

Overview of Benchmarking Tools

Tim Barklow

SLAC

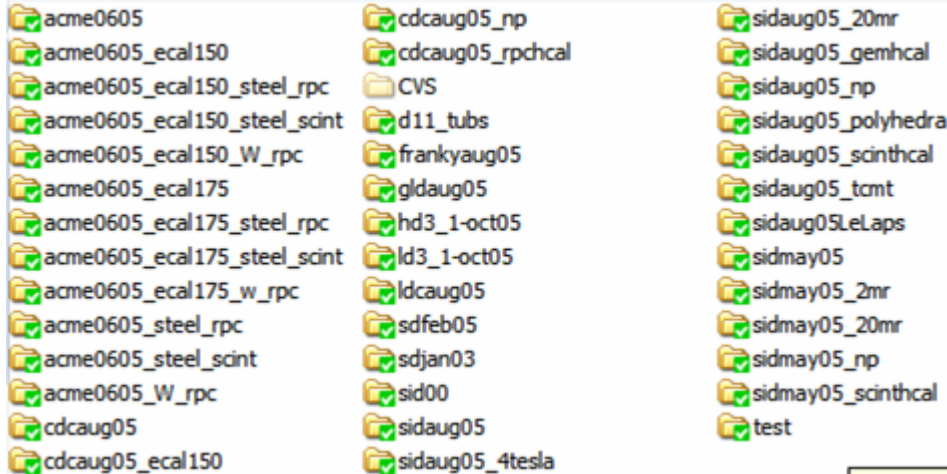
Oct 10, 2006

org.lcsim Goals

- “Second generation” ILC reconstruction/analysis framework
 - Builds on hep.lcd framework used since 1999
 - Full suite of reconstruction and analysis tools
 - Uses LCIO for IO and as basis for simulation, raw data and reconstruction event formats
 - Isolate users from raw LCIO structures
 - Maintain full interoperability with other LCIO based packages
 - Detector Independence
 - Make package independent of detector, geometry assumptions so can work with any detector
 - Read properties of detectors at runtime
 - Written using Java (1.5)
 - High-performance but simple, easy to learn, OO language
 - Enables us last 10 years of software developments in the “real world”
 - Ability to run standalone (command line or batch) or in JAS3 or IDE such as Netbeans, Eclipse
-

Available Detector Descriptions

- Although detector descriptions can live anywhere we maintain a CVS repository of detector descriptions
 - Exported to org.lcsim web site for automatic download
- 40 detector variants as of July 2006
- Many SiD variants, but also some gld, ldc



- You are welcome to contribute more

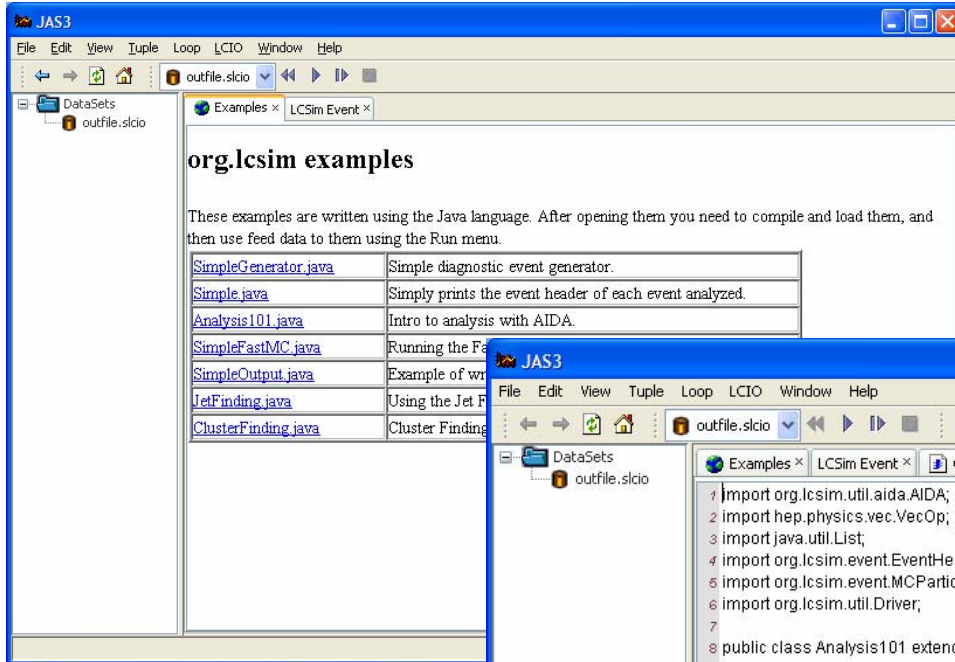
Org.lcsim Reconstruction

- Reconstruction package includes:
 - Physics utilities:
 - **Jet finders, event shape routines**
 - **Diagnostic event generator, stdhep reader/translator**
 - Histogramming/Fitting/Plotting (AIDA based)
 - Event Display
 - **Processor/Driver infrastructure**
 - **Fast MC**
 - **Track/Cluster smearing**
 - Reconstruction
 - Cheaters (perfect reconstruction)
 - Detector Response
 - CCDSim, Digisim
 - Clustering Algorithms
 - Cheater, DirectedTree, NearestNeighbour, Cone
 - Tracking Finding/Fitting Algorithms
 - TRF,
 - Muon Finding, Swimming
 - **Vertex Finding (ZvTop)**

Using org.lcsim with JAS3

- The org.lcsim can be used standalone, within an IDE, or inside JAS3. Same code can be used in all modes, so easy to move back and forth
 - E.g. develop in IDE and run in JAS3
 - E.g. develop in JAS3 and run in batch
 - JAS3 org.lcsim plugin adds:
 - Example Analysis Code
 - org.lcsim Event browser
 - Easy viewing of analysis plots
 - WIRED event display integration
-

