The *Simulation à Grande Vitesse* (SGV) Fast Simulation program

Mikael Berggren[1]

[1]DESY, Hamburg

LCWS, Sendai, Japan , Oct 2019

Outline

The need for fast simulation

- We have very good full simulation now.
- So why bother about fast simulation ?
- Answer:
  - R. Heuer at LCWS 2011: *We need to update the physics case continuously*.
  - Light-weight: run anywhere, no need to read tons of manuals and doxygen pages.
  - Anyhow, experience from both LOI and DBD showed that for physics, the FastSim studies often were good enough, if the FastSim has enough detail.

But most of all:

Fast simulation is Fast !

So...

Why do we need speed ?

The need for fast simulation

- We have very good full simulation now.
- So why bother about fast simulation ?
- Answer:
  - R. Heuer at LCWS 2011: *We need to update the physics case continuously*.
  - Light-weight: run anywhere, no need to read tons of manuals and doxygen pages.
  - Anyhow, experience from both LOI and DBD showed that for physics, the FastSim studies often were good enough, if the FastSim has enough detail.

But most of all:

Fast simulation is Fast !

So...

Why do we need speed ?

The need for fast simulation

- We have very good full simulation now.
- So why bother about fast simulation ?
- Answer:
  - R. Heuer at LCWS 2011: *We need to update the physics case continuously*.
  - Light-weight: run anywhere, no need to read tons of manuals and doxygen pages.
  - Anyhow, experience from both LOI and DBD showed that for physics, the FastSim studies often were good enough, if the FastSim has enough detail.

But most of all:

Fast simulation is Fast !

So...

Why do we need speed ?

The need for fast simulation

- We have very good full simulation now.
- So why bother about fast simulation ?
- Answer:
  - R. Heuer at LCWS 2011: *We need to update the physics case continuously*.
  - Light-weight: run anywhere, no need to read tons of manuals and doxygen pages.
  - Anyhow, experience from both LOI and DBD showed that for physics, the FastSim studies often were good enough, if the FastSim has enough detail.

But most of all:

Fast simulation is Fast !

So...

Why do we need speed ?

The need for fast simulation

- We have very good full simulation now.
- So why bother about fast simulation ?
- Answer:
  - R. Heuer at LCWS 2011: *We need to update the physics case continuously*.
  - Light-weight: run anywhere, no need to read tons of manuals and doxygen pages.
  - Anyhow, experience from both LOI and DBD showed that for physics, the FastSim studies often were good enough, if the FastSim has enough detail.

But most of all:

Fast simulation is Fast !

So...

Why do we need speed ?

$\gamma\gamma$ background

Total cross-section for $e^+e^- \to \gamma\gamma e^+e^- \to q\bar{q}e^+e^-$: 35 nb (PYTHIA)

- $\int \mathcal{L}dt = 500$ fb^{-1} $\to$ 18 $\star 10^9$ events are expected.
- 10 ms to generate one event.
- 10 ms to fastsim (SGV) one event.

10^8 s of CPU time is needed, ie more than 3 years. But:This goes to 3000 years with full simulation.

$\gamma\gamma$ background

Total cross-section for $e^+e^- \to \gamma\gamma e^+e^- \to q\bar{q}e^+e^-$: 35 nb (PYTHIA)

- $\int \mathcal{L}dt = 500$ fb^{-1} $\to$ 18 $\star 10^9$ events are expected.
- 10 ms to generate one event.
- 10 ms to fastsim (SGV) one event.

10^8 s of CPU time is needed, ie more than 3 years. But:This goes to 3000 years with full simulation.

$\gamma\gamma$ background

Total cross-section for $e^+e^- \to \gamma\gamma e^+e^- \to q\bar{q}e^+e^-$: 35 nb (PYTHIA)

- $\int \mathcal{L}dt = 500$ fb^{-1} $\to$ 18 $\star 10^9$ events are expected.
- 10 ms to generate one event.
- 10 ms to fastsim (SGV) one event.

10^8 s of CPU time is needed, ie more than 3 years. But:This goes to 3000 years with full simulation.

SUSY parameter scans

Simple example:

- MSUGRA: 4 parameters + sign of μ
- Scan each in eg. 20 steps
- Eg. 5000 events per point (modest requirement: in sps1a' almost 1 million SUSY events are expected for 500 fb^{-1} !)
- = $20^4 \times 2 \times 5000 = 1.6 \times 10^9$ events to generate...

Slower to generate and simulate than $\gamma\gamma$ events

Also here: CPU millenniums with full simulation

SUSY parameter scans

Simple example:

- MSUGRA: 4 parameters + sign of μ
- Scan each in eg. 20 steps
- Eg. 5000 events per point (modest requirement: in sps1a' almost 1 million SUSY events are expected for 500 fb^{-1} !)
- = $20^4 \times 2 \times 5000 = 1.6 \times 10^9$ events to generate...

Slower to generate and simulate than $\gamma\gamma$ events

Also here: CPU millenniums with full simulation

Fast simulation types, and the choice for ILC

Different types, with increasing level of sophistication:

- 4-vector smearing.
- Parametric. Eg.: Delphes
- Covariance matrix machines. Eg.: SGV

Common for all:

Detector simulation time $\approx$ time to generate event by an efficient generator like PYTHIA 6

For ILC:

Only Covariance matrix machines have sufficient detail. Here, I'll cover "la Simulation à Grande Vitesse", SGV.

Fast simulation types, and the choice for ILC

Different types, with increasing level of sophistication:

- 4-vector smearing.
- Parametric. Eg.: Delphes
- Covariance matrix machines. Eg.: SGV

Common for all:

Detector simulation time $\approx$ time to generate event by an efficient generator like PYTHIA 6

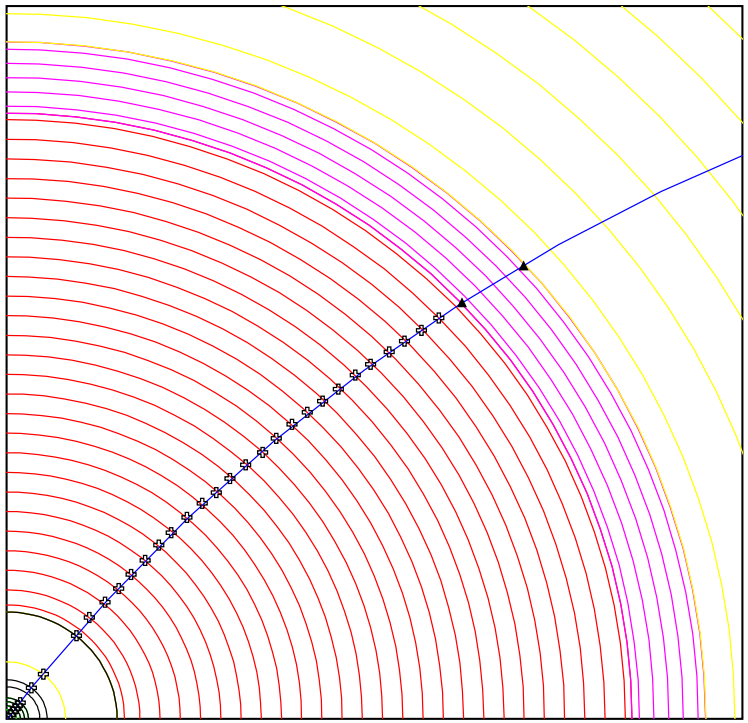
For ILC:

Only Covariance matrix machines have sufficient detail. Here, I'll cover "la Simulation à Grande Vitesse", SGV.

SGV: How tracking works

SGV is a machine to calculate covariance matrices

Tracking: Follow track-helix through the detector.

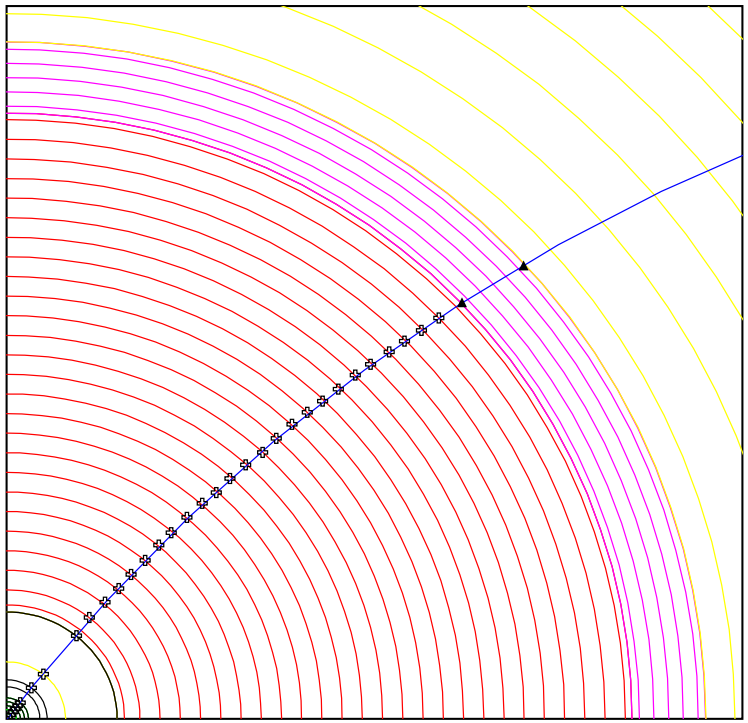


- Calculate cov. mat. at perigee, including material, measurement errors and extrapolation. NB: this is exactly what Your Kalman filter does!
- Smear perigee parameters (Choleski decomposition: takes all correlations into account)
- Helix *parameters* exactly calculated, *errors* with one approximation: helix moved to (0,0,0) for this.

SGV: How tracking works

SGV is a machine to calculate covariance matrices

Tracking: Follow track-helix through the detector.

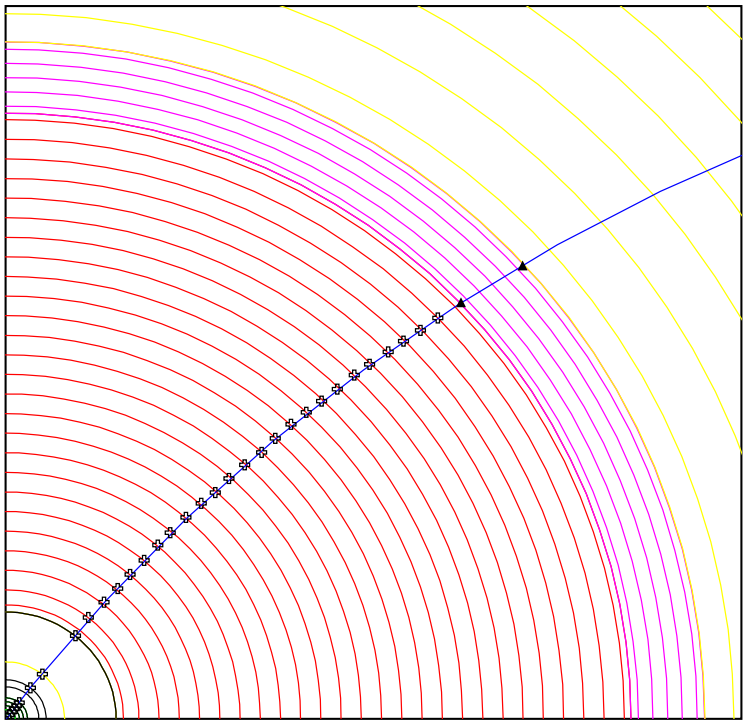


- Calculate cov. mat. at perigee, including material, measurement errors and extrapolation. NB: this is exactly what Your Kalman filter does!
- Smear perigee parameters (Choleski decomposition: takes all correlations into account)
- Helix *parameters* exactly calculated, *errors* with one approximation: helix moved to (0,0,0) for this.

SGV: How tracking works

SGV is a machine to calculate covariance matrices

Tracking: Follow track-helix through the detector.

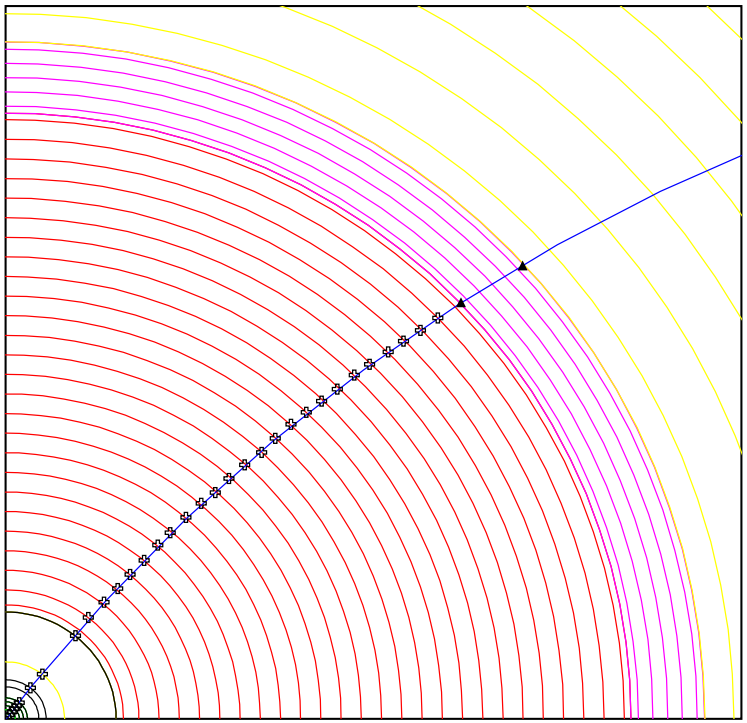


- Calculate cov. mat. at perigee, including material, measurement errors and extrapolation. NB: this is exactly what Your Kalman filter does!
- Smear perigee parameters (Choleski decomposition: takes all correlations into account)
- Helix *parameters* exactly calculated, *errors* with one approximation: helix moved to (0,0,0) for this.

