GETTING STARTED

What is IRT?

The Interactive RayTrace Program (hereafter referred to as IRT) is a software package specifically written to aid in the design, development and analysis of complicated optical systems. It is designed for use in the infrared, visible, ultraviolet and x-ray bands of the spectrum.

Versions of IRT

IRT comes in a variety of versions, suiting a wide range of hardware needs. The front end varies depending on the machine and operating system. The back end is nearly identical for all the versions. Series 1, 2 and 3 of IRT were written for use on workstations with VMS and UNIX environments. Series 4 is written for the DOS environment. Series 5 is for Windows

Why Use IRT?

IRT is fast, versatile and easy to use. From its inception IRT has been designed to be used interactively. The process of optical design and evaluation is iterative; the user begins with an idea or plan, evaluates it, implements indicated changes, and then re-evaluates. Such a process, requiring human intervention, is best handled in an interactive mode.

IRT was originally written in IDL (Interactive Data Language), a language marketed by Research Systems Inc. (RSI) of Denver, CO. IDL is a language specifically tailored to the needs of engineers and scientists. It has a versatile and easy-to-use plot package and a structure that handles vectors and arrays as easily as scalars.

However, IDL is not available on most machines, and is beyond the scope of the needs of many designers who could use the capabilities of IRT. To this end Parsec Technology has rewritten IRT based in the C language so that it can be easily transportable to the majority of machines.

How Does IRT Work?

IRT can be run in either an interactive mode whereby each command is entered by the user and executed immediately, or by creating programs (referred to as procedures) that are stored on disk and executed sequentially.

In IRT the user creates a trace procedure that is run as a program. Each procedure is composed of a series of calls to special purpose subroutines that have been incorporated into IRT. The effect is one of having a programming language for raytracing. The user runs a raytrace and, at the end of a run, the ray information is still available to the user in the interactive mode. IRT then provides a variety of routines that can be used interactively for analyzing the results. The procedure can be modified for subsequent runs of the program, or predefined variables can be reset by hand for changing a parameter on a subsequent run.

IRT is a geometric optics code that takes all rays through the system simultaneously, element by element. Thus it is possible to investigate the evolution of the light rays as they pass through a system.

How Does IRT Handle Complex Optical Systems?
The code's structure allows the user to develop special purpose routines which can be accessed on call. Users can develop their own special purpose libraries of routines. Entire trace procedures may be turned into subroutine calls of higher level routines, freeing the user to think only about the important or new part of the routine.

GETTING STARTED

IRT for windows uses the Windows Multiple Document Interface, so it has some of the look and feel of the Program Manager and File Manager programs that come with Windows. The top bar and menu bar belong to a frame window. The active windows are child windows of the frame window. IRT supports three varieties of child windows. When brought up fresh, IRT presents one of each kind, as above. In the upper left is an *Edit* window. The upper right is a *Graph* window, and across the bottom is a *Command* window. At any time, the active window has its title bar highlighted; in this case it is the *Edit* window that is active.

For a quick demonstration click on the command window to make it active. A caret should appear after the irt> prompt. Type *example*, and then hit the enter key. The screen should now look like this:

Notice that the Command window is now active, so it has the highlighted title bar, and that a spot diagram has appeared in the Graph window. Notice also that the items on the menu bar at the top changed when the focus moved from an Edit window to a command window.

You can create a new Graph window by choosing Window in the main menu and then picking New Graphs. (Choose the Window menu item by clicking on it with the mouse, or by using the alt-w key.) A new Graph window will appear in the center of your screen. One way to arrange the windows is to choose Window and then choose Tile. This will cause your screen to appear as four similar size rectangles.

If you type *example* again in the command window, then another graph will appear in the Graph 2 window, similar to that in Graph 1. IRT sends plot commands to the most recently selected graph window.

Numerous windows of each kind can be created and arranged to suit the users needs. They can be stretched and iconized in the usual manner. In the following sections we introduce each of the three kinds of windows and the commands that accompany it.

The Command Window

The Command window is a text interface similar to that used in DOS, UNIX, and VMS environments. The user types an IRT command and then sends it to the program by hitting the enter key. When the program is ready for more input it gives the user an irt> prompt. IRT can be run completely from the Command window, with text commands, however, a combination of text commands and mouse selections is, on the whole, more convenient and more powerful.