org.lcsim: Examples



JAS3

File Edit View Tuple Loop LCIO Window Help

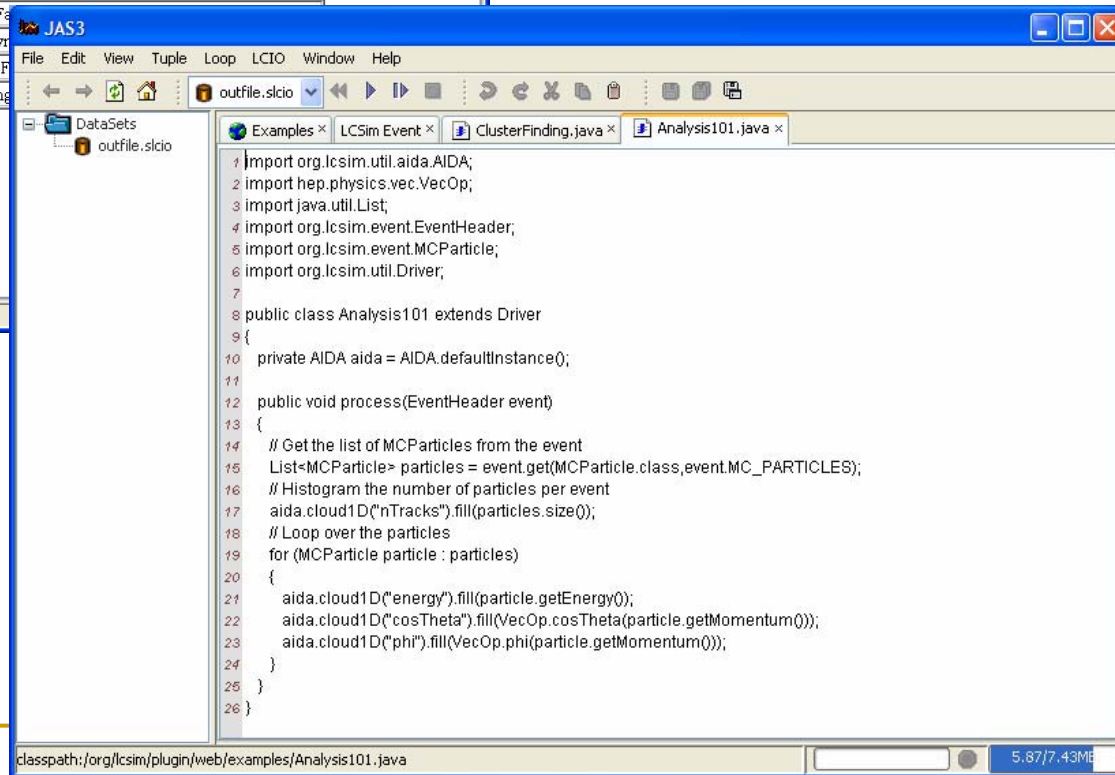
outfile.scio

DataSets
outfile.scio

org.lcsim examples

These examples are written using the Java language. After opening them you need to compile and load them, and then use feed data to them using the Run menu.

SimpleGenerator.java	Simple diagnostic event generator.
Simple.java	Simply prints the event header of each event analyzed.
Analysis101.java	Intro to analysis with AIDA.
SimpleFastMC.java	Running the Fast Monte Carlo
SimpleOutput.java	Example of writing output
JetFinding.java	Using the Jet Finder
ClusterFinding.java	Cluster Finding



JAS3

File Edit View Tuple Loop LCIO Window Help

outfile.scio

DataSets
outfile.scio

Examples × LCSim Event × ClusterFinding.java × Analysis101.java ×

```
1 import org.lcsim.util.aida.AIDA;
2 import hep.physics.vec.VecOp;
3 import java.util.List;
4 import org.lcsim.event.EventHeader;
5 import org.lcsim.event.MCParticle;
6 import org.lcsim.util.Driver;
7
8 public class Analysis101 extends Driver
9 {
10     private AIDA aida = AIDA.defaultInstance();
11
12     public void process(EventHeader event)
13     {
14         // Get the list of MCParticles from the event
15         List<MCParticle> particles = event.get(MCParticle.class,event.MC_PARTICLES);
16         // Histogram the number of particles per event
17         aida.cloud1D("nTracks").fill(particles.size());
18         // Loop over the particles
19         for (MCParticle particle : particles)
20         {
21             aida.cloud1D("energy").fill(particle.getEnergy());
22             aida.cloud1D("cosTheta").fill(VecOp.cosTheta(particle.getMomentum()));
23             aida.cloud1D("phi").fill(VecOp.phi(particle.getMomentum()));
24         }
25     }
26 }
```

classpath:/org/lcsim/plugin/web/examples/Analysis101.java

5.67/7.43MB

org.lcsim: Examples

The screenshot shows the JAS3 interface with the following components:

- File Explorer:** Shows a tree view with folders 'DataSets', 'outfile.slcio', 'Programs', and 'aida31133aida'.
- Event Details:**
 - LCIO Event Header:**

Run	0
Event	0
Time Stamp	Fri Mar 11 14:25:13 PST 2005
Detector Name	sdjan03
 - Blocks:**

Name	Type
HcalEndcapHitsNNClusters	org.lcsim.recon.cluster.nn.NearestNeighborCluster
HcalBarrHitsNNClusters	org.lcsim.recon.cluster.nn.NearestNeighborCluster
EcalEndcapHitsNNClusters	org.lcsim.recon.cluster.nn.NearestNeighborCluster
EcalBarrHitsNNClusters	org.lcsim.recon.cluster.nn.NearestNeighborCluster
MuonEndcapHitsNNClusters	org.lcsim.recon.cluster.nn.NearestNeighborCluster
TkrBarrHits	org.lcsim.util.lcio
TkrEndcapHits	org.lcsim.util.lcio
VtxBarrHits	org.lcsim.util.lcio
LumEndcapHitsNNClusters	org.lcsim.recon
EcalBarrHits	org.lcsim.util.lcio
EcalEndcapHits	org.lcsim.util.lcio
HcalBarrHits	org.lcsim.util.lcio
HcalEndcapHits	org.lcsim.util.lcio
LumEndcapHits	org.lcsim.util.lcio
MuonBarrHits	org.lcsim.util.lcio
MuonEndcapHits	org.lcsim.util.lcio
MCParticle	org.lcsim.event

Analized 1 records in 406ms

The screenshot shows the JAS3 interface with a table of EcalBarrHits data. The table has the following columns: layer, system, barrel, theta, phi, energy, x, y, z.

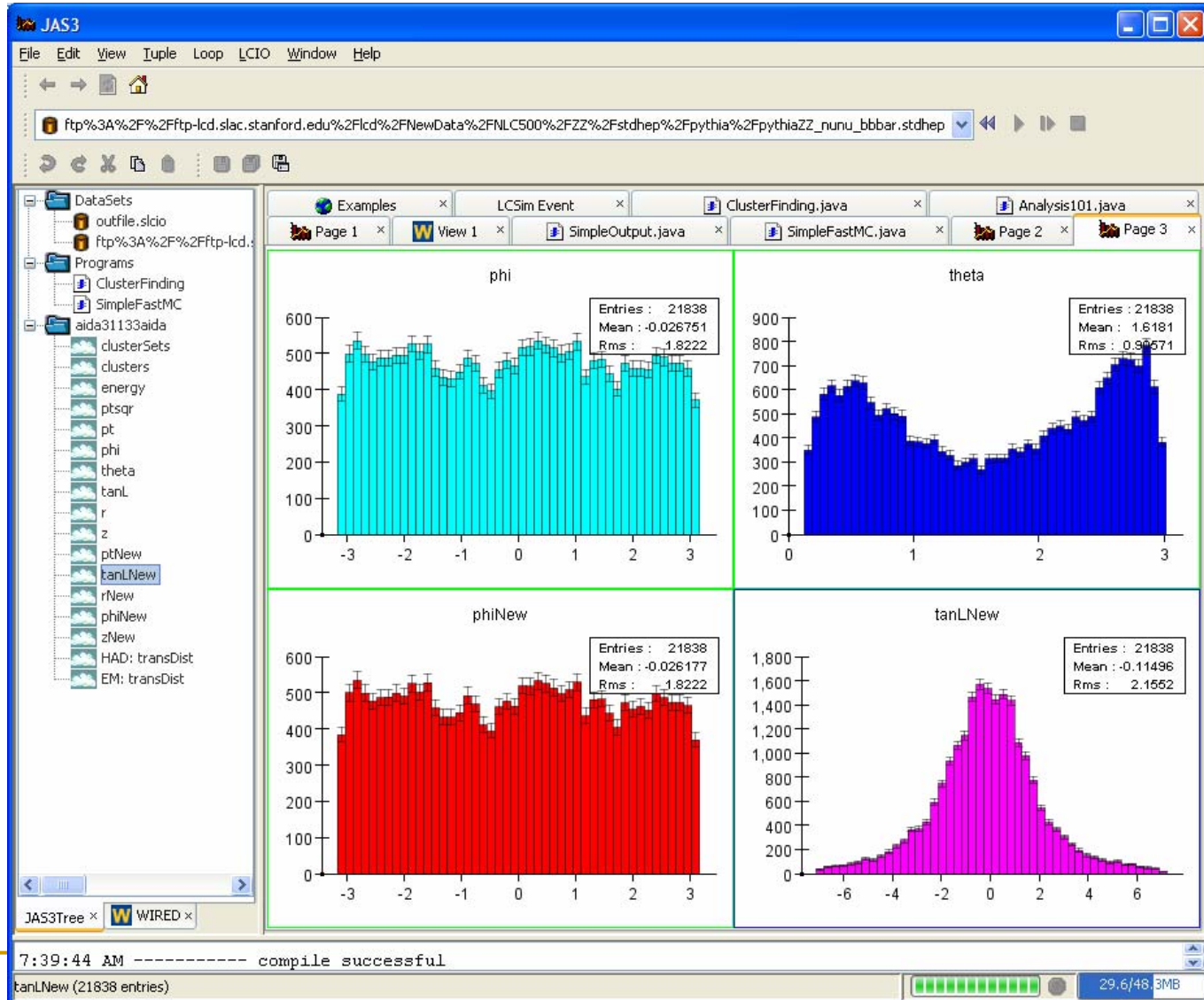
Collection: EcalBarrHits size:424 flags:a0000000

