

Application of Zemax Programming Language

Open Source Photonics

osphotonics.wordpress.com

Sponsored by

www.612photonics.com

Table of Contents

Preface

Chapter 1 Zemax optical design software and Zemax Programming Language (ZPL)

1.1 Introduction to Zemax

1.2 Introduction to Zemax Programming Language ZPL

Chapter 2 Basics of Zemax Programming Language

2.1 Basic Structure

2.2 Variable and Constant

2.3 Function

2.4 Keywords

2.5 Flow Control

2.6 Sub-Function

2.7 I/O and File Operation

Chapter 3 ZPL commands in details

3.1 Numerical Operation Functions

3.2 String Functions

3.3 Setting and Reading Zemax System Properties

3.4 Setting and Reading Lens Properties

3.5 Merit Function

3.6 Solve

3.7 Optimization

3.8 Ray Tracing

3.9 System Analysis

3.10 Non-Sequential Components

3.11 Multi-Configuration

3.12 Display

3.13 File Operation

3.14 ZBF File

Chapter 4 ZPL Application Examples

4.1 Sequential Optical Systems

4.1-1 Basic ray-tracing parameters

4.1-2 Light spot near focal plan

4.1-3 Geometrical beam and Gaussian beam comparison

4.1-4 Comparison of transmission property of different glass materials

4.1-5 Reading refractive index and transmission data of catalog glass

4.2 Non-Sequential Optical Systems

4.2-1 Light Pipe

4.2-2 Cosine Fourth Rule

4.2-3 Importance sampling

4.2-4 Interference fringes

4.2-5 Efficiency of the integrating sphere

4.2-6 Generating 3D light distribution with Detector Volume

Preface

The rapid development of modern computer technologies greatly changed the method and efficiency of optical design. Work previously can only be done by a few experts now becomes much easier with the aid of powerful optical design software such as Zemax. Any optical engineer, after trained, can now easily design complex optical systems.

Zemax is popular because it is powerful, flexible, easy to learn, and cost-effective. Besides many standard functions, Zemax also provides a tool called Zemax Programming Language (ZPL). This tool allows us to extend the standard function of Zemax to meet our special needs. In fact, this tool is so helpful that more and more Zemax users are trying hard to learn it and use it in their design works.

On the other hand, the learning process is usually not smooth, sometimes even quite frustrating. Since we have gone through this process ourselves, we fully understand that we need help during our learning. This is why we published a series of blogs on *osphotonics.wordpress.com* to share what we learned in the past and hope they can help readers to learn ZPL quicker and easier. Some of the examples and plots are based on older versions of Zemax, and some are based on more recent versions. However, the main idea should remain the same. We encourage readers to refer to official Zemax User's Manual for the updates on ZPL.

Chapter 1

Zemax optical design software and Zemax Programming Language

1.1 Introduction to Zemax

Zemax is a commonly used optical design program for Microsoft Windows sold by American company Radiant Zemax. It is used for the design and analysis of both imaging and illumination systems.

The main method used in Zemax is Ray Tracing, including Sequential Ray Tracing and Non-Sequential Ray Tracing.

In Sequential Ray Tracing mode, Zemax defines an optical system as being made up of various surfaces, and assumes that a light ray starts from the object surface, goes through the various surfaces of the system in a pre-defined order, and finally reaches the image surface. Figure 1.1-1 shows an example of Sequential Ray Tracing:

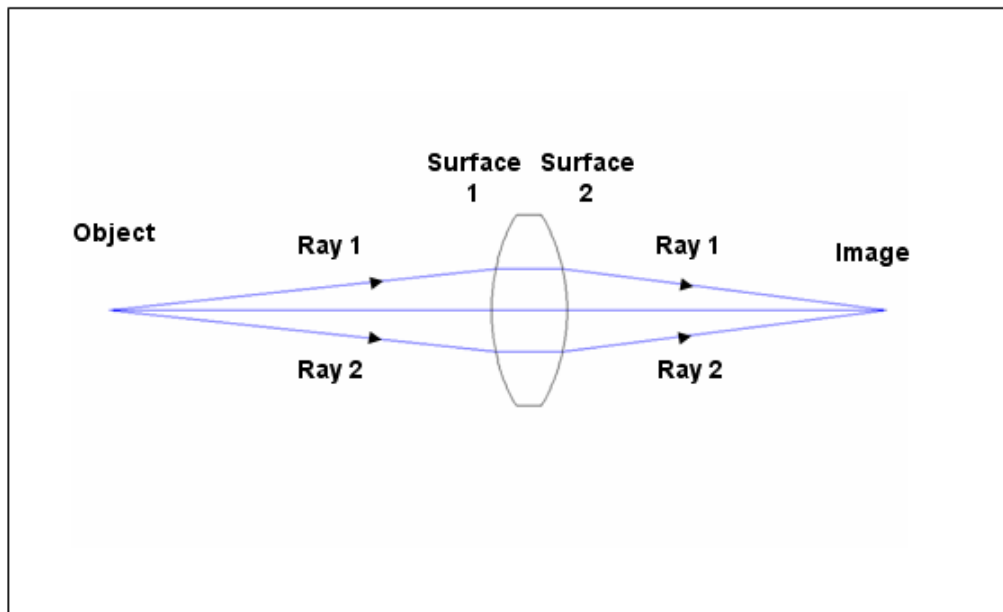


Fig. 1.1-1 Sequential ray-tracing.

As we can see, Ray 1 starts from the Object surface, goes through Surface 1 and Surface 2, and then arrives at Image surface. Ray 2 goes along a different path, but the sequence of the surfaces it goes through is exactly the same as that for Ray 1. In another word, the order each ray goes through the optical surfaces is exactly the same, therefore, the behavior of each ray in the optical system is predictable. By tracing the light path of each ray, Zemax knows the performance of the whole system.

However, in some other cases, different ray goes through the surfaces of the same optical system in different orders. Therefore, non-sequential ray tracing method is needed.

In Non-Sequential Ray Tracing mode, Zemax defines the optical system as being made up of many components (or solid modules), and each component is called an object. For example, a lens is an object with not only two surfaces, but also an edge that might scatter or absorb light, and even fattened outer faces for mounting. Other common objects supported in Non-Sequential Ray Tracing include prisms, light pipes, lens arrays, light sources, detectors, TIR reflectors, partial transmissive and partial reflective components, etc.

Figure 1.1-2 shows an example of non-sequential ray tracing:

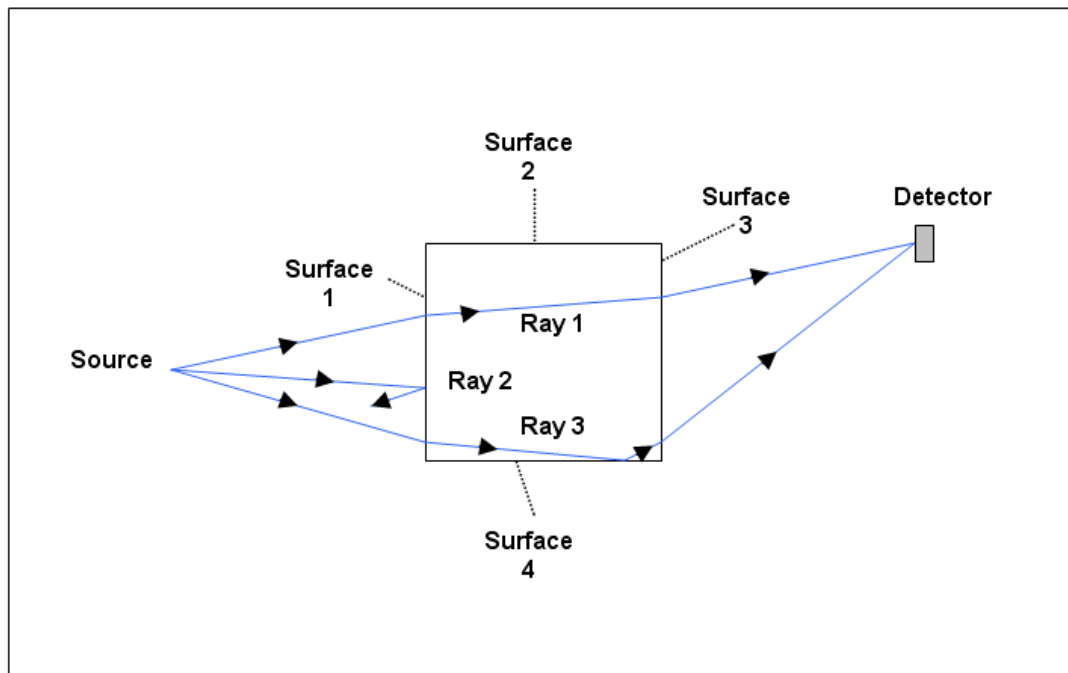


Fig. 1.1-2 Non-Sequential ray-tracing

As can be seen, Ray 1 starts from the Source, passes through Surface 1, Surface 3, and then reaches the Detector. Ray 2 starts also from the Source, however, it is reflected by surface 1, and never reaches the Detector. Ray 3 starts from the Source, passes through Surface 1, is reflected by Surface 4, passes through Surface 3, and finally reaches the Detector. This shows that in a non-sequential system, different rays may follow different paths, interact with some or all of the surfaces in different orders. Therefore, Zemax needs to trace each ray in the optical system to know its optical path, and get the overall performance of the system.