SGV: How tracking works

SGV is a machine to calculate covariance matrices

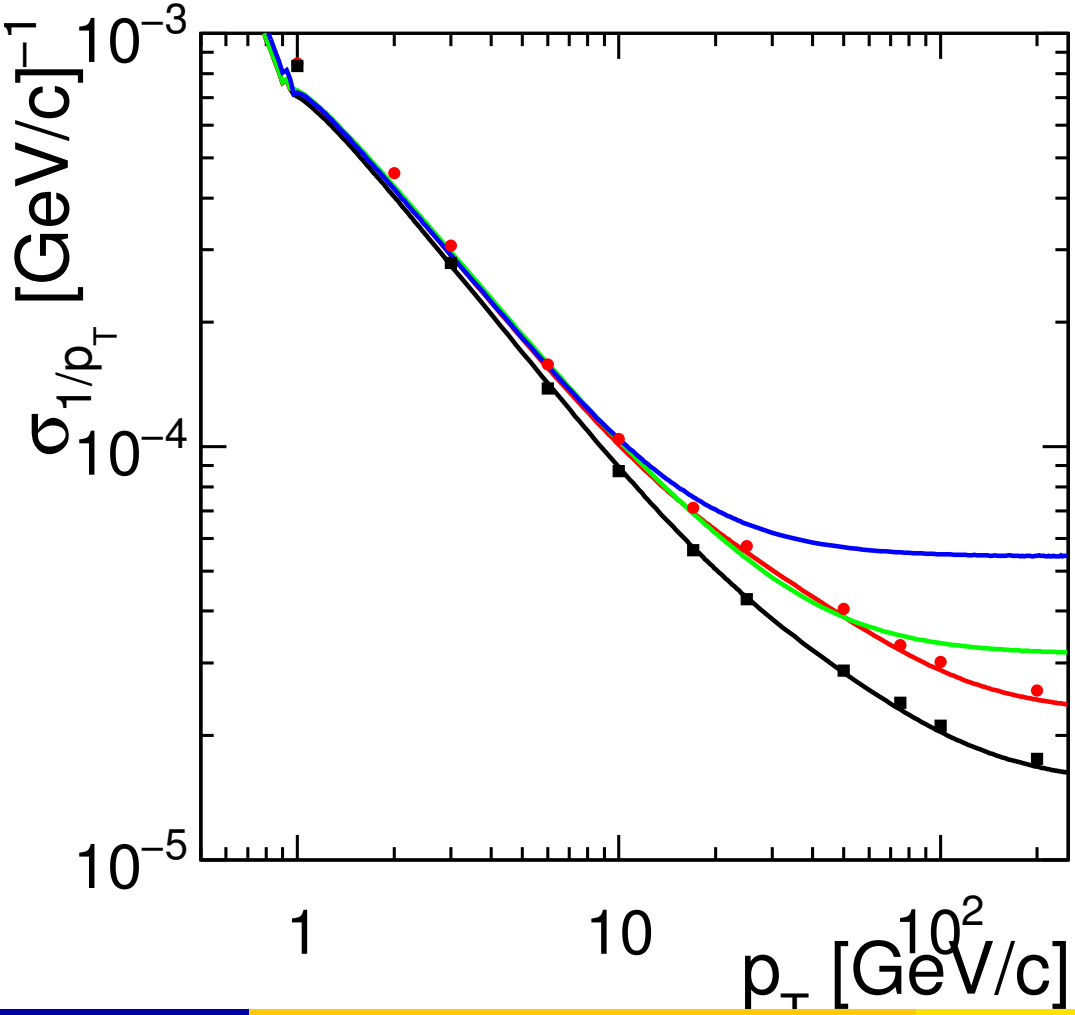
Tracking: Follow track-helix through the detector.



- Calculate cov. mat. at perigee, including material, measurement errors and extrapolation. NB: this is exactly what Your Kalman filter does!
- Smear perigee parameters (Choleski decomposition: takes all correlations into account)
- Helix *parameters* exactly calculated, *errors* with one approximation: helix moved to (0,0,0) for this.

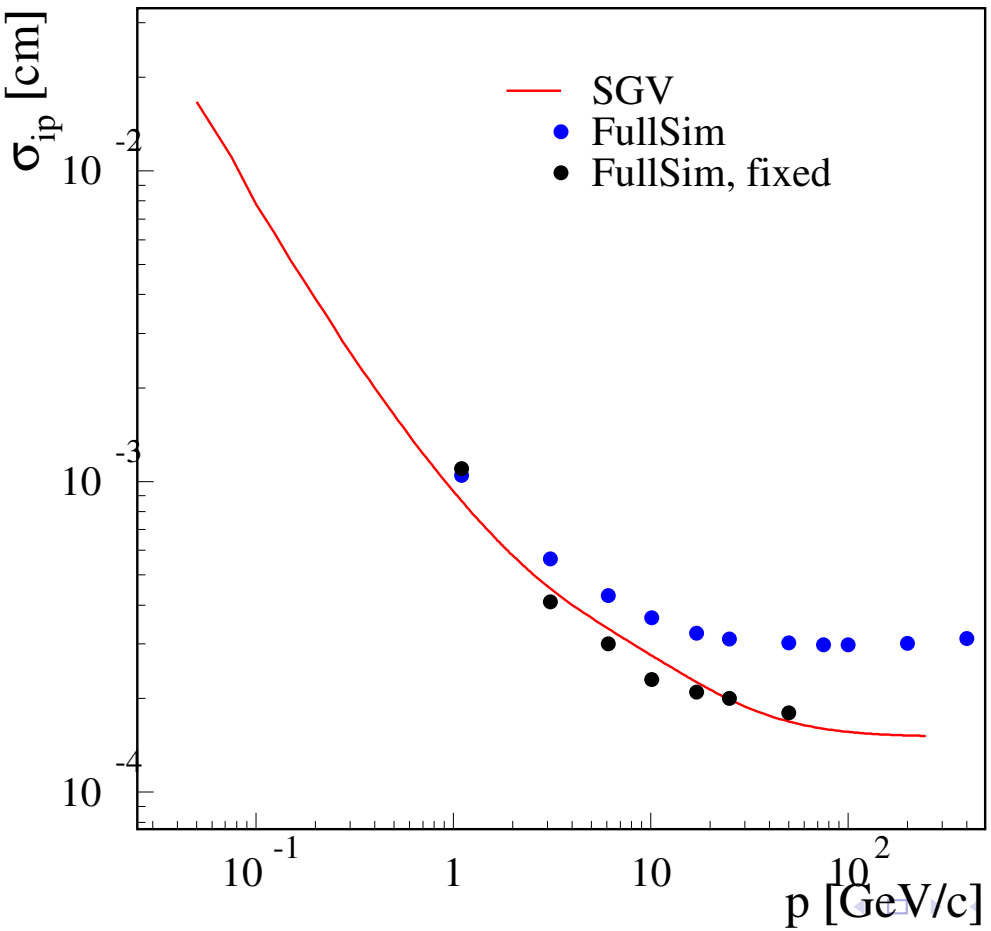
SGV and FullSim LDC/ILD: momentum resolution

Lines: SGV, dots: Mokka+Marlin



SGV and FullSim LDC/ILD: ip resolution vs P

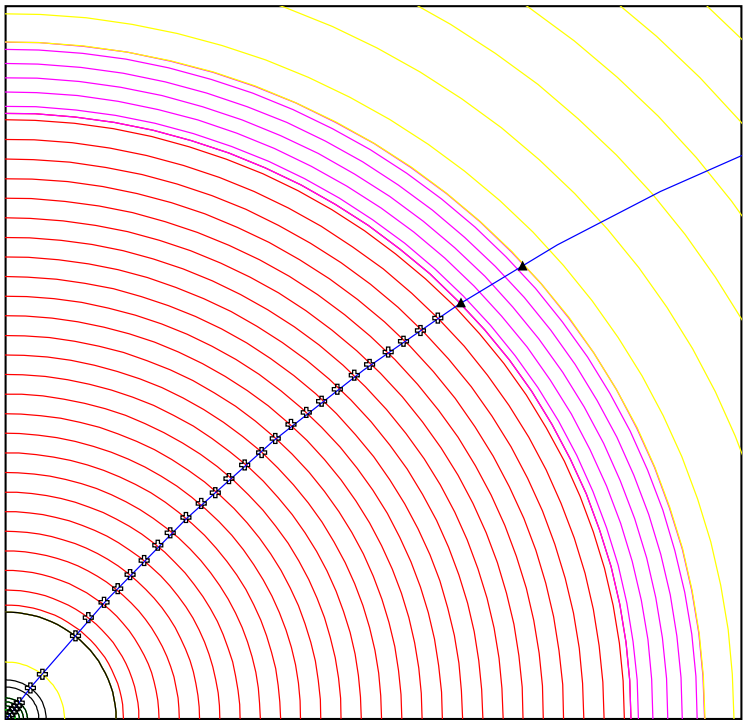
Lines: SGV, dots: Mokka+Marlin



SGV: How the rest works

SGV is a machine to calculate covariance matrices

Calorimeters: Follow particle to intersection with calorimeters.



- Response type: MIP, EM or hadronic shower, below threshold, etc.
- Simulate single particle response from parameters.
- Easy to plug in more sophisticated shower-simulation. Next slides.

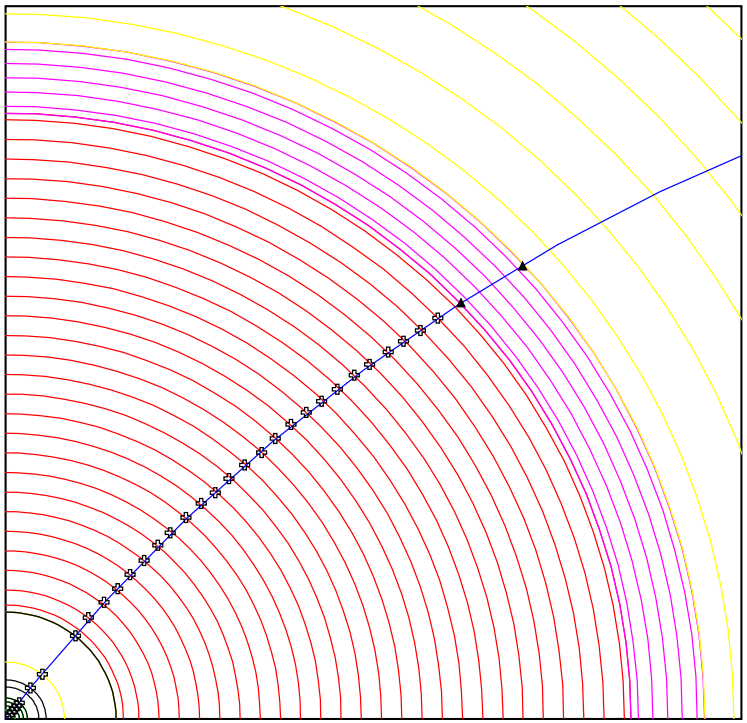
Other stuff:

- EM-interactions in detector material simulated
- Plug-ins for particle identification, track-finding efficiencies,...
- Information on hit-patterns accessible to analysis.

SGV: How the rest works

SGV is a machine to calculate covariance matrices

Calorimeters: Follow particle to intersection with calorimeters.



- Response type: MIP, EM or hadronic shower, below threshold, etc.
- Simulate single particle response from parameters.
- Easy to plug in more sophisticated shower-simulation. Next slides.

Other stuff:

- EM-interactions in detector material simulated
- Plug-ins for particle identification, track-finding efficiencies,...
- Information on hit-patterns accessible to analysis.

Something about vertex fits and all that

- User data, delivered in Module-global arrays:
  - Extended 4-vectors .
  - Track helix parameters with correlations.
  - Calorimetric clusters.
  - When relevant: true values.
  - Auxiliary information on particle history, detector-elements used etc.
  - Event-global variables.
- User Analysis tasks :
  - Jet-finding.
  - Event-shapes.
  - Primary and secondary vertex fitting.
  - Impact parameters.

  Can be calculated by routines, included in SGV. Access routines give an easy interface to the detector geometry.

Something about vertex fits and all that

- User data, delivered in Module-global arrays:
  - Extended 4-vectors .
  - Track helix parameters with correlations.
  - Calorimetric clusters.
  - When relevant: true values.
  - Auxiliary information on particle history, detector-elements used etc.
  - Event-global variables.
- User Analysis tasks :
  - Jet-finding.
  - Event-shapes.
  - Primary and secondary vertex fitting.
  - Impact parameters.

  Can be calculated by routines, included in SGV. Access routines give an easy interface to the detector geometry.

