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:

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

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

\[
P.\text{prepare}(\text{Nevents}) \\
P.\text{integral()} \rightarrow \text{returns } \sigma \\
P.\text{getEvent()} \rightarrow \text{returns a weight-1 event}
\]

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: \[ e^+ e^- \rightarrow t \bar{t} \rightarrow W^+ b W^- \bar{b} \rightarrow u \bar{d} b \tau^- \bar{\nu} \bar{b} \]

<table>
<thead>
<tr>
<th>parton ID</th>
<th>parent</th>
<th>final?</th>
<th>chain</th>
<th>sh.level</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>3</td>
<td>24</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>-1</td>
<td>3</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>6</td>
<td>-6</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>-5</td>
<td>6</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>-24</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>15</td>
<td>8</td>
<td>1</td>
<td>-11</td>
</tr>
<tr>
<td>10</td>
<td>-16</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
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 $\tau$ 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:

\[ e^+e^- \rightarrow W^+W^- \quad \sqrt{s} = 500 \text{ GeV} \]
- W mass
- W, l, \nu energies

\[ e^+e^- \rightarrow t\bar{t} \]
- W, t masses
- W, b, l, \nu energies

\[ e^+e^- \rightarrow Z^0h^0 \]
- h, Z energies
- W, Z masses in h decay to WW*, ZZ*
- h BR's
$e^+ e^- \rightarrow W^+ W^-$
$e^+ e^- \rightarrow W^+ W^-$
$e^+ e^- \rightarrow t \bar{t}$
$e^+ e^- \rightarrow Z^0 h^0$
$e^+ e^- \rightarrow z^0 h^0$
$e^+ e^- \rightarrow Z^0 h^0$
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

<table>
<thead>
<tr>
<th></th>
<th>NLC/JLC</th>
<th>TESLA</th>
<th>CLIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>500</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>$\beta_i$</td>
<td>1000</td>
<td>800</td>
<td>1000</td>
</tr>
<tr>
<td>$\sigma_i$</td>
<td>1500</td>
<td>3000</td>
<td>3000</td>
</tr>
</tbody>
</table>

or basic machine parameters

$N, \beta_i, \sigma_i$
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:

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

2 msec \hspace{1cm} 10 msec
Is there a better algorithm?

**pandora** is structured as

\[
\text{pandora} \leftarrow \text{VegasMC} \leftarrow \text{MonteCarlo}
\]

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 transformation 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:

   \[ \varepsilon_{+1} = \frac{1}{\sqrt{2}} (0, 1, i, 0) \quad \varepsilon_{-1} = \frac{1}{\sqrt{2}} (0, 1, i, 0) \]

   \[ \varepsilon_0 = \left( \frac{k}{m}, 0, 0, \frac{E}{m} \right) \]

   use 2-component notation for all fermions!

   for a massive fermion moving in the +3 direction

   \[ u_{+1/2} = \sqrt{E-p} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad u_{-1/2} = \sqrt{E+p} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \]

   \[ v_{+1/2} = \sqrt{E+p} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad v_{-1/2} = \sqrt{E-p} \begin{pmatrix} -1 \\ 0 \end{pmatrix} \]

   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

\[ W^+ W^- Z^0 t \bar{t} h^0 \ (SM) \]

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

  $$E_{ia} = \left( p + m_i \right)^{\frac{1}{2}}$$

  $$E = \left( p^2 + M^2 \right)^{\frac{1}{2}}$$

  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

\[ e^+e^- \quad e^-e^- \quad \gamma \gamma \quad \text{beam classes} \]

\[ e^+e^- \rightarrow l^+l^- \quad q\bar{q} \quad t\bar{t} \quad W^+W^- \quad Z^0\gamma \quad Z^0Z^0 \quad Z^0h^0 \]

\[ e^+e^- \rightarrow t\bar{t} \quad \text{with general form factors} \]

\[ \gamma\gamma \rightarrow l^+l^- \quad q\bar{q} \quad t\bar{t} \quad h^0 \quad W^+W^- \]

\[ e^-\gamma \rightarrow e^-\gamma \quad e^-Z^0 \quad \nu W^- \]
the current distributions of

pandora and pandora_pythia

can be found from the link

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