pandora:

an object-oriented event generator for linear collider physics

M. E. Peskin
August, 2000

pandora

is an event generator for e^+e^- linear collider physics processes,

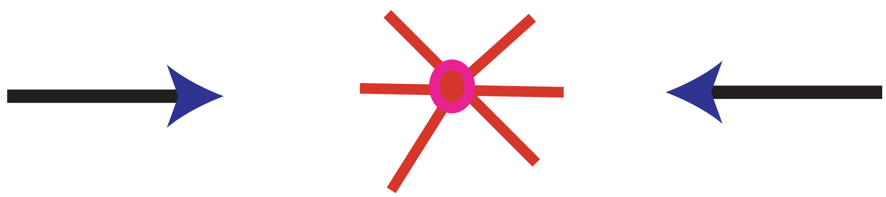
intended to handle:

- beam polarization
- beamstrahlung and ISR
- spin correlations and spin asymmetries
- inclusion of arbitrary hard processes

a general $e^+ e^-$ cross section has the form:

$$\sigma = \int d\vec{x}\, d\vec{y}\, d\vec{z} \cdot \frac{dP(h_1)}{d\vec{x}} \frac{d\sigma(h_1, h_2)}{d\vec{y}} \frac{dP(h_2)}{d\vec{z}}$$

beam process beam



assemble the integrand from beam and process functions

select weight-1 events from the full, correlated distribution

modular design $\rightarrow$ C++

- functionality of pandora
- beam simulation
- event simulation
- process construction
- current status

pandora is a class with constructor

pandora P(beam1, beam2, process)

and methods

P.prepare(Nevents)
P.integral() → returns σ
P.getEvent() → returns a weight-1 event

pandora returns parton-level events in the LEvent data structure, which includes for each parton

- 4-vector
- particle ID
- final?
- color chain
- shower level *

* thanks to K. Fujii

Illustration: $e^+e^- \rightarrow t\,\bar{t} \rightarrow W^+ b\, W^- \bar{b}$

$\rightarrow u\,\bar{d}\; b\; \tau^- \bar{\nu}\; \bar{b}$

parton	ID	parent	final?	chain	sh.level
1	6	0	0	-1	1
2	5	1	1	-1	3
3	24	1	0	0	3
4	-1	3	1	5	4
5	2	3	1	-1	4
6	-6	0	0	1	1
7	-5	6	1	2	2
8	-24	6	0	0	2
9	15	8	1	-11	0
10	-16	8	1	0	0

these partonic events can be hadronized by PYTHIA using an interface

pandora_pythia by Masako Iwasaki

this program

- inserts the pandora generator into PYTHIA as an external subprocess
- requests QCD showers, level by level
- requests hadronization according to the color connection
- decays polarized τ s using TAUOLA
- writes final events to an external file in StdHEP format

```
int main(int argc, char* argv[]){

char* outfile = argv[1];
int nEvent = atoi(argv[2]);

double Eb = 250.0;
double Pol_e  = 0.8;
ebeam b1(Eb,Pol_e,electron,electron);
ebeam b2(Eb,0.0,positron, positron);
b1.setup(NLC500);
b2.setup(NLC500);

eetottbar prtt;
pandora P(b1,b2,prtt);

pandorarun PR(P,epluseminus,ECM,NEvent)
PR.initialize(outfile);
PR.getevents();
PR.terminate();

}
```

Examples of pandora parton-level output:

