

APPENDIX B

SOURCE CODE FOR AVHT IMPLEMENTATION

The following *C* source code represents the actual code employed for the tracking and data analysis described in this text. The functions and header files are listed in roughly order of importance to the end user, except for *stat.h* and *stat.c* which are last only because they are designed solely for the purposes of analysis and do not contribute to tracking. The header files, listed first, describe the variables and functions of relevance to one employing the source files. Further comments within the source, where appropriate, elucidate the inner workings of the functions.

B.1 hougher.h

```
/*
*****
'C' Interface to the Houghing Tracker.

By Erik Kangas, 1/15/1995
Last Modified 4/28/95 by Erik Kangas
*****
*/

#ifndef BJTRACK
#define BJTRACK

#include "bjhough.h" /* Hougher module */

/*
*****
Global Variables

event: This is the global event structure. You need to fill it
with the event to be tracked before calling off_tracker_. You
can do the with functions like GetOFFEvent (in trackdef) or
GetTracks (in stat). Can be accessed before setup_tracker

tracks: This is the global Tracks structure that is filled by
off_tracker_ with the tracks found in the event. Do not
access before calling setup_tracker (as setup_tracker allocates
memory for it).

Verbose: =1, functions in hougher.c will print status information
to the screen at various stages of tracking and setup. =0, no
information will be displayed. default = 0
*****
*/
```

```

PRECOMPUTE flags whether or not to precompute hougher tube intercepts.
  1 = yes, 0 = no. This can only has effect before the call to
  Setup_Hougher. It defaults to 1.

Nevents: Counts the number of events passed to hougher_.
  Can be accessed globally.

ChiCutH, ChiCutT: What values to use for the Chi^2_v cut for
  head and tail tracking. Default to 3.0 for Heads and 3.0 for
  tails. These can be changed at ANY TIME.

MetCutH, MetCutT: Values for the track metric cut for head and
  tail tracking. Default to 2.5 for heads and 5.0 for tails.
  These can be changed at ANY TIME.

MinPlaH, MinPlaT: MinPlanes cut for head and tail tracking.
  Default to 7 and 14, respectively.
  These can be changed at ANY TIME.

*/

extern Event event; /* Event storage */
extern Tracks* tracks; /* Tracks found */

extern int PRECOMPUTE; /* DO precomputation ? */
extern int Verbose; /* Display status */
extern int Nevents; /* # events tracked */

extern double ChiCutH, /* Chi^2_v cuts to use for head/tail tracking */
              ChiCutT,
              MetCutH, /* Metric cut to use for both */
              MetCutT;
extern int MinPlaH, /* MinPlanes cut for heads and tails */
          MinPlaT;

/*****

setup_hougher_:
  Must be called at program initialization: This function sets up
  the tracking environment and loads the detector data file, etc.
  You must pass it the name of the detector data file to use.

hougher_:
  This function does event-by-event tracking. It assumes you have
  filled the global Event structure 'event' with the event and it returns
  the found tracks in the global Tracks structure 'tracks.' Error codes
  are returned as follows:
    0 == OK
   -1 == Tracks overflow (too many candidates found)
   -2 == Unknown Hougher Error

*/

void setup_hougher_(char* det_file);
int hougher_(void);

#endif

```

B.2 hougher.c

```

/*****

'C' Interface to the Houghing Tracker.

```

By Erik Kangas, 11/3/1994
 Last Modified 4/28/95 by Erik Kangas

```

*****/

#include "hougher.h"
#include "stdlib.h"
#include "string.h"

/*****
  Global Tracking Variables...
*/

Tracks *tracks,          /* Keep track of parts of tracks */
      *tails;           /* Used to hold current set of tracks */

Event event;

int   Nevents = 0;      /* Current event number */
int   Verbose = 1;     /* Print out status at various stages? */

double ChiCutH = 2.0,  /* Chi^2_v cuts to use for head/tail tracking */
       ChiCutT = 2.85,
       MetCutH = 2.5,  /* Metric cut to use for both */
       MetCutT = 5.0;

int   MinPlaH = 7,     /* MinPlanes cut for heads and tails */
       MinPlaT = 14;

/*****

  setup_hougher_:
  Must be called at program initialization: This function sets up
  the tracking environment and loads the detector data file, etc.
  You must pass it the name of the detector data file to use

*/

void close_hougher_(void);

void setup_hougher_(char* det_name) {
  int error;

  if (Verbose) printf("Setting up Tracker.\n");

  /* Initialize the variables you will use for tracking */
  /* Create the tracks data structure to hold tracks found */

  tracks = (Tracks*) malloc(sizeof(Tracks));
  tails = (Tracks*) malloc(sizeof(Tracks));

  if (!tracks || !tails) {
    printf("setup_hougher_: memory allocation error.\n\n");
    exit(1);
  }

  /* Setup the houghing environment. Pass the name of the detector */
  /* data file to use. The file should be in the current directory */

  if ((error = Setup_Hougher(det_name)) == -1) {
    printf("Error SetupHougher: Error Level %d.\n", error);
    exit(error);
  }

  atexit(close_hougher_);
}

/*****

```

```

close_tracker_:
    Closes down the tracking environment.  Frees used memory, etc.
    Must be called at program termination.  It's is set atexit by
    setup_tracker, so the user needn't worry about calling it explicitly.
*/

void close_hougher_(void) {
    free(tracks);
    free(tails);
}

/*****
    hougher_:
        This function does event-by-event tracking.  It assumes you have
        filled the global Event structure 'event' with the event and it returns
        the found tracks the th global Tracks structure 'tracks.'  Error codes
        are returned as follows:

            0    == OK
           -1    == Tracks overflow (too many candidates found)
           -2    == Unknown Hougher Error
*/

int hougher_(void) {
    static int i,j,k,bn,an;

    /* Initialize tracking variables */

    Nevents++;
    tracks->event      = event.event;
    tracks->nhits      = event.nwires;
    tracks->ntracks     = 0;

    for (i=tracks->mcut=bn=an=0;i<Planes;i++) {
        for (j=k=0;j<128;j++) k += event.w[i][j];
        if (k > tracks->mcut) tracks->mcut = k;
        if (i < LeadP) bn += k;          /* NHits before and after Pb */
        else an += k;
    }

    if (Verbose)
        printf("In Tracker: Event %ld,%d\tNHits: %d (%d/%d)\n",event.event,
            Nevents,event.nwires,bn,an);

    /* Setup for tracking Heads */

    FPlane      = 0;          /* Track from plane 0 - before the PbC */
    LPlane      = LeadP-1;
    ChiCut      = ChiCutH;    /* Chi^2_v cut on track quality */
    MetricCut   = MetCutH;    /* Cut on probability tracks are the same */

    for (i=0,MinPlanes = LeadP; MinPlanes >= MinPlaH; MinPlanes--) {
        Bj_hough(&event, tracks);
        for (;i<tracks->ntracks;i++)
            for (j=0;j<tracks->tracks[i].nwires;j++)
                event.w[tracks->tracks[i].wires[j].pn]
                    [tracks->tracks[i].wires[j].wn] = 0;
    }

    tracks->nheads = tracks->ntracks;

    /* Track after the lead looking for tails or gammas */

    tails->ntracks = 0;
    ChiCut      = ChiCutT;    /* Chi^2_v track quality cut */

```

```

MetricCut = MetCutT;      /* Cut of probability that tracks are the same */
FPlane    = LeadP;      /* Track all planes after the PbC */
LPlane    = Planes-1;

for (i=0,MinPlanes = Planes-LeadP; MinPlanes >= MinPlaT; MinPlanes--) {
    Bj_hough(&event, tails);
    for (;i<tails->ntracks;i++)
        for (j=0;j<tails->tracks[i].nwires;j++)
            event.w[tails->tracks[i].wires[j].pn]
                [tails->tracks[i].wires[j].wn] = 0;
}

/* Just keep heads and tails for now */
/* Keep as many tails as possible w/o exceeding MAXTRACKS */

tracks->ntails = (tracks->nheads + tails->ntracks <= MAXTRACKS) ?
                tails->ntracks : MAXTRACKS - tracks->nheads;
tracks->ntracks = tracks->nheads + tracks->ntails;

if (tracks->ntails > 0)
    memcpy(&tracks->tracks[tracks->nheads],tails->tracks,
           sizeof(Track)*tracks->ntails);

if (Verbose) printf("Total Found %d charged, %d gamma tracks\n",
                    tracks->nheads,tracks->ntails);

return(0);
}

```

B.3 trackdef.h

```

/*****
Hougher Tracker General Tracking Structure Declarations

By Erik Kangas
Case Western Reserve University
2-24-94
3rd generation tracker -- 11/17/94 -- Erik Kangas
Last Revised 5-8-95, Erik Kangas

This file contains Tracking constant definitions as well as
data structures and general purpose functions.
*/

#ifndef TRACK_DEF
#define TRACK_DEF

#include "coords.h"

/*****
*/
/* General Constants */

#define MAXTRACKS 350      /* Max Tracks in an Event */
#define MAXWIRES 48       /* Max wires in a track */
#define MAXTAILS 30       /* Max tails to associate in a group */
#define MAXGRPS 100       /* Max track groups / event */

extern long idum;         /* Random seed */

/*****
Event data structure

The Wire data structure holds individual wire information

```

```

    pn -> Plane number (0...Planes-1)
    wn -> Wire number (0...127)

The Event data structure holds information for a full event
    nwires -> number of active wires in the event (nhits)
    event -> event id number
    nw -> # wires in current AVHT tube
    w -> matrix of active wires
*/

typedef struct WIRE {
    unsigned char pn,wn;
} Wire;

typedef struct EVENT {
    int nwires;
    long event;
    char nw[Planes];
    char w[Planes][128];
} Event;

/*****
*
Track data structure
Track holds information pertaining to a single track.

    the track is defined to pass through the points p0 and p1
    p0.z, p1.z are the z coords of FPlane and LPlane
    These are in new coordinate system.
    chisqr -> reduced chi-squared value of the track's fit (or kinky fit)
    type -> Track type -- see the defines below
    nwires -> number of wires contributing to the track
    wires -> list of wires contributing.
    type -> is it a ttHead or a ttTail?

Tracks holds info for all tracks in an event
    nheads -> # of tracks from before the PbC
    ntails -> # of tracks from after the PbC
    ntracks-> total number of tracks
    nhits -> number of active wires in the event
    mcut -> occupancy of the most active chamber in the event
    tracks -> list of Track info structs for each track, heads first.

One might want to adjust MAXTRACKS to accommodate particularly
large multiplicities.
*/

#define ttHead 0 /* Before the lead segment */
#define ttTail 1 /* After the lead segment */

typedef struct TRACK {
    dVector p0,p1;
    Vtype chisqr;
    Wire wires[MAXWIRES];
    int type,nwires;

    /* Stuff for Geant DST tracks */

    Vtype E; /* Momentum or particle */
    int id; /* kind of particle */
    float x,y,z; /* Origin of particle */
} Track;

typedef struct TRACKS {
    int nheads,ntails,ntracks;
    int nhits,mcut;
    long event;

```

```

Track tracks[MAXTRACKS];
} Tracks;

/*****
*
Head-Tail matching data structure.
The GroupInfo structure holds info for 1 head and all of it's tails.
index -> index into tracks structure for this track's information.
ntails -> number of tails associated w/ this head.
primary -> index of primary tail.
tails -> list of 'ntails' indicies for all the tails associated.
v -> vertex information

for (1,1) tracks, you set index to the head index, ntails = 1,
primary = tails[0] = index of the tail. Vertex = best match
between head and tail (average or LSF).

for (1,n) you so the same, except you fill in more tails and change
ntails. Vertex taken to be wither place of the head or the avg.
tails position of the PbC.

for (0,n) you set index = -1 and fill in the rest w/ the tails.
Set the vertex to the best vertex on the PbC.

for (1,0) you set ntails = 0 and ignore the vertex information.

Groups:
Store information about separate groups here.

*/

typedef struct {
    int index; /* Which element in track's Structure? */
    int ntails; /* How many tails does it have? */
    int primary; /* index to it's primary tail, if any */
    int tails[MAXTAILS]; /* Store indicies to all of it's tails */
    dVector v; /* Vertex of tracks */
} GroupInfo;

typedef struct {
    int ngroups; /* Number of track groups in the event */
    GroupInfo groups[MAXGRPS]; /* Group information */
} Groups;

/*****
Interface to OFFLINE

EVTICOM:
structure that UNPACKER puts raw event into
FLAGS:
OFFLINE common block used to pass program flags

*/

/* UNPACKER Event structure */

typedef struct EventCommon {
    int length; /* Event Length in 32 bit words */
    int evnum; /* Event Number */
    int evtype; /* Event Type */
    int trigger; /* Trigger Bits */
    int nhits; /* # hits in event */
    int nadc; /* Number of ADC channels in event */
    int ntdc; /* Number of TDC channels in event */
    int nchamb; /* Number of CHAMBERS in event */
    int num_latch; /* Number of Nano Latches in experiment */
    int num_mich; /* Number of Mich Elec Channel in experiment */
}

```

```

int nhitmax;          /* Total number of wire channels in experiment */
int version_num;     /* Version number of the READEVENT.EVD file */
int chamb_num[3072]; /* chamber # for i'th hit */
int wire_num[3072];  /* wire # for i'th hit */
int pls_hgt[3072];   /* pls hgt for i'th hit (= 10000 for latches) */
float ped_sub_pls[3072]; /* pedestal value for i'th chamb, j'th wire */
float pedds[24][128]; /* pedestal subtracted pls hgt (=0. for latches)
*/
int adc[108];        /* pls hgt for i'th ADC chan (calorimeter cell) */
int tdc[48];         /* pedestal value for i'th ADC Channel */
float ped_sub_adc[108]; /* pedestal subtracted pulse height for i'th ADC
*/
float pedds_adc[108]; /* time for i'th counter */
int regist[32];      /* register value for i'th input */
int event_valid;     /* T if everything is OK */
int partial_event;   /* All valid's Ored */
int adc_valid;       /* T if ADC's read out OK */
int tdc_valid;       /* T if TDC's read out OK */
int nano_valid;      /* T if nano's read out OK */
int mich_valid;      /* T if mich read out OK */
} EVTCOM;

```

```

/*****
General purpose, generic functions

```

ClearEvent:

Clears an event structure -- zeros the counter variables.

ClearTrack:

Clears a track structure to zero.

DeleteTrack:

Removes a track from a Tracks list and adjusts the list and relevant variables appropriately (ngamma, ncharged, etc. adjusted).

CopyTracks:

Copys all the tracks from one track structure into another.

ClearGroups:

Zeros the groups structure.