layer	system	barrel	theta	phi	energy	x	y	z
0	2	0	333	1595	4.0386E-4	1210.1	-395.70	426.89
1	2	0	333	1594	1.1317E-4	1213.4	-401.80	428.57
9	2	0	341	1593	6.0089E-5	1249.8	-419.05	398.53
1	2	0	333	1595	.0025117	1214.9	-397.26	428.57
2	2	0	333	1595	3.3759E-4	1219.7	-398.81	430.24
0	2	0	416	881	1.1273E-4	-1257.9	-196.82	16.667
1	2	0	416	880	3.5485E-4	-1263.6	-192.87	16.733
2	2	0	416	880	1.1914E-4	-1268.5	-193.62	16.798
3	2	0	416	880	1.0678E-4	-1273.5	-194.38	16.863
4	2	0	416	880	1.3202E-4	-1278.4	-195.13	16.929
5	2	0	416	880	1.0821E-4	-1283.3	-195.89	16.994
6	2	0	416	880	1.4717E-4	-1288.3	-196.64	17.060
7	2	0	416	880	1.1575E-4	-1293.2	-197.40	17.125
8	2	0	416	880	1.2397E-4	-1298.2	-198.15	17.191
9	2	0	416	880	1.3174E-4	-1303.1	-198.90	17.256
10	2	0	416	879	1.1775E-4	-1308.8	-194.77	17.322
11	2	0	416	879	1.3348E-4	-1313.7	-195.50	17.387
12	2	0	416	879	3.6082E-4	-1318.7	-196.24	17.453
13	2	0	416	879	1.1621E-4	-1323.6	-196.97	17.518
14	2	0	416	879	1.0455E-4	-1328.6	-197.71	17.583
15	2	0	416	879	1.0607E-4	-1333.5	-198.45	17.649
16	2	0	416	879	1.2895E-4	-1338.5	-199.18	17.714
17	2	0	416	879	1.2762E-4	-1343.4	-199.92	17.780
18	2	0	416	879	1.0228E-4	-1348.4	-200.65	17.845

Analized 1 records in 406ms

7.22/7.43MB

org.lcsim: Plot Viewing



How hard is it to get started with org.lcsim?

Works on Linux, MacOSX, Windows

- Should take about 15 minutes to install JAS3 and org.lcsim plugin.

Case Study: SLAC Summer student

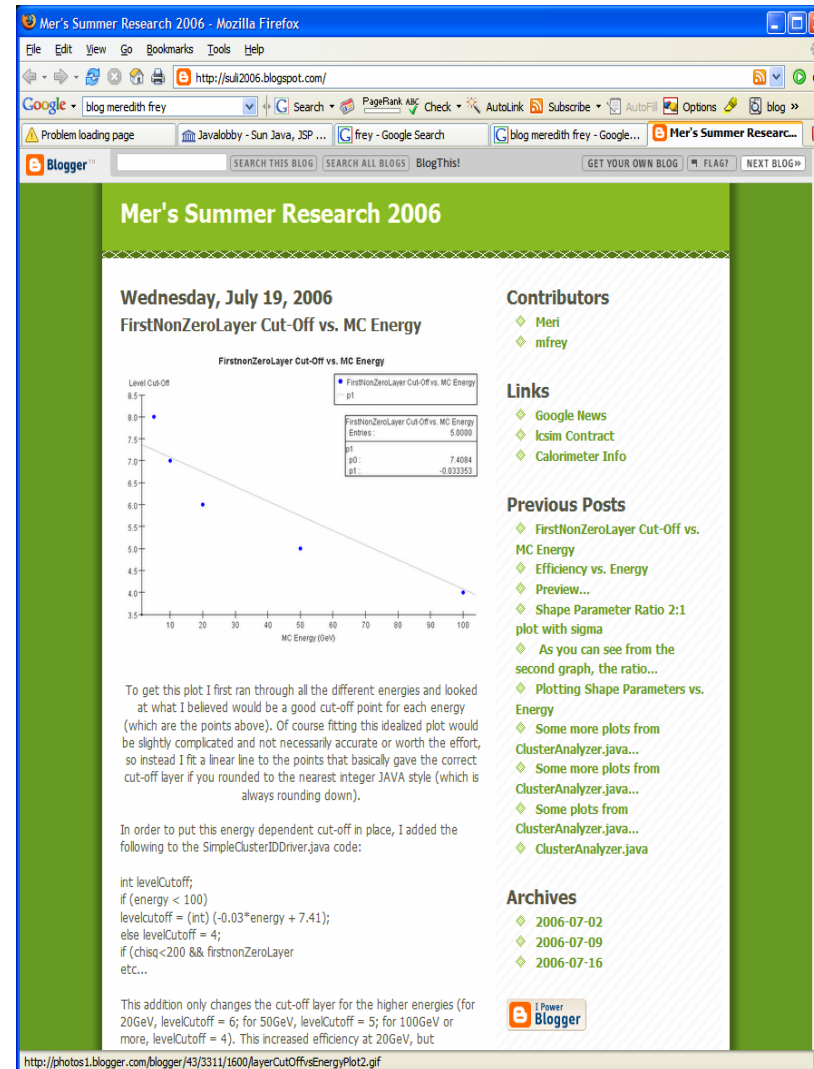
- 2 semesters of Java experience
 - (no C++, Fortran etc)
- Using tutorial on lcsim.org Wiki; installed software, downloaded data, and got useful results in one day (and fixed a few errors in the documentation along the way).
- Regular analysis updates have been appearing on her blog ever since!

