Dsimulator processes

Read file names:

File names are assigned to variables:

Object file -> objfile

Output file -> output

Mask design file -> mdf

Optional plot file -> plotfile

Is the file in standard format -> Boolean std\_format

## Read in telescope parameters:

Data\_init(indat)

This procedure is in dsim2.x

This creates a complex structure called **indat**, which contains the following fields

The parameters used by this procedure are:

Ra0, dec0, ha0, temp pressure, lambda\_cen, blue, red, sep\_slit, min\_slit, box\_sz, slit\_width, equinox, proj\_len, no\_overlap, maskid, guiname, observer, author, project, dateovs, DEIMOS

These are assigned to similar fields in the indat structure.

Example:

RA0\_FLD = DEGTORAD(15. \* ra0)

## This is followed by reading in the list of targets

Three file handles are open:

Fda= object file

Fdb = output file

Fdc = mask design file

The readin is done by the procedure targ\_init(fda, tdat, ntarg, indat)

Number of targets is a line count of fda

The structure tdat is then created and allocated, containing the following fields:

Index, ra0, dec0, ra, dec, pa, len1, len2, wid, code, sampl, stat, sel, xarc, yarcs, relpa, x1, y1, x2, y2, mag, band, slndx, line,

The size of the memory block is controlled by the number of targets

The ingestion loop is:

Get the first character, if it is 0 or #, skip the line

Read the mask center (idstr, alpha, delta, equinox, workstr, PA if it exists)

Convert alpha and delta into RA0\_FLD, and DEC0\_FLD, add them to the indat structure together with the equinox

Read following lines (idstr, alpha, delta, equinox, magn, passpand, prior

If less than 7 fields, complain and skip

Other fields: nlist, selcode, pangle, l1, l2

Count the number of arguments

If n <12, l1 and l2 are set to default indat(DEF\_HLEN)

If n<10, pangle= indef

If n<9, selcode=NO

If n<8, nlist=PRIMARY

If prior = CODE\_AS (whch is -2, defined in deimos.h) this is an alignment star, so set l1 and l2 to be the default for alingmnet stars (4 arcseconds, ldat(DEF\_BOXR))

Read the entire line of data into a new variable tdat(DATLINE), not sure why

Reset the equinox to the standard equinox if another one is specified, no support for precession

Tdat(INDEX)=current index

Tdat(RA0) = DEGTORAD(alpha\*15)

Tdat(DEC0)=DEGTORAD(delta)

Tdat(MAG)=magn

Truncate passband to 1 char and assign to tdat(PBAND)

Tdat(PBAND)=prior

Tdat(sampl)=nlist

Tdat(sel)=selcode

If pangle is indef, tdat(PA)=indef, otherwise degtorad(pangle)

Tdat(len1)=l1

Tdat(len2)=l2

If tdat(PCODE)== -2 (CODE\_AS):

Tdat(PA)=indef

Tdat(slwid) = 2 \* indat(DEF\_BOXR)

Else:

Tdat(slwid) = indat(DEF\_SLWID)

Increase index

## Refraction

Next step is refract the coordinates

Procedure refr\_coords (tdat, ntarg, indat) in dsim3.x

Calculate LST of the observation: lst =indat(RA0\_FLD)+indat(HA\_FLD)

Define parameters:

Lat = DEGTORAD(OBS\_LAT)

Htm = OBS\_ALT

Tdk = indat(TEMP) + 273.15

Pmb = indat(PRES)

Rel\_h20=OBS\_RH

W=indat(WAVER)

Calculate refraction coefficients using SLAlib functions

For the rslrfco routine, see refco.f in the dsimulator directory

It returns a and b in dZ=A tan Z + B tan\*\*3 Z

They are called r1 and t3 here, and assigned to the structure indat(REF1) and indat(REF3)

Apply to the field center

Ha = lst-indat(RA0\_FLD)

Use slde2h to transform ha, dec, latitude into Az, elevation. See de2h.f in dsimulator directory

Zd0=HALFPI – el

Use slrefz to adjust unrefracted zenith distsance to include the effect of atmospheric refraction

Zd = slrefz(zd0, r1, r3) see refz.f

El = HALFPI-zd

Use sldh2e to convert az, el to equatorial

Ha, indat(DEL\_FLD) = dh2e(Az, el, latitude)