WriteTEvent:

Function to write a set of found tracks to a storage file (DST)

Inputs:

out_file -> track storage file
tracks -> structure to be stored

Format used:

first line added to file:
(8d=evt_num,8d=nheads,8d=ntails,8d=mcut,7d=nhits)
For each track there are 2 lines:
(10.4f=x0,10.4f=y0,10.4f=z0,10.4f=x1,10.4f=y1,10.4f=z1,
10.4f=chi^2_v,10d=nwires,2d=type)
(6d=wire1,...,6d=wiren)

The wires are coded as plane * 128 + wire + 1 where plane and wire are counted from 0.

WriteEvent:

Writes out the given event simply as an event #, list of wire #s and a '-1\n' terminator. Can be read in again by LoadEvent.

LoadEvent:

Loads the next event from a file created by successive calls to 'WriteEvent' fills event number and active wires. Returns nhits.

LoadEventOFF:

Function to convert a single event loaded from UNPACKER or LeadGeantEvent().

Stores events in event structure.

Inputs:

evt -> UNPACKER event common block
event -> event to be overwritten.

LoadTEvent:

Function to load a single event from a track storage file (DST) into a Tracks structure. DST File assumed to have been created by successive calls to WriteTEvent.

Inputs:

in_file -> track storage file
tracks -> output structure

Returns:

0 = Ok, -1 = file error

LoadGeantEvent:

Loads the next event from a GEANT binary data file. Loads it into an event common block. This is a 'C' analog of 'unpacker' for GEANT. It must be modified to Mary changes the GEANT file fmt.

LoadGeantDST:

Loads the next event in the GEANT DST into a Tracks structure. It stores all the tracks found, the Energy, type and origin of each particle (see the GEANT-specific fields in Track). Nood to call SliceGeant function in stat.c to cut the tracks up into heads and tails.

Rand:

Returns a random number b/n zero and 1. You should seed idum with the time to produce a unique sequence. This is a Numerical Recipies random # generator.

GRand:

Returns gaussianly distributed random number with given mean and standard deviation

*/

```
void ClearTrack(Track* trk);
void ClearEvent(Event* evt);
void DeleteTrack(Tracks* tracks,int tn);
void CopyTracks(Tracks* dest, Tracks* source);
void ClearGroups(Groups* grp);

void WriteTEvent(FILE *out_file, Tracks *tracks);
void WriteEvent(FILE* outf, Event* evt);

void LoadEventOFF(EVTCOM* evt,Event* event);
int LoadTEvent(FILE *in_file, Tracks *tracks);
int LoadGeantEvent(EVTCOM* evt, FILE* inf);
int LoadGeantDST(Tracks* trk, FILE* inf);
int LoadEvent(FILE* inf, Event* evt);

Vtype Rand(void);
Vtype GRand(Vtype std, Vtype mean);
#endif
```

B.4 trackdef.c

/*

General Tracking Structure Declarations
General All-purpose defines and includes

By Erik Kangas
Case Western Reserve University
2-24-94

3rd generation tracker -- 11/17/94 -- Erik Kangas
Revised 5-8-95 -- Erik Kangas

```
Function declarations
Static data storage
*/

#include "trackdef.h"
#include "bjhough.h"
#include <string.h>
#include <math.h>
#include <stdlib.h>

/*****
Utility functions
*/

void DeleteTrack(Tracks* trk,int n) {
    if (n >= trk->nheads) trk->ntails--;
    else trk->nheads--;

    if (n < trk->ntracks-1)
        memmove(&trk->tracks[n],&trk->tracks[n+1],sizeof(Track)*
            (trk->ntracks - n - 1));

    trk->ntracks--;
}

void CopyTracks(Tracks* dest, Tracks* source) {
    memcpy(dest,source,sizeof(Tracks));
}

void ClearEvent(Event* evt) {
    memset(evt,0,sizeof(Event));
}

void ClearTrack(Track* trk) {
    memset(trk,0,sizeof(Track));
}

void ClearGroups(Groups* grp) {
    memset(grp,0,sizeof(Groups));
}

/*****/

void WriteTEvent(FILE *out_file, Tracks *tracks) {
    int i,j;

    fprintf(out_file, "%7ld %7d %7d %7d %7d\n",
        tracks->event, tracks->nheads, tracks->ntails, tracks->mcut,
        tracks->nhits);

    for (i = 0; i < tracks->ntracks; i++) {
        vdPrint(out_file, MMult( MCBtoOld, tracks->tracks[i].p0));
        vdPrint(out_file, MMult( MCBtoOld, tracks->tracks[i].p1));
        fprintf(out_file, "%9.4lf %9d %2d\n",
            tracks->tracks[i].chisqr, tracks->tracks[i].nwires,
            tracks->tracks[i].type);
        for (j=0;j<tracks->tracks[i].nwires;j++)
            fprintf(out_file,"%5d ",tracks->tracks[i].wires[j].pn*128 +
                (int)tracks->tracks[i].wires[j].wn + 1);
        fprintf(out_file,"\n");
    }
}
}
```

```

void WriteEvent(FILE* outf, Event* evt) {
    int p,n;
    fprintf(outf,"%ld ",evt->event);
    for (p=0;p<Planes;p++) for (n=0;n<128;n++)
        if (evt->w[p][n]) fprintf(outf,"%d ", p*128 + n);
    fprintf(outf,"-1\n");
}

int LoadGeantDST(Tracks* trk, FILE* inf) {
    static int evnum, nch, nnu, nmisc, mcut, nhits;
    static int i,w,id,wh,j,dummy,r;
    static Vtype c0,x,y,z,px,py,pz,dir;
    static char input[128];

    /* Read general event info */

    if (fgets(input,127,inf) == NULL) return -1;

    if ((r=sscanf(input,"%d %d %d %d %d %d %lf",&evnum,&nch,&nnu,
        &nmisc,&mcut,&nhits,&c0)) != 7) return -1;

    trk->event = evnum;
    trk->ntracks = nch + nnu + nmisc;
    trk->nheads = nch;
    trk->ntails = nnu;
    trk->mcut = mcut;
    trk->nhits = nhits;

    /* read info for each track */

    for (i=0;i<trk->ntracks;i++) {
        if (i >= MAXTRACKS) {
            printf("There are too many %d tracks in GEANT DST event #d\n",
                i,evnum);
            printf("Truncating the event, %d tracks lost\n",
                j = trk->ntracks-MAXTRACKS);
            i -= j;
            trk->ntracks -= j;
        }

        if (fgets(input,127,inf) == NULL) {
            printf("GEANT DST Error 3\n",r,input);
            return -1;
        }

        if ((r=sscanf(input,"%lf %lf %lf %lf %lf %lf %d %d %lf %d",
            &px,&py,&pz,&x,&y,&z,&id,&wh,&dir,&dummy)) != 10) {
            printf("GEANT DST Error 2, read %d, evt %d, trk %d, '%s'\n",r,
                evnum,i,input);
            return -1;
        }

        trk->tracks[i].E = sqrt( px*px + py*py + pz*pz );
        trk->tracks[i].id = id;
        trk->tracks[i].x = x;
        trk->tracks[i].y = y;
        trk->tracks[i].z = z;

        trk->tracks[i].nwires = wh;
        if (i < nch) trk->tracks[i].type = 0;
        else if (i < nch+nnu) trk->tracks[i].type = 1;
        else trk->tracks[i].type = 2;

        if (wh > MAXWIRES) {
            printf("Too many wires %d in GEANT DST Event %d, track %d\n",

```

```

        wh, evnum, i);
    exit(1);
}

for (j=0; j<wh; j++) {
    if (fscanf(inf, "%d", &w) != 1) {
        printf("GEANT DST Error 3, evt %d, trk %d, wire %d\n", evnum, i, j);
        exit(1);
    }
    trk->tracks[i].wires[j].pn = (w-1)/128;
    trk->tracks[i].wires[j].wn = ((w-1)%128);
}
fgets(input, 127, inf);      /* Get EOL sequence */
}

/***** TAKE THIS OUT LATER!!! *****/

trk->ntracks = nch + nnu;

return 0;
}

int LoadGeantEvent(EVTCOM* evt, FILE* inf) {
    long int nsec, ii, i, nwords;
    long int buf[15000];

    fread(&nwords, 4, 1, inf);      /* File length data */
    if (fread(&nwords, 4, 1, inf) != 1) return -1;
    if (fread(buf, 4, nwords-1, inf) != nwords-1) return -1;
    fread(&nwords, 4, 1, inf);      /* File length data */

    evt->length = nwords;
    evt->evnum = buf[0];
    nsec = buf[3];
    evt->nhits = buf[3 + 7*nsec + 1];
    ii = 3 + 7*nsec + 2;

    /* Read in wires and planes */

    for (i=0; i< evt->nhits; i++, ii+= 3) {
        evt->chamb_num[i] = buf[ii] + 100;
        evt->wire_num[i] = buf[ii+1];
        evt->pls_hgt[i] = buf[ii+2] / 100; /* in KeV */
    }

    /* Read in TDC, ADC and ECAL */

    for (i=0; i<8; i++, ii++) evt->adc[36+i] = buf[ii] / 100; /* Proton (MeV) */
    for (i=9; i<16; i++, ii++) evt->adc[39+i] = buf[ii] / 100; /* pbar (MeV) */
    for (i=16; i<23; i++, ii++) evt->adc[i-16] = buf[ii] / 100; /* A-D */

    ii++;      /* Skip E */

    for (i=0; i<16; i++, ii+=2) evt->adc[19+i] = buf[ii+1] / 10; /* Ecal */
    return 0;
}

void LoadEventOFF(EVTCOM *evt, Event *event) {
    static int i, p;

    ClearEvent(event);
    event->event = evt->evnum;
    event->nwires = evt->nhits;

    /* Mark all appropriate wires as active and set weights */

    for (i = 0; i < evt->nhits; i++) {

```

```

        p = (evt->chamb_num[i] -1)%100;
        event->w[p][evt->wire_num[i]-1] = 1;
    }
}

int LoadEvent(FILE* inf, Event* evt) {
    int p,n,w,nhits=0;
    ClearEvent(evt);
    if (fscanf(inf,"%ld",&evt->event) != 1) return -1;
    if (fscanf(inf,"%d",&w) != 1) return 0;

    while (w != -1) {
        p = w/128; n = w % 128;
        evt->w[p][n] = 1;
        evt->nw[p]++;
        evt->nwires++;
        nhits++;
        if (fscanf(inf,"%d",&w) != 1) return nhits;
    }
    return nhits;
}

int LoadTEEvent(FILE *in_file, Tracks *tracks) {
    int i,type, j, k;
    Vtype chisqr;
    dVector p0,p1;

    if ((i = fscanf(in_file, "%ld %d %d %d %d",
        &tracks->event, &tracks->nheads,
        &tracks->ntails,&tracks->mcut,&tracks->nhits)) != 5) return -1;

    tracks->ntracks = tracks->nheads + tracks->ntails;

    for (i = 0; i < tracks->ntracks; i++) {
        vdRead(in_file,&p0);
        vdRead(in_file,&p1);
        if (fscanf(in_file, "%lf %d %d",&chisqr,
            &tracks->tracks[i].nwires,&type) != 3) return(-1);

        for (j=0;j<tracks->tracks[i].nwires;j++) {
            fscanf(in_file,"%d",&k);
            tracks->tracks[i].wires[j].pn = (k-1) / 128;
            tracks->tracks[i].wires[j].wn = ((k-1)%128);
        }

        tracks->tracks[i].p0 = MMult(MCBtoNew,p0);
        tracks->tracks[i].p1 = MMult(MCBtoNew,p1);
        tracks->tracks[i].chisqr = chisqr;
        tracks->tracks[i].type = type;
    }

    return(0);
}

/*****
    Random #
*/

#define IA 16807
#define IM 2147483647L
#define AM (1.0/IM)
#define IQ 127773L
#define IR 2836
#define MASK 123459876L

long idum;

```

```

Vtype Rand(void) {
    long k;
    Vtype ans;
    idum ^= MASK;
    k=idum/IQ;
    idum=IA*(idum-k*IQ)-IR*k;
    if (idum<0) idum += IM;
    ans = AM*idum;
    idum ^=MASK;
    return ans;
}

Vtype GRand(Vtype std, Vtype mean) {
    Vtype x1,x2;
    x1 = Rand();
    x2 = Rand();
    return( std * sqrt(-2 * log(x1)) * cos( 2 * 3.1415 * x2) + mean);
}

```

B.5 bjhough.h

```

/*****
*

Bj Hough -- 2 Plane hougher
Front-Back adaptive tracker

By Erik Kangas 2-22-94

Revised 4-9-94 by Erik Kangas
Revised 11-4-94 by Erik Kangas
3rd Generation -- 11-17-94 -- Erik Kangas
Revised By Erik Kangas 1/23/95
Last Revision 5/9/95 by Erik Kangas

*/

#ifndef _BJHOUGH
#define _BJHOUGH

#include "trackdef.h"

#define VERSION "v4.01, Last Update 5/8/1995"

/*****
Tracking Parameters

MetricCut defines how close two tracks can get in metric space before
beginning grouped into the same track. This value is in probability.
default is 2.5 for heads and 5.0 for tails

ChiCut defines the poorest quality tracks accepted. This is the reduced
chi-squared. This defaults to 3.0

PRECOMPUTE flags whether or not to precompute hougher tube intercepts.
1 = yes, 0 = no. This only applies to the call to Setup_Hougher.
It defaults to 1.

ViewMem is the amount of memory for the hougher allocate for
precomputation. It defaults to 12 megabytes. This must be altered
before calling Setup_Hougher.

```

```

FPlane:
  Defines which plane to start tracking from (Ignore
  previous planes) -- defaults to Plane 0.

LPlane:
  Defines which plane to stop tracking at (Ignore later
  planes.) -- defaults to the last plane
  FPlane and LPlane can be changed during a run before calling
  Bj_hough.

MinPlanes:
  Defines the minimum number of planes the track must pass
  through to be a candidate.

MaxCombs:
  Maximum number of "potential" tracks in a tube for which we will
  fit ALL of them instead of recursing or using prediction. The
  default value is 16.

covar:
  This represents the covariance matrix used by LeastFit. The variance
  and covariance terms generated are available through this matrix
  after LeastFit has been called. For reference the order of parameters
  is taken to be (x0,y0,x1,y1) so covar[1][1] is the variance of x1 and
  covar[1][4] is the covariance of x0 and y1, etc.

*/

#define Sigma2  0.000841    /* Variance in measurement of wire */

extern Vtype  MetricCut;    /* Default metric cut is 50% */
extern Vtype  ChiCut;      /* Default Chi^2_v cut = 3.6 */
extern int    MaxCombs;    /* Max combinations to fit all */

extern int    PRECOMPUTE;  /* DO precomputation ? */
extern long   ViewMem;     /* 12Meg memory to use for precomputation */

extern long   FPlane,LPlane;
extern long   MinPlanes;

extern Vtype  cfx,cfy;     /* Center of acceptance on front plane */

extern nrctype** covar;   /* covariance matrix used in LeastFit */

extern Vtype  ICHead[4][4], /* Experimental inverse cov. matrices */
             ICTail[4][4];
extern Vtype  CTail[4][4], /* Experimental covariance matrices */
             CHead[4][4];

/*****
Function Prototypes

Setup_Hougher:
  This function must be passed the name of the detector data
  file to be used. It initializes the hougher environment, calling
  SetupCoords. It allocates memory for precomputation and calls
  MakeLSFData to precompute the least-squares fitting data. This
  function should be called before any other function in this
  module is used.
  Returns '0' if everything went OK, an error code < 0 otherwise.

Bj_hough:
  This function, named after the eminent Dr. James D. Bjorken who
  first suggested the AVHT, takes an event structure and implements
  the AVHT returning the tracks found in the tracks structure passed.
  This function will append the new tracks to any already residing
  in the tracks structure, so you may want to clear this structure

```

using "ClearTracks" before calling "Bj_Hough."