Calorimeter simulation

The issues:

- Clearly: Random E, shower position, shower shape.
- But also association errors:
  - Clusters might merge,split, or get wrongly associated to tracks
- Will depend on Energy, on distance to neighbour, on EM or hadronic, on Barrel or forward, ...
- Consequences:
  - If a (part of) a neutral cluster associated to track → Energy is lost.
  - If a (part of) a charged cluster not associated to any track → Energy is double-counted.
- Parametrisation:
  - Look at how PFA on FullSim has associated tracks and clusters: link MCParticle -> Track and/or true cluster -> Seen cluster.
  - Found that sets of p.d.f.'s with 28 parameters × 4 cases (em/had × double-counting/loss) can do this.

Calorimeter simulation

The issues:

- Clearly: Random E, shower position, shower shape.
- But also association errors:
  - Clusters might merge,split, or get wrongly associated to tracks
- Will depend on Energy, on distance to neighbour, on EM or hadronic, on Barrel or forward, ...
- Consequences:
  - If a (part of) a neutral cluster associated to track → Energy is lost.
  - If a (part of) a charged cluster not associated to any track → Energy is double-counted.
- Parametrisation:
  - Look at how PFA on FullSim has associated tracks and clusters: link MCParticle -> Track and/or true cluster -> Seen cluster.
  - Found that sets of p.d.f.'s with 28 parameters × 4 cases (em/had × double-counting/loss) can do this.

Calorimeter simulation

The issues:

- Clearly: Random E, shower position, shower shape.
- But also association errors:
  - Clusters might merge,split, or get wrongly associated to tracks
- Will depend on Energy, on distance to neighbour, on EM or hadronic, on Barrel or forward, ...
- Consequences:
  - If a (part of) a neutral cluster associated to track $\rightarrow$ Energy is lost.
  - If a (part of) a charged cluster not associated to any track $\rightarrow$ Energy is double-counted.
- Parametrisation:
  - Look at how PFA on FullSim has associated tracks and clusters: link MCParticle -> Track and/or true cluster -> Seen cluster.
  - Found that sets of p.d.f.'s with 28 parameters $\times$ 4 cases (em/had $\times$ double-counting/loss) can do this.

Calorimeter simulation

The issues:

- Clearly: Random E, shower position, shower shape.
- But also association errors:
  - Clusters might merge,split, or get wrongly associated to tracks
- Will depend on Energy, on distance to neighbour, on EM or hadronic, on Barrel or forward, ...
- Consequences:
  - If a (part of) a neutral cluster associated to track $\rightarrow$ Energy is lost.
  - If a (part of) a charged cluster not associated to any track $\rightarrow$ Energy is double-counted.
- Parametrisation:
  - Look at how PFA on FullSim has associated tracks and clusters: link MCParticle -> Track and/or true cluster -> Seen cluster.
  - Found that sets of p.d.f.'s with 28 parameters $\times$ 4 cases (em/had $\times$ double-counting/loss) can do this.

Proof of principle of the parametrisation

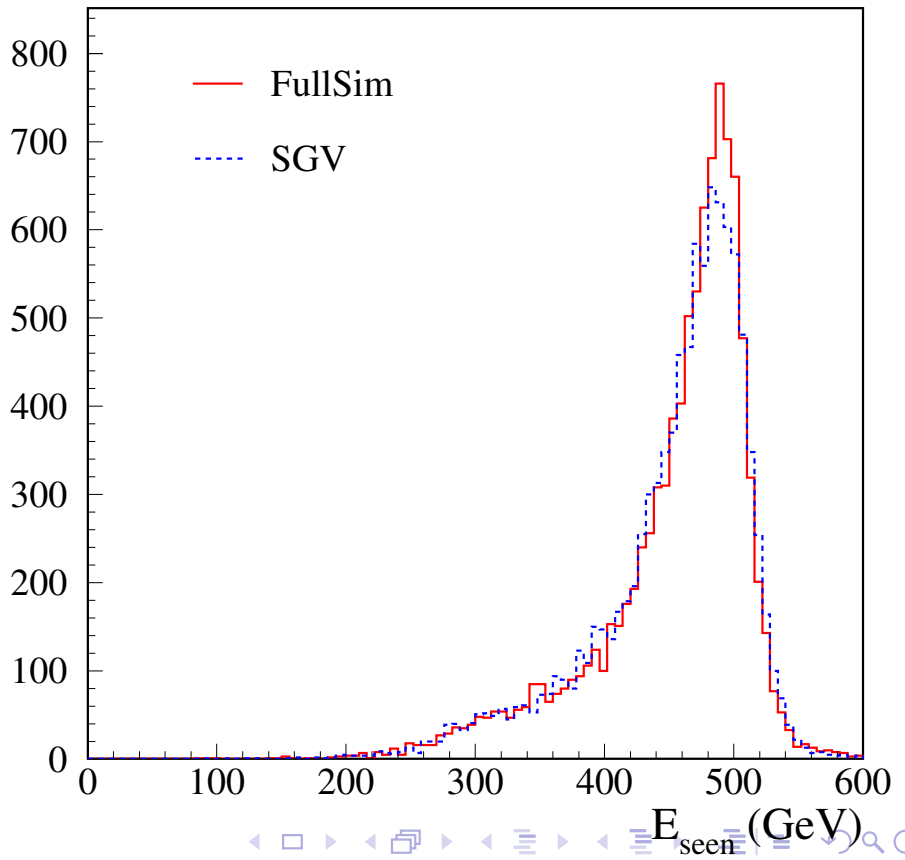
Feed exactly the same physics events through FullSim or SGV.

- Overall:
  - Total seen energy
- $e^+e^- \rightarrow ZZ \rightarrow$ four jets:
  - Reconstructed M_Z at different stages in FullSim.
  - Seen Reconstructed M_Z, FullSim and SGV.
  - Jet-Energy resoulution
- *Zhh* at 1 TeV:
  - Vissible E
  - Higgs Mass
  - b-tag

Proof of principle of the parametrisation

Feed exactly the same physics events through FullSim or SGV.

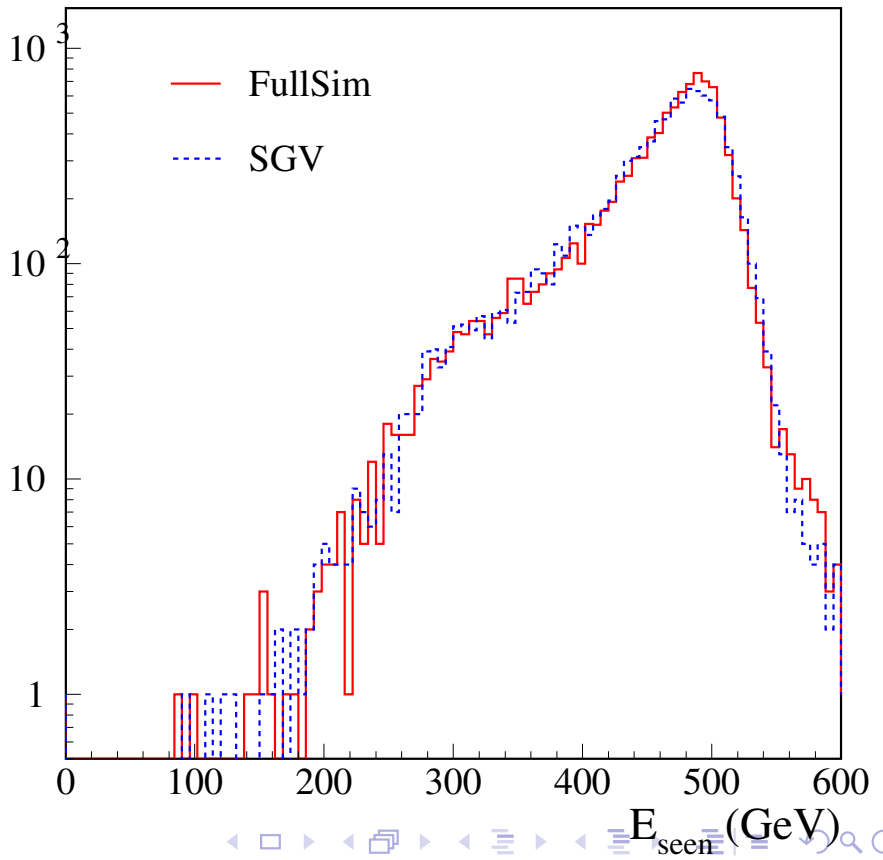
- Overall:
  - Total seen energy
- $e^+e^- \rightarrow ZZ \rightarrow$ four jets:
  - Reconstructed M_Z at different stages in FullSim.
  - Seen Reconstructed M_Z, FullSim and SGV.
  - Jet-Energy resoulution
- *Zhh* at 1 TeV:
  - Vissible E
  - Higgs Mass
  - b-tag



Proof of principle of the parametrisation

Feed exactly the same physics events through FullSim or SGV.

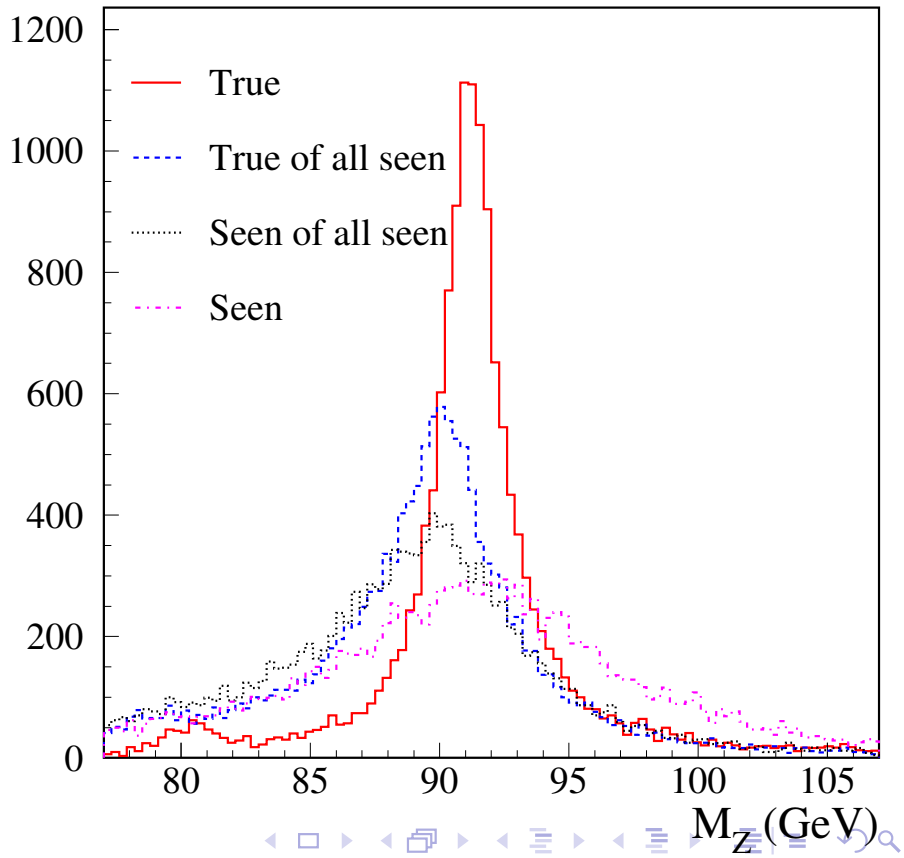
- Overall:
  - Total seen energy
- $e^+e^- \rightarrow ZZ \rightarrow$ four jets:
  - Reconstructed M_Z at different stages in FullSim.
  - Seen Reconstructed M_Z, FullSim and SGV.
  - Jet-Energy resoulution
- *Zhh* at 1 TeV:
  - Vissible E
  - Higgs Mass
  - b-tag



Proof of principle of the parametrisation

Feed exactly the same physics events through FullSim or SGV.