In order to set up a functional sequential optical system in Zemax, the following data needs to be provided:

- * number of surfaces
- * parameters of each surface
- * system aperture
- * working wavelength
- * field of view

In order to set up a functional non-sequential optical system in Zemax, at least the following data needs to be provided:

- * parameters of each object (including the source and the detector) and their relative position in the space
- * working wavelength

Besides the full sequential ray tracing mode and the full non-sequential ray tracing mode, Zemax also provides a mixed ray tracing mode (NSC with port). In this mode, part of the optical system is treated as sequential system, and part of the optical system is treated as non-sequential system.

The user interface of Zemax is made up of different types of windows, each of which serves a different purpose. When running Zemax, a default window called Main window will be seen, as shown in figure 1.1-3.

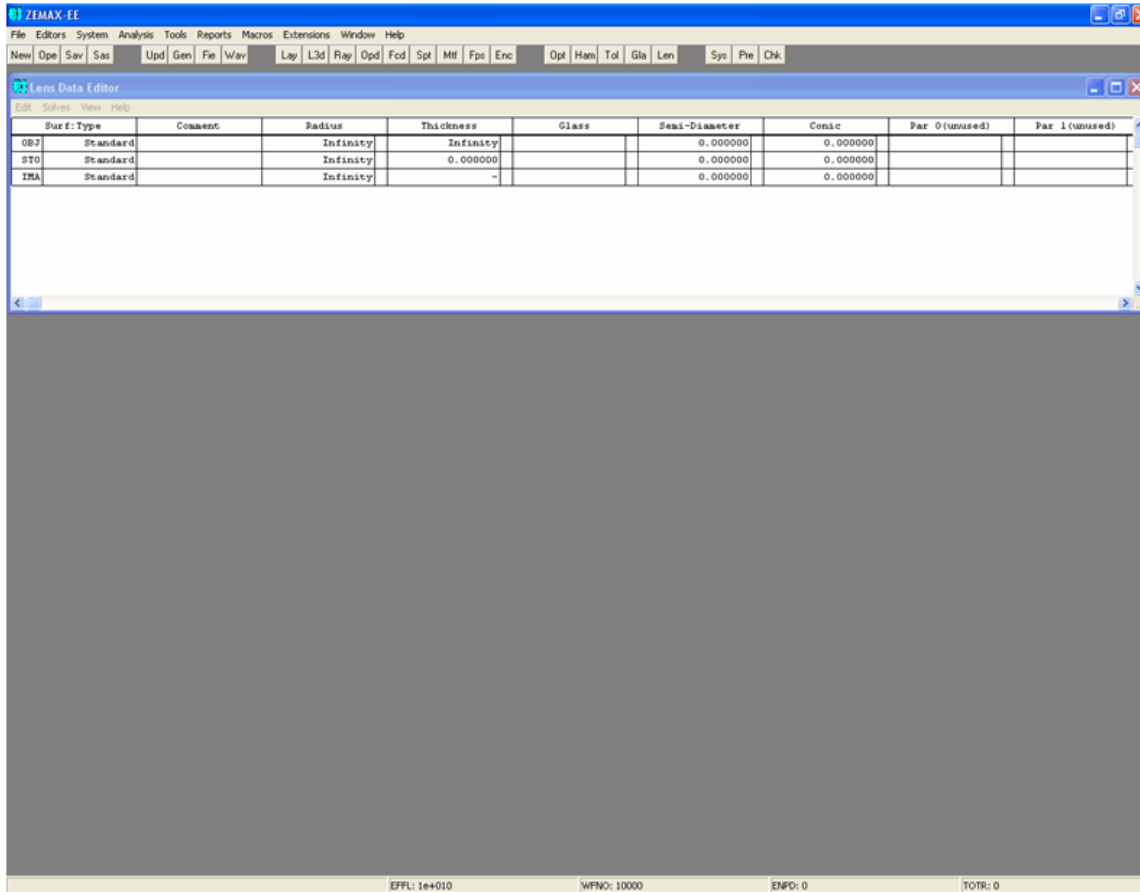


Fig. 1.1-3 Zemax Main Window

Besides the Main window, Zemax also provides other types of windows:

Editor windows: used to define and edit surface data and other data.

Graphic windows: used to display graphic data, such as layouts, ray fans, and MTF plots.

Text windows: used to display text data such as prescription data, aberration coefficients, and numerical data.

Dialogs: used to change options or data, or report error messages.

In the main window shown in figure 1.1-3, we already saw the lens data editor (LDE). Figure 1.1-4 ~ figure 1.1-11 show some editors and windows commonly used in Zemax.