$e^+e^- \rightarrow W^+W^-$ $\sqrt{s}$ = 500 GeV

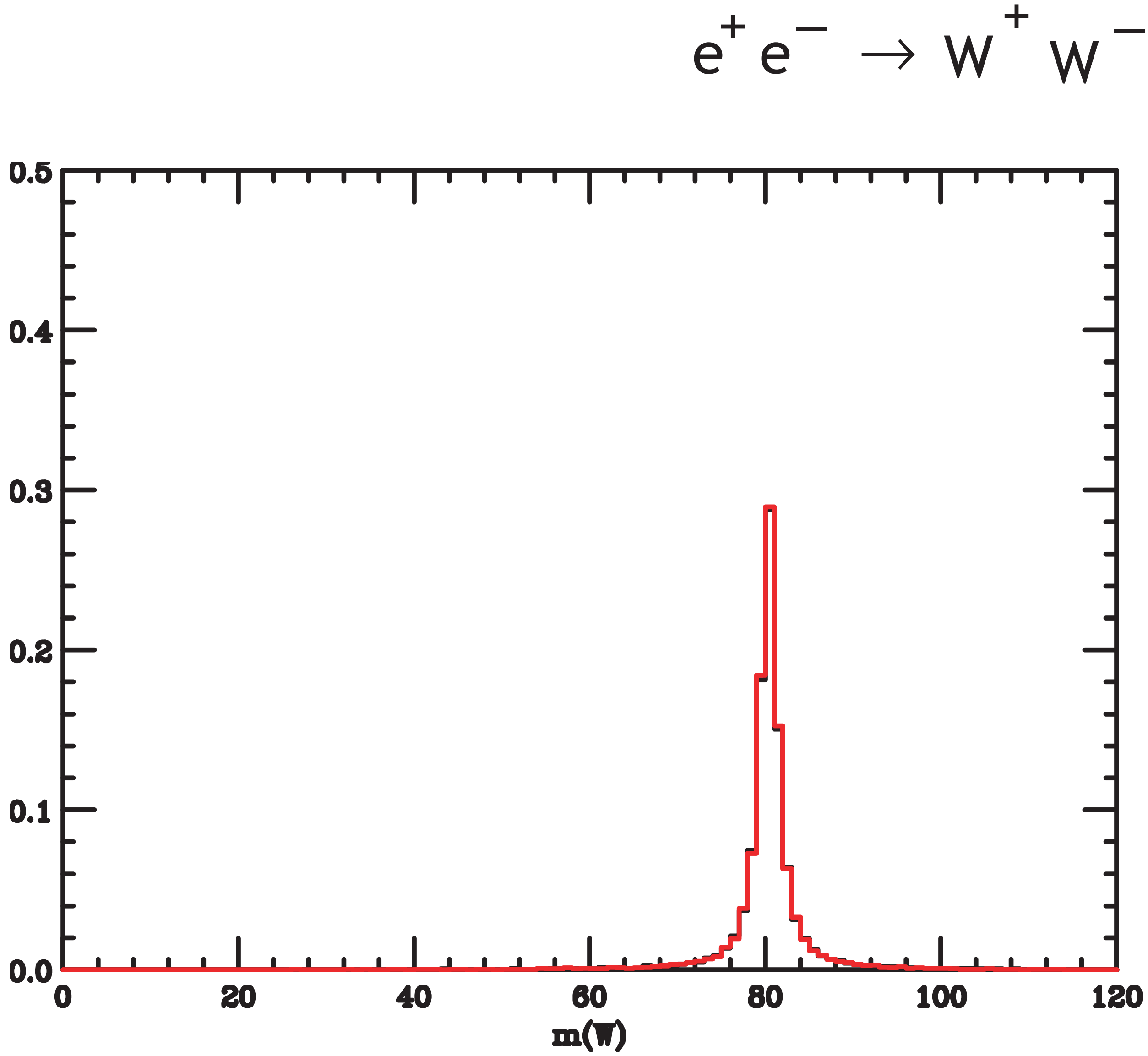
W mass
W, l , ν energies

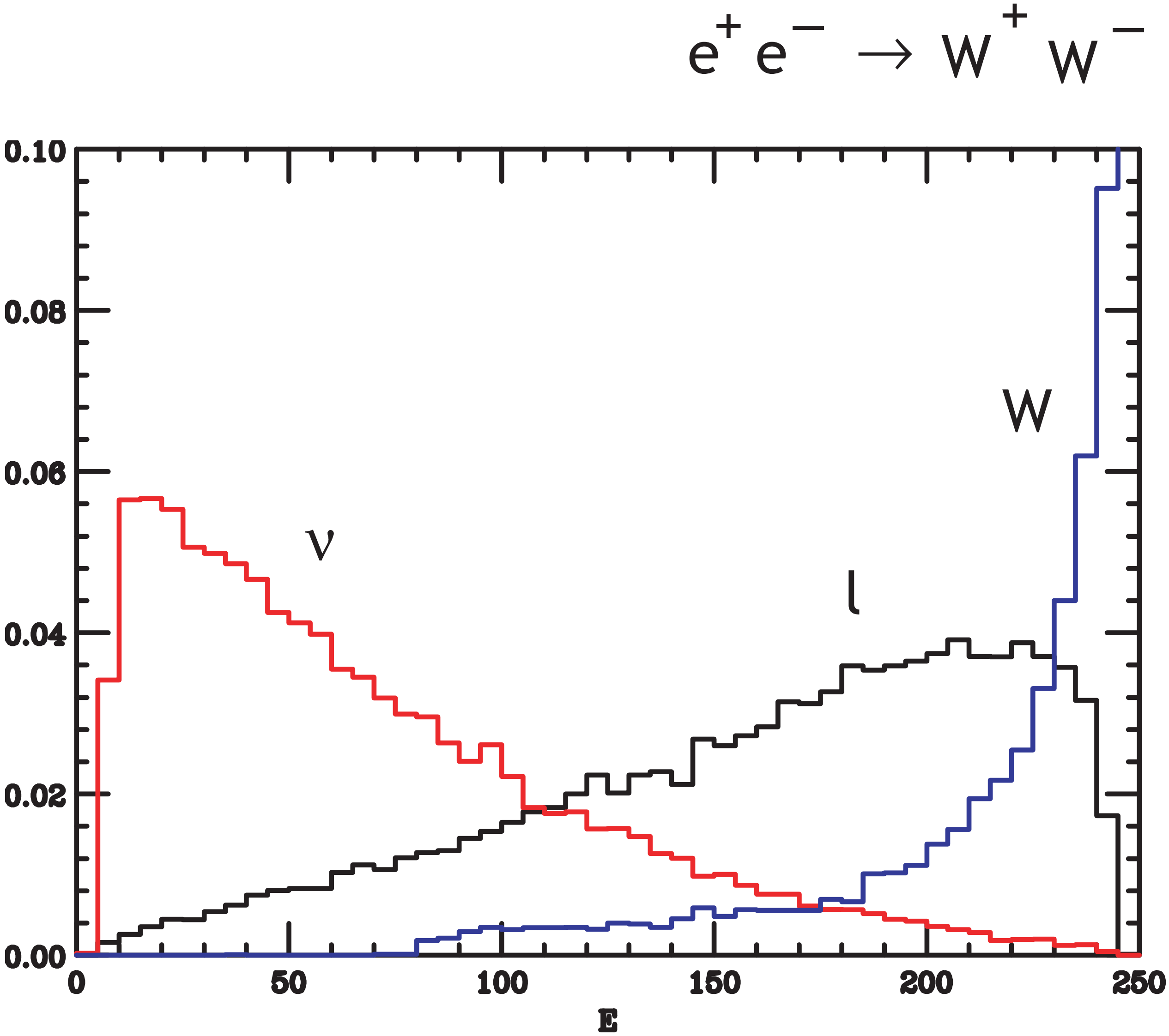
