

# The SGV fast Monte Carlo: a tool for rapid LC studies?

Mikael Berggren<sup>1</sup>

<sup>1</sup>DESY, Hamburg

ILD meeting, May 12, 2020



# Outline

- 1 The need for fast simulation
- 2 Fast simulation for ILC
- 3 SGV
  - Tracker simulation
    - Comparison with fullsim
  - Calorimeter simulation
- 4 Technicalities
- 5 SGV for physics
- 6 Outlook and Summary

# 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 ?

# NEED FOR SPEED™



# Full SM simulation

Total cross-section @500 GeV for  $e^+e^- \rightarrow 2f$  and  $4f$ : **350 pb** (Whizard), *not* including bhabha, nor  $\gamma\gamma$  events !

- $\int \mathcal{L} dt = 10 \text{ ab}^{-1} \rightarrow 3.5 \times 10^9$  events are expected.
- 10 ms to generate one event.
- 30 ms to fastsim (SGV) one event.

$4 \times 10^7$  s of CPU time is needed, ie more than 1 year. But: This goes to **1000 years** with full simulation.

Then add bhabha's and  $\gamma\gamma$  to that  $\sim$  an order of magnitude more  $\Rightarrow$  without FastSim, many of our results will be systematics dominated by lack of MC statistics!

# Full SM simulation

Total cross-section @500 GeV for  $e^+e^- \rightarrow 2f$  and  $4f$ : **350 pb** (Whizard), *not* including bhabha, nor  $\gamma\gamma$  events !

- $\int \mathcal{L} dt = 10 \text{ ab}^{-1} \rightarrow 3.5 \star 10^9$  events are expected.
- 10 ms to generate one event.
- 30 ms to fastsim (SGV) one event.

$4 \times 10^7$  s of CPU time is needed, ie more than 1 year. But: This goes to **1000 years** with full simulation.

Then add bhabha's and  $\gamma\gamma$  to that  $\sim$  an order of magnitude more  $\Rightarrow$  without FastSim, many of our results will be systematics dominated by lack of MC statistics!

# Full SM simulation

Total cross-section @500 GeV for  $e^+e^- \rightarrow 2f$  and  $4f$ : **350 pb** (Whizard), *not* including bhabha, nor  $\gamma\gamma$  events !

- $\int \mathcal{L} dt = 10 \text{ ab}^{-1} \rightarrow 3.5 \star 10^9$  events are expected.
- 10 ms to generate one event.
- 30 ms to fastsim (SGV) one event.

$4 \times 10^7$  s of CPU time is needed, ie more than **1 year**. **But:** This goes to **1000 years** with full simulation.

Then add bhabha's and  $\gamma\gamma$  to that  $\sim$  an order of magnitude more  $\Rightarrow$  without FastSim, many of our results will be systematics dominated by **lack of MC statistics!**

# SUSY parameter scans

Simple example:

- MSUGRA: 4 parameters + sign of  $\mu$
- Scan each in eg. 20 steps
- Eg. 5000 events per point (modest requirement: in sps1a' almost 1 million SUSY events are expected for  $500 \text{ fb}^{-1}$  !)
- =  $20^4 \times 2 \times 5000 = 1.6 \times 10^9$  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  $\mu$
- Scan each in eg. 20 steps
- Eg. 5000 events per point (modest requirement: in sps1a' almost 1 million SUSY events are expected for  $500 \text{ fb}^{-1}$  !)
- =  $20^4 \times 2 \times 5000 = 1.6 \times 10^9$  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, **needing** input from FullSim: Delphes
  - Ignores correlations, eg. between p measurement and ip:s.
  - Hard handle confusion in high granularity calorimeters.
  - No dE/dx, secondary vertices, effect of hit level inefficiencies ....
  - **But very fast**, and very condensed output.
  - By theoreticians, for theoreticians.
- Covariance matrix machines, **not needing** input from FullSim:SGV
  - Full covariance matrix available for each track-helix.
  - Individual shower shape and position generated  $\Rightarrow$  can approximate confusion.
  - Hit patterns known  $\Rightarrow$  dE/dX and hit-level efficiencies doable.
  - Covariance matrices available  $\Rightarrow$  vertex fitting.
  - Anything up to DST-level detail can be output.
  - Slower (factor 10) than Parametric, but still about **as fast as the event generator**.
  - By experimentalists, for both experimentalists and theoreticians.

# Fast simulation types, and the choice for ILC

Different types, with increasing level of sophistication:

- 4-vector smearing.
- Parametric, **needing** input from FullSim: Delphes
  - Ignores correlations, eg. between p measurement and ip:s.
  - Hard handle confusion in high granularity calorimeters.
  - No dE/dx, secondary vertices, effect of hit level inefficiencies ....
  - **But very fast**, and very condensed output.
  - By theoreticians, for theoreticians.
- Covariance matrix machines, **not needing** input from FullSim:SGV
  - Full covariance matrix available for each track-helix.
  - Individual shower shape and position generated  $\Rightarrow$  can approximate confusion.
  - Hit patterns known  $\Rightarrow$  dE/dX and hit-level efficiencies doable.
  - Covariance matrices available  $\Rightarrow$  vertex fitting.
  - Anything up to DST-level detail can be output.
  - Slower (factor 10) than Parametric, but still about **as fast as the event generator**.
  - By experimentalists, for both experimentalists and theoreticians.

# Fast simulation types, and the choice for ILC

Different types, with increasing level of sophistication:

- 4-vector smearing.
- Parametric, **needing** input from FullSim: Delphes
  - Ignores correlations, eg. between p measurement and ip:s.
  - Hard handle confusion in high granularity calorimeters.
  - No dE/dx, secondary vertices, effect of hit level inefficiencies ....
  - **But very fast**, and very condensed output.
  - By theoreticians, for theoreticians.
- Covariance matrix machines, **not needing** input from FullSim: **SGV**
  - Full covariance matrix available for each track-helix.
  - Individual shower shape and position generated  $\Rightarrow$  can approximate confusion.
  - Hit patterns known  $\Rightarrow$  dE/dX and hit-level efficiencies doable.
  - Covariance matrices available  $\Rightarrow$  vertex fitting.
  - Anything up to DST-level detail can be output.
  - Slower (factor 10) than Parametric, but still about **as fast as the event generator**.
  - By experimentalists, for both experimentalists and theoreticians.