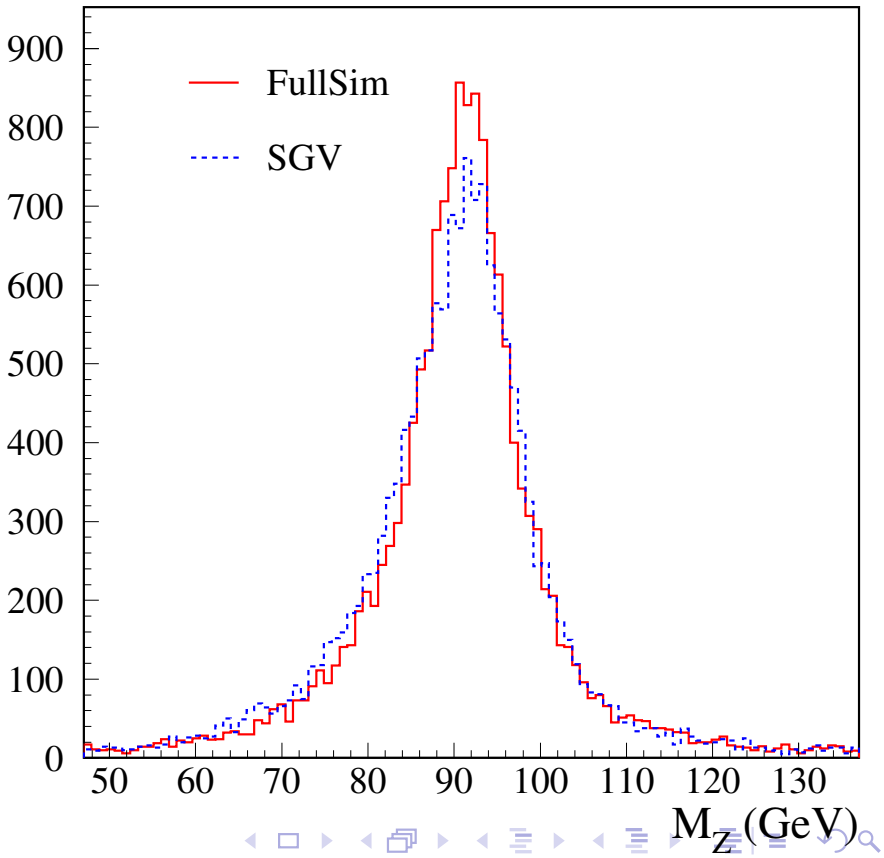
- Overall:
  - Total seen energy
- $e^+e^- \rightarrow ZZ \rightarrow$ four jets:
  - Reconstructed M_Z at different stages in FullSim.
  - Seen Reconstructed M_Z, FullSim and SGV.
  - Jet-Energy resoulution
- *Zhh* at 1 TeV:
  - Vissible E
  - Higgs Mass
  - b-tag



Proof of principle of the parametrisation

Feed exactly the same physics events through FullSim or SGV.

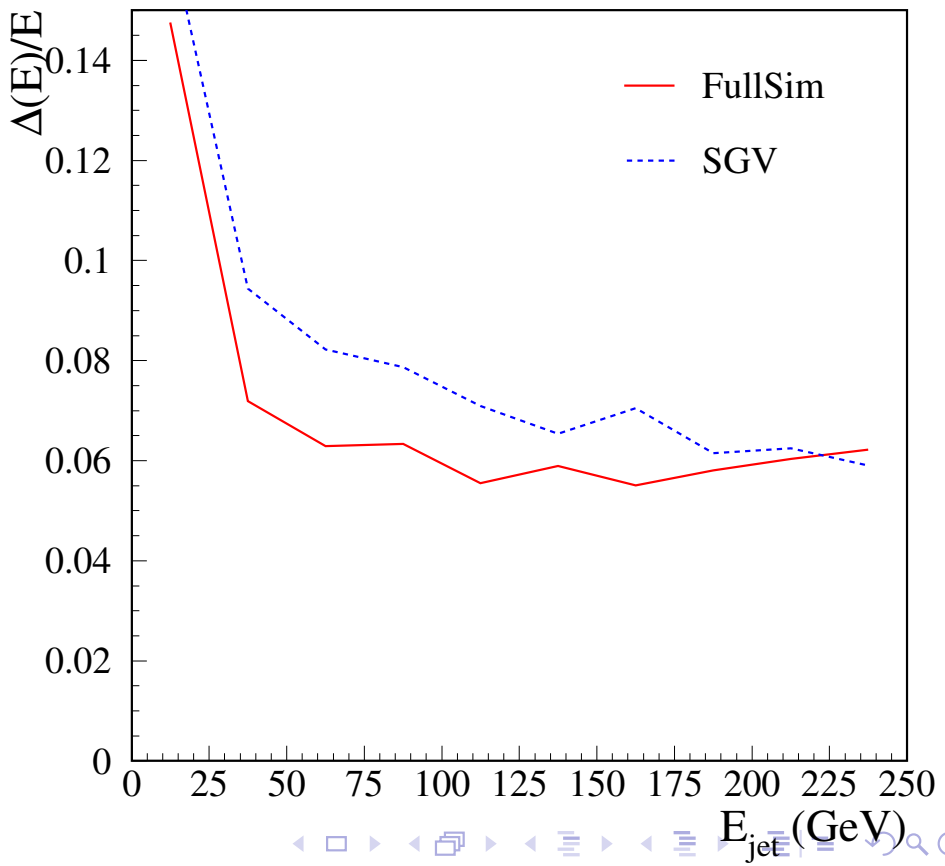
- Overall:
  - Total seen energy
- $e^+e^- \rightarrow ZZ \rightarrow$ four jets:
  - Reconstructed M_Z at different stages in FullSim.
  - Seen Reconstructed M_Z, FullSim and SGV.
  - Jet-Energy resoulution
- *Zhh* at 1 TeV:
  - Vissible E
  - Higgs Mass
  - b-tag



Proof of principle of the parametrisation

Feed exactly the same physics events through FullSim or SGV.

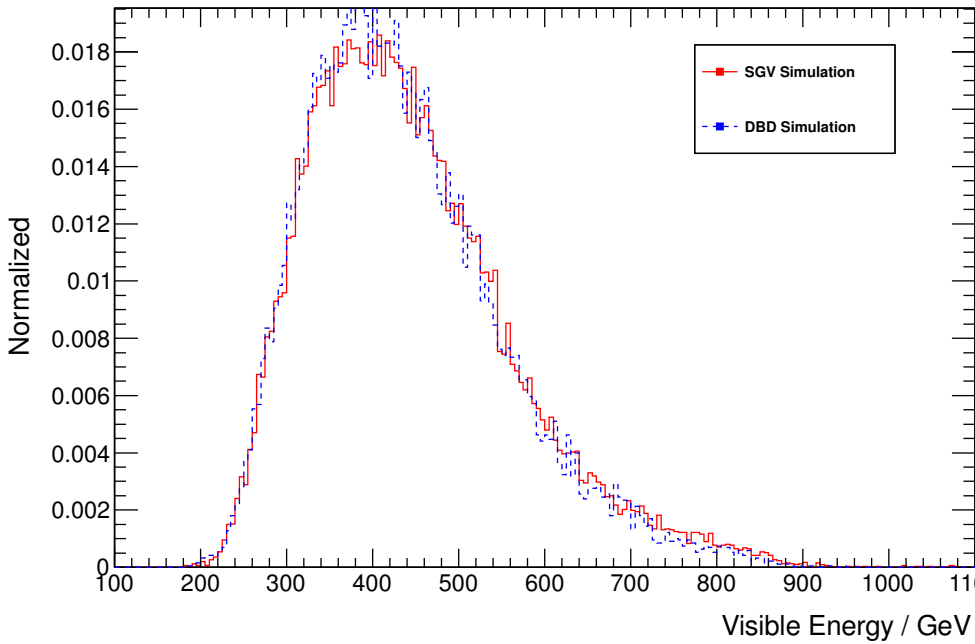
- Overall:
  - Total seen energy
- $e^+e^- \rightarrow ZZ \rightarrow$ four jets:
  - Reconstructed M_Z at different stages in FullSim.
  - Seen Reconstructed M_Z, FullSim and SGV.
  - Jet-Energy resoulution
- *Zhh* at 1 TeV:
  - Vissible E
  - Higgs Mass
  - b-tag



Proof of principle of the parametrisation

Feed exactly the same physics events through FullSim or SGV.

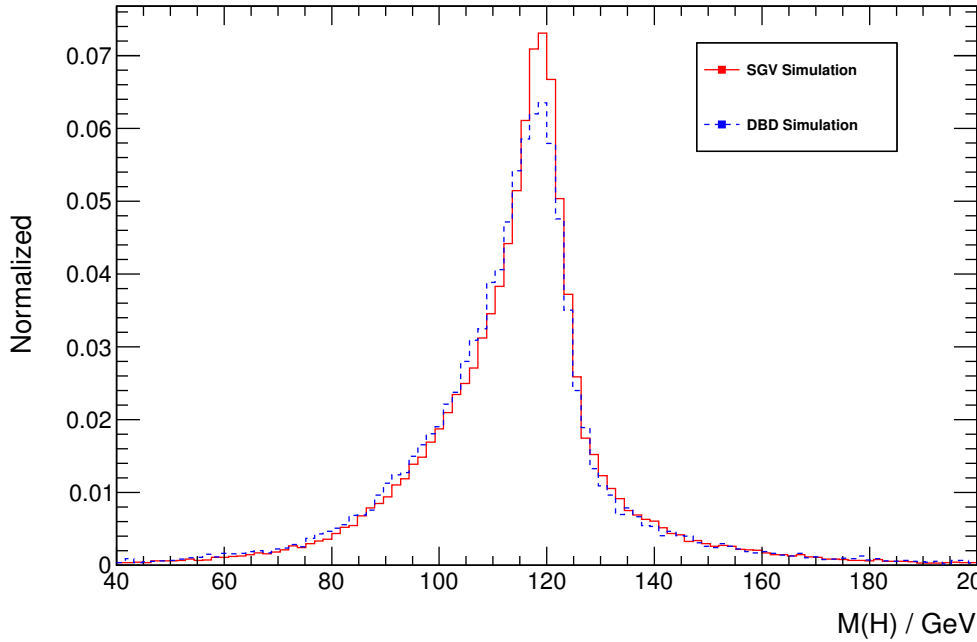
- Overall:
  - Total seen energy
- $e^+e^- \rightarrow ZZ \rightarrow$ four jets:
  - Reconstructed M_Z at different stages in FullSim.
  - Seen Reconstructed M_Z, FullSim and SGV.
  - Jet-Energy resoulution
- *Zhh* at 1 TeV:
  - Vissible E
  - Higgs Mass
  - b-tag



Proof of principle of the parametrisation

Feed exactly the same physics events through FullSim or SGV.

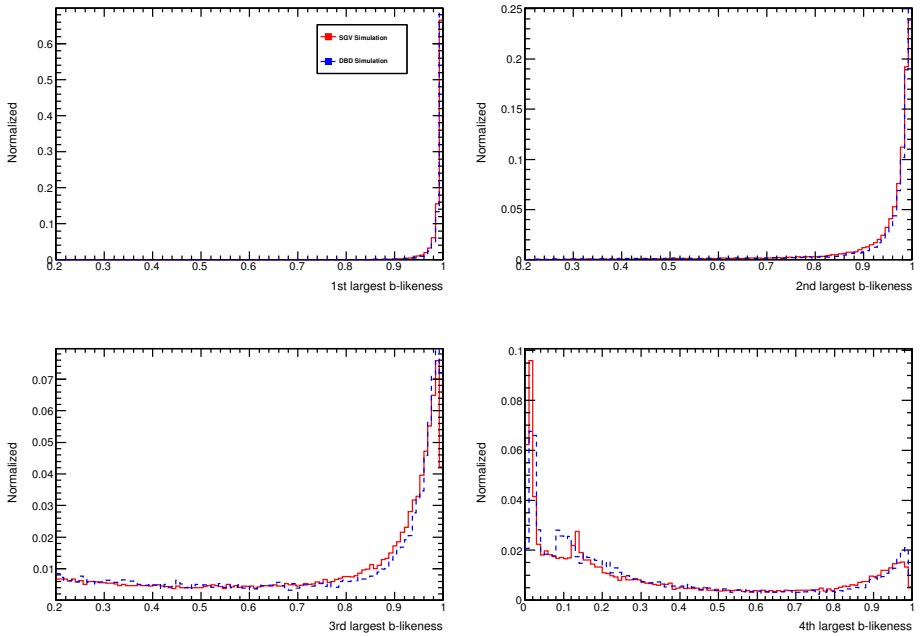
- Overall:
  - Total seen energy
- $e^+e^- \rightarrow ZZ \rightarrow$ four jets:
  - Reconstructed M_Z at different stages in FullSim.
  - Seen Reconstructed M_Z, FullSim and SGV.
  - Jet-Energy resoulution
- *Zhh* at 1 TeV:
  - Vissible E
  - Higgs Mass
  - b-tag



Proof of principle of the parametrisation

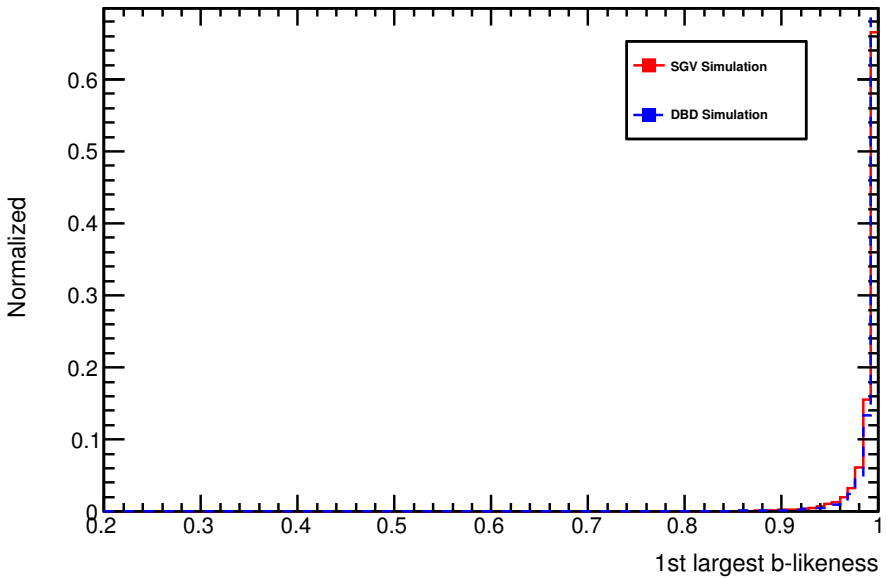
Feed exactly the same physics events through FullSim or SGV.