Even if you don't have Java experience you can get started almost as fast

- (the only thing you will miss is the core dumps)

Start here:

- <https://confluence.slac.stanford.edu/display/ilc/lcsim+Getting+Started>
- Problems? Attend Tuesday afternoon "Simulation" phone meeting or use discussion forum at <http://forum.linearcollider.org/>



SiD Benchmarking Tools

- MC Data sets (stdhep files) of all SM processes at $E_{cm}=500$ GeV assuming nominal ILC machine parameters
 - About 50 fb^{-1} with e- pol= \pm 90% available at
 - ftp://ftp-glast.slac.stanford.edu/glast.u32/simdet_output/simd401xx/whizdata.stdhep (-90% e- pol)
 - ftp://ftp-glast.slac.stanford.edu/glast.u32/simdet_output/simd402xx/whizdata.stdhep (+90% e- pol)
 - 1 ab^{-1} on SLAC mass storage with all initial e+,e- polarization states
- Many Monte Carlos (Pythia, Whizard) for producing additional stdhep files
- Fast MC which takes stdhep files as input and outputs the same kind of reconstructed particle LCIO objects that full event reconstruction software produces (LCIO bindings exist for C++, JAVA, FORTRAN). For old SIMDET users there exists software to transfer reconstructed particle LCIO data to SIMDET common blocks.

Fast MC Detector Simulation (I)

- In the context of SiD benchmarking the Fast Monte Carlo should be considered a *Fast Physics Object Monte Carlo*. It emulates the bottom line performance of the event reconstruction software in producing the electron, muon, charged hadron, photon and neutral hadron physics objects.
- Status of Fast MC used by SiD:
 - Tracker simulation uses parameterized covariance matrices based on tracker geometry and material
 - Electron and muon id given by min energy + overall efficiency
 - Photon and neutral hadron energies & angles smeared using single particle EM & hadronic energy & angle resolutions. Photons and neutral hadrons also have min energy and overall efficiency within detector volume.

Fast MC Detector Simulation (II)

- Fast MC with nominal single particle calorimeter response gives $17\%/\sqrt{E}$ jet energy resolution. This can be tuned to any value by varying the single particle EM & hadronic calorimeter energy resolutions and by replacing charged particle tracker momentum with calorimeter energy a certain fraction of the time.
- Will improve the parameterization of calorimeter response as we learn more from the particle flow algorithm studies.

Perfect PFA : What theory predicts

- Jet energy resolution

$$\sigma^2(E_{\text{jet}}) = \sigma^2(\text{ch.}) + \sigma^2(\gamma) + \sigma^2(h^0) + \sigma^2(\text{conf.})$$

- Excellent tracker :

$$\sigma^2(\text{ch.}) \ll \sigma^2(\gamma) + \sigma^2(h^0) + \sigma^2(\text{conf.})$$

- Perfect PFA : $\sigma^2(\text{conf.}) = 0$

$$\sigma^2(E_{\text{jet}}) = A_{\gamma}^2 E_{\gamma} + A_h^2 E_{h^0} = w_{\gamma} A_{\gamma}^2 E_{\text{jet}} + w_{h^0} A_h^2 E_{\text{jet}}$$

$$\sigma(E_{\gamma,h})/E_{\gamma,h} = A_{\gamma,h} / \sqrt{E_{\gamma,h}}$$

Typically $w_{\gamma} = 25\%$; $w_{h^0} = 13\%$

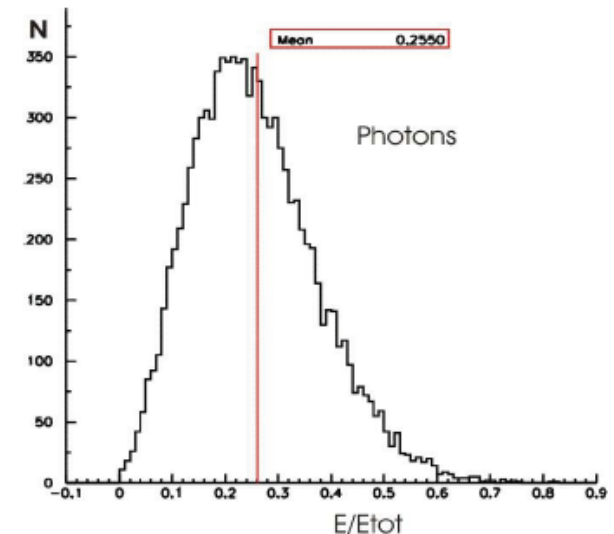
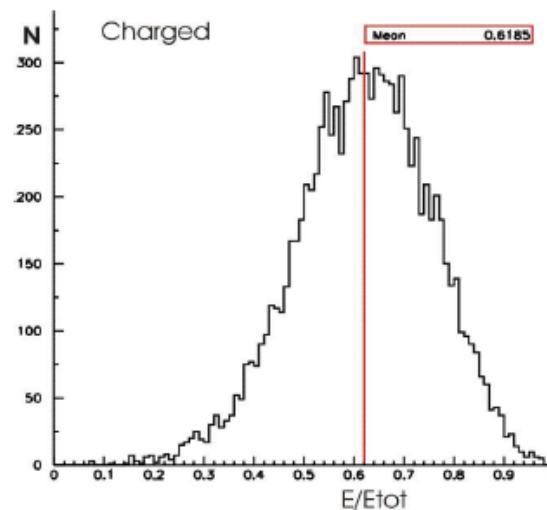
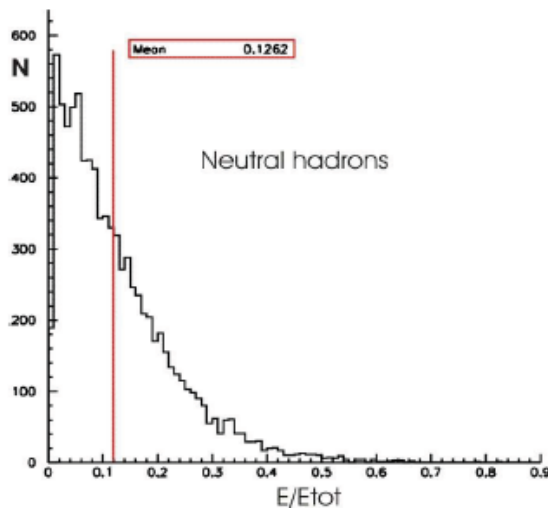
I find $w_{\gamma}=28\%$; $w_{h^0}=10\%$