# Fast simulation types, and the choice for ILC

Different types, with increasing level of sophistication:

- 4-vector smearing.
- Parametric, **needing** input from FullSim: Delphes
  - Ignores correlations, eg. between p measurement and ip:s.
  - Hard handle confusion in high granularity calorimeters.

For ILC:

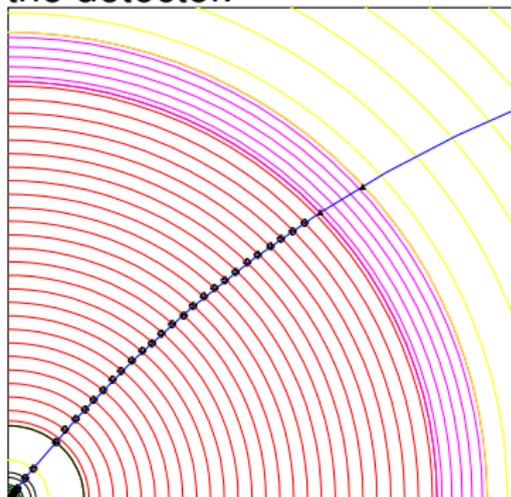
Only Covariance matrix machines have sufficient detail. Here, I'll cover "la Simulation à Grande Vitesse", **SGV**.

- Full covariance matrix available for each track-helix.
- Individual shower shape and position generated  $\Rightarrow$  can approximate confusion.
- Hit patterns known  $\Rightarrow$  dE/dX and hit-level efficiencies doable.
- Covariance matrices available  $\Rightarrow$  vertex fitting.
- Anything up to DST-level detail can be output.
- Slower (factor 10) than Parametric, but still about **as fast as the event generator**.

# SGV: How tracking works

**SGV** is a machine to calculate covariance matrices

**Tracking:** Follow track-helix through the detector.

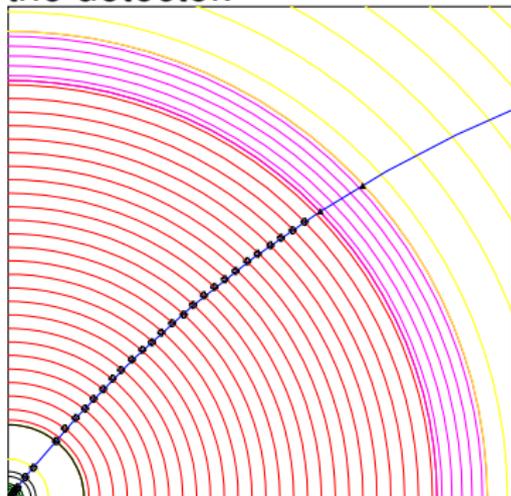


- Calculate cov. mat. at perigee, including material, measurement errors and extrapolation. NB: this is exactly what Your Kalman filter does!
- Smear perigee parameters (Choleski decomposition: takes all correlations into account)
- Helix *parameters* exactly calculated, *errors* with one approximation: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

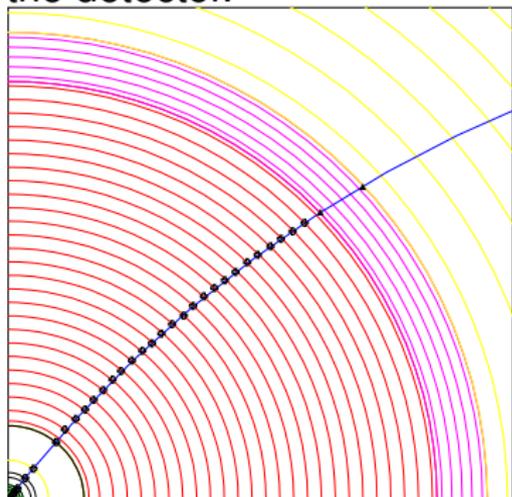


- Calculate cov. mat. at perigee, including material, measurement errors and extrapolation. **NB: this is exactly what Your Kalman filter does!**
- Smear perigee parameters (Choleski decomposition: takes all correlations into account)
- Helix *parameters* exactly calculated, *errors* with one approximation: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

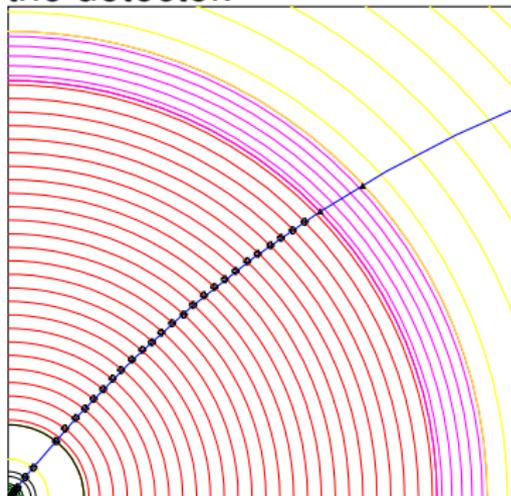


- Calculate cov. mat. at perigee, including material, measurement errors and extrapolation. **NB: this is exactly what Your Kalman filter does!**
- Smear perigee parameters (Choleski decomposition: takes all correlations into account)
- Helix *parameters* exactly calculated, *errors* with one approximation: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