SameTrackProb:

Returns the probability that two tracks are actually different representations of the same "real" track. This is based on the differences between the parameters and their respective variances. Returns $-\log(P)$. Tracks passed must be either heads or tails.

LeastFit:

Given a track structure in which the relevant wire information is stored (see trackdef.h), a least-squares fit is performed and the best parameters are stored in the p0,p1 structures. The track is defined to pass through the points p0,p1. 'chisqr' is set to the reduced chi-squared of the fit. 'sx0,sx1,sy0 and sy1' are filled with the variances of the respective parameters.

It is assumed that there are at least four wires in the track structure. If there are exactly four, then the reduced chi-squared is set to zero and the variances are undefined.

LeastFit returns '-1' if an error occurred inverting the matrix and returns '0' otherwise.

```
*/  
  
int   Setup_Hougher(char* det_name);  
void  Bj_hough(Event*,Tracks*);  
Vtype Prob(Vtype x);  
Vtype SameTrackProb(Track* t1, Track* t2);  
int   LeastFit(Track* trk);  
  
#endif
```

B.6 bjhough.c

```
/*  
*****  
*  
*   Bj Hough  
*   Front-Back adaptive tracker  
*  
*   Tracker Source code  
*  
*   By Erik Kangas  
*   2-22-94  
*  
*   Revision -- October 1994  
*   Revision -- 11/4/94  
*   3rd Generation -- 11/17/94  
*   Parallel Tracking Code Added -- 1/11/95  
*   Parallel Code Removed, Code Revised -- 1/26/95  
*   Newest Revision 5/8/95, by Erik Kangas  
*  
*/  
  
#include "bjhough.h"  
#include "stat.h"  
#include <stdlib.h>  
#include <string.h>  
#include <math.h>  
  
/*  
*****  
*   Tracking Definitions  
*  
*/  
  
Vtype MetricCut = 2.5;      /* Current metric cut */  
Vtype ChiCut    = 3.1;     /* Chi-squared cut for tracks */
```

```

int    MaxCombs = 16;          /* Max combinations to fit all */
int    PRECOMPUTE= 1;        /* Precompute memory? */
long   FPlane,
       LPlane,
       MinPlanes;           /* Range of planes to use for current hough */
                                   /* Minimum planes that must contribute */

#ifdef DOS
long   ViewMem = 12*1048576L; /* 12 Meg memory to use for precomputation */
#else
long   ViewMem = 1024*256;    /* DOS 256k */
#endif

/*****
    Internal variables
*/

long   maxpcwires1,maxpcwires2; /* Precompute intercepts storage */
char   **pcwires1,**pcwires2;
Vtype  TubeRad,TubeDia;       /* Radius of Initial hough tube */
Vtype  SideRad,SideDia;
Vtype  vx,vy,ix,iy;          /* Hough position variables */
int    depth;                /* Hough recursion depth */
unsigned long level;         /* Precomputation index for position */
char   Inter[Planes*2];     /* Temp wire intercept list */
Vtype  cfx,cfy;              /* Center of front MWPC */
Vtype  cbx,cby;              /* Center of back MWPC */
Tracks* bj_tracks;          /* Tracks we've found */

Vtype  ICHead[4][4] = {{183.615258,-8.153881,106.488967, 2.628316},
                       {-8.153881,58.342930, 13.023081,22.161975},
                       {106.488967,13.023081,280.380811,30.543160},
                       { 2.628316,22.161975, 30.543160,37.064161}};

Vtype  CHead[4][4] = {{ 0.007168, 0.001208, -0.002905, 0.001163},
                       { 0.001208, 0.022434, -0.000033,-0.013472},
                       {-0.002905,-0.000033, 0.005105,-0.003981},
                       { 0.001163,-0.013472, -0.003981, 0.038234}};

Vtype  CTail[4][4] = {{ 0.001439, 0.003864, -0.000528,-0.002698},
                       { 0.003864, 0.092374, -0.001170,-0.062805},
                       {-0.000528,-0.001170, 0.000733, 0.002100},
                       {-0.002698,-0.062805, 0.002100, 0.106902}};

Vtype  ICTail[4][4] = {{1054.341721,-44.133503,727.029395,-13.596800},
                       {-44.133503, 19.875233,-32.083248, 11.192972},
                       {727.029395,-32.083248,1946.45039,-38.730467},
                       {-13.596800, 11.192972,-38.730467, 16.347772}};

#define Round(c) ( ((c) - floor(c)) < .5) ? floor(c) : ceil(c)

#ifdef DOS
#define UNDEF -127
#else
#define UNDEF 255
#endif

Vtype pow2(Vtype a) { return a*a; }

/*****
    MakeLSFData:

    Setup the Linear Least-squares fitting matrices.
    Use Double precision to fend-off round-off errors.
*/

nrctype fitp[Planes][4][4];    /* Fitting matrices for all the planes*/

```

```

nrctype fitv[Planes][4][2];      /* Coordinate vector fitting variable */
nrctype** covar;                 /* least squares covariance matrix */
nrctype** parms;                 /* least squares solution sets */

void MakeLSFData(void) {
    nrctype ux,uy,Z,Z1,Z2;
    int p,i,j;

    /* Initialize fitting arrays */
    /* Use z0 = PList[0].Pl.z and z1 = LeadZ for heads */
    /* Use z0 = LeadZ and x1 = PList[Planes-1].Pl.z for tails */

    for (p=0;p<Planes;p++) {
        if (p < LeadP)
            Z = (PList[p].Pl.z - PList[0].Pl.z) / (LeadZ - PList[0].Pl.z);
        else
            Z = (PList[p].Pl.z - LeadZ) / (PList[Planes-1].Pl.z - LeadZ);

        Z1 = (1-Z);
        Z2 = Z1*Z1;
        ux = PList[p].IncU.x;      /* vector perpendiculat to wires */
        uy = PList[p].IncU.y;      /* In direction of increasing wire# */

        fitp[p][0][0] = ux*ux*Z2;  /* LSF matrix independent of wire */
        fitp[p][0][1] = ux*uy*Z2;  /* For a wire hitting plane p */
        fitp[p][0][2] = ux*ux*Z1*Z; /* Matrix is symmetric so only */
        fitp[p][0][3] = ux*uy*Z1*Z; /* Compute half of it */
        fitp[p][1][1] = uy*uy*Z2;
        fitp[p][1][2] = ux*uy*Z1*Z;
        fitp[p][1][3] = uy*uy*Z1*Z;
        fitp[p][2][2] = ux*ux*Z*Z;
        fitp[p][2][3] = ux*uy*Z*Z;
        fitp[p][3][3] = uy*uy*Z*Z;

        for (i=1;i<4;i++)          /* Fill other half of matrix */
            for (j=0;j<i;j++)
                fitp[p][i][j] = fitp[p][j][i];

        fitv[p][0][0] = ux*ux*Z1;  /* Coordinate vector coeffs */
        fitv[p][0][1] = ux*uy*Z1;
        fitv[p][1][0] = ux*uy*Z1;
        fitv[p][1][1] = uy*uy*Z1;
        fitv[p][2][0] = ux*ux*Z;
        fitv[p][2][1] = ux*uy*Z;
        fitv[p][3][0] = ux*uy*Z;
        fitv[p][3][1] = uy*uy*Z;
    }
}

/*****
Least Squares fit to a set of wires in the Track.
Assumed that the wires structures in track are filled upon calling.
Assumed that there are at least 4 wire in the track.
If there are only 4, chi^2 = 0 and the variences are indeterminent.
chisqr, p0, p1, sx0,sx1,sy0,sy1 are all set.

If is assumed that only heads or tails will be fit and that ALL wires
passed will be for only for one type of track. Otherwise, the
data created in MakeLSFData does not apply.

Returns -1 if error, 0 if ok.

Note this takes multiple wires on a single plane and uses their
average wire number for a more accurate fit.
*/

int LeastFit(Track * trk) {

```

```

static int      i,j,p,w,PN[Planes],planes;
static nrctype x,y,d,Z,wc[Planes][2],WI[Planes];

for (i=1;i<=4;i++)
    memset(&covar[i][1],parms[i][1] = 0,sizeof(nrctype)*4);
memset(PN,0,sizeof(int)*Planes);
memset(WI,0,sizeof(nrctype)*Planes);

/* Count # wires from each plane contributing and get
   sum of wires from each plane so can get the mean wire later */

for (planes = w = 0; w < trk->nwires; w++) {
    if (++PN[trk->wires[w].pn] == 1) planes++;
    WI[trk->wires[w].pn] += trk->wires[w].wn;
}

/* Sum the LSF matrix and coordinate vectors for all wires */

for (p=0;p<Planes;p++) {
    if (PN[p]==0) continue;
    WI[p] /= PN[p];
    wc[p][0] = PList[p].Pl.x + PList[p].Inc.x * WI[p];
    wc[p][1] = PList[p].Pl.y + PList[p].Inc.y * WI[p];
    for(i=0;i<4;i++) {
        for (j=0;j<4;j++) covar[i+1][j+1] += fitp[p][i][j];
        parms[i+1][1] += fitv[p][i][0] * wc[p][0] +
            fitv[p][i][1] * wc[p][1];
    }
}

/* Invert matrix and get solution */
/* Save the z0,z1 used in the fit */

if (gaussj(covar,4,parms,1) < 0) return -1;

trk->p0.x = parms[1][1]; trk->p0.y = parms[2][1];
trk->p1.x = parms[3][1]; trk->p1.y = parms[4][1];

if (trk->wires[0].pn < LeadP) {
    trk->p0.z = PList[0].Pl.z;
    trk->p1.z = LeadZ;
}
else {
    trk->p0.z = LeadZ;
    trk->p1.z = PList[Planes-1].Pl.z;
}

/* Get Chi-squared value of fit */

for (trk->chisqr=p=0;p<Planes;p++) {
    if (!PN[p]) continue;
    Z = (PList[p].Pl.z - trk->p0.z) / (trk->p1.z - trk->p0.z);
    x = trk->p0.x + (trk->p1.x - trk->p0.x) * Z - wc[p][0];
    y = trk->p0.y + (trk->p1.y - trk->p0.y) * Z - wc[p][1];
    d = x * PList[p].IncU.x + y * PList[p].IncU.y;
    trk->chisqr += d*d;
}

/* Compute the reduced Chi-squared. */
/* Take the standard error in measurement into account. */

trk->chisqr /= Sigma2;
trk->chisqr = (planes <= 4) ? 0 : trk->chisqr / (planes - 4);

return 0;
}

```

```

/*****
Set up and close down tracking environment

The setup routine will return error codes. It must be
called at program initialization.

Sets up the tracking environment and reads in the detector
configuration file as well as initializing all the necessary
environment variables for use most of the functions included here.
*/

int Setup_Hougher(char* det_name) {
    int err,p;

    printf("\nHough Transform Tracker %s\n\n",VERSION);

    covar = matrix(4,4);
    parms = matrix(4,1);
    if ((err = SetupCoords(det_name)) != 0) return err;

    /* Radius of initial hougher tube */

    SideDia = PlaneWidth * 1.25;    /* A little extra big to get corners */
    SideRad = SideDia / 2;
    TubeRad = SideRad * sqrt(2);
    TubeDia = TubeRad * 2;

    cfx = (PList[0].P1.x + PList[0].P2.x + PList[0].Inc.x * 127) / 2;
    cfy = (PList[0].P1.y + PList[0].P2.y + PList[0].Inc.y * 127) / 2;
    cbx = (PList[23].P1.x + PList[23].P2.x + PList[23].Inc.x * 127) / 2;
    cby = (PList[23].P1.y + PList[23].P2.y + PList[23].Inc.y * 127) / 2;

    /* Setup Least Squares Fitter for finding tracks with this detector
configuration. */

    MakeLSFData();

    /* Allocate memory for storing Wire Intercept information */
    /* One set for before the lead, one set for after */

    maxpcwires1 = (PRECOMPUTE) ?
        0.33 * ViewMem / (sizeof(char)*2* LeadP + sizeof(char*)) : 0;
    maxpcwires2 = (PRECOMPUTE) ?
        0.67 * ViewMem / (sizeof(char)*2*(Planes-LeadP) + sizeof(char*)) : 0;

    if (maxpcwires2) {
        pcwires1 = (char**) malloc(sizeof(char*)*maxpcwires1);
        pcwires2 = (char**) malloc(sizeof(char*)*maxpcwires2);
        for (p=0;p<maxpcwires1;p++) {
            pcwires1[p] = (char*) malloc(sizeof(char)*2*LeadP);
            pcwires1[p][0] = UNDEF;
        }
        for (p=0;p<maxpcwires2;p++) {
            pcwires2[p] = (char*) malloc(sizeof(char)*2*(Planes - LeadP));
            pcwires2[p][0] = UNDEF;
        }
    }

    return 0;
}

/*****
*
GetIntercepts:

Compute intercept list for Hougher tube defined by the
depth,vx,vy,ix,iy hougher parameters.

