Status of event generation and Status of SGV

Mikael Berggren[1]

[1]DESY, Hamburg

ILD SW pre-meeting, Fukuoka, May, 2012

Outline

1. Generator status
2. SGV
   - Calorimeter simulation
3. Summary and Outlook

Generator status: where we are

- After receiving the final beam-parameters at Christmas, the DBD bench-mark generation is soon complete:
- $t\bar{t}h$ is done.
- Beam-strahlung background is done, both at 1 TeV and 500 GeV.
- WW with it's backgrounds is done.
- $\nu\nu h$ with it's backgrounds is done, but not yet on the grid.
- $\gamma\gamma$ backgrounds (both low and high p_T) are done, but not yet on the grid.
- "mini-jets" will be produced at SLAC by the end of the week, and on the grid soon after.
- $t\bar{t}$ at 500 GeV (and other 6 fermions) is lagging behind, but should be done in a month
- 2- and 4-fermions at 500 GeV can be done in a week, once the beam-spectrum files for the new parameters are in place.

Generator status: where we are

- After receiving the final beam-parameters at Christmas, the DBD bench-mark generation is soon complete:
- $t\bar{t}h$ is done.
- Beam-strahlung background is done, both at 1 TeV and 500 GeV.
- WW with it's backgrounds is done.
- $\nu\nu h$ with it's backgrounds is done, but not yet on the grid.
- $\gamma\gamma$ backgrounds (both low and high p_T) are done, but not yet on the grid.
- "mini-jets" will be produced at SLAC by the end of the week, and on the grid soon after.
- $t\bar{t}$ at 500 GeV (and other 6 fermions) is lagging behind, but should be done in a month
- 2- and 4-fermions at 500 GeV can be done in a week, once the beam-spectrum files for the new parameters are in place.

Generator status: where we are

- After receiving the final beam-parameters at Christmas, the DBD bench-mark generation is soon complete:
- $t\bar{t}h$ is done.
- Beam-strahlung background is done, both at 1 TeV and 500 GeV.
- WW with it's backgrounds is done.
- $\nu\nu h$ with it's backgrounds is done, but not yet on the grid.
- $\gamma\gamma$ backgrounds (both low and high p_T) are done, but not yet on the grid.
- "mini-jets" will be produced at SLAC by the end of the week, and on the grid soon after.
- $t\bar{t}$ at 500 GeV (and other 6 fermions) is lagging behind, but should be done in a month
- 2- and 4-fermions at 500 GeV can be done in a week, once the beam-spectrum files for the new parameters are in place.

Generator status: where we are

- After receiving the final beam-parameters at Christmas, the DBD bench-mark generation is soon complete:
- $t\bar{t}h$ is done.
- Beam-strahlung background is done, both at 1 TeV and 500 GeV.
- WW with it's backgrounds is done.
- $\nu\nu h$ with it's backgrounds is done, but not yet on the grid.
- $\gamma\gamma$ backgrounds (both low and high p_T) are done, but not yet on the grid.
- "mini-jets" will be produced at SLAC by the end of the week, and on the grid soon after.
- $t\bar{t}$ at 500 GeV (and other 6 fermions) is lagging behind, but should be done in a month
- 2- and 4-fermions at 500 GeV can be done in a week, once the beam-spectrum files for the new parameters are in place.

Generator status: where we are

- After receiving the final beam-parameters at Christmas, the DBD bench-mark generation is soon complete:
- $t\bar{t}h$ is done.
- Beam-strahlung background is done, both at 1 TeV and 500 GeV.
- *WW* with it's backgrounds is done.
- $\nu\nu h$ with it's backgrounds is done, but not yet on the grid.
- $\gamma\gamma$ backgrounds (both low and high p_T) are done, but not yet on the grid.
- "mini-jets" will be produced at SLAC by the end of the week, and on the grid soon after.
- $t\bar{t}$ at 500 GeV (and other 6 fermions) is lagging behind, but should be done in a month
- 2- and 4-fermions at 500 GeV can be done in a week, once the beam-spectrum files for the new parameters are in place.