# SGV: How tracking works

**SGV** is a machine to calculate covariance matrices

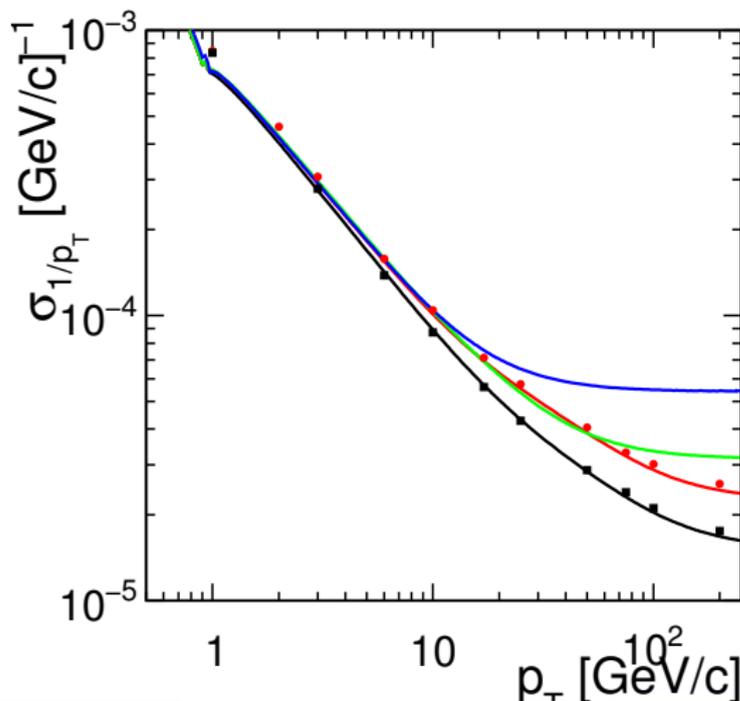
**Tracking:** Follow track-helix through the detector.



- 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.

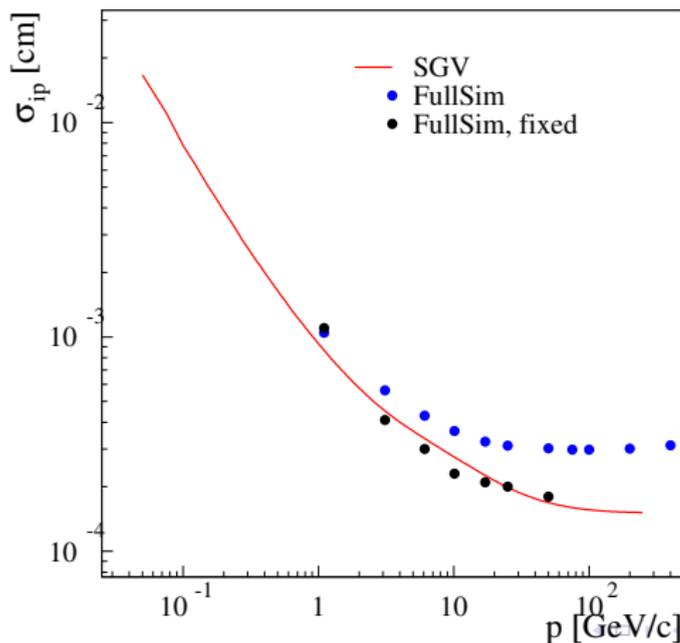
## SGV and FullSim LDC/ILD: momentum resolution

Lines: SGV, dots: Mokka+Marlin



## SGV and FullSim LDC/ILD: ip resolution vs P

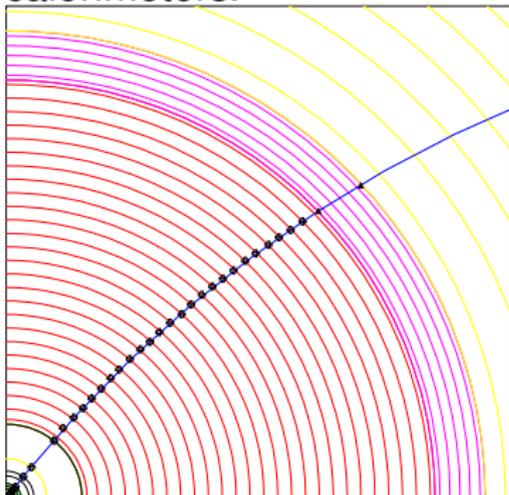
Lines: SGV, dots: Mokka+Marlin



# 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**.

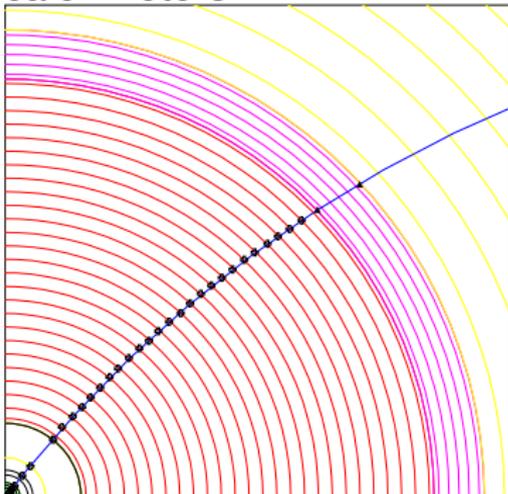
## Other stuff:

- 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.

# SGV: How the rest works

- 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.

# SGV: How the rest works

- 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** → **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.

# Proof of principle of the parametrisation

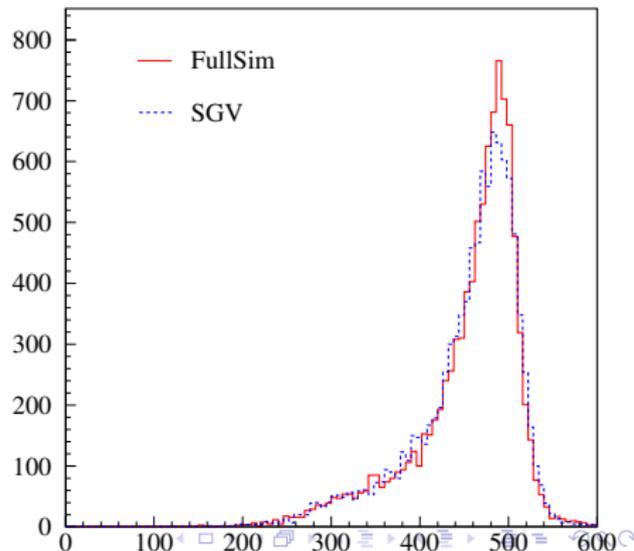
Feed **exactly the same** physics events through FullSim or SGV.

- 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 resolution (NB: r.m.s., including jet-finding uncertainties  $\Rightarrow$  not the standard plot for JER!)
- $Zhh$  at 1 TeV:
  - Visible E
  - Higgs Mass
  - b-tag

# Proof of principle of the parametrisation

Feed **exactly the same** physics events through FullSim or SGV.

