Main Table of Contents

Options

Pri	Print Diagnostics Flag
Halt	Stop Execution Part Way
PV	Print Rays at each Surface
Sunits	Change Size Units
Wunits	Change Wavelength Units

PRINT DIAGNOSTICS

If *pri* is set to 1, then a running account of what has happened in the trace will be printed on the screen.   Simply type *pri=1* before or during the trace to get the diagnostics.   To turn off the diagnostics type *pri=0* and continue.

```
;EXAMPLE OF PRI OPTION
pri=1
objp,0.,0.,1.e12,20,5000.
pupcr,0.,1.
disp,0.,0.,-1000.
flat
grating,1,10000.
disp,0.,0.,1000.
flat
print,'Rms in radial direction = ',rms(0)
end
```

Which will yield a screen printout that looks like:

```
Point Object of 20 rays     at x = 0.000000e+00, y = 0.000000e+00, z = 1.000000e+12
                                        wavelength = 5.000000e+03
Circular Random Entrance Pupil      radii, inner=0 , outer= 1   at x=0.000000e+00
                                                          y=0.000000e+00
Origin Moved to 0.000000e+00, 0.000000e+00, -1.000000e+03
Surface Number 1
Flat Surface in the z=0 Plane
Reflection Grating Surface     order = 1, d = 10000, psi = 10
Origin Moved to 0.000000e+00, 0.000000e+00, 1.000000e+03
Surface Number 2
Flat Surface in the z=0 Plane
Rms in radial direction = 0.491219
IRT>
```

AUTOMATIC HALT

If *halt* is set to 0 then the program automatically halts at the end of each optical element so that the current status of the rays can be studied.   To continue type *continue* (or just *c*).   If HALT is set to a positive integer, then IRT will automatically stop at the surface which has *nsurf* equal to the chosen integer.   To return to tracing without halting set *halt*=*-1*.
The *halt* option allows a certain amount of freedom for analysis.   For example, if *halt*=3, then IRT will stop at the third optical surface. A call to SPOT displays the rays at that point, and can often be very useful in debugging a TRACE procedure.

AUTOMATIC PRINT VECTOR

If *pv* is set to a non-negative integer then at the end of each procedure the ray values associated with ray number *pv* will be printed as a diagnostic. Thus, usually *pv* is set to 0 to print the values of the first ray. *pv*=-1 turns the option off.
Since *pv* can be called directly by the user, the automatic *pv* is usually used in the context of debugging or checking the TRACE. The automatic *pv* prints the ray values after each IRT procedure call, thereby allowing the user to follow the logic of the tracing and identify where it is having problems.

```
;EXAMPLE OF PV OPTION
pv=0
pri=1
objp,0.,0.,1.e12,20,5000.
pupcr,0.,1.
disp,0.,0.,-1000.
flat
grating,1,10000.
disp,0.,0.,1000.
print,'Rms in radial direction = ',rms(0)
end
```

Which creates and prints on the screen the following diagnostics:

```
Point Object of 20 rays     at x = 0.000000e+00, y = 0.000000e+00, z = 1.000000e+12
               wavelength = 5.000000e+03
Circular Random Entrance Pupil      radii, inner=0 , outer= 1      at x=0.000000e+00 ,
y=0.000000e+00
num = 0   x = -2.953049e-01   qx = -2.953049e-13   lam = 5.000000e+03
              y = -2.580074e-02   qy = -2.580074e-14   nind= 1.000000e+00
              z = 0.000000e+00    qz = -1.000000e+00
Origin Moved to 0.000000e+00, 0.000000e+00, -1.000000e+03
num = 0 x = -2.953049e-01   qx = -2.953049e-13   lam = 5.000000e+03
             y = -2.580074e-02   qy = -2.580074e-14   nind= 1.000000e+00
             z = 1.000000e+03    qz = -1.000000e+00
Surface Number 1
Flat Surface in the z=0 Plane
num = 0   x = -2.953049e-01   qx = -2.953049e-13   lam = 5.000000e+03
              y = -2.580074e-02   qy = -2.580074e-14   nind= 1.000000e+00
              z = 0.000000e+00    qz = -1.000000e+00
Reflection Grating Surface    order = 1, d = 10000, psi = 10
  num = 0   x = -2.953049e-01   qx = 5.000000e-01    lam = 5.000000e+03
                y = -2.580074e-02   qy = -2.580074e-14   nind= 1.000000e+00
                z = 0.000000e+00    qz = 8.660254e-01
Origin Moved to 0.000000e+00, 0.000000e+00, 1.000000e+03
num = 0 x = -2.953049e-01   qx = 5.000000e-01    lam = 5.000000e+03
             y = -2.580074e-02   qy = -2.580074e-14   nind= 1.000000e+00
             z = -1.000000e+03   qz = 8.660254e-01
Surface Number 2 Flat Surface in the z=0 Plane
num = 0 x = 5.770550e+02   qx = 5.000000e-01     lam = 5.000000e+03
             y = -2.580074e-02   qy = -2.580074e-14   nind= 1.000000e+00
             z = 0.000000e+00    qz = 8.660254e-01
Rms in radial direction = 0.491219
IRT>
```

SIZE UNITS

The common variable *sunits* contains a string variable denoting the size units being used for the trace. The options available are: MILLIMETERS, CENTIMETERS, METERS, INCHES. To change to meters (for example) type *sunits='METERS'* prior to running the trace. The standard default is 'MILLIMETERS'. It is used primarily because it is usually near the geometric mean of the size of the entire optical system and the size of the focal spots generated. Thus it generates reasonable size numbers for both.
The size units are used in several ways. First, it is used in the axis labels of the display routines. Second, for physical optics effects it is necessary to know the scaling factor between the optics scale and the wavelength scale. Appropriate factors based on *sunits* are used when diffraction effects are being calculated.

WAVELENGTH UNITS

The variable *wunits* contains a string variable denoting the wavelength units one is using in the trace.   The options available are: ANGSTROMS, NANOMETERS, MICRONS, MILLIMETERS, CENTIMETERS.   To change to microns (for example) simply type *wunits*='MICRONS' prior to running the trace.   The standard default is 'ANGSTROMS'.
The wavelength units are used primarily for two purposes.   First, when indices are retrieved using GLASS, the wavelength units must be known.   Second, the wavelength units are essential in calculating diffraction effects.

Objects

Objca	Circular Array
Objcr	Circular Random
Objfan	Fan Array (Cross Shape)
Objgr	Gaussian Random
Objp	Point
Objsa	Rectangualr Array
Objsr	Rectangular Random

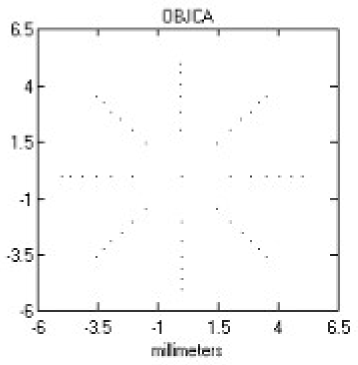
OBJCA, *ro, nrad, nsec, xc, yc, obd, lamt*

ro = outer radius of object circle (5.e10)
nrad = number of radii in array   (5)
nsec = number of sectors in array (5)
xc = x position of the center of the object (0.)
yc = y position of the center of the object (0.)
obd = z position of the object (1.e12)
lamt = wavelength of the rays (6328.)

OBJCA sets up a circular object centered at the position [xc,yc,obd] and lying in the z=obd plane.

It generates a geometrically weighted distribution of x,y pairs in a circle of radius *ro*.   It thus simulates photons emerging from an annular disk of uniform brightness.   It is particularly convenient for performing raytraces that allow the user to understand the origin of aberrations. OBJCA creates a *nrad* times *nsec* plus one source points.   The extra is the first ray, and lies at the center.

```
;Circular Array object with 8 arms and 6 points on each radius
;created for an f/10 beam.
_ti='OBJCA'
objca,5.e10,6,8,0.,0.,1.e12,6328.
pupp,0.,0.
disp,0,0,100.
flat
spot
end
```



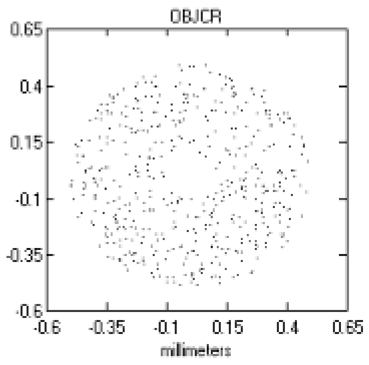
OBJCR, *ri, ro, xc, yc, obd, nr, lamt*

ri = inner radius of object annulus (0.)
ro = outer radius of object annulus (5.e10)
xc = x position of the center of the object (0.)
yc = y position of the center of the object (0.)
obd = z position of the object (1.e12)
nr = number of ray positions in object (100)
lamt = wavelength of the rays (6328.)

OBJCR sets up an annular object centered at the position [xc,yc,obd] and lying in the z=obd plane.

It generates a uniform random distribution of x,y pairs between the radii of *ri* and *ro*. It thus simulates photons emerging from an annular disk of uniform brightness. Of course, setting *ri* to 0 creates a circular object.

```
;Circular object with small hole in the center.
;1mm in diameter, 300m from pupil.
;500 uniformly distributed random rays.
_ti='OBJCR'
objcr,.1,.5,0.,0.,3.e5,500,6328.
pupcr,0.,500.
disp,0.,0.,3.e5
flat
; N.B. This call forces the rays back to the source for display
; purposes.   As such it will generate a warning that the rays
; moved backward.   Warning is printed only if pri is 1.
spot
end
```



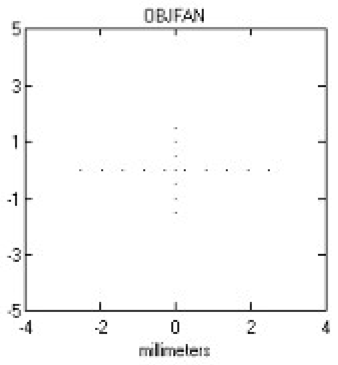
OBJFAN, *sx, nx, sy, ny, xc, yc, obd, lamt*

sx = distance from first to last point of x line (5.e10)
nx = number of points in x line (9)
sy = distance from first to last point of y line (5.e10)
ny = number of points in y line (0)
xc = x position of the center of the object (0.)
yc = y position of the center of the object (0.)
obd = z position of the object   (1.e12)
lamt = wavelength of the rays (6328.)

OBJFAN creates a cross pattern of light or a line pattern.

It creates *nx+ny* points. *nx* evenly spaced parallel to the x-axis, with a distance *sx* from the first to the last. Similarly, *ny* points are evenly spaced along a distance of *sy*. The center of the cross or the line is at [*xc,yc,obd*]. A value of *nx* of 0 will give a line object in the y direction, and *ny* of 0 gives a line object in x.

```
;OBJFAN EXAMPLE: A 10 by 7 cross at infinity
_ti='OBJFAN'
objfan,5.e10,10,3.e10,7,0.,0.,1.e12,6328.
pupp,0.,0.
disp,0,0,100.
flat
spot
end
```



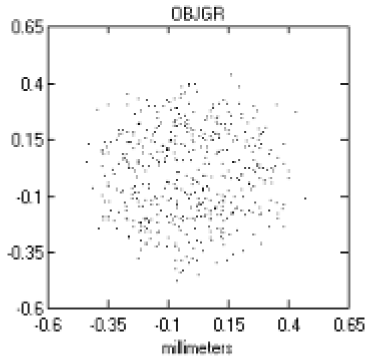
OBJGR, *sig, xc, yc, obd, nr, lamt*

sig = Standard deviation of object distribution (5.e10)
xc = x position of the center of the object (0.)
yc = y position of the center of the object (0.)
obd = z position of the object (1.e12)
nr = number of ray positions in object (100)
lamt = wavelength of the rays (6328.)

OBJGR sets up an object with circular symmetry centered at the position [xc,yc,obd] and lying in the z=*obd* plane.