```

```

vx,vy,ix,iy specify the fractional offset (0-1) into Hougher
viewing and imaging planes. Eg. the lower left corner of the
viewing plane is specified by the x,y pair

    (cfx -TubeRad + TubeDia*vx, cvy -TubeRad + Tubedia*vy)

The viewing plane is at PList[FPlane].Pl.z and the imaging plane
is at PList[LPlane].Pl.z.

depth specifies the size of the square. The side length of both
squares is given by:

    TubeDia / (1 << depth)

wires must be of form: char wires[Planes*2]

Fills 0 -> LPlane-FPlane+1 entries of wires with the range of wires
intercepted by the tube. Eg. wires[0] gives the lower bound
and wires[1] gives the upper bound in wire # space (counting
from 0) of wires from plane FPlane intersecting the tube.
*/

Vtype Bound(Vtype x) {
    x = Round(x);
    return (x < 0) ? 0 : ( (x>127) ? 127 : x );
}

char* GetIntercepts(char* wires) {
    static Vtype vcx,vcy; /* coordiante of centriod */
    static Vtype Z,fpz;
    static Vtype r,w,dcx,dcy,pcx,pcy;
    static int m,m2,diff;

    /* Need increment "vector" from corner of the square on the front to */
    /* The respective corner of the square on the back. */
    /* For 'x' this is given by... */
    /* (delta x) * TubeDia / (distance in z) */

    Z = 1.0 / (PList[LPlane].Pl.z - (fpz = PList[FPlane].Pl.z));
    dcx = Z * ((ix - vx) * SideDia + cbx-cfx);
    dcy = Z * ((iy - vy) * SideDia + cby-cfy);

    /* Currently assuming hough areas are SQUARES of same size */
    /* in front and back projctive planes */
    /* Get distance from edge to center of hough square */
    /* Get the absolute coords of the centers of each square */
    /* Get the radius of circumscribed circle in wires */

    r = SideRad/(1L<<depth); /* 1/2 length of side of the square */
    vcx = cfx-SideRad + vx*SideDia + r; /* Center on front plane x */
    vcy = cfy-SideRad + vy*SideDia + r; /* Center on front plane y */
    r = 15*r; /* Radius in wires (r*1.5/WireSpace)*/

    /* get wires that project into this square for given planes */
    /* Center of the box is the center of the viewing box + the */
    /* differential center along the connecting line. Then subtract */
    /* out the coord of the 1st wire. */
    /* If w is one boundary between wires, then r may get VARY small before */
    /* giving the wire range as 1 wire. So, if r<0.33 wires, we MAKE the */
    /* range only 1 wire and use the nearest wire -- preventing undo */
    /* recursion. */

    diff = (r > 0.33) ? 1 : 0;

    for (m = FPlane, m2 = 0; m <= LPlane; m++, m2+=2) {
        Z = PList[m].Pl.z - fpz;

```

```

    pcx = vcx + dcx * Z - PList[m].Pl.x;
    pcy = vcy + dcy * Z - PList[m].Pl.y;
    w = (pcx * PList[m].IncU.x + pcy * PList[m].IncU.y)/WireSpace;
    if (diff) {
        wires[m2] = Bound(w - r);
        wires[m2+1] = Bound(w + r);
    }
    else wires[m2] = wires[m2+1] = Bound(w);
}

return wires;
}

/*****
Prob:
    Returns the Probability that something is Sqrt[x] standard deviations
    or further from the mean of a gaussian. This is an approximation
    of the integral.

Metric:
    Returns the "probability" that 2 tracks are the same (0-1). Based
    upon their separation in each parameter and the standard errors in
    all the parameters. Returns -log(probability)
*/

Vtype Prob(Vtype y) {
    Vtype x = sqrt(y);
    return ( x < 0.9657 ) ? 1.0 - x * (0.797885 - 0.132981 * y) :
        0.531923 * exp( -0.5 * y ) / x;
}

Vtype SameTrackProb(Track* t1, Track* t2) {
    static Vtype y,v[4],x;
    static int i,j;

    /* Compute the Deviation vector between the tracks */

    v[0] = t1->p0.x - t2->p0.x;
    v[1] = t1->p0.y - t2->p0.y;
    v[2] = t1->p1.x - t2->p1.x;
    v[3] = t1->p1.y - t2->p1.y;

    /* Now determine the # of standard deviations^2 away this vector is */
    /* Variance = v . C^(-1) . v */

    if (t1->type == ttHead)
        for (i=y=0;i<4;i++) {
            for(x=j=0;j<4;j++) x += ICHead[i][j] * v[j];
            y += v[i] * x;
        }
    else
        for (i=y=0;i<4;i++) {
            for(x=j=0;j<4;j++) x += ICTail[i][j] * v[j];
            y += v[i] * x;
        }

    y /= 4.0; /* This is from propagation of errors w/ same cov. matrix */
            /* The extra factor of two is from the final formula */

    /* y = number of standard deviations away squared / 2. */
    /* p = exp(-y) * (1 + y); */
    /* return -log(p) */

    return y - log(1+y);
}

```

```

/*****
  AddTrack:
    Add a given track to the track found list
*/

void AddTrack(Track* track) {
  static int n;

  track->type = (FPlane == 0) ? ttHead : ttTail;

  /* See if there are any tracks close to this guy already found... */
  /* If so, keep the guy w/ the best chi-squared */
  /* Don't use DeleteTrack, use this faster implicit scheme */

  for (n=bj_tracks->ntracks-1; n>=0; n--)
    if (SameTrackProb(track,&bj_tracks->tracks[n]) <= MetricCut) {
      if (track->chisqr >= bj_tracks->tracks[n].chisqr) return;
      if (n < bj_tracks->ntracks-1)
        bj_tracks->tracks[n] = bj_tracks->tracks[bj_tracks->ntracks-1];
      bj_tracks->ntracks--;
    }

  /* Add track to list of tracks */
  /* If it doesn't fit in the list, ignore it */

  if (bj_tracks->ntracks < MAXTRACKS)
    bj_tracks->tracks[bj_tracks->ntracks++] = *track;
}

/*****
  TryCombs:
    Try all combinations of tracks in the event tube and add appropriate
    ones to list.
*/

void TryCombs(Event* event, char* wires,unsigned long com,Track* track) {
  static int i,j,k,n,index[Planes],basew,plane[Planes],planes,p,q;

  /* Setup */
  /* 'Stub' contains the part that won't change b/n combinations */
  /* Only keep track of planes that have more than 1 wire contributing */
  /* So we don't waste time looping over the other guys */

  memset(index,0,sizeof(int)*Planes);
  basew = track->nwires;
  planes = 0;

  for (p = FPlane; p <= LPlane; p++)
    if (event->nw[p] > 1) plane[planes++] = p;

  /* Try all combinations */

  for (i=0;i<com;i++) {

    /* Fill the track with current set of wires */

    track->nwires = basew;

    for (j=0;j<planes;j++) {
      p = plane[j];
      q = 2*(p - FPlane);
      n = 0;
      for (k=wires[q]; k<=wires[q+1]; k++)
        if (event->w[p][k]) {
          if (n<index[p]) { n++; continue; };
          track->wires[track->nwires].pn = p;
        }
    }
  }
}

```

```

        track->wires[track->nwires++].wn = k;
        break;
    }
}

/* Fit this set and see if it could be a track */
/* If so, add it to the list of tracks */

if (!LeastFit(track) && (track->chisqr <= ChiCut)) AddTrack(track);

/* Setup for next combination */

for (j=0;j<planes;j++) {
    p = plane[j];
    if (++index[p] == event->nw[p]) index[p] = 0;
    else break;
}
}

}

/*****
IsTrack:
    Given a hough tube, determine # of tracks in tube and add to
    bj_hough list. Return # found or -1 if indeterminate -- eg.
    need to recurse further.
*/

int offset[5] = {0,1,-1,2,-2};          /* Wire offset lookup table */

int IsTrack(Event* event, char* wires) {
    static int a,m,planes,n,nw,w;
    static unsigned long com;
    static Track track;
    static Vtype x,y,z,r;

    com = 1;
    a = track.nwires = planes = 0;

    for (m=FPlane; m<=LPlane; m++,a+=2) {

        /* Construct a "short" track out of wires from planes */
        /* That are only contributing 1 wire to the tube */
        /* Count # of contributing planes */
        /* Count # of wires in this track */
        /* Count # of possible tracks in tube */

        event->nw[m] = nw = 0;

        for (n = wires[a]; n <= wires[a+1]; n++)
            if (event->w[m][n] && (++event->nw[m] == 1)) {
                track.wires[track.nwires].wn = n;
                track.wires[track.nwires].pn = m;
            }

        if ((nw = event->nw[m]) > 0) {
            if (nw==1) track.nwires++;
            if (com < 1000) com *= nw;
            planes++;
        }
    }

    if (planes < MinPlanes) return 0;          /* Failed MinPlanes cut! */

    /* Check number of possible tracks -- */

```

```

/* if it is small enough, try them all */
/* say that we know about this tube */

if (com <= MaxCombs) {
    TryCombs(event,wires,com,&track);
    return 0;
}

/* Otherwise, use the track stub and generate a trail track */
/* Need >=4 wires so have valid variance info from the LSF */
/* Check the variance terms in the fit to see if the fit was */
/* "good enough" to possibly predict a track */

if (track.nwires < 4)                return -1; /* Not enough for a fit! */
if (LeastFit(&track))                return -1; /* Degenerate Fit! */
if (track.chisqr > ChiCut) {         /* Not a good fit */
    if (planes == MinPlanes) return 0;    /* Can't be a track */
    return -1;
}

/* Take the track stub and look on the other planes and get the */
/* closest wire -- if it is fairly close */

planes = track.nwires;                /* Planes contributing to this track */

for (m=FPlane;m<=LPlane;m++) {
    if (event->nw[m] < 2) continue;
    z = (PList[m].Pl.z - track.p0.z) / (track.pl.z - track.p0.z);
    x = track.p0.x + (track.pl.x-track.p0.x) * z - PList[m].Pl.x;
    y = track.p0.y + (track.pl.y-track.p0.y) * z - PList[m].Pl.y;
    w = Bound(r = (PList[m].IncU.x * x + PList[m].IncU.y * y)/WireSpace);

    /* For the case of passing b/n adjacent active wires, include */
    /* both wires in the fit */

    a = (int)r;                        /* Rounded down wire # */

    if ((a>=0) && (a<127) && event->w[m][a] && event->w[m][a+1]) {
        track.wires[track.nwires].pn = m;
        track.wires[track.nwires++].wn = a;
        track.wires[track.nwires].pn = m;
        track.wires[track.nwires++].wn = a+1;
        planes++;
        continue;
    }

    /* Get Closest wire (w/i 2 wire spaces) */

    for (n=0;n<5;n++) {
        a = w + offset[n];              /* Look at 5 nearest wires */
        if ((a < 0) || (a > 127)) continue;
        if (event->w[m][a]) {
            track.wires[track.nwires].pn = m;
            track.wires[track.nwires++].wn = a;
            planes++;
            break;
        }
    }
}

if (planes < MinPlanes)    return -1;    /* Fail MinPlanes cut */
if (LeastFit(&track))     return -1;    /* Degenerate fit */
if (track.chisqr > ChiCut) return 0;    /* No Track here */
AddTrack(&track);
return 0;
}

```

```

/*****
Recursive bj_hough routine for finding tracks from all viewpoints

Must have called Setup_Hougher() before calling Bj_hough.
Inputs:
    event -> event list holding wires hit
    bj_tracks -> list of tracks found is returned in this.
*/

void Bj_hough_R(Event* event) {
    static char* wires;
    Vtype ovx,ovy,oix,oiy,delt; /* Old position in Hough space */
    int i; /* indexing variables */

    /* Setup Local Hougher */

    ovx = vx; ovy = vy; /* Remember beginning of Hough area */
    oix = ix; oiy = iy;
    delt = 1.0 / (1L << depth); /* Fractional width of sub-areas */

    /* Loop through all 16 possible tubes */

    for (i=0;i<16;i++) {
        level++;
        vx = ovx + (i&1) *delt; /* Get Corners of current area */
        vy = ovy + ((i&2)==2)*delt; /* In fractional Hough space */
        ix = oix + ((i&4)==4)*delt; /* Need this for GetIntercepts */
        iy = oiy + ((i&8)==8)*delt;

        /* Get wire intercept list */

        if (FPlane==0) {
            if (level < maxpcwires1) {
                if (pcwires1[level][0] != UNDEF) wires = pcwires1[level];
                else wires = GetIntercepts(pcwires1[level]);
            }
            else wires = GetIntercepts(Inter);
        }
        else {
            if (level < maxpcwires2) {
                if (pcwires2[level][0] != UNDEF) wires = pcwires2[level];
                else wires = GetIntercepts(pcwires2[level]);
            }
            else wires = GetIntercepts(Inter);
        }

        /* If IsTrack return -1, it means that the tube is too
        dirty to track and thus we must recurse still more */

        if (IsTrack(event,wires) == -1) {
            depth++; /* New depth */
            level *= 17; /* Set new level */
            Bj_hough_R(event); /* Recurse */
            depth--; /* reset for current depth */
            level /= 17;
        }
    } /* End of For Loop */
}

void Bj_hough(Event* evt, Tracks* trks) {

    /* Set Up Hough Environment */
    /* Initialize bjhougher's recursive variables */
    /* Set the track structure to use as global */
    /* Get tracks */

```

```

bj_tracks = trks;
vx=vy=ix=iy=level=0;
depth = 1;
Bj_hough_R(evt);
}

```

B.7 coords.h

```

/*****
Minimax Hough Tracker
Coordinate System and Linear Algebra Definitions

By Erik Kangas
Case Western Reserve University
2-24-94

Last Revised 5/1/95

*/

#ifndef COORDS
#define COORDS

#include <stdio.h>
#include "nrc.h"

/*****
Linear Algebra functions

    These functions DO NOT modify the arguments passed; you must assign
    the result to a new vector, when appropriate. Use the vdVector function
    to create vectors from doubles.

Vtype is the data type used in the vectors.
    Note: if you change this (e.g. to float), you will need to change all
    printf and scanf functions accordingly!

Most functions are self explanatory.

vdDistance(x1,y1,x2,y2)
    Gives the distance between the s lines at their closest approach.
    The lines are defined as:
        x1 + y1 s and x2 + y2 s
    where x1 and x2 are offset vectors, y1 and y2 are unit vectors in the
    direction of the respective lines and s is a parameterization of the
    line.

vdMidpoint(x1,y1,x2,y2)
    Same parameters as vdDistance. Returns vector pointing to the
    midpoint of the line connecting the two input lines at their
    distance of closest approach -- Eg. a 1st approximation vertex.

vd2DDist(point,line,dir)
    Returns the distance between point and line where dir defines the
    direction of positive distance.

MMult:
    multiplies a NRC matrix m[1..3][1..3] by a vector and returns the
    resulting vector

vdPrint:
    prints a vector to a file as 3 floating point numbers separated by
    a space with a single trailing space.

