

**COMPUTER SIMULATION OF THE STRAWMAN DIGITIZER
FOR THE DUMAND II EXPERIMENT**

S.T. Dye and E.S. Hazen

Boston University

17 December 1990

Introduction

The depths of the two on-chip buffers are critical. They must be deep enough to avoid loss of useful data, yet as small as possible to reduce design complexity and chip costs. For initial designs, the depths were determined by back-of-the-envelop calculations for worst-case scenarios. However, cost/performance optimization of buffer depths requires realistic simulation of data streams resulting from physical processes, known and theorized, which are observable by DUMAND. A simulation program, written for this purpose, is described.

It is sufficient for the digitizer design to consider one string of 24 optical modules (OMs) each spaced 10 m apart. Pulse-width encoded temporal signals resulting from illuminating a string of OMs by a variety of sources are calculated and propagated as a data stream through the simulated digitizer. The maximum number of words stored in the buffers at any given time during data processing are recorded for each event. The buffer depths can be determined from these distributions.

Optical Module Illumination

The light sources considered are:

- 1) single fast cone (propagates with $\beta \equiv \frac{v}{c} = 1$ with emission at Čerenkov angle);
- 2) two fast parallel cones of random separation;
- 3) stationary single isotropic.

They are designed to simulate single and di-muons, and electromagnetic showers (or proton decay) respectively. The single and di-muon simulations are more realistic in terms of illumination. Each type of event is simulated 10,000 times. Each track has a randomly chosen impact parameter ($0 < y_0 < 20$ m) and vertical point of closest approach ($100 < z_0 < 330$ m) (the full range of OM positions). Muons have a randomly chosen zenith angle ($-1 < \cos\theta_z < 1$).

The zero time plane is perpendicular to the muon tracks and intersects the first chosen track 100 m back from the point ($x = 0, y_0, z_0$).

OM illumination is by direct light only. Shadowing of one OM by another is not considered. OMs are pointlike. Pulse heights, in photoelectrons, are estimated using

$$Q = 2.66 \times \left(1 + \frac{37}{D} \times e^{-0.015 \times D}\right) \times \epsilon \text{ pes},$$

where D is the source-OM distance in meters, and ϵ is the OM angular efficiency. The angular efficiency function has the form

$$\epsilon = .52 - .48 \times \cos\theta_i,$$

where θ_i is the light incidence angle measured from the vertical (OMs face down). The expression for Q provides double the OM response determined by Clem (see figure 24 of his thesis). OMs trigger with random probability described by the Poisson distribution of mean value equal to Q. For the following pulse height dependent timing values the condition $Q \geq 1$ is forced. OM trigger times are delayed by

$$t_d = 40 - 10 \times \log_{10} Q \text{ ns}$$

and have Gaussian smearing

$$t_{FWHM} = 6 \times \ln(Q + 1) \text{ ns}.$$

Pulse width times are calculated from

$$t_{pw} = 24 + 46 \times \log_{10} Q \text{ ns}.$$

Noise hits are added randomly at 100 kHz per OM, with random pulse heights $1.0 < Q < 1.5$. OMs are allowed to trigger twice during the event simulation. Light propagates in the medium surrounding the OMs with velocity $0.75c$, where c is the speed of light in vacuum.

Data Flow in Digitizer

OM pulses travel towards the digitizer with velocity $0.67c$. For the 10 m intermodule distance, this is 50 ns. The time series data stream is checked for impossible OM hits (pulse rise time during a previous pulse on a given channel), sorted, zero suppressed, and converted to digitizer counts $1/1.024 \text{ ns ct}^{-1}$. With every clock count, the digitized data stream is checked for transitions in the state of any channel (rising or falling pulse). If a transition in any

channel occurs, then a channel number encoded word is put in the first buffer. Every three counts one time word per channel transition, if available, is read into the second buffer. When the time words for all channel transitions for a given count are in the second buffer, the channel encoded word is removed from the first buffer. Every eighty counts two time words, if available, are removed from the second buffer and stored in an array (Hotrod). This array is then checked against the original data stream for errors. The simulation ends after the last channel transition is read into the first buffer. The interested reader is encouraged to inspect the computer code, which is listed in an attached appendix.

Results

The results of the simulations are presented in two sets of distributions which reflect the viability of the described strawman digitizer. Figures 1-4 derive from simulations in which OM triggers are calculated from a Poisson probability distribution described by the simulated Q . Hopefully, this represents reality conservatively. For reference, a worst-case scenario in which OMs have trigger efficiency $\epsilon = 1.0$ is simulated. These results are shown in Figures 5-8.

Figure 1 shows the frequency of the maximum number of words in the first buffer. This indicates the required depth of the buffer. None of the 30,000 simulated events required a word depth of greater than 3. Figure 2 displays the frequency of the maximum number of pulses coincident in one digitizer count. The frequency of the maximum number of words in the second buffer is shown in Figure 3. The frequency of the number of words left in the second buffer is shown in Figure 4. Multiplying this value by 40 ns gives the time required to empty the buffer. The same distributions are presented in Figures 5-8 for the worst-case scenario. Note the abscissa scale change on Figures 5, 7, and 8. The most dramatic difference between the “realistic” and worst-case simulations is the maximum number of words in the first buffer (compare Figures 1 and 5). It inflated from 3 to 32 for di-muons, from 2 to 7 for single muons, and from 3 to 9 for isotropic events. In all cases, the event geometries responsible for the increases are similar. For muons, they are tracks passing near the top of the string with a zenith angle of $\sim 45^\circ$, and vertices near the top of the string for isotropic events.

The zenith angle dependence of buffer depths is explored with greater statistics for downward (upward) double-tracks passing near the string top (bottom). For each of ten intervals

in $\cos\theta_z$, between down (up) and horizontal, 1000 events are simulated with 100% OM triggering. Impact parameters are randomly distributed ($0 < y_0 < 20 \text{ m}$). The maximum numbers of words in the two buffers are presented in Table 1a (1b). The deep buffer requirements of downward double-track events passing near the string top with ($0.4 < \cos\theta_z < 0.8$) is clearly demonstrated.

To determine if this geometry causes such extreme pileup with "realistic" OM triggering, 10,000 events of each type are simulated with restricted vertical points of closest approach ($330 < z_0 < 350 \text{ m}$) and zenith angles ($0.6 < \cos\theta_z < 0.7$). The frequency distributions of maximum words in the first and second buffers are shown in Figures 9 and 10. The inefficiency of OMs illuminated from the top greatly decreases pileup in the buffers, for events of this geometry.

Conclusions

The initial digitizer design specified depths of 5 and 100 for the first and second buffers. These seem adequate in the "realistic" simulation. In the worst-case simulation a first buffer depth of 5 is too small by at least a factor of five and a second buffer depth of 100 begins to be marginally adequate. However, the event geometries causing the pileup in the first buffer illuminate OMs from the top side where their efficiency is very small. Clearly 100% OM triggering efficiency at these angles is a gross overestimate. It bears mentioning that no upward-going event, be it a single or double muon, required a first buffer depth greater than 5.

The simulation program has much room for improvement. At present, each OM is only allowed two hits per event. This restriction will be lifted after restructuring a few subroutines. Just one noise hit per OM per event is possible. This too must be generalized for multiple hits. OM response parameters, timing and pulse height, are estimates. Measured values for both JOM and EOM need to be incorporated. The digitizer design is very sensitive to these parameters. Data streams only from OMs. Data from environmental and calibration modules, and clock roll-over words need to be added. Illumination of OMs by calibration modules should be included. Other light sources, such as nuclearites, Rubakov monopoles, etc., could be simulated. The case not considered could be the most troublesome.

Preliminary results of the digitizer simulation are useful for setting lower and upper bounds on the depths of the buffers. The first buffer should be at least 5 words deep, and probably 35-40 words is the deepest to consider. The depth of the second buffer is more certain. A value of 100 words seems adequate for the simulation cases considered.

Table 1a: Maximum buffer depths for downward double-tracks incident on the top of a string with 100% OM triggering.

String top 330-350 m		
$\cos\theta_z$	Fifo 1	Fifo 2
1.0–0.9	2	10
0.9–0.8	2	14
0.8–0.7	35	63
0.7–0.6	37	70
0.6–0.5	31	74
0.5–0.4	19	72
0.4–0.3	9	57
0.3–0.2	4	46
0.2–0.1	4	40
0.1–0.0	3	36

Table 1b: Maximum buffer depths for upward double-tracks incident on the bottom of a string with 100% OM triggering.

String bottom 80-100 m		
$-\cos\theta_z$	Fifo 1	Fifo 2
1.0–0.9	2	8
0.9–0.8	2	7
0.8–0.7	2	11
0.7–0.6	2	21
0.6–0.5	2	37
0.5–0.4	2	34
0.4–0.3	2	26
0.3–0.2	2	24
0.2–0.1	2	22
0.1–0.0	2	23

Figure Captions

Figure 1. Frequency of maximum number of words in the first buffer for calculated OM trigger efficiencies.

Figure 2. Frequency of maximum number of pulses coincident in one digitizer count for calculated OM trigger efficiencies.

Figure 3. Frequency of maximum number of words in the second buffer for calculated trigger efficiencies.

Figure 4. Frequency of maximum number of words left in the second buffer after the last OM pulse in the event arrives at the first buffer.

Figure 5. Same as Figure 1 with OM trigger efficiencies 100%.

Figure 6. Same as Figure 2 with OM trigger efficiencies 100%.

Figure 7. Same as Figure 3 with OM trigger efficiencies 100%.

Figure 8. Same as Figure 4 with OM trigger efficiencies 100%.