It simulates a normally blurred source by generating *nr* pairs of x,y coordinates with a gaussian distribution. *sig* is given in the same distance units as are used elsewhere.

```
;OBJGR Example:
;Gaussian distribution of one arcsec diameter at infinity
;500 normally distributed random rays.
_ti='OBJGR'
objgr,2.5e6,0.,0.,1.e12,500,6328.
pupp,0.,0.
disp,0,0,10000.
flat
; N.B.
; This call forces the rays back to the source for display
; purposes. As such it will generate a warning that the rays
; moved backward.   Warning is printed only if  pri is 1.
spot
end
```



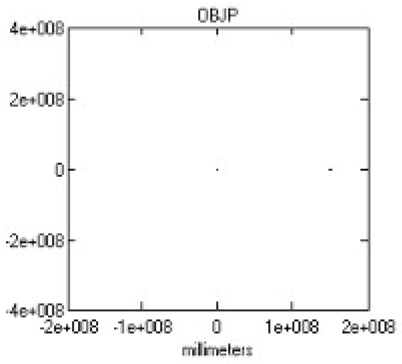
OBJP, *xc, yc, obd, nr, lamt*

xc = x position of the object (0.)
yc = y position of the object (0.)
obd = z position of the object (1.e12)
nr = number of rays starting at point (100)
lamt = wavelength of the rays (6328.)

OBJP simulates an ideal point source at the position [*xc,yc,obd*].

It creates *nr* rays, all of which start at the same point.

```
;OBJP Example: Point sources 0 and 30 arcseconds off axis
;500 rays each
_ti='OBJP'
offax=1.5e8
objp,0.,0.,1.e12,500,6328.
objp,offax,0.,1.e12,500,6328.
pupcr,0.,50.
disp,0,0,1.e12
flat
; N.B. This call forces the rays back to the source for display
; purposes.   As such it will generate a warning that the rays
; moved backward.   Warning is printed only if pri is 1.
spot
end
```



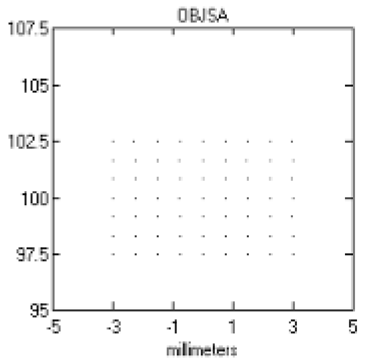
OBJSA, *sx, nx, sy, ny, xc, yc, obd, lamt*

sx = length of x side of array (5.e10)
nx = number of points along x side of array (5)
sy = length of y side of array (5.e10)
ny = number of points along y side of array (5)
xc = x position of the center of the object (0.)
yc = y position of the center of the object (0.)
obd = z position of the object (1.e12)
lamt = wavelength of the rays (6328.)

OBJSA sets up a rectangular object centered at the position [*xc,yc,obd*] and lying in the z=*obd* plane.

It generates a geometrically weighted distribution of x,y pairs in a rectangle of size *sx* by *sy*. It thus simulates photons emerging from a rectangular source of uniform brightness. It is particularly convenient for performing raytraces that allow the user to understand the origin of aberrations.

```
 ;OBJSA Example: Rectangular Array object with 9 (x) by 7 (y) format
_ti='OBJSA'
objsa,8.e10,9,6.e10,7,0.,0.,1.e12,6328.
pupp,0.,0.
disp,0,0,100.
flat
spot
end
```



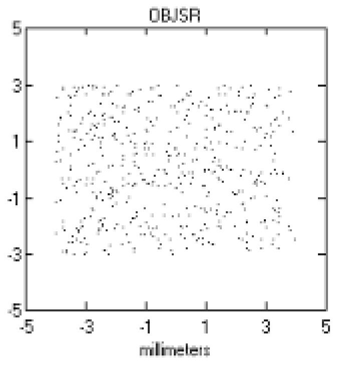
OBJSR, *xl, yl, xc, yc, obd, nr, lamt*

xl = x length of object (5.e10)
yl = y length of object (5.e10)
xc = x position of the center of the object (0.)
yc = y position of the center of the object (0.)
obd = z position of the object   (1.e12)
nr = number of ray positions in object (100)
lamt = wavelength of the rays (6328.)

OBJSR sets up a rectangular object centered at the position [*xc,yc,obd*] and lying in the z=*obd* plane.

It generates a   uniform random distribution of x,y pairs in a rectangle of size *xl* by *yl.* It thus simulates photons emerging from   rectangle of uniform brightness.

```
;OBJSR Example: Rectangular Array object with random format.
_ti='OBJSR'
objsr,8.e10,6.e10,0.,0.,1.e12,500,6328.
pupp,0.,0.
disp,0,0,100.
flat
; N.B. This call forces the rays back to the source for display
; purposes.   As such it will generate a warning that the rays
; moved backward.   Warning is printed only if  pri is 1.
spot
end
```



Pupils

Pupca	Circular Array
Pupcr	Circular Random
Pupfan	Fan Array
Pupp	Point
Pupsa	Rectangular Array
Pupsr	Rectangular Random

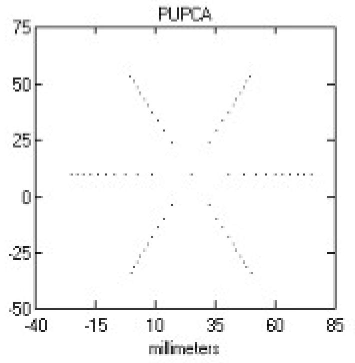
PUPCA, *ro, nrad, nsec, xc, yc, zc*

ro = radius of entrance pupil (50.)
nrad = number of radii (5)
nsec = number of angles (5)
xc = x center of pupil (0.)
yc = y center of pupil (0.)
zc = z center of pupil (0.)

PUPCA creates a circular array of entrance pupil points.

The points lie in concentric circles, the number of circles controlled by *nrad*, and the number of points in each circle controlled by *nsec*. The separation between the circles varies to achieve correct weighting of the point distribution so that it approximates uniform illumination. From each point in the object a ray is drawn to each point in the pupil. This is in sharp distinction to the random pupil arrays which draw one ray from each object point. Thus the total number of rays to be traced will be the product of the number of object points and the number of pupil points. It is very easy to make the number of rays very large. In practice, however, this pupil is usually used with OBJP which creates only a single object point.
PUPCA creates *nrad* times *nsec* plus one points. The extra point is first and lies at the center.

```
;PUPCA EXAMPLE: pupil with 50mm radius aperture. Points in a
; circular array with 6 spokes containing 10 points each.
_ti='PUPCA'
objp,0.,0.,1.e12,1,6328.
pupca,50.,10,6,25.,10.
spot
end
```



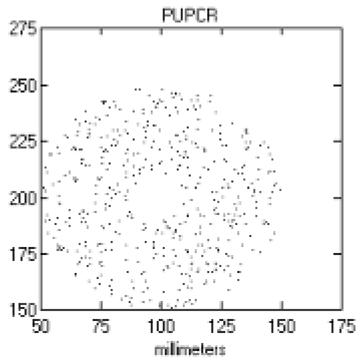
PUPCR, *ri, ro, xc, yc, zc*

ri = inner radius of annulus (0.)
ro = outer radius of annulus (50.)
xc = x center of pupil (0.)
yc = y center of pupil (0.)
zc = z center of pupil (0.)

PUPCR generates an annular (circular) entrance pupil lying in the x-y plane and centered at the origin.

Each ray generated by the object calls is given a position on the entrance pupil. The distribution of points is uniform with area and random.

```
;PUPCR EXAMPLE: pupil with 50mm radius aperture. Points in a
; circular array with uniform random distribution. 10mm radius
; hole in center.
_ti='PUPCR'
objp,0.,0.,1.e12,500,6328.
pupcr,10.,50.,100.,200.
spot
end
```



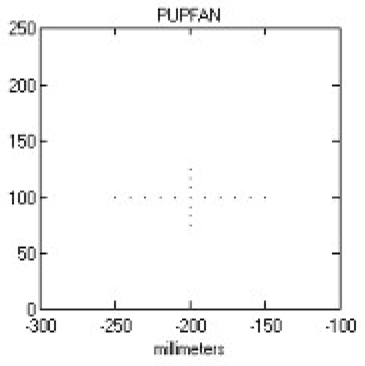
PUPFAN, *xo, nx, yo, ny, xc, yc, zc*

xo = length of x side of cross (100.)
nx = number of points on x bar (10)
yo = length of y side of cross (100.)
ny = number of points on y bar (0)
xc = x center of pupil (0.)
yc = y center of pupil (0.)
zc = z center of pupil (0.)

PUPFAN creates a cross of entrance pupil points.

The points lie on two orthogonal lines, with *nx* points along the x direction and *ny* points along y. The lines of points are centered at the origin, and are *xo* from one end of the x line to the other and *yo* long in y. From each point in the object a ray is drawn to each point in the pupil. This is in sharp distinction to the random pupil arrays which draw one ray from each object point. Thus the total number of rays to be traced will be the product of the number of object points and the number of pupil points. It is very easy to make the number of rays very large. In practice, however, this pupil is usually used with OBJP which creates only a single object point.

```
;PUPFAN EXAMPLE: pupil in the shape of a cross.
; 11 points along a length of 100mm in the x direction.
; 7 points along a length of 50mm in the y   direction.
_ti='PUPFAN'
objp,0.,0.,1.e12,1,6328.
pupfan,100.,11,50.,7,-200.,100.
spot
end
```



PUPP, *xo, yo, zo*

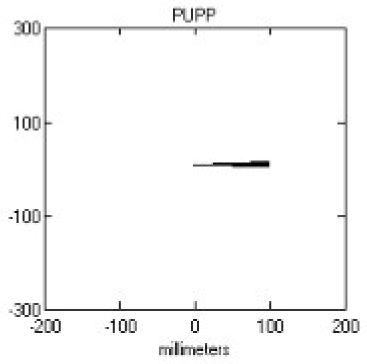
xo = x position of entrance point (0.)
yo = y position of entrance point (0.)
zo = z center of pupil (0.)

PUPP creates a point-like entrance pupil.

It has many practical uses. It forces all rays from the object to pass through the point *xo,yo*. This allows one to create a "chief ray" trace when coupled to OBJP or simulate light diverging from a point source at *xo,yo*.

If PUPP is used with a random OBJ then a single ray is drawn from each object point to the point *xo,yo*. If used with an array of object points the same thing happens. Since there is only one pupil point no multiplier effect is encountered.

```
;PUPP EXAMPLE: Point pupil at the position [10.,0.,0.]
_ti='PUPP'
objfan,1.e11,5, , ,0.,0.,1.e12,6328.
pupp,10.,0
disp,0.,0.,-100.
flat
surfaces
end
```



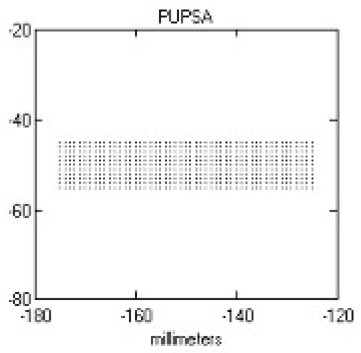
PUPSA, *xo, nx, yo, ny, xc, yc, zc*

xo = x length of entrance pupil (100.)
nx = number of points along x (3)
yo = y length of entrance pupil (100.)
ny = number of points along y (3)
xc = x center of pupil (0.)
yc = y center of pupil (0.)
zc = z center of pupil (0.)

PUPSA creates a rectangular array of entrance pupil points.

The number of points along the x side is given by *nx*, the number along y is given by *ny*. *xo* and *yo* specify the dimensions of the pupil. From each point in the object a ray is drawn to each point in the pupil. This is in sharp distinction to the random pupil arrays which draw one ray from each object point. Thus the total number of rays to be traced will be the product of the number of object points and the number of pupil points. It is very easy to make the number of rays very large. In practice, however, this pupil is nearly always used with OBJP which creates only a single object point.

```
;PUPSA EXAMPLE: pupil with 50mm by 10mm aperture. Points
; separated by 1 mm.
_ti='PUPSA'
objp,0.,0.,1.e12,1,6328.
pupsa,50.,51,10.,11,-150.,-50.
spot
end
```



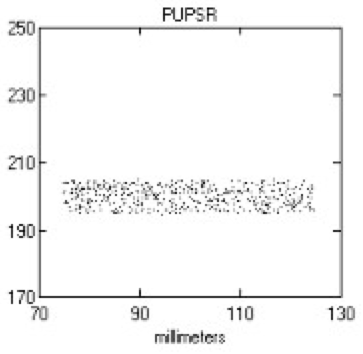
PUPSR, *xo, yo, xc, yc, zc*

xo	= x length of object (100.)
yo	= y length of object (100.)
xc	= x center of pupil (0.)
yc	= y center of pupil (0.)
zc	= z center of pupil (0.)

PUPSR generates a rectangular (square) entrance pupil lying in the x-y plane and centered at the origin.