$e^+e^- \rightarrow t\,\bar{t}$

W, t masses
W, b, l, ν energies

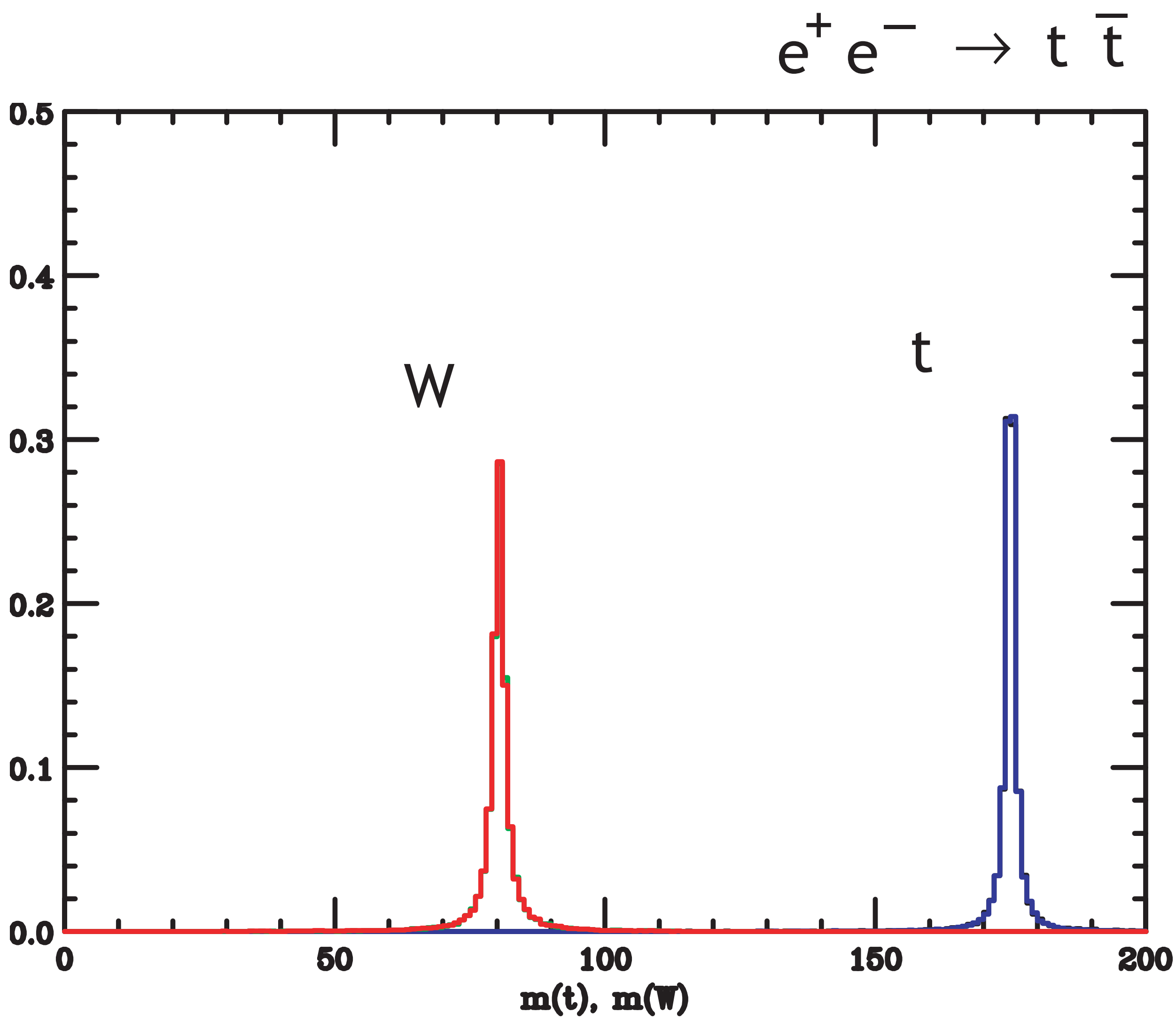
$e^+e^- \rightarrow Z^0\, h^0$

h, Z energies