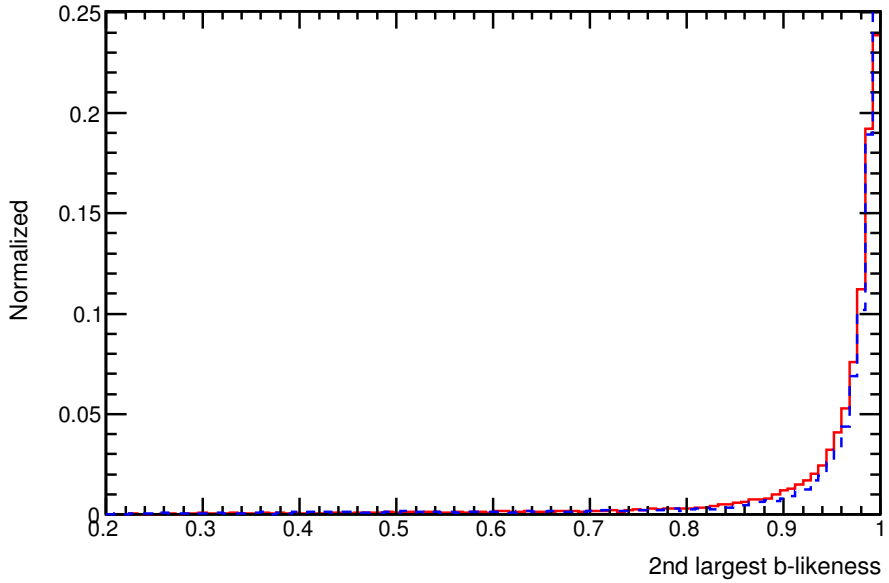
- Overall:
  - Total seen energy
- $e^+e^- \rightarrow ZZ \rightarrow$ four jets:
  - Reconstructed M_Z at different stages in FullSim.
  - Seen Reconstructed M_Z, FullSim and SGV.
  - Jet-Energy resoulution
- *Zhh* at 1 TeV:
  - Vissible E
  - Higgs Mass
  - b-tag

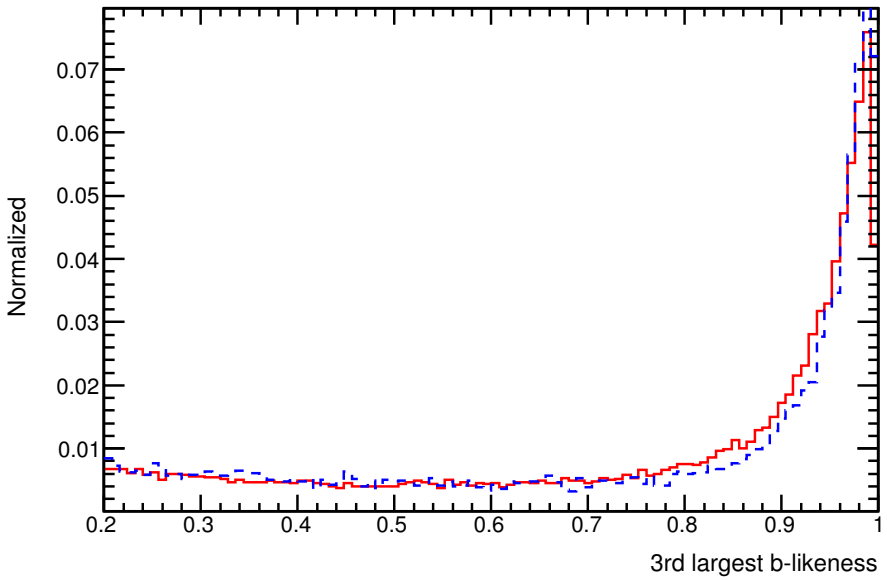


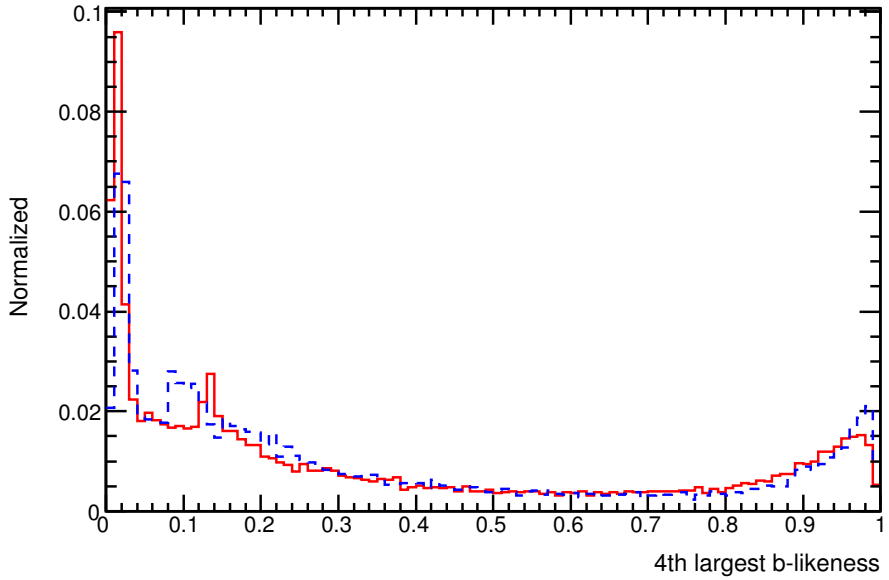
P

F


SGV Simulation
DBD Simulation
Normalized
1st largest b-likeness


Normalized
2nd largest b-likeness

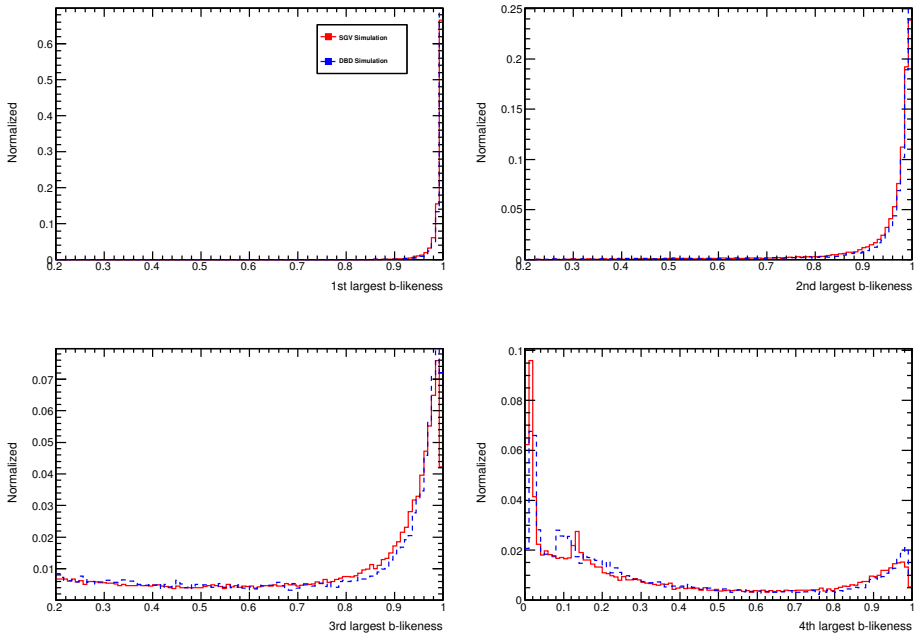

Normalized
3rd largest b-likeness


Normalized
4th largest b-likeness

Proof of principle of the parametrisation

Feed exactly the same physics events through FullSim or SGV.

- Overall:
  - Total seen energy
- $e^+e^- \rightarrow ZZ \rightarrow$ four jets:
  - Reconstructed M_Z at different stages in FullSim.
  - Seen Reconstructed M_Z, FullSim and SGV.
  - Jet-Energy resoulution
- *Zhh* at 1 TeV:
  - Vissible E
  - Higgs Mass
  - b-tag



Technicalities

- Written in Fortran 08, a re-write of the Fortran77-based SGV2 series.
- Managed in SVN.Install script included.
- Features:
  - Callable PYTHIA, Whizard (v. 1.x, only, since v2.x is not yet callable as a subroutine).
  - Input from PYJETS, stdhep, slcio, GuineaPig.
  - Output of generated event to PYJETS, stdhep or slcio.
  - Produce LCIO-DST look-alike of reconstructed events.
  - samples subdirectory with steering and code for eg. scan single particles, create ntuples with "all" information (right now, as hbook ntuples, so need CERNLIB for this, can be converted to ROOT w/ h2root. Direct Root interface comming) .
  - Development on calorimeters (see later)
- Typical generation+simulation+reconstruction time $\mathcal{O}(10)$ ms.

Technicalities

- Written in Fortran 08, a re-write of the Fortran77-based SGV2 series.
- Managed in SVN.Install script included.
- Features:
  - Callable PYTHIA, Whizard (v. 1.x, only, since v2.x is not yet callable as a subroutine).
  - Input from PYJETS, stdhep, slcio, GuineaPig.
  - Output of generated event to PYJETS, stdhep or slcio.
  - Produce LCIO-DST look-alike of reconstructed events.
  - samples subdirectory with steering and code for eg. scan single particles, create ntuples with "all" information (right now, as hbook ntuples, so need CERNLIB for this, can be converted to ROOT w/ h2root. Direct Root interface comming) .
  - Development on calorimeters (see later)
- Typical generation+simulation+reconstruction time $\mathcal{O}(10)$ ms.

Technicalities

- Written in Fortran 08, a re-write of the Fortran77-based SGV2 series.
- Managed in SVN.Install script included.
- Features:
  - Callable PYTHIA, Whizard (v. 1.x, only, since v2.x is not yet callable as a subroutine).
  - Input from PYJETS, stdhep, slcio, GuineaPig.
  - Output of generated event to PYJETS, stdhep or slcio.
  - Produce LCIO-DST look-alike of reconstructed events.
  - samples subdirectory with steering and code for eg. scan single particles, create ntuples with "all" information (right now, as hbook ntuples, so need CERNLIB for this, can be converted to ROOT w/ h2root. Direct Root interface comming) .
  - Development on calorimeters (see later)
- Typical generation+simulation+reconstruction time $\mathcal{O}(10)$ ms.

Technicalities

- Written in Fortran 08, a re-write of the Fortran77-based SGV2 series.
- Managed in SVN.Install script included.
- Features:
  - Callable PYTHIA, Whizard (v. 1.x, only, since v2.x is not yet callable as a subroutine).
  - Input from PYJETS, stdhep, slcio, GuineaPig.
  - Output of generated event to PYJETS, stdhep or slcio.
  - Produce LCIO-DST look-alike of reconstructed events.
  - samples subdirectory with steering and code for eg. scan single particles, create ntuples with "all" information (right now, as hbook ntuples, so need CERNLIB for this, can be converted to ROOT w/ h2root. Direct Root interface comming) .
  - Development on calorimeters (see later)
- Typical generation+simulation+reconstruction time $\mathcal{O}(10)$ ms.

Installing SGV

Do

svn co https://svnsrv.desy.de/public/sgv/trunk/ sgv/

Then

cd sgv ; . ./install

This will take you about 30 seconds ...

- Study README do get the first test job done (another 30 seconds)
- Look README in the samples sub-directory, to enhance the capabilities, eg.:
  - Get the LCIO i/o set up.
  - Get STDHEP installed.
  - Get Pythia installed.
  - Get Whizard (basic or ILC-tuned) installed.
  - Get CERNLIB installed in native 64bit, iff hbook needed.

Installing SGV

Do

svn co https://svnsrv.desy.de/public/sgv/trunk/ sgv/

Then

cd sgv ; . ./install

This will take you about 30 seconds ...

- Study README do get the first test job done (another 30 seconds)
- Look README in the samples sub-directory, to enhance the capabilities, eg.:
  - Get the LCIO i/o set up.
  - Get STDHEP installed.
  - Get Pythia installed.
  - Get Whizard (basic or ILC-tuned) installed.
  - Get CERNLIB installed in native 64bit, iff hbook needed.

Installing SGV

Do

svn co https://svnsrv.desy.de/public/sgv/trunk/ sgv/

Then

cd sgv ; . ./install

This will take you about 30 seconds ...

- Study README do get the first test job done (another 30 seconds)
- Look README in the samples sub-directory, to enhance the capabilities, eg.:
  - Get the LCIO i/o set up.
  - Get STDHEP installed.
  - Get Pythia installed.
  - Get Whizard (basic or ILC-tuned) installed.
  - Get CERNLIB installed in native 64bit, iff hbook needed.

LCIO DST mass-production

SGV has been used to produce ILD LCIO DST:s for the full DBD benchmarks- several times.

- 43 Mevents.
- $\sim$ 1 hour of wall-clock time (first submit to last completed) on the German NAF.
- Used to filter out the *seeable pairs* in 100 000 bunch-crossings from GuineaPig.

LCIO DST mass-production

SGV has been used to produce ILD LCIO DST:s for the full DBD benchmarks- several times.

- 43 Mevents.
- $\sim$ 1 hour of wall-clock time (first submit to last completed) on the German NAF.
- Used to filter out the *seeable pairs* in 100 000 bunch-crossings from GuineaPig.