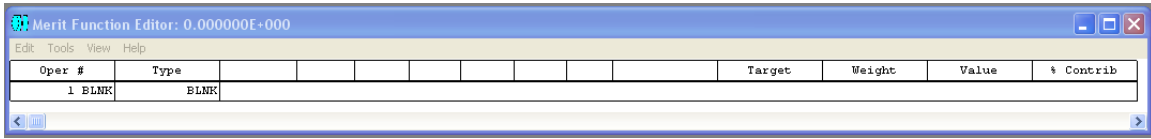


Fig. 1.1-4 Merit Function Editor

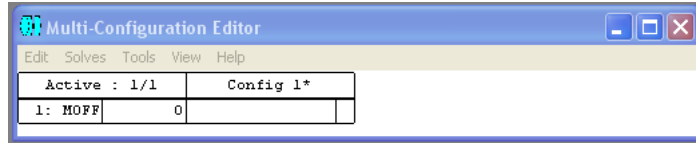


Fig. 1.1-5 Multi-Configuration Editor

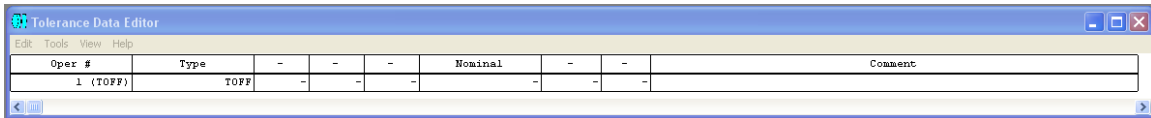


Fig. 1.1-6 Tolerance Data Editor

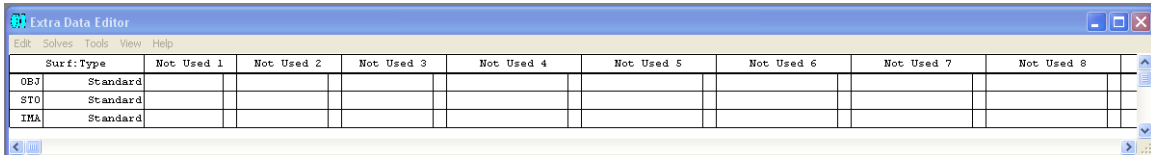


Fig. 1.1-7 Extra Data Editor

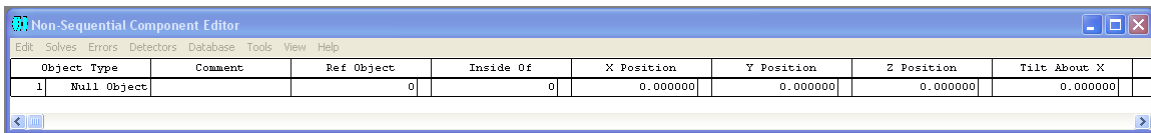


Fig. 1.1-8 Non-Sequential Component Editor

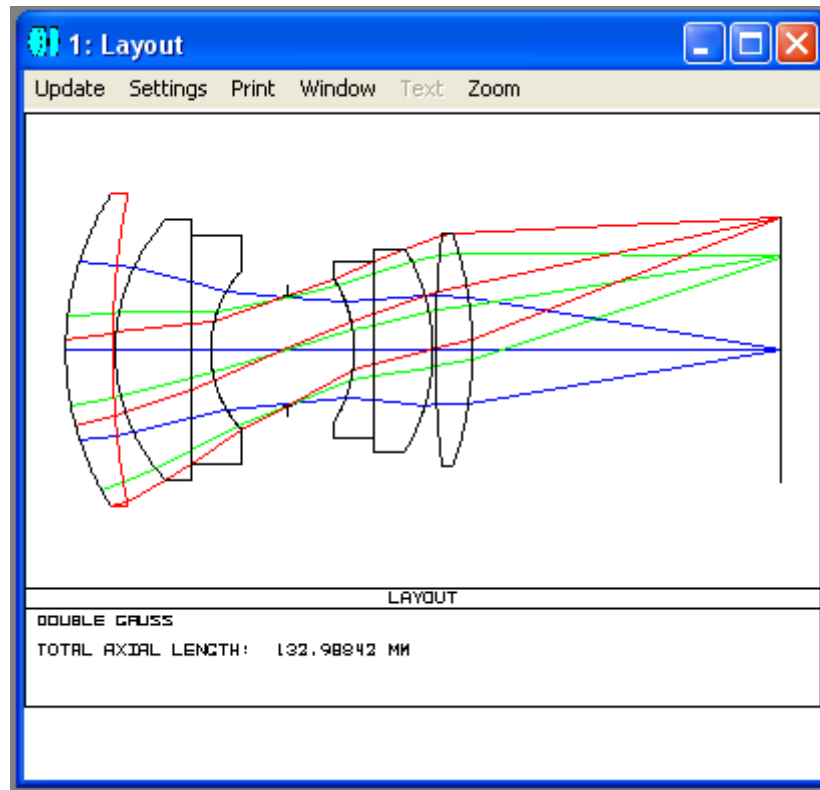


Fig. 1.1-9 Graphic Window

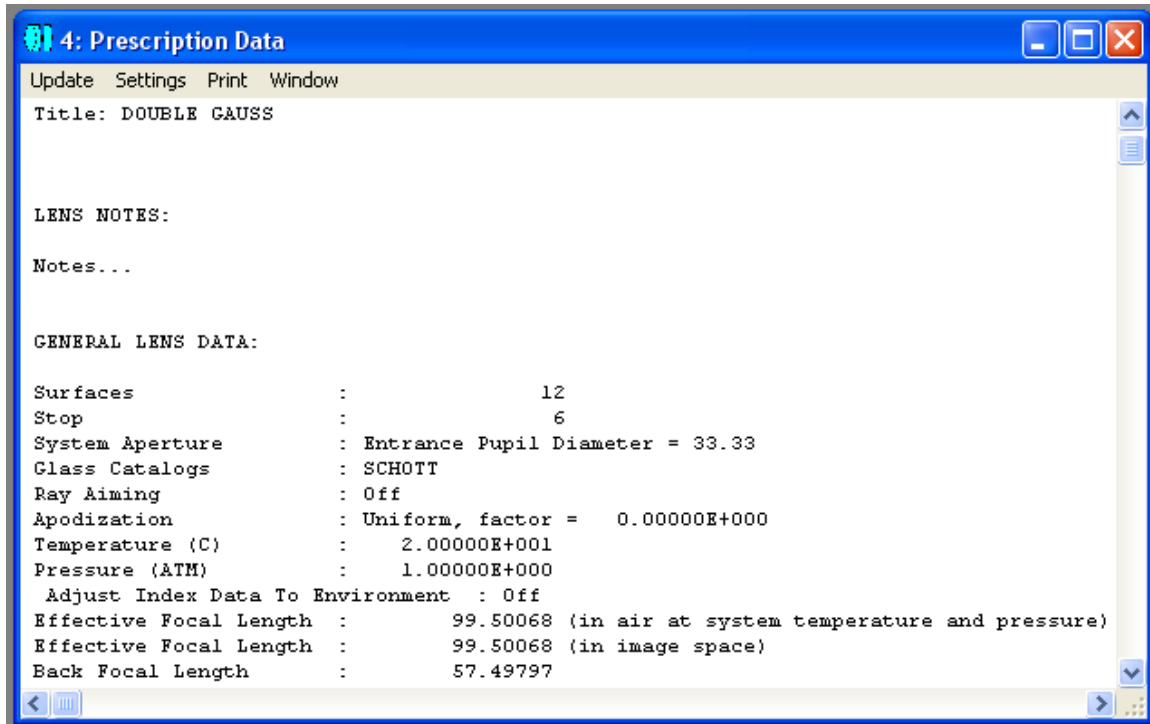


Fig. 1.1-10 Text Window

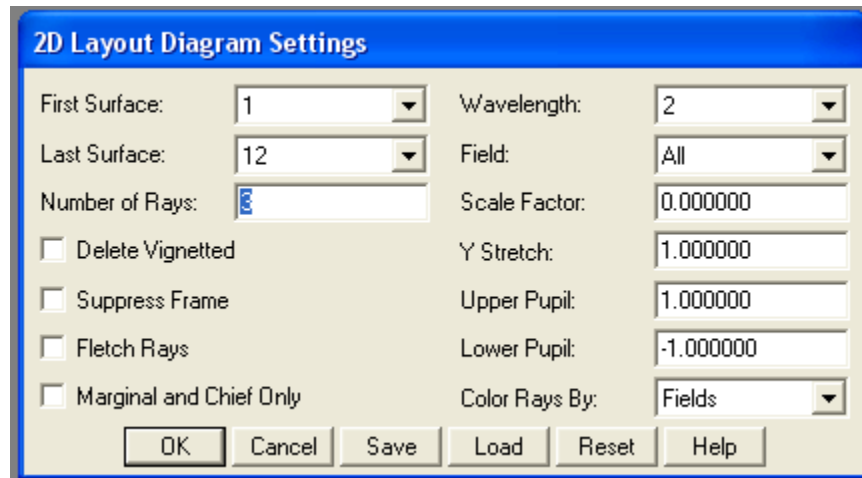


Fig. 1.1-11 Dialog Box

In general, Zemax has very powerful optical design capabilities. It can accurately calculate light path, refraction and reflection, phase and optical path difference, optical image and distortion, polarization, transmission and absorption in thin film coating, scattering, etc. However, despite its powerfulness, Zemax cannot help you to master basic principles in optical design. A good optical designer should only use Zemax as an effective tool to help his or her design, but cannot simply rely on this tool. When needed, the optical designer needs to know how to extend this tool to make it more capable.

For better knowledge of how to use Zemax, please refer to Zemax User's Manual.

For better knowledge of general optical design, please refer to the following book list:

Bass, *Handbook of Optics*, McGraw-Hill
Born & Wolf, *Principles of Optics*, Pergamon Press
Fischer & Tadic-Galeb, *Optical System Design*, McGraw-Hill
Geary, Joseph M., *Introduction to Lens Design: With Practical Zemax Examples*, Willmann-Bell
Hecht, *Optics*, Addison Wesley
Kingslake, Rudolph, *Lens Design Fundamentals*, Academic Press
Laikin, Milton, *Lens Design, Third Edition*, Marcel Dekker
Mahajan, Virendra, *Aberration Theory Made Simple*, SPIE Optical Engineering Press
O' Shea, Donald, *Elements of Modern Optical Design*, John Wiley and Sons
Rutten and van Venrooij, *Telescope Optics*, Willmann-Bell
Shannon, Robert, *The Art and Science of Optical Design*, Cambridge University Press
Smith, Gregory Hallock, *Practical Computer-Aided Lens Design*, Willmann-Bell, Inc.
Smith, Warren, *Modern Optical Engineering*, McGraw-Hill
Smith, Warren, *Modern Lens Design*, McGraw-Hill
Welford, *Aberrations of Optical Systems*, Adam Hilger Ltd.
Welford, *Useful Optics*, University of Chicago Press

1.2 Introduction to Zemax Programming Language ZPL

As mentioned in last section, although Zemax is already very powerful in optical design, there are times the designer needs to further extend its functions to fit some special design needs. Therefore, Zemax provided a tool called Zemax Programming Language (ZPL) to allow users to write their own procedures. ZPL is a macro language specifically designed for use with Zemax. It's similar to the BASIC programming language, except not all BASIC constructs and keywords are supported, and capabilities and functions unique to ray tracing have been added.

ZPL macros can be created and edited with any text editor (such as Notepad editor in Windows). The file may have any name but must end in the .ZPL extension. File name may include letter (upper case and lower case letters are treated as the same) and numbers, but may not include some special characters such as ~ () = + - * / ! > < ^ & | #. The file must be placed in the ZPL Folder, which by default is "\Macros". The default folder can be changed through Zemax main menu: File → Preferences → Directories, as shown in figure 1.2-1:

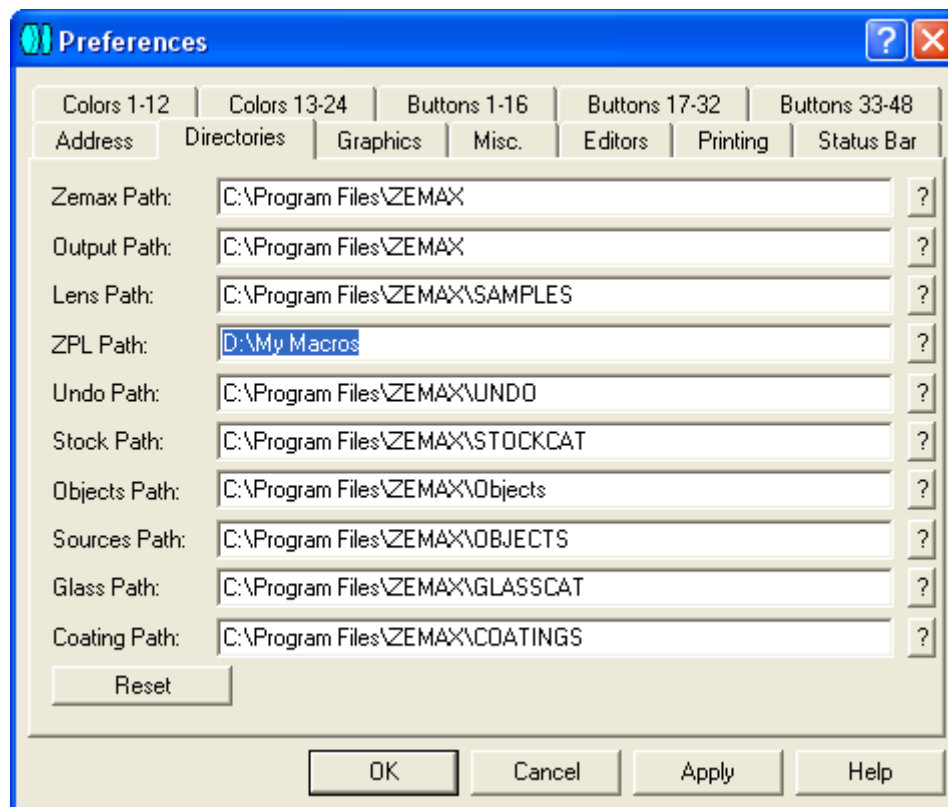


Fig. 1.2-1 ZPL path setting.

Each time after modifying the path, the macro list should be refreshed through Zemax main menu: Macros → Refresh Macro List, so the ZPL files in the updated folder can be seen properly.