W, Z masses in h decay to WW*, ZZ*
h BR's

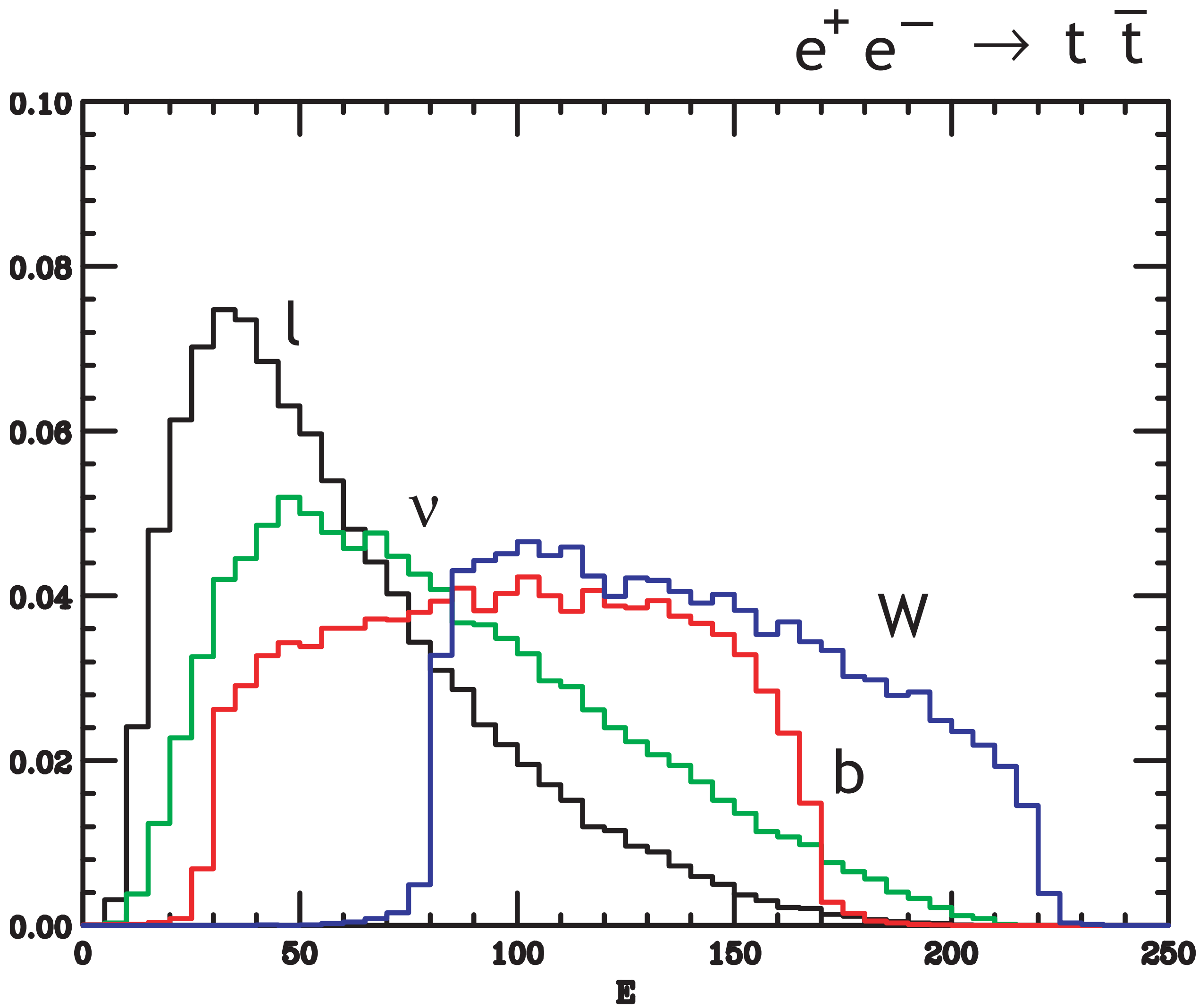

$e^+e^- \to W^+W^-$
0.5
0.4
0.3
0.2
0.1
0.0
0
20
40
60
80
100
120
m(W)