Recently added features

- Generator input also as LCIO MCParticles and GuineaPig (in addtion to StdHep and native PYJETS)
- Module to output LCIO-DST after reconstruction.
- Filter-mode fully implemented. Possible to skip events at
  - Generator level
  - Simulation level
  - Reconstruction/analysis level

  and conditionally output generated event at any point, as LCIO or StdHep.
- Native Fortran08 interface to LCIO, automatically built from LCIO DoxyGen.
- No CERNLIB dependence by default:
  - FFREAD replace by namelists, for free format input (NB: means that steering-files need to be slightly modified)
  - A few needed routines from CERNLIB now in reimplemeted SGV
  - Usage of ZEBRA and HBOOK conditional, by default not set.
  - Set NOCERNLIB env variable to avoid that scripts checks for a working cernlib!

Recently added features

- Generator input also as LCIO MCParticles and GuineaPig (in addtion to StdHep and native PYJETS)
- Module to output LCIO-DST after reconstruction.
- Filter-mode fully implemented. Possible to skip events at
  - Generator level
  - Simulation level
  - Reconstruction/analysis level

  and conditionally output generated event at any point, as LCIO or StdHep.
- Native Fortran08 interface to LCIO, automatically built from LCIO DoxyGen.
- No CERNLIB dependence by default:
  - FFREAD replace by namelists, for free format input (NB: means that steering-files need to be slightly modified)
  - A few needed routines from CERNLIB now in reimplemeted SGV
  - Usage of ZEBRA and HBOOK conditional, by default not set.
  - Set NOCERNLIB env variable to avoid that scripts checks for a working cernlib!

Recently added features

- Generator input also as LCIO MCParticles and GuineaPig (in addtion to StdHep and native PYJETS)
- Module to output LCIO-DST after reconstruction.
- Filter-mode fully implemented. Possible to skip events at
  - Generator level
  - Simulation level
  - Reconstruction/analysis level

  and conditionally output generated event at any point, as LCIO or StdHep.
- Native Fortran08 interface to LCIO, automatically built from LCIO DoxyGen.
- No CERNLIB dependence by default:
  - FFREAD replace by namelists, for free format input (NB: means that steering-files need to be slightly modified)
  - A few needed routines from CERNLIB now in reimplemeted SGV
  - Usage of ZEBRA and HBOOK conditional, by default not set.
  - Set NOCERNLIB env variable to avoid that scripts checks for a working cernlib!

Recently added features

- Generator input also as LCIO MCParticles and GuineaPig (in addtion to StdHep and native PYJETS)
- Module to output LCIO-DST after reconstruction.
- Filter-mode fully implemented. Possible to skip events at
  - Generator level
  - Simulation level
  - Reconstruction/analysis level

  and conditionally output generated event at any point, as LCIO or StdHep.
- Native Fortran08 interface to LCIO, automatically built from LCIO DoxyGen.
- No CERNLIB dependence by default:
  - FFREAD replace by namelists, for free format input (NB: means that steering-files need to be slightly modified)
  - A few needed routines from CERNLIB now in reimplemeted SGV
  - Usage of ZEBRA and HBOOK conditional, by default not set.
  - Set NOCERNLIB env variable to avoid that scripts checks for a working cernlib!

Recently added features

- Generator input also as LCIO MCParticles and GuineaPig (in addtion to StdHep and native PYJETS)
- Module to output LCIO-DST after reconstruction.
- Filter-mode fully implemented. Possible to skip events at
  - Generator level
  - Simulation level
  - Reconstruction/analysis level

  and conditionally output generated event at any point, as LCIO or StdHep.
- Native Fortran08 interface to LCIO, automatically built from LCIO DoxyGen.
- No CERNLIB dependence by default:
  - FFREAD replace by namelists, for free format input (NB: means that steering-files need to be slightly modified)
  - A few needed routines from CERNLIB now in reimplemeted SGV
  - Usage of ZEBRA and HBOOK conditional, by default not set.
  - Set NOCERNLIB env variable to avoid that scripts checks for a working cernlib!

Planed developments

- Minimal interface to Root:
  - Open and close Root-files
  - Create and fill ntuples and histograms
  - Output objects
- Studying light-weight alternative to Root
  - netCDF/HDF5 - free,available on any Linux, API for Fortran, C++, C, Java, ..
  - Interfaces to R, MathLAB, Mathematica, Python, Octave (and Root)
  - Unclear how well this works on TB size data?
- Zero B-field: Also usable for fixed taget

Planed developments

- Minimal interface to Root:
  - Open and close Root-files
  - Create and fill ntuples and histograms
  - Output objects
- Studying light-weight alternative to Root
  - netCDF/HDF5 - free,available on any Linux, API for Fortran, C++, C, Java, ..
  - Interfaces to R, MathLAB, Mathematica, Python, Octave (and Root)
  - Unclear how well this works on TB size data?
- Zero B-field: Also usable for fixed taget

Planed developments

- Minimal interface to Root:
  - Open and close Root-files
  - Create and fill ntuples and histograms
  - Output objects
- Studying light-weight alternative to Root
  - netCDF/HDF5 - free,available on any Linux, API for Fortran, C++, C, Java, ..
  - Interfaces to R, MathLAB, Mathematica, Python, Octave (and Root)
  - Unclear how well this works on TB size data?
- Zero B-field: Also usable for fixed taget

Summary

- The SGV FastSim program for ILC physics simulation was presented, and (I hope) was shown to be up to the job, both in physics and computing performance.
- The method to emulate the performance of FullReco particle-flow (PandoraPFO) was explained.
- Comparisons to FullSim (DDsim/Mokka+Marlin) was shown to be quite good.
- SGV mass production works
  - Is done in $\mathcal{O}(1)$ hour.

Summary

- The SGV FastSim program for ILC physics simulation was presented, and (I hope) was shown to be up to the job, both in physics and computing performance.
- The method to emulate the performance of FullReco particle-flow (PandoraPFO) was explained.
- Comparisons to FullSim (DDsim/Mokka+Marlin) was shown to be quite good.
- SGV mass production works
  - Is done in $\mathcal{O}(1)$ hour.

Summary

- The SGV FastSim program for ILC physics simulation was presented, and (I hope) was shown to be up to the job, both in physics and computing performance.
- The method to emulate the performance of FullReco particle-flow (PandoraPFO) was explained.
- Comparisons to FullSim (DDsim/Mokka+Marlin) was shown to be quite good.
- SGV mass production works
  - Is done in $\mathcal{O}(1)$ hour.

Summary

- The SGV FastSim program for ILC physics simulation was presented, and (I hope) was shown to be up to the job, both in physics and computing performance.
- The method to emulate the performance of FullReco particle-flow (PandoraPFO) was explained.
- Comparisons to FullSim (DDsim/Mokka+Marlin) was shown to be quite good.
- SGV mass production works
  - Is done in $\mathcal{O}(1)$ hour.

Summary

- The SGV FastSim program for ILC physics simulation was presented, and (I hope) was shown to be up to the job, both in physics and computing performance.
- The method to emulate the performance of FullReco particle-flow (PandoraPFO) was explained.
- Comparisons to FullSim (DDsim/Mokka+Marlin) was shown to be quite good.
- SGV mass production works
  - Is done in $\mathcal{O}(1)$ hour.

Summary

- The SGV FastSim program for ILC physics simulation was presented, and (I hope) was shown to be up to the job, both in quite good.
- SGV mass production works
  - Is done in $\mathcal{O}(1)$ hour.

Installing SGV

svn co https://svnsrv.desy.de/public/sgv/trunk/ sgv/

Then

cd sgv ; . ./install

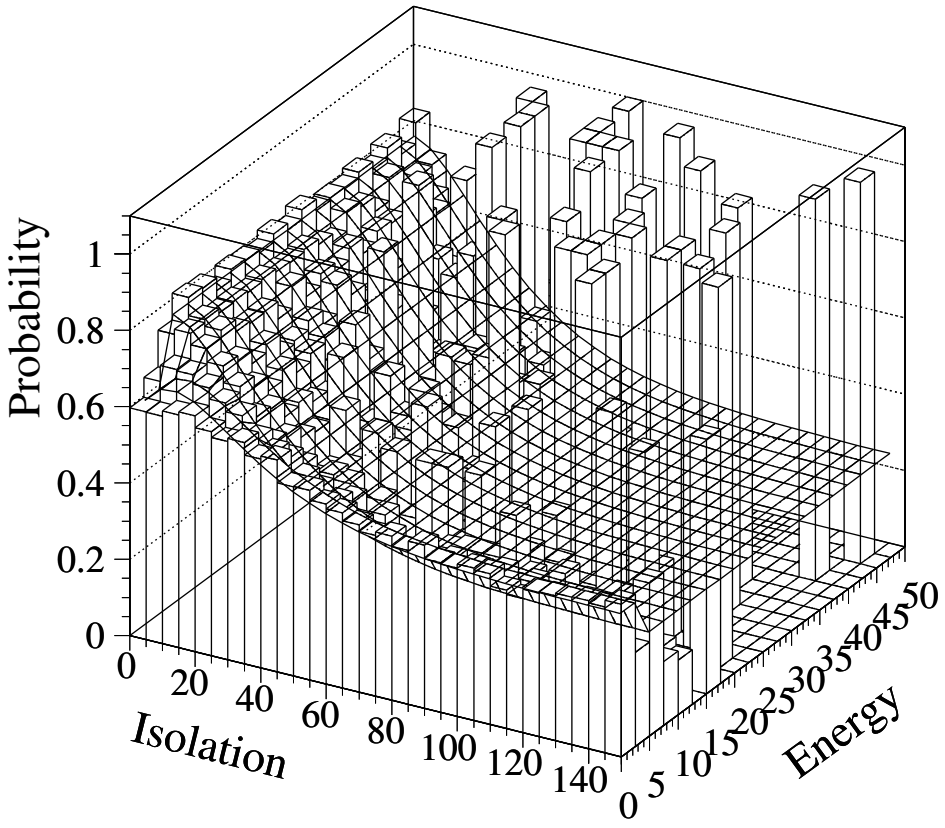
Thank You !

Backup

BACKUP SLIDES

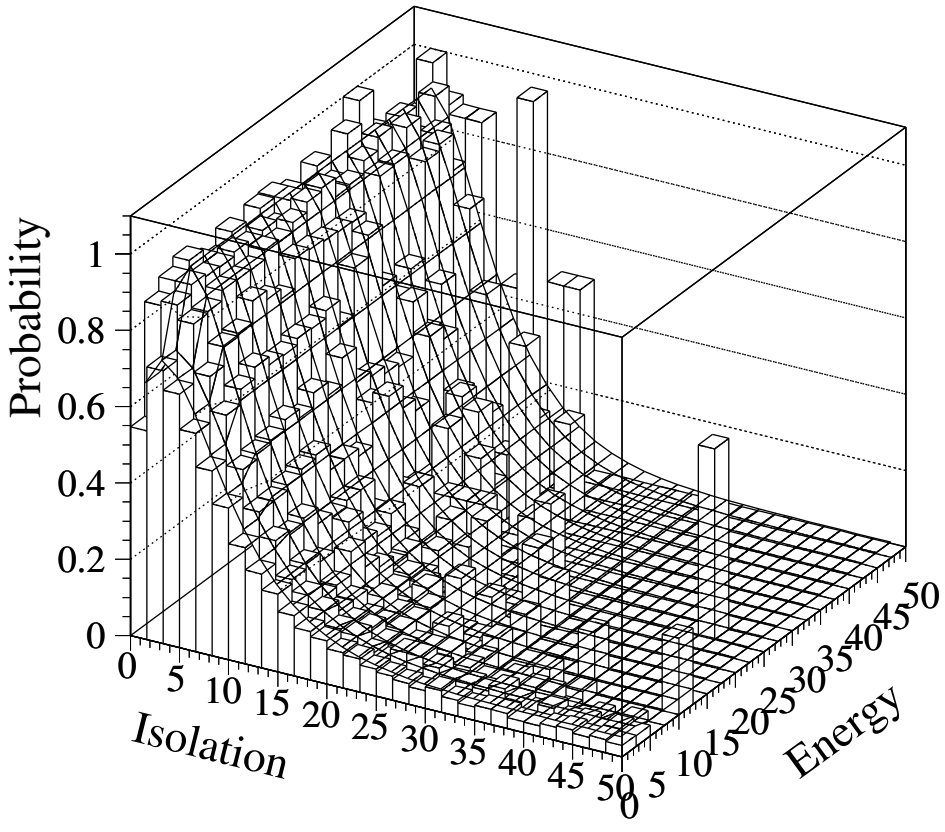
Observed distributions

- Probability to split (charged had or γ)
- Fraction the energy vs distance
- ... and vs E
- Fit of the Distribution of the fraction
- Average fraction vs. E and distance.