Each ray which was generated by the object calls is given a position on the entrance pupil. The distribution of points is uniform with area and random.

```
;PUPSR EXAMPLE: pupil with 50mm by 10mm aperture. 500 points
; in random uniform distribution.
_ti='PUPSR'
objp,0.,0.,1.e12,500,6328.
pupsr,50.,10.,100.,200.
spot
end
```



Coordinate Routines

Disp	Move the Coordinate Origin
New_Coord	Set Specific Coordinate System
Retrace	Return to former coordinates
Rot	Rotate Coordinate System
Untrace	Return to Pupil Coordinates

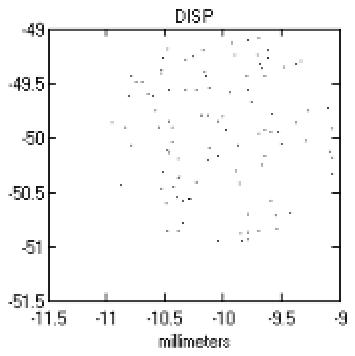
DISP, *dx, dy, dz*

dx = displacement in the x direction (0.)
dy = displacement in the y direction (0.)
dz = displacement in the z direction (0.)

This routine changes the coordinate system.

The origin is redefined to be at the point which was called [*dx,dy,dz*] in the old coordinate system. No rotation is performed. *dmat* matrix is automatically updated.

```
;example disp
_ti='DISP'
objp,0.,0.,1.e12,100,6328.
pupcr,0.,1.
disp,10.,50.,0.
spot
end
```



NEW_COORD,*d0,d1,d2,r0,r1,r2,r3,r4,r5,r6,r7,r8*

d0 = first element of new dmat (0.)
d1 = second element of new dmat (0.)
d2 = third element of new dmat (0.)
ro = element of new rmat (1.)
r1 = element of new rmat (0.)
r2 = element of new rmat (0.)
r3 = element of new rmat (0.)
r4 = element of new rmat (1.)
r5 = element of new rmat (0.)
r6 = element of new rmat (0.)
r7 = element of new rmat (0.)
r8 = element of new rmat (1.)

This routine may be used to move the rays to a coordinate system defined in an absolute sense by *rmat* and *dmat*.

Although one can, in principle, derive the input parameters from scratch, in practice, the input values are printed out in a prior run of the code. Thus, when one wishes to save an absolute coordinate system at the end of a run, use the commands *print,dmat* and *print,rmat*. The numbers are entered into NEW_COORD in the same order that IRT prints them out.
One use of this routine is to freeze each element of the optical system in absolute space, so that changes in the position of other elements will not affect it. This is usually not done unless the system is complicated and is to be studied in great detail. Another use is to allow multiple paths to impinge on the same element.

```
;EXAMPLE ROT
;ANOTHER WAY TO RETURN TO THE PUPIL COORDINATE SYSTEM
_ti='NEW_COORD'
objp,0.,0.,1.e12,1,6328.
pupfan,2.,0,2.,3
disp,0.,0.,-50.
rot,10.,0.,0.
flat
mirror
disp,0.,0.,50.
new_coord,0.,0.,0.,1.,0.,0.,0.,1.,0.,0.,0.,1.
flat
end
```

RETRACE

This routine undoes the effects of UNTRACE. It is typically used to restore the rays to the TRACE coordinate system after using UNTRACE to examine where the rays are relative to the pupil.

```
;EXAMPLE RETRACE:
_ti='RETRACE'
objp,0.,0.,1.e12,100,6328.
pupcr,0.,1.
disp,0.,0.,-500.
flat
pv
untrace
pv
retrace
pv
end
```

The output will be:

```
Point Object of 100 rays     at x = 0.000000e+00, y = 0.000000e+00, z = 1.000000e+12
                                        wavelength = 6.328000e+03
Circular Random Entrance Pupil      radii, inner=0 , outer= 1
                                        at x=0.000000e+00 , y=0.000000e+00
Origin Moved to 0.000000e+00, 0.000000e+00, -5.000000e+02
Rays Traveled to the z = 0.000000e+00 Plane
   num = 0 x = -2.953049e-01   qx = -2.953049e-13    lam = 6.328000e+03
               y = -2.580074e-02    qy = -2.580074e-14    nind= 1.000000e+00
               z = 0.000000e+00      qz = -1.000000e+00
Rays Returned to the Pupil Coordinate System
     from dmat = 0 0 500
          rmat = 1 0 0
                     0 1 0
                     0 0 1
   num = 0   x = -2.953049e-01   qx = -2.953049e-13   lam = 6.328000e+03
                 y = -2.580074e-02   qy = -2.580074e-14   nind= 1.000000e+00
                 z = -5.000000e+02   qz = -1.000000e+00
Rays Returned to Saved Coordinate System
   num = 0 x = -2.953049e-01   qx = -2.953049e-13   lam = 6.328000e+03
               y = -2.580074e-02   qy = -2.580074e-14   nind= 1.000000e+00
               z = 0.000000e+00     qz = -1.000000e+00
```

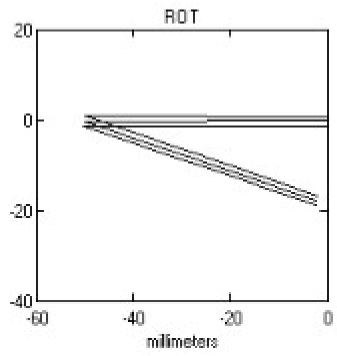
ROT, *dx, dy, dz*

dx = degrees of rotation about the x-axis (0.)
dy = degrees of rotation about the y-axis (0.)
dz = degrees of rotation about the z-axis (0.)

ROT performs the coordinate rotations that allow the rays to approach the optical elements at the correct orientation.

ROT performs rotations about the current axes in the order x then y then z. It accepts the angles in units of degrees. Rotations are performed in a right hand positive sense. If the rays are considered fixed then the coordinate axes move counterclockwise in response to a positive angle. Alternatively, if one wishes to consider the axes fixed, the positions and directions of the rays are rotated clockwise.

```
;EXAMPLE ROT
_ti='ROT'
objp,0.,0.,1.e12,1,6328.
pupfan,2.,0,2.,3
disp,0.,0.,-50.
rot,10.,0.,0.
flat
mirror
disp,0.,0.,50.
flat
rays,90.,0.
end
```



UNTRACE

UNTRACE returns the ray positions and angles to the coordinate system of the entrance pupil.

This is particularly useful in debugging complicated raytraces. By calling untrace, one can see where the rays are in an absolute sense. This can be very useful in determining mirror images, and the direction of rotations in the code. It is also useful in interfacing a design with mechanical engineers. By calling UNTRACE, the positions of the rays on each element of the system can be obtained in a consistent coordinate system.

Furthermore, if UNTRACE is called twice in a row it will return nonsense on the second call because the rotation and displacement matrices were not updated. Not updating is convenient because then RETRACE allows return to the trace coordinate system. If however, one wishes to remain in the pupil system and continue tracing, one should use NEW_COORD with the default values.

```
;EXAMPLE UNTRACE:
_ti='UNTRACE'
objp,0.,0.,1.e12,100,6328.
pupcr,0.,1.
disp,0.,0.,-500.
flat
pv
untrace
pv
end
```

The printout is:

```
Point Object of 100 rays    at x = 0.000000e+00, y = 0.000000e+00, z = 1.000000e+12
                                          wavelength = 6.328000e+03
Circular Random Entrance Pupil     radii, inner=0 , outer= 1
                                          at x=0.000000e+00 , y=0.000000e+00
Origin Moved to 0.000000e+00, 0.000000e+00, -5.000000e+02
Rays Traveled to the z = 0.000000e+00 Plane
  num = 0 x = -2.953049e-01   qx = -2.953049e-13    lam = 6.328000e+03
              y = -2.580074e-02   qy = -2.580074e-14   nind= 1.000000e+00
              z = 0.000000e+00    qz = -1.000000e+00
Rays Returned to the Pupil Coordinate System
   from dmat = 0 0 500
        rmat = 1 0 0
                  0 1 0
                  0 0 1
  num = 0 x = -2.953049e-01   qx = -2.953049e-13   lam = 6.328000e+03
             y = -2.580074e-02   qy = -2.580074e-14   nind= 1.000000e+00
             z = -5.000000e+02   qz = -1.000000e+00
```

Housekeeping Routines

Compact	Remove vignetted rays
Addrays	Add rays saved on disk
Getrays	Get rays saved on disk
Saverays	Save rays on disk

ADDRAYS, *name*

name = name of disk file from which ray information is to be read. ('irt')

This routine is valuable for combining the results of a number of runs of TRACE. Each run stores its rays in a separate disk file. ADDRAYS retrieves the information from disk and appends the new rays to the already existing ones. It requires that there already be rays in the common blocks. If one is pulling all the information freshly from disk a call to GETRAYS is required before a call to ADDRAYS.

```
;ADDRAYS EXAMPLE
_ti='ADDRAYS'
objp,0.,0.,1.e12,10,6328.
pupp,0.,0.
saverays,'test0'
objp,0.,0.,1.e12,10,6328.
pupp,1.,0.
saverays,'test1'
objp,0.,0.,1.e12,10,6328.
pupp,2.,0.
saverays,'test2'
objp,0.,0.,1.e12,10,6328.
pupp,3.,0.
saverays,'test3'
addrays,'test0'
addrays,'test1'
addrays,'test2'
spot
end
```

COMPACT

COMPACT is used to remove vignetted rays from the vectors.   During tracing, when a ray is vignetted, the value of status is set to zero for that ray, and from that point onward all IRT routines will ignore the ray.   If, however, one wishes to apply PCL routines to the ray vector information that contains vignetted rays, the presence of the vignetted ray information will affect the outcome.   To remove the vignetted rays and thereby decrease the length of the ray vectors, call compact.   Then all remaining rays will be unvignetted.

```
;COMPACT EXAMPLE
objp,0.,0.,1.e12,1,6328.
pupsa,5.,2,5.,2
knife
print,'x before = 'x
compact
print,'x after =',x
end
```

will yield:

```
x before = -2.5   -2.5   2.5   2.5
x after = 2.5   2.5
```

GETRAYS, *name*

name = name of storage file ('irt')

This routine retrieves the ray information from disk where it was previously stored by SAVERAYS. *name* is a string variable specifying the name of the storage file on disk. If *name* is not specified then the routine searches for the default name *irt.sav*. Any rays currently held in common are lost.

```
;GETRAYS EXAMPLE
_ti='GETRAYS'
objp,0,0,1.e12,50,6328.
pupcr,4.,5.
saverays,'test1'
objp,0,0,1.e12,10,6328.
pupp,0.,1.
getrays,'test1'
spot
end
```

SAVERAYS, *name*

name = name of disk file on which ray information is to be stored. ('irt')

This routine saves the results of a TRACE on disk for future use. It saves the entire contents of the common blocks. It is thus possible to save a set of rays as they exist in the middle of a complicated system. Instead of tracing the whole system repeatedly while designing the latter part of the system, use SAVERAYS once, and use GETRAYS as the starting point from then on. Be warned that a large ray vector can chew up quite a bit of disk space. Use this option sparingly. The rays are saved in a file with a .sav extension. The default save file name is irt.sav. The first call to saverays in the example will create a file called test1.sav.

```
;SAVERAYS EXAMPLE
_ti='SAVERAYS'
objp,0.,0.,1.e12,50,6328.
pupcr,4.,5.
saverays,'test1'
objp,0.,0.,1.e12,10,6328.
pupp,0.,1.
getrays,'test1'
spot
end
```

Surface Routines

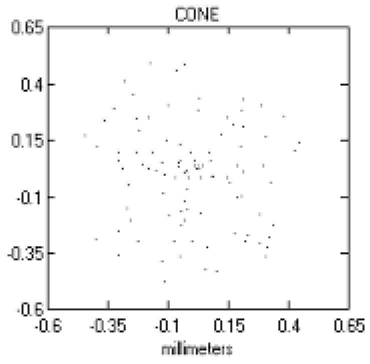
Cone	Cone
Conic	Conic Section of Rotation
Conicx	Conic Section - One Dimensional
Ellipsoid	Ellipsoidal Surface
Flat	Flat Surface
Focus	Find the best focus
Knife	Knife Edge Vignette
Mask	Mask Vignetting
Torus	Toroidal Surface
Travel	Move Rays to Plane

CONE, *halfang*

halfang = half angle of cone in degrees (45.)