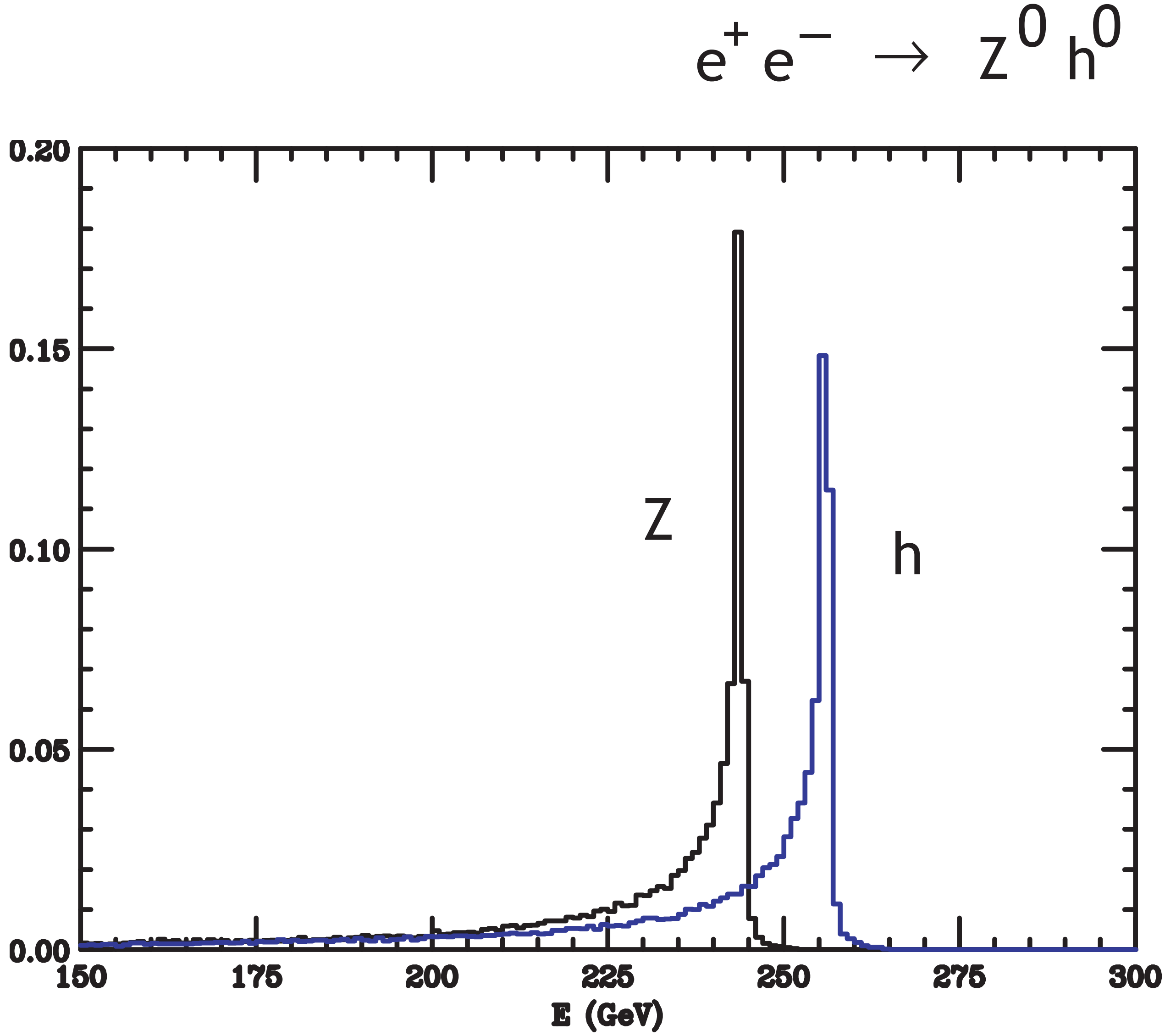
$e^+e^- \rightarrow W^+W^-$
0.10
0.08
0.06
0.04
0.02
0.00
0
50
100
150
200
250
E
ν
l
W