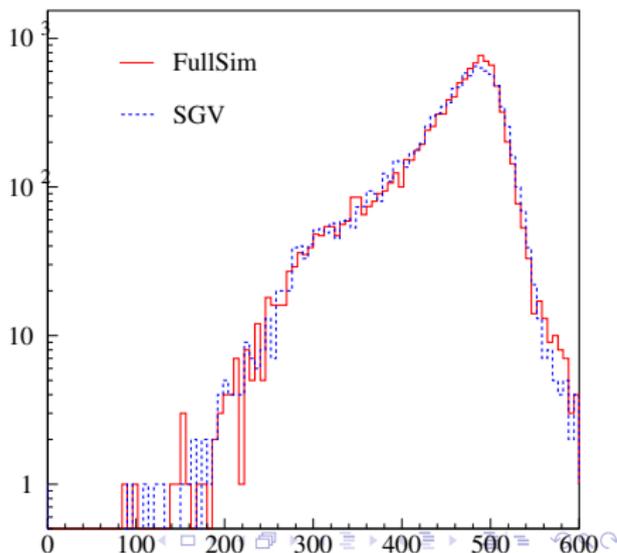
- Overall:
  - Total seen energy
- $e^+e^- \rightarrow ZZ \rightarrow$  four jets:
  - Reconstructed  $M_Z$  at different stages in FullSim.
  - Seen Reconstructed  $M_Z$ , FullSim and SGV.
  - Jet-Energy resolution (NB: r.m.s., including jet-finding uncertainties  $\Rightarrow$  not the standard plot for JER!)
- $Zhh$  at 1 TeV:
  - Visible E
  - Higgs Mass
  - b-tag



# Proof of principle of the parametrisation

Feed **exactly the same** physics events through FullSim or SGV.

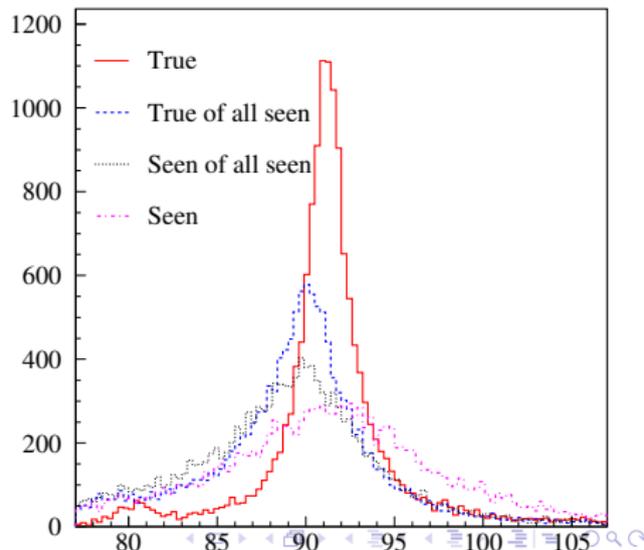
- Overall:
  - Total seen energy
- $e^+e^- \rightarrow ZZ \rightarrow$  four jets:
  - Reconstructed  $M_Z$  at different stages in FullSim.
  - Seen Reconstructed  $M_Z$ , FullSim and SGV.
  - Jet-Energy resolution (NB: r.m.s., including jet-finding uncertainties  $\Rightarrow$  not the standard plot for JER!)
- $Zhh$  at 1 TeV:
  - Visible E
  - Higgs Mass
  - b-tag



# Proof of principle of the parametrisation

Feed **exactly the same** physics events through FullSim or SGV.

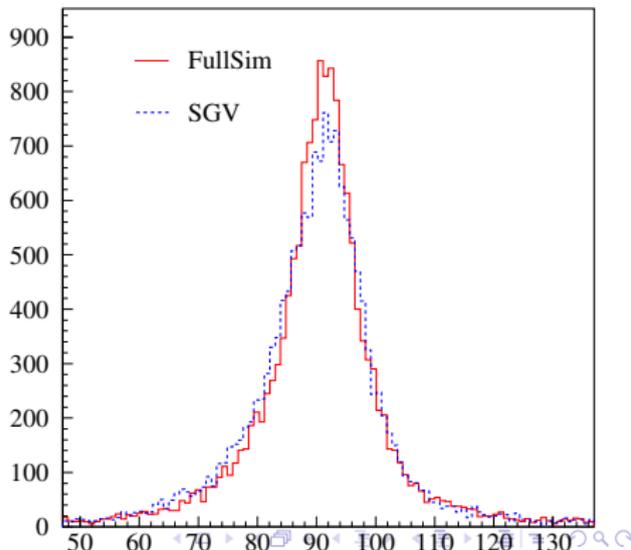
- Overall:
  - Total seen energy
  - $e^+e^- \rightarrow ZZ \rightarrow$  four jets:
    - Reconstructed  $M_Z$  at different stages in FullSim.
    - Seen Reconstructed  $M_Z$ , FullSim and SGV.
    - Jet-Energy resolution (NB: r.m.s., including jet-finding uncertainties  $\Rightarrow$  not the standard plot for JER!)
- $Zhh$  at 1 TeV:
  - Visible E
  - Higgs Mass
  - b-tag



# Proof of principle of the parametrisation

Feed **exactly the same** physics events through FullSim or SGV.