When a Command window is selected, three choices appear on the main Menu: File, Window and Help. When File is selected, you are given a choice of Save, Save As, Print, and Exit. Save saves the contents of the command window to an ASCII text file with the default, or previously set file name. Save As does the same thing, but allows the user to change the name of the file first, through a standard windows dialog box. Print sends the text output directly to the printer. Exit is a route out of IRT: it terminates execution and returns the user to Windows. It first, however, checks to be sure that is what you meant.

File

Save	Saves contents of command window as text file
Save As	Saves contents, prompts for new file name
Print	Outputs contents on printer
Exit	Exits IRT after verification

If you select the Window menu item, then you get the following choices of ways to work with the window selection and arrangement:

Windows

New Command	Create new command window
New Graphs	Create new graphics window
New Edit	Create new edit window
Cascade	Arrange windows in cascade
Tile	Arrange windows in tiles
Arrange Icons	Arrange icons neatly
Close All	Close all the windows
Graph 1	Activate Graph window #1
Command	Activate Command window
Edit 1	Activate Edit window #1

Help

Help	Tells user to use command window for help
About IRT	Gives version and copyright information

The Edit Window

The edit window allows the user to bring the raytrace code into a screen editor and to run and save modifications from the window.

File

Save	Saves contents of edit window as text file
Save As	Saves contents, prompts for new file name
Compile	Save contents, then compile, but do not run
Run	Save, compile, and run the trace file
Print	Outputs contents on printer
Exit	Exits IRT after verification

Edit

Undo	Undo last edit command
Cut	Remove selected text to clipboard
Copy	Copy selected text to clipboard
Paste	Insert text from clipboard
Delete	Delete selected text
Select All	Select all the text

Search

Find	Find text written in popup box
Find Next	Find the next instance of the text string
Replace	Find a string and replace it

The next menu item is Run. This is the same as Run under File. It saves, compiles, and runs the edited version of the code, overwriting the old version on disk. This is in the main menu because it is used heavily.

Windows

New Command	Create new command window
New Graphs	Create new graphics window
New Edit	Create new edit window
Cascade	Arrange windows in cascade
Tile	Arrange windows in tiles
Arrange Icons	Arrange icons neatly
Close All	Close all the windows
Graph 1	Activate Graph window #1
Command	Activate Command window
Edit 1	Activate Edit window #1

Help

Help	Tells user to use command window for help
About IRT	Gives version and copyright information

The Graph Window

The graph window is where the graphics output is displayed. It is not directly interactive, but has its own menu to support graphics interaction. Similarly, graphics can be controlled from the command window, or from .irt files.

File

Save	Saves graph as device independent bitmap into file irt.bmp
Save As	Saves graph as device independent bitmap, queries file name
Print	Outputs graph on printer
Exit	Exits IRT after verification

Display

SPOT	Spot plot. Same as typing spot in command window
XPROF	X profile plot. Same as typing xprof in command window.
MTF	Modulation transfer function. Same as MTF command.
VIEW	View elements. Same as VIEW command.
RAYS	Show ray paths. Same as RAYS command.

Colors

Foreground	Set plot foreground color
Background	Set plot background color

Windows

New Command	Create new command window
New Graphs	Create new graphics window
New Edit	Create new edit window
Cascade	Arrange windows in cascade
Tile	Arrange windows in tiles
Arrange Icons	Arrange icons neatly
Close All	Close all the windows

Graph 1	Activate Graph window #1
Command	Activate Command window
Edit 1	Activate Edit window #1

Help

Help	Tells user to use command window for help
About IRT	Gives version and copyright information

STRUCTURE:

Running IRT requires four steps:

1. Run IRT.EXE program.
2. Create a trace procedure using the editor.
3. Run the trace procedure.
4. Analyze and output results.

Starting IRT

IRT carries its own initial parameters and defaults as described throughout this manual, but the user can set personal defaults automatically. When IRT is invoked it automatically looks for a file called DEFAULTS.IRT and runs it as a raytrace. DEFAULTS.IRT contains statements defining parameters and, in fact, can be an entire raytrace. IRT searches for this file first in the local directory, and then in the directory called \IRT.

Procedures

The next commands from the user set any variables and options, and initiate the TRACE procedure which has been written at an earlier time. If the user types the command *trace*, IRT will search for a file called TRACE.IRT on disk, and execute the commands it finds therein. The structure of the TRACE procedure is discussed extensively in Chapter 2. Upon completion of execution the computer returns the IRT> prompt and waits for further instructions. Since the raytracing has presumably been completed, the user will then display and analyze the results using IRT commands.

If corrections are desired (and they usually are) then the user edits the TRACE.IRT file as he/she would any source program. This can be done without leaving IRT by using the Edit window. Procedures should, of course, have names other than TRACE to avoid confusion.

