Pandora

ME Peskin

July 2000

Pandora

is an event generator for e^+e^- linear colliders

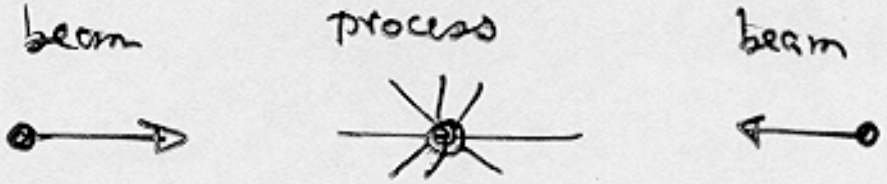
physics processes,

intended to handle:

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

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

$$\sigma = \int dx_1\, dx_2\, dx_3 \; \frac{dP(h_1)}{dx_1} \; \frac{d\sigma(h_1,h_2)}{dx_2} \; \frac{dP(h_3)}{dx_3}$$



assemble the integrand from beam and process functions

select weight-1 events from the full distribution

modular design → C++

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

pandora is a class w. constructor

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
- ID
- parent
- final ?
- color chain ← color linkage
- shower level ← order to carry out QCD showers

```
class LEvent{

  /*  an LEvent contains the following components:

  LV            --   an LVlist giving the 4-vectors of partons
  IInfo         --   an IMatrix giving information on the partons
    ID          =    IInfo[n][1]  --   parton ID
    Parent      =    IInfo[n][2]  --   entry of parent, or 0 if top-level
    Final       =    IInfo[n][3]  --   1 if final, 0 if intermediate state
    colorChain  =    IInfo[n][4]  --   entry of next state along a color-
                                       connected chain, of -1 if the final color
                                            0 if color-singlet
                                  -11 in this place signals a tau-L
                                  -12 in this place signals a tau-R
                                  -13 in this place signals a tau+L
                                  -14 in this place signals a tau+R
    showerLevel =    IInfo[n][5]  --   order in which to carry out parton
                                     showers; the two partons with 1, 2, etc
                                      are shower partners
                                0 in this place signals a nonshowering color singlet
                                                                        */


 public:

    friend class LVlist;

    LEvent(int N):  LV(N), IInfo(1,N,1,5), highestlevel(0){};
    LEvent(const LEvent & LE);
    LEvent(const LVlist & L);

    int n() const ;    /*  returns number of vectors in the event   */

    void read(int m, int id, int parent, int final,
                     int chain, const LVector & V);
    void readid(int m, int id, int parent, int final, int chain);

    int level() const ;                /* return highestlevel  */

    void raiselevels();  /* increase all parton shower levels by 1,
                                    and increase highestlevel by 1 */
    void addshower(int i, int j);
        /*  add a shower between particles i and j, raising all other
              showers to a higher level    */
```

these partonic events can be hadronized

by PYTHIA

Masako Iwasaki pandora-pythia

- insert the events in PYTHIA as external processes
  - requests QCD shower
  - requests hadronization according to color linkage
  - decays polarized τ's w. 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]);

  /*    define the pandora event selection in this space      */

  double ECM = 500.  ;  // Center of mass energy
  double Pol_e = 0.  ;  // Polarization for electron

  ebeam b1(ECM/2.0, Pol_e, electron, electron);
  ebeam b2(ECM/2.0,   0.0, positron, positron);

  b1.setup(NLC500);
  b2.setup(NLC500);

  eetottbar pr;

  pandora P(b1,b2,pr);


  pandorarun  PR(P, epluseminus, ECM, nEvent);

  /*        end of definition                                  */


  PR.initialize(outfile);

  PR.getevents();

  PR.terminate();

}
```

code for pandora

} calls to PYTHIA, TAUOLA

LEvent fn

$e^+e^- \rightarrow t\bar{t} \rightarrow W^+b \quad W^-\bar{b}$

$\hookrightarrow \bar{d}u \quad \hookrightarrow \tau^-\bar{\nu}_\tau$

```
Parton : 1  ID: 6  Parent: 0  Final?: 0  Chain to: -1  Shower Level: 1
   Vector: <   252.402 96.8278 -136.447        70.3312>
Parton : 2  ID: 5  Parent: 1  Final?: 1  Chain to: -1  Shower Level: 3
   Vector: <   117.633 -4.74508        -96.9145        66.5024>
Parton : 3  ID: 24  Parent: 1  Final?: 0  Chain to: 0  Shower Level: 3
   Vector: <   134.769 101.573 -39.5325        3.82875>
Parton : 4  ID: -1  Parent: 3  Final?: 1  Chain to: 5  Shower Level: 4
   Vector: <   76.7714 63.0539 -16.5043        40.5668>
Parton : 5  ID: 2  Parent: 3  Final?: 1  Chain to: -1  Shower Level: 4
   Vector: <   57.9974 38.519  -23.0282        -36.7381>
Parton : 6  ID: -6  Parent: 0  Final?: 0  Chain to: 1  Shower Level: 1
   Vector: <   247.598 -96.8278        136.447 -70.3312>
Parton : 7  ID: -5  Parent: 6  Final?: 1  Chain to: 2  Shower Level: 2
   Vector: <   51.6563 13.0274 38.5341 31.8399>
Parton : 8  ID: -24  Parent: 6  Final?: 0  Chain to: 0  Shower Level: 2
   Vector: <   195.942 -109.855        97.9129 -102.171>
Parton : 9  ID: 15  Parent: 8  Final?: 1  Chain to: -11  Shower Level: 0
   Vector: <   66.8357 -7.28161        33.6148 -57.3065>
Parton : 10  ID: -16  Parent: 8  Final?: 1  Chain to: 0  Shower Level: 0
   Vector: <   129.106 -102.574        64.2982 -44.8646>
```

Event fn

$e^+e^- \rightarrow t\bar{t} \rightarrow W^+b \; W^-\bar{b}$

$\hookrightarrow \bar{s}c \quad \hookrightarrow d\bar{u}$