CONE raytraces a conical surface. The vertex of the cone lies at the origin and the z-axis is its axis of symmetry. CONE finds the intersections of the rays with the cone. It chooses the intersection that is closest to the origin and on the positive side if possible.

```
;EXAMPLE CONE
_ti='CONE'
objp,0.,0.,1.e12,100,6328.
pupcr,51.365,52.365
disp,0.,0.,-3000.
cone,1.
mirror
focus,0
spot
end
```



CONIC, *e, rad, sol, pol*

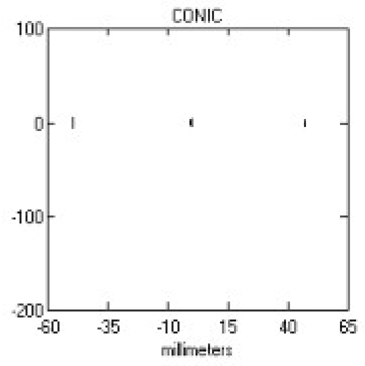
e = eccentricity (0.)
rad = radius of curvature at vertex (1000.)
sol = intersection solution (0)
pol = polynomial deformation of surface (0)

This routine assumes that there is a surface which is formed from a conic section of rotation about the z-axis. Both the vertex and focus of the conic lie on the z-axis. The vertex lies at the origin. This routine thus covers spheres, ellipsoids, paraboloids, and hyperboloids. A positive radius of curvature gives a surface which curves upward and lies on the positive side of the x-y plane. A negative curvature bends downwards.

The intersection of a line with a conic section can have zero, one or two solutions. An example of no solutions is a ray that completely misses intersecting a sphere. In CONIC rays with no solutions are vignetted. An example of one solution is an on-axis ray striking a paraboloid. An example of two solutions is an off-axis ray and a paraboloid; one intersection point may be near the focus and the other at grazing incidence. CONIC uses the solution closest to the vertex unless the parameter *sol* is set to 1, in which case it uses the other solution.

pol is a polynomial deformation of the conic surface. *pol* is a vector with each element containing a coefficient of the deformation. The first element is the coefficient for r0, the second element is for r1, the third for r2, etc. This deviates from standard practice in optics where only even powers are included to avoid a mathematical cusp at r=0. We include the odd powers because they can come in handy in simulating grazing optics where the mirror surfaces do not cover r=0, and many high power terms are needed. There is no limit on how high the powers can go. The routine starts where the ray intersects the conic and performs a Newton's loop convergence from there.

```
;EXAMPLE CONIC
; A ROWLAND CIRCLE MOUNT WITH CURVED FOCAL PLANE
_ti='CONIC'
d=8771.4
lamt=[5000.,6000.,7000.]
objcr,0.,5.e10,0.,0.,1.e12,100,lamt
pupp,0.,0.
disp,0.,0.,-375.88
rot,0.,-20.,0.
conic,0.,400.
grating,1,d
rot,0.,180.,0.
disp,0.,0.,-200.
rot,0.,40.,0.
disp,0.,0.,-200.
conic,0.,200.
spot
end
```

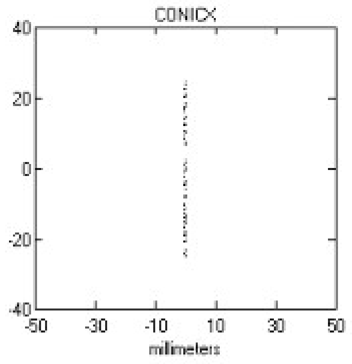

CONIC
100
0
-100
-200
-60
-35
-10
15
40
65
milimeters

CONICX, *e, rad, sol*

e = eccentricity (0.)
rad = radius of curvature at vertex (1000.)
sol = intersection solution (0)

This routine is identical to CONIC except that it traces one dimensional conics which have curvature in the x direction only. Thus they lie with their axis of symmetry being the X=0 plane, and their vertices lie along the Y-axis.

```
;EXAMPLE CONICX
; TRACE A CYLINDER
_ti='CONICX'
objp,0.,0.,1.e12,100,6328.
pupsr,50.,50.
disp,0.,0.,-1000.
conicx,0.,2000.
mirror
disp,0.,0.,1000.
flat
spot
end
```



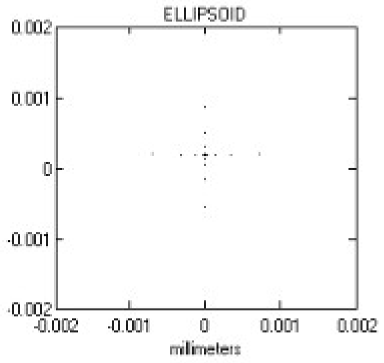
ELLIPSOID, *a, b ,c , sol*

a = radius of curvature in x direction (1000.)
b = radius of curvature in y direction (1000.)
c = radius of curvature in z direction (1000.)
sol = intersection solution (0)

ELLIPSOID creates an ellipsoidal surface defined by the equation

In most cases *c* is the same as either *a* or *b*.

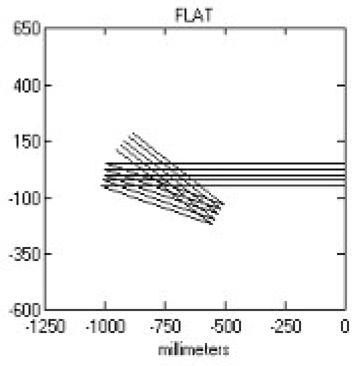
```
;EXAMPLE ELLIPSOID
_ti='ELLIPSOID'
objfan,1.e10,10,1.e10,10,0.,0.,1.e12,6328.
pupp,0.,0.
rot,-80.,0.,0.
disp,0.,171.01,-30.154
ellipsoid,173.65,1000.,1000.
mirror
rot,100.,0.,0.
disp,0.,0.,-173.65
flat
spot
end
```



FLAT

FLAT creates a flat surface.   Rays are moved to the z=0 plane. Ray directions are not changed.

```
;EXAMPLE FLAT SURFACE
_ti='FLAT'
objp,0.,0.,1.e12,1,6328.
pupfan,0.,100.,0,5
disp,0.,0.,-1000.
rot,10.,0.,0.
flat
mirror
rot,10.,0.,0.
disp,0.,0.,500.
rot,10.,0.,0.
flat
mirror
rot,10.,0.,0.
disp,0.,0.,-500.
flat
rays,0,90
end
```

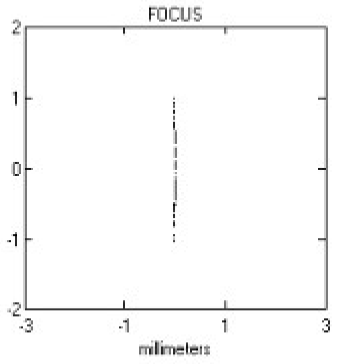


FOCUS, *ax*

ax = axis for focus: 0=radial, 1=x, 2=y (0)

FOCUS finds the best (i.e. minimum rms) focus position of the rays and moves the rays to that plane. It then redefines the best focus position as the origin.

```
;EXAMPLE FOCUS
; A ROWLAND CIRCLE MOUNT
_ti='FOCUS'
objcr,0.,5.e10,0.,0.,1.e12,100,7000.
pupp,0.,0.
disp,0.,0.,-396.11
rot,0.,-8.,0.
conic,0.,400.
grating,1,21556.
focus,1
spot
end
```

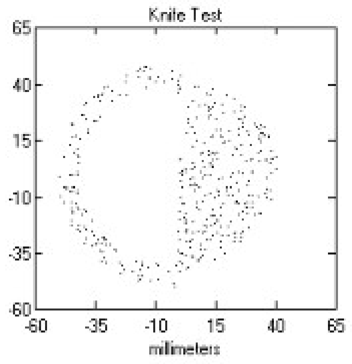


KNIFE, *edge*

edge = x position of knife edge (0.)

KNIFE is a convenient means of simulating an optical knife-edge test. KNIFE vignettes all the rays with an x position less than *edge*. A typical use of KNIFE to move to a system focus, cut the rays and then return back to the surface where the rays originated. The distribution of rays at the surface will then tell the user a great deal about the sources of blur in the design.

```
_ti='Knife Test'
objp,0.,0.,1.e12,800,6328.
pupcr,0.,50.
disp,0,0,-1000.
conic,0.,2000.
mirror
disp,0,0,1000.
flat
focus
knife,0.
disp,0,0,-1000.
conic,0.,2000.
spot
end
```



MASK, *shape, a, b, xc, yc*

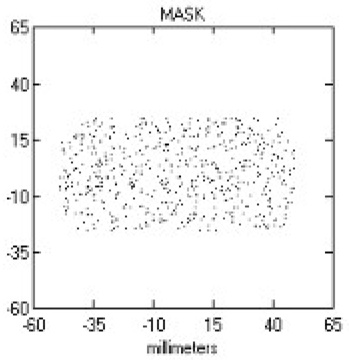
shape = shape of mask: 1=annular, 2=rectangular (1)
a = inner radius for annular (1.)
= x length for rectangular
b = outer radius for annular (10.)
= y length for rectangular
xc = x center of mask (0.)
yc = y center of mask (0.)

This routine simulates stops and masks in an optical system. When a ray encounters a mask it either proceeds freely or is vignetted. MASK does not move the rays at all. It merely tests the current values of x and y to determine whether they satisfy the mask criteria. Thus, if the routine is to be properly used, one must be sure that the rays are at the position of the mask. This may seem like an unnecessary complexity, but it actually can be very useful. For example, to simulate the finite extent of a concave mirror, call CONIC and follow it immediately by MASK. The projection of the rays onto the z=0 plane will not be performed, and instead will test for position on the mirror.

MASK assumes that the stop is symmetric about the point *xc,yc*. Two shapes are available: annular and rectangular. An annular mask will stop all rays out to a radius *a*, then allow all rays out to radius *b* to pass, and vignette all rays outside *b*. A rectangular mask will pass only those rays which have x within *a/2*. of *xc* and y within *b/2.* of *yc*. Choosing annular versus rectangular is set by *shape*.

To form an inverse mask set *shape* to the negative of the value that would be chosen otherwise. An inverse mask stops where the regular mask passes and passes where the regular mask stops.

```
_ti='MASK'
objp,0.,0.,1.e12,1000,6328.
pupcr,0.,50.
mask,2,200.,50.
end
```



TORUS, *rx, ry*

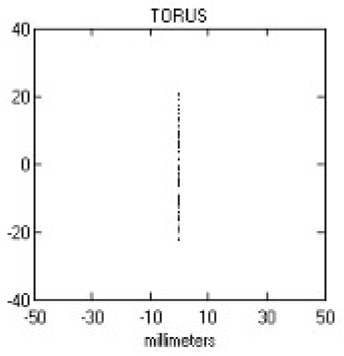
rx = radius of curvature in x direction (1000.)
ry = radius of curvature in y direction (1000.)

This routine raytraces toroidal surfaces. The two radii of curvature in the torus are defined by *rx* and *ry.* The intersection of the y=0 plane with the mirror is a circle of radius *rx*. The intersection of the x=0 plane is a circle of radius *ry.* The toroid is tangent to the z=0 plane at the origin. Rays are moved to the surface, their reflected directions calculated and then left there.

Since a ray can intersect a toroid at up to four places, it is important to understand that this routine calculates only the intersection which is closest to the origin. A positive radius of curvature gives a surface curves upward and lies on the positive side of the x-y plane. A negative curvature bends downward.

TORUS treats rays as though the surface were infinitely thin. Thus, a positive curvature conic and rays moving in the positive z direction will act like a convex lens; a positive curvature and rays moving in the negative z direction will produce a concave lens.

```
;EXAMPLE TORUS
_ti='TORUS'
objp,0.,0.,1.e12,100,6328.
pupcr,0.,50.
disp,0.,0.,-1000.
torus,500.,1000.
mirror
disp,0.,0.,250.
flat
spot
end
```



TRAVEL, *ax, dis*

ax = axis for direction of travel. 1=x, 2=y, 3=z. (3)
dis = value of plane to travel to.   (0.)