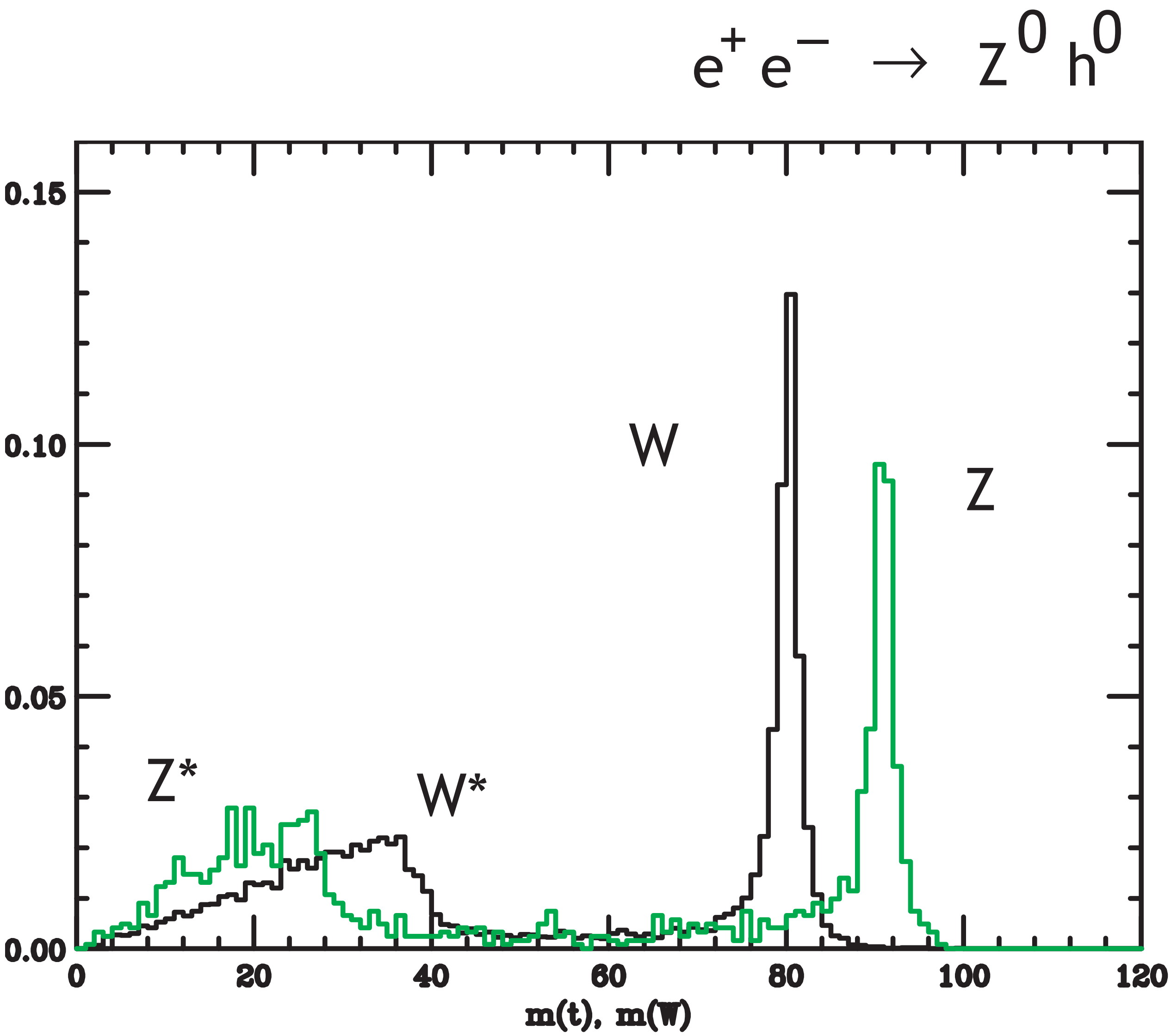
$e^+ e^- \rightarrow t\ \bar{t}$
W
t
0.0
0.1
0.2
0.3
0.4
0.5
0
50
100
150
200
m(t), m(W)



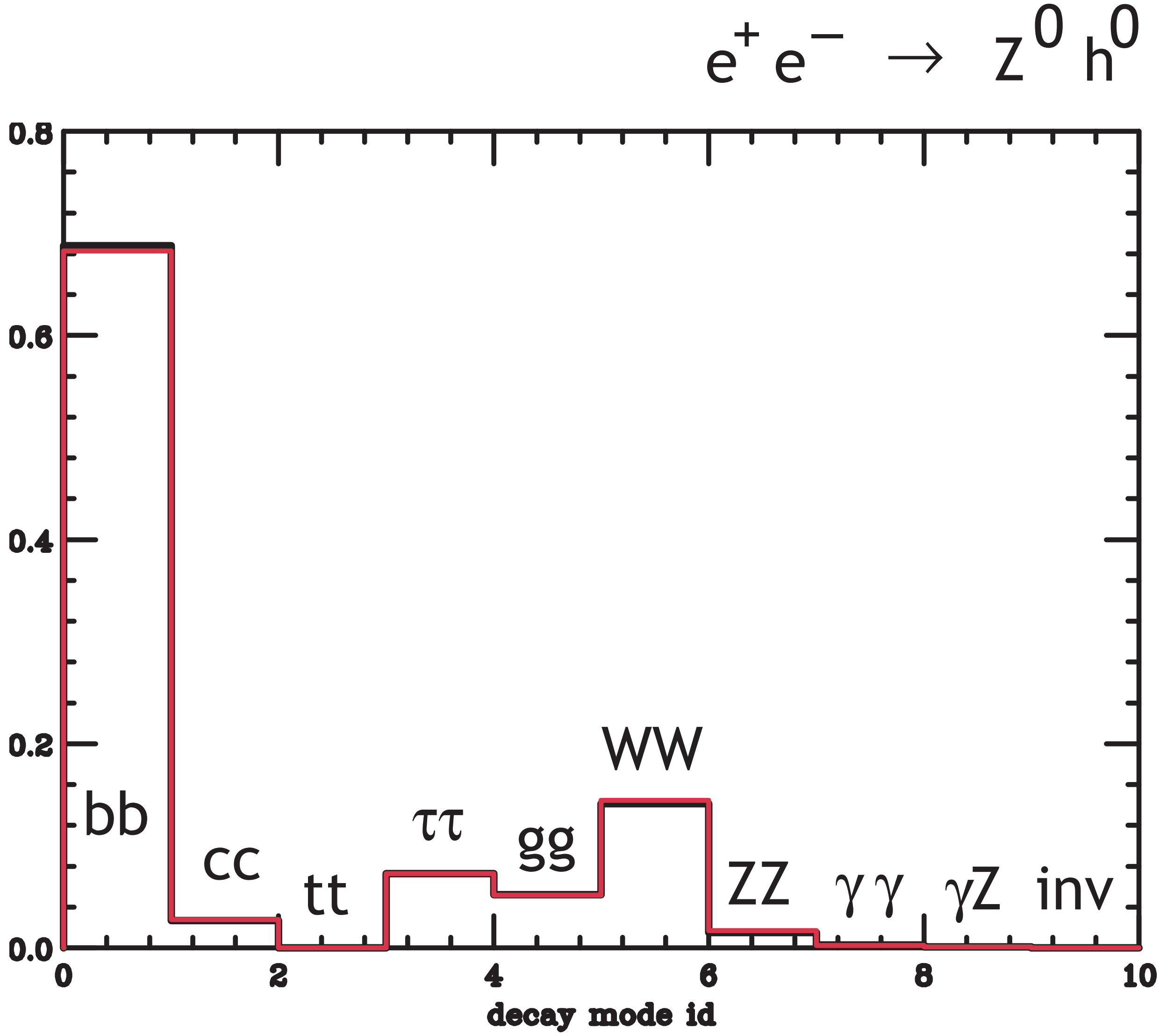
e+ e− → t t̄
0.10
0.08
0.06
0.04
0.02
0.00
0
50
100
150
200
250
E
l
ν
W
b