```
Parton : 1  ID: 6  Parent: 0  Final?: 0  Chain to: -1  Shower Level: 1
   Vector: <   249.857 -86.9575          -98.2031          121.226>
Parton : 2  ID: 5  Parent: 1  Final?: 1  Chain to: -1  Shower Level: 4
   Vector: <   62.9771 -43.2992          -35.0278          -29.3998>
Parton : 3  ID: 24  Parent: 1  Final?: 0  Chain to: 0  Shower Level: 4
   Vector: <   186.88  -43.6583          -63.1753          150.626>
Parton : 4  ID: -3  Parent: 3  Final?: 1  Chain to: 5  Shower Level: 5
   Vector: <   138.207 -29.05  -81.3364          107.897>
Parton : 5  ID: 4  Parent: 3  Final?: 1  Chain to: -1  Shower Level: 5
   Vector: <   48.6721 -14.6083          18.1611 42.7287>
Parton : 6  ID: -6  Parent: 0  Final?: 0  Chain to: 1  Shower Level: 1
   Vector: <   250.143 86.9575 98.2031 -121.226>
Parton : 7  ID: -5  Parent: 6  Final?: 1  Chain to: 2  Shower Level: 2
   Vector: <   55.7517 -43.0487          17.6867 -30.6959>
Parton : 8  ID: -24  Parent: 6  Final?: 0  Chain to: 0  Shower Level: 2
   Vector: <   194.392 130.006 80.5164 -90.5302>
Parton : 9  ID: 1  Parent: 8  Final?: 1  Chain to: -1  Shower Level: 3
   Vector: <   121.393 105.81  21.1496 -55.6167>
Parton : 10  ID: -2  Parent: 8  Final?: 1  Chain to: 9  Shower Level: 3
   Vector: <   72.9989 24.1966 59.3668 -34.9135>
```

example of pandora parton-level output:

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

W mass

W l ν energy

$e^+e^- \to t\bar{t}$

W, t mass

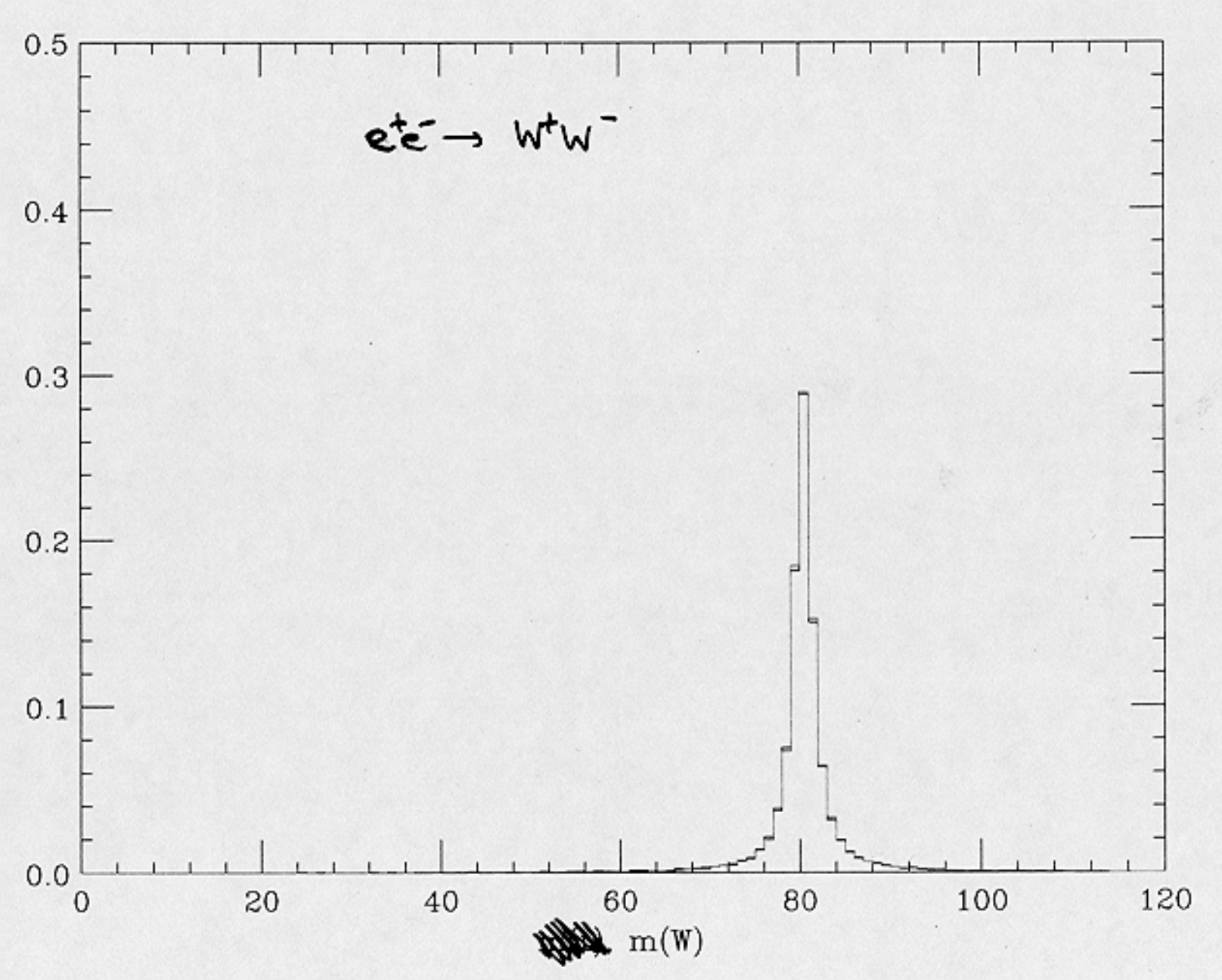