TRAVEL is used to cause the rays to travel to a specified plane. Usually *ax* is 3 so that the rays travel to a specified value of z (*dis*}.

```
;EXAMPLE TRAVEL
_ti='TRAVEL'
objfan,1.e11,10,1.e11,10,0.,0.,1.e12,6328.
pupca,50.,5,5
thin,1000.
disp,0.,0.,-1000.
travel,3,0.
spot
end
```

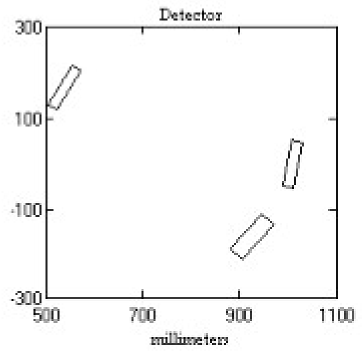
Element Routines

Detector	Detector at this Surface
Grating	Reflection Grating
Holograt	Holographic Reflection Grating
Lens	Spherical Surface Lens
Mirror	Mirrored Surface
Refract	Refracting Surface
Thin	Thin Lens Simulator
Trangrat	Transmission Grating

DETECTOR

DETECTOR does not manipulate the ray information in any way. It merely marks the preceeding surface as the detector for purposes of display with post-trace routines.

```
;example flat surface
title='DETECTOR'
objp,0.,0.,1.e12,1,6328.
pupfan,100.,5,1.,0
disp,0.,0.,-1000.
rot,0.,10.,0.
flat
mirror
rot,0.,10.,0.
disp,0.,0.,500.
rot,0.,10.,0.
flat
mirror
rot,0.,10.,0.
disp,0.,0.,-500.
flat
detector
view
end
```

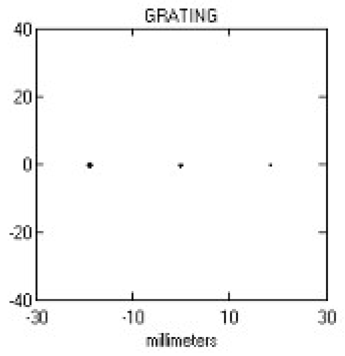


GRATING, *nord, d*

nord = order number for rays (1)
d = groove spacing in wavelength units (3.333e4)

This is a generalized reflection grating routine that can be used with any kind of surface. GRATING is called immediately after the call to the surface routine. GRATING adjusts the ray directions as if there were a reflection grating ruled on the surface. The rulings are lines that are the intersection of the surface with planes perpendicular to the x-axis. Calls to ROT and DISP are allowed to occur between the surface routine and GRATING allowing investigation of some unusual geometries. This routine uses *nord* for the order number. Rays which cannot satisfy the grating equation are vignetted.

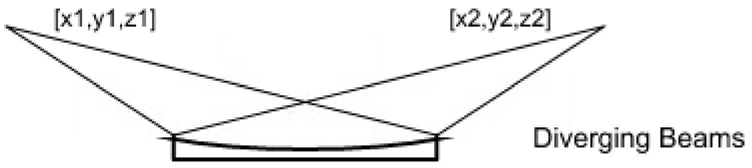
```
;EXAMPLE GRATING
; A ROWLAND CIRCLE MOUNT WITH PARABOLOID GRATING
_ti='GRATING'
d=2.155e4
lamt=[5000.,6000.,7000.]
objcr,0.,5.e10,0.,0.,1.e12,100,lamt
pupp,0.,0.
disp,0.,0.,-399.01
rot,0.,-8.,0.
conic,1.,400.
grating,1,d
rot,0.,188.,0.
disp,0.,0.,-399.01
flat
spot
end
```

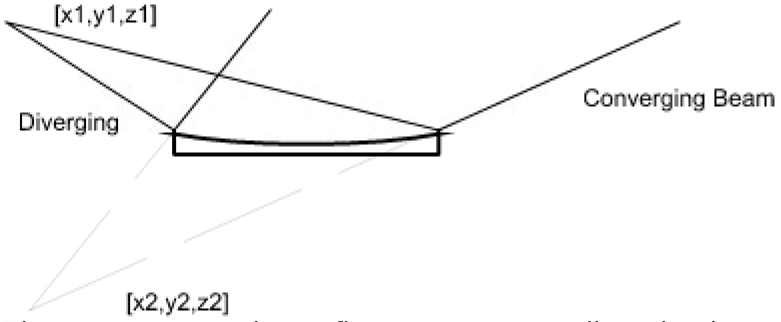


HOLOGRAT, *nord, x1, y1, z1, x2, y2, z2, laser*

nord = order number for rays (1)
x1 = x position of source 1 (0.)
y1 = y position of source 1 (0.)
z1 = z position of source 1 (1.e12)
x2 = x position of source 2 (1.e12)
y2 = y position of source 2 (0.)
z2 = z position of source 2 (1.e12)
laser = wavelength of recording laser (6328.)

This is a generalized reflection grating routine that can be used with any kind of surface. HOLOGRAT is called immediately after the call to the surface routine. HOLOGRAT adjusts the ray directions as if there were a reflection grating ruled on the surface. Calls to ROT and DISP are allowed to occur between the surface routine and HOLOGRAT allowing investigation of some unusual geometries. This routine uses *nord* for the order number. Rays which cannot satisfy the grating equation are vignetted.
HOLOGRAT supports all first generation holographic gratings, i.e., those with parallel or spherically diverging wavefronts from two coherent sources. The actual recording geometry used to fabricate the grating is input to the routine. After specifying the order number for diffraction, just as in GRATING, the next six parameters are the spatial coordinates of the two coherent point sources. The last calling parameter is the wavelength of the laser used, in the same units as the wavelengths used in the object call.

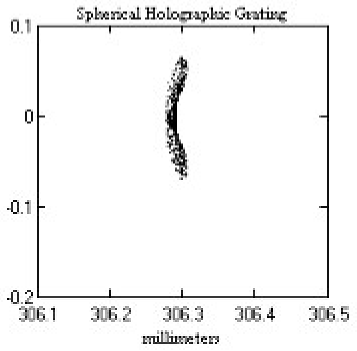




The geometry as shown first assumes two diverging beams. If a parallel beam is desired, place the source position at a distance on the order of 1012mm. If a converging beam is desired, place the source at its virtual focus. IRT will notice that the source is below the surface, and assume a beam converging to the virtual focus.

```
; Holographic Spherical Grating test routine
; As per Grange, Applied Optics 1993
;
if( n_elements( wav ) == 0 ) wav = 970.
objcr,0.,6.25e10,0.,0.,1.e12,1000,wav
pupp
flat
```

```
disp,0,0,-1692.016
rot,0.,-22.953,0
conic,0.,1837.5
rot,0,0,180.
a1=-72.49
r1=5164.626
a2=75.738-180.
r2=4985.934
dr=3.1415927/180.
holograt,1.,r1*sin(a1*dr),0.,r1*cos(a1*dr),r2*sin(a2*dr),0.,r2*cos(a2*dr),3336.
rot,0.,180.,0.
disp,0,0,-1837.5
conicx,0.,1837.5/2.
end
```

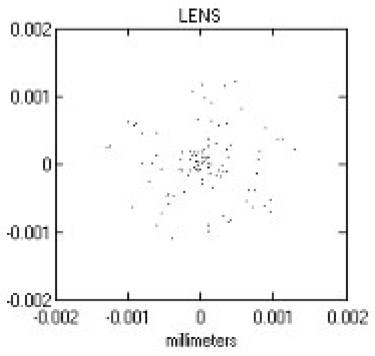


LENS, *index, r1, thick, r2*

index = index of refraction of material being entered (1.)
r1 = radius of curvature of front surface (0.)
thick = thickness of lens at center (0.)
r2 = radius of curvature of back surface (0.)

LENS is a routine that calculates the behavior of standard lenses. The lens should have spherical or planar surfaces, and the spheres must be coaligned. *r1* and *r2* are the radii of curvature of the front and back surfaces, just as defined in CONIC. *thick* is the thickness of the lens at the center. LENS assumes that the rays are going from vacuum downward (negative z direction) into the lens and back into vacuum afterwards. LENS displaces the coordinate system downward along the z-axis by the amount *thick*, and leaves the rays on the rear surface. If the back surface is actually a different glass it is OK to call LENS twice in a row. This is less efficient than calling REFRACT, but is accurate.
If *r1* is positive, then the front surface will be curved upward, forming a concave front surface. If *r1* is 0., then it is a flat surface. If *r1* is negative, then the front surface is convex. If *r2* is positive, then the back surface will be curved upward, forming a convex surface. If *r2* is 0., then it is a flat surface. If *r2* is negative, the back surface is concave.
If *index* is a scalar, or a vector of length one, then that value of index is used for all the rays. If *index* is a vector of length equal to the number of rays, then the values are assigned one to one. If the length of *index* exceeds the number of rays, the extra indices are ignored. If the number of rays exceeds the number of indices, the excess rays are all assigned the value of the last index.
The values in *index* are often set with a call to the function GLASS (see the manual entry for GLASS to see an example). The example below shows how to evaluate a lens based solely on the input index of refraction.

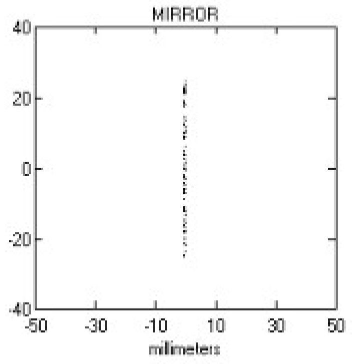
```
;from Laikin "Lens Design" p 39
_ti='LENS'
if( length( wav ) == 0 ) wav=5000.
if( length( xoff ) == 0 ) xoff=0.
if( length( yoff ) == 0 ) yoff=0.
objp,xoff*5.e6,yoff*5.e6,1.e12,100.,wav
pupcr,0.,3.
i1=glass('BK7', lam)
lens,i1,-29.2097,.655,19.9047
disp,0,0,-.032
i2=glass('SF2',lam)
lens,i2,19.8979,.578,66.3937
disp,0,0,.578+.032+.655-48.646
flat
spots
end
```


LENS
0.002
0.001
0
-0.001
-0.002
-0.002
-0.001
0
0.001
0.002
milimeters

MIRROR

MIRROR follows a surface routine such as CONIC, TORUS, or CONICX to convert it to a mirror. MIRROR uses the normal vector components *ux,uy* and *uz* defined in the surface routine to reflect the rays.

```
;EXAMPLE MIRROR
; TRACE A CYLINDRICAL MIRROR
_ti='MIRROR'
objp,0.,0.,1.e12,100,6328.
pupsr,50.,50.
disp,0.,0.,-1000.
conicx,0.,2000.
mirror
disp,0.,0.,1000.
flat
spot
end
```

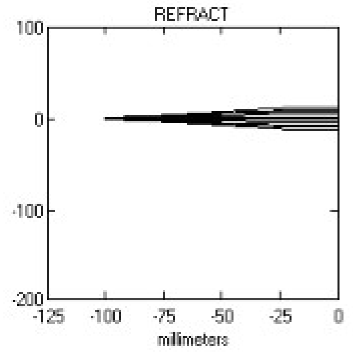


REFRACT, *index*

index = index of refraction of material being entered (1.)

REFRACT is a general routine that calculates the behavior of a lens of arbitrary surface shape.   REFRACT should follow the surface call immediately, with only calls to DISP and ROT in between.   The surface routine returns a *ux,uy,uz* vector of normals at the ray intersections.   REFRACT performs the calculations of new directions using Snell's Law.
If *index* is a scalar, or a vector of length one, then that value of index is used for all the rays. If *index* is a vector of length equal to the number of rays, then the values are assigned one to one. If the length of *index* exceeds the number of rays, the extra indices are ignored.   If the number of rays exceeds the number of indices, the excess rays are all assigned the value of the last index.
The values in *index* are often set with a call to the function GLASS (see the manual entry for GLASS to see an example).   The example below shows how to evaluate a lens based solely on the input index of refraction.

```
;_ti='REFRACT'
objp,0.,0.,1.e12,1,6328.
pupfan,0.,0,25.,10
disp,0.,0.,-10.
conic,0.,-100.
refract,1.6
disp,0.,0.,-10.
conic,0.,100.
refract,1.
focus
flat
rays,90.,0.
end
```

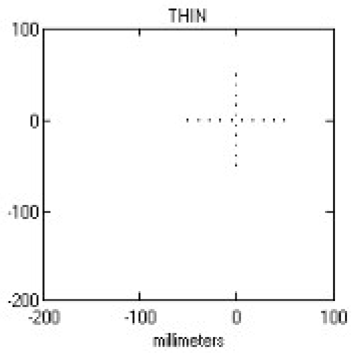


THIN, *f*

f = focal length of thin lens (1000.)