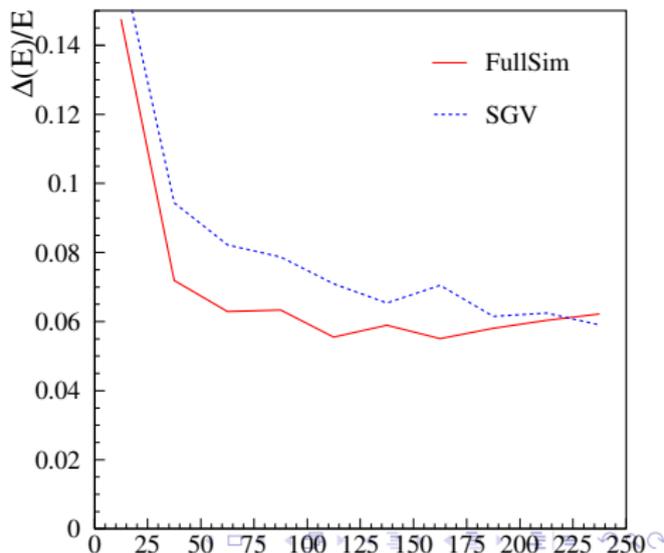
- Overall:
  - Total seen energy
  - $e^+e^- \rightarrow ZZ \rightarrow$  four jets:
    - Reconstructed  $M_Z$  at different stages in FullSim.
    - Seen Reconstructed  $M_Z$ , FullSim and SGV.
    - Jet-Energy resolution (NB: r.m.s., including jet-finding uncertainties  $\Rightarrow$  not the standard plot for JER!)
- $Zhh$  at 1 TeV:
  - Visible E
  - Higgs Mass
  - b-tag



# Proof of principle of the parametrisation

Feed **exactly the same** physics events through FullSim or SGV.

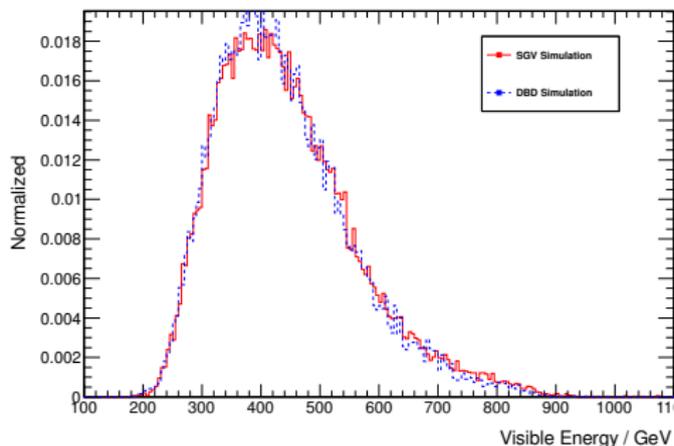
- Overall:
  - Total seen energy
- $e^+e^- \rightarrow ZZ \rightarrow$  four jets:
  - Reconstructed  $M_Z$  at different stages in FullSim.
  - Seen Reconstructed  $M_Z$ , FullSim and SGV.
  - Jet-Energy resolution (NB: r.m.s., including jet-finding uncertainties  $\Rightarrow$  not the standard plot for JER!)
- $Zhh$  at 1 TeV:
  - Visible E
  - Higgs Mass
  - b-tag



# Proof of principle of the parametrisation

Feed **exactly the same** physics events through FullSim or SGV.

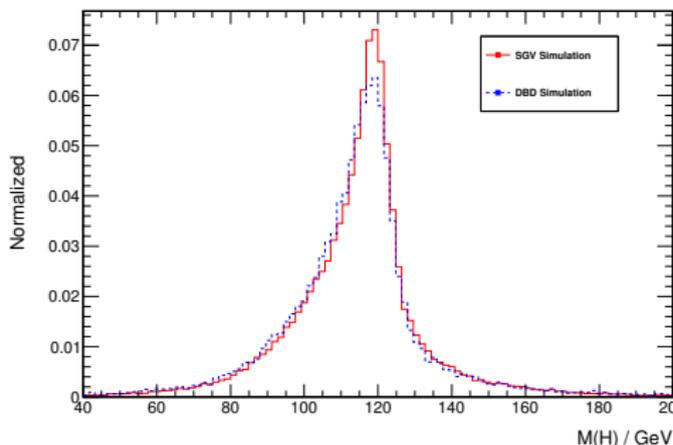
- Overall:
  - Total seen energy
- $e^+e^- \rightarrow ZZ \rightarrow$  four jets:
  - Reconstructed  $M_Z$  at different stages in FullSim.
  - Seen Reconstructed  $M_Z$ , FullSim and SGV.
  - Jet-Energy resolution (NB: r.m.s., including jet-finding uncertainties  $\Rightarrow$  not the standard plot for JER!)
- $Zhh$  at 1 TeV:
  - Visible E
  - Higgs Mass
  - b-tag



# Proof of principle of the parametrisation

Feed **exactly the same** physics events through FullSim or SGV.

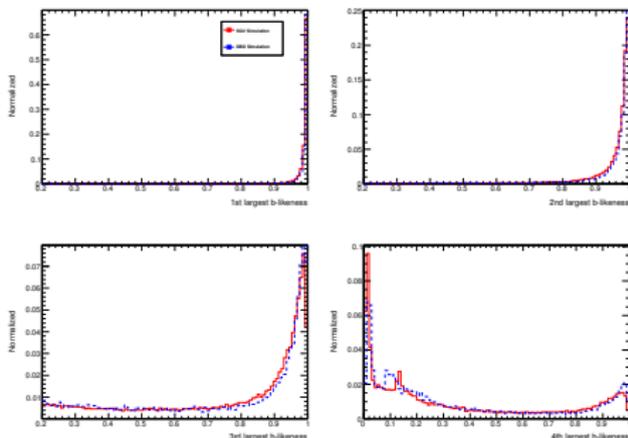
- Overall:
  - Total seen energy
- $e^+e^- \rightarrow ZZ \rightarrow$  four jets:
  - Reconstructed  $M_Z$  at different stages in FullSim.
  - Seen Reconstructed  $M_Z$ , FullSim and SGV.
  - Jet-Energy resolution (NB: r.m.s., including jet-finding uncertainties  $\Rightarrow$  not the standard plot for JER!)
- $Zhh$  at 1 TeV:
  - Visible E
  - Higgs Mass
  - b-tag