Figure 9. Same as Figure 1 with ($330 < z_0 < 350 \text{ m}$) and ($0.6 < \cos\theta_z < 0.7$).

Figure 10. Same as Figure 3 with ($330 < z_0 < 350 \text{ m}$) and ($0.6 < \cos\theta_z < 0.7$).

FIGURE 1:

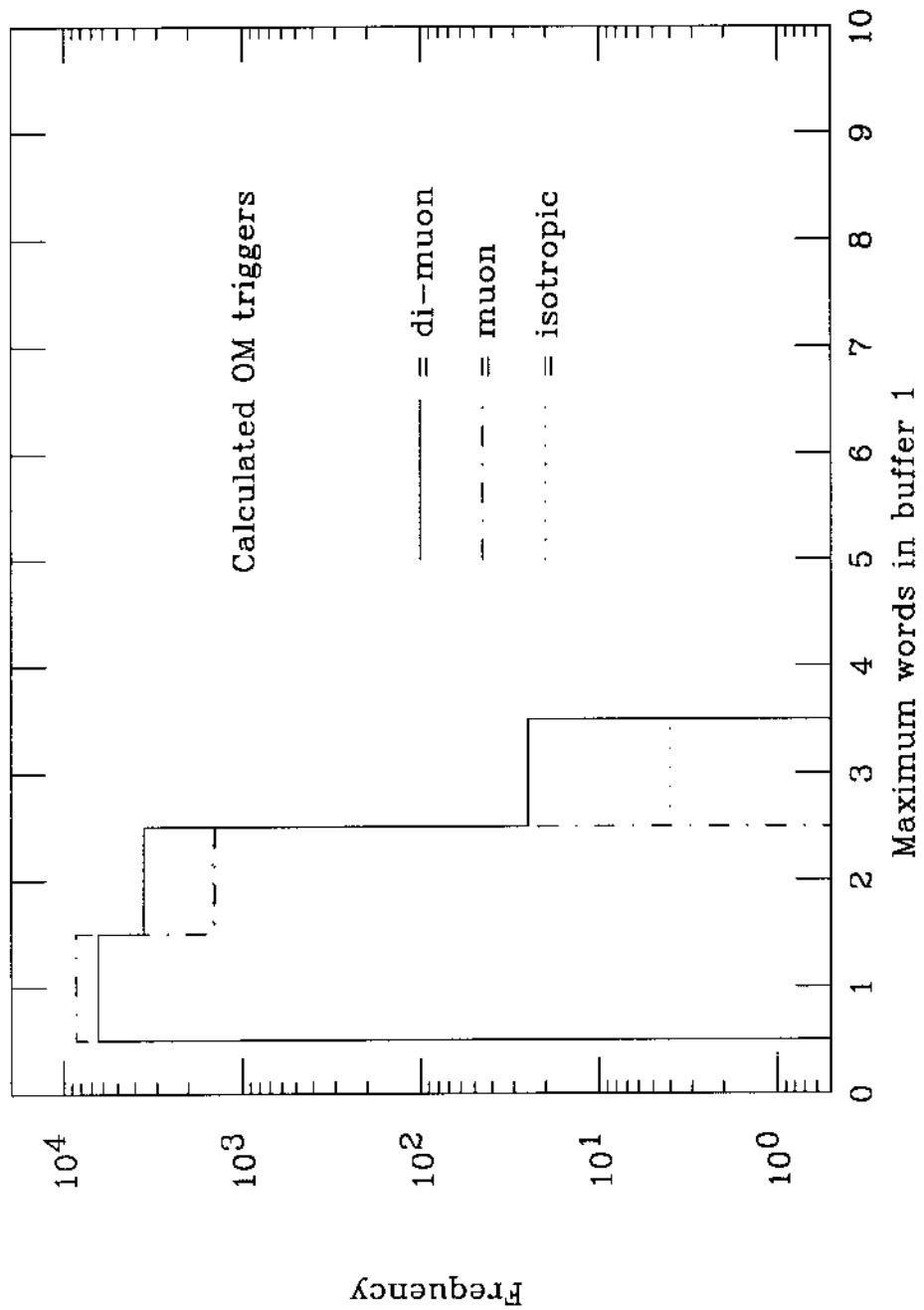


FIGURE 2:

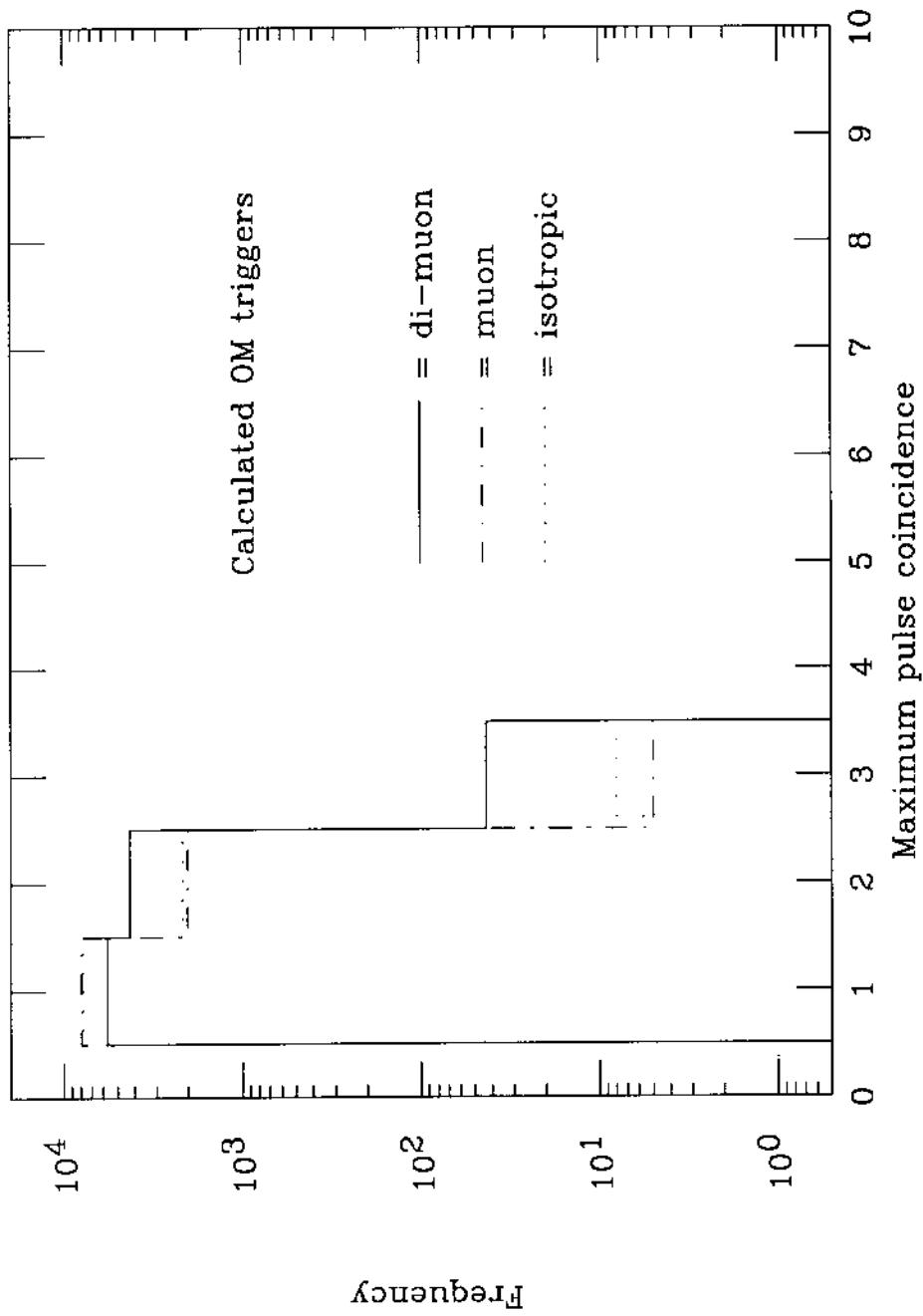


FIGURE 3:

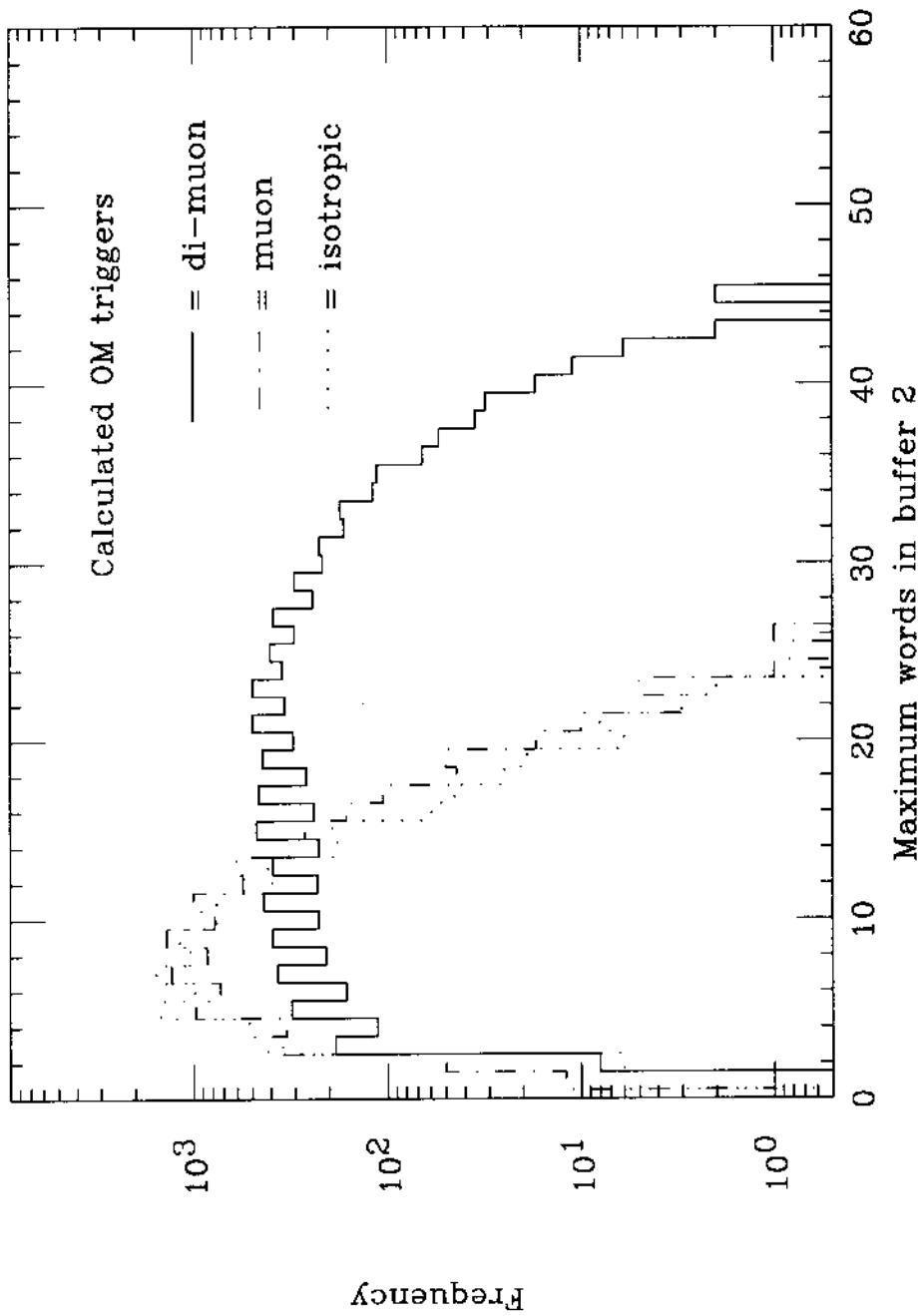


FIGURE 4:

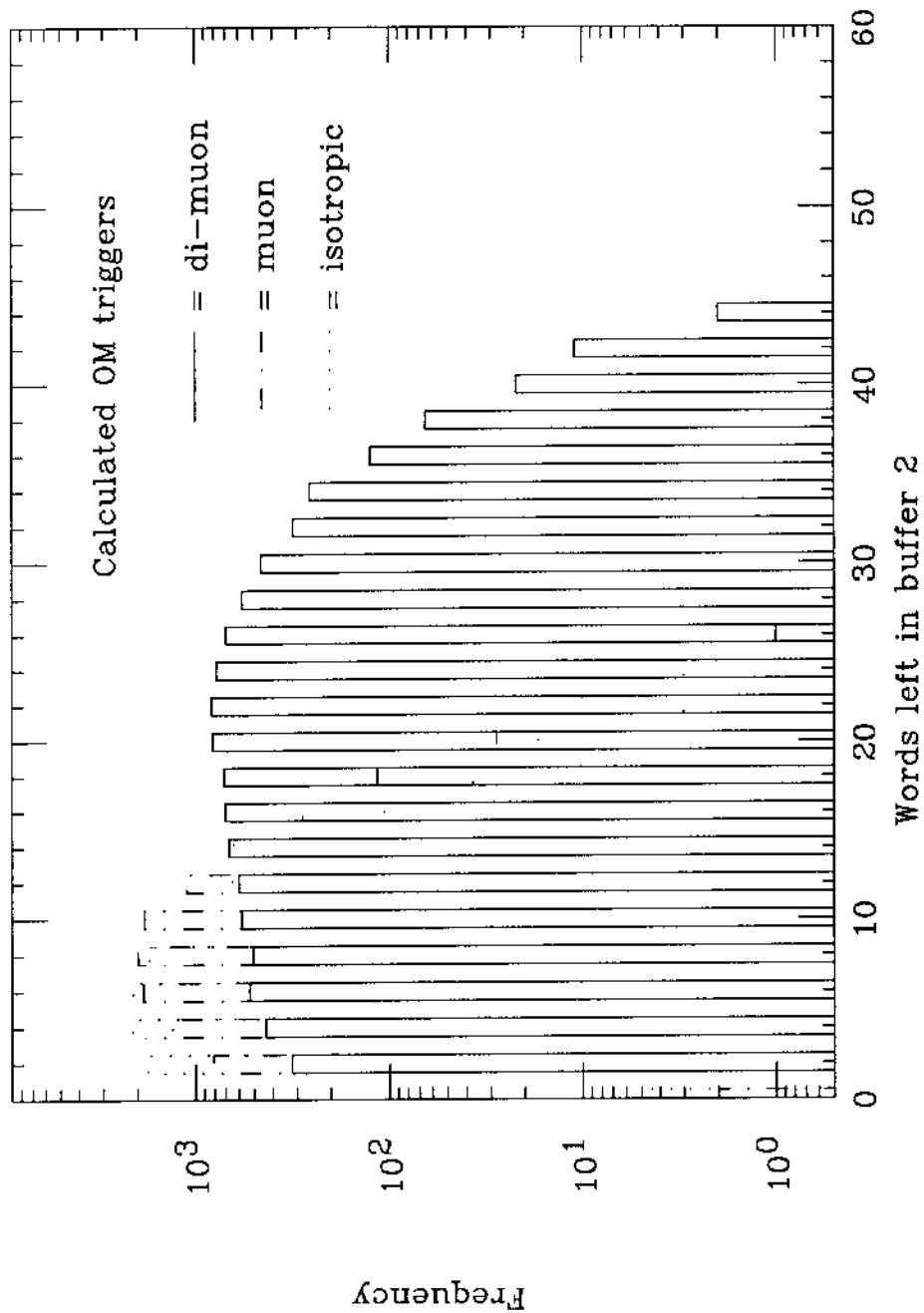


FIGURE 5:

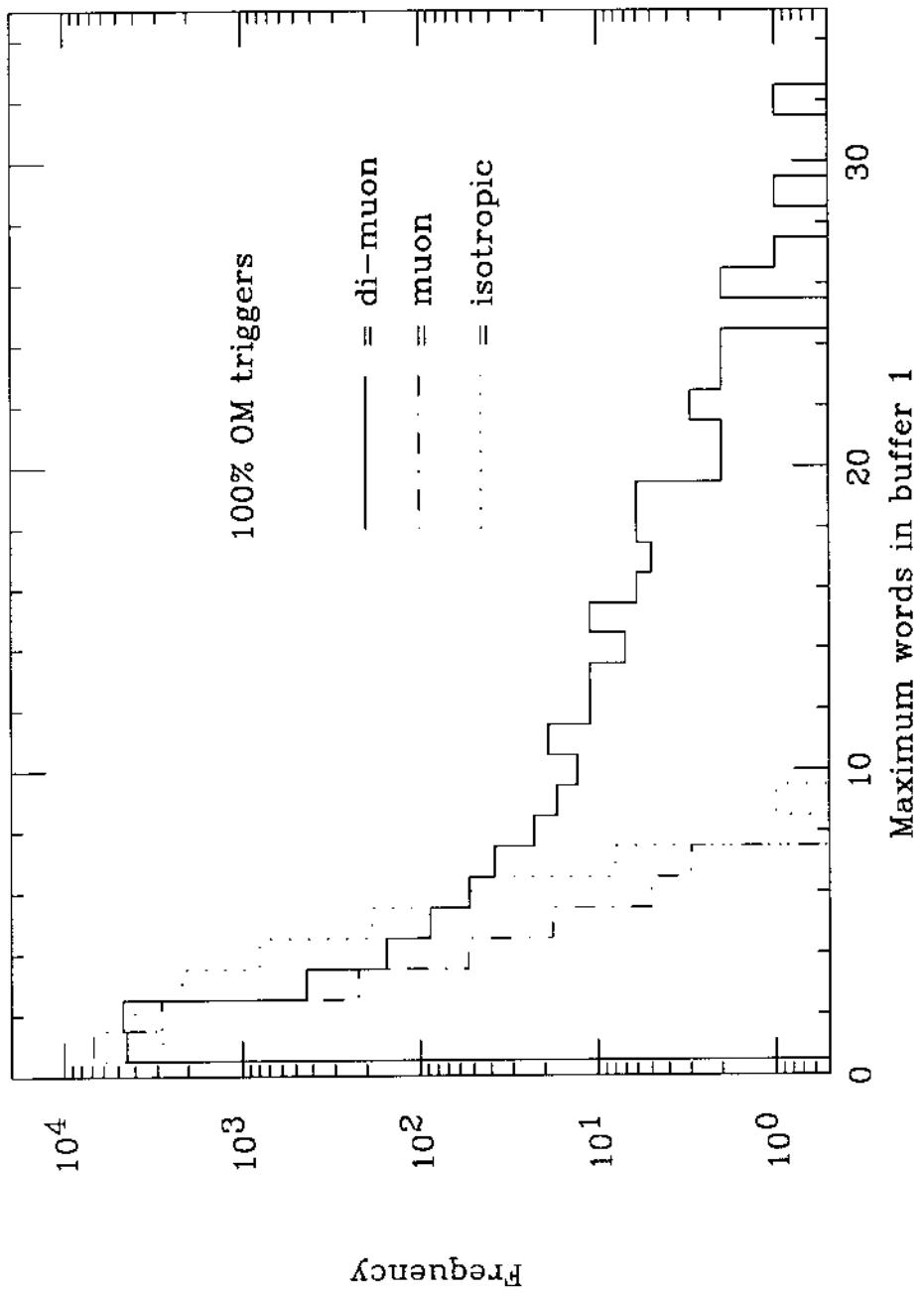


FIGURE 6:

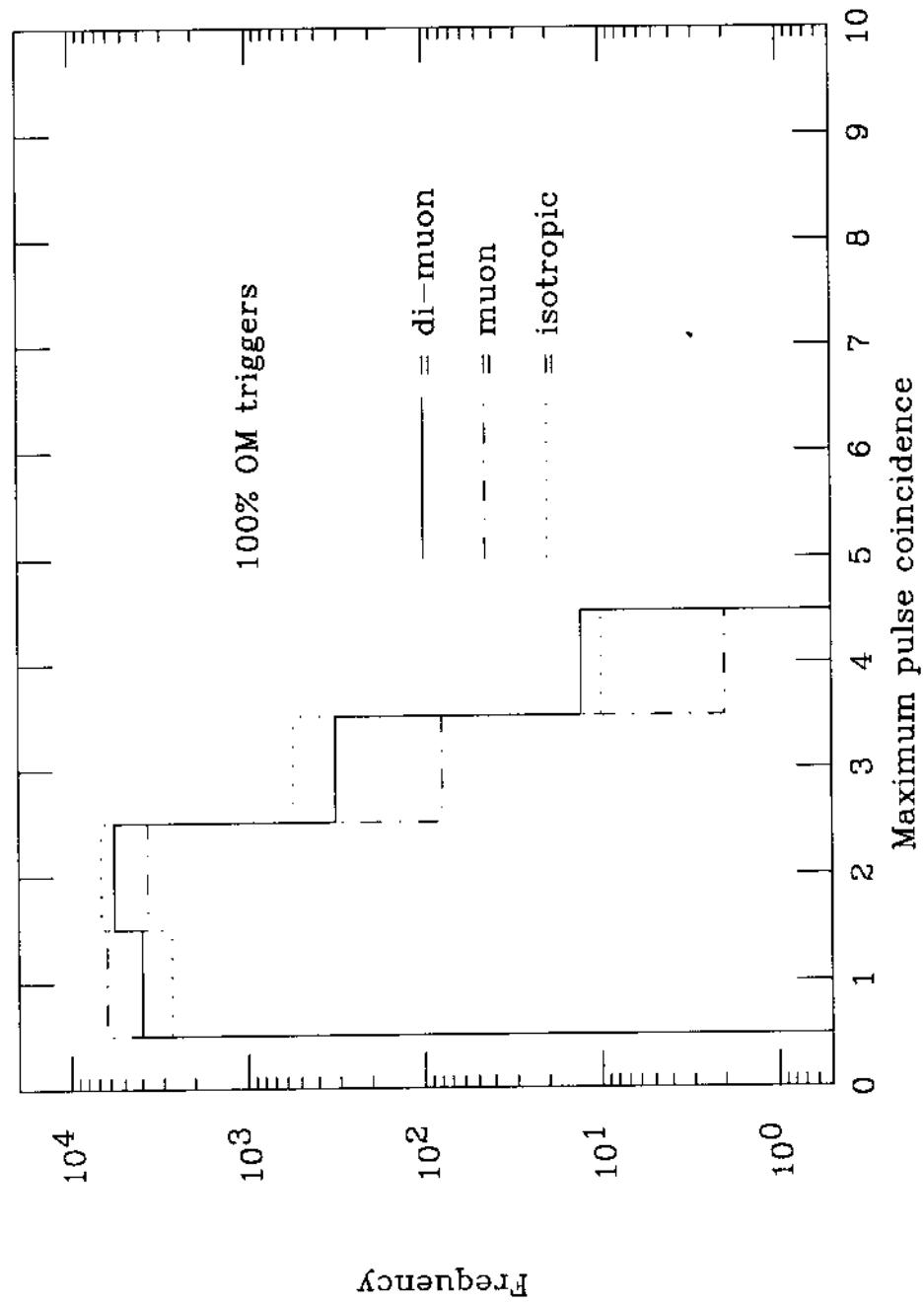


FIGURE 7:

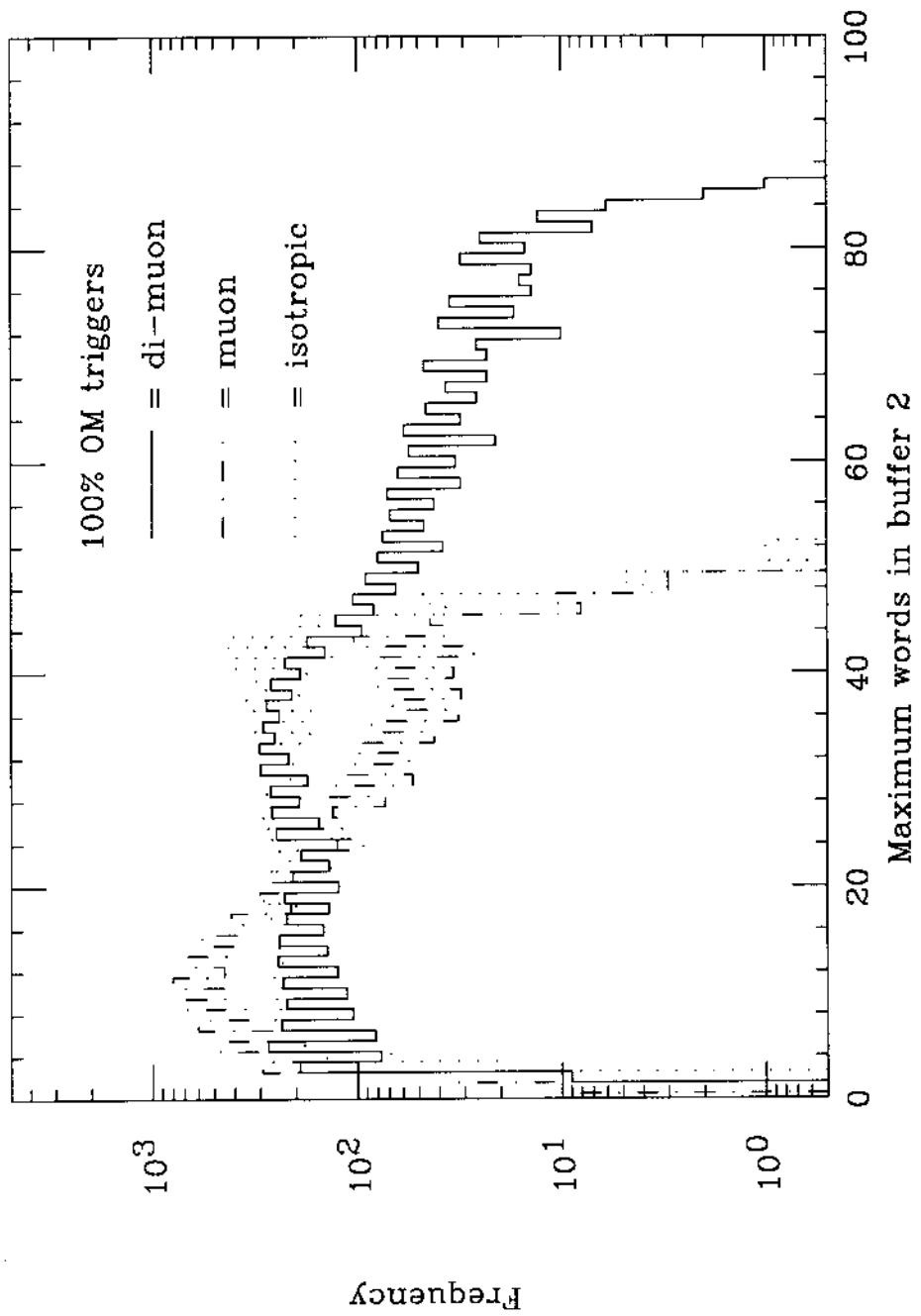


FIGURE 8:

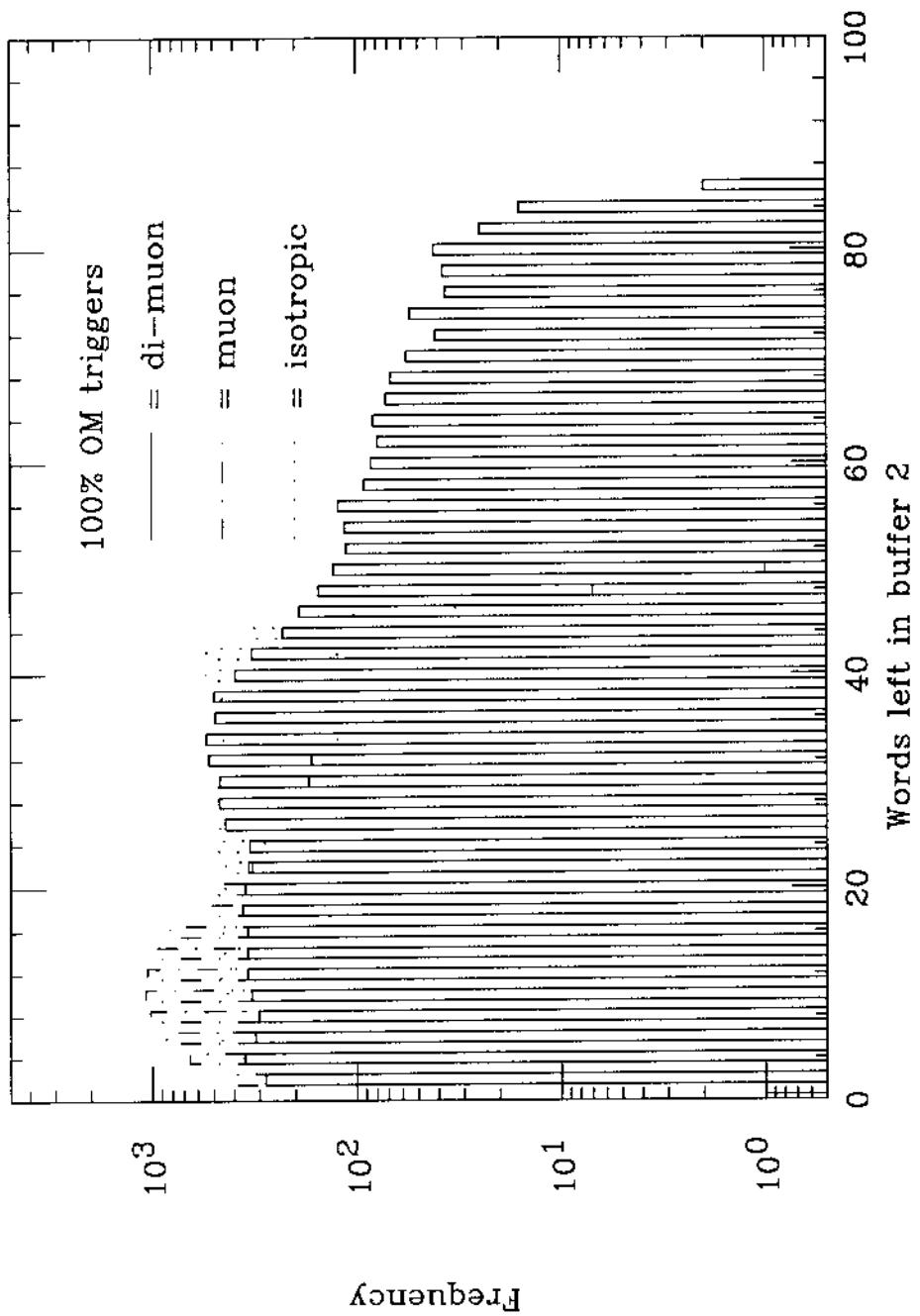


FIGURE 9:

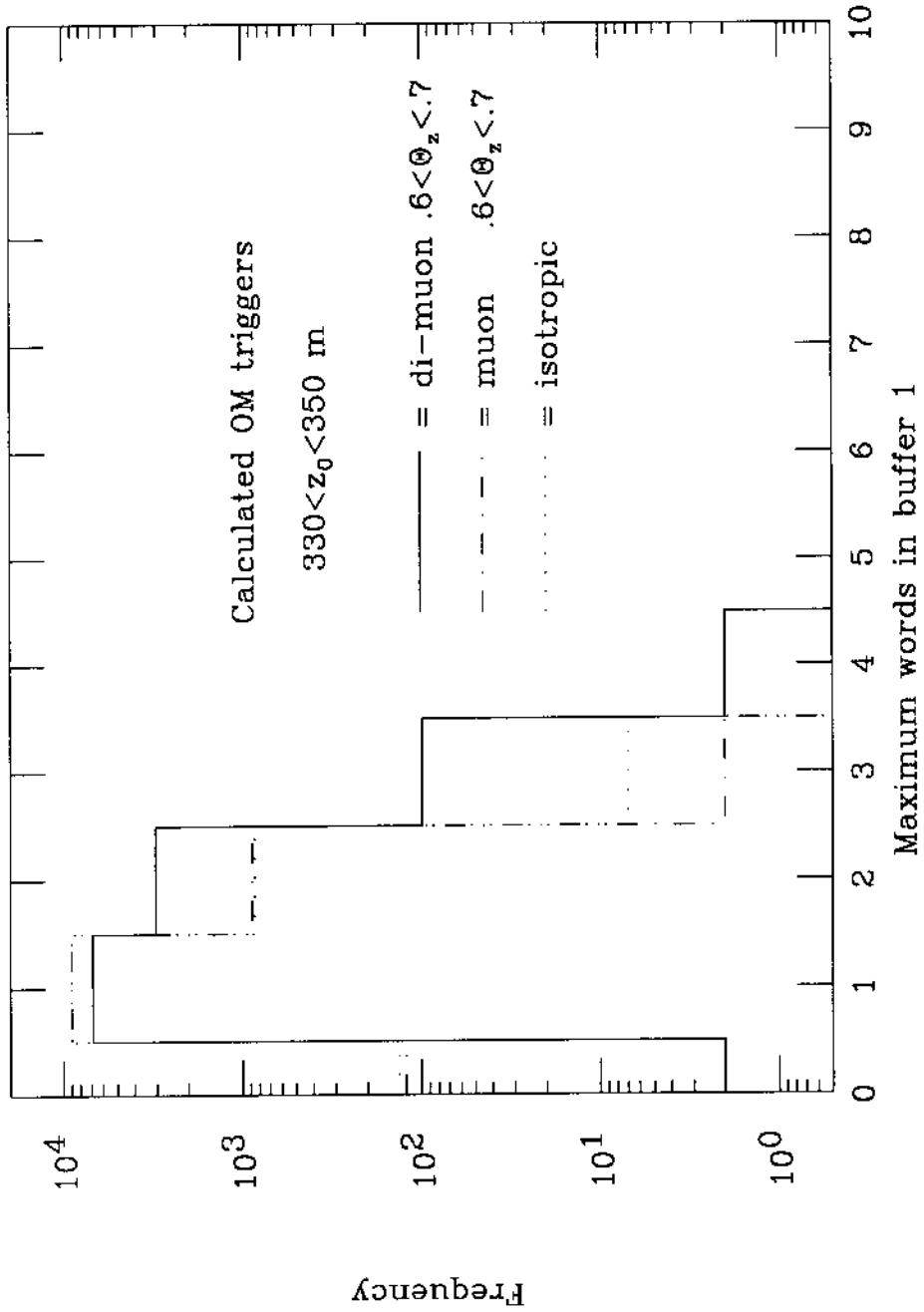
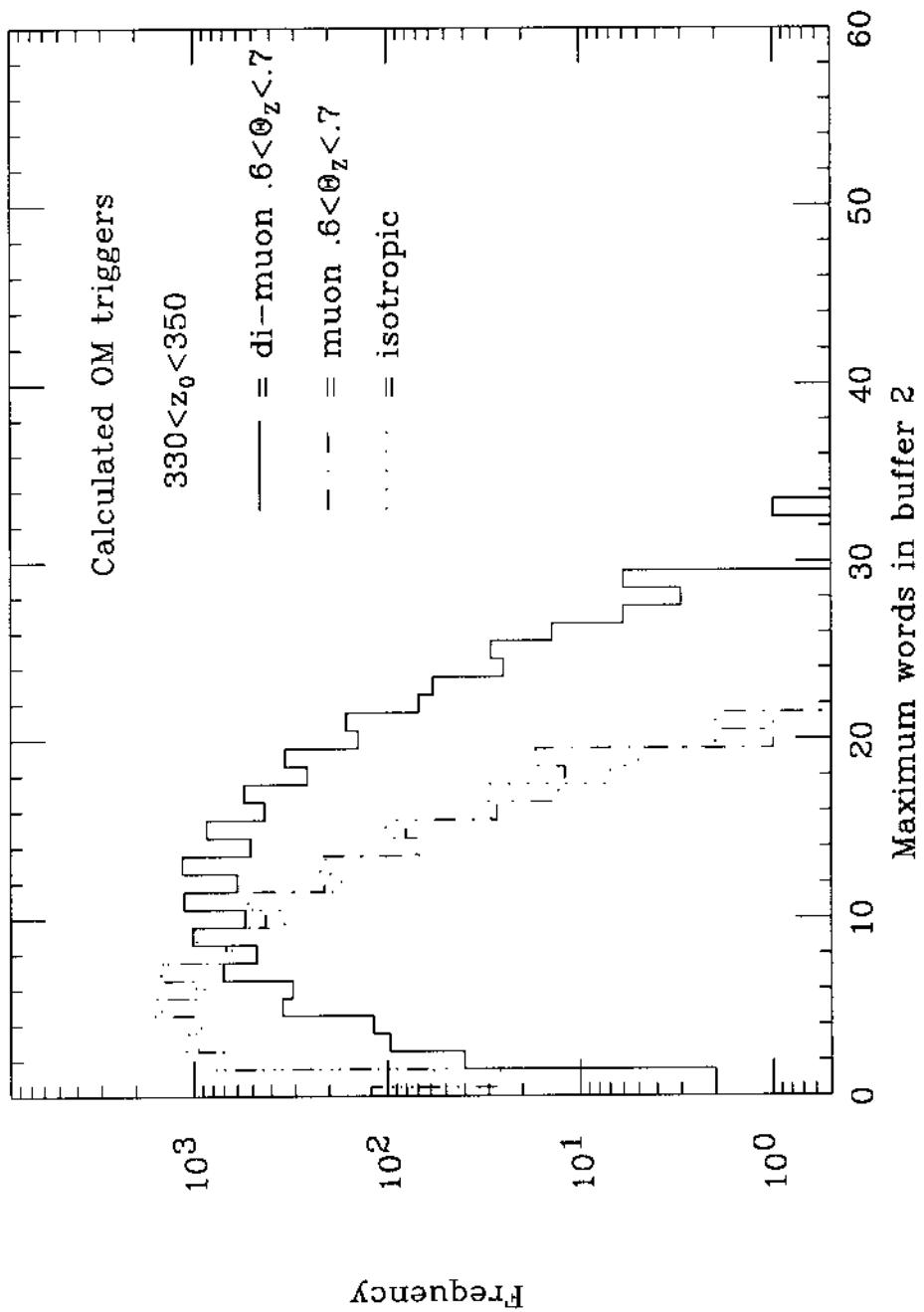