Generator status: where we are

- After receiving the final beam-parameters at Christmas, the DBD bench-mark generation is soon complete:
- $t\bar{t}h$ is done.
- Beam-strahlung background is done, both at 1 TeV and 500 GeV.
- *WW* with it's backgrounds is done.
- $\nu\nu h$ with it's backgrounds is done, but not yet on the grid.
- $\gamma\gamma$ backgrounds (both low and high p_T) are done, but not yet on the grid.
- "mini-jets" will be produced at SLAC by the end of the week, and on the grid soon after.
- $t\bar{t}$ at 500 GeV (and other 6 fermions) is lagging behind, but should be done in a month
- 2- and 4-fermions at 500 GeV can be done in a week, once the beam-spectrum files for the new parameters are in place.

Generator status: where we are

- After receiving the final beam-parameters at Christmas, the DBD bench-mark generation is soon complete:
- $t\bar{t}h$ is done.
- Beam-strahlung background is done, both at 1 TeV and 500 GeV.

File locations:

Sub-directories of:
lfn:/grid/ilc/prod/ilc/mc-dbd/generated/1000-B1b_ws/
lfn:/grid/ilc/prod/ilc/mc-dbd/generated/xxx-TDR_ws/
(xxx=250, 350, 500)

- "mini-jets" will be produced at SLAC by the end of the week, and

Information on samples:

http://ilcsoft.desy.de/dbd/generated/

be done in a month.

- 2- and 4-fermions at 500 GeV can be done in a week, once the beam-spectrum files for the new parameters are in place.

Generator status

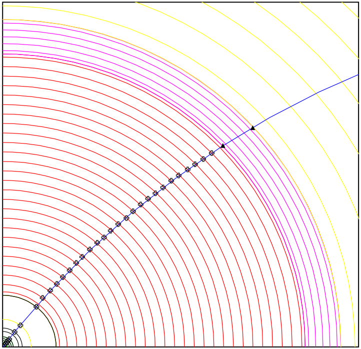
This was just the bottom line !!

See the talk tomorrow in the “Physics analysis” slot at 14:00 for the details.

SGV: How tracking works

SGV is a machine to calculate covariance matrices

Tracking: Follow track-helix through the detector.

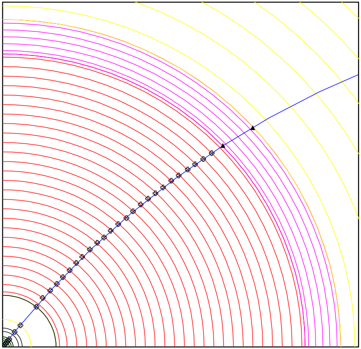


- Calculate cov. mat. at perigee, including material, measurement errors and extrapolation. NB: this is exactly what Your track fit does!
- Smear perigee parameters (Choleski decomposition: takes all correlations into account)
- Information on hit-pattern accessible to analysis. Co-ordinates of hits accessible.

SGV: How tracking works

SGV is a machine to calculate covariance matrices

Tracking: Follow track-helix through the detector.

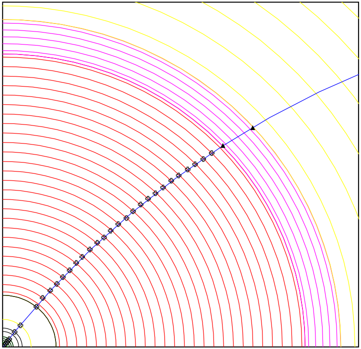


- Calculate cov. mat. at perigee, including material, measurement errors and extrapolation. NB: this is exactly what Your track fit does!
- Smear perigee parameters (Choleski decomposition: takes all correlations into account)
- Information on hit-pattern accessible to analysis. Co-ordinates of hits accessible.

SGV: How tracking works

SGV is a machine to calculate covariance matrices

Tracking: Follow track-helix through the detector.

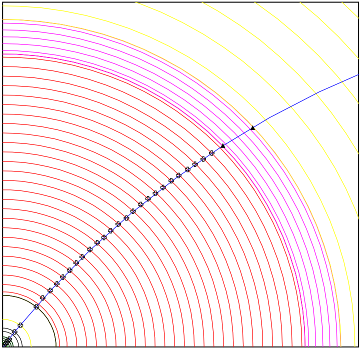


- Calculate cov. mat. at perigee, including material, measurement errors and extrapolation. NB: this is exactly what Your track fit does!
- Smear perigee parameters (Choleski decomposition: takes all correlations into account)
- Information on hit-pattern accessible to analysis. Co-ordinates of hits accessible.