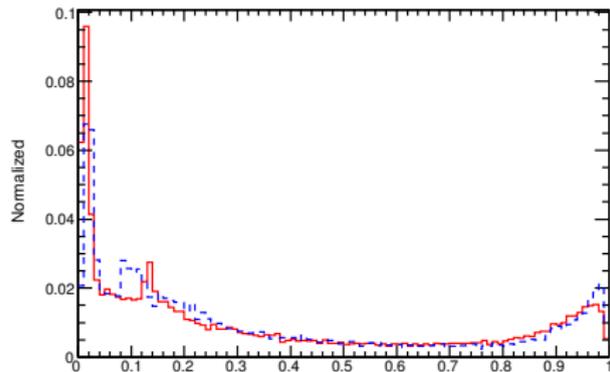
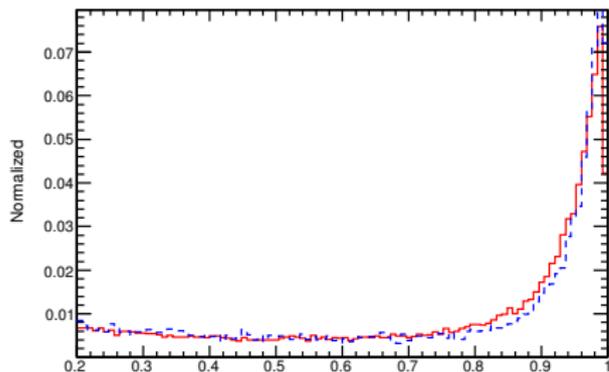
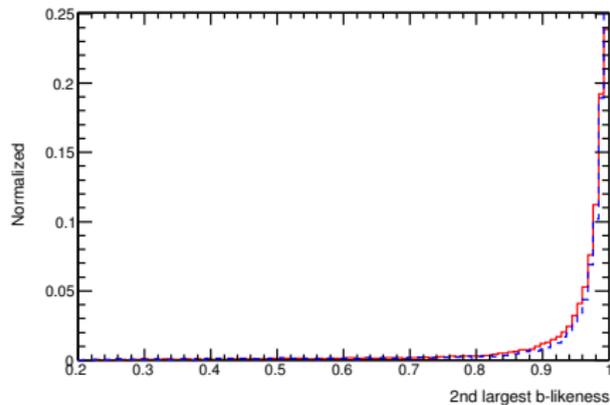
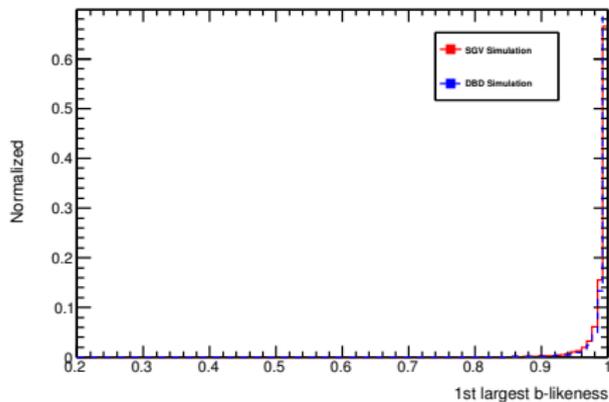
# Proof of principle of the parametrisation

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 resolution (NB: r.m.s., including jet-finding uncertainties  $\Rightarrow$  not the standard plot for JER!)
- $Zhh$  at 1 TeV:
  - Visible E
  - Higgs Mass
  - b-tag



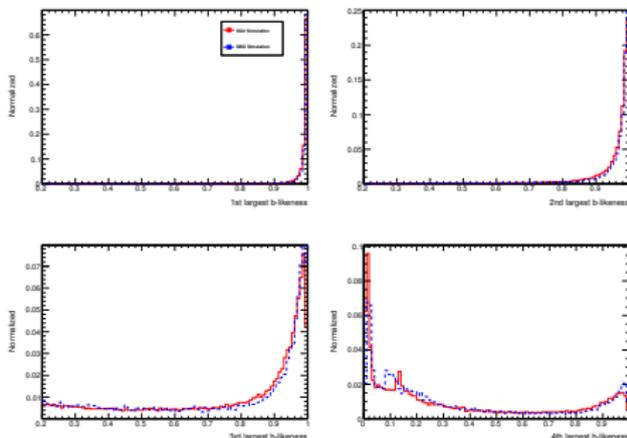
# Proof of principle of the parametrisation

F<sub>1</sub>

# 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 resolution (NB: r.m.s., including jet-finding uncertainties  $\Rightarrow$  not the standard plot for JER!)
- $Zhh$  at 1 TeV:
  - Visible 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 **ROOT ntuples** with "all" information
  - Development on calorimeters.
- 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 **ROOT ntuples** with "all" information
  - Development on calorimeters.
- 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 **ROOT ntuples** with “all” information
  - Development on calorimeters.
- 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 **ROOT ntuples** with “all” information
  - Development on calorimeters.
- 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 ROOT interface set up.

# 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 ROOT interface set up.

# 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 **ROOT** interface set up.

# LCIO DST mass-production

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

- 43 Mevents.
- ~ 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.
- ~ 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.

## ... a tool for rapid LC studies?

Peer-reviewed papers using **SGV**

- Phys. Rev D101 (2020) 7, 075053
- ILD-PHYS-2019-001 (Accepted by Phys. ReV. D)
- Eur.Phys.J.C 76 (2016) 4,183
- Eur.Phys.J.C 75 (2015) 12, 617
- Phys. Rev D 91 (2015) 113007
- Phys. Rev D 90 (2014) 114029
- Phys. Rev D 89 (2014) 11, 113006
- Eur.Phys.J.C 73 (2013) 12,2660
- Eur.Phys.J.C 72 (2012) 2213
- Phys. Rev. D 82 (2010) 055016
- NIM A 579 (2007) 750
- Eur. Phys. J. C 31 (2003) 421
- Eur. Phys, J. direct (2000) 1

+ innumerable theses, reports, arXiv submissions and conference proceedings. Including the **Tesla TDR, Lol, TDR, the IDR and ILC/ILD inputs to EPPSU and Snowmass 2013.**

## ... a tool for rapid LC studies ?

SGV was used for

- Defining the forward tracking geometry of LDC:
  - Vienna 2005. ▶ LDC and ▶ tracking
- The utility (or not) of the silicon envelope
  - ▶ Valencia 2006
- Merge of LDC and GLD into ILD
  - ▶ Cambridge 2008
- Define the options for the IDR
  - ▶ KEK 2015
- Also: **SGV** is part of the FullSim machinery: It is used to select which part of the pairs-background to overlay.