W b lν energies

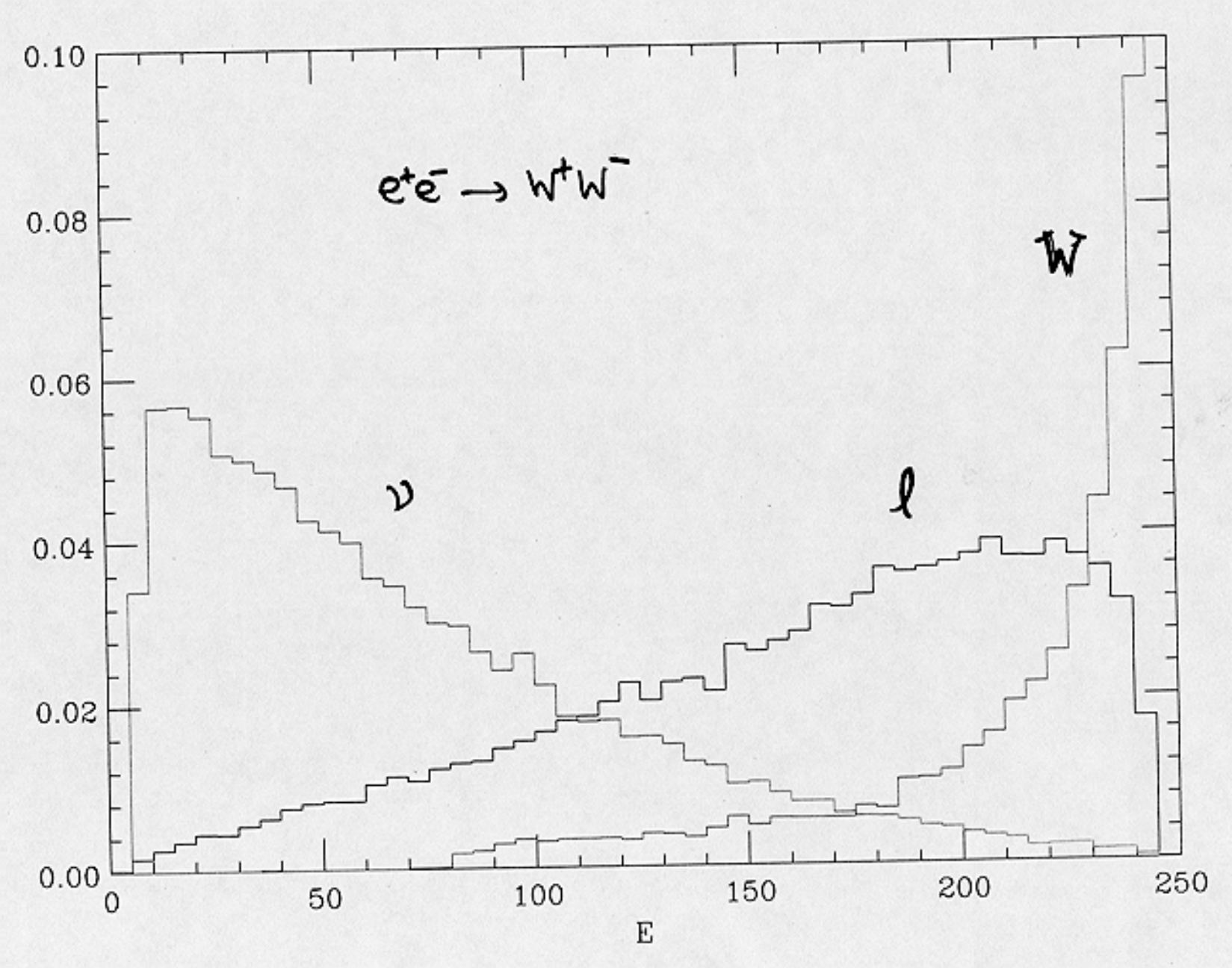
$e^+e^- \to h^0Z^0$

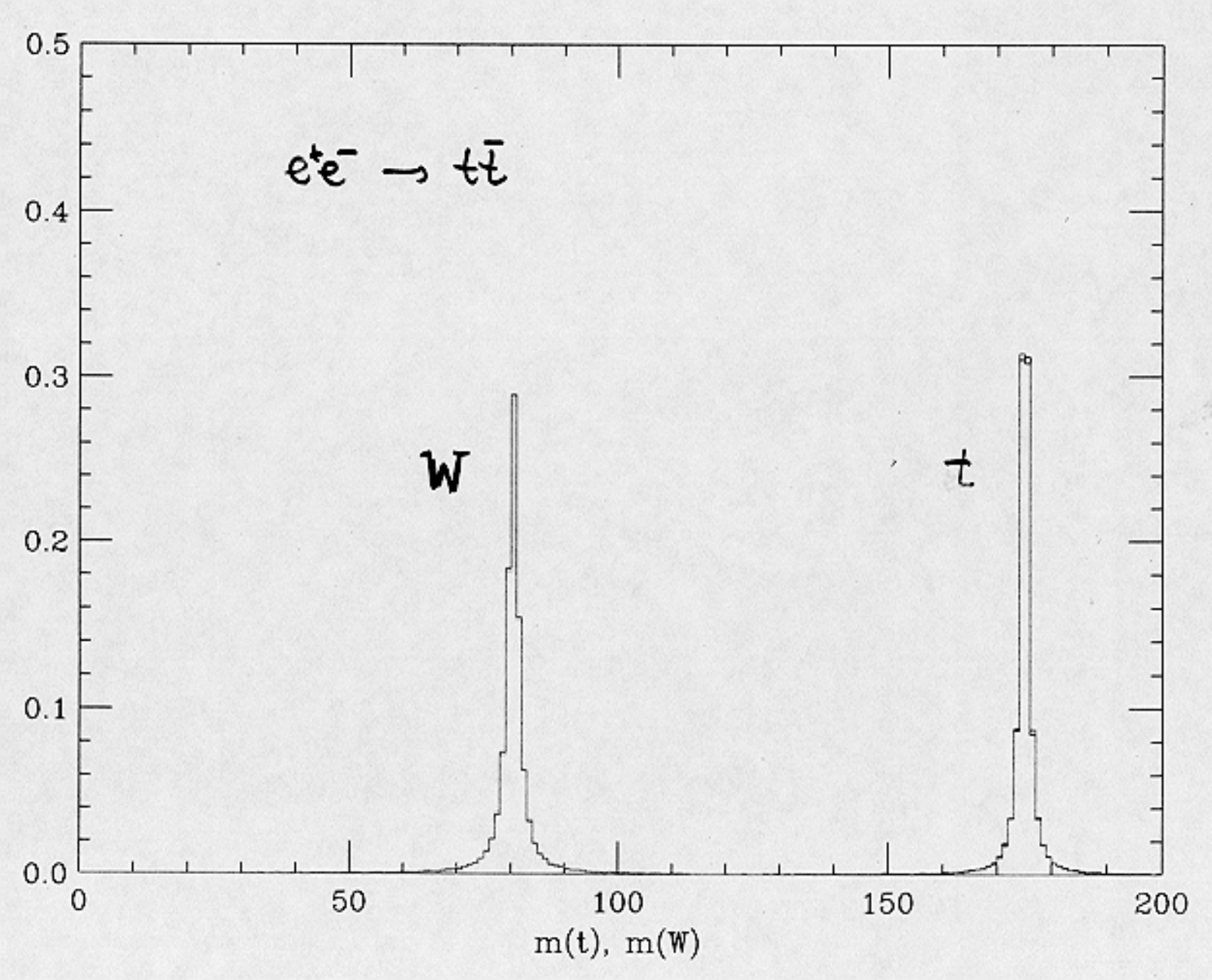
h^0Z^0 energies

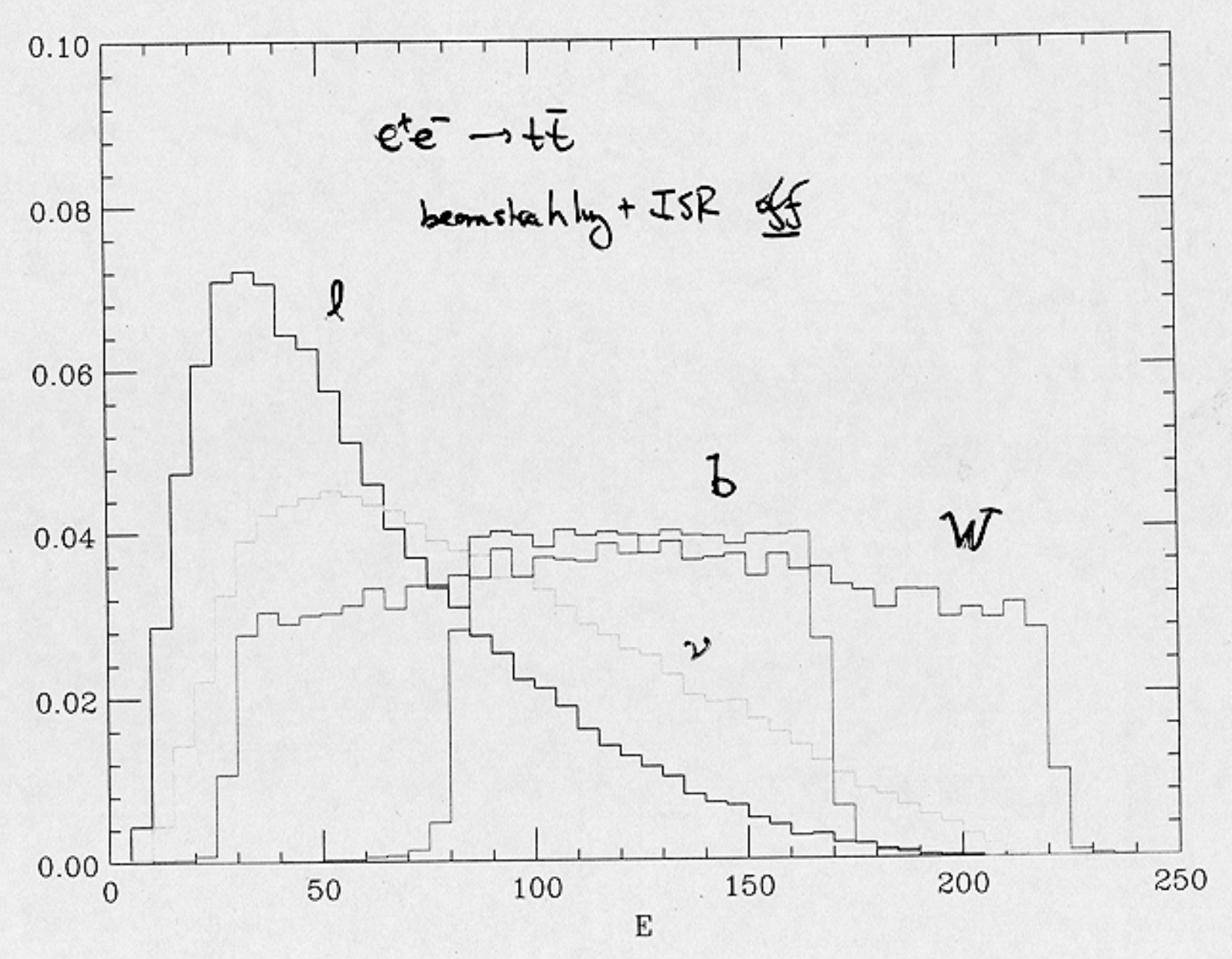
WZ masses in $h^0 \to WW^*, ZZ^*$

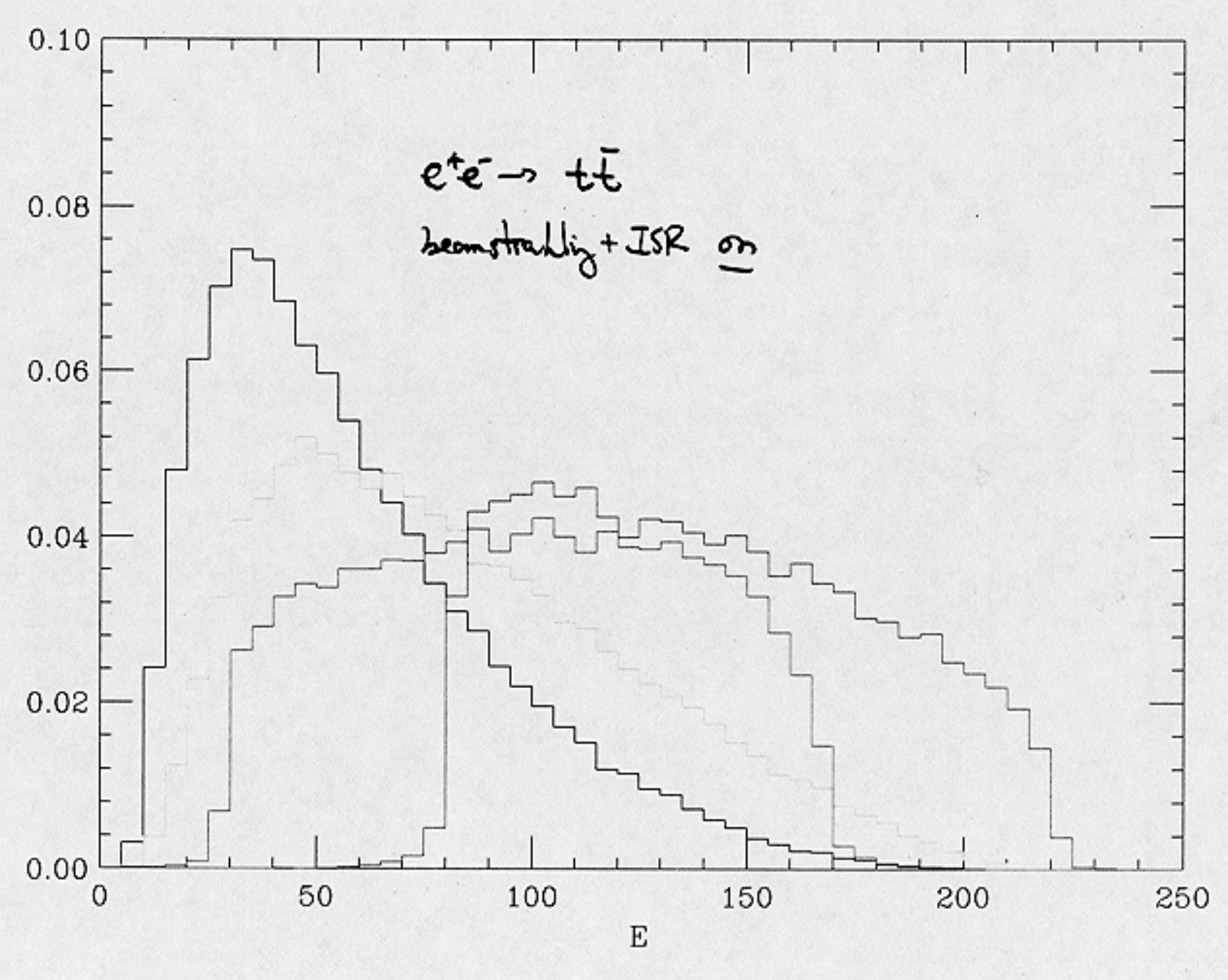
h^0 BR's

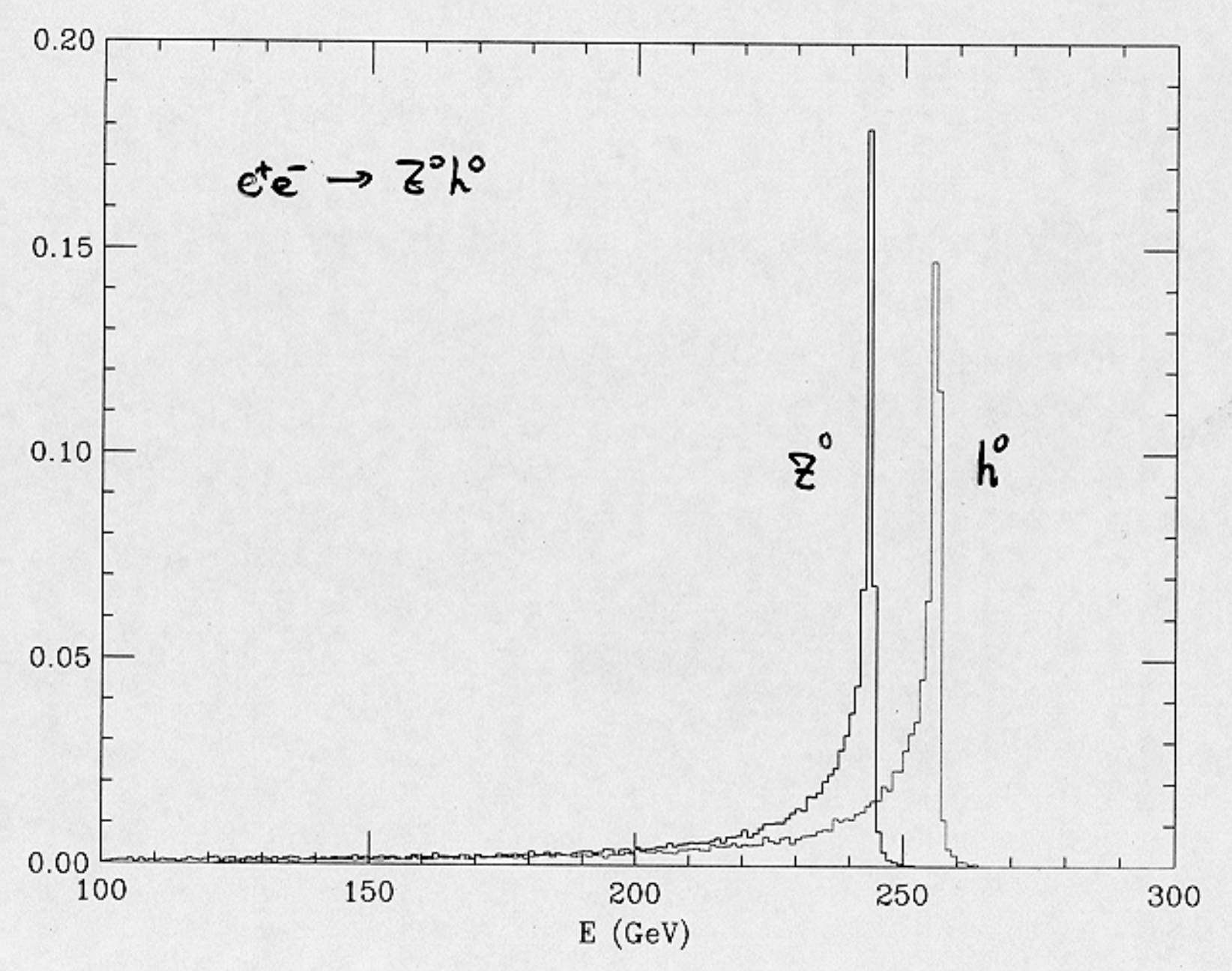

$e^+e^- \rightarrow W^+W^-$
0.5
0.4
0.3
0.2
0.1
0.0
0
20
40
60
80
100
120
m(W)

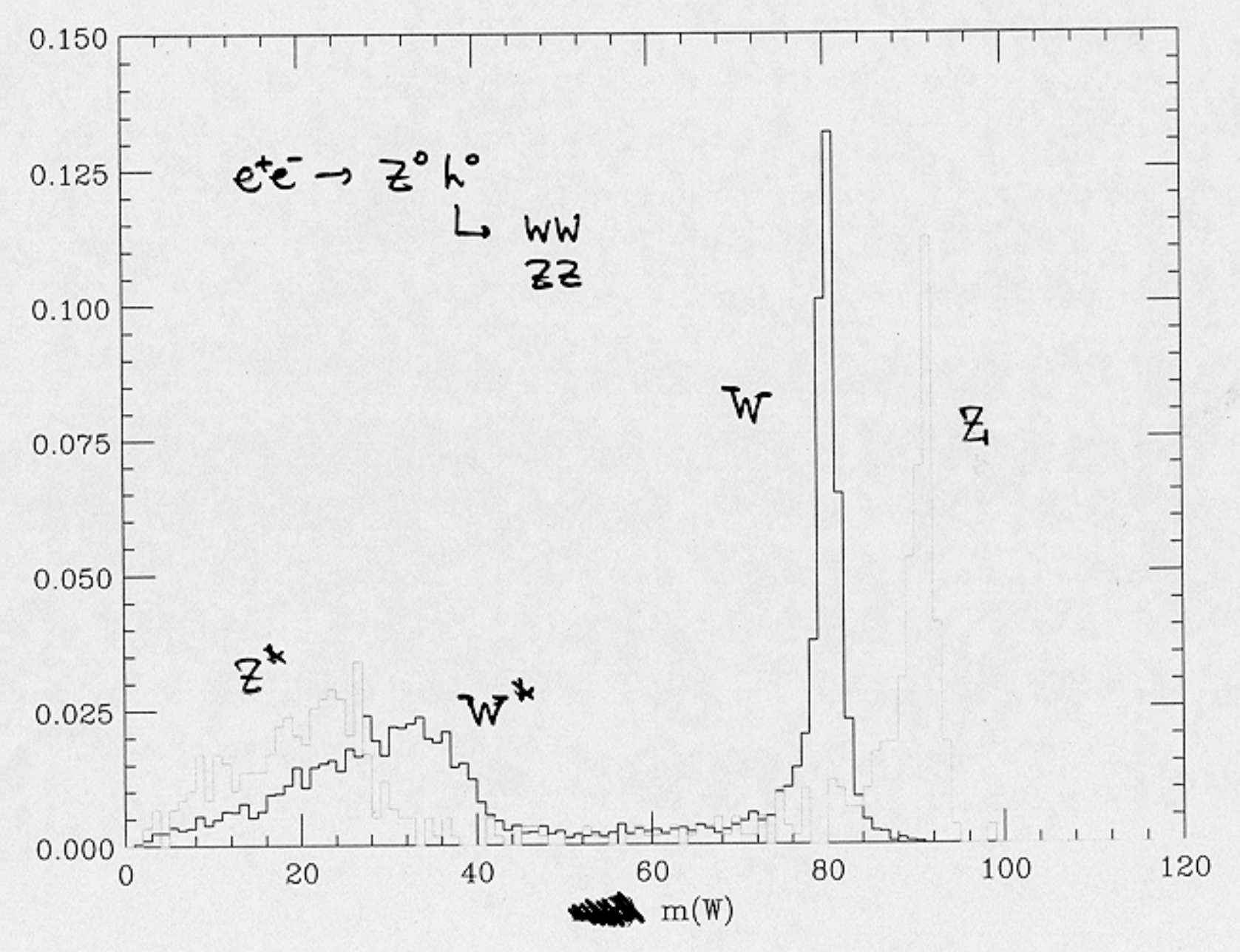

$e^+e^- \to W^+W^-$
W
ν
l
0.10
0.08
0.06
0.04
0.02
0.00
0
50
100
150
200
250
E

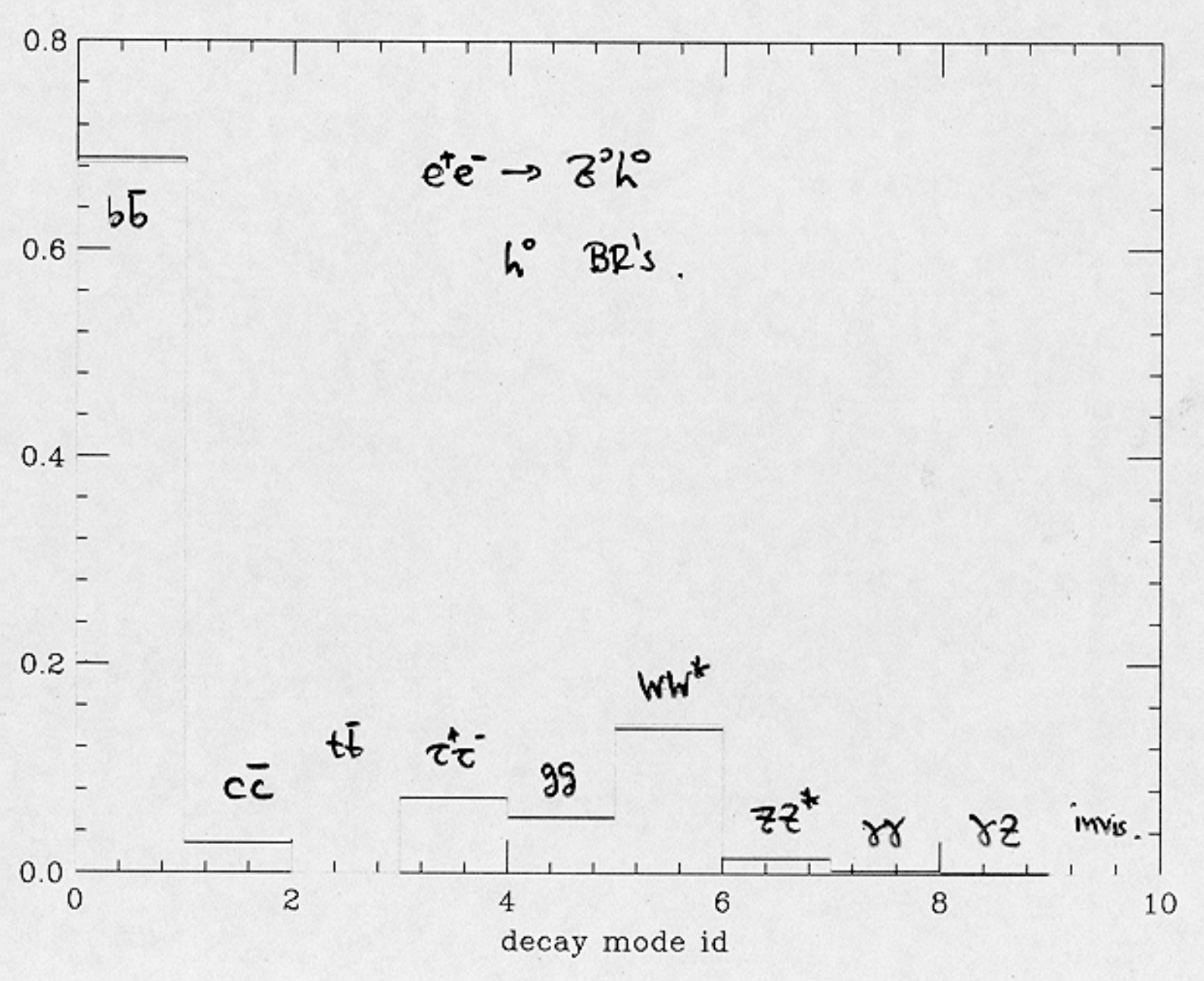

$e^+e^- \to t\bar{t}$
W
t
0.5
0.4
0.3
0.2
0.1
0.0
0
50
100
150
200
m(t), m(W)


$e^+e^- \to t\bar{t}$
beamstrahlung + ISR
l
b
W
ν
0.10
0.08
0.06
0.04
0.02
0.00
0
50
100
150
200
250
E


$e^+e^- \to t\bar{t}$
beamstrahlung + ISR on
0.10
0.08
0.06
0.04
0.02
0.00
0
50
100
150
200
250
E


$e^+e^- \to Z^0h^0$
Z^0
h^0
0.20
0.15
0.10
0.05
0.00
100
150
200
250
300
E (GeV)


0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
20
40
60
80
100
120
m(W)
e+e- → Z° h°
↳ WW
ZZ
W
Z
Z*
W*


$e^+e^- \to Z^0 h^0$
h^0 BR's.
$b\bar{b}$
$c\bar{c}$
$t\bar{t}$
$\tau^+\tau^-$
gg
WW^*
ZZ^*
$\gamma\gamma$
γZ
invis.
0.0
0.2
0.4
0.6
0.8
0
2
4
6
8
10
decay mode id

beam class:

$e^{\pm}$ beams have

ISR Fadin - Kuraev

\+

beamstrahlung 'consistent Yokoya - Chen'

input

a standard design

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

or machine parameters

$N \quad \beta_x \beta_y \quad \sigma_x \sigma_y \sigma_z$