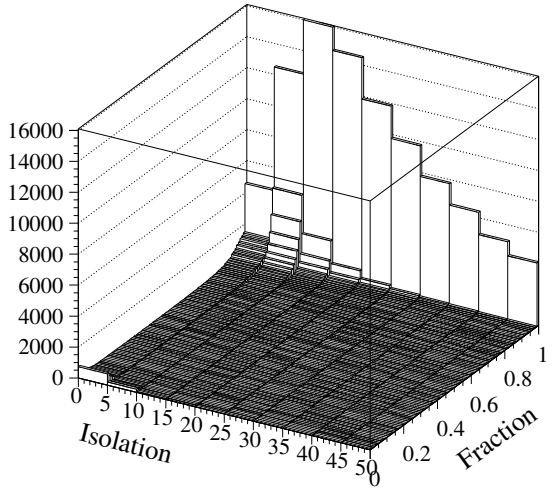
Observed distributions

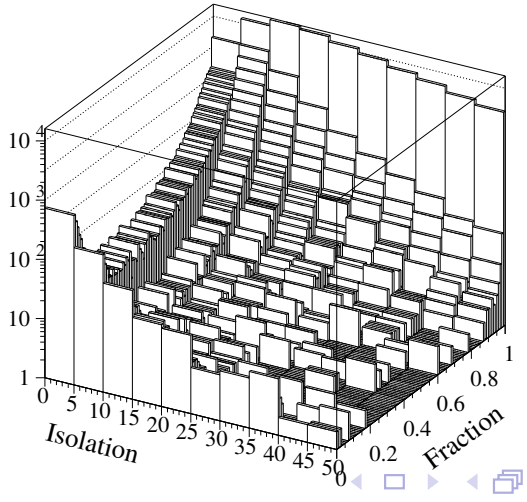
- Probability to split (charged had or γ)
- Fraction the energy vs distance
- ... and vs E
- Fit of the Distribution of the fraction
- Average fraction vs. E and distance.



Observed distributions

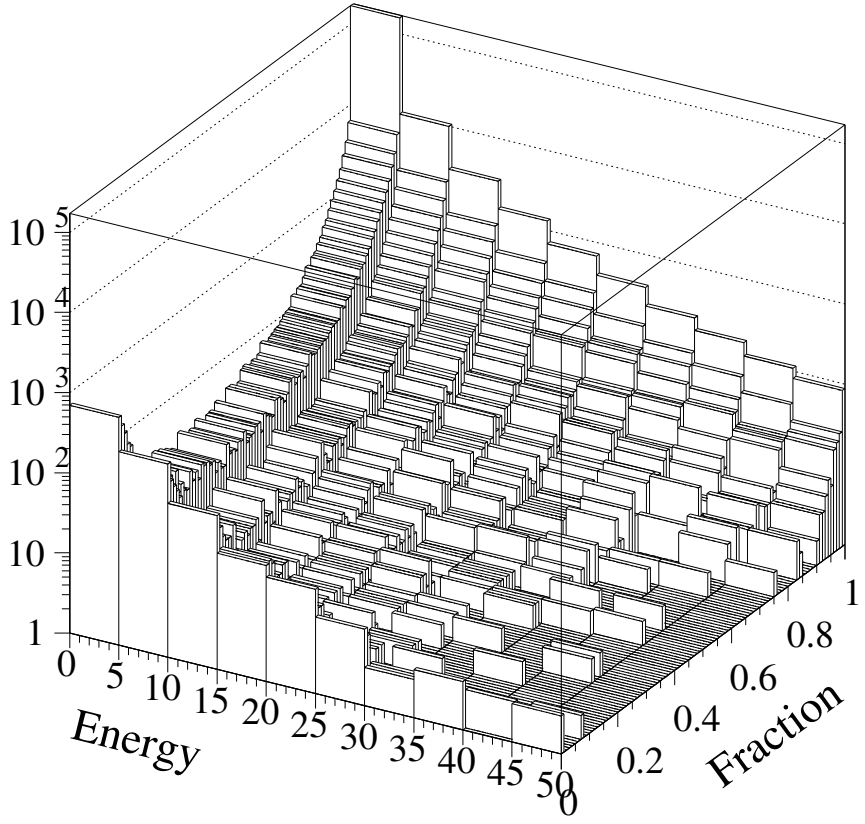
- Probability to split (charged had or γ)
- Fraction the energy vs distance
- ... and vs E
- Fit of the Distribution of the fraction
- Average fraction vs. E and distance.





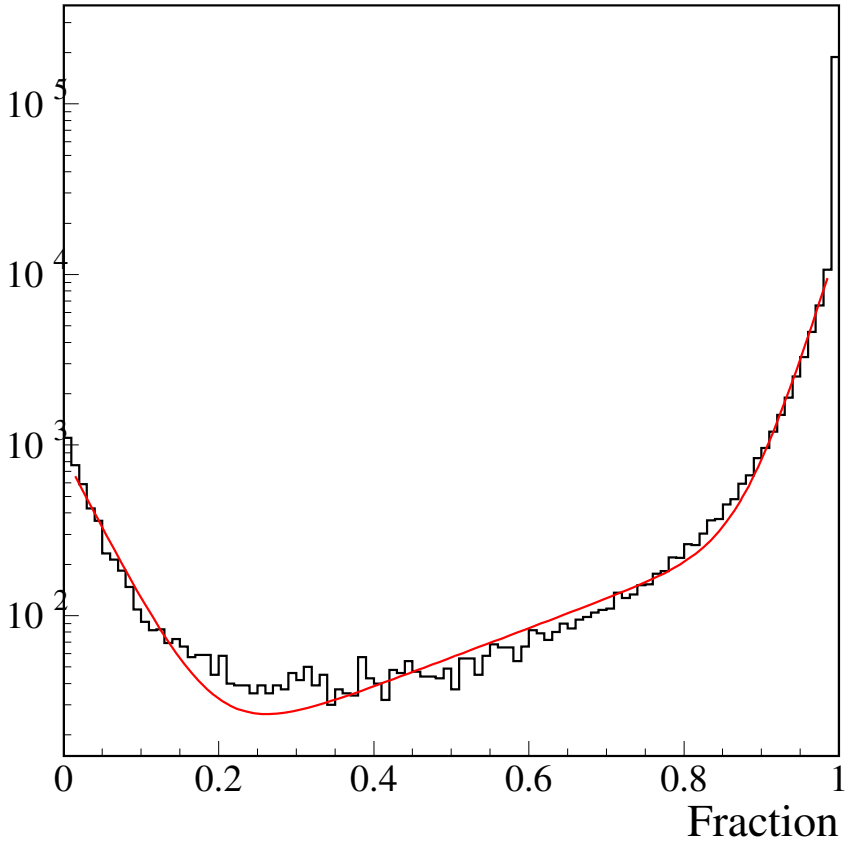
Observed distributions

- Probability to split (charged had or γ)
- Fraction the energy vs distance
- ... and vs E
- Fit of the Distribution of the fraction
- Average fraction vs. E and distance.



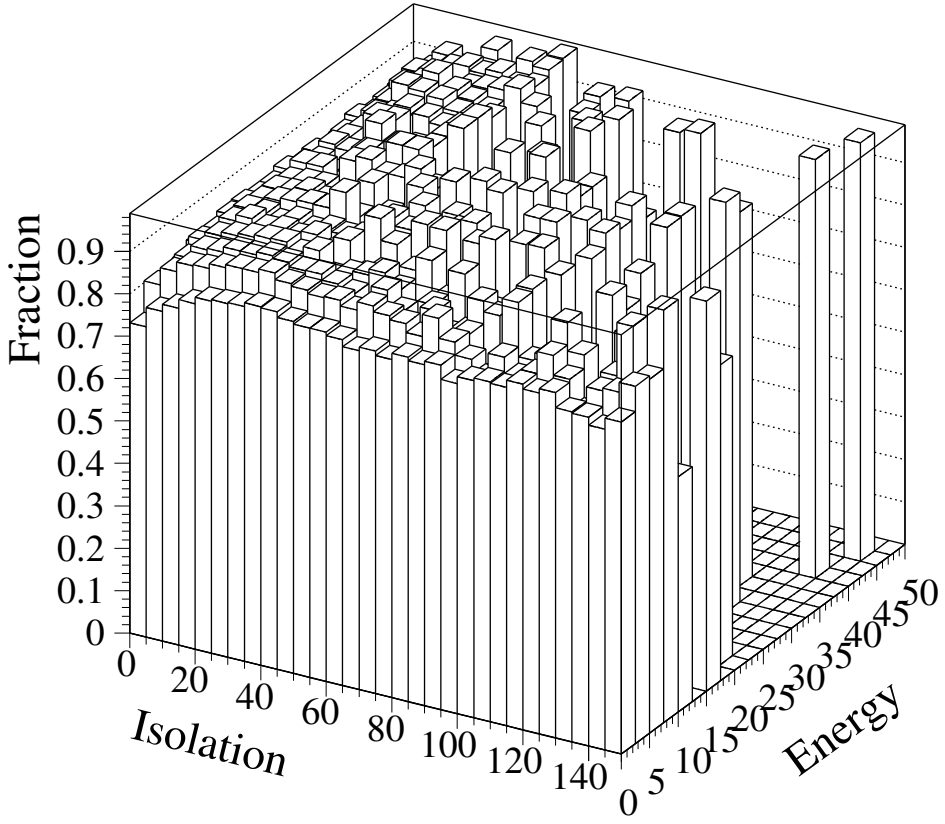
Observed distributions

- Probability to split (charged had or γ)
- Fraction the energy vs distance
- ... and vs E
- Fit of the Distribution of the fraction
- Average fraction vs. E and distance.



Observed distributions

- Probability to split (charged had or γ)
- Fraction the energy vs distance
- ... and vs E
- Fit of the Distribution of the fraction
- Average fraction vs. E and distance.



$\gamma\gamma$ background

Total cross-section for $e^+e^- \to \gamma\gamma e^+e^- \to q\bar{q}e^+e^-$: 35 nb (PYTHIA)

- $\int \mathcal{L}dt = 500$ fb^{-1} $\to$ 18 $\star 10^9$ events are expected.
- 10 ms to generate one event.
- 10 ms to fastsim (SGV) one event.

10^8 s of CPU time is needed, ie more than 3 years. But:This goes to 3000 years with full simulation.

$\gamma\gamma$ background

Total cross-section for $e^+e^- \to \gamma\gamma e^+e^- \to q\bar{q}e^+e^-$: 35 nb (PYTHIA)

- $\int \mathcal{L}dt = 500$ fb^{-1} $\to$ 18 $\star 10^9$ events are expected.
- 10 ms to generate one event.
- 10 ms to fastsim (SGV) one event.

10^8 s of CPU time is needed, ie more than 3 years. But:This goes to 3000 years with full simulation.

$\gamma\gamma$ background

Total cross-section for $e^+e^- \to \gamma\gamma e^+e^- \to q\bar{q}e^+e^-$: 35 nb (PYTHIA)

- $\int \mathcal{L}dt = 500$ fb$^{-1} \to 18 \star 10^9$ events are expected.
- 10 ms to generate one event.
- 10 ms to fastsim (SGV) one event.

10^8 s of CPU time is needed, ie more than 3 years. But:This goes to 3000 years with full simulation.

SUSY parameter scans

Simple example:

- MSUGRA: 4 parameters + sign of μ
- Scan each in eg. 20 steps
- Eg. 5000 events per point (modest requirement: in sps1a' almost 1 million SUSY events are expected for 500 fb^{-1} !)
- = $20^4 \times 2 \times 5000 = 1.6 \times 10^9$ events to generate...

Slower to generate and simulate than $\gamma\gamma$ events

Also here: CPU millenniums with full simulation

SUSY parameter scans

Simple example:

- MSUGRA: 4 parameters + sign of μ
- Scan each in eg. 20 steps
- Eg. 5000 events per point (modest requirement: in sps1a' almost 1 million SUSY events are expected for 500 fb^{-1} !)
- = $20^4 \times 2 \times 5000 = 1.6 \times 10^9$ events to generate...

Slower to generate and simulate than $\gamma\gamma$ events

Also here: CPU millenniums with full simulation

Use-cases at the ILC

- Used for fastsim physics studies, eg. arXiv:hep-ph/0510088, arXiv:hep-ph/0508247, arXiv:hep-ph/0406010, arXiv:hep-ph/9911345 and arXiv:hep-ph/9911344.
- Used for flavour-tagging training.
- Used for overall detector optimisation, see Eg. Vienna ECFA WS (2007), See Ilcagenda > Conference and Workshops > 2005 > ECFA Vienna Tracking
- GLD/LDC merging and LOI, see eg. Ilcagenda > Detector Design & Physics Studies > Detector Design Concepts > ILD > ILD Workshop > ILD Meeting, Cambridge > Agenda >Sub-detector Optimisation I

The latter two: Use the Covariance machine to get analytical expressions for performance (ie. *not* simulation)

White paper

- Written in Fortran 95.
- CERNLIB dependence. Much reduced wrt. old F77 version, mostly by using Fortran 95's built-in matrix algebra.
- Managed in SVN.Install script included.
- Features:
  - Callable PYTHIA, Whizard.
  - Input from PYJETS or stdhep.
  - Output of generated event to PYJETS or stdhep.
  - samples subdirectory with steering and code for eg. scan single particles, create hbook ntuple with "all" information (can be converted to ROOT w/ h2root). And: output LCIO DST.
  - Development on calorimeters (see later)
- Tested to work on both 32 and 64 bit out-of-the-box.
- Timing verified to be faster (by 15%) than the f77 version.