$A_{\gamma} = 11\%$; $A_{h^0} = 34\%$

$\Rightarrow \sigma(E_{\text{jet}})/E_{\text{jet}} = 12\%/\sqrt{E_{\text{jet}}}$

$A_{\gamma} = 11\%$; $A_{h^0} = 50\%$

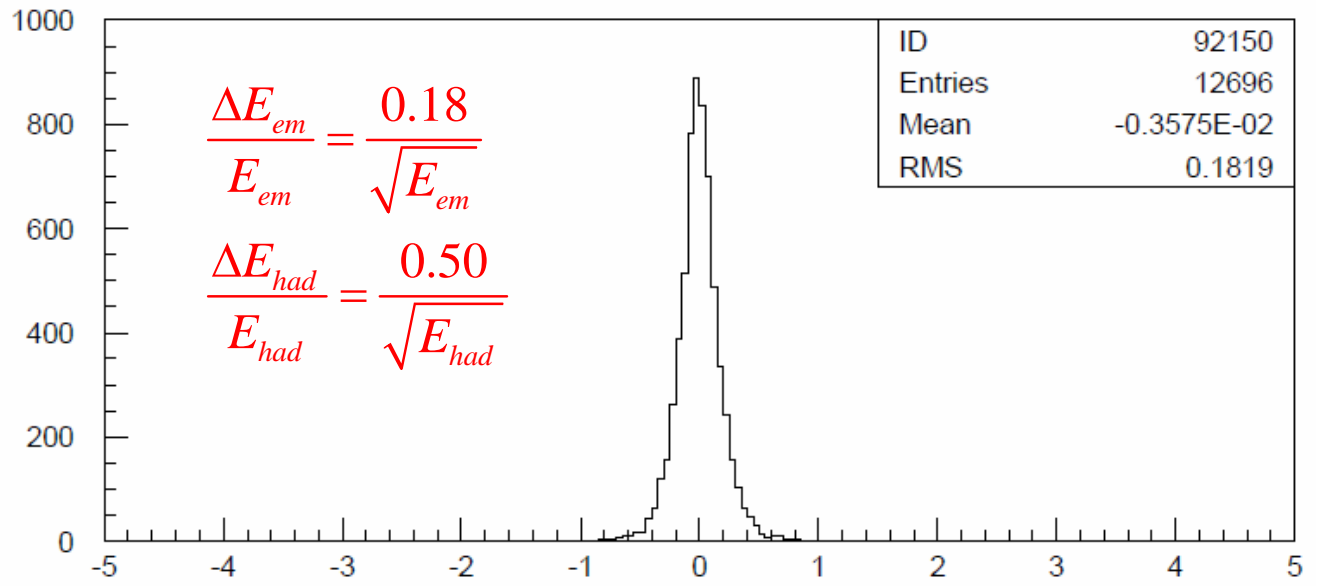
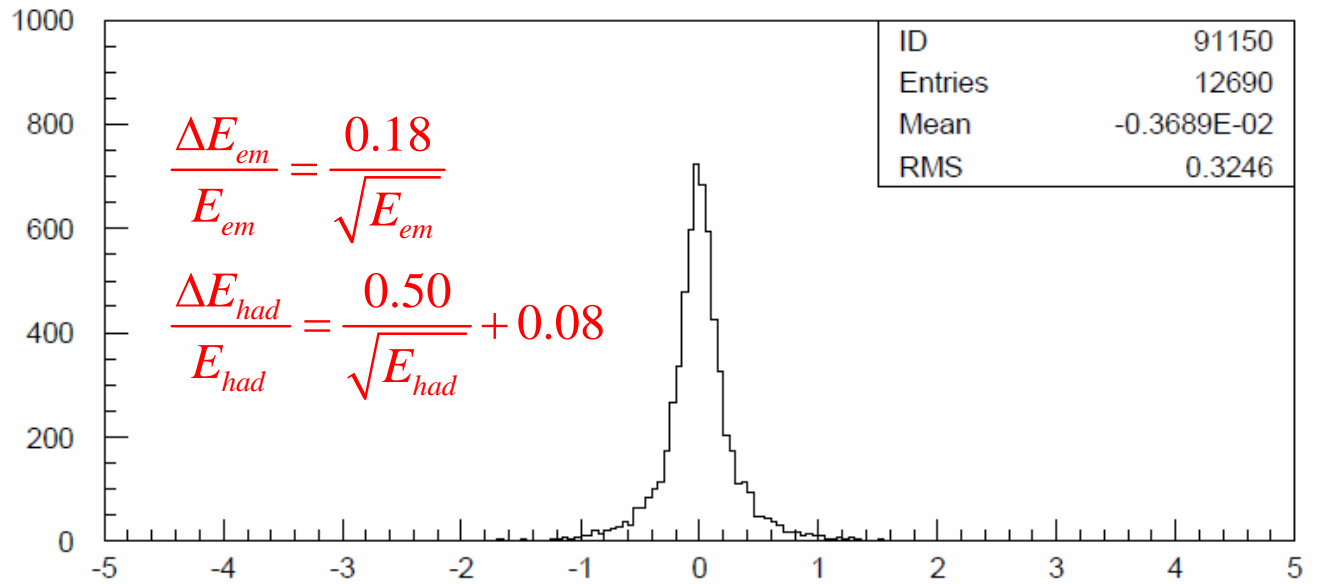
$\Rightarrow \sigma(E_{\text{jet}})/E_{\text{jet}} = 17\%/\sqrt{E_{\text{jet}}}$



$$\sqrt{s} = 500 \text{ GeV}$$

$$e^+e^- \rightarrow u\bar{u}$$

E_{true} is adjusted
for neutrinos and
particles outside
detector acceptance



$$\Delta E_{jet} = (E_{rec} - E_{true}) / \sqrt{E_{true}}$$

Drop constant term in single particle resolution for now. Assume negligible contribution from charged particles to jet energy resolution and write

$$\sigma^2 = (1 + \lambda(1 - r))A_\gamma^2 w_\gamma E_{jet} + (1 + \lambda r)A_h^2 w_h E_{jet} = c^2 E_{jet}$$

where $c = 0.3, 0.4, 0.5, 0.6$

$r =$ hadronic resolution degradation fraction

($r = 1$ to only degrade hadronic resolution

$r = 0$ to only degrade em resolution)