Subfolders under the updated path can also be added, and ZPL files can be put into the subfolders without modifying the path settings. This is convenient and can help managing files. For example, under the path “D:\My Macros” shown in figure 1.2-1, two subfolders Project 1 and Project 2 can be created, and different ZPL files can be put in each of the subfolders, and shown in figure 1.2-2:

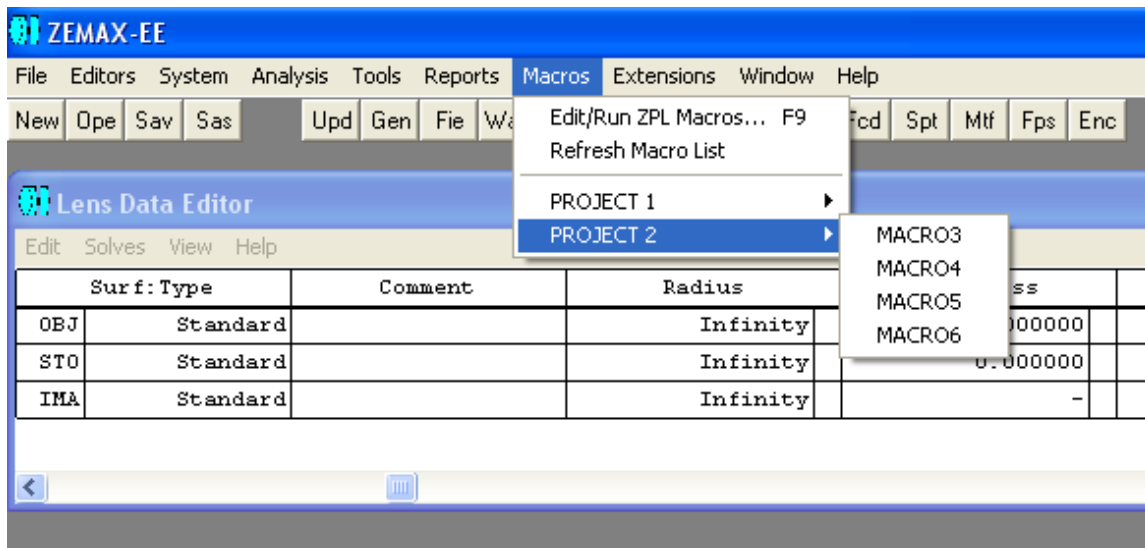


Fig. 1.2-2 Managing ZPL files through subfolders

Remember to refresh the macro list under the main menu each time after creating new subfolders and writing new ZPL programs.

ZPL programs can be run from the main menu, select Macros → Edit/ Run ZPL Macros. When doing so, ZPL control dialog box will pop up in the main window, as shown in figure 1.2-3:

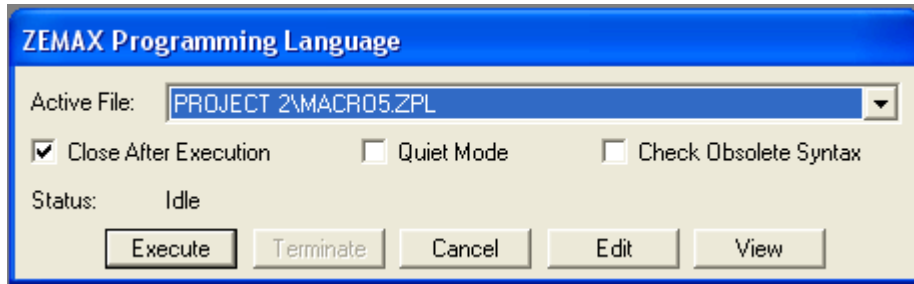


Fig. 1.2-3 ZPL control dialog box

The following options are in the ZPL control dialog box:

Active File: A drop-down list of macros available. The target macro can be chosen here.

Close After Execution: If checked, the ZPL control dialog box will automatically close after the macro execution.

Quiet Mode: If checked, the default output text window will not be shown. This is useful for graphics macros that do not generate useful text.

Check Obsolete Syntax: If checked, Zemax will test the macro for use of obsolete syntax.

Status: During execution of the macro, Zemax will use this area to print a status message stating the line number of the macro being executed. The status message is updated every quarter second.

Terminate: The terminate button will stop execution of the macro currently running.

Cancel: The cancel button terminates the current macro if one is running. If no macro is running, cancel closes the ZPL control dialog box.

Edit: The edit button invokes the Windows NOTEPAD editor. The editor can be used to modify or rename the macro.

View: The view button will display the contents of the macro file in a text window which can be scrolled or printed. No editing is allowed in the view window.

To run a macro, simply select the macro from the "Active File" list, and then click on Execute

Besides using ZPL control dialog box to run ZPL programs, one can also directly click the macro name in the "Macros" menu under the main menu to run it, as shown in figure 1.2-4:

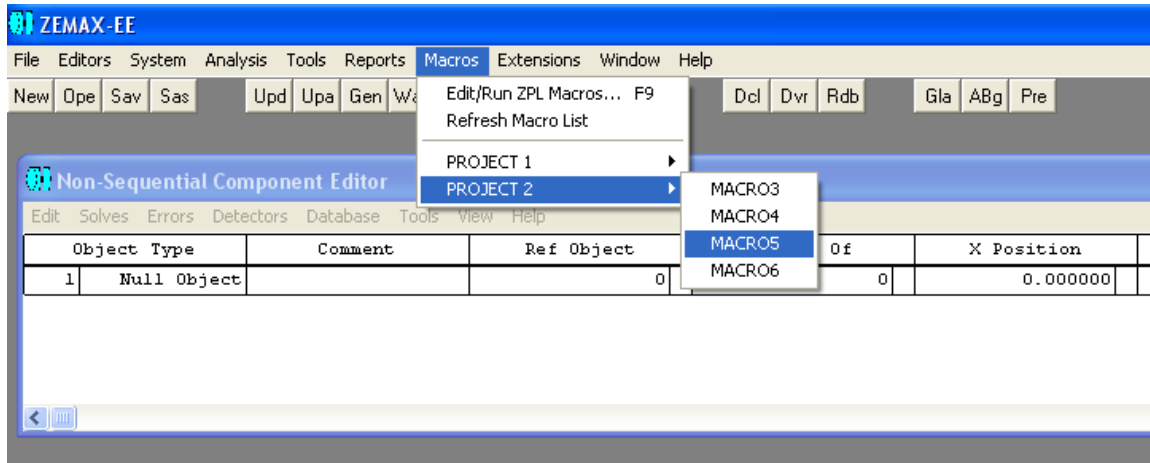


Fig. 1.2-4 Run ZPL program directly

Zemax also provides some shortcut buttons to allow convenient access to frequently used macros. By assigning macros to some buttons, one can directly run the macro by simply pressing the shortcut button. Macros can be assigned from the main menu, select File → Preferences, and in the Preferences dialog, a macro can be connected to a button, as shown in figure 1.2-5:

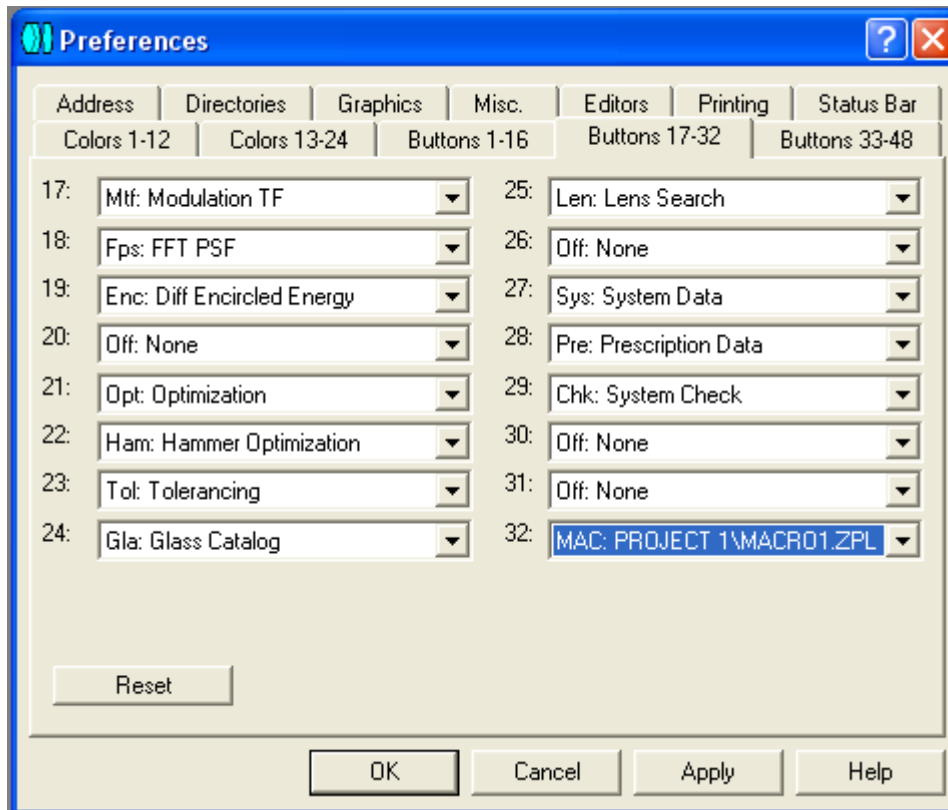


Fig. 1.2-5 Shortcut button setting

Chapter 2

Basics of Zemax Programming Language

2.1 Basic Structure

First, let's take a look at a basic ZPL program, as shown in example 2.1-1:

Example 2.1-1: basic structure of ZPL program

```
1 ! ex20101
2 ! This program introduces the basic structure of ZPL
3
4 ! The following 3 lines show 3 different ways of Comment
5 REM This is the first way
6 ! This is the second way
7 # This is the third way
8
9 ! The following 3 lines show some Assignments
10 x = 3
11 y = SINE(x) # built in function
12 newString$ = "This is a string!" # string assignment
13
14 ! The following line shows the PRINT Keyword
15 PRINT "Hello Optics!"
16
```

From the example we can see that ZPL program is a text file made up of a series of command lines. The content of the command line can be comments, assignments or keywords. We added a line mark in front of each line, but it's just for the convenience of explanation. The line marks don't exist in actual ZPL programs.