Very sophisticated tracing can be performed. Individual pieces of a trace procedure can be broken off and turned into subroutines. Thus a fully developed part of an optical system can be relegated to a single line of main program and be easily handled. Hierarchies of code can be developed by the user in this way.

Online Help

IRT provides some basic online help to aid in ease of interactive sessions. The help is rudimentary and not meant to be a substitute for this manual. To access help, type *help* in response to the IRT> prompt. For help in a specific area, type *help,topic* (e.g. *help,conic*).

Common Variables

Within IRT are a large number of permanent subroutines, each with a special function. They all must access the ray information, so the rays are kept in common storage in RAM. The values of the rays can be accessed by the user using either the *print* or *pv* command.

x,y,z are each one dimensional vectors of double precision numbers. The first element of each vector gives the current x, y or z coordinate of the first ray. The second element of each give information about the second ray, and so on. The *qx,qy,qz* vectors contain the three direction cosines for each ray; combined they form the unit vector for direction. *lam* contains the wavelength of each ray. *nind* contains the index of refraction for the ray in the current material. *ux, uy,* and *uz* are the direction cosines of the unit normal vector for the last surface at the position where the ray struck. *status* is a long integer status flag for each ray. It is 1 if the ray is active, 0 if the ray has been vignetted.

IRT also stores the information detailing the way the rays were prior to the most recent surface. They serve a number of useful purposes including checking that the rays are traveling forward.

Other common variables exist that carry basic information such as the relation of the current coordinate system to the initial system, and system variables such as the flags that turn on and off the various options.

CREATING A TRACE PROCEDURE

Structure Of A Trace Procedure

The series of commands defining the system to be traced is stored in a text file on disk. The name of the procedure is arbitrary, but it must have a .irt extension. We shall assume it is called TRACE.IRT. The tracing is started by the command *TRACE.* If trace.irt is anew file (i.e. one that has not been used since IRT was initiated) then the file will be read into RAM , stored, and executed. Subsequent calls will use this version stored in memory.

The TRACE procedure will always have the same sequence of types of commands.

OBJECTS One or more object routines which define the starting positions and
wavelengths of the rays to be traced. The OBJECT call simulates

the light source.

PUPIL	A single routine which defines the entrance pupil of the system to be. The PUPIL call moves the rays to their starting positions and defines their directions.
ELEMENTS	A series of calls to subroutines defining the optical system.
ANALYSIS	Routines which are used for analyzing the results of the trace. These are often called by the user in an interactive mode.
DISPLAY	Routines for graphic representation of the results of trace. These are usually called by the user in an interactive mode.

In the remainder of this chapter we take the user through the steps which define a simple raytrace, showing by way of example how a TRACE is created and used. The example will be to raytrace a simple parabolic mirror.

CREATING RAYS

OBJECT and PUPIL Before rays can be traced through an optical system they must be defined as to position, direction, wavelength etc. In IRT this is usually accomplished by a pair of procedure calls. *The first call is to define an OBJECT, the actual source of the rays*. The position, shape and wavelength of the origin of the rays are defined in this call. Multiple objects can be called to superimpose the ray sources. The IRT supported OBJECT routines are all called by names that start with the three letters OBJ.

The **OBJECT(s)** must be followed by a single call to an entrance PUPIL routine. These routines are all called by names that start with PUP. They define the entrance aperture to the optical system. By defining the positions where the rays strike the entrance pupil, and comparing to the positions where the rays originated in the object, the directions of the rays are defined. After the pupil call the vectors in common are fully defined and the program is ready to proceed to the optical elements.

The choices for OBJECT routines are:

OBJCA	circular shape, area weighted array
OBJCR	annular shape, uniform random distribution
OBJFAN	cross or line object
OBJGR	normally distributed random distribution
OBJP	single point
OBJSA	rectangular shape, area weighted array
OBJSR	rectangular shape, uniform random distribution

We will choose OBJP as we wish to investigate the aberrations introduced by a mirror. OBJP simulates an ideal point source of light. Its calling parameters are:

XC	x position of object
YC	y position of object
OBD	z position of object (object distance)
NR	number of rays to be generated at each wavelength
LAM	wavelength of rays (floating point scalar or vector)

For XC we start by choosing on-axis rays. Since the pupil will be centered at the origin and lie in the Z=0 plane, we choose XC=0. and YC=0. To investigate parallel light from infinity we would like OBD to be infinite, but as computers have a difficult time with the concept of infinity we instead choose a very large number. OBD=1.E12 is a good choice, being far enough away to effectively be infinity, but not so large that the computer is likely to overflow or underflow. NR is a matter of choice. The larger the number, the slower the system will run. A typical trace contains about 100 rays. Let NR be 100. Let LAM be 5000.