## ... a tool for rapid LC studies ?

SGV was used for

- Defining the forward tracking geometry of LDC:
  - Vienna 2005. ▶ LDC and ▶ tracking
- The utility (or not) of the silicon envelope
  - ▶ Valencia 2006
- Merge of LDC and GLD into ILD
  - ▶ Cambridge 2008
- Define the options for the IDR
  - ▶ KEK 2015
- Also: **SGV** is part of the FullSim machinery: It is used to select which part of the **pairs-background** to overlay.

## ... a tool for rapid LC studies !

SGV was used for

- Defining the forward tracking geometry of LDC:
  - Vienna 2005. ▶ LDC and ▶ tracking
- The utility (or not) of the silicon envelope
  - ▶ Valencia 2006
- Merge of LDC and GLD into ILD
  - ▶ Cambridge 2008
- Define the options for the IDR
  - ▶ KEK 2015
- Also: **SGV** is part of the FullSim machinery: It is used to select which part of the **pairs-background** to overlay.

# Planned developments

- Deleop interface to Root:
  - Also output ROOT objects, not just column-wise ntuples.
  - $\Rightarrow$  DELPHES-like output.
- 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 target
- Also include energy-loss in covariance matrix (not only M.S.)
- Work on particle-flow implementation, CPU-timing profiling, ....

# Planned developments

- Deleop interface to Root:
  - Also output ROOT objects, not just column-wise ntuples.
  - $\Rightarrow$  DELPHES-like output.
- 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 target
- Also include energy-loss in covariance matrix (not only M.S.)
- Work on particle-flow implementation, CPU-timing profiling, ....

# Planned developments

- Deleop interface to Root:
  - Also output ROOT objects, not just column-wise ntuples.
  - $\Rightarrow$  DELPHES-like output.
- 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 target
- Also include energy-loss in covariance matrix (not only M.S.)
- Work on particle-flow implementation, CPU-timing profiling, ....

# Planned developments

- Deleop interface to Root:
  - Also output ROOT objects, not just column-wise ntuples.
  - $\Rightarrow$  DELPHES-like output.
- 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 target
- Also include **energy-loss** in covariance matrix (not only M.S.)
- Work on particle-flow implementation, CPU-timing profiling, ....

# Planned developments

- Deleop interface to Root:
  - Also output ROOT objects, not just column-wise ntuples.
  - $\Rightarrow$  DELPHES-like output.
- 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 target
- Also include **energy-loss** in covariance matrix (not only M.S.)
- Work on particle-flow implementation, CPU-timing profiling, ....

# 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.
- A covariance-machine like **SGV** is needed not to be cornered between the systematic errors of a parameterised fast-sim, and the statistical errors of FullSim

# 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.
- A covariance-machine like **SGV** is needed not to be cornered between the systematic errors of a parameterised fast-sim, and the statistical errors of FullSim

# 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.
- A covariance-machine like **SGV** is needed not to be cornered between the systematic errors of a parameterised fast-sim, and the statistical errors of FullSim

# 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.
- A covariance-machine like **SGV** is needed **not to be cornered** between the systematic errors of a parameterised fast-sim, and the statistical errors of FullSim

# 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.
- A covariance-machine like **SGV** is needed **not to be cornered** between the **systematic errors** of a parameterised fast-sim, and the **statistical errors** of FullSim

# 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.

## Installing SGV

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

Then

```
cd sgv ; . ./install
```

- A covariance-machine like **SGV** is needed **not to be cornered** between the systematic errors of a parameterised fast-sim, and the statistical errors of FullSiM

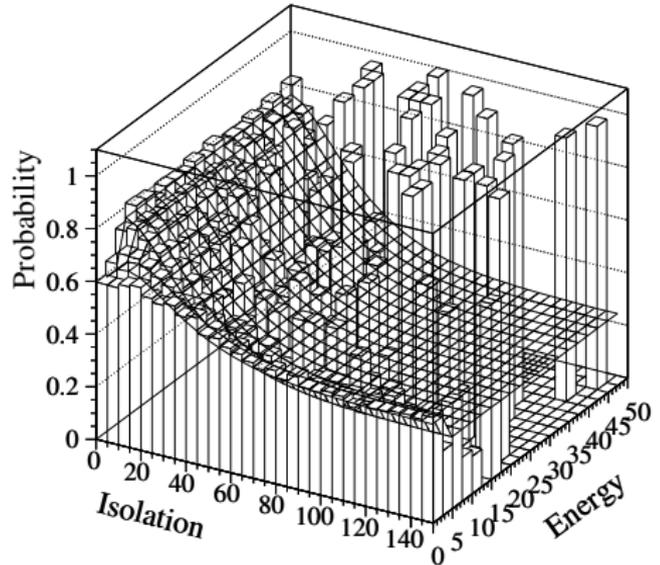
# Thank You !

# Backup

# BACKUP SLIDES

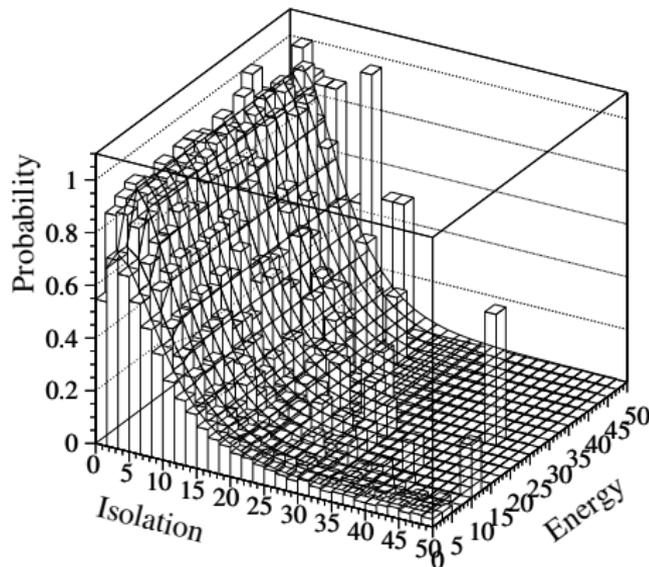
# Observed distributions

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



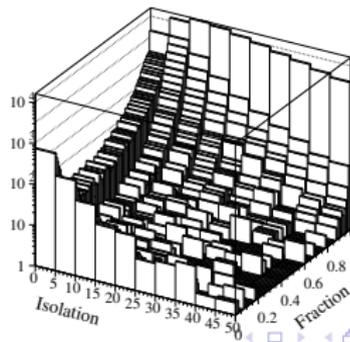
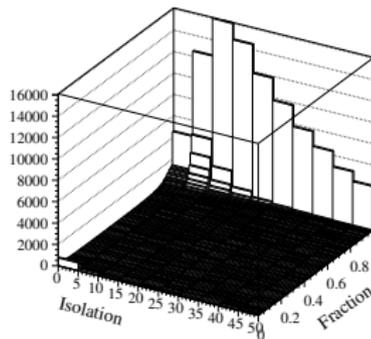
# Observed distributions

