored in G4Event in advance to processing an event.
  - G4PrimaryVertex and G4PrimaryParticle classes
  - These classes don't have any dependency to G4ParticleDefinition nor G4Track.
  - They will become "primary tracks" only at Begin-of-Event phase and put into a "stack"



G4ParticleGun

- Concrete implementations of G4VPrimaryGenerator
  - A good example for experiment-specific primary generator implementation
- It shoots one primary particle of a certain energy from a certain point at a certain time to a certain direction.
  - Various C++ set methods are available
- Intercoms commands are also available for setting initial values

/gun/List	List available particles
/gun/particle	Set particle to be generated
/gun/direction	Set momentum direction
/gun/energy	Set kinetic energy
/gun/momentum	Set momentum
/gun/momentumAmp	Set absolute value of momentum
/gun/position	Set starting position of the particle
/gun/time	Set initial time of the particle
/gun/polarization	Set polarization
/gun/number	Set number of particles to be generated (per event)
/gun/ion	Set properties of ion to be generated [usage] /gun/ion Z A Q

Motivation for GPS

- After first simple tutorial trials, modelling sources in realistic set-up soon requires relatively more complex sources

- G4ParticleGun can be used in most cases

  (as in the series of examples during this tutorial), but
  - users still needs to code (C++) almost every change and
  - add related UI commands for interactive control

- Requirements for advanced primary particle modelling are often common to many users in different communities
  - E.g. uniform vertex distribution on a surface, isotropic generation, energy spectrum,…

What is GPS?

- The General Particle Source (GPS) offers as pre-defined many common options for particle generation (energy, angular and spatial distributions)
  - GPS is a concrete implementation of G4VPrimaryGenerator (as G4ParticleGun but more advanced)
  - G4 class name: G4GeneralParticleSource (in the event category)
- User cases: space radiation environment, medical physics, accelerator (fixed target)
- First development (2000) University of Southampton (ESA contract), maintained and upgraded now mainly by QinetiQ and ESA

G4GeneralParticleSource

- A concrete implementation of G4VPrimaryGenerator
  - Suitable especially to space applications

```
MyPrimaryGeneratorAction::
          MyPrimaryGeneratorAction()
{ generator = new G4GeneralParticleSource; }
void MyPrimaryGeneratorAction::
          GeneratePrimaries(G4Event* anEvent)
{ generator->GeneratePrimaryVertex(anEvent); }
```

- Detailed description

  Geant4 GPS manual

  LXR code browser (expgps)

Summary of GPS features

- Primary vertex can be randomly positioned with several options
  - Emission from point, plane,…
- Angular emission
  - Several distributions; isotropic, cosine-law, focused, …
  - With some additional parameters (min/max-theta, min/max-phi,…)
- Kinetic energy of the primary particle can also be randomized.
  - Common options (e.g. mono-energetic, power-law), some extra shapes (e.g. black-body) or user defined
- Multiple sources
  - With user defined relative intensity
- Capability of event biasing (variance reduction).
  - By enhancing particle type, distribution of vertex point, energy and/or direction

G4GeneralParticleSource (GPS)

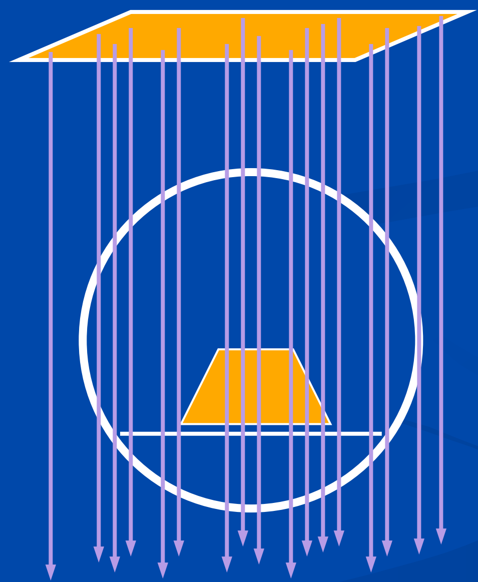
- An advanced concrete implementation of G4VPrimaryGenerator
- Offers as pre-defined many common (and not so common) options
  - Position, angular and energy distributions
  - Multiple sources, with user defined relative intensity
- Capability of event biasing (variance reduction).
- All features can be used via C++ or command line (or macro) UI