```

```

vdRead:
    reads the vector back in.

Invert2x2:
    takes a matrix for the form nrctype m[2][2] and inverts it. Note:
    this is a different format than that given by the "matrix" function
    in nrc.c.
*/

typedef double Vtype;          /* Data type to use for vectors */

typedef struct {              /* Vector structure for doubles */
    Vtype x,y,z;              /* 3-vector */
} dVector;

Vtype  vdDot(dVector a, dVector b);          /* a . b */
dVector vdCross(dVector a, dVector b);      /* a x b */
dVector vdSubtract(dVector a, dVector b);   /* a - b */
dVector vdAdd(dVector a, dVector b);        /* a + b */
dVector vdMultiply(dVector a, Vtype m);     /* a * m */
dVector vdDivide(dVector a, Vtype m);       /* a / m */
dVector vdVector(Vtype x, Vtype y, Vtype z); /* (x,y,z) */
Vtype  vdMagnitude(dVector a);              /* |a| */
dVector vdNormalize(dVector a);             /* a / |a| */
int     vdEqual(dVector a, dVector b);      /* a == b */

Vtype  vdDistance(dVector x1, dVector y1, dVector x2, dVector y2);
dVector vdMidpoint(dVector x1, dVector y1, dVector x2, dVector y2);
Vtype  vd2DDist(dVector point, dVector line, dVector dir);

dVector MMult(nrctype** m, dVector v);      /* m . v */
void    vdPrint(FILE* f, dVector v);
void    vdRead(FILE* f, dVector *v);
void    Invert2x2(nrctype m[2][2]);        /* m^(-1) */

/*****
/* General Detector Constants */

#define WireSpace  0.1    /* Space in inches b/n wires */
#define PlaneWidth 12.6  /* Effective width of plane in inches */
#define LeadP      8     /* Plane after the Pb (Counting from 0) */
#define SLeadZ    184.8  /* Z of Lead in inches */
#define Planes    24     /* Number of planes in detector */

/*****
Detector plane data structure

Coordinate units : inches

Coordinate system:
    standard tracking (not accellerator)
    y - up, x - points from the pipe to the hall,
    z - points from C0 to the detector.
    Origin at C0.

hougher coordinates:
    origin at C0
    y parallel to wires in the "parallel" planes
    x = y (X) z

Use MCB... to change from one system to the other.

P1,P2 - x,y,z vector coordinates of the end points of the first
active wire in the plane.
Inc - vector in direction perpendicular to the wires and of length
WireSpace.

```

```

    IncU - vector parallel to 'Inc' but of unit length.
    Unit - unit vector parallel to the wires.

*/

typedef struct PData {          /* Detector plane data structure */
    dVector P1,P2;
    dVector Inc,Unit;
    dVector IncU;
    int parallel;
} PlaneData;

/* Global Variables setup by SetupCoords */

extern PlaneData PList[Planes]; /* Detector info in hougher coords */
extern PlaneData SPList[Planes]; /* Detector info in standard coords */
extern dVector C0; /* C0 in hougher coordaintes */
extern dVector Beam; /* Unit vector in beam direction (hough) */
extern nrctype LeadZ; /* Z of lead in hougher coords */
extern nrctype** MCBtoNew; /* matrix of change of basis from hough */
extern nrctype** MCBtoOld; /* To standard system and back */

/*****

Function Definitions:

SetupCoords(char* name) :
    Pass the name of the detector configuration file.
    Reads in the configuration file and analyzes the planes.
    Sets up the New Hougher Coordinate system.
        This configuration should go a long way towards diagnolizing the
        covariance matrix in the LSF of BjHough
    Sets up other global constants.
    returns 0 = Ok, -1 = Error

The Format of the detector configuration file is the following:

f11 f12 f13 f14 p1
....
fn1 fn2 fn3 fn4 pn

Where f represents floating point numbers.
There must be 'Planes' enteries in this file.
This file corresponds DIRECTLY to Tom Jenkin's Alignment report format
with the p(i) integers appended to each line. They should be '1' if the
plane is a "parallel" plane and 0 otherwise.

*/

int SetupCoords(char* name);

#endif

```

B.8 coords.c

```

/*****
Minimax Hougher Tracker
Coordinate System Definitions
And definitions of vector manipulation functions

By Erik Kangas
Case Western Reserve University

```

2-24-94

Last Revised 5/1/95

```
*/

#include "coords.h"
#include <math.h>

/* Global Variables */

PlaneData PList[Planes];      /* Detector info in hougher coords      */
PlaneData SPList[Planes];     /* Detector info in standard coords */
dVector   C0;                 /* C0 in hougher coordaintes */
dVector   Beam;               /* Unit vector of beam direction */
nrctype** MCBtoNew;           /* matrix of change of basis from hougher */
nrctype** MCBtoOld;           /* To standard system and back */
nrctype   LeadZ;              /* Z pos of lead in new coords */

/*****
/* Function declartions */

int SetupCoords(char* fname) {
    dVector e1,e2,e3;
    Vtype x_0,y_0,z_0,theta,Cos,Sin;
    FILE* file;
    int p,par;

    if ((file=fopen(fname,"r")) == NULL) return (-1);

    /* Read in data for each detector plane */
    /* angle or rotation and x,y,z of plane in Tom Jenkin's format */
    /* Store information in SPList -- the standard tracking coordinate */
    /* system, in case others want access to it. */

    for (p=0;p<Planes;p++) {
        if (5 != fscanf(file, "%lf %lf %lf %lf %d", &theta,
            &x_0, &y_0, &z_0, &par)) return(-1);

        /* Get corners of wire 1, increment vectors to the next wires */
        /* And also unit vectors along the wire */

        SPList[p].parallel = par;
        SPList[p].IncU      = vdVector(cos(theta),sin(theta),0);
        SPList[p].Inc       = vdMultiply(SPList[p].IncU,WireSpace);

        Cos = sin(theta);      /* angle along the wire */
        Sin = -cos(theta);     /* Tom's angle referenced different than mine*/

        SPList[p].P1 = vdVector(x_0 - Cos * PlaneWidth/2 + SPList[p].Inc.x,
            y_0 - Sin * PlaneWidth/2 + SPList[p].Inc.y,
            z_0);
        SPList[p].P2 = vdAdd(SPList[p].P1,vdVector(PlaneWidth*Cos,
            PlaneWidth*Sin,0));
        SPList[p].Unit = vdVector(Cos,Sin,0);
    }
    fclose(file);

    /* Now, We have all the info we need in Std. coordinates, let's build
    the new coordinate system. */
    /* MCBtoOld is given by the normalized basis vectors of the new coords */
    /* Z unit vector along same z axis so that the MWPC planes are still
    in x-y plane */

    MCBtoNew = matrix(3,3);
    MCBtoOld = matrix(3,3);
}
```

```

e3      = vdVector(0,0,1);

/* Now the hard part is finding the average parallel wire direction */
/* Y is parallel to the wires in the parallel planes, X = Y (x) Z  */
/* This config. should mostly diagonalize the covariance matrix of LSF */

e2 = vdVector(0,0,0);
for (p=par=0;p<Planes;p++)
    if (SPList[p].parallel) {
        par++;
        e2 = vdAdd(e2,SPList[p].Unit);
    }
e2 = vdNormalize(vdDivide(e2,par));
e1 = vdCross(e2,e3);

MCBtoOld[1][1] = e1.x; MCBtoOld[2][1] = e2.x; MCBtoOld[3][1] = e3.x;
MCBtoOld[1][2] = e1.y; MCBtoOld[2][2] = e2.y; MCBtoOld[3][2] = e3.y;
MCBtoOld[1][3] = e1.z; MCBtoOld[2][3] = e2.z; MCBtoOld[3][3] = e3.z;
MCBtoNew[1][1] = e1.x; MCBtoNew[2][1] = e2.x; MCBtoNew[3][1] = e3.x;
MCBtoNew[1][2] = e1.y; MCBtoNew[2][2] = e2.y; MCBtoNew[3][2] = e3.y;
MCBtoNew[1][3] = e1.z; MCBtoNew[2][3] = e2.z; MCBtoNew[3][3] = e3.z;

gaussj(MCBtoNew,3,NULL,0);

/* Now, Convert Plane info in the new system! */

for (p=0;p<Planes;p++) {
    PList[p].parallel = SPList[p].parallel;
    PList[p].P1 = MMult(MCBtoNew, SPList[p].P1);
    PList[p].P2 = MMult(MCBtoNew, SPList[p].P2);
    PList[p].Inc = MMult(MCBtoNew, SPList[p].Inc);
    PList[p].IncU= MMult(MCBtoNew, SPList[p].IncU);
    PList[p].Unit= MMult(MCBtoNew, SPList[p].Unit);
}

/* Misc constants into new system */

LeadZ = MMult(MCBtoNew, vdVector(0,0,SLeadZ)).z;
Beam   = MMult(MCBtoNew, vdVector(0,0,1));
C0     = vdVector(0,0,0);
return 0;
}

/*****
Vector and Matrix Manipulation Functions
*/

Vtype vdDot(dVector a, dVector b) {          /* Dot-product */
    return a.x*b.x + a.y*b.y + a.z*b.z;
}

dVector vdCross(dVector a, dVector b) {      /* Cross product */
    dVector t;                               /* axb */
    t.x = (a.y*b.z - a.z*b.y);
    t.y = (b.x*a.z - a.x*b.z);
    t.z = (a.x*b.y - b.x*a.y);
    return(t);
}

dVector vdSubtract(dVector a, dVector b) {   /* Subtract two dVectors */
    a.x -= b.x;
    a.y -= b.y;
    a.z -= b.z;
    return(a);
}

```

```

dVector vdAdd(dVector a, dVector b) {          /* Add two dVectors */
    a.x += b.x;
    a.y += b.y;
    a.z += b.z;
    return(a);
}

dVector vdMultiply(dVector a, Vtype m) {      /* Vector * scalar */
    a.x *= m;
    a.y *= m;
    a.z *= m;
    return(a);
}

dVector vdDivide(dVector a, Vtype m) {       /* Vector / scalar */
    if (m==0) return(a);
    a.x /= m;
    a.y /= m;
    a.z /= m;
    return(a);
}

dVector vdVector(Vtype x, Vtype y, Vtype z) { /* Create a Vector */
    dVector t;
    t.x= x;
    t.y= y;
    t.z= z;
    return(t);
}

void vdPrint(FILE* f, dVector v) {
    fprintf(f,"%9.4lf %9.4lf %9.4lf ",v.x,v.y,v.z);
}

void vdRead(FILE* f, dVector *v) {
    fscanf(f,"%lf %lf %lf",&v->x,&v->y,&v->z);
}

Vtype vdMagnitude(dVector a) {               /* Magnitude of a dVector */
    return( sqrt( vdDot(a,a)));              /* ||a|| */
}

dVector vdNormalize(dVector a) {             /* a = a/||a|| */
    return( vdDivide(a,vdMagnitude(a)));
}

int vdEqual(dVector a, dVector b) {         /* Equality? a == b */
    return( (a.x==b.x)&&(a.y==b.y)&&(a.z==b.z) );
}

Vtype vdDistance(dVector x1, dVector y1, dVector x2, dVector y2) {
    static dVector x3;
    static Vtype s,t,u;

    x3 = vdSubtract(x2,x1);
    u = vdDot(y1,y2) * vdDot(y1,y2) - 1;
    if (fabs(u) < 0.0000000001) return -1;
    s = ( vdDot(y2,x3) - vdDot(y1,y2)*vdDot(y1,x3) )/u;
    t = ( vdDot(y1,y2)*vdDot(y2,x3) - vdDot(y1,x3) )/u;
    return vdMagnitude( vdSubtract( vdSubtract( vdMultiply(y1,t),x3),
        vdMultiply(y2,s)));
}

dVector vdMidpoint(dVector x1, dVector y1, dVector x2, dVector y2) {
    dVector x3,l1,l2;
    Vtype s,t,u;
    x3 = vdSubtract(x2,x1);
}

```

```

    u = vdDot(y1,y2)*vdDot(y1,y2) - 1;
    s = ( vdDot(y2,x3) - vdDot(y1,y2)*vdDot(y1,x3) )/u;
    t = ( vdDot(y1,y2)*vdDot(y2,x3) - vdDot(y1,x3) )/u;
    l1 = vdAdd(x1, vdMultiply(y1,t));
    l2 = vdAdd(x2, vdMultiply(y2,s));
    return vdAdd(l2, vdMultiply( vdSubtract(l1,l2), 0.5 ) );
}

Vtype vd2DDist(dVector point, dVector line, dVector dir) {
    int d;
    d = (vdCross(line,dir).z >= 0) ? 1 : -1;    /* direction of positive */
    return d * vdCross(line,point).z;
}

dVector MMult(nrctype** m, dVector v) {
    dVector out = vdVector(0,0,0);
    int i,j;
    for (i=0;i<3;i++) for (j=0;j<3;j++)
        (&out.x)[i] += m[i+1][j+1] * (&v.x)[j];
    return out;
}

void Invert2x2(nrctype m[2][2]) {
    nrctype det,x;

    det = m[0][0] * m[1][1] - m[0][1] * m[1][0];
    x = m[1][1];
    m[0][1] = - m[0][1] / det;
    m[1][0] = - m[1][0] / det;
    m[0][0] = m[1][1] / det;
    m[1][1] = x/det;
}

```

B.9 nrc.h

```

/*****
    Modified Numerical Recipies in C header file

    Implemented by Matt Knepley and Erik Kangas
    For use with CWRU Minimax Tracking Software

    Last updated 4/2/95
*/

#ifndef _NR_UTILS_H_
#define _NR_UTILS_H_

/*
 * Numerical Recipies General Defines
 * Modify nrctype to the floating point precision desired.
 */

#define NR_END      1
#define FREE_ARG    char*
#define SWAP(a, b)  {temp = (a); (a) = (b); (b) = temp;}
#define POW2(a)     ((a)*(a))

typedef double nrctype;    /* Standard data type */

/*
 * Numerical Recipies Function Prototypes
 * See the book for descriptions
 */

```

```

double  *dvector(long nl, long nh);
int      *ivector(long nl, long nh);
nrctype **matrix(long nrh, long nch);
void     free_dvector(nrctype *v, long nl, long nh);
void     free_ivector(int *v, long nl, long nh);
void     free_matrix(nrctype **m, long nrh);
int      gaussj(nrctype **a, int n, nrctype **b, int m);

```

```
#endif
```

B.10 nrc.c

```

/*****
  Modified Numerical Recipes in C code file

  Implemented by Matt Knepley and Erik Kangas
  For use with CWRU Minimax Tracking Software

  Last updated 4/2/95
*/

#include "nrc.h"
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

nrctype *dvector(long nl, long nh) {
    nrctype *v;
    v = (nrctype *) malloc((size_t) ((nh-nl+1+NR_END)*sizeof(nrctype)));
    if (!v) {
        printf("Memory allocation failure in dvector()\n");
        exit(1);
    }
    return v-nl+NR_END;
}

int *ivector(long nl, long nh) {
    int *v;
    v = (int *) malloc((size_t) ((nh-nl+1+NR_END)*sizeof(int)));
    if (!v) {
        printf("Memory allocation failure in ivector()\n");
        exit(1);
    }
    return v-nl+NR_END;
}

nrctype **matrix(long nrh, long nch) {
    long i;
    nrctype **m;

    m = (nrctype **) malloc((size_t) (nrh*sizeof(nrctype *))) - 1;
    if (!m) {
        printf("Memory allocation failure 1 in matrix()\n");
        exit(1);
    }
    for (i=1; i<=nrh; i++) {
        m[i] = (nrctype *) malloc((size_t) (nch*sizeof(nrctype))) - 1;
        if (!m[i]) {
            printf("Memory allocation failure 2 in matrix()\n");
            exit(1);
        }
    }
    return m;
}