SGV: How tracking works

SGV is a machine to calculate covariance matrices

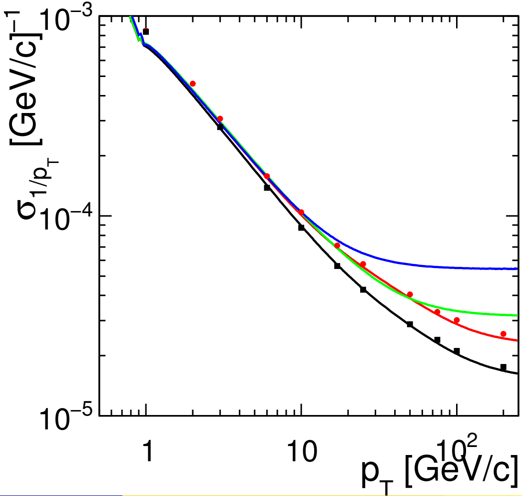
Tracking: Follow track-helix through the detector.



- Calculate cov. mat. at perigee, including material, measurement errors and extrapolation. NB: this is exactly what Your track fit does!
- Smear perigee parameters (Choleski decomposition: takes all correlations into account)
- Information on hit-pattern accessible to analysis. Co-ordinates of hits accessible.

SGV and FullSim LDC/ILD: momentum resolution

Lines: SGV, dots: Mokka+Marlin



SGV: How the rest works

Calorimeters:

- Follow particle to intersection with calorimeters. Simulate:
  - Response type: MIP, EM-shower, hadronic shower, below threshold, etc.
  - Simulate response from parameters.

Other stuff:

- EM-interactions in detector material simulated
- Plug-ins for particle identification, track-finding efficiencies,...

SGV: How the rest works

Calorimeters:

- Follow particle to intersection with calorimeters. Simulate:
  - Response type: MIP, EM-shower, hadronic shower, below threshold, etc.
  - Simulate response from parameters.

Other stuff:

- EM-interactions in detector material simulated
- Plug-ins for particle identification, track-finding efficiencies,...

SGV: Technicalities

Features:

- Written in Fortran 95.
- 20 000 lines + 10 000 lines of comments.
- Some CERNLIB dependence.
- Re-write of battle-tested f77 SGV 2-series (LEP, Tesla, LOI, ...)
- Managed in SVN.Install script included, much ameliorated in the HEAD version.
- Callable PYTHIA, Whizard or input from PYJETS or stdhep.
- Output of generated event to PYJETS or stdhep.
- samples subdirectory with READMEs, steering and code.
- output LCIO DST.

Installing SGV

Do

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

Then

```
cd sgv ; bash 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 STDHEP installed.
  - Get CERNLIB installed in native 64bit.
  - Get Whizard (basic or ILC-tuned) installed.
  - Get the LCIO-DST writer set up

Installing SGV

Do

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

Then

```
cd sgv ; bash 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 STDHEP installed.
    - Get CERNLIB installed in native 64bit.
    - Get Whizard (basic or ILC-tuned) installed.
    - Get the LCIO-DST writer set up