Example:
Proton source

- Vertices on rectangle along **xz** at edge of World
- Parallel emission along **-y**
- Monoenergetic: **500 MeV**

- Macro

```
/gps/particle proton

/gps/ene/type Mono
/gps/ene/mono 500 MeV

/gps/pos/type Plane
/gps/pos/shape Rectangle
/gps/pos/rot1 0 0 1
/gps/pos/rot2 1 0 0
/gps/pos/halfx 46.2 cm
/gps/pos/halfy 57.2 cm
/gps/pos/centre 0. 57.2 0. cm

/gps/direction 0 -1 0
```



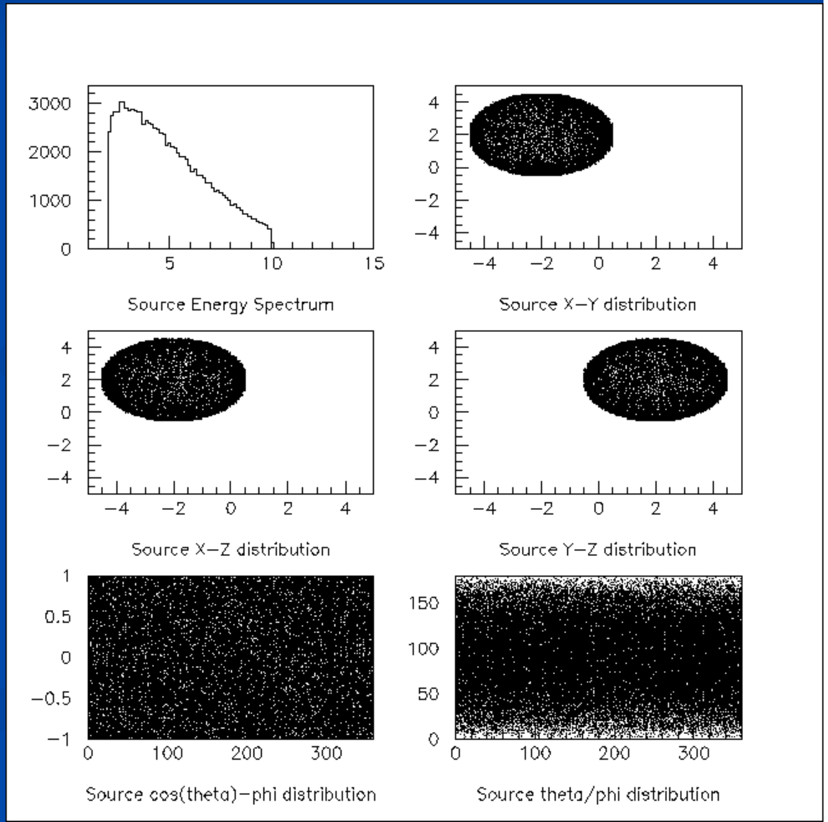
GPS

Example 6

- Vertex on sphere surface
- Isotropic emission
- Pre-defined spectrum (black-body)

- Macro

```
/gps/particle geantino

/gps/pos/type Surface
/gps/pos/shape Sphere
/gps/pos/centre -2. 2. 2. cm
/gps/pos/radius 2.5 cm

/gps/ang/type iso

/gps/ene/type Bbody
/gps/ene/min 2. MeV
/gps/ene/max 10. MeV
/gps/ene/temp 2e10
/gps/ene/calculate
```