White paper

- Written in Fortran 95.
- CERNLIB dependence. Much reduced wrt. old F77 version, mostly by using Fortran 95's built-in matrix algebra.
- Managed in SVN.Install script included.
- Features:
  - Callable PYTHIA, Whizard.
  - Input from PYJETS or stdhep.
  - Output of generated event to PYJETS or stdhep.
  - samples subdirectory with steering and code for eg. scan single particles, create hbook ntuple with "all" information (can be converted to ROOT w/ h2root). And: output LCIO DST.
  - Development on calorimeters (see later)
- Tested to work on both 32 and 64 bit out-of-the-box.
- Timing verified to be faster (by 15%) than the f77 version.

White paper

- Written in Fortran 95.
- CERNLIB dependence. Much reduced wrt. old F77 version, mostly by using Fortran 95's built-in matrix algebra.
- Managed in SVN.Install script included.
- Features:
  - Callable PYTHIA, Whizard.
  - Input from PYJETS or stdhep.
  - Output of generated event to PYJETS or stdhep.
  - samples subdirectory with steering and code for eg. scan single particles, create hbook ntuple with "all" information (can be converted to ROOT w/ h2root). And: output LCIO DST.
  - Development on calorimeters (see later)
- Tested to work on both 32 and 64 bit out-of-the-box.
- Timing verified to be faster (by 15%) than the f77 version.

White paper

- Written in Fortran 95.
- CERNLIB dependence. Much reduced wrt. old F77 version, mostly by using Fortran 95's built-in matrix algebra.
- Managed in SVN.Install script included.
- Features:
  - Callable PYTHIA, Whizard.
  - Input from PYJETS or stdhep.
  - Output of generated event to PYJETS or stdhep.
  - samples subdirectory with steering and code for eg. scan single particles, create hbook ntuple with "all" information (can be converted to ROOT w/ h2root). And: output LCIO DST.
  - Development on calorimeters (see later)
- Tested to work on both 32 and 64 bit out-of-the-box.
- Timing verified to be faster (by 15%) than the f77 version.

White paper

- Written in Fortran 95.
- CERNLIB dependence. Much reduced wrt. old F77 version, mostly by using Fortran 95's built-in matrix algebra.
- Managed in SVN.Install script included.
- Features:
  - Callable PYTHIA, Whizard.
  - Input from PYJETS or stdhep.
  - Output of generated event to PYJETS or stdhep.
  - samples subdirectory with steering and code for eg. scan single particles, create hbook ntuple with "all" information (can be converted to ROOT w/ h2root). And: output LCIO DST.
  - Development on calorimeters (see later)
- Tested to work on both 32 and 64 bit out-of-the-box.
- Timing verified to be faster (by 15%) than the f77 version.

Installing SGV

```
svn export https://svnsrv.desy.de/public/sgv/tags/SGV-3.0rc1/
SGV-3.0rc1/
```

Then

```
bash install
```

This will take you about a minute ...
Study README, and README in the samples sub-directory, to eg.:

- Get STDHEP installed.
- Get CERNLIB installed in native 64bit.
- Get Whizard (basic or ILC-tuned) installed, with complications solved.
- Get the LCIO-DST writer set up

Installing SGV

```
svn export https://svnsrv.desy.de/public/sgv/tags/SGV-3.0rc1/
SGV-3.0rc1/
```

Then

```
bash install
```

This will take you about a minute ...
Study README, and README in the samples sub-directory, to eg.:

- Get STDHEP installed.
- Get CERNLIB installed in native 64bit.
- Get Whizard (basic or ILC-tuned) installed, with complications solved.
- Get the LCIO-DST writer set up

Installing SGV

```
svn export https://svnsrv.desy.de/public/sgv/tags/SGV-3.0rc1/
SGV-3.0rc1/
```

Then

```
bash install
```

This will take you about a minute ...
Study README, and README in the samples sub-directory, to eg.:

- Get STDHEP installed.
- Get CERNLIB installed in native 64bit.
- Get Whizard (basic or ILC-tuned) installed, with complications solved.
- Get the LCIO-DST writer set up

Installing SGV

```
svn export https://svnsrv.desy.de/public/sgv/tags/SGV-3.0rc1/
SGV-3.0rc1/
```

Then

```
bash install
```

This will take you about a minute ...
Study README, and README in the samples sub-directory, to eg.:

- Get STDHEP installed.
- Get CERNLIB installed in native 64bit.
- Get Whizard (basic or ILC-tuned) installed, with complications solved.
- Get the LCIO-DST writer set up

Calorimeter simulation: SGV strategy

- Concentrate on what really matters:
  - True charged particles splitting off (a part of) their shower: double-counting.
  - True neutral particles merging (a part of) their shower with charged particles: enetgy loss.
- Don't care about neutral-neutral or charged-charged merging.
- Nor about multiple splitting/merging.
- Then: identify the most relevant variables available in fast simulation:
  - Cluster energy.
  - Distance to nearest particle of "the other type"
  - EM or hadron.
  - Barrel or end-cap.

Calorimeter simulation: SGV strategy

- Concentrate on what really matters:
  - True charged particles splitting off (a part of) their shower: double-counting.
  - True neutral particles merging (a part of) their shower with charged particles: enetgy loss.
- Don't care about neutral-neutral or charged-charged merging.
- Nor about multiple splitting/merging.
- Then: identify the most relevant variables available in fast simulation:
  - Cluster energy.
  - Distance to nearest particle of "the other type"
  - EM or hadron.
  - Barrel or end-cap.

Calorimeter simulation: SGV strategy

- Concentrate on what really matters:
  - True charged particles splitting off (a part of) their shower: double-counting.
  - True neutral particles merging (a part of) their shower with charged particles: enetgy loss.
- Don't care about neutral-neutral or charged-charged merging.
- Nor about multiple splitting/merging.
- Then: identify the most relevant variables available in fast simulation:
  - Cluster energy.
  - Distance to nearest particle of "the other type"
  - EM or hadron.
  - Barrel or end-cap.

Collections

- Added sensible values to all collections that will (probably) be there on the DST from the fullSim production.
  - BuildUpVertex
  - BuildUpVertex_RP
  - MarlinTrkTracks
  - PandoraClusters
  - PandoraPFOs
  - PrimaryVertex
  - RecoMCTruthLink
  - MCParticlesSkimmed
  - V0Vertices
  - V0RecoParticles
  - BCALParticles
  - BCALClusters
  - BCALMCTruthLink
  - PrimaryVertex_RP
- Also added more relation links:
  - MCTruthRecoLink
  - ClusterMCTruthLink
  - MCTruthClusterLink
  - MCTruthTrackLink
  - TrackMCTruthLink
  - MCTruthBcalLink

Comments

Secondary vertices (as before):

- Use true information to find all secondary vertices.
- For all vertices with ≥ 2 seen charged tracks: do vertex fit.
- Concequence:
  - Vertex *finding* is too good.
  - Vertex *quality* should be comparable to FullSim.

In addition: Decide from parent pdg-code if it goes into BuildUpVertex or V0Vertices !

MCParticle :

- There might be some issues with history codes in the earlier part of the event (initial beam-particles, 94-objects, ...)

Comments

Clusters:

- Are done with the Pandora confusion parametrisation on.
- Expect $\sim$ correct dispersion of jet energy, but a few % to high central value.
- See my talk three weeks ago.
- Warning: Clusters are always only in one detector , so don't use E_{had}/E_{EM} for e/π: It will be $\equiv$ 100 % efficient !

Navigators

- All the navigators that the TruthLinker processor makes when all flags are switched on are created:
  - Both Seen to True and True to Seen (weights are different !)
  - Seen is both PFOs, tracks and clusters.
  - The standard RecoMCTruthLink collection is as it would be from FullSim ie. weights between 0 and 1.

Outlook

- Include a filter-mode:
  - Generate event inside SGV.
  - Run SGV detector simulation and analysis.
  - Decide what to do: Fill some histos, fill ntuple, output LCIO, or better do full sim
  - In the last case: output STDHEP of event
- Update documentation and in-line comments, to reflect new structure.
- Consolidate use of Fortran 95/203/2008 features. Possibly - when gcc/gfortran 4.4 (ie. Fortran 2003) is common-place - Object Orientation, if there is no performance penalty.
  - Use of user-defined types.
  - Use of PURE and ELEMENTAL routines.
  - Optimal choice between pointer, allocatable and automatic and/or assumed-size, assumed-shape, and explicit arrays.
- I/O over FIFO:s to avoid storage and I/O rate limitations.
- The Grid.
- Investigate running on GPU:s.

Outlook

- Include a filter-mode:
  - Generate event inside SGV.
  - Run SGV detector simulation and analysis.
  - Decide what to do: Fill some histos, fill ntuple, output LCIO, or better do full sim
  - In the last case: output STDHEP of event
- Update documentation and in-line comments, to reflect new structure.
- Consolidate use of Fortran 95/203/2008 features. Possibly - when gcc/gfortran 4.4 (ie. Fortran 2003) is common-place - Object Orientation, if there is no performance penalty.
  - Use of user-defined types.
  - Use of PURE and ELEMENTAL routines.
  - Optimal choice between pointer, allocatable and automatic and/or assumed-size, assumed-shape, and explicit arrays.
- I/O over FIFO:s to avoid storage and I/O rate limitations.
- The Grid.
- Investigate running on GPU:s.

Outlook

- Include a filter-mode:
  - Generate event inside SGV.
  - Run SGV detector simulation and analysis.
  - Decide what to do: Fill some histos, fill ntuple, output LCIO, or better do full sim
  - In the last case: output STDHEP of event
- Update documentation and in-line comments, to reflect new structure.
- Consolidate use of Fortran 95/203/2008 features. Possibly - when gcc/gfortran 4.4 (ie. Fortran 2003) is common-place - Object Orientation, if there is no performance penalty.
  - Use of user-defined types.
  - Use of PURE and ELEMENTAL routines.
  - Optimal choice between pointer, allocatable and automatic and/or assumed-size, assumed-shape, and explicit arrays.
- I/O over FIFO:s to avoid storage and I/O rate limitations.
- The Grid.
- Investigate running on GPU:s.

Outlook

- Include a filter-mode:
  - Generate event inside SGV.
  - Run SGV detector simulation and analysis.
  - Decide what to do: Fill some histos, fill ntuple, output LCIO, or better do full sim
  - In the last case: output STDHEP of event
- Update documentation and in-line comments, to reflect new structure.
- Consolidate use of Fortran 95/203/2008 features. Possibly - when gcc/gfortran 4.4 (ie. Fortran 2003) is common-place - Object Orientation, if there is no performance penalty.
  - Use of user-defined types.
  - Use of PURE and ELEMENTAL routines.
  - Optimal choice between pointer, allocatable and automatic and/or assumed-size, assumed-shape, and explicit arrays.
- I/O over FIFO:s to avoid storage and I/O rate limitations.
- The Grid.
- Investigate running on GPU:s.

Outlook

- Include a filter-mode:
  - Generate event inside SGV.
  - Run SGV detector simulation and analysis.
  - Decide what to do: Fill some histos, fill ntuple, output LCIO, or better do full sim
  - In the last case: output STDHEP of event
- Update documentation and in-line comments, to reflect new structure.
- Consolidate use of Fortran 95/203/2008 features. Possibly - when gcc/gfortran 4.4 (ie. Fortran 2003) is common-place - Object Orientation, if there is no performance penalty.
  - Use of user-defined types.
  - Use of PURE and ELEMENTAL routines.
  - Optimal choice between pointer, allocatable and automatic and/or assumed-size, assumed-shape, and explicit arrays.
- I/O over FIFO:s to avoid storage and I/O rate limitations.
- The Grid.
- Investigate running on GPU:s.

Outlook

- Include a filter-mode:
  - Generate event inside SGV.
  - Run SGV detector simulation and analysis.
  - Decide what to do: Fill some histos, fill ntuple, output LCIO, or better do full sim
  - In the last case: output STDHEP of event
- Update documentation and in-line comments, to reflect new structure.
- Consolidate use of Fortran 95/203/2008 features. Possibly - when gcc/gfortran 4.4 (ie. Fortran 2003) is common-place - Object Orientation, if there is no performance penalty.
  - Use of user-defined types.
  - Use of PURE and ELEMENTAL routines,
  - Optimal choice between pointer, allocatable and automatic and/or assumed-size, assumed-shape, and explicit arrays.
- I/O over FIFO:s to avoid storage and I/O rate limitations.
- The Grid.
- Investigate running on GPU:s.

Outlook

- Include a filter-mode:
  - Generate event inside SGV.
  - Run SGV detector simulation and analysis.
  - Decide what to do: Fill some histos, fill ntuple, output LCIO, or better do full sim
  - In the last case: output STDHEP of event
- Update documentation and in-line comments, to reflect new structure.
- Consolidate use of Fortran 95/203/2008 features. Possibly - when gcc/gfortran 4.4 (ie. Fortran 2003) is common-place - Object Orientation, if there is no performance penalty.
  - Use of user-defined types.
  - Use of PURE and ELEMENTAL routines,
  - Optimal choice between pointer, allocatable and automatic and/or assumed-size, assumed-shape, and explicit arrays.
- I/O over FIFO:s to avoid storage and I/O rate limitations.
- The Grid.
- Investigate running on GPU:s.