THIN simulates an ideal thin lens of focal length *f.* It moves the rays to the z=0 plane where it focuses them according to the thin lens law. Of course, this is not a true optical element, but suffices quite well when the details of a lens are not required.

```
;EXAMPLE THIN
_ti='THIN'
objfan,1.e11,10,1.e11,10,0.,0.,1.e12,6328.
pupca,50.,5,5
thin,1000.
disp,0.,0.,-1000.
flat
spot
end
```



TRANGRAT, *nord, d*

nord = diffraction order number (1)
d = groove spacing in wavelengths (3.333e4)

TRANGRAT raytraces a transmission grating laid on the previous surface. Just as in grating the rulings as projected onto the z=0 plane are uniformly spaced *d* apart, running parallel to the x=0 line. TRANGRAT functions properly inside a material with an index of refraction other than 1, so can be used after a call to REFRACT to simulate a transmission grating ruled on a glass surface.
To simulate a thin transmission grating such as used in x-ray astronomy call a surface routine and then follow it with TRANGRAT.

```
;EXAMPLE TRANGRAT
_ti='TRANGRAT'
wav=[10.,20.,30.,40.,50.,60.,70.,80.,90.]
objp,0.,0.,1.e12,10,wav
pupcr,0.,600.
thin,9000.
trangrat,1,20000.
disp,0.,0.,-9000.
flat
spot
end
```

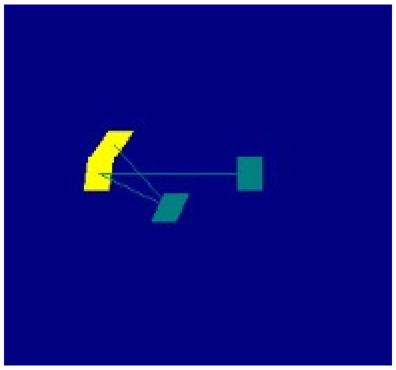
Display Routines

Layout	Surface Orientation Display
MTF	Modulation Transfer Function
Oprof	Overplot ray histogram
Rays	Ray Paths
Rprof	Radial ray distribution
Rspot	Spot plots of remembered surfaces
Spot	Spot Plots
Spots	Spots from all the surfaces
View	Element Display
Xprof	Histogram of x values of rays

LAYOUT, *theta, phi, nf, nl*

theta = altitude of view angle wrt z-axis in degrees (90.)
phi = azimuth of view angle wrt to z-axis in degrees (90.)
nf = first surface number in plot (pupil = 0)
nl = last surface number in plot (nsurf = last surface)

This routine will create screen plots of the surfaces in the optical system. This can be a very effective way to determine if the TRACE is doing what you think it is doing. Obviously, if large numbers of surfaces are traced, the plot can become confused. Surfaces are shown as squares of the correct orientation and size to represent the optical surface. The line joining the origin of each surface to the next is shown to guide the eye.
theta and *phi* control the angle of viewing for the system. The default shows a plot of x vs. z as defined at the pupil. *nf* and *nl* allow the user to pick some subset of the whole trace for examination. During the trace the print diagnostics will indicate the surface number associated with each optical element. *thet* and *phi* allow the system to be viewed from a chosen direction. This routine will fail if *recap* was set to 0, because the information needed will not have been stored during the tracing.



```
objp,0.,0.,1.e12,1,6328.
pupfan,160.,5,160.,5
disp,0.,0.,-1000.
rot,10.,0.,0.
flat
mirror
rot,10.,0.,0.
disp,0.,0.,500.
rot,10.,0.,0.
rot,0.,180.,0.
flat
rot,0.,180.,0.
mirror
rot,10.,0.,0.
disp,0.,0.,-500.
flat
;rays,90,0
layout,45.,0.
end
```

MTF, freqmax, *nbins*

freqmax = maximum frequency in mtf plot   (auto scaling)
nbins = number of bins   (32)

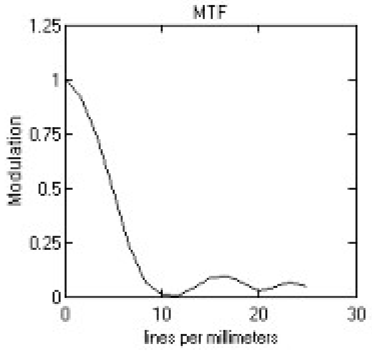
MTF calculates the modulation transfer function of the geometric ray distribution.   The MTF is the discrete fourier transform of the ray distribution and gives specific information about resolving power of the image.   The MTF is actually the modulus of the FFT as a function of frequency.   The phase information, which is included in the OTF (optical transfer function) is not given.   MTF creates a plot of the modulation up to Nyquist frequency, beyond which there is no useful additional information.

freqmax sets the frequency at the right of the mtf plot.   Use it to investigate very high frequency response, which may not be properly covered by the automatic scaling.

nbins controls the number of bins in the FFT.   The default is 32 bins, which yields a plot only 16 points long, but is usually adequate.   *N.B. nbins* must be a power of 2 for use in the FFT, or the procedure will fail. There is one exception.   Because of the limited plotting power of the ascii plots, if *plotter* is set to 'ascii', the calling value of *nbins* is ignored and the default is set to 128.

Note, that it is a good idea to trace a large number of rays (i.e. around a thousand or more) if MTF is to be used.   The FFT will be very noisy if the number of rays is small.

```
;EXAMPLE MTF
_ti='MTF'
objp,0,0,200.,1000,7000.
pupcr,0.,18.
conic,0.,-105.674
refract,1.903566
disp,0,0,-22.
conic,0.,-59.654
refract,1.599812
disp,0,0,-36.
conic,0.,206.969
refract,1.
disp,0,0,-28.06
conic,0.,-333.879
refract,1.782476
disp,0,0,-22.
conic,0.,328.771
refract,1.
disp,0,0,-1.
conic,0.,-169.781
refract,1.708303
disp,0,0,-45.
conic,0.,150.
refract,1.641188
disp,0,0,-7.
conic,0.,3608.069
refract,1.
disp,0,0,-58.472
flat
mtf
end
```

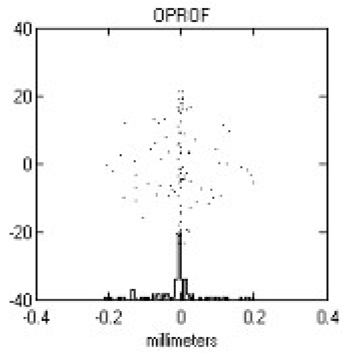

MTF
Modulation
lines per millimeters
1.25
1
0.75
0.5
0.25
0
0
10
20
30

OPROF, *xbin*

xbin = size of bins (extent/60.)

OPROF causes a histogram of the spot distribution to be plotted across the lower part of a previously plotted SPOT plot. The purpose of this is to aid the eye in projecting a two dimensional image into a one dimensional linear response function. It does not, however, provide an axis scale for interpretation of the histogram.

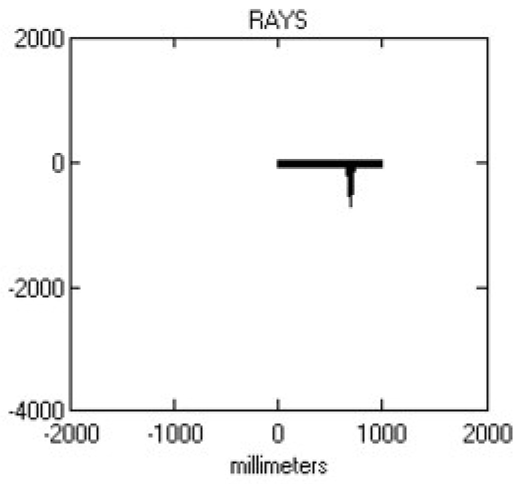
```
;EXAMPLE SPOT
_ti='OPROF'
objp,0.,0.,1.e12,100,6328.
pupcr,0.,50.
disp,0.,0.,-1000.
torus,500.,1000.
mirror
disp,0.,0.,250.
flat
spot,1
oprof,.005
end
```



RAYS, *theta, phi, nf, nl*

theta = altitude of view angle wrt z-axis in degrees (90.)
phi = azimuth of view angle wrt to z-axis in degrees (90.)
nf = first surface number in plot (pupil = 0)
nl = last surface number in plot (nsurf = last surface)

This routine will create screen plots of the paths of the rays through the optical system. This can be a very effective way to determine if the TRACE is doing what you think it is doing. Obviously, if large numbers of rays are traced, the plot can become confused. This routine is similar to SPOTS, but differs in that it draws the ray paths, and not the spots. *theta* and *phi* control the angle of viewing for the system. The default shows a plot of x vs. z as defined at the pupil. *nf* and *nl* allow the user to pick some subset of the whole trace for examination. During the trace the print diagnostics will indicate the surface number associated with each optical element. *thet* and *phi* allow the system to be viewed from a chosen direction. This routine will fail if *recap* was set to 0, because the information needed will not have been stored during the tracing.



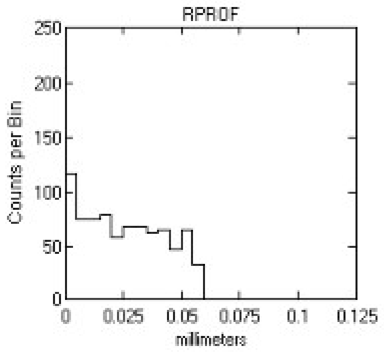
```
_ti='RAYS'
objp,0.,0.,1.e12,1,6328.
pupfan,100.,11
disp,0.,0.,-1000.
conic,0.,2000.
mirror
disp,0.,0.,300.
rot,0,45.,0
flat
mirror
rot,0,45.,0.
disp,0,0,-700.
flat
rays
end
```

RPROF, *rbin, xc, yc*

```
    rbin = size of bins   (default: rmax/60.)
      xc     = x position of radial center   (0.)
      yc     = y position of radial center   (0.)
```

RPROF creates histograms of the spot density as a function of radius about the point [xc,yc]. If bin size is undefined IRT will calculate the ray with the maximum radius and make the bin size one sixtieth of that radius.

```
;EXAMPLE RFOC
_ti='RFOC'
objp,0.,0.,1.e12,100,6328.
pupcr,0,50.
disp,0.,0.,-1000.
conic,0.,1000.
mirror
disp,0.,0.,500.
flat
rprof,.005
end
```



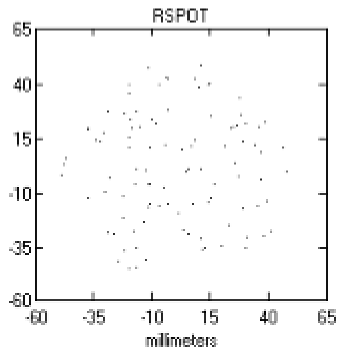
RSPOT, *nel*

nel = surface number to be plotted (0)

RSPOT makes spot plots of the remembered surfaces. It plots the x and y positions of the rays with equal scales on both axes if scale is 0. It independently scales the two axes if *scale* is set to 1. If *recap* has been 1 during the trace, then all the ray positions have been saved, and a spot plot can be created for any of the surfaces in the raytrace by using RSPOT and calling its surface number. RSPOT plots the ray positions in the coordinate system of the surface being recalled.

This routine will fail if *recap* was set to 0, because the information needed will not have been stored during the tracing.

```
;EXAMPLE RSPOT
_ti='RSPOT'
objp,0.,0.,1.e12,100,6328.
pupcr,0.,50.
disp,0.,0.,-1000.
torus,500.,1000.
mirror
disp,0.,0.,250.
flat
rspot,1
;plots the rays at the torus
end
```

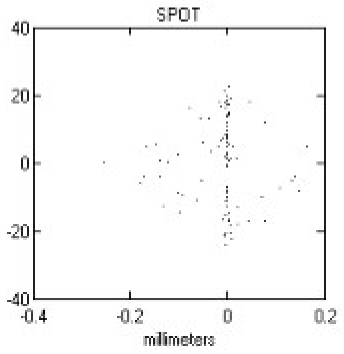


SPOT, *scale*

scale = stretched or square scale.. 0=square, 1=stretched (0)

SPOT is the basic routine for displaying the results of a TRACE. It plots the x and y positions of the rays with equal scales on both axes if scale is 0. It independently scales the two axes if *scale* is set to 1.