The first line of the procedure will thus be:

```
    objp,0.,0.,1.e12,100,5000.
```

Type this line into the edit window. After the object we must define a pupil. The choices of pupil type are:

PUPCA	annular shape, weighted array of points
PUPCR	annular shape, uniform random distribution of points
PUPFAN	cross or line pupil
PUPP	single point
PUPSA	rectangular shape, array of points
PUPSR	rectangular shape, uniform random distribution

For our purposes any of the above (except PUPP) will do. To simulate random distribution of light on a circular mirror use PUPCR. Its calling parameters are RI and RO, which are the inner and outer radii of the annulus. For a full circle RI is 0. Let us investigate a 100mm diameter mirror. Then RO will be 50. The pupil will lie in the z=0 plane centered on the origin. Add an end statement to mark the last line of the program.

Our routine now looks like:

```
objp,0.,0.,1.e12,100,5000.
pupcr,0.,50.
    end
```

This is now a fully executable trace procedure.

TRACE ELEMENTS

Once the rays have been defined we proceed into the main body of the trace. The main body typically consists of two types of commands; optical element procedures that define the surfaces, and coordinate procedures that define the spatial relationships of the optical surfaces.

Coordinate Routines

The relative position and orientation of the optical elements are controlled primarily through the use of two IRT procedures: DISP and ROT. DISP moves the position of the origin from its current position to another, thereby changing the position values of the rays. ROT rotates the coordinate system about the origin, thereby modifying the direction and position of the rays. With this combination of routines it is easy to move from the coordinate system of one element to that of another.

For our example it is now necessary to decide where the mirror rests with respect to the entrance pupil. Let us assume that it is 500mm behind the pupil. Since the rays were generated at Z=1.E12 and traveled to Z=0 at the pupil, they are traveling in the negative z direction. Thus the mirror should be placed at Z=-500. Because mirrors are defined to have their vertices at the origin, we must move the origin to the place which is currently [0.,0.,-500.]. This is accomplished by using DISP. The call is DISP,0.,0.,-500. After this call is executed all the rays will still be at the pupil, but their Z values will all be 500.

Optical Elements

These are the routines that actually simulate the physical elements of a system. They include mirrors, gratings and lenses. Each class of element is given its own procedure. Most commercially available element designs are included in the basic IRT procedures.

All routines have certain assumptions in common. They all have the Z-axis as their axis of symmetry, and (where appropriate) they have their vertex tangent to the X-Y (Z=0) plane at the origin. One-dimensional optics have curvature as a function of x only. Gratings have grooves that project onto the X-Y plane as straight lines parallel to the Y-axis.

Our example requires a parabolic mirror. Conic sections of rotation are supported by the routine CONIC. It has two parameters: *e* and *rad.* *e* is the eccentricity, which is 1 for a parabola. *rad* is the radius of curvature at the vertex. If we wish to investigate an f/10 parabola then *rad* should be 20 times the diameter of the mirror. Thus we wish *rad* to be 2000. We call MIRROR afterwards to communicate that the element is a mirror (as opposed to a lens, grating, etc.).

Our routine now looks like:

```
objp,0.,0.,1.e12,100,5000.
pupcr,0.,50.
disp,0.,0.,-500.
```

```
conic,1.,2000.
mirror
end
```

After CONIC the rays will have traveled to their intersection with the surface of the parabola. They are left on the surface with their directions unchanged. The call to MIRROR leaves the positions unchanged but changes the directions to those they will be going after reflection. Since optical element routines leave the rays at the optical surface we must move the rays to the focal plane before we are finished. This can be accomplished by moving the origin to the focal plane and then calling FLAT. The coordinate change is achieved by the call DISP,0.,0.,1000.

Thus our final, ready to run routine, looks like:

```
objp,0.,0.,1.e12,100,5000.
pupcr,0.,50.
disp,0.,0.,-500.
conic,1.,2000.
mirror
disp,0.,0.,1000.
flat
end
```

We now leave the editor. Use the *save as* feature under File to save this text in a file called trace.irt. We are ready to run the program. This can be simply performed by clicking RUN in the menu, or by switching focus to the command window and typing *trace*.

In the command window type the following.