```
class beam{

 protected:

    double Ebeam, polarization;
                /* basic beam parameters: energy and polarization  */

    int from, to;
                /*  particle ID's:
                    from ->  particle nominally entering the  collision
                    to   ->  particle making the hard reaction
                                                                    */

 public:

    int n;              /* number of integration variables */

    DVector distrib;
     /*  placeholder for the distribution functions for each helicity
              generated by the beam   */

    inline double getEbeam(){return Ebeam;}
    inline double getPolarization(){return polarization;}

    beam(double EB, double Pol, int From, int To, int N):Ebeam(EB),
           polarization(Pol), n(N), from(From), to(To), distrib(-1,1){};

    virtual double computeX(DVector & Z)= 0;
       /*  return the value of the longitudinal fraction x determined
                     by the integration variables Z ,
           and write the needed intermediate fractions to class variables

    virtual void distribution() = 0;
       /*  use the results stored in computeX to
         compute the distribution function f(x) for each helicity
           at the position determined by the integration variables Z
     the distribution is normalized to:  sum(helicity,to)  int f(x) dx  =

 };
```

— = consistent Yokoya-Chen

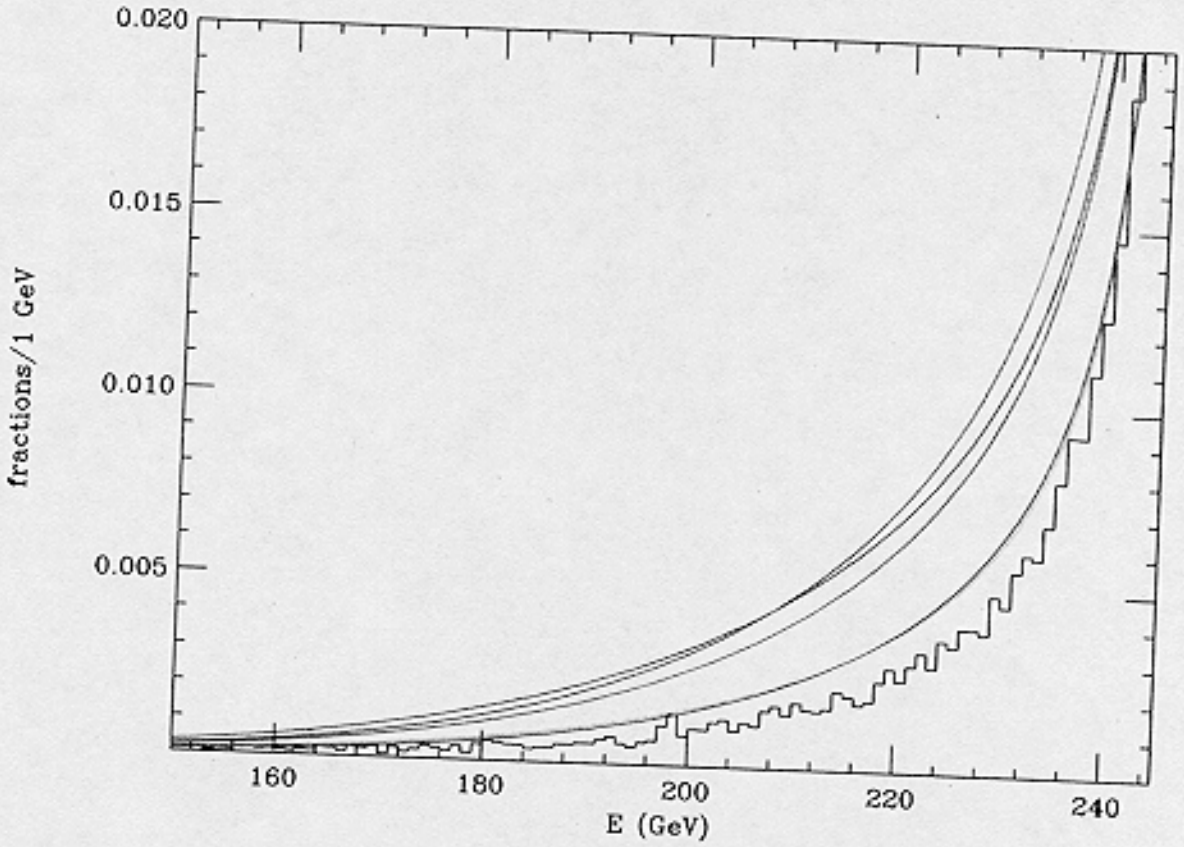


Figure 1: Comparison of Guinea Pig simulation data on the electron energy spectrum with various analytic approximations for the NLC-500B design parameters: blue-P1, green-P2, red-YC1 over YC0, magenta-C2.

Guinea Pig simulation data from Kathy Thompson !

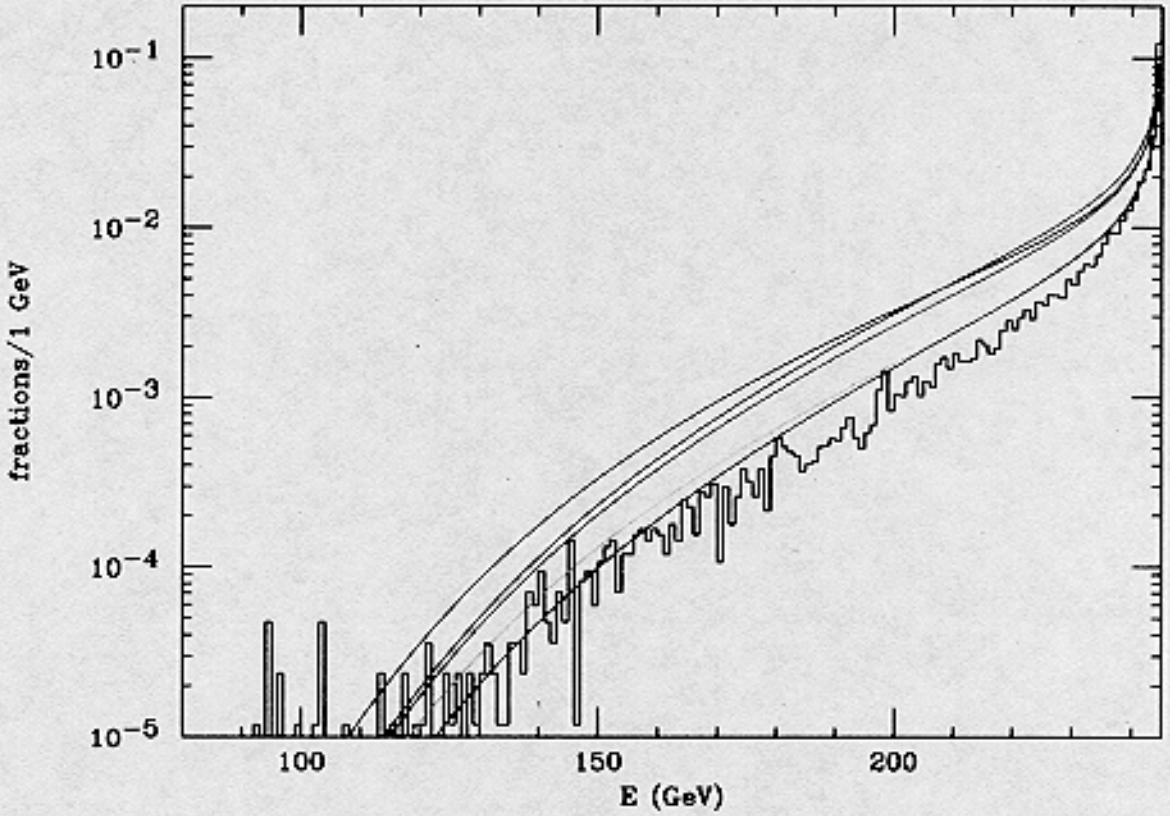


Figure 2: Figure 1 with a logarithmic scale, showing the low-energy, low-x tail of the distribution.

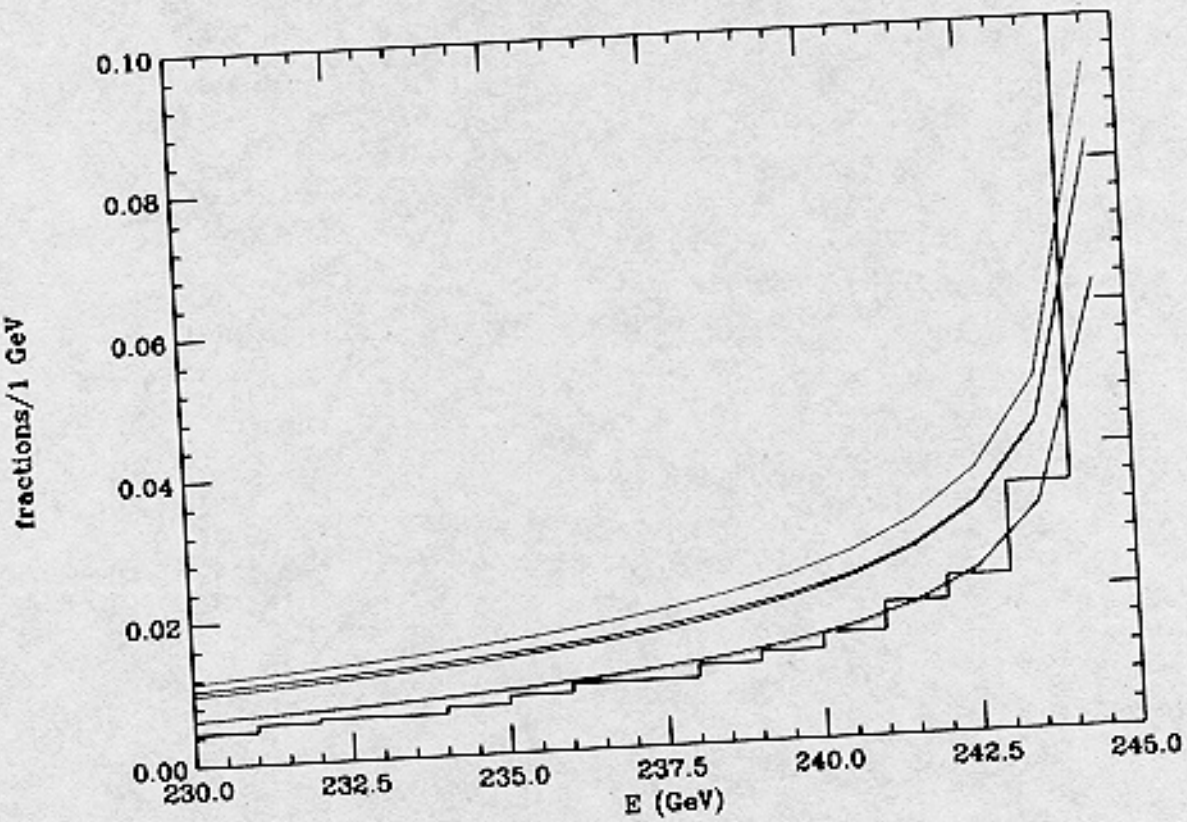


Figure 3: Blowup of Figure 1 concentrating on the highest-x bins.

event selection

weight - 1 ; no approximation after $\frac{d\sigma}{dx}$ is given

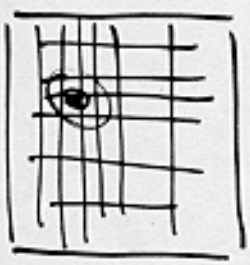
use VEGAS algorithm as in BASES/SPRING

model function by adjusting a coordinate grid

find max. weight

keep pts w. probability

$$\left(\frac{\text{weight}}{\text{max. weight}}\right)$$



standard VEGAS chooses grid to minimize variance

If instead minimize max. weight, factor 4-10 speedup!

event selection times! (Flora)

$e^+e^- \to t\bar{t}$ --- $e^+e^- \to l^+l^-$ $e^+e^- \to Z^0$

2 msec. --- 10 msec.

Is there better algorithm ?

actually

Pandora ← Vegas MC ← MonteCarlo
(abstract class)

Pandora uses only methods (interface)
of MonteCarlo

so any other subclass of MonteCarlo can be
freely substituted for VegasMC

Jadach: Foam

Y. Chen: (another fractal model)