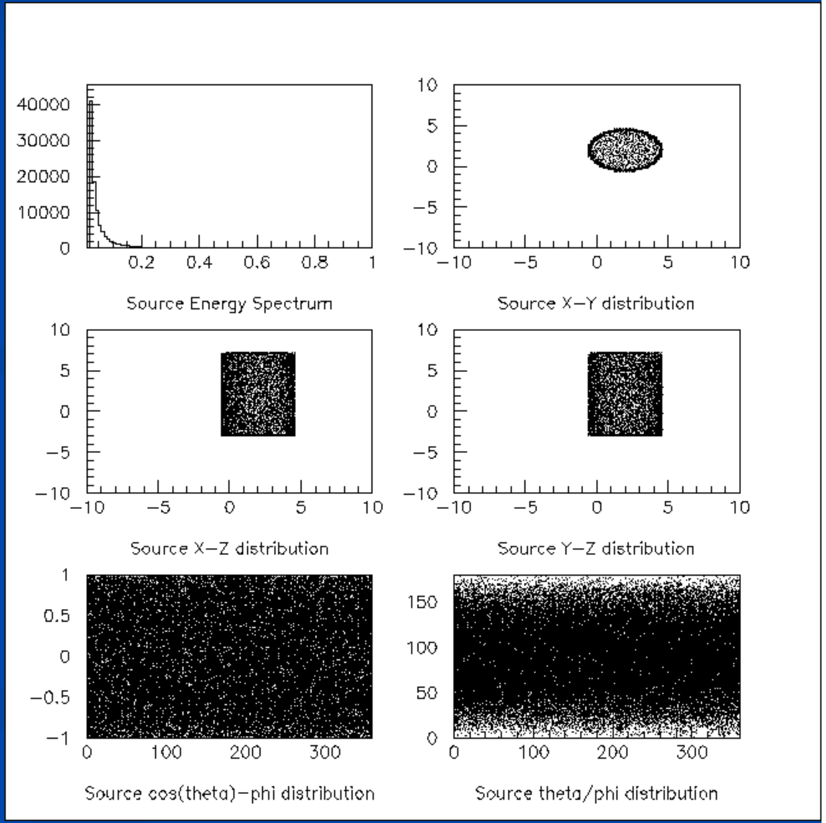
GPS

Example 7

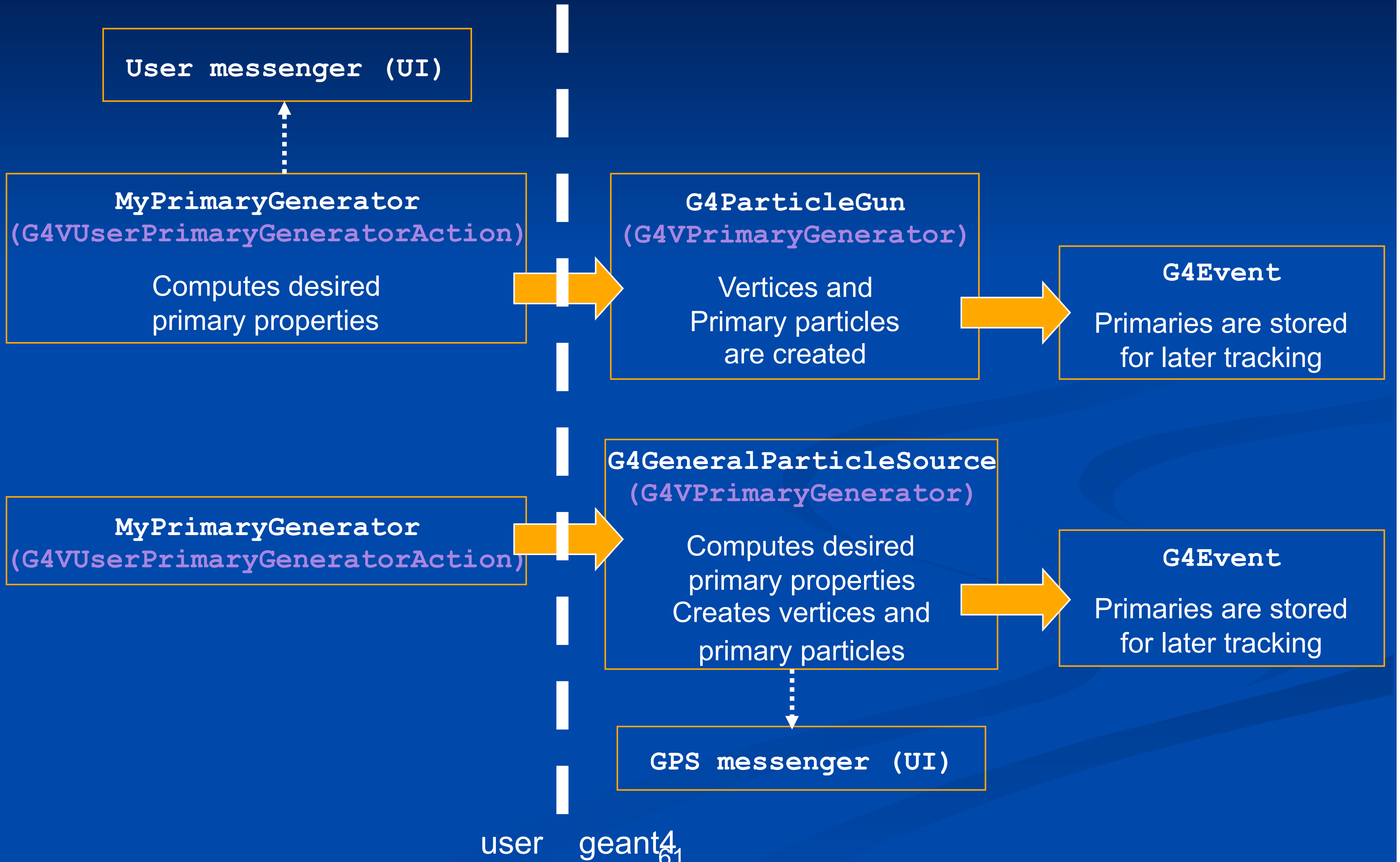
- Vertex on cylinder surface
- Cosine-law emission
  (to mimic isotropic source in space)
- Pre-defined spectrum
  (Cosmic Diffuse Gamma)