There are 3 different ways to add comments in ZPL, as shown in lines 5, 6 and 7, respectively. The first way starts with key word REM, indicating this line is a comment line. The second way starts with symbol "!", also indicates this line is a comment line. The third way is to add symbol "#" at any place in a line, indicating that any content following this symbol in the same line is comment, and won't be executed. Comments make programs easier to understand and modify, and have no effect on performance.

The basic assignment format in ZPL is:

variable = (expression)

In the assignment, variable should start with a letter, and can be any combination of letters and numbers including “_”, but cannot include special characters such as “~ () = + - * / ! > < ^ & | #” and space. Upper case letters and lower case letters are treated as the same. The total length of the variable should not exceed 28 characters. Some examples of valid variables are x, y1, variable_z, myVariable, etc. However, variable shouldn't use ZPL reserved keywords and function names. As a good practice, the name of a variable should be simple and easy to understand, especially when many people work on the same program.

There are 3 types of variables in ZPL: numeric variables, array variables and string variables. We will discuss those different types of variables in details in next blog.

In an assignment, the (expression) may consist of a constant value, other variable name containing some preassigned value, or a complex mathematical expression involving functions, constants, and variables. When an assignment is executed, the expression on the right side of the equal sign is evaluated, and the result is assigned to the variable designated on the left.

The keywords in ZPL are used to fulfill some special tasks. For example, the PRINT keyword in the program is used to display result on the screen.

2.2 Variable and Constant

In ZPL, constant values are some pre-determined values or strings, such as 0, -5, 3.1416, “Here is my string”, etc. Variables are divided into numeric variables, array variables, and string variables. They are used to store different types of data.

1. Numeric variables

Numeric variables are used to store numeric values whose exact value may not be known when the program is written, but is defined when the program is run. If not specifically mentioned, the default memory space allocated to a numeric variable by Zemax is 64 bit, and the values are stored in the format of double precision numbers. ZPL also supports 32 bit (including sign bit) integer variables. Unlike many other high level programming languages, numeric variables don't need to be declared in ZPL.

2. Calculation of numeric variables

Basic arithmetic operations can be done directly on variables in ZPL, including addition (+), subtraction (-), multiplication (*), and division (/). More complicated operations such as power and square root can be done through different numeric functions defined internally by ZPL. We will discuss those functions later.

3. Array variables

Array variables are used to store single- or multi-dimensioned arrays numeric values. Unlike simple numeric variables we discussed before, array variables must be declared prior to their use. The declaration syntax is:

```
DECLARE name, type, num_dimensions, dimension1 [, dimension 2 [, dimension 3 [, dimension 4] etc...]
```

The name may be any legal variable name as described in the previous section. The type must be either DOUBLE or INTEGER, indicating the type of array variable. The integer value num_dimensions defines the number of dimensions of the array (not the size), and must be between 1 and 4, inclusive. The integers dimension1, dimension2, etc., define the size of the array in that dimension. Note that array variables start at index 1, and thus an array of size 10 has valid indices from 1 to 10. An exception is that ZPL defined 4 one-dimension array variables VEC1 ~ VEC4 with indices from 0.

Array variables may be defined anywhere inside the program, not necessarily at the beginning.

To release the memory associated with an array variable, use the RELEASE keyword. The syntax is

RELEASE name

The RELEASE keyword is optional, as the memory associated with the declared variable is automatically released when the program terminates. However the RELEASE keyword is useful for conserving memory if large arrays are only needed during a portion of the program execution.

Array variables are assigned values using the following syntax:

name (index1, index2, ...) = value

It can assign numeric value to the array variable name(index1, index2, ...). After assignment, the values stored in the array may be retrieved with the following syntax:

newValue = name (index1, index2, ...)

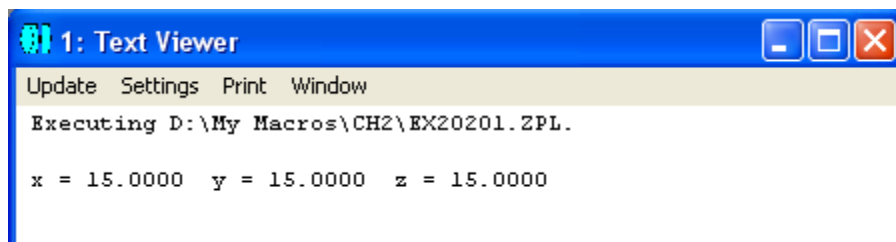
Now let's see an example of array variables.

Example 2.2-1 Assignment and operation on numeric variables:

```
1 ! ex20201
2 ! This example shows declaration and operations of Array Variables
3
4 DECLARE A, INTEGER, 2, 3, 3
5 A(1,1) = 8
6 A(1,2) = 1
7 A(1,3) = 6
8 A(2,1) = 3
9 A(2,2) = 5
10 A(2,3) = 7
11 A(3,1) = 4
12 A(3,2) = 9
13 A(3,3) = 2
14
15 x = A(1,1) + A(1,2) + A(1,3)
16 y = A(1,1) + A(2,1) + A(3,1)
17 z = A(1,1) + A(2,2) + A(3,3)
18
19 PRINT
20 PRINT "x = ", x, " y = ", y, " z = ", z
21
22 RELEASE A
```

In this example, we defined a 3 x 3 integer magic matrix. This matrix is a 2 dimensional array with the same summation of each row, each column, and two diagonals.

Save this program as EX20201.ZPL. Follow the method introduced in blog 1.2, click the program EX20201 in Macros of Main Menu, and the following result can be obtained:



```
1: Text Viewer
Update Settings Print Window
Executing D:\My Macros\CH2\EX20201.ZPL.
x = 15.0000 y = 15.0000 z = 15.0000
```

Fig. 2.2-1 Execution result of program ex20201.ZPL

For user's convenience, ZPL defined 4 one-dimensional array (vector) VEC1, VEC2, VEC3 and VEC4 to store double precision floating numbers. The default length of each array is 1000. Those 4 arrays can be directly used without additional definition. It needs to be pointed out that the index of those 4 arrays can start from 0, such as VEC1(0), so the default length 1000 can have 1001 array elements, i.e. VEC1(0) ~ VEC1(1000). If the length of the array needs to be modified, one can use keyword SETVECSIZE to do it. We will discuss more on this later on.

4. Numeric logical operators

Logical operators are used to construct complex commands which ultimately evaluate to one or zero. Most logical operations take the form (left_expression) (operator) (right_expression), similar to mathematical expressions such as $1 + 2$. The exception is the not operator "!" which takes only a single argument, of the form !(right_expression).

The following table lists numeric logical operators supported by ZPL:

Table 2.2-1 Numeric logical operators supported by ZPL

Logical	Description
&	And, returns 1 only if both expressions are non-zero.
	Or, returns 1 if at least one expression is non-zero.
^	Xor, returns 1 if only one expression is non-zero.
!	Not, returns 0 if (right_expression) is non-zero, else returns 1.
= =	Equality, returns 1 if expressions are equal.
>	Greater than, returns 1 if left_expression is greater than right_expression.
<	Less than, returns 1 if left_expression is less than right_expression.
>=	Greater than or equal to, returns 1 if left_expression is greater than or equal to right_expression.
<=	Less than or equal to, returns 1 if left_expression is less than or equal to right_expression.
!=	Inequality, returns 1 if expressions are unequal.

5. String variables and operations

ZPL supports string variables and basic string operations. Like numerical variables, string variables don't need to be declared. A string variable ends with symbol "\$". The way to assign a value to a string variable is:

```
myString$ = "Here is my string"
```

where *myString\$* is a string variable, and the content between "" (exclusive) is the string constant.

ZPL also defines many string related function. We will discuss them later.

Different strings (including string constants and string variables) can be joined by operator +, as shown below:

```
total$ = "A$ is " + A$ + " and B$ is " + B$
```

The content of a string can be displayed in a text window by using keyword PRINT. Note that PRINT only supports single string variable, and thus neither string operation nor string function can be allowed with PRINT. If one wants to display different strings in the same line, he can use "," to do it, as shown below:

```
PRINT A$, B$, C$
```

Also, if the last character in a PRINT command is ",", no new line will be started after the finish of current command, and the next print line starts to display at the current cursor location.

PRINT is also often used with another keyword REWIND. The function of REWIND is to erase the last line of message displayed by PRINT command, and change the cursor to the end of last line. In this way new messages can be displayed by overwriting old ones. This is often used when counters are needed.

6. String logical operators

String logical operators are very similar to the numeric logical operators. The major difference is that the expressions being compared are strings rather than numbers. The supported string logical operators are defined in the following table.