Installing SGV

Do

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

Then

```
cd sgv ; bash 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 STDHEP installed.
  - Get CERNLIB installed in native 64bit.
  - Get Whizard (basic or ILC-tuned) installed.
  - Get the LCIO-DST writer set up

Calorimeter simulation

The issues:

- Clearly: Random E, shower position, shower shape.
- But also association errors:
  - Clusters might merge.
  - Clusters might split.
  - Clusters might get wrongly associated to tracks.
- 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.
  - Other errors (split neutral cluster, charged cluster associated with wrong track ....) are of less importance.

Calorimeter simulation

The issues:

- Clearly: Random E, shower position, shower shape.
- But also association errors:
  - Clusters might merge.
  - Clusters might split.
  - Clusters might get wrongly associated to tracks.
- 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.
  - Other errors (split neutral cluster, charged cluster associated with wrong track ....) are of less importance.

Calorimeter simulation

The issues:

- Clearly: Random E, shower position, shower shape.
- But also association errors:
  - Clusters might merge.
  - Clusters might split.
  - Clusters might get wrongly associated to tracks.
- 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.
  - Other errors (split neutral cluster, charged cluster associated with wrong track ....) are of less importance.

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.

Observations

- Identify and factorise:
  1. Probability to split
  2. If split, probability to split off/merge the entire cluster.
  3. If split, but not 100 %: Form of the p.d.f. of the fraction split off.
- Depends on:
  1. Isolation - strongly for merging, slightly for splitting - but can be treated in two energy bins with no energy dependence in the bin. %5 over-all dependence on barrel/endcap.
  2. Energy. Is small for splitting, important for merging at low E.
  3. Energy and isolation (very little for splitting) via the average.
- All cases (EM/had - split/merge - Barrel/endcap) can be described by the same functional shapes.
- Functions are combinations of exponentials and lines.
- 28 parameters × 4 cases (em/had × double-counting/loss)
- See http://ilcagenda.linearcollider.org/getFile.py/access?contribId=5&resId=0&materialId=slides&confId=5445

Observations

- Identify and factorise:
  1. Probability to split
  2. If split, probability to split off/merge the entire cluster.
  3. If split, but not 100 %: Form of the p.d.f. of the fraction split off.
- Depends on:
  1. Isolation - strongly for merging, slightly for splitting - but can be treated in two energy bins with no energy dependence in the bin. %5 over-all dependence on barrel/endcap.
  2. Energy. Is small for splitting, important for merging at low E.
  3. Energy and isolation (very little for splitting) via the average.
- All cases (EM/had - split/merge - Barrel/endcap) can be described by the same functional shapes.
- Functions are combinations of exponentials and lines.
- 28 parameters $\times$ 4 cases (em/had $\times$ double-counting/loss)
- See http://ilcagenda.linearcollider.org/getFile.py/access?contribId=5&resId=0&materialId=slides&confId=5445

Observations

- Identify and factorise:
  1. Probability to split
  2. If split, probability to split off/merge the entire cluster.
  3. If split, but not 100 %: Form of the p.d.f. of the fraction split off.
- Depends on:
  1. Isolation - strongly for merging, slightly for splitting - but can be treated in two energy bins with no energy dependence in the bin. %5 over-all dependence on barrel/endcap.
  2. Energy. Is small for splitting, important for merging at low E.
  3. Energy and isolation (very little for splitting) via the average.
- All cases (EM/had - split/merge - Barrel/endcap) can be described by the same functional shapes.
- Functions are combinations of exponentials and lines.
- 28 parameters × 4 cases (em/had × double-counting/loss)
- See http://ilcagenda.linearcollider.org/getFile.py/access?contribId=5&resId=0&materialId=slides&confId=5445