- Macro

```
/gps/particle gamma

/gps/pos/type Surface
/gps/pos/shape Cylinder
/gps/pos/centre 2. 2. 2. cm
/gps/pos/radius 2.5 cm
/gps/pos/halfz 5. cm

/gps/ang/type cos

/gps/ene/type Cdg
/gps/ene/min 20. keV
/gps/ene/max 1. MeV
/gps/ene/calculate
```



GPS vs G4ParticleGun



GPS control: scripting UI

- All features can be used via C++ or command line (or macro) UI
- Example of isotropic emission in UserPrimaryGenerator C++ code:

examples/advanced/human_phantom/src/G4HumanPhantomPrimaryGeneratorAction.cc

```
G4double a,b,c;
G4double n;
do {
  a = (G4UniformRand()-0.5)/0.5;
  b = (G4UniformRand()-0.5)/0.5;
  c = (G4UniformRand()-0.5)/0.5;
  n = a*a+b*b+c*c;
} while (n > 1 || n == 0.0);
n = std::sqrt(n);
a /= n;
b /= n;
c /= n;
G4ThreeVector direction(a,b,c);
particleGun->SetParticleMomentumDirection(direction);
```

- Equivalent GPS (script)

```
/gps/ang/type iso
```

Position distributions /gps/pos/…

- Point

  E.g.
  ```
  /gps/pos/type Point
  /gps/pos/centre 0. 0. 0. cm
  ```

- Beam

  E.g.
  ```
  /gps/pos/type Beam
  /gps/pos/shape Circle
  /gps/pos/radius 1. mm
  /gps/pos/sigma_r 2. mm
  ```

- Plane
  - Shape: Circle, Annulus, Ellipsoid, Square or Rectangle

  E.g.
  ```
  /gps/pos/type Plane
  /gps/pos/shape Rectangle
  /gps/pos/halfx 50 cm
  /gps/pos/halfy 70 cm
  ```

- Surface or Volume
  - Shape: Sphere, Ellipsoid, Cylinder or Para
  - Surface: zenith automatically oriented as normal to surface at point

  E.g.
  ```
  /gps/pos/type Surface
  /gps/pos/shape Sphere
  /gps/pos/radius 1. m
  ```

Position distributions
`/gps/pos/...` (2)

- Some shared UI commands

- `/gps/pos/centre`
- `/gps/pos/halfx | y | z`
- `/gps/pos/radius`
- `/gps/pos/inner_radius`
- `/gps/pos/sigmar`
- `/gps/pos/sigmax | y`
- `/gps/pos/rot1`
- `/gps/pos/rot2`

- When usinig Volume type, one can limit the emission from within a certain volume in the “mass” geometry

`/gps/pos/confine your_physical_volume_name`

Angular distributions /gps/ang/…

- Isotropic (**`iso`**)
- Cosine-law (**`cos`**)
  - See next slides for more information
- Planar wave (**`planar`**)
  - Standard emission in one direction
    (it's also implicitly set by **`/gps/direction x y z`**)
- Accelerator beam
  - 1-d or 2-d gaussian emission, **`beam1d`** or **`beam2d`**
- Focusing to a point (**`focused`**)
- User-defined (**`user`**)

Energy distributions
/gps/ene/…

Kinetic energy of the primary particle can also be randomized, with several predefined options:

- Common options (e.g. mono-energetic, power-law, exponential, gaussian, etc)
  - mono-energetic (**Mono**)
  - linear (**Lin**)
  - power-law (**Pow**)
  - exponential (**Exp**)
  - gaussian (**Gauss**)

- Some extra predefined spectral shapes (bremsstrahlung, black-body, cosmic diffuse gamma ray,…)
  - bremsstrahlung (**Brem**)
  - black-body (**Bbody**)
  - cosmic diffuse gamma ray (**Cdg**)

- User defined
  - user-defined histogram (**User**)
  - arbitrary point-wise spectrum (**Arb**) and
  - user-defined energy per nucleon histogram (**Epn**)

Multiple sources

- Definition of multiple "parallel" sources
- One source per event is used
- Sampling according to relative intensity

- First source is always already present (implicitly created)
  - one can add intensity information

```
/gps/source/intensity 5.
```

- Additional sources must be added explicitly

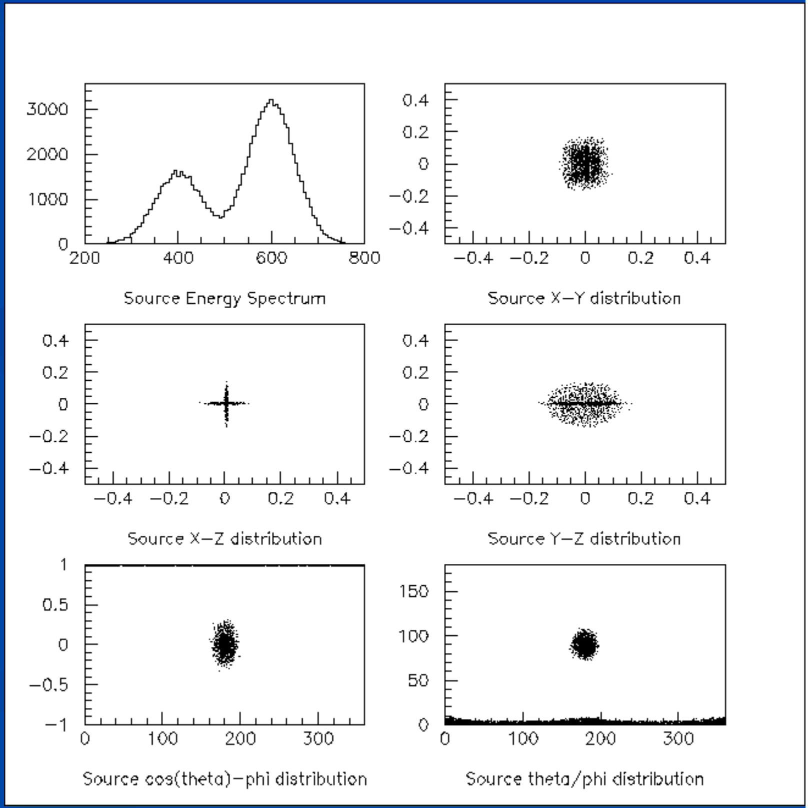
```
/gps/source/add 10.
```

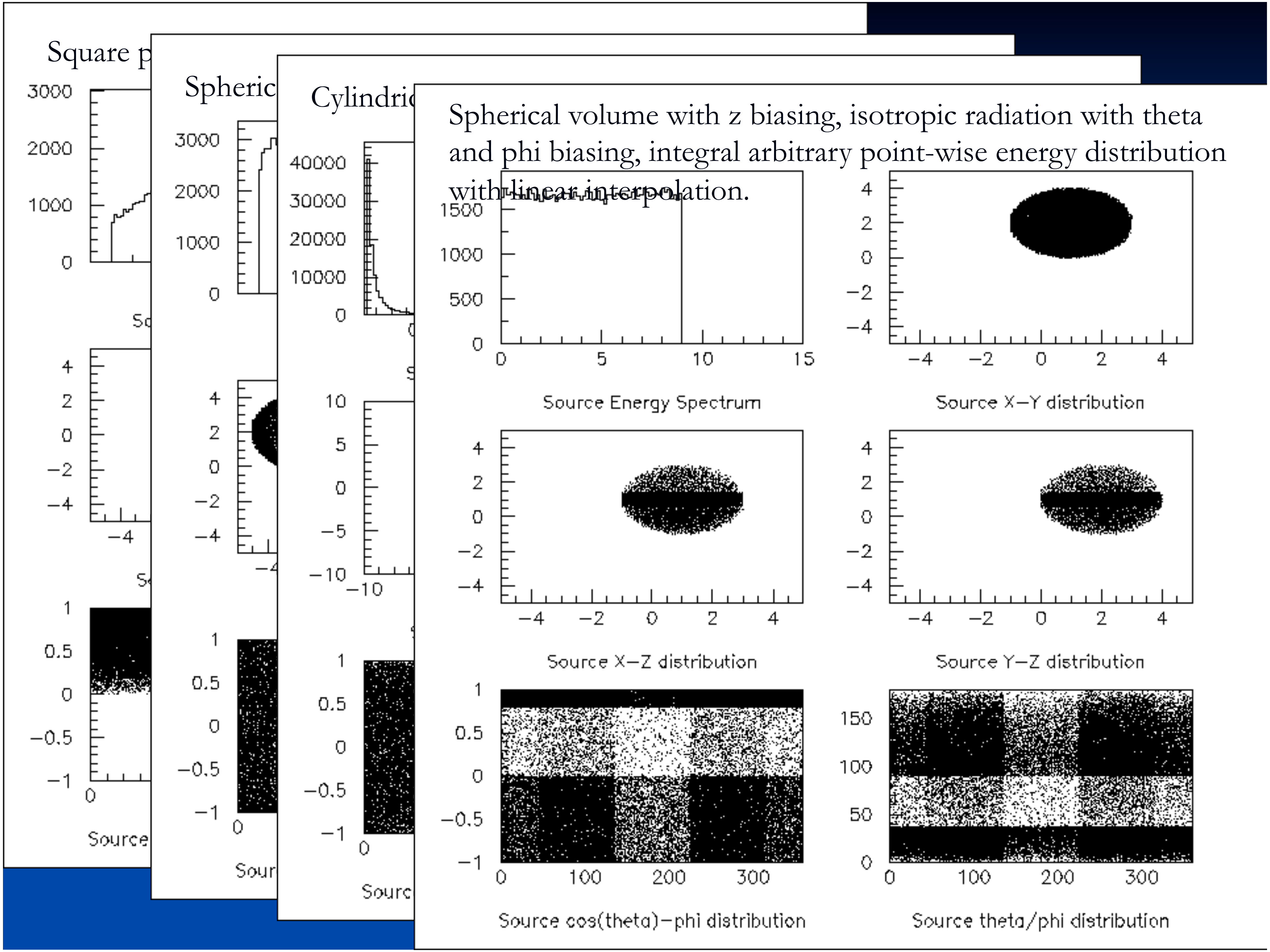
GPS
Example 31

- Two-beam source definition
  (multiple sources)
- Gaussian profile
- Can be focused / defocused

- Macro

```
# beam #1
    # default intensity is 1,
    # now change to 5.
    /gps/source/intensity 5.