Table 2.2-2 String logical operators supported by ZPL

Logical	Description
\$=	Equality, returns 1 if left_string and right_string are identical.
\$>	Greater than, returns 1 if left_string is greater than right_string.
\$<	Less than, returns 1 if left_string is less than right_string.
\$>=	Greater than or equal to, returns 1 if left_string is greater than or identical to right_string.
\$<=	Less than or equal to, returns 1 if left_string is less than or identical to right_string.
\$!=	Inequality, returns 1 if left_string and right_string are not identical.

2.3 Function

A lot of numerical functions are defined in ZPL, and they can be used to calculate various numeric values. Those functions may require no arguments, one argument, or multiple arguments. In all cases, a pair of round brackets () are needed to follow the function name. If arguments are needed, they should be put in the brackets. All functions return a single value.

Example 2.3-1 shows the usage of sinusoidal function SINE(x)

```
1 ! ex20301
2 ! This program shows how to use ZPL functions
3
4 pi = 3.1416 # define a constant
5
6 x = 45 # angle in degree
7
8 y = SINE(x*pi/180)
9
10 PRINT "SINE(",x," degree) = ", y
```

Many string functions are also defined in ZPL, and their return values are strings. We will discuss them in details later.

2.4 Keywords

Keywords is an important part of ZPL. They are used to control instruction flow, output result, and do other important tasks such as modifying lens parameters and ray tracings. The words DECLARE, RELEASE and PRINT we've seen are some common keywords of ZPL.

The general syntax for a keyword is

KEYWORD argument1, argument2, argument3...

Some keywords have no arguments, others have many. Arguments may be either numeric expressions or string constants or string variables. Some keywords accept a mixture of numeric and string arguments.

We will discuss ZPL keywords in details later.

As a reference, here are the list of keywords used in ZPL:

APMN, APMX, APTP, APXD, APYD	DELETEFILE
ATYP, AVAL	DELETEMCO
BEEP	DELETEMFO
CALLMACRO	DELETEOBJECT
CALLSETDBL	DELETETOL
CALLSETSTR	EDVA
COAT	END
CLOSE	EXPORTBMP
CLOSEWINDOW	EXPORTCAD
COLOR	EXPORTJPG
COMMAND	EXPORTWMF
COMMENT	FINDFILE
CONI	FLDX, FLDY, FWGT, FVDX, FVDY, FVCX, FVCY,
CONVERTFILEFORMAT	FVAN
COPYFILE	FOR, NEXT
CURV	FORMAT
DECLARE	FTYP
DEFAULTMERIT	GCRS
DELETE	GDATE
DELETECONFIG	GETEXTRADATA

GETGLASSDATA	MAKEFOLDER
GETLSF	MODIFYSETTINGS
GETMTF	NEXT
GETPSF	NSLT
GETSYSTEMDATA	NSTR
GETTEXTFILE	NUMFIELD
GETVARDATA	NUMWAVE
GETZERNIKE	OPEN
GLAS	OPENANALYSISWINDOW
GLASSTEMPLATE	OPTIMIZE
GLENSNAME	OPTRETURN
GLOBALTOLOCAL	OUTPUT
GOSUB, SUB, RETURN, and END	PARAM
GOTO	PARAXIAL
GRAPHICS	PAUSE
GTEXT	PIXEL
GTEXTCENT	PLOT
GTITLE	PLOT2D
HAMMER	POLDEFINE
IF-THEN-ELSE-ENDIF	POLTRACE
IMA	POP
IMAGECOMBINE	PRINT
IMAGEEXTRACT	PRINTFILE
IMASHOW	PRINTWINDOW
IMASUM	PWAV
IMPORTEXTRADATA	QUICKFOCUS
INPUT	RADI
INSERT	RANDOMIZE
INSERTCONFIG	RAYTRACE
INSERTMCO	RAYTRACEX
INSERTMFO	READ
INSERTOBJECT	READNEXT
INSERTTOL	READSKIP
LABEL	READSTRING
LINE	RELEASE
LOADARCHIVE	RELOADOBJECTS
LOADCATALOG	REM, !, #
LOADDETECTOR	REMOVEVARIABLES
LOADLENS	RENAMEFILE
LOADMERIT	RETURN
LOADTOLERANCE	REWIND
LOCALTOGLOBAL	SAVEARCHIVE
LOCKWINDOW	SAVEDETECTOR
MAKEFACETLIST	SAVELENS

SAVEMERIT	STOPSURF
SAVETOLERANCE	SUB
SAVEWINDOW	SURFTYPE
SCATTER	TELECENTRIC
SDIA	TESTPLATEFIT
SETAIM	THIC
SETAIMDATA	TIMER
SETAPODIZATION	TOLERANCE
SETCONFIG	UNLOCKWINDOW
SETDETECTOR	UPDATE
SETMCOPERAND	VEC1, VEC2, VEC3, VEC4
SETNSCPARAMETER	WAVL, WWGT
SETNSCPOSITION	XDIFFIA
SETNSCPROPERTY	ZBF2MAT
SETOPERAND	ZBFCLR
SETSTDD	ZBFMULT
SETSURFACEPROPERTY, SURP	ZBFPROPERTIES
SETSYSTEMPROPERTY, SYSP	ZBFREAD
SETTEXTSIZE	ZBFRESAMPLE
SETTITLE	ZBFSHOW
SETTOL	ZBFSUM
SETUNITS	ZBFTILT
SETVAR	ZBFWRITE
SETVECSIZE	ZRD2MAT
SETVIG	ZRDAPPEND
SHOWBITMAP	ZRDFILTER
SHOWFILE	ZRDPLAYBACK
SOLVEBEFORESTOP	ZRDSAVERAYS
SOLVERRETURN	ZRDSUM
SOLVETYPE	

2.5 Flow Control

Flow control is a key part of computer programming. ZPL provided the following keywords for flow control:

FOR-NEXT
IF-THEN-ELSE-ENDIF
LABEL
GOTO
PAUSE
GOSUB-SUB-RETURN-END

We will discuss FOR-NEXT, IF-THEN-ELSE-ENDIF, LABEL, GOTO and PAUSE in this section. Keywords GOSUB-SUB-RETURN-END will be discussed in the next section.

FOR-NEXT are always used together to define a program block that needs to be run a specific number of times. The syntax is:

FOR variable, start_value, stop_value, increment
(commands)
NEXT

The keyword FOR marks the beginning of a group of commands to be executed a multiple number of times. FOR requires a variable to be specified which acts as a counter (it need not be an integer), a starting value for the counter, a stop value, and an increment. The increment value should always be an integer. The NEXT keyword marks the end of the group of commands. FOR-NEXT loops may be nested. The number of FOR and NEXT commands must be the same.

The “,” after the first variable in the FOR command line can also be replace with “=”, i.e.

FOR variable = start_value, stop_value, increment

Many programmers prefer this format for its readability.

Also, in the NEXT command line, other characters can be added after the keyword NEXT without impacting the execution of the program. Many programmers like to add corresponding loop variable after NEXT to make program more readable, especially when multiple loops are nested. But we need to

know that any character after NEXT doesn't have actual impact on the program. Its only purpose is to improve readability of the program.

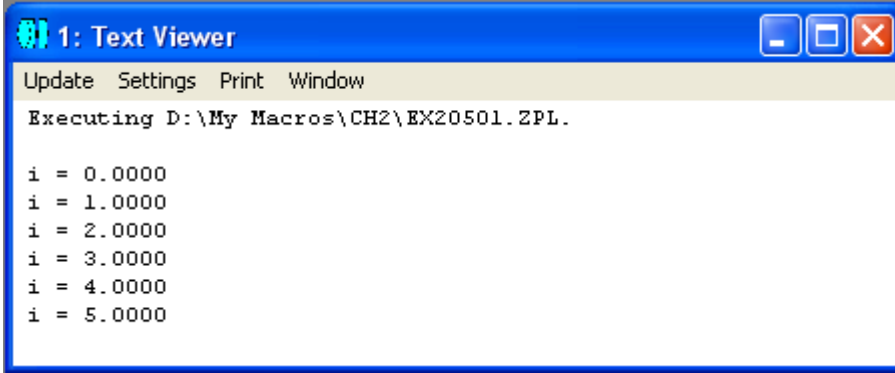
Upon reaching a FOR command, the expressions for the start, stop, and increment values are evaluated and saved. The stop and increment values are not evaluated again, even if the expressions defining the values consist of variables whose values change within the program block. Only the values valid at the beginning of the FOR loop are used.

If the start value and stop value are the same, the loop executes exactly once. If the start value is less than the stop value, then the loop continues until the counter variable is greater than the stop value. If the start value is greater than stop value, then the loop continues until the counter variable is less than the stop value.

Example 2.5-1 shows an application of FOR-NEXT loop.

```
1 ! ex20501
2 ! This program shows how to use FOR/NEXT key words
3
4 start_value = 0
5 stop_value = 5
6 increment = 1
7
8 PRINT
9 FOR i, start_value, stop_value, increment
10   start_value = 8 # try to modify loop constant
11   stop_value = 60 # try to modify loop constant
12   increment = 2 # try to modify loop constant
13   PRINT "i = ",i
14 NEXT
15
```

Please note that even we tried to assign new values to variables start_value, stop_value and increment within the loop, however, the result was not impacted, as shown in figure 2.5-1.



```
1: Text Viewer
Update Settings Print Window
Executing D:\My Macros\CH2\EX20501.ZPL.

i = 0.0000
i = 1.0000
i = 2.0000
i = 3.0000
i = 4.0000
i = 5.0000
```

Fig. 2.5-1 Result of program ex20501.ZPL

IF-THEN-ELSE-ENDIF provides conditional macro execution and branching. The syntax is:

```
IF (expression)
  (commands)
ELSE
  (commands)
ENDIF
```

or

```
IF (expression) THEN (command)
```

The value of expression is considered false if it is zero, otherwise it is considered true. When the expression is true, commands after IF will be executed, otherwise commands after ELSE will be executed. Please note that parenthesis () can be omitted here. In general, IF and ENDIF are always used together, and ELSE is optional. Keywords IF-ENDIF may be nested.