Observations

- Identify and factorise:
  1. Probability to split
  2. If split, probability to split off/merge the entire cluster.
  3. If split, but not 100 %: Form of the p.d.f. of the fraction split off.
- Depends on:
  1. Isolation - strongly for merging, slightly for splitting - but can be treated in two energy bins with no energy dependence in the bin. %5 over-all dependence on barrel/endcap.
  2. Energy. Is small for splitting, important for merging at low E.
  3. Energy and isolation (very little for splitting) via the average.
- All cases (EM/had - split/merge - Barrel/endcap) can be described by the same functional shapes.
- Functions are combinations of exponentials and lines.
- 28 parameters × 4 cases (em/had × double-counting/loss)
- See http://ilcagenda.linearcollider.org/getFile.py/access?contribId=5&resId=0&materialId=slides&confId=5445

Observations

- Identify and factorise:
  1. Probability to split
  2. If split, probability to split off/merge the entire cluster.
  3. If split, but not 100 %: Form of the p.d.f. of the fraction split off.
- Depends on:
  1. Isolation - strongly for merging, slightly for splitting - but can be treated in two energy bins with no energy dependence in the bin. %5 over-all dependence on barrel/endcap.
  2. Energy. Is small for splitting, important for merging at low E.
  3. Energy and isolation (very little for splitting) via the average.
- All cases (EM/had - split/merge - Barrel/endcap) can be described by the same functional shapes.
- Functions are combinations of exponentials and lines.
- 28 parameters $\times$ 4 cases (em/had $\times$ double-counting/loss)
- See http://ilcagenda.linearcollider.org/getFile.py/access?contribId=5&resId=0&materialId=slides&confId=5445

Observations

- Identify and factorise:
  1. Probability to split
  2. If split, probability to split off/merge the entire cluster.
  3. If split, but not 100 %: Form of the p.d.f. of the fraction split off.
- Depends on:
  1. Isolation - strongly for merging, slightly for splitting - but can be treated in two energy bins with no energy dependence in the bin. %5 over-all dependence on barrel/endcap.
  2. Energy. Is small for splitting, important for merging at low E.
  3. Energy and isolation (very little for splitting) via the average.
- All cases (EM/had - split/merge - Barrel/endcap) can be described by the same functional shapes.
- Functions are combinations of exponentials and lines.
- 28 parameters $\times$ 4 cases (em/had $\times$ double-counting/loss)
- See http://ilcagenda.linearcollider.org/getFile.py/access?contribId=5&resId=0&materialId=slides&confId=5445

Parametrisation in SGV

Call ZAUPFL in ZAUSER (instead of ZAUSHO which is for course-grained calorimeters):

- On first call: Reads and stores parameters
- ZAUMKC makes clusters from simulated showers of each particle.
- Loop (once) over all particles.
  - Determine shortest distance to neighbour of right type and charge in the same calorimeter. Fortran95's whole-array capabilities very useful to produce efficient code !
  - Simulate:
    - No split or complete split or partial split
    - If partial: fraction split off
    - Compare E and p. If too different, try again a few times.
  - Split shower - combine cluster.
- Remove and push out any empty showers or clusters. Adjust pointers.

Analyse as before: Charged particles from module ZATRS, neutrals from ZACLU.

Parametrisation in SGV

Call ZAUPFL in ZAUSER (instead of ZAUSHO which is for course-grained calorimeters):

- On first call: Reads and stores parameters
- ZAUMKC makes clusters from simulated showers of each particle.
- Loop (once) over all particles.
  - Determine shortest distance to neighbour of right type and charge in the same calorimeter. Fortran95's whole-array capabilities very useful to produce efficient code !
  - Simulate:
    - No split or complete split or partial split
    - If partial: fraction split off
    - Compare E and p. If too different, try again n failures
  - Split shower - combine cluster.
- Remove and push out any empty showers or clusters. Adjust pointers.