Indat(RA\_FLD)=lst-ha

Loop and apply to targets:

For each targ:

Ha = lst-tdat(RA0,i)

Az,el = de2h(ha, tdat(DEC0,i), lat)

Zd=halfpi – el

Zd=refz(zd,r1,f3)

El=halfpi-zd

Ha,tdat(DEC,i) = dh2e(az, el, lat)

Tdat(RA,i)=lst-ha

Now workout atmospheric dispersion:

Zd=refz(zd0,r1,r3)

W1=indat(wavemn)

W2=indat(wavemx)

W is the reference walength

A,b = slatmd(tdk,pmb,rel\_h20, w, r1, r3, w1)

Zd1 = refz(zd0, a,b)

A,b = slatmd(tdk,pmb,rel\_h20,w,r1,r3,w2)

Zd2 = refz(zd0,a,b)

Calculate amount

Indat(ad1) = (zd1-zd)\*206205

Indat(ad2) = (zd2-zd)\*206205

Parallactic angle

Indat(par\_ang) = slpa(indat(HA\_FLD), indat(DEC\_FLD), lat) (see pa.f)

Airmass

Indat(amass) = slarms(zd) (see airmas.f)

## Calculate position with reference to the telescope axis

Function fld2telax in dsim.x

This function converts field center and PA into coordinates of telescope axis

R = degtorad(sqrt(fldcen\_x\*\*2 + fldcen\_Y\*\*2)/3600)

Fldcen\_x and fldcen\_Y are defined in dsimulator.h as being (0, 270)

Pa\_fld = atan2(fldcen\_y, fldcen\_x)

Cosr = cos(r)

Sinr = sin(r)

Cosd = cos(indat(DEC\_FLD))

Sind = sin(indat(DEC\_FLD))

Cost = cos(indat(PA\_ROT) – pa\_fld)

Sint = sin(indat(PA\_ROT)-pa\_fld)

Sina = sinr\*sint/cosd

Cosa = sqrt(1-sina\*\*2)

Indat(RA\_TEL) = indat(RA\_FLD)-asin(sina)

Indat(DEC\_TEL)=asin((sind\*cosd\*cosa- cosr\*sinr\*cost)/(cosr\*cosd\*cosa-sinr\*sind\*cost))

## Calculate positions respect to the telescope axis

Using function tel\_coords (tdat, ntarg, indat) defined in sim2.x

Long function, I am going to report it here:

ra0 = RA\_TEL(indat)

dec0 = DEC\_TEL(indat)

pa0 = PA\_ROT(indat)

do i = 0, ntarg-1 {

dec\_obj = DEC(tdat,i)

del\_ra = RA(tdat,i) - ra0

cosr = sin (dec\_obj) \* sin (dec0) +

cos (dec\_obj) \* cos (dec0) \* cos (del\_ra)

r = acos (cosr)

sinp = cos (dec\_obj) \* sin (del\_ra) / sqrt (1. - cosr\*cosr)

cosp = sqrt (max ((1. - sinp\*sinp), 0.))

if (dec\_obj < dec0)

cosp = -cosp

p = atan2 (sinp, cosp)

# For now, convert radii to arcsec XXX

# XXX NB: I am not sure this is correct! We should still be on SPHERICAL surf.

# but these are EUCLIDEAN relations. Options: work in spherical coord

# OR work in tan projection.

# More: The difference at 10 arcmin between the tan and angle is < 0.002 arcsec

# If we convert "r" to tan(r) we should have the tan projection

r = tan(r) \* 206264.8

# r = RADTODEG(r) \* 3600.

XARCS(tdat,i) = r \* cos (pa0 - p)

YARCS(tdat,i) = r \* sin (pa0 - p)

if (PA(tdat,i) == INDEF) {

RELPA(tdat,i) = INDEF

rangle = 0.