$$A_\gamma = 0.18 \quad A_h = 0.50 \quad w_\gamma = 0.28 \quad w_h = 0.10$$

Given a desired jet energy resolution c the parameter λ is given by

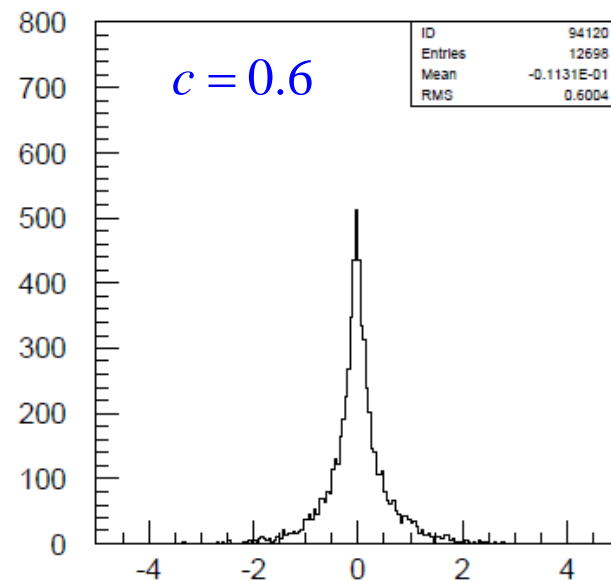
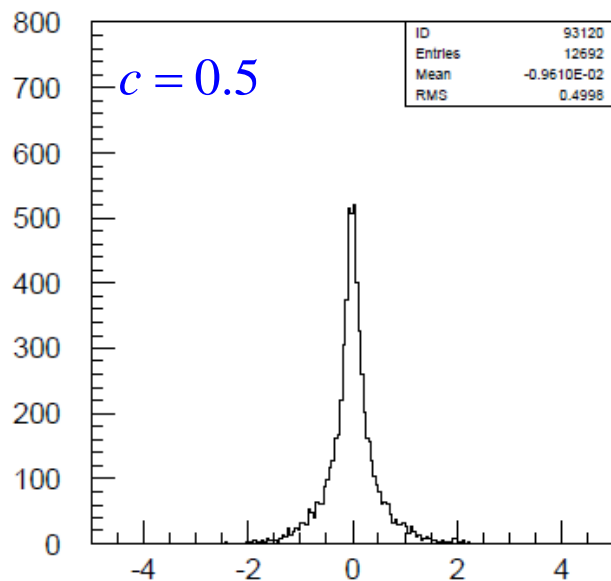
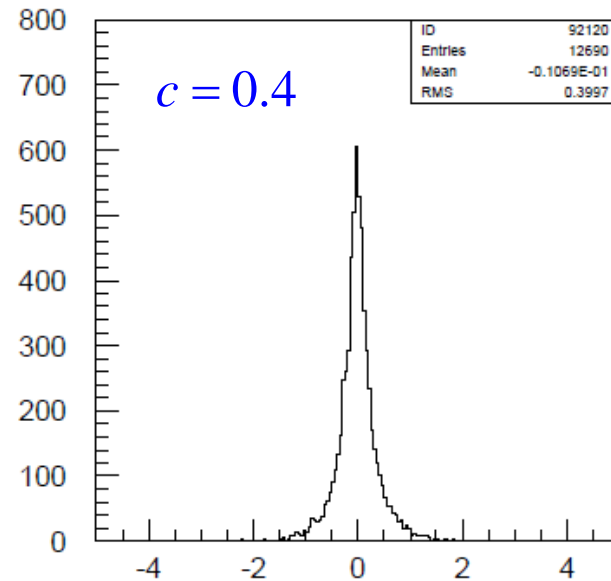
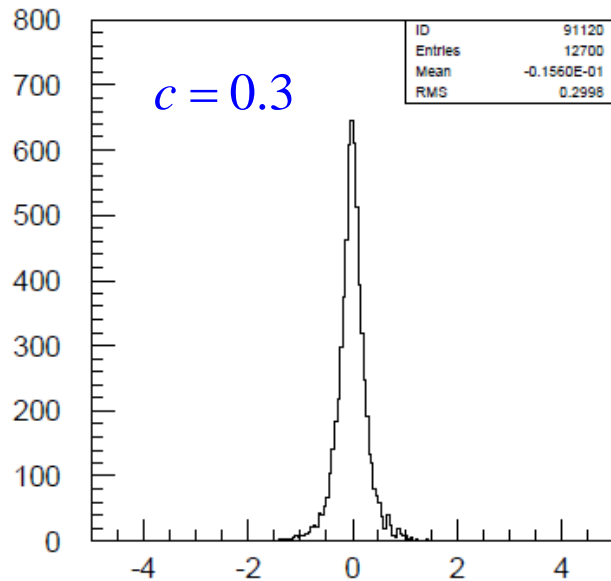
$$\lambda = \frac{c^2 - A_\gamma^2 w_\gamma - A_h^2 w_h}{(1 - r)A_\gamma^2 w_\gamma + rA_h^2 w_h}$$

$$e^+e^- \rightarrow u\bar{u}$$

$$\sqrt{s} = 500 \text{ GeV}$$

$$r = 1.0$$

(only degrade
had resolution)



$$\Delta E_{jet} = (E_{rec} - E_{true}) / \sqrt{E_{true}}$$

$$\Delta E_{jet} = (E_{rec} - E_{true}) / \sqrt{E_{true}}$$

call this the
"non-Gaussian
parameterization"

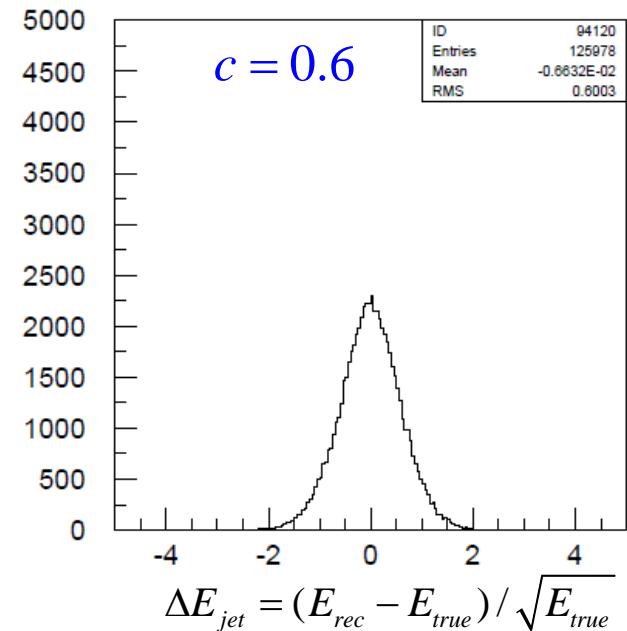
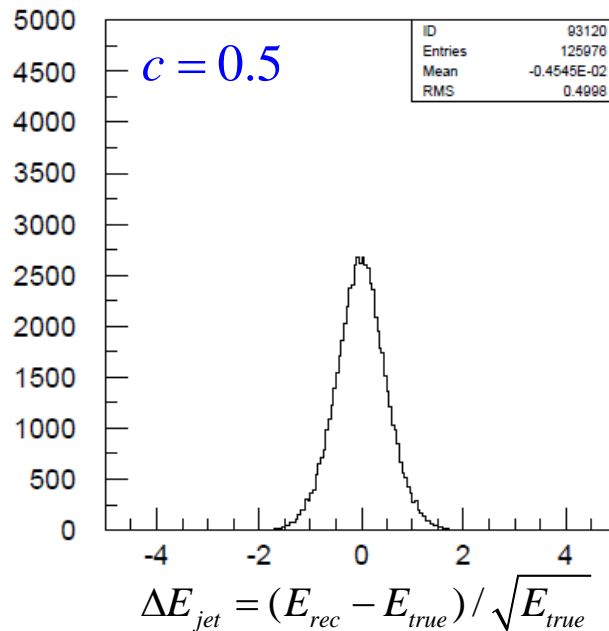
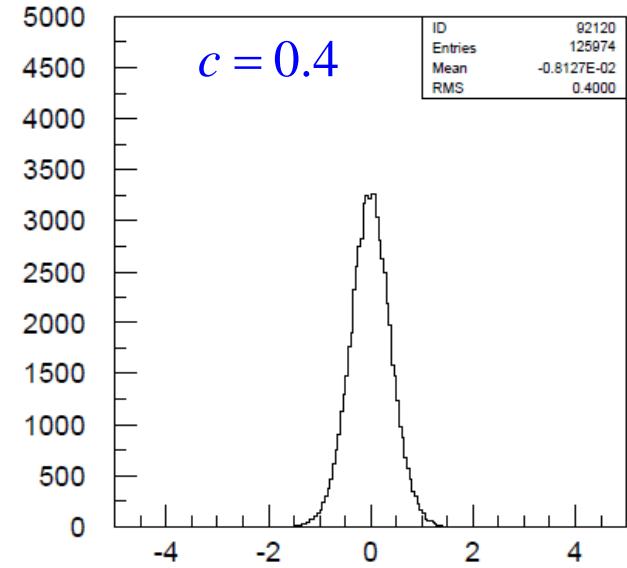
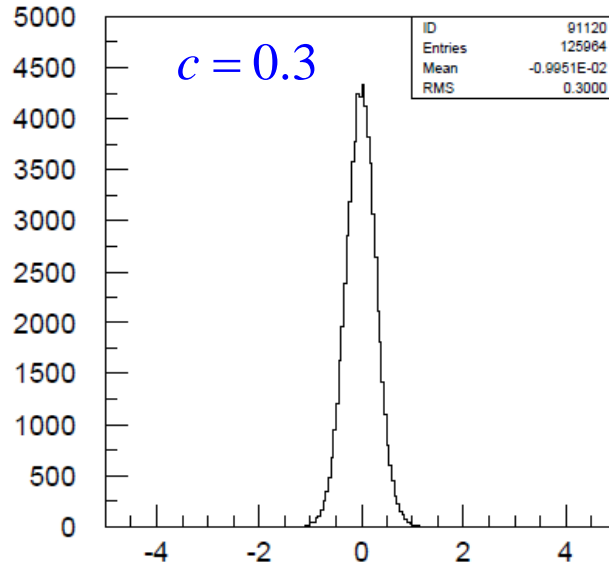
$$e^+e^- \rightarrow u\bar{u}$$