Analyse as before: Charged particles from module ZATRS, neutrals from ZACLU.

Parametrisation in SGV

Call ZAUPFL in ZAUSER (instead of ZAUSHO which is for course-grained calorimeters):

- On first call: Reads and stores parameters
- ZAUMKC makes clusters from simulated showers of each particle.
- Loop (once) over all particles.
  - Determine shortest distance to neighbour of right type and charge in the same calorimeter. Fortran95's whole-array capabilities very useful to produce efficient code !
  - Simulate:
    - No split or complete split or partial split
    - If partial: fraction split off
    - Compare E and p: If too different, try again a few times
  - Split shower - combine cluster.
- Remove and push out any empty showers or clusters. Adjust pointers.

Analyse as before: Charged particles from module ZATRS, neutrals from ZACLU.

Parametrisation in SGV

Call ZAUPFL in ZAUSER (instead of ZAUSHO which is for course-grained calorimeters):

- On first call: Reads and stores parameters
- ZAUMKC makes clusters from simulated showers of each particle.
- Loop (once) over all particles.
  - Determine shortest distance to neighbour of right type and charge in the same calorimeter. Fortran95's whole-array capabilities very useful to produce efficient code !
  - Simulate:
    - No split or complete split or partial split
    - If partial: fraction split off
    - Compare E and p: If too different, try again a few times
  - Split shower - combine cluster.
- Remove and push out any empty showers or clusters. Adjust pointers.

Analyse as before: Charged particles from module ZATRS, neutrals from ZACLU.

Parametrisation in SGV

Call ZAUPFL in ZAUSER (instead of ZAUSHO which is for course-grained calorimeters):

- On first call: Reads and stores parameters
- ZAUMKC makes clusters from simulated showers of each particle.
- Loop (once) over all particles.
  - Determine shortest distance to neighbour of right type and charge in the same calorimeter. Fortran95's whole-array capabilities very useful to produce efficient code !
  - Simulate:
    - No split or complete split or partial split
    - If partial: fraction split off
    - Compare E and p: If too different, try again a few times
  - Split shower - combine cluster.
- Remove and push out any empty showers or clusters. Adjust pointers.

Analyse as before: Charged particles from module ZATRS, neutrals from ZACLU.

Parametrisation in SGV

Call ZAUPFL in ZAUSER (instead of ZAUSHO which is for course-grained calorimeters):

- On first call: Reads and stores parameters
- ZAUMKC makes clusters from simulated showers of each particle.
- Loop (once) over all particles.
  - Determine shortest distance to neighbour of right type and charge in the same calorimeter. Fortran95's whole-array capabilities very useful to produce efficient code !
  - Simulate:
    - No split or complete split or partial split
    - If partial: fraction split off
    - Compare E and p: If too different, try again a few times
  - Split shower - combine cluster.
- Remove and push out any empty showers or clusters. Adjust pointers.

Analyse as before: Charged particles from module ZATRS, neutrals from ZACLU.

Parametrisation in SGV

Call ZAUPFL in ZAUSER (instead of ZAUSHO which is for course-grained calorimeters):

- On first call: Reads and stores parameters
- ZAUMKC makes clusters from simulated showers of each particle.
- Loop (once) over all particles.
  - Determine shortest distance to neighbour of right type and charge in the same calorimeter. Fortran95's whole-array capabilities very useful to produce efficient code !
  - Simulate:
    - No split or complete split or partial split
    - If partial: fraction split off
    - Compare E and p: If too different, try again a few times
  - Split shower - combine cluster.
- Remove and push out any empty showers or clusters. Adjust pointers.

Analyse as before: Charged particles from module ZATRS, neutrals from ZACLU.

Parametrisation in SGV

Call ZAUPFL in ZAUSER (instead of ZAUSHO which is for course-grained calorimeters):

- On first call: Reads and stores parameters
- ZAUMKC makes clusters from simulated showers of each particle.