} else {

RELPA(tdat,i) = PA(tdat,i) - pa0

rangle = RELPA(tdat,i)

}

# For simplicity, we calculate the endpoints in X here; note use of FLIP

xgeom = FLIP \* cos (rangle)

ygeom = sin (rangle)

if (PROJ\_LEN(indat) == YES) {

xgeom = xgeom / abs (cos (rangle))

ygeom = ygeom / abs (cos (rangle))

}

# We always want X1 < X2, so:

if (xgeom > 0) {

X1(tdat,i) = XARCS(tdat,i) - LEN1(tdat,i) \* xgeom

Y1(tdat,i) = YARCS(tdat,i) - LEN1(tdat,i) \* ygeom

X2(tdat,i) = XARCS(tdat,i) + LEN2(tdat,i) \* xgeom

Y2(tdat,i) = YARCS(tdat,i) + LEN2(tdat,i) \* ygeom

} else {

X2(tdat,i) = XARCS(tdat,i) - LEN1(tdat,i) \* xgeom

Y2(tdat,i) = YARCS(tdat,i) - LEN1(tdat,i) \* ygeom

X1(tdat,i) = XARCS(tdat,i) + LEN2(tdat,i) \* xgeom

Y1(tdat,i) = YARCS(tdat,i) + LEN2(tdat,i) \* ygeom

}

# Calc STAT

x = XARCS(tdat,i) # unclear if XYARCS will be real or dbl

y = YARCS(tdat,i)

STAT(tdat,i) = chk\_stat (x, y, YES)

}

note that here we have the function stat(tdat,i), which checks if an object is within the mask

this is the function

#

## CHK\_STAT: is object within the slitmask? REAL INPUTS; INSTRUMENT SPECIFIC

## This could also be replaced by a bunch of limiting curves

## Note that some slit info should be passed along, also -- width, tilt, ends

#

int procedure chk\_stat (x, y, full\_check)

real x, y

int full\_check

real r

begin

r = sqrt (x\*x + y\*y)

# Is object within 10 arcmin radius?

if (r > 600.)

return (NO)

# inner edge of mask

if (y < YMSKMIN)

return (NO)

# outer edge of mask

if (y > YMSKMAX)

return (NO)

# outer edge of mask

if (x > XUPP\_LIM || x < XLOW\_LIM)

return (NO)

# cut corner

if (x > -0.98273 \* y + 833.0)

return (NO)

if (full\_check == NO)

return (YES) # OK to put slit there

# within radius of camera obscuration/vignetting?

if (x\*x+(y-YCAMCEN)\*\*2 < RADVIGN\*\*2)

return (NO)

# near gaps in mosaic?

# XXX needs definition; things like this should be contained in defines and

# limits (how close) in parameters

# if (abs (x+250.) < 4. || abs (x) < 4. || abs(x-250.) < 4.)

if (abs (x-GAP1CEN) < GAP1HWD)

return (NO)

if (abs (x-GAP2CEN) < GAP2HWD)

return (NO)

if (abs (x-GAP3CEN) < GAP3HWD)

return (NO)

# appears OK...

return (YES)

Following this there is a long section that deals with non-standard format objects. It seems to me as if it is trying to load fits files, or files generated by somebody called marc davies at Berkley…. I won’t go into this.

## Graphics interface

The next step is to launch the graphics interface. We will not go into that, except to try to understand the output

My guess is that using the tdat and idat structures, the procedure returns a new structure called sdat (selected), and a new number of slits (nslit).

Steps are:

### Read mask outline

Procedure get\_outlines, defined in dsimgraph.x, which reads the file src/foc\_plane.dat

It creates a structure called mdat, with elements PTFPLX, PTFPLY, PTFPLZ, PTFPWIN

### Draw the focal plane

Function fp\_layout,

This function:

* Calculates the boundaries of the display window based on mdat(FPWIN\_X) and mdat(FPWIN\_Y)
* Sets background colors
* Draws the fp boundaries, essentially the values read from foc\_plane.dat
* Uses mark\_obj to plot the objects
* If slits are defined, uses mark\_slit to plot the slits
* Plots auxiliary plots

### Mark objects

Calculations:

This is FOR EACH TARGET

X = FLIP\*tdat(XARCS,i)

Y = tdat(YARCS,i)

Xa=FLIP\*Tdat(X1,i)

xb = FLIP\*Tdat(X2,i)

ya=tdat(Y1,i)

yb=tdat(Y2,i)

Assign colors:

If object is ok:

Green for regular objects

Magenta for Alignment stars

Magenta for guide stars

Blue for additional objects (non primary)

If object not ok:

Red

If the object is selected (tdat(sel,i)=Yes), white

If pcode is CODE\_AS (-2, alignment star), draw a box around it

If it is a guide star, draw an asterisk

This part I don’t understand:

if (PCODE(tdat,i) == CODE\_AS) {

szbx = -1. \* abs (xa-xb) # note negative

call gmark (gp, x, y, GM\_BOX, szbx, szbx)

} else if (PCODE(tdat,i) == CODE\_GS) {

call gmark (gp, x, y, GM\_CROSS+GM\_PLUS, -5., -5.)

} else {

call gamove (gp, xa, ya)

call gadraw (gp, xb, yb)

if (SEL(tdat,i) == YES)

call gmark (gp, x, y, GM\_BOX, -1.\*SLWID(tdat,i), -1.\*SLWID(tdat,i)) # TMP

else

call gmark (gp, x, y, GM\_PLUS, -1.\*SLWID(tdat,i), -1.\*SLWID(tdat,i)) # TMP

}

### Mark slit

For each object on sdat

Use cyan

Yoff = 0.5\*sdat(SLWID,i)

X = FLIP\*sdat(X1,i)

Y = sdat(Y1,i)

Essentially we are drawing a rectangle with edges:

(x,y+yoff) (x,y-yoff)

X= flip\*sdat(x2,i)

Y=sdat(y2,i)

(x,y-yoff) (x,y+yoff)

x=flip\*sdat(x1,i)

y=sdat(y1,i)

(x,y+yoff)

### Start interactive loops with keys:

h,j,k,l: These keys modify indat(RA\_FLD, DEC\_FLD)

p: modifies indat(PA\_ROT)

s: calls selector to select objects, then calls mark\_obj, and calculates the total priority

a: calls get\_sw\_nearest to find a slit nearby, and add it to the selected slits

d: call get\_sw\_nearest to find a slit nearby, and deletes it

i: sets all the tdat(sel,i) to NO

g: calls slit\_free and then get\_slits to generate slits for the selected targets

z: changes zoom

spacebar: info on the nearest target

t: finds nearest target with get\_sw\_nearest,, if PCODE is CODE\_GS, change it to CODE\_AS, and viceversa

If the key is one of h,j,k,l,p,n:

Cosdec=cos(indat(DEC\_FLD))

Cosp=cos(indat(PA\_ROT))

Sinp = sin(indat(PA\_ROT))

Call fld2telax (to update indat with the new position)

Call tel\_coords(to update tdat with the new relative positions to the telescope axis)

Call slit\_free

If the key is one of r,c,h,j,k,l,p,n,f,I,w,g,z:

Redraw the focal plane layout

Seems like the important key to follow is “s”, which selects the targets, and “g” which generates the slits.

### Automatic slit selection

Key “s” automatically selects targets:

Procedure “selector”, with arguments:

Indat (the telescope and input params), tdat (the targets), ntarg (number of targets),

2\*(indat(DEF\_HLEN)+indat(SLIT\_GAP))

Returns psum, sum of achieved total priorities

Complex algorithm, worth exploring. The result is simply to set tdat(SEL)=YES for the targets that are selected.

### Automatic slit generation

Key “g” generates the slits

Calls:

Slit\_free

Gen\_slits

Slit\_free: this procedure seems just a long sequence of calls to mfree, which frees the memory allocated by the sdat structure.

The good thing is that is shows what sdat is supposed to contain. These are the fields:

PTINDEX

PTRA0

PTDEC0

PTPA

PTLEN1

PTLEN2

PTWID

PTCODE

PTXARCS

PTYARCS

PTRELPA  
PTX1

PTX2

PTY1

PTY2

PTSTAT

PTSCOOR

Calling this procedure also sets sdat = 0, and nslit=0, essentially resets the slit assignments

Gen\_slits:

1. Count the selected targets (tdat(SEL)=YES)
2. Allocate memory for the vectors (opposite of slit\_free)
3. Loop on selected targets (index ndx is the running index)
   1. X=tdat(XARCS)
   2. Y=tdat(YARCS)
   3. If X,Y are outside of the focal plane, skip
   4. Sdat(index)=ndx
   5. If tdat(PA)==indef, sdat(PA)=indat(PA\_ROT) else sdat(PA)=tdat(PA)
   6. Sdat(RELPA)=tdat(RELPA)
   7. Sdat(PCODE)=tdat(PCODE)
   8. Sdat(X1,y1,x2,y2)=tdat(x1,y1,x2,y2)
   9. (until the final sky coordinates are calculated, went X/YARCS to represent objects: sdat(XARCS,YARCS) = tdat(XARCS,YARCS)
   10. sdat(SLWID)=tdat(SLWID)
   11. tdat(SLNDX)=ndx (this connected the tdat structure with the corresponding sdat slit)
4. call len\_slits to adjust the length of the slits so that they don’t overlap

### Conversion to sky coordinates

Function sky\_coords (sdat, nslit, indat)

Converts xarcs, yarcs in telescope coordinates onto sky

Ra0= indat(RA\_TEL)

Dec0=indat(DEC\_TEL)

Pa0=indat(PA\_ROT)

For each slit:

X = 0.5\*(sdat(X1)+sdat(X2))

Y = 0.5\*(sdat(Y1)+sdat(Y2))

R=srqt(x\*\*2 + y\*\*2)

R=atan(r/2062648D0)

Phi=pa0-atan2(y,x)

Sind = sin(dec0) \* cos(r) + cos(dec0)\* sin(r)\*cos(phi)

Sina = sin(r) \* sin(phi)/sqrt(1-sind\*\*2)

Sdat(dec)=asin(sind)

Sdat(RA) = ra0+asin(sina)

# calculate centers and lengths of the slits

Sdat(Xarcs) = 0.5\*(sdat(X1)+sdat(X2))

Sdat(Yarcs) = 0.5\*(sdat(Y1)+sdat(Y2))

# slit lengths are defined as TOTAL lenght

X = sdat(x2)-sdat(x1)

Y = sdat(y2)-sdat(y1)

Sdat(len1) = 0.5\*sqrt(x\*\*2+y\*\*2)

Sdat(len2) = sdat(len1)

### Revert the application of the atmospheric refraction

Call unrefr\_coords from dsim3

No need to go into details, it basically removes the refraction

Here is the function

begin

lst = RA\_FLD(indat) + HA\_FLD(indat) # XXX Verify correct/see above

lat = DEGTORAD (OBS\_LAT) # radians

# Apply to field center

ha = lst - RA\_FLD(indat) ## XXX Clean up (see above)

call slde2h (ha, DEC\_FLD(indat), lat, az, el)

zd = HALFPI - el

tanz = tan (zd)

zd = zd + REF1(indat) \* tanz + REF3(indat) \* tanz\*\*3

el = HALFPI - zd

call sldh2e (az, el, lat, ha, DEC0\_FLD(indat))

RA0\_FLD(indat) = lst - ha

call eprintf ("Final Center: %13.3h %12.2h\n")

call pargd (RADTODEG(RA0\_FLD(indat))/15.d0)

call pargd (RADTODEG(DEC0\_FLD(indat)))

call eprintf ("IMPT!! No unrefract to TEL -- FIX!! \n")

# Loop and apply to targets:

do i = 0, nslit-1 {

ha = lst - RA(sdat,i)

call slde2h (ha, DEC(sdat,i), lat, az, el)

zd = HALFPI - el

tanz = tan (zd)

zd = zd + REF1(indat) \* tanz + REF3(indat) \* tanz\*\*3

el = HALFPI - zd

call sldh2e (az, el, lat, ha, DEC0(sdat,i))

RA0(sdat,i) = lst - ha

}

### Calculate position with respect to telescope axis

Call tel\_coords (sdat, nslit, indat) from dsim2

This converts alpha,dec into offsets from the telescope center

Here is the procedure:

ra0 = RA\_TEL(indat)

dec0 = DEC\_TEL(indat)

pa0 = PA\_ROT(indat)

do i = 0, ntarg-1 {

dec\_obj = DEC(tdat,i)

del\_ra = RA(tdat,i) - ra0

cosr = sin (dec\_obj) \* sin (dec0) +

cos (dec\_obj) \* cos (dec0) \* cos (del\_ra)

r = acos (cosr)

sinp = cos (dec\_obj) \* sin (del\_ra) / sqrt (1. - cosr\*cosr)

cosp = sqrt (max ((1. - sinp\*sinp), 0.))