    /gps/particle proton
    /gps/pos/type Beam

    # the incident surface is
    # in the y-z plane
    /gps/pos/rot1 0 1 0
    /gps/pos/rot2 0 0 1

    # the beam spot is centered
    # at the origin and is
    # of 1d gaussian shape
    # with a 1 mm central plateau
    /gps/pos/shape Circle
    /gps/pos/centre 0. 0. 0. mm
    /gps/pos/radius 1. mm
    /gps/pos/sigma_r .2 mm

    # the beam is travelling
    # along the X_axis
    # with 5 degrees dispersion
    /gps/ang/rot1 0 0 1
    /gps/ang/rot2 0 1 0
    /gps/ang/type beam1d
    /gps/ang/sigma_r 5. deg

    # the beam energy is in
    # gaussian profile centered
    # at 400 MeV
    /gps/ene/type Gauss
    /gps/ene/mono 400 MeV
    /gps/ene/sigma 50. MeV

# beam #2
    # 2x the instensity of beam #1
    /gps/source/add 10.

    # this is a electron beam
    ...
```




Square p
Spheric
Cylindric
Spherical volume with z biasing, isotropic radiation with theta and phi biasing, integral arbitrary point-wise energy distribution with linear interpolation.
Source Energy Spectrum
Source X–Y distribution
Source X–Z distribution
Source Y–Z distribution
Source cos(theta)–phi distribution
Source theta/phi distribution