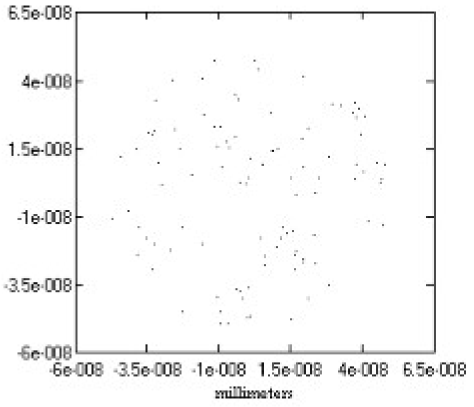
```
IRT>pri=1
IRT>trace
Point Object of 100 rays    at x = 0.000000e+00, y = 0.000000e+00, z =
1.000000e+12    wavelength = 5.000000e+03
Circular Random Entrance Pupil     radii, inner=0 , outer= 50
at x=0.000000e+00 , y=0.000000e+00
Origin Moved to 0.000000e+00, 0.000000e+00, -5.000000e+02
Surface Number 1
Conic Section of Rotation
eccentricity = 1.000000      rad          = 2000.000000      sol          =
0
Mirrored Surface
Origin Moved to 0.000000e+00, 0.000000e+00, 1.000000e+03
Surface Number 2
Flat Surface in the z=0 Plane
IRT>
```

In this example the commands that follow the IRT> prompt are typed by the user. At the end of execution the user is returned to the interactive environment, and the information about the rays is available for inspection directly or for continued tracing. For example, typing *pv* will result in a screen listing of the status of ray 0.

DISPLAY ROUTINES

After the execution of TRACE is complete, the user typically wishes to view the results of the computation. With a graphics terminal a number of different IRT routines can be used to display the results. This is accomplished by typing the command interactively, but it is equally valid for including a display procedure in the TRACE routine.

The most typical thing to do after the IRT> prompt at the end of a trace is to respond: IRT>*spot* This command will generate a diagram similar to that in the next figure. As parabolas focus on-axis rays perfectly, what one is seeing in the figure is machine error.



ANALYSIS ROUTINES

Analysis of the results in some quantitative way is equally important to the process of raytracing. Thus IRT supports some basic analytic tools. For example, the command *print*,*rms*(0) will yield the root mean square size of the photon distribution.

CHANGE AND TRY AGAIN

After running TRACE and looking at the result, one typically answers the first question and generates another. The result is that the code should be modified and run again.

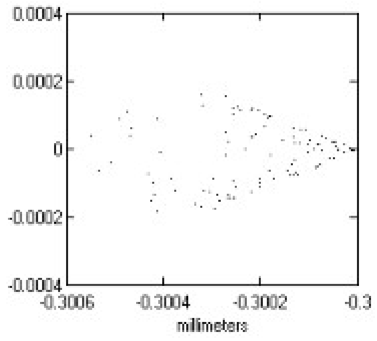
Continuing with our example let us imagine that we now wish to examine the off-axis performance of the parabola. Enter the edit window and change the first line so that it reads:

objp,ox,0.,1.e12,100,5000.

To run this new version of TRACE one must first define the variable *ox* which now appears in the first line. *ox* represents the distance off-axis of the source. Thus to investigate the parabola one arcminute off-axis do the following in the command window:

IRT>*ox=3.e8*
IRT>*trace*
IRT>*spot*

This results in something that looks like:



To check it 2 arcminutes off-axis it is only necessary to type:

IRT>*ox=6.e8*
IRT>*trace*

This completes our simple example.

As another simple example suppose that we now wish to establish the position of the best focus for the same parabola if the source of light is a point ten meters from the mirror. Enter the edit window and type the following lines:

```
pri=1
objp,0.,0.,1.e4,100,5000.
pupcr,0.,50.
conic,1.,2000.
mirror
focus,0
end
```

Use the *save-as* command to save this procedure as trace2.irt.

```
IRT>trace2
Point Object of 100 rays     at x = 0.000000e+00, y = 0.000000e+00, z =
1.000000e+04
                                                  wavelength = 5.000000e+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, -5.000000e+02
Surface Number 1
Conic Section of Rotation        eccentricity = 1.000000        rad =
2000.000000       sol = 0
Mirrored Surface
Focus Found
Origin Moved to -2.083626e-04, -9.164620e-05, 1.105255e+03
IRT>
```

The goal has been accomplished. The focus lies 1105.2mm above the vertex of the parabola. To leave IRT respond to the prompt with *exit*, or double click on the button in the extreme upper left.