$$\sqrt{s} = 500 \text{ GeV}$$

$$r = 1.0$$

but use calor E
for all chg had

$$\Rightarrow w_h = 0.71$$



call this the
"Gaussian
parameterization"

Monte Carlo Production

- WHIZARD Monte Carlo is used to generate all 0,2,4,6-fermion and t quark dominated 8-fermion processes.
- 1 ab^{-1} @ 0.5 TeV & 2 ab^{-1} @ 1.0 TeV using ILC & NLC params respectively have been generated so far.
- 100% electron and positron polarization is assumed in all event generation. Arbitrary electron, positron polarization is simulated by properly combining data sets.
- Fully fragmented MC data sets are produced. PYTHIA is used for final state QED & QCD parton showering, fragmentation, particle decay.

SM Final States

0-fermion

$$e^+e^- \rightarrow \begin{array}{l} \gamma\gamma \\ \gamma\gamma\gamma \\ \gamma\gamma\gamma\gamma \\ \gamma\gamma\gamma\gamma\gamma \end{array}$$

2-fermion

$$e^+e^- \rightarrow \begin{array}{l} ff \quad f \neq \nu \\ \nu\nu\gamma \\ \nu\nu\gamma\gamma \\ \nu\nu\gamma\gamma\gamma \end{array}$$

$$e^-\gamma \rightarrow e^-\gamma$$

$$\gamma e^+ \rightarrow e^+\gamma$$

4-fermion

$$e^+e^- \rightarrow \begin{array}{l} \nu\nu\nu\nu\gamma \quad 6 \text{ total} \\ u_j\bar{d}_j d_k\bar{u}_k \quad 25 \text{ total} \\ \nu_e e^+ e^- \bar{\nu}_e \\ \nu_e e^+ \mu^- \bar{\nu}_\mu \\ \nu_e e^+ \tau^- \bar{\nu}_\tau \\ \nu_e e^+ d\bar{u} \\ \cdot \\ \cdot \\ c\bar{s}s\bar{c} \\ u_j\bar{u}_j u_k\bar{u}_k \quad 9 \text{ total} \\ u_j\bar{u}_j d_k\bar{d}_k \quad 25 \text{ total} \\ d_j\bar{d}_j d_k\bar{d}_k \quad 21 \text{ total} \\ \gamma\gamma \rightarrow f\bar{f} \quad 8 \text{ total} \\ e_L^- \gamma \rightarrow \nu_e d_k\bar{u}_k \quad 5 \text{ total} \\ e^- \gamma \rightarrow e^- f\bar{f} \quad 10 \text{ total} \\ \gamma e_R^+ \rightarrow \bar{\nu}_e u_k\bar{d}_k \quad 5 \text{ total} \\ \gamma e^+ \rightarrow e^+ f\bar{f} \quad 10 \text{ total} \end{array}$$

6-fermion

$$e^+e^- \rightarrow \begin{array}{l} u_i\bar{u}_i u_j\bar{d}_j d_k\bar{u}_k \quad 125 \text{ total} \\ d_i\bar{d}_i u_j\bar{d}_j d_k\bar{u}_k \quad 150 \text{ total} \\ u_i\bar{u}_i u_j\bar{u}_j u_k\bar{u}_k \quad 25 \text{ total} \\ u_i\bar{u}_i u_j\bar{u}_j d_k\bar{d}_k \quad 65 \text{ total} \\ u_i\bar{u}_i d_j\bar{d}_j d_k\bar{d}_k \quad 75 \text{ total} \\ d_i\bar{d}_i d_j\bar{d}_j d_k\bar{d}_k \quad 56 \text{ total} \end{array}$$

$$\gamma\gamma \rightarrow \begin{array}{l} u_j\bar{d}_j d_k\bar{u}_k \quad 25 \text{ total} \\ u_j\bar{u}_j u_k\bar{u}_k \quad 9 \text{ total} \\ u_j\bar{u}_j d_k\bar{d}_k \quad 25 \text{ total} \\ d_j\bar{d}_j d_k\bar{d}_k \quad 21 \text{ total} \end{array}$$

$$e_L^- \gamma \rightarrow \begin{array}{l} \nu_e u_j\bar{u}_j d_k\bar{u}_k \quad 25 \text{ total} \\ \nu_e d_j\bar{d}_j d_k\bar{u}_k \quad 30 \text{ total} \end{array}$$

$$e^- \gamma \rightarrow \begin{array}{l} e^- u_j\bar{d}_j d_k\bar{u}_k \quad 20 \text{ total} \\ e^- u_j\bar{u}_j u_k\bar{u}_k \quad 10 \text{ total} \\ e^- u_j\bar{u}_j d_k\bar{d}_k \quad 20 \text{ total} \\ e^- d_j\bar{d}_j d_k\bar{d}_k \quad 21 \text{ total} \end{array}$$

$$\gamma e_R^+ \rightarrow \begin{array}{l} \bar{\nu}_e u_j\bar{d}_j u_k\bar{u}_k \quad 25 \text{ total} \\ \bar{\nu}_e u_j\bar{d}_j d_k\bar{d}_k \quad 30 \text{ total} \end{array}$$

$$\gamma e^+ \rightarrow \begin{array}{l} e^+ u_j\bar{d}_j d_k\bar{u}_k \quad 20 \text{ total} \\ e^+ u_j\bar{u}_j u_k\bar{u}_k \quad 10 \text{ total} \\ e^+ u_j\bar{u}_j d_k\bar{d}_k \quad 20 \text{ total} \\ e^+ d_j\bar{d}_j d_k\bar{d}_k \quad 21 \text{ total} \end{array}$$

8-fermion

$$e^+e^- \rightarrow f\bar{f}t\bar{t}$$

$$\gamma\gamma \rightarrow t\bar{t}$$

$$e^- \gamma \rightarrow e^- t\bar{t}$$

$$\nu_e b\bar{t}$$

$$\gamma e^+ \rightarrow e^+ t\bar{t}$$

$$\bar{\nu}_e t\bar{b}$$

There are currently 14 MC production groups:

- 0-2-4-fermion
- 6-fermion/ddi-udj-duk
- 6-fermion/eminus-gamma
- 6-fermion/gamma-eplus
- 6-fermion/gamma-gamma
- 6-fermion/uui-udj-duk
- 6-fermion/zzz_1
- 6-fermion/zzz_2
- 8-fermion/
- bench-point-5
- ffh
- ffhh
- tesla_bosons
- tth

The production group directories are located in

`/afs/slac/g/nld/whizard/xxxx`

where `xxxx=0-2-4-fermion` e.g.