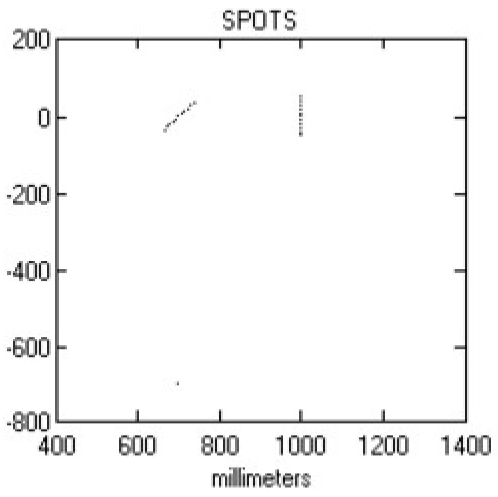
```
;EXAMPLE SPOT
_ti='SPOT'
objp,0.,0.,1.e12,100,6328.
pupcr,0.,50.
disp,0.,0.,-1000.
torus,500.,1000.
mirror
disp,0.,0.,250.
flat
spot,1
end
```



SPOTS, *theta, phi, nf, nl*

theta = altitude of view angle wrt z-axis in degrees (90.)
phi = azimuth of view angle wrt to z-axis in degrees (90.)
nf = first surface number in plot (pupil = 0)
nl = last surface number in plot (nsurf = last surface)

This routine will create screen plots of the positions of the rays at each element surfaces through the optical system. This can be a very effective way to determine if the TRACE is doing what you think it is doing. Obviously, if large numbers of rays are traced, the plot can become confused. This routine differs from SPOT in that it shows all the spot plots at once, in their proper relative positions. It differs from rays in that it draws the spots, and not the paths from one to the next.
theta and *phi* control the angle of viewing for the system. The default shows a plot of x vs. z as defined at the pupil. *nf* and *nl* allow the user to pick some subset of the whole trace for examination. During the trace the print diagnostics will indicate the surface number associated with each optical element. *theta* and *phi* allow the system to be viewed from a chosen direction.
This routine will fail if *recap* was set to 0, because the information needed will not have been stored during the tracing.



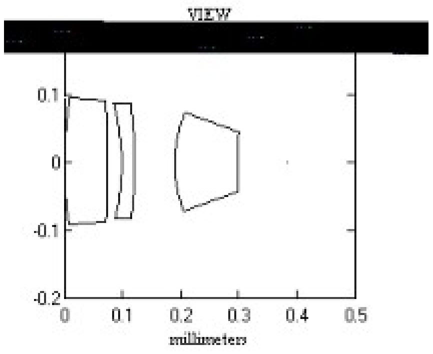
```
_ti='SPOTS'
objp,0.,0.,1.e12,1,6328.
pupfan,100.,11
disp,0.,0.,-1000.
conic,0.,2000.
mirror
disp,0.,0.,300.
rot,0,45.,0
flat
mirror
rot,0,45.,0.
disp,0,0,-700.
flat
spots
```

```
end
```

VIEW, *theta, phi, nf, nl*

theta = altitude of view angle wrt z-axis in degrees (90.)
phi   = azimuth of view angle wrt to z-axis in degrees (90.)
nf   = first surface number in plot (pupil = 0)
nl   = last   surface number in plot (nsurf = last surface)

This routine will create screen plots of the elements through the optical system.   This can be a very effective way to determine if the TRACE is doing what you think it is doing. This routine is similar to SPOTS, but differs in that it draws the element surfaces, and not the spots or rays.
theta and *phi* control the angle of viewing for the system. The default shows a plot of x vs. z as defined at the pupil. *nf* and *nl* allow the user to pick some subset of the whole trace for examination.   During the trace the print diagnostics will indicate the surface number associated with each optical element.   *thet* and *phi* allow the system to be viewed from a chosen direction.
This routine will fail if *recap* was set to 0, because the information needed will not have been stored during the tracing.
VIEW is limited in its ability to show three dimensional objects.   It shows only the cross section of the elements in the x-z plane of the element itself.   Thus it serves well for axi-symmetric systems like a series of lenses, but for fully three dimensional systems it may provide inadequate representation.



```
;from Laikin p 233
;
if(n_elements(wav)==0)wav=6328.
if(n_elements(xoff)==0)xoff=0.
if(n_elements(yoff)==0)yoff=0.
objp,xoff*5.e6,yoff*5.e6,1.e12,100.,wav
pupcr,0.,.1
i1=glass('SF6',lam)
lens,i1,-.4605,.075,1.1942
disp,0,0,-.024
i2=glass('BK7',lam)
lens,i2,.2759,.022,.5778
disp,0,0,-.069
lens,i1,-.1488,.109,-.5985
disp,0,0,.075+.024+.022+.069+.109-.384
flat
```

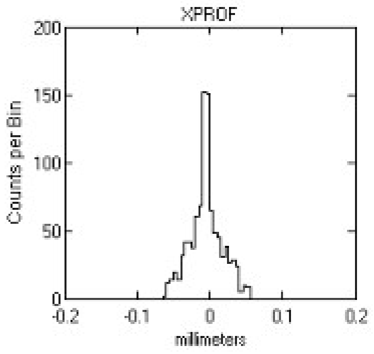
```
end
```

XPROF, *xbin, xc*

xbin = size of bins (extent/60.)
xc = x center for the histogram

XPROF creates histograms of the spot density as a function of x. Because of its one dimensional nature it is particularly suited to displaying resolution of spectrographs. If bin size is undefined then IRT calculates the full extent of the ray positions and divides it by sixty to form the bin size.

```
;EXAMPLE XPROF
_ti='XPROF'
objp,0.,0.,1.e12,1000,6328.
pupcr,0.,50.
disp,0.,0.,-1000.
conic,0.,1000.
mirror
disp,0.,0.,500.
flat
xprof,.005
end
```



Analysis Routines

PV	Print a Vector (ray)
QRMS	RMS of ray cosines
RFOC	Show Radial Focus vs Z axis
RMS	RMS of ray x and y values
XFOC	Show X Focus vs Z axis

PV, *n*

n = ray number to be printed (0)

PV creates a brief printout of the status of one ray. It prints ray 0 unless otherwise specified. PV is called automatically by each routine through use of the PV user option, but can also be called directly by the user.

```
;EXAMPLE PV
pri=1
objp,0.,0.,1.e12,10,6328.
pupcr,0.,50.
disp,0.,0.,-1000.
rot,20.,30.,0.
pv,1
end
```

Which creates the screen printout:

```
Point Object of 10 rays     at x = 0.000000e+00, y = 0.000000e+00, z = 1.000000e+12
wavelength = 6.328000e+03
Circular Random Entrance Pupil      radii, inner=0 , outer= 50
                                    at x=0.000000e+00 , y=0.000000e+00
Origin Moved to 0.000000e+00, 0.000000e+00, -1.000000e+03
Coordinate Rotation of 20.000000 about X-axis
Coordinate Rotation of 30.000000 about Y-axis
   num = 1   x = -4.795497e+02   qx = 4.698463e-01   lam = 6.328000e+03
   status=1   y = -3.552379e+02   qy = 3.420201e-01   nind= 1.000000e+00
                      z = 8.026403e+02      qz = -8.137977e-01
```

QRMS*(ax)*

ax = rms axis: 0=r, 1=x, 2=y (0)

QRMS is a function which returns as its value the root mean square of the ray direction cosines about the center of mass. If *ax* is 1 it returns the rms of the qx values, if *ax* is 2 it returns the rms of the qy values. If *ax* is 0 it returns the blur which is the square root of the sum of the squares of all the qx and qy values. In all cases the average value is subtracted so that the result is independent of direction.

```
;example qrms
objp,0.,0.,1.e12,100,6328.
pupcr,0.,50.
disp,0.,0.,-1000.
torus,1000.,500.
mirror
disp,0.,0.,500.
flat
print,'qrms in radial direction = ',qrms(0)
print,'qrms in x direction = ',qrms(0)
print,'qrms in y direction = ',qrms(0)
end
```

Which creates the following printout:

```
QRMS in radial direction = 1.15291e-01
QRMS in X direction = 4.63602e-02
QRMS in Y direction = 1.05560e-01
```

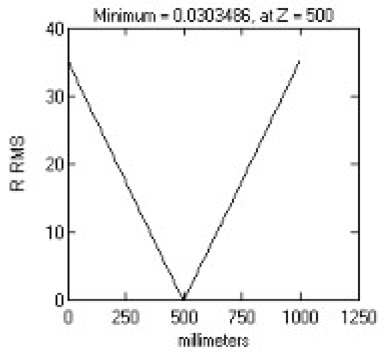
RFOC, *zmin, zmax*

zmin = minimum z to be considered (-1.)
zmax = maximum z to be considered (1.)

This routine is primarily for use after the raytrace is complete, and analysis of the result is desired. RFOC assumes that the rays are traveling more or less parallel to the z-axis. It splits the interval between *zmin* and *zmax* into 60 points and for each of these points it calculates the rms blur of the rays in the radial direction. It automatically plots the blur as a function of z over the *zmin* to *zmax* interval. It also prints the value of z at which the rms is a minimum, and the value of the minimum.

A typical application of this routine is for finding the best focal plane for an optical system. One or two runs of RFOC will rapidly yield information on the best focus.

```
;example rfoc
_ti='rfoc'
objp,0.,0.,1.e12,100,6328.
pupcr,0.,50.
disp,0.,0.,-1000.
conic,0.,1000.
mirror
rfoc,0.,1000.
end
```



RMS*(ax)*

ax = rms axis: 0=r, 1=x, 2=y (0)

RMS is a function which returns as its value the root mean square of the ray positions about the center of mass. If *ax* is 1 it returns the rms of the x positions, if *ax* is 2 it returns the rms of the y positions. If *ax* is 0 it returns the rms blur which is the square root of the sum of the squares of all the x and y positions. In all cases the average position is subtracted so that the result is independent of position.

```
;example rms
objp,0.,0.,1.e12,100,6328.
pupcr,0.,50.
disp,0.,0.,-1000.
torus,1000.,500.
mirror
disp,0.,0.,500.
flat
print,'RMS in radial direction = ',rms(0)
print,'RMS in X direction = ',rms(0)
print,'RMS in Y direction = ',rms(0)
end
```

Which creates the following printout:

```
RMS in radial direction = 2.68369e+01
RMS in X direction = 4.67547e-02
RMS in Y direction = 2.68368e+01
```

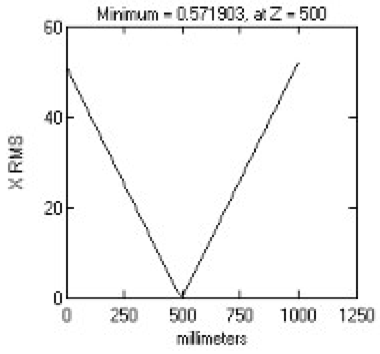
XFOC, *zmin, zmax*

zmin = minimum z to be considered (-1.)
zmax = maximum z to be considered (1.)

This routine is primarily for use after the raytrace is complete, and analysis of the result is desired. XFOC assumes that the rays are traveling more or less parallel to the z-axis. It splits the interval between *zmin* and *zmax* into 60 points and for each of these points it calculates the rms blur of the rays in the x direction. It automatically plots the blur as a function of z over the *zmin* to *zmax* interval. It also prints the value of z at which the rms is a minimum, and the value of the minimum.

A typical application of this routine is for finding the best focal plane for an optical system. One or two runs of XFOC will rapidly yield information on the best place to put the detector.

```
;example xfoc
_ti='xfoc'
objp,0.,0.,1.e12,100,6328.
pupcr,0.,50.
disp,0.,0.,-1000.
conic,0.,1000.
mirror
xfoc,0.,1000.
end
```



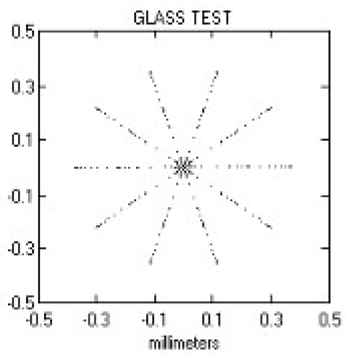
The IRT routine LENS needs index of refraction information to operate. This must be provided by the user. A single index can be used for all the rays in a call to LENS, or the user can explicitly provide a vector of indices. This vector can be calculated in a user-written routine for any special application. However, the most common use of the lens routine involves visible light and standard kinds of glass. IRT makes the indices of refraction for over 200 glasses from Schott Inc. easily available. They are stored in the ascii file glass.dat, and can be read directly.

GLASS*(mat, lam)*

mat = name of glass to be used ('BK7')
lam = wavelength array (6328.)