FIGURE 10:



Appendix: Fortran code for simulation program

1

```

program digitizer
common /stuff/ xpm, ypm, zpm, pmtime, pmq
common /worst/ pessimism

parameter brange=2000.0                      !default impact range (cm)
parameter zbot=10000.0                         !default min z-point (cm)
parameter ztop=33000.0                          !default max z-point (cm)
parameter backtrk=10000.0                        !default back track (cm)
data iseed/6783451/
dimension xpm(24), ypm(24), zpm(24)
dimension pmtime(2,24), pmq(2,24)
dimension pos(3), dir(3), start(3), pnext(3)
dimension dgttime(96), datime(96)
integer itime(96), ichan(96)
integer nlarray(0:50), n2array(0:100), idiff(0:24), ileft(0:100)
logical pessimism

* let the user define the type of events to simulate
type *, ' enter light source code: muon=0, isotropic=1'
read(5,6) icode
if((icode.lt.0).or.(icode.gt.1)) stop

if(icode.eq.0) then
  type *, ' select zenith angle: random=0, specific range=1'
  read(5,6) izen
  if((izen.gt.1).or.(izen.lt.0)) stop
  if(izen.eq.1) then
    type *, ' negative is downward, positive is upward'
    type *, ' enter minimum cosine(angle)'
    read(5,7) coszmin
    if(abs(coszmin).gt.1) stop
    type *, ' enter maximum cosine(angle)'
    read(5,7) coszmax
    if(abs(coszmax).gt.1) stop
  end if
  type *, ' muon multiplicity?'
  read(5,6) multi
end if
format(f)

7
type *, zbot,'=bottom default', ztop,'=top default'
type *, ' want to select point of closest approach? no=0, yes=1'
read(5,6) iptclose
if(iptclose.eq.1) then
  type *, ' enter minimum z pt'
  read(5,7) zonbot
  type *, ' enter maximum z pt'
  read(5,7) zontop
end if

type *, ' number of events?'
read(5,6) nevet
format(i)

6
type *, ' enter OM trigger efficiency code: calculate=0, 100%=1'
read(5,6) iomeff
if((iomeff.lt.0).or.(iomeff.gt.1)) stop
if(iomeff.eq.0) pessimism=.false.
if(iomeff.eq.1) pessimism=.true.

type *, ' want topdrawer files? 1=yes, 0=no'
read(5,6) itopd
if((itopd.lt.0).or.(itopd.gt.1)) stop

* read in OM positions
call geometry(zbot,xpm, ypm, zpm)

```

```

itot=0
do while(itot.lt.nevet)
itot=itot+1

* clear all arrays and counters
nlit=0
do i=1,24
  do j=1,2
    pmtime(j,i)=0.
    pmq(j,i)=0.
  end do
end do
do i=1,96
  dgtime(i)=0.
end do

if(icode.eq.0) then                                !begin muon loop

* get muon direction cosines
call pickdir(izen,coszmin,coszmax,dir)

* pick an impact parameter randomly between 0 and 20 meters for muons
bimpac=ran(iseed)*brange

* pick z-coordinate of closest approach between 100 and 330 meters
if(ipclose.eq.0) then
  zclose=zbot+ran(iseed)*(ztop-zbot)
  if(multi.eq.2) zcls2=zbot+ran(iseed)*(ztop-zbot)
end if
if(ipclose.eq.1) then
  zclose=zonbot+ran(iseed)*(zontop-zonbot)
  if(multi.eq.2) zcls2=zonbot+ran(iseed)*(zontop-zonbot)
end if

* start muon 100 meters back on track from point of closest approach
* muons always travels in x,z plane, string on z-axis
start(1)=-backtrk*dir(1)                         !always positive
start(2)=bimpac                                    !always positive
start(3)=zclose-backtrk*dir(3)                     !going back

* follow muon along track until z=zclose in 20 cm steps
iend=int(backtrk)
do i=0,iend,20
  do k=1,3
    pos(k)=start(k)+i*dir(k)
  end do
  tzero=float(i)/30.
  call tubeslit(pos,dir,nlit,tzero)
end do

do m=2,multi                                      !do all muons in cluster
  call cluster(start,dir,zcls2,pnext,range,bimp2)
  do i=0,nint(range),20
    do k=1,3
      pos(k)=pnext(k)+i*dir(k)
    end do
    tzero=float(i)/30.
    call tubeslit(pos,dir,nlit,tzero)
  end do
end do

end if                                              !end muon loop

if(icode.eq.1) then                                !begin isotropic loop
  bimpac=ran(iseed)*brange

```

```

        if(ipclose.eq.0) zclose=zbot+ran(iseed)*(ztop-zbot)
        if(ipclose.eq.1) zclose=zonbot+ran(iseed)*(zontop-zonbot)
        start(1)=0.
        start(2)=bimpac
        start(3)=zclose
        call isolit(start,nlit)
    end if                                !end isotropic loop

* convert OM hit time to time rising pulse arrives at SBC (add 50ns/10m),
* add falling pulse times according to pulse width, and generate noise hits
    call noise(pmtime,pmq,dgtme)

* remove impossible hits, encode channel number, and sort pulse arrival times
    call reality(dgtme,datime,ichan,index)

* zero suppress and convert to digitizer counts
    call encoder(dgtme,datime,index,itime)

* send data stream to chip
    call chip(itime,ichan,index,max1,max2,maxdif,nleft,nmiss)

* check for buffer overflows
    if((nmiss.gt.0).or.(max2.ge.100)) then           ! inspect results
        misses=misses+1
        type *,' zenith cos',dir(3)
        type *,' impact parameter (m)',bimpac/100.,bimp2/100.
        type *,' z-pt of closest approach (m)',zclose/100.,zcls2/100.
        if(multi.eq.2) type *,' range of muon#2 (m)',range/100.
        type *
        type *,' evt#',itot,' #of lit OMs',nlit
        type *,nmiss,'#missed fifo words'
        type *
        type *,' OM pes'
        type *,(pmq(1,n),n=1,24)
        type *
        type *,(pmq(2,n),n=1,24)
        type *
        type *,' OM hit times'
        type *,(pmtime(1,n),n=1,24)
        type *
        type *,(pmtime(2,n),n=1,24)
        type *
        type *,' rising and falling pulses with noise'
        type *,(dgtme(n),n=1,96)
        type *
        type *,' digitized data stream'
        type *,index,' data words'
        do i=1,index
            type *,i,ichan(i),itime(i)
        end do
        type *
        type *,' maximum number of hits in fifo#1',max1
        type *
        type *,' maximum number of hits in fifo#2',max2
        type *
    end if

* fill arrays for histograms
    do i=0,50
        if(i.eq.max1) nlarray(i)=nlarray(i)+1
    end do
    do i=0,100
        if(i.eq.max2) n2array(i)=n2array(i)+1
    end do
    do i=0,24
        if(i.eq.maxdif) idiff(i)=idiff(i)+1
    end do

```

```

    end do
    do i=0,100
        if(i.eq.nleft) ileft(i)=ileft(i)+1
    end do

    end do

* end of run results
type *
type *
type *,' number of events with bugged data =',misses
type *
type*,' frequency of maximum hits in fifo#1'
type *,(nlarray(n),n=0,50)
type *
type*,' frequency of maximum hits in fifo#2'
type *,(n2array(n),n=0,100)
type *
type*,' frequency of maximum state changes'
type *,(idiff(n),n=0,24)
type *
type*,' frequency of #hits left in fifo#2'
type *,(ileft(n),n=0,100)

if(itopd.eq.1) then                                !write Topdrawer data files
    do i=0,50
        write(76,888) float(i),float(nlarray(i))
    end do
    do i=0,24
        write(78,888) float(i),float(idiff(i))
    end do
    do i=0,100
        write(77,888) float(i),float(n2array(i))
    end do
    do i=0,100
        write(79,888) float(i),float(ileft(i))
    end do
end if
888 format(2(5x,f))
stop
end

```

```

subroutine geometry(zbot,xpm,ypm,zpm)
parameter deltaz=1000.0                      !intermodule spacing (cm)
dimension xpm(24),ypm(24),zpm(24)

```

```

* make PMT position array
do i=1,24
    xpm(i)=0.
    ypm(i)=0.
    zpm(i)=zbot+(i-1)*deltaz
end do

return
end

```

```

subroutine pickdir(izen,coszmin,coszmax,dir)
dimension dir(3)
data iseed/6783453/

if(izen.eq.0) then
    dir(3)=ran(iseed)
        if(ran(iseed).lt.0.5) dir(3)=dir(3)*-1.
end if