IF (expression) THEN (command) is a simplified format of conditional expression, and is usually used when only a single command needs to be executed. In the simplified format, ENDIF is not needed, and ELSE is not supported.

Example 2.5-2 shows an example of using IF-THEN-ELSE-ENDIF conditional expression. A random number generation function RAND(x) is used in the program to generate a random floating number between 0 and x.


```
1 ! ex20502
2 ! This program shows how to use IF-ELSE-ENDIF key words
3
4 theta0 = 45
5 theta = RAND(90)
6
7 PRINT
8 IF (theta > theta0)
9     PRINT theta, " is larger than 45 degree"
10 ELSE
11     IF (theta == theta0) THEN PRINT "theta equals to 45 degree"
12     IF (theta < theta0) THEN PRINT theta, " is smaller than 45 degree"
13 ENDIF
14
```

In some special cases, the program needs to jump to another place to continue to execute, so keyword GOTO is needed. Keyword GOTO is always used with another keyword LABEL. The syntax is:

LABEL label_number

...

GOTO label-number

or

LABEL text_label

...

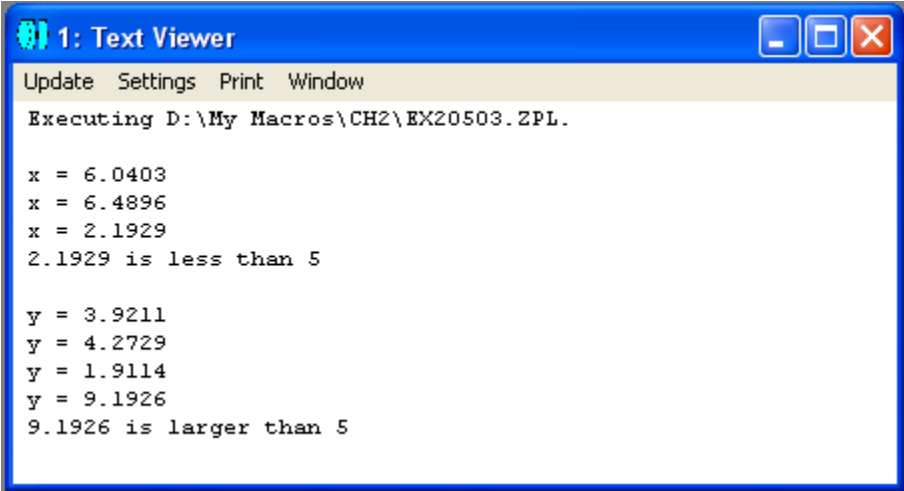
GOTO text_label

Keyword LABEL can be followed with any number or string (here we can think of numbers as a special string), and can be put at the front of any line in the program. When GOTO command is executed, the program will jump to corresponding LABEL line, and continue to execute the commands after that.

Example 2.5-3 shows an application of GOTO command:

```
1 ! ex20503
2 ! This program shows how to use GOTO/LABEL key words
3
4 PRINT
5 LABEL 01.23
6 x = RAND(10)
7 PRINT "x = ", x
8 IF x >= 5 THEN GOTO 01.23
9 PRINT x, " is less than 5"
10 PRINT
11
12 LABEL anotherLabel
13 y = RAND(10)
14 PRINT "y = ", y
15 IF y <= 5 THEN GOTO anotherLabel
16 PRINT y, " is larger than 5"
```

The result is:



```
1: Text Viewer
Update Settings Print Window
Executing D:\My Macros\CH2\EX20503.ZPL.

x = 6.0403
x = 6.4896
x = 2.1929
2.1929 is less than 5

y = 3.9211
y = 4.2729
y = 1.9114
y = 9.1926
9.1926 is larger than 5
```

Fig. 2.5-2 Result of program ex20503.ZPL

It needs to be pointed out that in structured programming, GOTO command is in general not recommended, because it can often cause unclearness of the program structure and is hard to debug. Unfortunately, ZPL doesn't support conditional loop command such as WHILE in C language, so we can only use GOTO-LABEL command to do the job, and extra attention needs to be paid.

ZPL also provides another keyword PAUSE for program control. It is used to pause the execution of the current program, display information in the message window, and wait for user response. When the user hits the OK button, the current program will continue to run from where it pauses. The syntax is:

PAUSE

or

PAUSE message

Where message can be any number or string.

Example 2.5-4 shows an application of PAUSE command:

```
1 ! ex20504
2 ! This program shows how to use PAUSE key word
3
4 PRINT
5 FOR i, 1, 5, 1
6   PRINT "i = ", i
7   IF i == 5
8     PRINT "Waiting for approval ..."
9     PAUSE "We just finished i = 5, please responde..."
10  ENDIF
11 NEXT
12 PRINT
13 PRINT "APPROVED!"
14
```

When the program runs to `i == 5`, a message window will be displayed, and the program will be paused, as shown in figure 2.5-3:

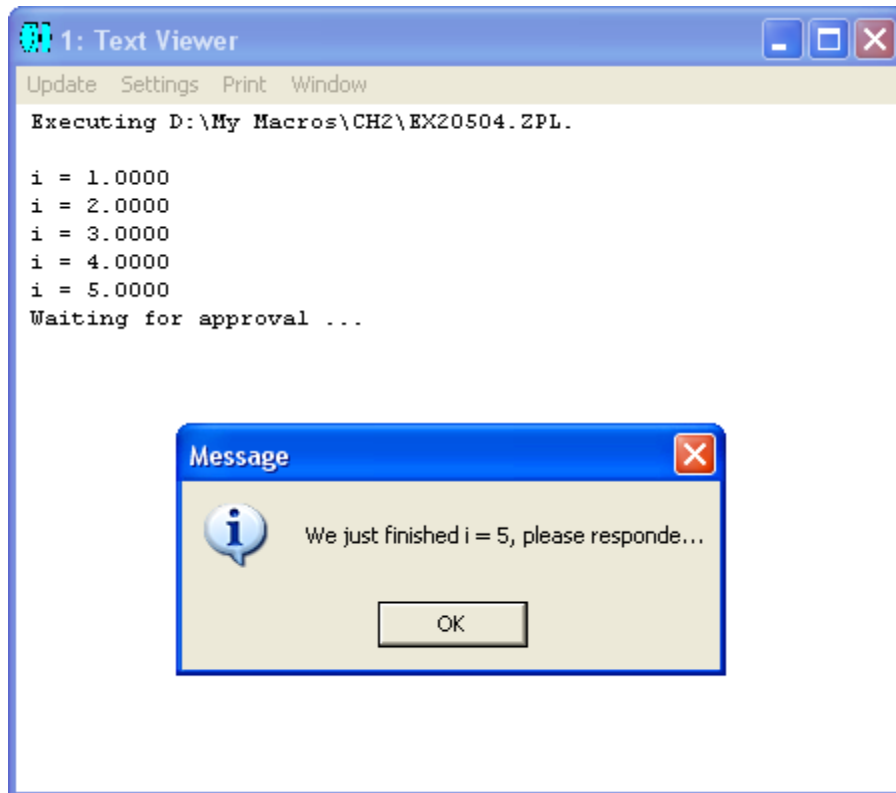


Fig. 2.5-3 Result of program ex20504.ZPL at pause.

After hit OK button, the program continue to execute, and the final result is shown below:

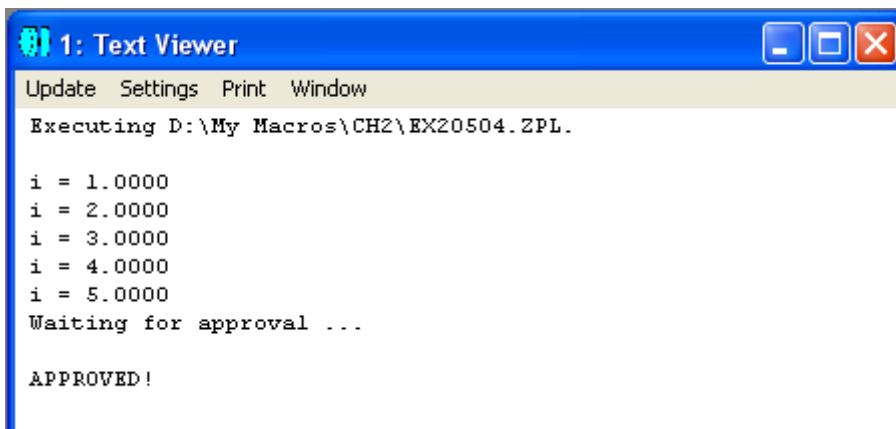


Fig. 2.5-4 Final result of program ex20504.ZPL

2.6 Sub-Function

Sub-function or sub-program can be defined in a ZPL program, and can be called in the main program or other sub-programs. The way to define sub-program is:

```
SUB sub_name  
(commands)  
RETURN
```

The sub-program starts with keyword SUB, followed by the name of the sub-program sub_name. The commands part is the main body of the sub-program, which is put together to finish a special task. The sub-program must end with RETURN, but other RETURN commands can also be used at other places within the main body of the sub-program. Sometimes for the sake of readability, the name of the sub-program can be added after RETURN in the same line, however, please remember that the name has no actual impact on the program so it can be anything, so special attention needs to be paid in order not to make any confusion.