The index of refraction for each ray is calculated by GLASS and is returned as a double precision vector. The vector is stored in a user-defined parameter, which is then used as the calling parameter to LENS.
GLASS first opens the disk file called glass.dat, searching the three default directories. Within glass.dat it searches for a glass with the same name as *mat*. When it finds the file it uses the parameters associated with the glass and plugs them into an equation which gives the index as a function of wavelength. Hence, one must be careful that the wavelengths and wavelength units are consistent. The default units are angstroms, but will automatically convert it to the system parameter *wunits* if it has been set to one of the other standard options. If GLASS fails to find the glass it prints a warning message and returns all ones.

```
;EXAMPLE GLASS
pri=1
pv=0
_ti='GLASS TEST'
objp,0.,0.,1.E12,1,[4047.,6328.,10140.]
pupca,10.,10,10
disp,0.,0.,-10.
conic,0.,-100.
inds = glass( 'BK7', lam)
lens,inds, -100.,10.,100.
disp,0.,0.,-91.6134
flat
spot
end
```



Optimization Routines

Optimize	Old Optimizer
Simplex	Simplex Optimizer

The optimization routines in IRT have been written to be very general in nature.
The user creates a raytrace which uses parameters to define the aspect of the design that needs optimization.   Included in the .irt file the user defines a parameter called *merit*, which measures quantitatively the goodness of the raytrace. The smaller the value of merit, the better the design.   Then, the user defines the parameters to be optimized and calls OPTIMIZE or SIMPLEX.   The process is best understood by example.
As a rule, SIMPLEX is superior to OPTIMIZE.   It will find the optimum faster, and will be less likely to give up the search before finding the true optimum.   OPTIMIZE uses a linear parameter search along each axis independently.   If the optimizing parameters are correlated, this can lead to very lengthy runs.   SIMPLEX, using the simplex optimization mathematics, identifies the optimum direction for improvement and moves that way.
These routines have been written to give complete generality in approach to parameter optimization, but without sacrificing ease of use for the operator.

OPTIMIZE, *file, par0, par1, -- ,par9*

file = name of .irt file to be optimized
par = names of parameters to be optimized

Optimize will change the value of the parameter as the search proceeds. When the search stalls, the user can reduce the step size and continue. It is better to use the array versions of pupil and objects, otherwise the random errors will interfere with the smooth convergence of the optimization.
An Example: Create a file called OPT2.IRT that contains the following commands:

```
objp,0.,0.,1.e12,1,6328.
pupca,50.,5,5
disp,0.,0.,-300.
conic,e,rad
mirror
disp,0,0,300.
flat
merit=rms(0)
end
```

Then type the following commands:

```
rad=650.
e=1.1
optimize,opt2,rad,e
```

Here we have a trace where the parameters to be optimized are the radius of curvature and eccentricity of a conic section mirror. The figure of merit is simply the rms spot size on-axis. The optimum solution is, of course, a parabola (e=1) with a radius of curvature of 600 that will focus on the defined focal plane. OPTIMIZE will find this solution. However the closer one starts to the solution, the faster IRT will converge.
For a more sophisticated example consider a design which optimizes the eccentricity and radius of curvature so that the on-axis and two degrees off-axis response are at the best balance. Change OPT2.IRT to be:

```
objp,0.,0.,1.e12,1,6328.
pupca,50.,5,5
disp,0.,0.,-300.
conic,e,rad
mirror
disp,0,0,300.
flat
merit=rms(0)
objp,6.e10,0.,1.e12,1,6328.
pupca,50.,5,5
disp,0.,0.,-300.
conic,e,rad
mirror
disp,0,0,300.
flat
merit=merit+rms(0)
end
```

Then type:

```
rad=600.
e=1.
optimize,opt2,rad,e
```

The optimization search then starts forcing the eccentricity lower, toward spherical, and slightly increasing the radius of curvature. In fact, IRT forces the value of e negative, which is a non-physical solution. In such a case, the user should intervene and set the value of e to its minimum acceptable value, then optimize the remaining parameters. With e set to 0. the best radius of curvature is 601.354.
Optimization makes use of two system parameters. First, is *merit*, which IRT uses as the definition of the measure of the optimization of the solution. If *merit* is not defined in the .irt file, then optimization cannot proceed. The second parameter is *stepsize*. This parameter is used to let the algorithm estimate how accurately the parameters must be specified. The default for *stepsize* is .0001, which suffices for many applications. However, if precision does not need to be so good, an increase is advised, simply by setting the parameter (eg stepsize=.1). If precision is needed in excess of .0001, then stepsize should be reduced. Sometimes in an optimization, the precision required of the multiple parameters to be simultaneously optimized varies widely. For example, if one parameter specifies the focal length, then precision of .01 is ample, while another parameter may specify a deformation and require precision of 1.e-14. In such a case, one can set *stepsize* to 1.e-14, and proceed. This will slow it greatly, and in some cases cause a failure to properly converge. Two solutions are available. First, one can proceed to optimize independently. This works if the parameters are not closely coupled. An alternative is to optimize a dummy parameter that needs the same stepsize, and then is modified before being used in the .irt file. For example, a parameter called *delta* may require precision of 1.e-14. Instead, optimize a parameter called *delta2*, and include a line delta=delta2*1.e-10 in the .irt file before *delta* is used.

SIMPLEX, *file, par0, par1, -- ,par9*

file = name of .irt file to be optimized
par = names of parameters to be optimized

SIMPLEX will change the value of the parameter as the search proceeds. It is better to use the array versions of pupil and objects, otherwise the random statistical errors will interfere with the smooth convergence of the optimization.
An Example:
Create a file called OPT2.IRT that contains the following commands:

```
objp,0.,0.,1.e12,1,6328.
pupca,50.,5,5
disp,0.,0.,-300.
conic,e,rad
mirror
disp,0,0,300.
flat
merit=rms(0)
end
```

Then type the following commands:

```
rad=650.
e=1.1
simplex,opt2,rad,e
```

Here we have a trace where the parameters to be optimized are the radius of curvature and eccentricity of a conic section mirror. The figure of merit is simply the rms spot size on-axis. The optimum solution is, of course, a parabola (e=1) with a radius of curvature of 600 that will focus on the defined focal plane. SIMPLEX will find this solution. However the closer one starts to the solution, the faster IRT will converge.
For a more sophisticated example consider a design which optimizes the eccentricity and radius of curvature so that the on-axis and two degrees off-axis response are at the best balance. Change OPT2.IRT to be:

```
objp,0.,0.,1.e12,1,6328.
pupca,50.,5,5
disp,0.,0.,-300.
conic,e,rad
mirror
disp,0,0,300.
flat
merit=rms(0)
objp,6.e10,0.,1.e12,1,6328.
pupca,50.,5,5
disp,0.,0.,-300.
conic,e,rad
mirror
disp,0,0,300.
flat
merit=merit+rms(0)
end
```

Then type:

```
rad=600.
e=1.
simplex,opt2,rad,e
```

The optimization search then starts forcing the eccentricity lower, toward spherical, and slightly increasing the radius of curvature. In fact, IRT forces the value of e negative, which is a non-physical solution.   In such a case, the user should intervene and set the value of e to its minimum acceptable value, then optimize the remaining parameters.   With e set to 0. the best radius of curvature is 601.354.

SIMPLEX makes use of four system parameters.   First, is *merit*, which IRT uses as the definition of the measure of the optimization of the solution. If *merit* is not defined in the .irt file, then optimization cannot proceed.   The second parameter is *verbose*.   If it is set to one, then IRT will provide a printout of the value of merit at the end of each iteration, otherwise, it will not report back until the end of the session, which may make you feel abandoned and tempted to reboot the system.

The parameter *stepsize* is used to let the algorithm estimate how accurately the parameters must be specified.   The default for *stepsize* is .001, which suffices for many applications. However, if precision does not need to be so good, an increase is advised, simply by setting the parameter (eg   stepsize=.1).   If precision is needed in excess of .001, then *stepsize* should be reduced.

The fourth parameter is *scale*.   It is initialized to 1, and that is usually sufficient.   *scale* sets how far outward in parameter space the algorithm tests the value of merit.   If you encounter difficulties with SIMPLEX converging in a locally correct, but globally wrong direction, you can try increasing or decreasing *scale*.

Sometimes in an optimization, the precision required of the multiple parameters to be simultaneously optimized varies widely.   For example, if one parameter specifies the focal length, then precision of .01 is ample, while another parameter may specify a deformation and require precision of 1.e-14.   In such a case, one can set *stepsize* to 1.e-14, and proceed. This will slow it greatly, and in some cases cause a failure to properly converge.   Two solutions are available. First, one can proceed to optimize independently.   This works if the parameters are not closely coupled.   An alternative is to optimize a dummy parameter that needs the same stepsize, and then is modified before being used in the .irt file.   For example, a parameter called *delta* may require precision of 1.e-14.   Instead, optimize a parameter called *delta2*, and include a line   delta=delta2*1.e-10 in the .irt file before *delta* is used.

MULTIPATH SYSTEMS

The procedures described in this chapter have several purposes. First, they greatly simplify the tracing of nested optical systems such as are frequently encountered in grazing incidence systems. Second, they allow beams to be split and recombined after traveling different optical paths such as in interferometers. Third, they allow entirely separate traces to be easily combined.
The structure of a basic multipath system looks like the following:

```
object
pupil
tracing
multipath
group
tracing
groupend
group
tracing
groupend    etc.
multiend
tracing
end
```

The way this works is that MULTIPATH establishes that the beam is about to be split. It stores the information about the present state of the rays in a disk file that the user does not see. GROUP identifies that a subpath is being started. Tracing of that particular subpath then proceeds until GROUPEND is encountered. GROUPEND stores the traced subpath rays in yet another file, or if this is not the first subpath appends them to the rays already there. GROUPEND then restores the rays in the state they were stored by MULTIPATH. Another GROUP is then started. This continues with rays being accumulated until MULTIEND is encountered. MULTIEND puts the accumulated rays in the normal common block so that raytracing can continue from there as normal. The MULTIPATH environment deletes all files with a .tem extension. Therefore, the user should not keep files by that name in the directory being used for tracing.
Note that each group may have a different set of coordinate changes, and thus rays cannot be simply put back together. To handle this GROUPEND moves the rays to the PUPIL coordinates via UNTRACE before storing them. MULTIEND returns the rays to the coordinate system that was in place when MULTIPATH was called.

MULTIPATH

MULTIPATH sets up the splitting of the beam by storing the rays information as described above.

GROUP

GROUP starts a subgroup to be traced through a subpath that follows.

GROUPEND

GROUPEND closes out and saves the group being traced and restores the original ray set.

MULTIEND MULTIEND closes out the splitting of the beam and return all rays to the coordinates used when MULTIPATH was encountered.

```
;example of multipath
;four co-aligned spherical mirrors
_ti='multipath'
objp,0.,0.,1.e12,500,6328.
pupcr,0.,1000.
disp,0.,0.,-1000.
;
multipath
;
group
disp,-70.,0.,0.
mask,1,0.,300.
conic,0.,2000.
mirror
groupend
;
group
disp,70.,0.,0.
mask,1,0.,300.
conic,0.,2000.
mirror
groupend
;
group
disp,0.,-70.,0.
mask,1,0.,300.
conic,0.,2000.
mirror
groupend
;
group
disp,0.,70.,0.
mask,1,0.,300.
conic,0.,2000.
mirror
groupend
;
multiend
disp,0.,-70.,1000.
flat
spot
end
```

TOLERANCE ANALYSIS

The simulation of tolerance errors is initiated by setting *tolon* to 1.   The default is 0, and causes no tolerance adjustments to be made.   The values of the various tolerances can be set at any time and are used only when *tolon* is 1.

Position and Orientation

To simulate an error in position or orientation IRT internally calls DISP and ROT, calls the optical element, and then calls ROT and DISP to reverse the misplacement.   The displacement is called first, with a call DISP,*tolx,toly,tolz* followed by ROT,*tolqx,tolqy,tolqz.* These statements are called automatically at the beginning and end of each surface. Diagnostics listing the coordinate changes are given if the print diagnostics are turned on.

```
;example position tolerance
; a rowland circle mount
_ti='conicgrat'
tolx=10.
toly=.1
tolz=.1
tolqx=.01
tolqy=.01
tolqz=.01
d=2156.
objcr,0.,5.e10,0.,0.,1.e12,100,6000.
pupp,0.,0.
disp,0.,0.,-396.11
rot,0.,-8.,0.
tolon=1
conic,0.,400.
grating,1,d
tolon=0
rot,0.,188.,0.
disp,0.,0.,-396.11
flat
spot
end
```