$e^+ e^- \rightarrow Z^0 h^0$
0.20
0.15
0.10
0.05
0.00
Z
h
150
175
200
225
250
275
300
E (GeV)



$e^+ e^- \rightarrow Z^0 h^0$
0.15
0.10
0.05
0.00
0
20
40
60
80
100
120
m(t), m(W)
W
Z
Z*
W*



$e^+ e^- \rightarrow Z^0 h^0$
0.8
0.6
0.4
0.2
0.0
bb
cc
tt
ττ
gg
WW
ZZ
γ γ
γZ
inv
0
2
4
6
8
10
decay mode id

beam class:

e^+e^- beams have

ISR

using Fadin-Kuraev structure fcns.

beamstrahlung

using `consistent Yokoya-Chen'

to construct a beam, input using setup()

a standard design

NLC/JLC	500	1000	1500
TESLA	500	800	
CLIC	500	1000	3000

or basic machine parameters

N , β_i , σ_i

γ beams have

γ, e spectra from Compton backscattering

ISR for scattered electrons

there is also a beam class for $e^- e^-$.

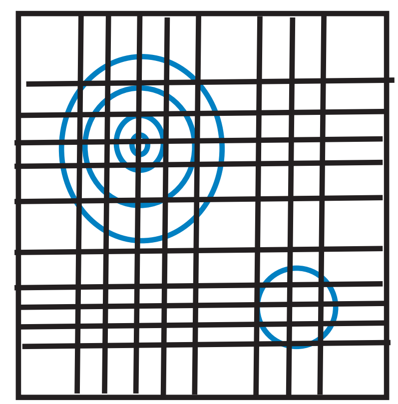
All beam classes take full account of beam polarization.

event selection:

pandora must select weight-1 events with no approximations after $d\sigma / dxdydz$ is given

use the VEGAS algorithm as in BASES/SPRING

model function by a coordinate grid
find maximum weight
keep pts with probability weight/max. weight



standard VEGAS chooses the grid to minimize the variance
if instead, one minimizes the maximum weight, this gives a factor 4 - 10 speedup.

event selection times:

$e^+e^- \to t\,\bar{t} \quad \cdots \quad l^+l^-\,,\quad Z^0 h^0$

2 msec   10 msec

Is there a better algorithm?

pandora is structured as

pandora $\leftarrow$ VegasMC $\leftarrow$ MonteCarlo

but in such a way that pandora uses only method of the virtual class MonteCarlo

thus, any other subclass of MonteCarlo can be freely substituted for VegasMC

Yue Chen is working on an eventual selector based on a fractal model similar to Jadach's FOAM.

```
class MonteCarlo{

int N;   // number of integration vars.
MonteCarlo(int N);

virtual double surface(DVector & X);
      // function integrated

virtual void prepare(int Nevents);
virtual DVector getPoint();
virtual DVector getPoint(double weight);

double integral(double & sd);

int presented, accepted, bad;
void printStatistics();

};
```

the process class

is essentially an interface that implements the operations:

- tell if $\vec{x}$ is in the allowed phase space
- compute the differential cross section
- construct the partonic final state in the CM frame

more specifically, a process must implement four functions ...

```
class process{

int n;   /* number of integration
           variables for the process */
char * name    /* identifying string */
DMatrix cs;
 /* helicity-dependent cross section */

virtual int computeKinematics(
   double & J, DVector & X, double s,
                 double beta);
     /* returns 0 if X is invalid;
         J is the Jacobian of the
           transoformation from X
           to useful variables  */

virtual void crosssection();
virtual LVlist buildVectors();
virtual LEvent buildEvent();

};
```

to save time, allocate all vectors and matrices before starting the repetitive phase of event generator (speedup by a factor 20)

How does one construct a process class?