```

```

#pragma argsused
void free_dvector(nrctype *v, long nl, long nh) {
    free((FREE_ARG) (v + nl - NR_END));
}

#pragma argsused
void free_ivector(int *v, long nl, long nh) {
    free((FREE_ARG) (v + nl - NR_END));
}

void free_matrix(nrctype **m, long nrh) {
    int i;
    for (i=1;i<=nrh;i++) free(m[i]+1);
    free(m+1);
}

/* Numerical Recipies in 'C', second edition */
/* Modified slightly to return an error code if there is */
/* a singular matrix */

int gaussj(nrctype **a, int n, nrctype **b, int m) {
    int *indx,*indxr,*ipiv;
    int i,icol,irow,j,k,l,ll;
    nrctype big,dum,pivinv,temp;

    indx=ivector(1,n);
    indxr=ivector(1,n);
    ipiv=ivector(1,n);
    for (j=1;j<=n;j++) ipiv[j]=0;
    for (i=1;i<=n;i++) {
        big=0.0;
        for (j=1;j<=n;j++)
            if (ipiv[j] != 1)
                for (k=1;k<=n;k++) {
                    if (ipiv[k] == 0) {
                        if (fabs(a[j][k]) >= big) {
                            big=fabs(a[j][k]);
                            irow=j;
                            icol=k;
                        }
                    } else if (ipiv[k] > 1) {
                        printf("gaussj: Singular Matrix-1\n");
                        return -1;
                    }
                }
        ++(ipiv[icol]);
        if (irow != icol) {
            for (l=1;l<=n;l++) SWAP(a[irow][l],a[icol][l])
            for (l=1;l<=m;l++) SWAP(b[irow][l],b[icol][l])
        }
        indxr[i]=irow;
        indx[i]=icol;
        if (a[icol][icol] == 0.0) {
            printf("gaussj: Singular Matrix-2\n");
            return -1;
        }
        pivinv=1.0/a[icol][icol];
        a[icol][icol]=1.0;
        for (l=1;l<=n;l++) a[icol][l] *= pivinv;
        for (l=1;l<=m;l++) b[icol][l] *= pivinv;
        for (ll=1;ll<=n;ll++)
            if (ll != icol) {
                dum=a[ll][icol];
                a[ll][icol]=0.0;
                for (l=1;l<=n;l++) a[ll][l] -= a[icol][l]*dum;
            }
    }
}

```

```

        for (l=1;l<=m;l++) b[l][l] -= b[icol][l]*dum;
    }
}
for (l=n;l>=1;l--) {
    if (indxr[l] != indxc[l])
        for (k=1;k<=n;k++)
            SWAP(a[k][indxr[l]],a[k][indxc[l]]);
}
free_ivector(ipiv,1,n);
free_ivector(indxr,1,n);
free_ivector(indxc,1,n);
return 0;
}

```

B.11 stat.h

```

/*****
CWRU MINIMAX TRACKING STATISTICS FUNCTIONS

By Erik Kangas

Last Revised 5/10/95
*****/

#ifndef _STAT
#define _STAT

#include <stdio.h>
#include "trackdef.h"

/*****
/* Global Statistics Definitions with default values given in comments */
/*                               */
/* Special data types */

#define hNoFlags      0
#define hOverflow     1
#define hUnderflow   2

typedef double htype;      /* If you change this, change all i/o functs too
*/

typedef struct {          /* Hisotgram data structure */
    int bins;            /* # of bins in 1D */
    int flags;          /* Over/under flow flags, etc. */
    int sets;          /* # of data sets used */
    htype low,high;    /* bounds of histogram */
    htype inc;         /* numerical increment b/n bins */
    htype** hist;      /* data */
    htype *ovrfl,*undfl; /* Special bins */
} Histogram;

/*****
Statistics Function prototypes...

The Statistics Library was designed so it would be easy to create
and manipulate 1D Histograms with arbitrary bin sizes and ranges.
Functions allow dynamic creation and deletion of these objects as well
as automatic binning and writing to a file (in Standard format readable
by gnuplot, origin, excel, etc.).

```

Typical usage is to use CreateHist to define a histogram, Bin to build the histogram up and WriteHist to output the results.

OutFileBase:

Sets the base name 'out_name' to the given string. All output files will be opened with 'out_name' + extension as pre OutFile. Eg. Using the 'WriteHist' function will output using 'out_name' as the base name and 'ext' as the extension using a string concatenation procedure. The default value of 'out_name' is 'stat'.

OutFile:

Opens a file with the name out_name + ext. does not error check the file*.

CreateHist:

Creates a histogram with "bins" bins from low to high. It zeros all the elements in the histogram. Does not check that lo < hi. You must specify and special types for the histogram as named in the above defines. You can specify more than one data set. In this case, you can bin to whichever set you wish and all sets will be written out by WriteHist.

DeleteHist:

Frees up the memory used by a histogram. You should call DeleteHist when you are done w/ the histogram.

Bin:

Bins the value into the histogram -- incrementing the appropriate bin by 1. The values passed will be range checked and ignored if out of range (or put into the appropriate over/underflow bin). Any number >= high or < low will NOT be binned. Note the equals sign in the high. You must the data set number to use (default should be set=0 for the first or only set).

WriteHist:

Writes out a histogram in "bin\tcounts" format where the bin is identified with the low value of the current bin. All data sets are written out at once.

hMean:

Returns the "mean" of the histogram where each bin is defined as the value of it's midpoint. Does not include the over/underflow bins.

hStdDev:

Computes the standard error in the hMean.

hElements:

Returns the number of times something has been binned -- not including the over/underflow bins

*/

```
void OutFileBase(char* name);
FILE* OutFile(char* ext);
```

```
Histogram CreateHist(int bins, htype low, htype high, int flags, int sets);
void DeleteHist(Histogram* hist);
void Bin(Histogram* hist, htype value, int set);
void WriteHist(char* ext, Histogram* hist);
```

```
long hElements(Histogram* hist, int set);
htype hMean(Histogram* hist, int set);
htype hStdDev(Histogram* hist, int set);
```

/*
Statistical Analysis Utility Functions

SplitTracks:

This is for converting the output of the tracker or the tracker's DST into two subsidiary lists of heads and tails.

SliceGeant:

This function takes a tracks structure 'gcut' obtained from a GEANT DST (by LoadGeantDST) and converts it into 2 Tracks structures -- one holding heads and one holding tails. The only cut made is the MinPlanes Cut -- a head must have at MinPlaH wires and a Tail must have at least MinPlaT wires. You may wish to use MakeCuts on these subsidiary lists of heads and tails. This function is important because it takes care of LSFin the wires from the GEANT dst to determine all the parameters in the tracks as well as the track types and chi² values.

This function also groups the heads and tails into standard shower groups based upon position and identity information in the DST file plus standard cuts.

MakeCuts:

Takes a Tracks structure, a metric and a chi² cut and applies the cuts to the tracks in the structure. First, all tracks with chi² < CCut are deleted, then all tracks within MCut of each other are merged and the one with the best chi² is kept. This function works best when the Tracks structure holds only heads or only tails. It is useful when you want to apply cuts more stringent than those made when a DST file was created.

CompTracks:

This takes a list of Tracks found from tracking and a list of tracks from something like GEANT and compares them. The two sets of tracks must be EITHER ONLY heads or ONLY tails. You must supply a metric cut to use in the determination of whether tracks are the same (you can use MetCutH, MetCutT as defaults). The number of tracks in 'gea' that were within 'MetCut' of tracks on 'avht' is returned in 'found,' The number of tracks in 'avht' not within MetCut of any tracks in 'gea' are returned in 'spur' (spurious tracks). Finally, if more than one track from 'avht' is within MetCut of a given track in 'gea' the excess is stored in 'mult' (multiple occurrences). Note: if one 'avht' track is within MetCut of more than one 'gea' track, then all the 'gea' tracks will be 'found.'

ComputePos:

Give a 'Z-position' along a track, the (x,y,z) position on the track at that z are returned in 'pos' as well as the 2x2 covariance matrix between x-y obtained from propagation of errors (using CHead and CTail defined in bjhough.c).

SourceProb:

This function used ZDCA to get the z of closest approach to the TeV beam and ComputePos to determine the position and errors here. It then uses these errors plus a 17" std dev. in z to determine the probability that the track came from C0. The value returned is S=-ln(P) where P is the probability from (0-1).

ZDCA:

This takes 2 lines, each parameterized by (xi + s yi) where xi is a vector, yi is a unit vector and 's' is a parameter along the line. This returns the Z-position on line 2 where the line 2 approaches closest to line 1. (No error checking for parallel lines).

```

HeadTailProb:
    This function takes a head track and a tail track and determines the
    probability that they "match." Returned is  $S = -\ln(P)$ .
    Also does tail-tail at the PbC.

trkDCA:
    Gives the distance of closest approach in inches between two
    tracks using the function vdDistance.

trkVertex:
    Gives the midpoint between the points of closest approach of two
    tracks forming a nominal vertex. This uses vdMidPoint.

GroupTracks:
    This function is supposed to apply the data analysis algorithm
    described in Section 4.4 of my thesis. It stores the groups
    in the grp variable and deletes the supposed spurious tracks
    from the lists. The features currently implemented are:
    Applying ChiCutH, ChiCutT, MetCutH and MetCutT to the tracks.
    Determines which heads form verticies by applying DCA < 1" and
    making a cut on the (x,y,z) position of the vertex.
    Makes a source cut of 5 on the tracks not participating in
    verticies, removing them from the heads list.
    Groups Heads and Tails using HeadTailProb < 6
    Groups Tails using DCA < 1" + a vertex cut
    Removes heads not belonging to tails or verticies that should
*/

void SplitTracks(Tracks* tracks, Tracks* heads, Tracks* tails);
void SliceGeant(Tracks* gcut, Tracks *heads, Tracks* tails, Groups *grp);
void MakeCuts(Tracks* htracks, float MCut, float CCut);
void CompTracks(Tracks* avht, Tracks* gea, float MetCut, int* found,
               int* spur, int* mult);
void ComputePos(dVector* pos, double err[2][2], Track* trk, Vtype z);
double SourceProb(Track* trk);
Vtype ZDCA(dVector x1, dVector y1, dVector x2, dVector y2);
double HeadTailProb(Track* seg1, Track* seg2);
dVector trkVertex(Track* t1, Track* t2);
Vtype trkDCA(Track* t1, Track* t2);

void GroupTracks(Tracks* th, Tracks* tt, Groups* grp);

/*****
Statistical Analysis Routines

These functions allow automatic analysis of common data features:

SetupStatistics:
    This initializes variables needed for statistical analysis. You
    must pass the types of analysis to be preformed. You may add
    or OR the types together. Realize that you should ONLY use
    analysis types consistent with the types of data available --
    eg. GEANT and / or AVHT dst info. SetupStatistics should be
    called after Setup_Hougher and before analysis. It automatically
    takes care of writing out the results at program completion.

DoStatistics:
    For EACH EVENT, pass separate Tracks for the heads and tails for
    the GEANT and AVHT DST files. If you are not using one of these,
    pass NULL instead.

Statistics types:

```

```

stCovar:
  Requires both GEANT and AVHT information.
  Computes the sample covariance matrices for heads and tails by
  comparing AVHT tracks to GEANT tracks in 1-track events and
  ignoring spurious tracks as much as possible. This function
  is only effective if the AVHT DST has few or no Same-Track cuts
  applied. The matrices output are in units of inches^2.
  It does not include AVHT tracks that are more than 1.5" away from
  the GEANT track in ANY parameter -- this is the measure to
  eliminate spurious tracks.

stMetGea:
  Requires only GEANT info. Gets the probability of coincidence
  distribution for ALL PAIRS or tracks in each event. The
  distribution is in terms of 'S'. Data sets 0-4 are for heads with
  occupancies 2-6; data sets 5-9 are for tails w/ occupancies
  2-6 tracks.

stMetAvh:
  Requires AVHT & GEANT info. Gets the probability of coincidence
  distribution for AVHT tracks in events that have only one head or
  one tail (says geant). Data set 0 = heads, 1 = tails.
  This is also in terms of 'S'.

stMetCom:
  Requires both. Gets the probability of coincidence distribution for
  AVHT tracks with respect to the GEANT tracks. Data sets 0-4 are
  GEANT head multiplicities 1-5, sets 5-9 are tail multiplicities 1-5.

stChi:
  This requires both AVHT & GEANT. It makes the chi^2 distribution
  for heads/tails found and not found. Found is defined as passing
  withing MetCutH/MetCutT of any GEANT DST track for that event.
  The data sets are 0=heads found, 1=heads spurious, 2=tails found,
  3=tails spurious.

stSource:
  This uses either AVHT or GEANT or both (depending on what is not
  set to NULL, but only applies to the heads. It creates the distribution
  of probabilities for the heads coming from C0. It uses SourceProb
  for this determination. The data sets are 0 = Geant, 1 = AVHT real
  tracks, 2 = AVHT spurious tracks

stEffic:
  Requires both AVHT & GEANT. Compares, event by event, AVHT tracks
  to GEANT tracks via CompTracks() to determine efficiencies. MetCutH/
  MetCutT are used to determine whether tracks are found. The data sets
  returned are: 0 = # of events processed for each multiplicity, 1 =
  % found for each multiplicity (0-1), 2 = number of spurious tracks per
  event for each multiplicity, 3 = number of multiples per event
  for each multiplicity. 4-7 are the same but for tails.

stDebug:
  opens and closes a debug info file "debug.txt" to which you can
  have stat.c function output debugging information.

*/

#define stCovar      1
#define stMetGea    2
#define stMetAvh    4
#define stMetCom    8
#define stChi       16
#define stSource    32
#define stEffic     64
#define stDebug     128

```

```

void SetupStatistics(long types);
void DoStatistics(Tracks *AVHTheads, Tracks *AVHTtails,
                 Tracks *GEANTheads, Tracks *GEANTtails,
                 Groups *GEANTgrps);

#endif

```

B.12 stat.c

```

/*****
CWRU MINIMAX TRACKING STATISTICS FUNCTIONS

By Erik Kangas

Last Revised 5/10/95
*****/

#include <string.h>
#include <stdlib.h>
#include <math.h>
#include "stat.h"
#include "hougher.h"

char out_file[50] = "stat";          /* Output base file name */

/*****
/* General Function declarations */

void OutFileBase(char* name) {
    strcpy(out_file,name);
}

FILE* OutFile(char* ext) {
    static char name[100];
    strcpy(name,out_file);
    strcat(name,ext);
    return fopen(name,"w");
}

/*****
/* Histogram object declarations */

Histogram CreateHist(int bins, htype lo, htype hi,int flags,int sets) {
    Histogram hist;
    int i;

    hist.flags = flags;
    hist.bins = bins;
    hist.low = lo;
    hist.high = hi;
    hist.inc = (hi - lo) / bins;
    hist.sets = sets;

    hist.hist = (htype**) malloc(sizeof(htype)*sets);
    if (!hist.hist) {
        printf("CreateHist: Memory Allocation Error\n");
        exit(1);
    }
    for (i=0;i<sets;i++) {
        hist.hist[i] = (htype*) malloc(sizeof(htype)*bins);
    }
}