```
class MonteCarlo{

 public:

    int N;     /*  number of variables integrated over  [0,1]    */

    MonteCarlo(int N);       /* initializes MonteCarlo variables  */

    virtual double surface(DVector & X)=0;      /* function integrated */

    virtual void prepare(int nevents, int nseed = 1)=0;
         /* adaptation; nevents is the number of events
                        to be used in the adaptation step  */

    virtual DVector  getPoint()=0;
    virtual DVector  getPoint(double & weight)= 0;

    void reset(int nseed = 1);
         /*  resets the Monte Carlo counters,
                and resets integration results to 0  */

    void resetMC(int nseed = 1);
    /*   reset the Monte Carlo counters only   */

    double integral(double & sd);
        /*   returns the accumulated value of the integral, and the error *

        /*   data for computation of the integral   */

    double Ip, Np, I2p;

         /* running estimates of the integral
               integral = Ip/Np;
               sd       = sqrt(I2p - Ip*Ip/Np)/Np    */

    double maxweight, threshold, maxratio;

        /*  data of the Monte Carlo selection   */

    int presented, accepted, bad;
                /*   Monte Carlo counters:
                     presented = events sampled
                     accepted  = events accepted
                     bad       = events with weight > threshold
                  by default, threshold is set at 2.0 * maxweight, but
                     this can be adjusted by setThreshold below
                       maxratio = highest weight found  */

    void setThreshold(double x);
          /*  resets threshold = x * maxweight    */

    void printIntegral();

    void printStatistics();
```

process class

must implement the operations:

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

more specifically $\longrightarrow$

```
class process{

 public:

  int n;    /* number of integration variables X[i] in [0,1]
                 needed to specify the final state of the process    */

  DMatrix cs;

  process(int N): n(N), cs(-1,1,-1,1){}

  virtual int validEvent(DVector & X, double s, double beta) = 0;
    /* determine whether the LEvent, boosted longitudinally by beta,
         satisfies the kinematic cuts defining the cross section   */
  virtual double computeKinematics(DVector & X, double s) = 0;
    /*  compute the kinematic variables which determine the final-state
          momenta, and write the answers into appropriate class variables
            of the process
         the function should return J = the Jacobian of the transformation
           to the variables X_i from the usual variables for
         expressing differential cross sections.  For example, if there
          is one integration variable x1 = (1 + cos theta)/2, then
              J =  d cos theta/ dx1  = 2
        This form allows crosssection above to return the usual formula for
           the differential cross section     */

  virtual void crosssection() = 0;
    /* compute the cross section from the kinematic variables
         fixed by computeKinematics, without consideration of cuts
     The cross section should be returned by filling the class variable
         DMatrix cs(-1,1,-1,1), a 3 x 3 matrix which can hold the
          the cross sections for various initial helicities
             from [-1,-1] to [1,1]      */

  virtual LEvent buildEvent() = 0;
    /* return the parton-level LEvent determined by the kinematic variables
              fixed by computeKinematics    */

  virtual LVlist buildVectors() = 0;
    /*  use the results of computeKinematics to
         compute the list of 4-vectors determined by the kinematic variables
         fixed by computeKinematics */
}
```

example of $e^+e^- \to t\bar{t}$

```
#ifndef EETOTTBAR_H
#define EETOTTBAR_H

#include "pandora.h"
#include "WZtdecay.h"
#include "processes.h"

class eetottbar: public twototwomm {

  /*  integration variables:

            cos theta    = 2 X[1] - 1     in the CM
            m(t)         =
              sqrt( M*M + M*Gamma * tan [(PI - 2 Gamma/M)* (X[2] - 1/2)]
            m(tbar)        =
              sqrt( M*M + M*Gamma * tan [(PI - 2 Gamma/M)* (X[3] - 1/2)]
            cos chi      = 2 X[4] - 1      top decay angles
            psi          =  2 PI X[5]
            cos chib     = 2 X[6] - 1      tbar decay angles
            psib         =  2 PI X[7]
            m(W+)         =
              sqrt( M*M + M*Gamma * tan [(PI - 2 Gamma/M)* (X[8] - 1/2)]
            cos chiW      = 2 X[9] - 1     W+ decay angles
            psiW          =  2 PI X[10]
            m(W-)         =
              sqrt( M*M + M*Gamma * tan [(PI - 2 Gamma/M)* (X[11] - 1/2)]
            cos chiWb     = 2 X[12] - 1     W- decay angles
            psiWb         =  2 PI X[13]
                                        */

 public:

  eetottbar();

  int validEvent(DVector & X, double s, double beta);
  double computeKinematics(DVector & X, double s);
  void crosssection();
  LVlist buildVectors();
  LEvent buildEvent();

  double simplecrosssection(double cost, double s);
     /* returns the unpolarized differential cross section as a
          function of cos theta     */

   /*  buildEvent creates an LEvent with 10 LVectors with the following
          IDs:

          1    ->  t
    +     2    ->  b
          3    ->  W+
    +     4    ->  W+ decay l+ or qbar
    +     5    ->  W+ decay nu or q
          6    ->  tbar
    +     7    ->  bbar
          8    ->  W-
    +     9    ->  W- decay l- or q
    +    10    ->  W- decay nubar or qbar
```

How does one write a process class ?

① Compute helicity amplitudes for the process.

pandora's conventions: view processes in the event plane
(works up to $2 \to 3$)

for a vector boson in $+\hat{3}$ direction

$$\varepsilon^{\mu}_{+1} = \frac{1}{\sqrt{2}}(0\,1\,i\,0) \qquad \varepsilon^{\mu}_{0} = \left(\frac{k}{m}, 0, 0, \frac{E}{m}\right) \qquad \varepsilon^{\mu}_{-1} = \frac{1}{\sqrt{2}}(0, 1, -i, 0)$$

for a massive fermion in $+\hat{3}$ direction
(use 2-component notation !)

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

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

+ rotate in plane to other configurations.

② Call standard decay amplitudes for massive particles

decay classes are written for

W Z t h° (SM Higgs)

(It is crucial to adhere to the common conventions.)