1. Compute helicity amplitudes for the process.

pandora's conventions:
view process in the event plane
(works for up to 3-body final states)

for a vector boson moving in the +3 direction:

$$\varepsilon_{+1} = \frac{1}{\sqrt{2}}(0, 1, i, 0) \qquad \varepsilon_{-1} = \frac{1}{\sqrt{2}}(0, 1, i, 0)$$

$$\varepsilon_0 = (k/m , 0 , 0, E/m)$$

use 2-component notation for all fermions!

for a massive fermion moving in the +3 direction

$$u_{+1/2} = \sqrt{E-p} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad u_{-1/2} = \sqrt{E+p} \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$v_{+1/2} = \sqrt{E+p} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \qquad v_{-1/2} = \sqrt{E-p} \begin{pmatrix} -1 \\ 0 \end{pmatrix}$$

rotate in the 3-1 plane to other orientations

2. Call standard decay amplitudes for massive and unstable partons

decay classes have been written for

$W^+ \quad W^- \quad Z^0 \quad t \quad \bar{t} \quad h^0$ (SM)

it is crucial to adhere to a common, boost-invariant, convention

```
class Wplusdecay: public decaytotwozz{

Wplusdecay();

CVector Camp;
   /* the helicity-dependent decay
              amplitude   */
void decayamp();  /* fill Camp  */

  /*  inherited from decaytotwozz:
int computeDecayKinematics(double & J,
     DVector & X, int i, double m);
LVlist buildDecayVectors(); */

void placeIDs(LEvent & LE, int i,
                       int parent);

};
```

3. Inherit from classes which compute the reaction kinematics

- these classes include finite width effects with the prescription:

  a Breit-Wigner about M gives masses m_1 m_2
  from these and $\sqrt{s}$, compute p (CM)
  from this, compute

  $$E_{ia} = (p^2 + m_i^2)^{1/2} \qquad E = (p^2 + M^2)^{1/2}$$

  use E_{ia} to compute kinematics,
  E to compute production amplitudes

- these classes cut off kinematic singularities appropriately in their constructors

```
class twototwomzt:  public process {

/* cos theta = tanh((2 X[1]-1) * 10)
   m = sqrt(M*M + M*Gamma *
      * tan((PI-2Gamma/M)(X[2]-1/2))*/

twototwomzt(int N, double M, double G,
   double thetamin, double ptmin,
   double Emin);
twototwomzt(int N, double M, double G);
   /* implements default choices  */

int validEvent(DVector & X, double s,
                    double beta);

int computeKinematics(double & J,
  DVector & X, double s, double beta);

LVlist buildVectors();

};
```

4. Code the full quantum-mechanical amplitudes for production and decay, using the helicity bases.

5. Construct the final state parton momenta using boost and rotations of 4-vector lists provided by the decay classes.

6. Add ID's to the LEvent using functions from the decay classes.

```
LVlist eetottbar::buildVectors(){
    /* t decay products  */
 LVlist Lt = TD.buildDecayVectors();
 Lt.boost(p/E1a);
    /* tbar decay products  */
 LVlist Ltb = TBD.buildDecayVectors();
 Ltb.boost(p/E2a);
 Ltb.reverseinplane();
    /*  finish  */
 LVlist L = merge(Lt,Ltb);
 L.rotateinplane(cost);
 L.rotate(phi);
 return L;
}

LEvent eetottbar::buildEvent(){
 LEvent LE(buildVectors());
 TD.placeIDs(LE, 1, 0);
 TBD.placeIDs(LE, 6, 0);
 LE.connect(6,1);
 LE.connect(7,2);
 LE.addshower(1,6);
 return LE;
}
```

processes included in pandora 2.1

e^+e^- e^-e^- $\gamma\gamma$ beam classes

$$e^+e^- \to l^+l^- \quad q\bar{q} \quad t\bar{t} \quad W^+W^-$$
$$Z^0\gamma \quad Z^0Z^0 \quad Z^0h^0$$

$e^+e^- \to t\bar{t}$ with general form factors

$$\gamma\gamma \to l^+l^- \quad q\bar{q} \quad t\bar{t} \quad h^0 \quad W^+W^-$$

$$e^-\gamma \to e^-\gamma \quad e^-Z^0 \quad \nu W^-$$

the current distributions of

pandora and pandora_pythia

can be found from the link

www.slac.stanford.edu/~mpeskin/
LC/pandora.html