if (dec\_obj < dec0)

cosp = -cosp

p = atan2 (sinp, cosp)

# For now, convert radii to arcsec XXX

# XXX NB: I am not sure this is correct! We should still be on SPHERICAL surf.

# but these are EUCLIDEAN relations. Options: work in spherical coord

# OR work in tan projection.

# More: The difference at 10 arcmin between the tan and angle is < 0.002 arcsec

# If we convert "r" to tan(r) we should have the tan projection

r = tan(r) \* 206264.8

# r = RADTODEG(r) \* 3600.

XARCS(tdat,i) = r \* cos (pa0 - p)

YARCS(tdat,i) = r \* sin (pa0 - p)

if (PA(tdat,i) == INDEF) {

RELPA(tdat,i) = INDEF

rangle = 0.

} else {

RELPA(tdat,i) = PA(tdat,i) - pa0

rangle = RELPA(tdat,i)

}

# For simplicity, we calculate the endpoints in X here; note use of FLIP

xgeom = FLIP \* cos (rangle)

ygeom = sin (rangle)

if (PROJ\_LEN(indat) == YES) {

xgeom = xgeom / abs (cos (rangle))

ygeom = ygeom / abs (cos (rangle))

}

# We always want X1 < X2, so:

if (xgeom > 0) {

X1(tdat,i) = XARCS(tdat,i) - LEN1(tdat,i) \* xgeom

Y1(tdat,i) = YARCS(tdat,i) - LEN1(tdat,i) \* ygeom

X2(tdat,i) = XARCS(tdat,i) + LEN2(tdat,i) \* xgeom

Y2(tdat,i) = YARCS(tdat,i) + LEN2(tdat,i) \* ygeom

} else {

X2(tdat,i) = XARCS(tdat,i) - LEN1(tdat,i) \* xgeom

Y2(tdat,i) = YARCS(tdat,i) - LEN1(tdat,i) \* ygeom

X1(tdat,i) = XARCS(tdat,i) + LEN2(tdat,i) \* xgeom

Y1(tdat,i) = YARCS(tdat,i) + LEN2(tdat,i) \* ygeom

}

# Calc STAT

x = XARCS(tdat,i) # unclear if XYARCS will be real or dbl

y = YARCS(tdat,i)

STAT(tdat,i) = chk\_stat (x, y, YES)

}

end

### Generate mask coordinates for the slits

Call procedure mask\_coords from dsim.x

Converts (x,y) on sky to xmm, ymm on slitmask

Here is the procedure:

asec\_rad = 206264.8D0

# offset from telescope axis to slitmask origin, IN SLITMASK COORDS

yoff = ZPT\_YM \* (1. - cos (DEGTORAD(M\_ANGLE)))

yoff = 0. # XXX check! Am not sure where the above comes from

xoff = 0.

do i = 0, nslit-1 {

# XXX For now, carry through the RELPA thing; in end, must be specified!

if (RELPA(sdat,i) != INDEF) {

cosa = cos (RELPA(sdat,i))

sina = sin (RELPA(sdat,i))

} else {

cosa = 1.

sina = 0.

}

# cosa = cos (RELPA(sdat,i)) # XXX

# sina = sin (RELPA(sdat,i)) # XXX

# This is a recalculation ... prob not needed

X1(sdat,i) = XARCS(sdat,i) - LEN1(sdat,i) \* cosa \* FLIP

Y1(sdat,i) = YARCS(sdat,i) - LEN1(sdat,i) \* sina

X2(sdat,i) = XARCS(sdat,i) + LEN2(sdat,i) \* cosa \* FLIP

Y2(sdat,i) = YARCS(sdat,i) + LEN2(sdat,i) \* sina

# XXX cuidado! I am not sure that the tan-projection of the rel PA is the

# same as the rel PA -- MUST CHECK! (This code comes from gen\_slits)

# The focal plane coordinates are now simply a tan projection of (x,y) arcsec

# Need to verify that these are truly symmetric:

# xfp = FL\_TEL \* tan (DEGTORAD(X1(sdat,i)/3600.))

# yfp = FL\_TEL \* tan (DEGTORAD(Y1(sdat,i)/3600.)) / cos (DEGTORAD(X1(sdat,i)/3600.))

# X1,Y1 are now tan projections already!

xfp = FL\_TEL \* X1(sdat,i) / asec\_rad

yfp = FL\_TEL \* (Y1(sdat,i) - 0.5\*SLWID(sdat,i)) / asec\_rad

pa = 0.