```

```

if(izen.eq.1) then
    czmin=min(coszmax,coszmin)
    czmax=max(coszmax,coszmin)
    zrange=czmax-czmin
    dir(3)=czmin+ran(iseed)*zrange
end if
dir(2)=0.
dir(1)=-sqrt(1.-dir(3)**2.)           !always negative

return
end

subroutine cluster(start,dir,zcls2,pnext,range,bimp2)
dimension start(3),pos(3),dir(3),dnum(3),pnext(3)
data mseed/8665321/

* find intersection with T0 plane of parallel track passing through new point
dnorm=0                                !find direction numbers
do j=1,3
    dnorm=dnorm+start(j)**2
end do
dnorm=sqrt(dnorm)
dcon=0.
do j=1,3
    dnum(j)=dir(j)*dnorm
    dcon=dcon+start(j)*dnum(j)
end do
bimp2=ran(mseed)*2000.                  !new bimpac
pos(3)=zcls2
pos(2)=bimp2
pos(1)=0.
sum=0.
do j=1,3
    sum=sum+dnum(j)*pos(j)
end do
range=abs((sum-dcon)/dnorm)             !dist from new pt to T0 plane
pnext(1)=-range*dir(1)                  !always positive
pnext(2)=pos(2)                        !always positive
pnext(3)=pos(3)-range*dir(3)           !going back

return
end

subroutine noise(pmtime,pmq,dgtme)
dimension pmtime(2,24),pmq(2,24),dgtme(96)
parameter rnoise=100.                    !OM noise rate in kHz
parameter alpha=24.,beta=46.            !pulse width fitting parameters
data jseed/7345621/

timax=0.
timin=999999.
do i=1,24
    if(pmtime(1,i).gt.0.) then
        dgtme(2*i-1)=pmtime(1,i)+(i-1)*50.      ! rising pulse
        pulwid=alpha+beta*log10(pmq(1,i))
        pulwid=max(pulwid,5.)                      ! minimum of 5 ns
        pulwid=min(pulwid,1000.)                   ! saturate at 1000 ns
        dgtme(2*i)=dgtme(2*i-1)+pulwid          ! falling pulse
        timin=min(dgtme(2*i-1),timin)
        timax=max(dgtme(2*i),timax)
    end if
    if(pmtime(2,i).gt.0.) then
        dgtme(2*i+47)=pmtime(2,i)+(i-1)*50.     ! rising pulse
        pulwid=alpha+beta*log10(pmq(2,i))
    end if
end

```

```

        pulwid=max(pulwid,5.)                      ! minimum of 5 ns
        pulwid=min(pulwid,1000.)                     ! saturate at 1000 ns
        dgttime(2*i+48)=dgttime(2*i+47)+pulwid     ! falling pulse
        timin=min(dgttime(2*i+47),timin)
        timax=max(dgttime(2*i+48),timax)

    end if
end do

* add noise
do i=1,24
    tau=-log(1.-ran(jseed))/(rnoise*(1.e-6))      !OM time of noise hit
    time=taut+(i-1)*50.                            !digitizer time
    if((time.lt.timax).and.(time.gt.timin-100.)) then
        if(pmptime(1,i).eq.0.) then
            dgttime(2*i-1)=time                   ! rising pulse
            pulwid=alpha+beta*aalog10(1.+ran(jseed)*0.5)
            pulwid=max(pulwid,5.)                  ! minimum of 5 ns
            pulwid=min(pulwid,1000.)                ! saturate at 1000 ns
            dgttime(2*i)=dgttime(2*i-1)+pulwid    ! falling pulse
        else
            if(pmptime(2,i).eq.0.) then
                dgttime(2*i+47)=time             ! rising pulse
                pulwid=alpha+beta*aalog10(1.+ran(jseed)*0.5)
                pulwid=max(pulwid,5.)          ! minimum of 5 ns
                pulwid=min(pulwid,1000.)        ! saturate at 1000 ns
                dgttime(2*i+48)=dgttime(2*i+47)+pulwid  ! falling pulse
            end if
        end if
    end if
end do
return
end

```

```

        end if
    end if
end do

* zero array, collect hits, and encode channel
index=0
do i=1,96
    datime(i)=0.
    if(dgttime(i).gt.0.) then
        index=index+1
        datime(index)=dgttime(i)
        if(i.lt.49) then
            nchan=int((i+1)/2.)
        else
            nchan=int((i-47)/2.)
        end if
        ichan(index)=nchan
    end if
end do

* use old sort routine
DO I=1,index      !LOOP THRU LOOKING FOR 1ST SMALLEST, ETC
nchannel=i
NSMALLEST=I      !INDEX OF SMALLEST IN REST OF LIST
SMALL=datime(I)      !SMALLEST time
nchan=ichan(i)
DO J=I+1,index  !LOOP THRU REST OF LIST LOOKING FOR BEST
    IF(datime(J).LT.SMALL) then
        NSMALLEST=J      !THIS IS NEW SMALLEST
        SMALL=datime(J)
        nchannel=j
        nchan=ichan(j)
    ENDIF
ENDDO
TEMP=DATIME(I)      !SWAP THE SMALLEST WITH THE NEXT PLACE IN LIST
DATIME(I)=DATIME(NSMALLEST)
DATIME(NSMALLEST)=TEMP
interim=ichan(i)
ichan(i)=ichan(nchannel)
ichan(nchannel)=interim

C NOW FIND THE TAPE ON INPUT FILE & DUMP IT UNLESS IT IS A DUPLICATE
IF(I.EQ.1 .OR.DATIME(I).NE.DATIME(I-1)) THEN !IS EVENT NON-DUPLICATE?
c      TYPE *,,'SORTED EVT#',I,' IS TIME',DATIME(I)
      GO TO 200
ELSE      !EVENT WAS A DUPLICATE
      TYPE *
      TYPE *,,'>>> DUPLICATE ENTRY #',I,' TIME',DATIME(I),DATIME(I-1)
      TYPE *
ENDIF
200 ENDDO
      RETURN
END

```

```

subroutine encoder(dgttime,datime,index,itime)
dimension dgttime(96),datime(96)
integer itime(96)

```

```

* suppress zero and convert from ns to digitizer counts
do i=1,index
    itime(i)=nint((datime(i)-datime(1))/1.024)+1
end do

return
end

```

```

subroutine chip(itime,ichan,index,max1,max2,maxdif,nleft,nmiss)
parameter if1=48,if2=100                                !buffer depths
parameter it1=3,it2=80                                 !readout times
integer itime(96),ichan(96)
integer ififo1(50),ififo2(if2+4),hotrod(if2+4)
integer*4 channel
logical scramble,full1,full2

* initialize counter and buffer values
nhtrd=0
do i=1,104
  ififo2(i)=0
  hotrod(i)=0
end do
do i=1,50
  ififol(i)=0
end do
icount=1
nmax1=0
max1=0
nfl=0
ibuf1=0
ibuf2=0
max2=0
nf2=0
maxdif=0
full1=.false.
full2=.false.
nmiss=0

* process data stream
do i=1,itime(index)                                     !loop through the times
  ibufl=ibufl+1                                         !increment timer1
  ibuf2=ibuf2+1                                         !increment timer2
  nchng=0                                                 !zero change counter
  channel=0                                              !clear state register
  do while(i.eq.itime(icount))                           !we got a hit
    channel=jiset(channel,ichan(icount))                !set channel bit
    icount=icount+1                                       !bump hit counter
  end do
  if(channel.ne.0) then                                  !at least 1 channel state changed
    if(((nmax1.eq.if1).and.(ibufl.lt.it1))
+      .or.((nmax1.gt.if1).and.(ibufl.eq.it1))) then
      nmiss=nmiss+1                                      !count misses
      do n=24,1,-1                                       !check each channel bit
        if(bjtest(channel,n)) then                      !channel n changed
          nchng=nchng+1                                  !count how many changes
        end if
      end do
      type *,i,' *** FIFO #1 Overflow ***'
      type *,nmiss,' misses',nchng,' channel hits'
      full1=.true.                                       !raise full flag#1
    end if
    if(.not.full1) then                                 !OK to process
      do n=24,1,-1                                       !check each channel bit
        if(bjtest(channel,n)) then                      !channel n changed
          nchng=nchng+1                                  !count how many changes
          nfl=nfl+1                                     !bump fifo #1 index
          ififol(nfl)=i                                !count#=~time
        end if
      end do
      nmax1=nmax1+1                                     !bump fifol word counter
    end if
  end if
end if

```

```

maxdif=max(nchng,maxdif)                                !find max# of state changes

    if(ibuf1.eq.it1) then
* time to read a word, if available, into fifo#2
        ibuf1=0                                         !reset counter
        if(nf1.ge.1) then                               !send a word to fifo#2
            if(((nf2.eq.if2).and.(ibuf2.lt.it2))
+                .or.((nf2.gt.if2+1).and.(ibuf2.eq.it2))) then
                type *,i,' *** OVERFLOW FIFO #2 ***'
                full2=.true.                            !raise full flag#2
            end if
            if(.not.full2) then
                nf2=nf2+1                           !bump fifo #2 counter
                ififo2(nf2)=ififo1(1)                  !count#=~time
            end if
            do k=1,nf1
                ififo1(k)=ififo1(k+1)                !cascade fifo#1
            end do
            nf1=nf1-1                                !bump fifo#1 index
        end if
        if(nf1.eq.0) then
            if(nmax1.ge.1) then
                nmax1=nmax1-1
                if(nmax1.le.if1) full1=.false.      !reset full flag#1
            end if
        else
            if(ififo1(1).ne.ififo2(nf2)) then      !working on new time
                if(nmax1.ge.1) then
                    nmax1=nmax1-1                  !bump down max1 counter
                    if(nmax1.le.if1) full1=.false.  !reset full flag#1
                end if
            end if
        end if
        end if
        max1=max(nmax1,max1)                        !find max# in fifo#1

        if(ibuf2.eq.it2) then
* time to check buffer#2 for words to go to Hotrod
            ibuf2=0
            if(nf2.ge.2) then
                do k=1,2
                    nhtrd=nhtrd+1
                    hotrod(nhtrd)=ififo2(k)
                end do
                do k=1,nf2
                    ififo2(k)=ififo2(k+2)          !cascade fifo#2
                end do
                nf2=nf2-2
                if(nf2.lt.if2) full2=.false.    !bump fifo#2 counter
            end if
        end if
        max2=max(nf2,max2)                          !find max# in fifo#2
    end do

* compare output data with input
    scramble=.false.
    do i=1,nhtrd
        if(itime(i).ne.hotrod(i)) then
            if(.not.scramble) then
                type *,' scrambled or missing data in Mr. Hotrod'
            end if
            scramble=.true.
        end if
    end do
    do i=1,nf2