- Loop (once) over all particles.
  - Determine shortest distance to neighbour of right type and charge in the same calorimeter. Fortran95's whole-array capabilities very useful to produce efficient code !
  - Simulate:
    - No split or complete split or partial split
    - If partial: fraction split off
    - Compare E and p: If too different, try again a few times
  - Split shower - combine cluster.
- Remove and push out any empty showers or clusters. Adjust pointers.

Analyse as before: Charged particles from module ZATRS, neutrals from ZACLU.

Parametrisation in SGV

Call ZAUPFL in ZAUSER (instead of ZAUSHO which is for course-grained calorimeters):

- On first call: Reads and stores parameters
- ZAUMKC makes clusters from simulated showers of each particle.
- Loop (once) over all particles.
  - Determine shortest distance to neighbour of right type and charge in the same calorimeter. Fortran95's whole-array capabilities very useful to produce efficient code !
  - Simulate:
    - No split or complete split or partial split
    - If partial: fraction split off
    - Compare E and p: If too different, try again a few times
  - Split shower - combine cluster.
- Remove and push out any empty showers or clusters. Adjust pointers.

Analyse as before: Charged particles from module ZATRS, neutrals from ZACLU.

Parametrisation in SGV

Call ZAUPFL in ZAUSER (instead of ZAUSHO which is for course-grained calorimeters):

- On first call: Reads and stores parameters
- ZAUMKC makes clusters from simulated showers of each particle.
- Loop (once) over all particles.
  - Determine shortest distance to neighbour of right type and charge in the same calorimeter. Fortran95's whole-array capabilities very useful to produce efficient code !
  - Simulate:
    - No split or complete split or partial split
    - If partial: fraction split off
    - Compare E and p: If too different, try again a few times
  - Split shower - combine cluster.
- Remove and push out any empty showers or clusters. Adjust pointers.

Analyse as before: Charged particles from module ZATRS, neutrals from ZACLU.

Parametrisation in SGV

Call ZAUPFL in ZAUSER (instead of ZAUSHO which is for course-grained calorimeters):

- On first call: Reads and stores parameters
- ZAUMKC makes clusters from simulated showers of each particle.
- Loop (once) over all particles.
  - Determine shortest distance to neighbour of right type and charge in the same calorimeter. Fortran95's whole-array capabilities very useful to produce efficient code !
  - Simulate:
    - No split or complete split or partial split
    - If partial: fraction split off
    - Compare E and p: If too different, try again a few times
  - Split shower - combine cluster.
- Remove and push out any empty showers or clusters. Adjust pointers.

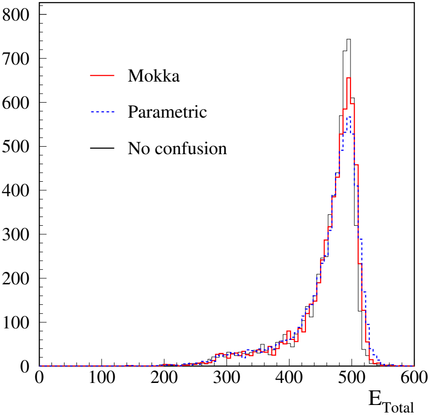
Analyse as before: Charged particles from module ZATRS, neutrals from ZACLU.

Checking the parametrisation

- Twiddle knobs: E vs p, overall split - probability.
- Total seen energy
- Total neutral energy
- Lost and double counted energy.

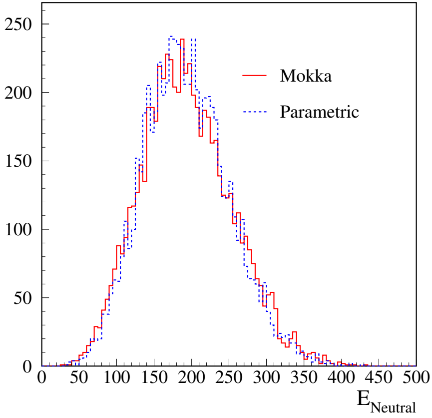
Checking the parametrisation

- Twiddle knobs: E vs p, overall split - probability.
- Total seen energy
- Total neutral energy
- Lost and double counted energy.



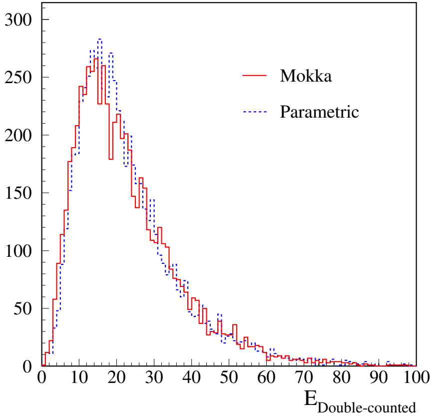
Checking the parametrisation

- Twiddle knobs: E vs p, overall split - probability.
- Total seen energy
- Total neutral energy
- Lost and double counted energy.



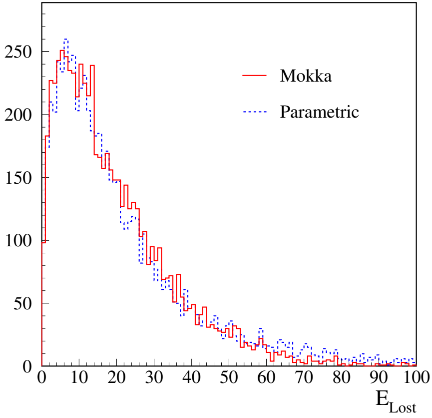
Checking the parametrisation

- Twiddle knobs: E vs p, overall split - probability.
- Total seen energy
- Total neutral energy
- Lost and double counted energy.



Checking the parametrisation

- Twiddle knobs: E vs p, overall split - probability.
- Total seen energy
- Total neutral energy
- Lost and double counted energy.

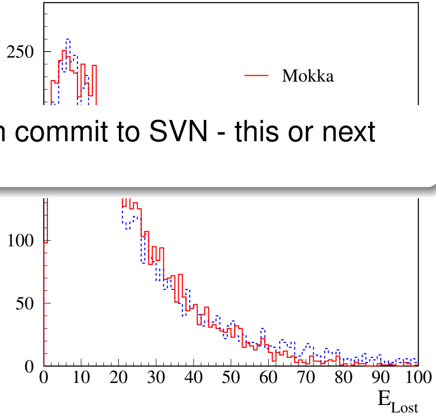


Checking the parametrisation

- Twiddle knobs: E vs p, overall split - probability.
- Total seen energy

Looks good ! Some further tests, then commit to SVN - this or next week.

energy.



Summary