```

```

        if (!hist.hist[i]) {
            printf("CreateHist: Memory Allocation Error\n");
            exit(1);
        }
        memset(hist.hist[i],0,sizeof(htype)*bins);
    }

    hist.ovrfl = (htype*) malloc(sizeof(htype)*sets);
    hist.undfl = (htype*) malloc(sizeof(htype)*sets);
    if (!hist.ovrfl || !hist.undfl) {
        printf("CreateHist: Memory Allocation Error\n");
        exit(1);
    }
    memset(hist.ovrfl,0,sizeof(htype)*sets);
    memset(hist.undfl,0,sizeof(htype)*sets);

    return hist;
}

void DeleteHist(Histogram* hist) {
    int i;
    for (i=0;i<hist->sets;i++) free(hist->hist[i]);
    free(hist->hist);
    free(hist->undfl);
    free(hist->ovrfl);
}

void Bin(Histogram* hist, htype value, int set) {
    int bin = (int) ( (value - hist->low) / hist->inc );
    if (set < 0 || set >= hist->sets) return;
    if ((bin >=0) && (bin < hist->bins)) hist->hist[set][bin]++;
    else if ((bin < 0) && (hist->flags & hUnderflow))
        hist->undfl[set]++;
    else if ((bin > hist->bins) && (hist->flags & hOverflow))
        hist->ovrfl[set]++;
}

void WriteHist(char* ext, Histogram* hist) {
    FILE* outf = OutFile(ext);
    int i,j;
    if (hist->flags & hUnderflow) {
        fprintf(outf,"%lf",hist->low - hist->inc);
        for (j=0;j<hist->sets;j++)
            fprintf(outf,"\t%lf",hist->undfl[j]);
        fprintf(outf,"\n");
    }
    for(i=0;i< hist->bins; i++) {
        fprintf(outf,"%lf", hist->low + hist->inc * i);
        for (j=0;j<hist->sets;j++)
            fprintf(outf,"\t%lf", (htype) hist->hist[j][i]);
        fprintf(outf,"\n");
    }
    if (hist->flags & hOverflow) {
        fprintf(outf,"%lf",hist->low + i*hist->inc);
        for (j=0;j<hist->sets;j++)
            fprintf(outf,"\t%lf",hist->ovrfl[j]);
        fprintf(outf,"\n");
    }
    fclose(outf);
}

htype hMean(Histogram* hist, int set) {
    htype m;
    int n,i;

    if (set < 0 || set >= hist->sets) {
        printf("Set out of bounds in hMean\n");
    }
}

```

```

        exit(1);
    }

    for (i=n=m=0;i<hist->bins;i++) {
        n += hist->hist[set][i];
        m += hist->hist[set][i] * (hist->low + hist->inc * (i+0.5) );
    }
    return ( (n) ? m/n : 0 );
}

long hElements(Histogram* hist,int set) {
    int i,n;
    if (set < 0 || set >= hist->sets) {
        printf("Set out of bounds in hElements\n");
        exit(1);
    }
    for (i=n=0;i < hist->bins;i++) n += hist->hist[set][i];
    return n;
}

hType hStdDev(Histogram* hist, int set) {
    hType std,x,offs = hist->low - hMean(hist,set);
    int i,n = hElements(hist,set);

    for (i=std=0;i<hist->bins;i++) {
        x = offs + hist->inc * (i + 0.5);
        std += hist->hist[set][i] * x*x;
    }
    return ( (n) ? sqrt(std / (n-1)) : 0 );
}

/*****
    Statistical Analysis Utility Functions
*/

void SplitTracks(Tracks* tracks, Tracks* heads, Tracks* tails) {
    heads->nheads = heads->ntracks = tracks->nheads;
    tails->ntails = tails->ntracks = tracks->ntails;
    heads->event = tails->event = tracks->event;
    heads->ntails = tails->nheads = 0;
    heads->nhits = tails->nhits = tracks->nhits;
    heads->mcut = tails->mcut = tracks->mcut;

    if (heads->ntracks)
        memcpy(heads->tracks,tracks->tracks,
            sizeof(Track)*heads->ntracks);
    if (tails->ntracks)
        memcpy(tails->tracks,&tracks->tracks[heads->ntracks],
            sizeof(Track)*tails->ntracks);
}

void SliceGeant(Tracks* gcut, Tracks *gh, Tracks* gt, Groups* grp) {
    Track head,tail;
    int i,j,ih,it;
    long plh,plt,pl,p;

    ClearGroups(grp);
    gt->ntracks = gh->ntracks = 0;
    gh->event = gt->event = gcut->event;
    gh->nhits = gt->nhits = gcut->nhits;
    gh->mcut = gt->mcut = gcut->mcut;
    gh->ntails = gt->nheads = 0;
    head.type = ttHead;
    tail.type = ttTail;
}

```

```

for (i=0;i<gcut->ntracks;i++) {

    head.nwires = tail.nwires = 0;
    head.E = tail.E = gcut->tracks[i].E;

    if (i < gcut->nheads) {
        head.x = gcut->tracks[i].x;
        head.y = gcut->tracks[i].y;
        head.z = gcut->tracks[i].z;
    }
    else if (i<gcut->nheads + gcut->ntails) {
        tail.x = gcut->tracks[i].x;
        tail.y = gcut->tracks[i].y;
        tail.z = gcut->tracks[i].z;
    }

    for (plh=plt=pl=j=0;j<gcut->tracks[i].nwires;j++) {
        if ( gcut->tracks[i].wires[j].pn < LeadP ) {
            p=head.wires[head.nwires].pn = gcut->tracks[i].wires[j].pn;
            head.wires[head.nwires++].wn = gcut->tracks[i].wires[j].wn;
            if (!(pl & (1<<p))) { pl |= (1<<p); plh++; }
        }
        else {
            p=tail.wires[tail.nwires].pn = gcut->tracks[i].wires[j].pn;
            tail.wires[tail.nwires++].wn = gcut->tracks[i].wires[j].wn;
            if (!(pl & (1<<p))) { pl |= (1<<p); plt++; }
        }
    }

    if (plh >= MinPlaH) {
        LeastFit(&head);
        gh->tracks[ih = gh->ntracks++] = head;
    }

    if (plt >= MinPlaT) {
        LeastFit(&tail);
        gt->tracks[it = gt->ntracks++] = tail;
    }

    /* Now Do Grouping */

    if (i < gcut->nheads && (plt >= MinPlaT || plh >= MinPlaH)) {

        if (plh >= MinPlaH) grp->groups[grp->ngroups].index = ih;
        else grp->groups[grp->ngroups].index = -1;
        if (plt >= MinPlaT) {
            grp->groups[grp->ngroups].primary = it;
            grp->groups[grp->ngroups].tails[0] = it;
            grp->groups[grp->ngroups].ntails = 1;
        }
        else {
            grp->groups[grp->ngroups].ntails = 0;
        }
        grp->ngroups++;
    }

    else if (i<gcut->nheads + gcut->ntails && plt >= MinPlaT) {

        for (j=0;j<grp->ngroups;j++)
            if (grp->groups[j].index == -1 &&
                grp->groups[j].v.x == tail.x &&
                grp->groups[j].v.y == tail.y &&
                grp->groups[j].v.z == tail.z) {

                grp->groups[j].tails[grp->groups[j].ntails++] = it;
                break;
            }
    }
}

```

```

        if (j == grp->ngroups) {
            grp->groups[grp->ngroups].index = -1;
            grp->groups[grp->ngroups].ntails = 1;
            grp->groups[grp->ngroups].tails[0] = it;
            grp->groups[grp->ngroups].v = vdVector(tail.x,tail.y,tail.z);
            grp->ngroups++;
        }
    } /* End of grouping */

} /* end splitting up geant tracks */

gh->nheads = gh->ntracks;
gt->nheads = gt->ntracks;
}

void MakeCuts(Tracks* htracks, float MCut, float CCut) {
    int i,j,t;

    for (i=1;i<htracks->ntracks;i++) {

        t = htracks->tracks[i].type;          /* Is it a head or tail? */

        /* Make chi^2 cut on tracks */

        if (htracks->tracks[i].chisqr > CCut) {
            DeleteTrack(htracks,i);
            i--;
            continue;
        }

        /* Apply SameTrackProb Cut to tracks */

        for (j=0;j<i;j++) {
            if (t != htracks->tracks[j].type) continue;

            if (SameTrackProb(&htracks->tracks[i],&htracks->tracks[j]) <= MCut)
            {
                if (htracks->tracks[i].chisqr > htracks->tracks[j].chisqr) {
                    DeleteTrack(htracks,i);
                    i--;
                    break;
                }
                else {
                    DeleteTrack(htracks,j);
                    i--;
                    j--;
                }
            }
        }
    }
}

void CompTracks(Tracks* avht, Tracks* gea, float MetCut,
               int* found, int* spur, int* mult) {
    static int i,j,fnd[50];
    static float met;

    (*found) = (*spur) = (*mult) = 0;
    memset(fnd,0,sizeof(int)*50);

    for (i=0;i<avht->ntracks;i++) {
        for (j=0;j<gea->ntracks;j++) {
            met = SameTrackProb(&avht->tracks[i],&gea->tracks[j]);
            if (met <= MetCut) {

```

```

        if (fnd[j]) (*mult)++;
        else {
            fnd[j] = 1; (*found)++;
        }
    }
}
if (j == gea->ntracks) (*spur)++;
}
}

void ComputePos(dVector* pos, double err[2][2], Track* trk, Vtype z) {
    Vtype Z = (z - trk->p0.z) / (trk->p1.z - trk->p0.z);
    double dx[4] = {0,0,0,0},
           dy[4] = {0,0,0,0};
    int i,j;

    dx[0] = -Z; dx[2] = Z;
    dy[1] = -Z; dx[3] = Z;
    *pos = vdVector(trk->p0.x + (trk->p1.x - trk->p0.x)*Z,
                   trk->p0.y + (trk->p1.y - trk->p0.y)*Z, z);

    err[0][0] = err[1][0] = err[1][1] = 0;

    if (trk->type == ttHead)
        for (i=0;i<4;i++) for (j=0;j<4;j++) {
            err[0][0] += dx[i] * dx[j] * CHead[i][j];
            err[1][1] += dy[i] * dy[j] * CHead[i][j];
            err[1][0] += dx[i] * dy[i] * CHead[i][j];
        }
    else
        for (i=0;i<4;i++) for (j=0;j<4;j++) {
            err[0][0] += dx[i] * dx[j] * CTail[i][j];
            err[1][1] += dy[i] * dy[j] * CTail[i][j];
            err[1][0] += dx[i] * dy[i] * CTail[i][j];
        }

    err[0][1] = err[1][0];
}

Vtype ZDCA(dVector x1, dVector y1, dVector x2, dVector y2) {
    dVector x3;
    Vtype s,u;
    x3 = vdSubtract(x2,x1);
    u = vdDot(y1,y2)*vdDot(y1,y2) - 1;
    s = ( vdDot(y2,x3) - vdDot(y1,y2)*vdDot(y1,x3) )/u;
    return vdAdd(x2, vdMultiply(y2,s)).z;
}

double SourceProb(Track* trk) {
    double z,r;
    dVector pos;
    double cov[2][2];

    z = ZDCA(C0,Beam,trk->p0,vdNormalize(
            vdSubtract(trk->p1,trk->p0)));
    ComputePos(&pos,cov,trk,z);
    Invert2x2(cov);
    r = Prob(z*z/289);
    z = pos.x*pos.x * cov[0][0] +
        pos.y*pos.y * cov[1][1] +
        2*pos.x*pos.y * cov[1][0] -
        (r ? log(r) : 999);
    return z;
}

double HeadTailProb(Track* seg1, Track* seg2) {

```

```

static int i,j;
static double cov[2][2],v[2],y;

if (seg1->type != seg2->type) {
    cov[0][0] = CHead[2][2] + CTail[0][0];
    cov[0][1] = CHead[2][3] + CTail[0][1];
    cov[1][1] = CHead[3][3] + CTail[1][1];
    cov[1][0] = cov[0][1];
    Invert2x2(cov);
    v[0] = seg1->p1.x - seg2->p0.x;
    v[1] = seg1->p1.y - seg2->p0.y;
}
else if (seg1->type == ttTail) {
    cov[0][0] = CTail[0][0];
    cov[0][1] = CTail[0][1];
    cov[1][1] = CTail[1][1];
    cov[1][0] = cov[0][1];
    Invert2x2(cov);
    v[0] = seg1->p0.x - seg2->p0.x;
    v[1] = seg1->p0.y - seg2->p0.y;
}

for (i=y=0;i<2;i++) for (j=0;j<2;j++)
    y += v[i] * cov[i][j] * v[j];

return y/2;
}

Vtype trkDCA(Track* t1, Track* t2) {
    dVector x1,y1,x2,y2;
    x1 = t1->p0; x2 = t2->p0;
    y1 = vdNormalize(vdSubtract(t1->p1,t1->p0));
    y2 = vdNormalize(vdSubtract(t2->p1,t2->p0));
    return vdDistance(x1,y1,x2,y2);
}

dVector trkVertex(Track* t1, Track* t2) {
    dVector x1,y1,x2,y2;
    x1 = t1->p0; x2 = t2->p0;
    y1 = vdNormalize(vdSubtract(t1->p1,t1->p0));
    y2 = vdNormalize(vdSubtract(t2->p1,t2->p0));
    return vdMidpoint(x1,y1,x2,y2);
}

void GroupTracks(Tracks* th, Tracks* tt, Groups* grp) {
    char vert[MAXTRACKS],grh[MAXTRACKS],grt[MAXTRACKS];
    double d,x,y;
    dVector v;
    int i,j,k,g;

    memset(vert,0,sizeof(char)*MAXTRACKS);
    memset(grh,-1,sizeof(char)*MAXTRACKS);
    memset(grt,-1,sizeof(char)*MAXTRACKS);

    /* Make Chi^2 and same-track cuts on the AVHT DST tracks */

    ClearGroups(grp);
    MakeCuts(th,MetCutH,ChiCutH);
    MakeCuts(tt,MetCutT,ChiCutT);

    /* Detect head-verticies with DCA informaton */
    /* must pass w/i one inch of each other. */
    /* Only keep verticies between -100" and PList[0].P1.Z" */
    /* Only keep verticies w/ y between -5" and + 20" */