```

```

        if(itime(nhtrd+i).ne.ififo2(i)) then
            if(.not.scramble) then
                type *,' scrambled or missing data left in FIFO#2'
            end if
            scramble=.true.
        end if
    end do
    if(scramble) then
        type *
        type *,' scrambled or incomplete data stream printout'
        do i=1,index
            type *,i,ichan(i),itime(i),hotrod(i)
        end do
    end if

* find number of hits left in fifo#2
    nleft=index-nhtrd

    return
end

```

```

*****
SUBROUTINE TUBESLIT(POS,DIR,NLIT,TZERO)
*****

```

C FLAGS TUBES ILLUMINATED BY TRACK SEGMENT

C POS(3) GIVES TRACK START POSITION, DIR(3) GIVES DIRECTION COSINES,
C RLENGTH GIVES LENGTH OF TRACK SEGMENT

```

common /stuff/ xpm, ypm, zpm, pmtime, pmq
dimension xpm(24), ypm(24), zpm(24)
dimension pmtime(2,24), pmq(2,24)
DIMENSION POS(3), DIR(3), DIRCOS(3)
PARAMETER COTANTH=1.13 !COTAN OF CERENKOV ANGLE
PARAMETER SINTH=0.66 !SINE OF CERENKOV ANGLE
PARAMETER RLEN=20. !SHORT TRACK SEGMENT
PARAMETER VH2O=22.56 !LIGHT SPEED IN H2O (cm/ns)
* OM response function parameters 2x estimate from Clem thesis figure 24.
PARAMETER A=3700.,B=.00015,C=2.66 !for distance in cm
LOGICAL FIRE

```

```

* NORMALIZE DIRECTION COSINES IF POSSIBLE
DNORM=DIR(1)**2+DIR(2)**2+DIR(3)**2
IF(DNORM.EQ.0)THEN
    TYPE *,'!!SPECIFIED DIRECTION VECTOR IS ZERO, ASSHOLE!!'
    RETURN
ELSE
    DO I=1,3
        DIRCOS(I)=DIR(I)/DNORM           !NORMALIZE DIRCOSINES
    END DO
END IF

```

```

* LOOP THRU & FIND TUBES INSIDE RING
DO I=1,24
    DX=XPM(I)-POS(1)                  !GET VECTOR FROM PT TO TUBE
    DY=YPM(I)-POS(2)
    DZ=ZPM(I)-POS(3)
    D2=DX**2+DY**2+DZ**2             !DIST**2 TO TUBE
    DP=DX*DIRCOS(1)+DY*DIRCOS(2)+DZ*DIRCOS(3) !DOT PROD
    PERP=SQRT(ABS(D2-DP**2))
    DIST=DP-COTANTH*PERP
    *** !DISTANCE PROJ. BACK ALONG TRACK TO POINT
    IF(DIST.GT.0.AND.DIST.LT.RLEN) THEN
        IF(PMTIME(1,I).EQ.0.) THEN      !FIRST HIT SO RECORD TIME (ns)

```

```

PHOTON=PERP/SINTH           !PHOTON DIRECT FLIGHT DISTANCE
COSPOL=(POS(3)-ZPM(I))/PHOTON
QPE=C*(1.+(A/PHOTON)*EXP(-B*PHOTON)) !ESTIMATE OF Q (pe)
CALL OMTRIG(QPE,COSPOL,FIRE)
IF(FIRE) THEN
    NLIT=NLIT+1           !COUNT OF HIT TUBES
    CALL JITTER(QPE,TJIT)
    TIME=PHOTON/VH2O+TZERO+TJIT
    PMTIME(1,I)=TIME
    PMQ(1,I)=QPE
END IF
ELSE
    IF(PMTIME(2,I).EQ.0.) THEN !SECOND HIT SO RECORD TIME (ns)
        PHOTON=PERP/SINTH           !PHOTON DIRECT FLIGHT DISTANCE
        COSPOL=(POS(3)-ZPM(I))/PHOTON
        QPE=C*(1.+(A/PHOTON)*EXP(-B*PHOTON)) !ESTIMATE OF Q (pe)
        CALL OMTRIG(QPE,COSPOL,FIRE)
        IF(FIRE) THEN
            CALL JITTER(QPE,TJIT)
            TIME=PHOTON/VH2O+TZERO+TJIT
            PMTIME(2,I)=TIME
            PMQ(2,I)=QPE
        END IF
    END IF
END IF
END DO
RETURN
END

```

```
*****
SUBROUTINE ISOLIT(POS,NLIT)
*****
```

C FLAGS TUBES ILLUMINATED BY TRACK SEGMENT

C POS(3) GIVES TRACK START POSITION, DIR(3) GIVES DIRECTION COSINES,
C RLENGTH GIVES LENGTH OF TRACK SEGMENT

```

common /stuff/ xpm, ypm, zpm, pmtime, pmq
dimension xpm(24), ypm(24), zpm(24), pmtime(2,24), pmq(2,24)
DIMENSION POS(3), DIR(3), DIRCOS(3)
PARAMETER COTANH=1.13 !COTAN OF CERENKOV ANGLE
PARAMETER SINTH=0.66 !SINE OF CERENKOV ANGLE
PARAMETER RLEN=20. !SHORT TRACK SEGMENT
PARAMETER VH2O=22.56 !LIGHT SPEED IN H2O (cm/ns)
* OM response function parameters 2x estimate from Clem thesis figure 24.
PARAMETER A=3700., B=.00015, C=2.66
LOGICAL FIRE

```

* LOOP THRU & FIND TUBES LIT TIMES

```

DO I=1,24
    IF(PMTIME(1,I).EQ.0.) THEN           !FIRST HIT SO RECORD TIME (ns)
        DX=XPM(I)-POS(1)                 !GET VECTOR FROM PT TO TUBE
        DY=YPM(I)-POS(2)
        DZ=ZPM(I)-POS(3)
        DIST=SQRT(DX**2+DY**2+DZ**2)     !DIST TO TUBE
        QPE=C*(1.+(A/DIST)*EXP(-B*DIST)) !ESTIMATE OF Q (pe)
        COSPOL=(POS(3)-ZPM(I))/DIST
        CALL OMTRIG(QPE,COSPOL,FIRE)
        IF(FIRE) THEN
            NLIT=NLIT+1           !COUNT OF HIT TUBES
            TIME=DIST/VH2O
            CALL JITTER(QPE,TJIT)
            PMTIME(1,I)=TIME+TJIT
        END IF
    END IF
END DO

```

```

        PMQ(1,I)=QPE
    END IF
ELSE
    IF(TIME.LT.PMTIME(2,I)) THEN      !RECORD TIME (ns)
        DX=XPM(I)-POS(1)           !GET VECTOR FROM PT TO TUBE
        DY=YPM(I)-POS(2)
        DZ=ZPM(I)-POS(3)
        DIST=SQRT(DX**2+DY**2+DZ**2) !DIST TO TUBE
        QPE=C*(1.+(A/DIST)*EXP(-B*DIST)) !ESTIMATE OF Q (pe)
        IF(QPE.LT.1.) QPE=1.
        COSPOL=(POS(3)-ZPM(I))/DIST
        CALL OMTRIG(QPE,COSPOL,FIRE)
        IF(FIRE) THEN
            TIME=DIST/VH2O
            CALL JITTER(QPE,TJIT)
            PMTIME(2,I)=TIME+TJIT
            PMQ(2,I)=QPE
        END IF
    END IF
END IF
END DO
RETURN
END

```

```

subroutine omtrig(qpe,cospol,fire)
common /worst/ pessimism
logical pessimism

```

```

* angular response function parameters from S. Matsuno et al., 1989.
PARAMETER APOL=.52,BPOL=.48          !ANGULAR EFFICIENCY PARAMETERS
LOGICAL FIRE
data mseed/8664533/

POLEFF=APOL-BPOL*COSPOL             !POLAR ANGLE EFFICIENCY
qpe=qpe*poleff
if(pessimism) then                  !every possible tube fires
    fire=.true.
    qpe=max(qpe,1.)
    return
end if

* corrected Q (pe) is mean of Poisson distribution
* find P(n>0)=1-exp(-Q)
qpe=max(qpe,0.001)
prob=exp(-qpe)
if(ran(mseed).gt.prob) then
    fire=.true.
else
    fire=.false.
end if
* return corrected Q value, but not less than 1
qpe=max(qpe,1.)

return
end

```

```

subroutine jitter(qpe,tdly)

```

```

parameter fwhm=6.                      !Hamamatsu 15" PMT FWHM jitter (ns)
* Propagation delay fit parameters taken from S. Matsuno et al., 1989
parameter a=40.,b=10.                   !time delay fit parameters (ns)
data nseed/8456321/

```

```

tjit=fwhmalog(1.+qpe)

```

```
c      sigma=tjit/2.35
* find randomly distributed Gaussian variable
* cheat for now
  tjit=2.*tjit*ran(nseed)*ran(nseed)*ran(nseed)

* add jitter time to delay time
  tdly=(a-b*log10(qpe))+tjit

  return
end
```