- The status of the SGV program was presented.
- The particle-flow parametrisation, presented in earlier meetings, has been integrated into SGV.
- First comparisons to Mokka/Marlin with a first tentative tuning was shown.

Installing SGV

svn export https://svnsrv.desy.de/public/sgv/trunk/ sgv/
Then
cd sgv ; . ./install

Summary

- The status of the SGV program was presented.
- The particle-flow parametrisation, presented in earlier meetings, has been integrated into SGV.
- First comparisons to Mokka/Marlin with a first tentative tuning was shown.

Installing SGV

svn export https://svnsrv.desy.de/public/sgv/trunk/ sgv/
Then
cd sgv ; . ./install

Summary

- The status of the SGV program was presented.
- The particle-flow parametrisation, presented in earlier meetings, has been integrated into SGV.
- First comparisons to Mokka/Marlin with a first tentative tuning was shown.

Installing SGV

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

Then

cd sgv ; . ./install

Summary

- The status of the SGV program was presented.
- The particle-flow parametrisation, presented in earlier meetings, has been integrated into SGV.
- First comparisons to Mokka/Marlin with a first tentative tuning was shown.

Installing SGV

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

Then

cd sgv ; . ./install

Outlook

(On time-scale days to weeks)

- 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
- Finish up particle flow parametrisation.
- Fix a few identified issues, then Release SGV3.0 (no rc1).
- Add secondary vertecies to LCIO output.
- Produce LCIO DST:s for the DBD bench-marks: DBD analyses can start $\approx$ now, while waiting for full-sim.

Outlook

(On time-scale days to weeks)

- 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
- Finish up particle flow parametrisation.
- Fix a few identified issues, then Release SGV3.0 (no rc1).
- Add secondary vertecies to LCIO output.
- Produce LCIO DST:s for the DBD bench-marks: DBD analyses can start ≈ now, while waiting for full-sim.

Outlook

(On time-scale days to weeks)

- 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
- Finish up particle flow parametrisation.
- Fix a few identified issues, then Release SGV3.0 (no rc1).
- Add secondary vertecies to LCIO output.
- Produce LCIO DST:s for the DBD bench-marks: DBD analyses can start $\approx$ now, while waiting for full-sim.

Outlook

(On time-scale days to weeks)

- 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
- Finish up particle flow parametrisation.
- Fix a few identified issues, then Release SGV3.0 (no rc1).
- Add secondary vertecies to LCIO output.
- Produce LCIO DST:s for the DBD bench-marks: DBD analyses can start $\approx$ now, while waiting for full-sim.

Outlook

(On time-scale days to weeks)

- 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
- Finish up particle flow parametrisation.
- Fix a few identified issues, then Release SGV3.0 (no rc1).
- Add secondary vertecies to LCIO output.
- Produce LCIO DST:s for the DBD bench-marks: DBD analyses can start ≈ now, while waiting for full-sim.

Outlook

(On time-scale days to weeks)

- 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
- Finish up particle flow parametrisation.
- Fix a few identified issues, then Release SGV3.0 (no rc1).
- Add secondary vertecies to LCIO output.
- Produce LCIO DST:s for the DBD bench-marks: DBD analyses can start ≈ now, while waiting for full-sim.

Outlook

(On time-scale days to weeks)

- 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
- Finish up particle flow parametrisation.
- Fix a few identified issues, then Release SGV3.0 (no rc1).
- Add secondary vertecies to LCIO output.
- Produce LCIO DST:s for the DBD bench-marks: DBD analyses can start $\approx$ now, while waiting for full-sim.

Thank You !

Backup

BACKUP SLIDES

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

- 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.

- Further reduce CERNLIB dependence - at a the cost of backward compatibility on steering files ? HBOOK dependence will remain in the forseable future - but only for user convenience : SGV itself doesn't need it.