(`xxxx` will stand for a production group from here on)

For each Production Group There are 5 Steps Needed to Produce MC Data Sets: (corresponding shell script is shown in italics)

1. Generate Executable*

/afs/slac/g/nld/fa/whizard-1.50/remake_process_class

2. Submit MC Integration Jobs

/afs/slac/g/nld/whizard/ILC/multiple_whiz_ini

3. Repair MC Integration Jobs

/afs/slac/g/nld/whizard/ILC/multiple_whiz_ini_cleanup

4. Submit MC Event Generation Jobs

/afs/slac/g/nld/whizard/ILC/multiple_whiz_run

5. Repair MC Event Generation Jobs

/afs/slac/g/nld/whizard/ILC/multiple_whiz_run_repair

*Event generation at $E_{\text{cm}}=500$ GeV used an older whizard version

/nfs/slac/g/lcd/mc/prj/sw/dist/whizard/v1r4p0/whizard-v1r4p0/remake_process_class

Data Analysis

- Hundreds of stdhep files are produced with 100% initial state polarization. For data analysis a subset of these files have to be combined in the proper proportion according to the processes to be analyzed and the desired initial state polarization.
- Software has been written to read out stdhep files according to user-specified processes and initial state polarization using the SLAC mstore mass storage facility and the SLAC LINUX batch system. The Makefile for an executable that uses this software can be found at
`~timb/grace/six_fermion/a6f/analysis/lcio/Makefile`
- Examples of analysis job output can be found at
`/afs/slac/g/nld/fa/lcio_physics_analysis/`
- When running over all SM processes 78 batch jobs are submitted simultaneously. Template input files for the 78 batch jobs can be found at
`~timb/grace/six_fermion/a6f/analysis/lcio/whizdata_sm_xx.in ,xx=01,02,...,78`
The script that submits the 78 batch jobs is
`/afs/slac/g/nld/whizard/ILC/multiple_lcio_ini`
The 78 process classes are described in
http://www.slac.stanford.edu/~timb/ilc_2ab_mc_data_set/process_classes.pdf

Polarization, \sqrt{s} , and specific processes are defined in whizdata.in:

```
&whizdata_input
path_root = '/afs/slac.stanford.edu/g/nld/whizard/'
data_root = '/nfs/mstore/g/lcddata/'
i_sqrts = 1000
luminosity = 2000.
n_events_max=120000
mbyte_max = 200.
pol_eminus = -1.0
pol_eplus = 1.0
seed = 520027
output_events = F
process =
  "e1,E1    q,q,q,q"
  "e1,E1    l,q,l,q"
  "e1,E1    l,v,l,v,q,q"
  "e1,A     f,l,l,q,q"
  "e1,A     e1,e1,E1,e2,E2"
/
```

where q,l,v,f,x are defined as:

```
q=u,d,s,c,b,U,D,S,C,B
l=e1,e2,e3,E1,E2,E3
v=n1,n2,n3,N1,N2,N3
f=q,l,v
x=f,A
e3=e3,E3
E3=E3
```

Access to 1 ab⁻¹ Standard Model Monte Carlo Data Sets

- Stdhep files are on SLAC MSTORE Mass Storage
- We have tried in the past to put a subset of this data on permanent disk; we now believe that for SiD Benchmarking studies it is better to have users of this data obtain SLAC computing accounts. We will post info on obtaining SLAC accounts on the Benchmarking web site.
- We will endeavor to improve the user-friendliness of the software that reads out the mstore stdhep files according to user-specified processes and initial state polarization

Appendix: WHIZARD Event Generation Details

1. Generate Executable

remake_process_class copies the file **xxxx/whizard.prc** to WHIZARD's conf directory, does 'make prg', and then copies the results of the make to **xxxx/results**.

2. Submit MC Integration Jobs

multiple_whiz_ini loops through the processes in `xxxx/results/whizard.prc` and submits 4 batch jobs for each process (1 job for each initial state e^+e^- helicity combination).

For each job a directory `/afs/slac/g/nld/fa/mmmm/whizyyyyy` is created where `mmmm` is the center-of-mass energy in GeV and `yyyyy` is a unique 5-digit job number.

multiple_whiz_ini uses the file `xxxx/results/multiple_cardswhiz_in` to build the batch job's `whizard.in` file

multiple_whiz_ini uses the file `/afs/slac/g/nld/whizard/ILC/iniwhiz` to build the batch job's executable script.

3. Repair MC Integration Jobs

multiple_whiz_ini_cleanup loops through the job output in the directories `/afs/slac/g/nld/fa/mmmm/whizttttt` through `/afs/slac/g/nld/fa/mmmm/whizyyyyy` and verifies that the integration was completed successfully. Here `mmm`, `tttt`, `yyyy` are input arguments to the script.

If the integration failed then *multiple_whiz_ini_cleanup* resubmits the job. WHIZARD saves intermediate integration results, so the new job essentially picks up where the old one left off.

4. Submit MC Event Generation Jobs

multiple_whiz_run loops through the MC integration job output directories `/afs/slac/g/nld/fa/mmmm/whizttttt` through `/afs/slac/g/nld/fa/mmmm/whizyyyyy` and submits a run job for every MC integration job which had a cross-section above some minimum value.

For each run job a directory `/afs/slac/g/nld/fa/mmmm/run_output/wkkkkk/run_01` is created where `mmmm` is the center-of-mass energy in GeV and `kkkkk` is the 5-digit MC integration job number.

multiple_whiz_run copies most of the files in the directory `/afs/slac/g/nld/fa/mmmm/whizkkkkk` into the directory `/afs/slac/g/nld/fa/mmmm/run_output/wkkkkk/run_01`.

Parameters specific to event generation are added to the `whizard.in` file before it is copied to `/afs/slac/g/nld/fa/mmmm/run_output/wkkkkk/run_01`.

multiple_whiz_run uses the file `/afs/slac/g/nld/whizard/ILC/runwhiz` to build the batch job's executable script.

5. Repair Event Generation Jobs

multiple_whiz_run_repair loops through the MC run job output directories `/afs/slac/g/nld/fa/mmmm/run_output/wttttt/run_01` through `/afs/slac/g/nld/fa/mmmm/run_output/wyyyyy/run_01` and verifies that the jobs completed successfully. If the job failed *multiple_whiz_run_repair* resubmits the job. If the job completed successfully but additional runs are required it

will submit new run jobs after creating directories of the form

`nfs/slac/g/lcd/mc/mmmm/run_output/wkkkkk/run_02`

`nfs/slac/g/lcd/mc/mmmm/run_output/wkkkkk/run_03`

•

•

•

`nfs/slac/g/lcd/mc/mmmm/run_output/wkkkkk/run_nn`