```
class Wplusdecay: public decaytotwozz {

 /*     chi is the angle from the W direction to the right-handed fermion
                              (ie,  1+ )
        psi is the rotation angle about the W direction of motion   */
public:

    Wplusdecay();

    DVector BRs();      /* returns the BR's to (u,c,e, mu, tau)
                                                  as a DVector(1,5) */


    void decayamp(double coschi, double psi);
      /*  fill the  vector Camp[Whelicity]     */

    CVector Camp;
    double divideby;
        /*  for an on-shell W decay,
              integrate the squared amplitude over    d coschi  d phi;
                then divide by divideby      */

    /*  Wplusdecay inherits the method:

      LVlist buildDecayVectors(double m, double coschi, double psi)

        returns a list of three LVectors in the order
                      1    W+
                      2    1+   or qbar
                      3    nu   or q
        the kinematics of the decay are based on m, not on the argument of
                                          the constructor
                                                          */


    void placeIDs(LEvent & LE, int i, int parent);
      /*  appropriately fills in 3 rows of an LEvent, beginning with
            row i, with the ID information for a real W+
            decay to quark or lepton pairs
          Pass the row corresponding to the parent of the W+ in parent   */

    void placeIDs(LEvent & LE, int i, int parent, int ID);
      /*  appropriately fills in 3 rows of an LEvent, beginning with
            row i, with the ID information for the products of a
            virtual W decay to quark or lepton pairs.
          Pass the row corresponding to the parent of the decaying particle
              in parent, and pass the ID of this particle in ID  */
```

```
class Higgsdecay : public generaldecay {

public:

   Higgsdecay(double mh);

   /*  The Higgs decays to final states containing different numbers of
       final particles and different numbers of integration
        variables.  The value x passed to the Higgsdecay functions
        keeps track of this.  Based on the value of x, the Higgsdecay
        class method choosecase will choose one of the following channe
          and the other decay functions will act accordingly:

                 1  b bbar
                 2  c cbar
                 3  t tbar
                 4  tau+ tau-
                 5  g g
                 6  W W (including W W*)
                 7  Z Z (including Z Z*)
                 8  gamma gamma
                 9  Z gamma
                10  invisible final states
                       (coded in the LEvent as nu nubar )

      The default is to take the partial widths given in the Minimal
        Standard Model.  However, the following commands allow one
        to modify these widths:
   */

   void newWidth(int channel, double Width);
      /*  resets Gamma(h-> channel) to the indicated value,
             and recomputes the branching ratios   */
   void onlyDecay(int channel);
    /*  sets all decay widths except Gamma(h-> channel) to zero */

   DVector partialwidths();
         /* returns the partial widths as a DVector(1,10)*/
   DVector BRs();
         /* returns the  BRs as a  DVector(1,10) */
   DVector SMwidths();
     /* returns the Standard Model partial widths as a DVector(1,10)*/

   void newhmass(double mh);
         /*  set the Higgs mass to the new value, and sets
               partial widths to the new Standard Model values  */
   double mass(){ return mh; }

   int validDecay(DVector & X, int i, double MH);
        /*  returns 0 if X does not correspond to a physically correct
              decay configuration; returns 1 otherwise
            this function uses the 13 doubles in X beginning with X[i]
              and the mass MH of the h computed by the production class
```

③ Inherit from classes which compute reaction kinematics

these include finite width effects and cut off singularities (eg. at $\cos\theta \to 0$)

treatment of finite width:

Breit Wigner about M → $m_1 m_2 + \sqrt{s}$

$\sqrt{s}$

P

use $E_{ia} = (p^2 + m_i^2)^{\frac{1}{2}}$ for kinematics

$E = (p^2 + M^2)^{\frac{1}{2}}$ to compute amplitudes

```
class twototwomm : public process {

  /*  integration variables:
     cos theta =  2.0 * X[1] - 1.0   in the CM
     mass1  = sqrt( M1*M1 + M1*Gamma1 *
                                  tan [(PI - 2 Gamma/M) * (X[2] - 1/2)]
     mass2  = sqrt( M2*M2 + M2*Gamma2 *
                                  tan [(PI - 2 Gamma/M) * (X[3] - 1/2)]
                                                          */


 public:

   twototwomm(int N, double M1, double G1, double M2, double G2);

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

   double computeKinematics(DVector & X, double s);
   LVlist buildVectors();

 protected:

   double M1, G1, M2, G2;
   double m1, m2;
   double s, p, E1a, E2a, E1, E2, cost, sint, phi;
          /* center of mass momentum and energy and cos theta, sin theta
             E1a = (p*p + m1*m1),    E2 = (p*p + m2*m2),
             E1 = (p*p + M1*M1),     E2 = (p*p + M2*M2),   */
```

```
class twototwomzt : public process {

  /*  integration variables:
            cos theta =  tanh( (2 X[1]-1) * 10)
        to give the cross section in the CM up to 10^-5 rad
      mass  = sqrt( M*M + M*Gamma * tan [(PI - 2 Gamma/M)* (X[2] - 1/2)]
                                                                  */

 public:

   twototwomzt(int N, double M, double G);
   twototwomzt(int N, double M,  double G, double thetamin,
                              double ptmin, double Emin);
   twototwomzt(int N, double M,  double G, double thetamin, double thetamax,
         double ptmin, double ptmax,  double Emin, double Emax);

  /* the second of these computes the cross section over the region
      in which the massless particle (usually a photon) obeys,
        in the lab frame, | theta | > thetamin,  E > Emin, |pt| > ptmin.
      The default values are: thetamin = 10 mrad, ptmin =  0, Emin = 2 GeV.
              These default values
          are implemented in the constructor with no arguments.
      The third constructor allows one to produce disjoint event samples.
                                                                  */

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

   double computeKinematics(DVector & X, double s);
   LVlist buildVectors();
```

④ Code the q.m. helicity amplitudes

⑤ Construct the final state parton momenta

using boosts + rotations of

4-vector lists from decay classes

⑥ Add ID's etc. to LEvent

using functions from decay classes

```
LVlist eetottbar::buildVectors(){
      /* prepare t decay products    */
   LVlist Lt = TD.buildDecayVectors(m1,mW,coschi,psi,coschiW,psiW);
   Lt.boost(p/E1a);
        /* prepare tbar decay products    */
   LVlist Ltb = TBD.buildDecayVectors(m2,mWb,coschib,psib,coschiWb,psiWb);
   Ltb.boost(p/E2a);
   Ltb.reverseinplane();
   LVlist LL = merge(Lt,Ltb);
   LL.rotateinplane(cost);
   LL.rotate(phi);
   return LL;
}

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

current status of included processes :

e^+e^-, e^-e^- beam classes

$e^+e^- \to l^+l^-$, e^+e^-, $\gamma\gamma$, $t\bar{t}$

W^+W^-, $Z\gamma$, Z^0Z^0, Z^0h^0

$e^+e^- \to t\bar{t}$ w. nonstandard couplings (C. Ferretti)

$\gamma\gamma \to l^+l^-$ e^+e^- $t\bar{t}$

goal for next distribution (by end of July 2000)

$\gamma\gamma$ beam class

$e^-\gamma \to e^-\gamma$ e^-Z^0 νW^-

$\gamma\gamma \to h^0$