call gnom\_to\_dproj (xfp, yfp, xfp, yfp) # (allowed)

call proj\_to\_mask (xfp, yfp, pa, xsm, ysm, pa)

XMM1(sdat,i) = xsm + xoff

YMM1(sdat,i) = ysm + yoff

xfp = FL\_TEL \* X2(sdat,i) / asec\_rad

yfp = FL\_TEL \* (Y2(sdat,i) - 0.5\*SLWID(sdat,i)) / asec\_rad

pa = 0.

call gnom\_to\_dproj (xfp, yfp, xfp, yfp) # (allowed)

call proj\_to\_mask (xfp, yfp, pa, xsm, ysm, pa)

XMM2(sdat,i) = xsm + xoff

YMM2(sdat,i) = ysm + yoff

xfp = FL\_TEL \* X2(sdat,i) / asec\_rad

yfp = FL\_TEL \* (Y2(sdat,i) + 0.5\*SLWID(sdat,i)) / asec\_rad

pa = 0.

call gnom\_to\_dproj (xfp, yfp, xfp, yfp) # (allowed)

call proj\_to\_mask (xfp, yfp, pa, xsm, ysm, pa)

XMM3(sdat,i) = xsm + xoff

YMM3(sdat,i) = ysm + yoff

xfp = FL\_TEL \* X1(sdat,i) / asec\_rad

yfp = FL\_TEL \* (Y1(sdat,i) + 0.5\*SLWID(sdat,i)) / asec\_rad

pa = 0.

call gnom\_to\_dproj (xfp, yfp, xfp, yfp) # (allowed)

call proj\_to\_mask (xfp, yfp, pa, xsm, ysm, pa)

XMM4(sdat,i) = xsm + xoff

YMM4(sdat,i) = ysm + yoff

}

call metal\_check (sdat, nslit)

## Perhaps we want to force YMM4-YMM1 == YMM3-YMM2; for non-tilted slits,

## this should produce a cleaner edge; otherwise, jumps can occur.

end

### Note that this procedure contains a call to metal\_check

Call to metal\_check

Checks to make sure that metal limits are not violated.

A lot of hardcoded numbers

begin

do i = 0, nslit-1 {

xmin = min (XMM1(sdat,i), XMM4(sdat,i))

xmax = max (XMM2(sdat,i), XMM3(sdat,i))

ymin = min (YMM1(sdat,i), YMM2(sdat,i))

ymax = max (YMM4(sdat,i), YMM4(sdat,i))

if (xmin < -373.) {

call eprintf ("slit=%d; xmin=%6f \n")

call pargi (i)

call pargr (xmin)

call fatal (0, "xmin < -373.")

}

if (xmax > 373.) {

call eprintf ("slit=%d; xmax=%6f \n")

call pargi (i)

call pargr (xmax)

call fatal (0, "xmax > 373.")

}

# XXX should check for guide stars here

# if (ymin < 2.) {

# call eprintf ("slit=%d; ymin=%6f \n")

# call pargi (i)

# call pargr (ymin)

# call fatal (0, "ymin < 2.")

# }

if (ymax > 225.17) {

call eprintf ("slit=%d; ymax=%6f \n")

call pargi (i)

call pargr (ymax)

call fatal (0, "ymax > 225.17")

}

# y+x < 490.

if (XMM3(sdat,i)+YMM3(sdat,i) > 496.) {

call eprintf ("offending slit = %d \n")

call pargi (i)

call fatal (0, "beyond cut edge!")

}

}

end

### Write design file

Call write\_design(indat, tdat, ntarg, sdat, nslit, delpa, std\_fmt, protfile) from dswd.x

Rather complex and long procedure that produces the fits file. Not worth discussing until we are ready