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



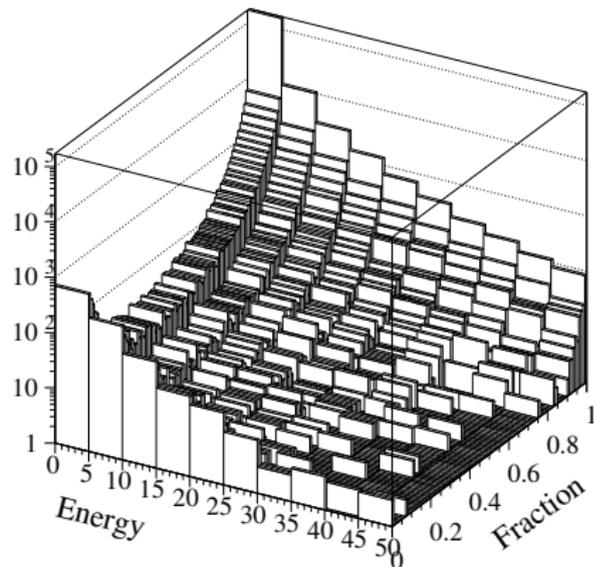
# Observed distributions

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



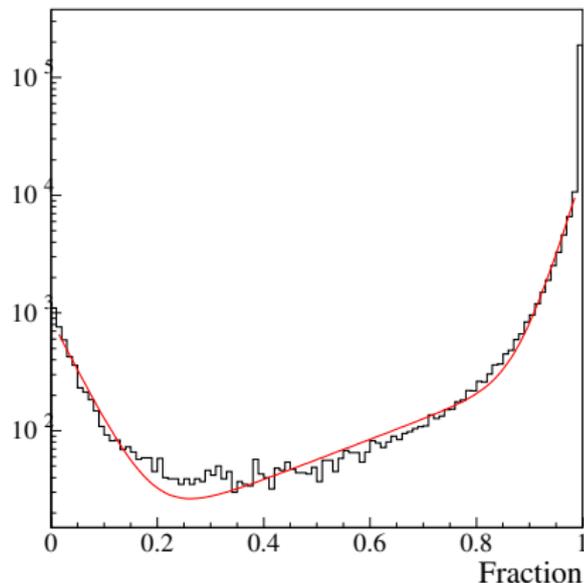
# Observed distributions

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



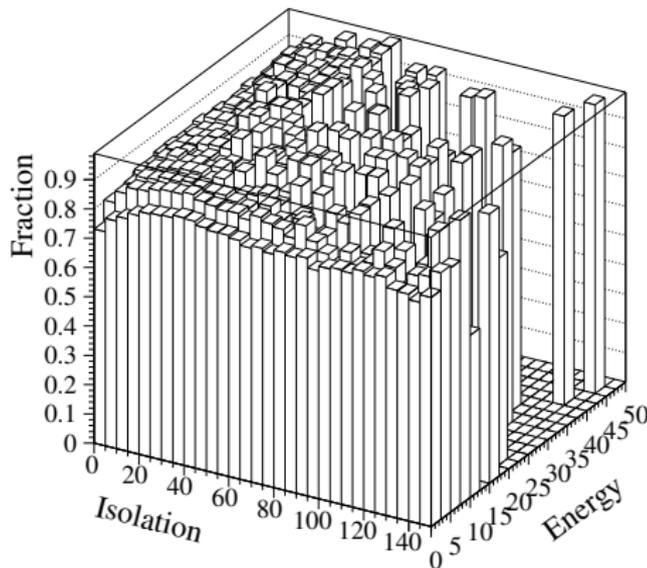
# Observed distributions

- Probability to split (charged had or  $\gamma$ )
- 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  $\gamma$ )
- **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  $e^+e^- \rightarrow \gamma\gamma e^+e^- \rightarrow q\bar{q}e^+e^-$ : **35 nb** (PYTHIA)

- $\int \mathcal{L} dt = 500 \text{ fb}^{-1} \rightarrow 18 \times 10^9$  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  $e^+e^- \rightarrow \gamma\gamma e^+e^- \rightarrow q\bar{q}e^+e^-$ : **35 nb** (PYTHIA)

- $\int \mathcal{L} dt = 500 \text{ fb}^{-1} \rightarrow 18 \times 10^9$  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  $e^+e^- \rightarrow \gamma\gamma e^+e^- \rightarrow q\bar{q}e^+e^-$ : **35 nb** (PYTHIA)

- $\int \mathcal{L} dt = 500 \text{ fb}^{-1} \rightarrow 18 \times 10^9$  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  $\mu$
- Scan each in eg. 20 steps
- Eg. 5000 events per point (modest requirement: in sps1a' almost 1 million SUSY events are expected for  $500 \text{ fb}^{-1}$  !)
- =  $20^4 \times 2 \times 5000 = 1.6 \times 10^9$  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  $\mu$
- Scan each in eg. 20 steps
- Eg. 5000 events per point (modest requirement: in sps1a' almost 1 million SUSY events are expected for  $500 \text{ fb}^{-1}$  !)
- =  $20^4 \times 2 \times 5000 = 1.6 \times 10^9$  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: **energy 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: **energy 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: **energy 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  $\geq 2$  seen charged tracks: do vertex fit.
- Consequence:
  - 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  $e/\pi$ : 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.