```

```

for (g=0,i=1;i<th->ntracks;i++) for (j=0;j<i;j++) {
  d = trkDCA(&th->tracks[i],&th->tracks[j]);
  v = trkVertex(&th->tracks[i],&th->tracks[j]);
  if (d <= 1 &&
      v.z >= -100 && v.z <= PList[0].Pl.z &&
      v.y >= -10 && v.y <= 10 &&
      v.x >= -5 && v.x <= 20) {

    /* Assign them a vertex number */

    if (!vert[i] && vert[j]) vert[i] = vert[j];
    else if (!vert[j] && vert[i]) vert[j] = vert[i];
    else if (!vert[j] && !vert[i]) vert[i] = vert[j] = ++g;
    else for (k=0;k<=i;k++)
      if (vert[k] == vert[i]) vert[k] = vert[j];
  }
}

/* Apply source cut to tracks w/o a vertex! */

for (i=0;i<th->ntracks;i++) {
  if (vert[i]) continue;
  if (SourceProb(&th->tracks[i]) > 5) {
    DeleteTrack(th,i);
    memmove(&vert[i],&vert[i+1],sizeof(char)*(MAXTRACKS-i-1));
    i--;
  }
}

/* Match Heads to Tails probabilistically looking for charged
tracks passing through the whole detector */

for (i=g=0;i<th->ntracks;i++) for (j=0;j<tt->ntracks;j++) {
  d = HeadTailProb(&th->tracks[i],&tt->tracks[j]);
  if (d < 6.0) {
    grh[i] = grt[j] = g;
    grp->groups[g].index = i;
    grp->groups[g].ntails = 1;
    grp->groups[g].primary = grp->groups[g].tails[0] = j;
    g++;
    break;
  }
}
grp->ngroups = g;

/* Now Group Tails into showers */
/* Use a DCA cut of 1.0" with a vertex cut near the PbC */

for (i=0;i<tt->ntracks;i++) {
  if (grt[i]) continue;          /* Already Grouped */

  /* First check existing groups */

  for (j=0;j<grp->ngroups;j++) {
    d = HeadTailProb(&th->tracks[grp->groups[j].index],&tt->tracks[i]);
    if (d < 6.0) {
      grp->groups[j].tails[grp->groups[j].ntails++] = i;
      grt[i] = j;
      break;
    }
  }
  for (k=0;k<grp->groups[j].ntails;k++) {
    d = trkDCA(&tt->tracks[i],&tt->tracks[grp->groups[j].tails[k]]);
    v = trkVertex(&tt->tracks[i],&tt->tracks[grp->groups[j].tails[k]]);
    if (d < 1.0 &&
        v.z >= LeadZ - 5 && v.z <= LeadZ + 5 &&
        sqrt( POW2(v.x-cfx) + POW2(v.y-cfy)) < 10) {

```

```

        grp->groups[j].tails[grp->groups[j].ntails++] = i;
        grt[i] = j;
        break;
    }
}
if (k != grp->groups[j].ntails) break;
}
if (j != grp->ngroups) continue;      /* Next one! */

/* Now check against other tails */

for (j=0;j<i;j++) {
    if (grt[j]) continue;
    d = trkDCA(&tt->tracks[i],&tt->tracks[j]);
    v = trkVertex(&tt->tracks[i],&tt->tracks[j]);
    if (d < 1.0 &&
        v.z >= LeadZ - 5 && v.z <= LeadZ + 5 &&
        sqrt( POW2(v.x-cfx) + POW2(v.y-cfy)) < 10) {
        g = grp->ngroups;
        grp->groups[g].index = -1;
        grp->groups[g].primary = -1;
        grp->groups[g].ntails = 2;
        grp->groups[g].tails[0] = i;
        grp->groups[g].tails[1] = j;
        grp->ngroups++;
        grt[i] = grt[j] = g;
        break;
    }
}

if (j != i) continue;                /* matched */
g = grp->ngroups;                      /* Group to itself */
grp->groups[g].index = -1;
grp->groups[g].primary = i;
grp->groups[g].ntails = 1;
grp->groups[g].tails[0] = i;
grp->ngroups++;
grt[i] = g;
}

/* Remove any heads that do not have a vertex or tail & should */

for (i=0;i<th->ntracks;i++) {
    if (vert[i] || (grh[i] != -1)) continue;
    d = (PList[LeadP+MinPlat-1].Pl.z - th->tracks[i].p0.z) /
        (th->tracks[i].pl.z - th->tracks[i].p0.z);

    x = (th->tracks[i].pl.x - th->tracks[i].p0.x) * d;
    y = (th->tracks[i].pl.y - th->tracks[i].p0.y) * d;

    if (sqrt(POW2(x-cfx) + POW2(y-cfy)) < 9) {
        DeleteTrack(th,i);
        memmove(&vert[i],&vert[i+1],sizeof(char)*(MAXTRACKS-i-1));
        memmove(&grh[i],&grh[i+1],sizeof(char)*(MAXTRACKS-i-1));
    }
}
}

/*****
Statistical Analysis Routines
*/

long    StatTypes = 0;                /* Analyses to perform */

```

```

nrctype** covh,**covt;                                /* stCovar */
long      ncovh=0,ncovt=0,
          ncovhe=0,ncovte=0;

Histogram MetAvh,                                     /* stMetAvh */
          MetGea,                                     /* stMetGea */
          MetCom,                                     /* stMetCom */
          hChi,                                       /* stChi */
          hSource,                                    /* stSource */
          hMatch,                                    /* stMatch */
          hMatchZ,
          hEffic;                                     /* stEffic */

FILE*     tfile;

void CloseStatistics(void) {
    FILE* f;
    int i,j;

    /* Sample Covariance matrix information */

    if (StatTypes & stCovar) {
        f = OutFile(".cov");
        fprintf(f,"Sample Covariance Matrix For Heads in inches^2\n\n");
        fprintf(f,"%ld events, %ld measurements\n\n",ncovhe,ncovh);
        for (i=1;i<5;i++) {
            for (j=1;j<5;j++) fprintf(f,"%10.6lf\t",covh[i][j]/=ncovh);
            fprintf(f,"\n");
        }
        gaussj(covh,4,NULL,0);
        fprintf(f,"\nSample Inverse Covariance Matrix For Heads\n\n");
        for (i=1;i<5;i++) {
            for (j=1;j<5;j++) fprintf(f,"%10.6lf\t",covh[i][j]);
            fprintf(f,"\n");
        }
        fprintf(f,"\nSample Covariance Matrix For Tails\n\n");
        fprintf(f,"%ld events, %ld measurements\n\n",ncovte,ncovt);
        for (i=1;i<5;i++) {
            for (j=1;j<5;j++) fprintf(f,"%10.6lf\t",covt[i][j]/=ncovt);
            fprintf(f,"\n");
        }
        gaussj(covt,4,NULL,0);
        fprintf(f,"\nSample Inverse Covariance Matrix For Tails\n\n");
        for (i=1;i<5;i++) {
            for (j=1;j<5;j++) fprintf(f,"%10.6lf\t",covt[i][j]);
            fprintf(f,"\n");
        }
        fclose(f);
    }

    if (StatTypes & stMetGea) WriteHist(".mg",&MetGea);
    if (StatTypes & stMetAvh) WriteHist(".ma",&MetAvh);
    if (StatTypes & stMetCom) WriteHist(".mc",&MetCom);
    if (StatTypes & stChi)    WriteHist(".chi",&hChi);
    if (StatTypes & stSource) WriteHist(".sor",&hSource);

    if (StatTypes & stEffic) {
        for (i=0;i<20;i++) {
            if (hEffic.hist[0][i]) {
                if (i) hEffic.hist[1][i] /= (i*hEffic.hist[0][i]);
                hEffic.hist[2][i] /= hEffic.hist[0][i];
                hEffic.hist[3][i] /= hEffic.hist[0][i];
            }
            if (hEffic.hist[4][i]) {
                if (i) hEffic.hist[5][i] /= (i*hEffic.hist[4][i]);
            }
        }
    }
}

```

```

        hEffic.hist[6][i] /= hEffic.hist[4][i];
        hEffic.hist[7][i] /= hEffic.hist[4][i];
    }
}
WriteHist(".eff",&hEffic);
}

if (StatTypes & stDebug) fclose(tfile);
}

void SetupStatistics(long types) {
    int i,j;

    StatTypes = types;

    if (types & stCovar) {
        covh = matrix(4,4);
        covt = matrix(4,4);
        for (i=1;i<5;i++) for (j=1;j<5;j++)
            covh[i][j] = covt[i][j] = 0;
    }
    if (types & stMetGea) MetGea = CreateHist(100,0,20,hOverflow,10);
    if (types & stMetAvh) MetAvh = CreateHist(100,0,20,hOverflow,10);
    if (types & stMetCom) MetCom = CreateHist(100,0,20,hOverflow,10);
    if (types & stChi) hChi = CreateHist(100,0,6,0,4);
    if (types & stSource) hSource = CreateHist(100,0,20,hOverflow,3);
    if (types & stEffic) hEffic = CreateHist(20,0,20,0,8);
    if (types & stDebug) tfile = fopen("debug.txt","w");

    atexit(CloseStatistics);
}

void DoStatistics(Tracks*th, Tracks*tt, Tracks*gh, Tracks*gt, Groups*ggrp) {
    double v[5],x,y;
    int i,j,k,f,m,s;
    dVector vert;

    /* Sample Covariance matrices */

    if (StatTypes & stCovar) {
        if (gh->ntracks == 1) {
            ncovhe++;
            for (i=0;i<th->ntracks;i++) {
                v[1] = gh->tracks[0].p0.x - th->tracks[i].p0.x;
                v[2] = gh->tracks[0].p0.y - th->tracks[i].p0.y;
                v[3] = gh->tracks[0].p1.x - th->tracks[i].p1.x;
                v[4] = gh->tracks[0].p1.y - th->tracks[i].p1.y;
                if (fabs(v[1]) > 1.5 || fabs(v[2]) > 1.5 ||
                    fabs(v[3]) > 1.5 || fabs(v[4]) > 1.5) continue;
                for (j=1;j<5;j++) for (k=1;k<5;k++)
                    covh[j][k] += v[j] * v[k];
                ncovh ++;
            }
        }

        if (gt->ntracks == 1) {
            ncovte++;
            for (i=0;i<tt->ntracks;i++) {
                v[1] = gt->tracks[0].p0.x - tt->tracks[i].p0.x;
                v[2] = gt->tracks[0].p0.y - tt->tracks[i].p0.y;
                v[3] = gt->tracks[0].p1.x - tt->tracks[i].p1.x;
                v[4] = gt->tracks[0].p1.y - tt->tracks[i].p1.y;
                if (fabs(v[1]) > 1.5 || fabs(v[2]) > 1.5 ||
                    fabs(v[3]) > 1.5 || fabs(v[4]) > 1.5) continue;
                for (j=1;j<5;j++) for (k=1;k<5;k++)

```

```

        covt[j][k] += v[j] * v[k];
        ncovt ++;
    }
}

if (StatTypes & stMetGea) {
    if (gh->ntracks < 7)
        for (i=1;i<gh->ntracks;i++)
            for (j=0;j<i;j++) {
                x = SameTrackProb(&gh->tracks[j],&gh->tracks[i]);
                Bin(&MetGea,x,gh->ntracks-2);
            }

    if (gt->ntracks < 7)
        for (i=1;i<gt->ntracks;i++)
            for (j=0;j<i;j++) {
                x = SameTrackProb(&gt->tracks[j],&gt->tracks[i]);
                Bin(&MetGea,x,gt->ntracks-2+5);
            }
}

if (StatTypes & stMetAvh) {
    if (gh->ntracks == 1)
        for (i=1;i<th->ntracks;i++)
            for (j=0;j<i;j++) {
                x = SameTrackProb(&th->tracks[j],&th->tracks[i]);
                Bin(&MetAvh,x,0);
            }

    if (gt->ntracks == 1)
        for (i=1;i<tt->ntracks;i++)
            for (j=0;j<i;j++) {
                x = SameTrackProb(&tt->tracks[j],&tt->tracks[i]);
                Bin(&MetAvh,x,1);
            }
}

if (StatTypes & stMetCom) {
    if (gh->ntracks < 6)
        for (i=0;i<gh->ntracks;i++)
            for (j=0;j<th->ntracks;j++) {
                x = SameTrackProb(&th->tracks[j],&gh->tracks[i]);
                Bin(&MetCom,x,gh->ntracks-1);
            }

    if (gh->ntracks < 6)
        for (i=0;i<gt->ntracks;i++)
            for (j=0;j<tt->ntracks;j++) {
                x = SameTrackProb(&tt->tracks[j],&gt->tracks[i]);
                Bin(&MetCom,x,5+gt->ntracks-1);
            }
}

if (StatTypes & stChi) {
    for (i=0;i<th->ntracks;i++) {
        for (j=0;j<gh->ntracks;j++)
            if (SameTrackProb(&th->tracks[i],&gh->tracks[j]) <= MetCutH) {
                Bin(&hChi,th->tracks[i].chisqr,0);
                break;
            }
        if (j == gh->ntracks)
            Bin(&hChi,th->tracks[i].chisqr,1);
    }
    for (i=0;i<tt->ntracks;i++) {
        for (j=0;j<gt->ntracks;j++)

```

```

        if (SameTrackProb(&tt->tracks[i],&gtt->tracks[j]) <= MetCutH) {
            Bin(&hChi,tt->tracks[i].chisqr,2);
            break;
        }
    if (j == gt->ntracks)
        Bin(&hChi,tt->tracks[i].chisqr,3);
    }
}

if (StatTypes & stSource) {
    if (gh != NULL)
        for (i=0;i<gh->ntracks;i++)
            Bin(&hSource,SourceProb(&gh->tracks[i]),0);
    if (th != NULL && gh != NULL)
        for (i=0;i<th->ntracks;i++) {
            for (j=0;j<gh->ntracks;j++)
                if (SameTrackProb(&gh->tracks[j],&th->tracks[i]) < MetCutH) {
                    Bin(&hSource,SourceProb(&th->tracks[i]),1);
                    break;
                }
            if (j == gh->ntracks)
                Bin(&hSource,SourceProb(&th->tracks[i]),2);
        }
    }

if (StatTypes & stEffic) {
    if (gh->ntracks < 20) {
        CompTracks(th,gh,MetCutH,&f,&s,&m);
        Bin(&hEffic,gh->ntracks,0);
        hEffic.hist[1][gh->ntracks] += f;
        hEffic.hist[2][gh->ntracks] += s;
        hEffic.hist[3][gh->ntracks] += m;

        if (StatTypes & stDebug && f != gh->ntracks) {
            WriteTEvent(tfile,gh);
            WriteTEvent(tfile,th);
        }
    }

    if (gt->ntracks < 20) {
        CompTracks(tt,gt,MetCutT,&f,&s,&m);
        Bin(&hEffic,gt->ntracks,4);
        hEffic.hist[5][gt->ntracks] += f;
        hEffic.hist[6][gt->ntracks] += s;
        hEffic.hist[7][gt->ntracks] += m;
    }
}
}

```