As a rule, ZPL requires that if sub-program is used in a program, at least one END command is needed to mark the end of the main program, and the main program needs to be put in front of the sub-program.

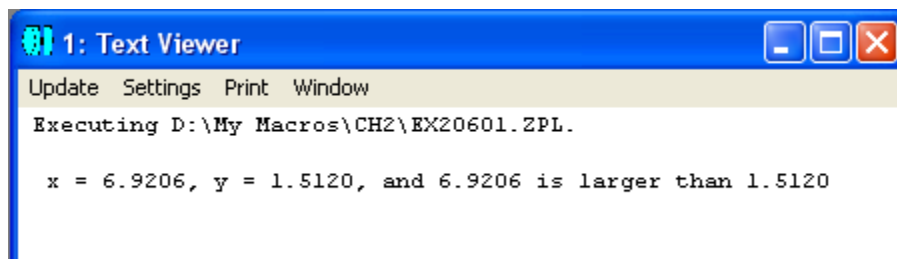
It's important to remember that the variables in ZPL are global variable. Therefore, if a variable is modified in a sub-program, the value of the same variable at other places in the whole ZPL program will also be modified.

Example 2.6-1 shows an application of sub-program:

```
1 ! ex20601
2 ! This program shows how to use GOSUB-SUB-RETURN-END keywords
3
4 ! This is the main program
5 x = 3
6 y = 2
7 GOSUB findMax
8 PRINT
9 PRINT " x = ", x, ", y = ", y, ", and ", max, " is larger than ", min
10 END # This is the end of the main program
11
12 ! This is the sub-routine
13 SUB findMax
14 x = RAND(10)
15 y = RAND(10)
16 IF x > y
17   max = x
18   min = y
19   RETURN # This is another RETURN
20 ENDIF
21 max = y
22 min = x
23 RETURN # This is the end of the sub-routine
```

Please notice that we first assigned values to variables x and y in the main program, but in the sub-program, new x and y values generated by random function replaced the old values, and thus in the final result the displayed x and y values are their new values. Also, we used two RETURN commands in the sub-program. If $x > y$, the sub-program will end at the first RETURN, and go back to the main program without running the rest of the sub-program.

Figure 2.6-1 shows the result of the program:



```
1: Text Viewer
Update Settings Print Window
Executing D:\My Macros\CH2\EX20601.ZPL.

x = 6.9206, y = 1.5120, and 6.9206 is larger than 1.5120
```

Fig. 2.6-1 Result of program ex20601.ZPL

2.7 I/O and File Operation

ZPL provides a keyword INPUT to allow user type in numerical or string information when a program is running. The syntax of INPUT is:

INPUT "Prompt String", variable

INPUT variable

INPUT "Prompt String", string_variable\$

INPUT string_variable\$

Example 2.7-1 shows an application of keyword INPUT:

```
1 ! ex20701
2 ! This program shows how to use INPUT keyword
3
4 INPUT "Enter value for x:", x
5 PRINT "x = ", x
6
7 INPUT "Enter string:", string$
8 PRINT "string is '", string$, "'"
```

When the program runs to each INPUT command, a dialog window will pop up, showing corresponding messages and waiting for the input, as shown in figure 2.7-1:

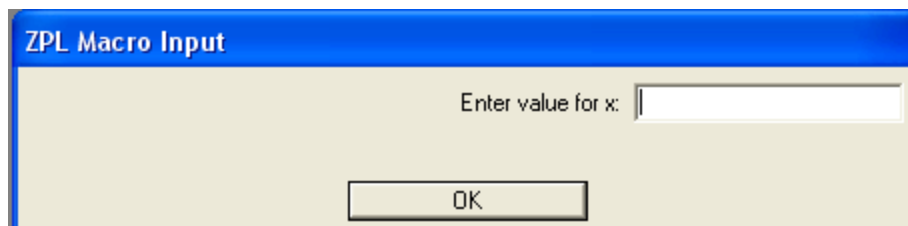


Fig. 2.7-1 The first pop-up window of running program ex20701.ZPL.

When the user type in the numerical or string information and press OK button to confirm, the value typed in will be stored in the corresponding variable, and the program continues to run.

We know in previous sections that ZPL can output numerical or string information to the message window using keyword PRINT. In fact, PRINT not only can output messages to the display, but can also output messages to files. This is controlled by keyword OUTPUT. The syntax of OUTPUT is:

```
OUTPUT SCREEN
OUTPUT filename$
OUTPUT filename$, APPEND
```

If OUTPUT is followed by SCREEN, then the following PRINT command will display the result on the screen. If OUTPUT is followed by filename\$, then the following PRINT command will output the result into the corresponding file. Further, if APPEND is used with OUTPUT, then the result will be added at the end of the corresponding file without overwriting the existing content of the file.

Example 2.7-2 shows an application of keyword OUTPUT:

```
1 ! ex20702
2 ! This program shows how to use OUTPUT keyword
3
4 filename$ = "D:\My Macros\ch2\output_files\test1.txt"
5
6 OUTPUT filename$
7 PRINT "You will see this line in the file, but not on the screen."
8
9 OUTPUT SCREEN
10 PRINT
11 PRINT "You will see this line on the screen, but not in the file."
12
13 OUTPUT filename$, APPEND
14 PRINT "You will see this line in the file as a new line."
15
16 OUTPUT "D:\My Macros\ch2\output_files\test2.txt"
17 PRINT "You will see this line in a different file."
```

In this example, we assume the folder "D:\My Macros\ch2\output_files\" already exists, otherwise ZEMAX will report error message. Figure 2.7-2 is the result of the program seen on the screen and in different files:

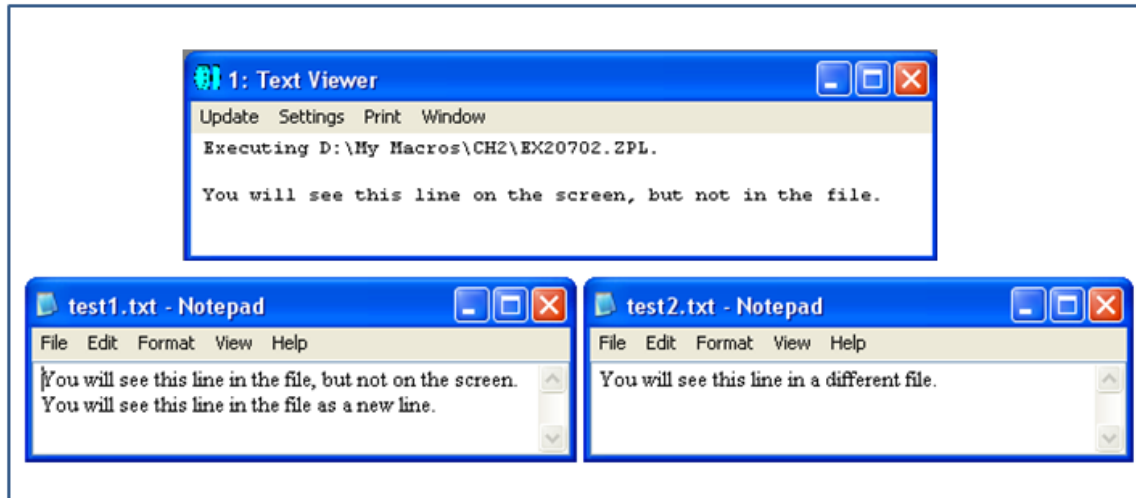


Fig. 2.7-2 Result of program ex20702.ZPL

Keyword PRINT is often used with another keyword FORMAT. FORMAT can be used to control the output numerical precision format of the message window or the file. The syntax is:

FORMAT m.n

FORMAT m.n EXP

FORMAT m [INT]

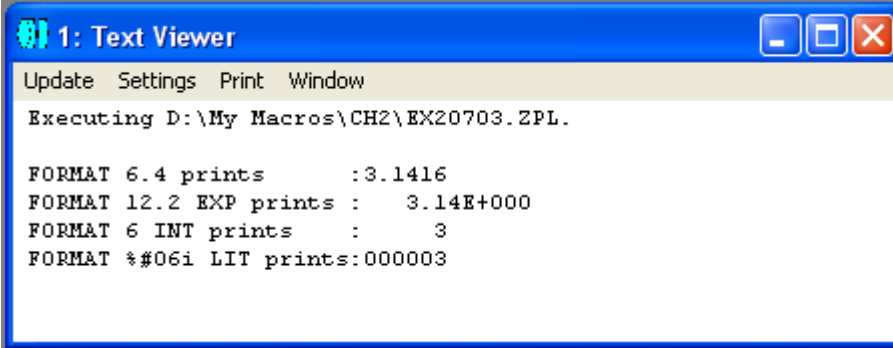
FORMAT "C_format_string" LIT

The integers m and n are separated by a decimal point. The values for m and n used must be explicit, i.e. values stored as variables cannot be used. The value m refers to the total number of characters to be printed, even though some of them may be blank. The value n refers to the number of places to display after the decimal point. The optional keyword EXP after the m.n expression indicates exponential notation should be used. The optional keyword INT indicates the value should be first converted to an integer and printed in integer format using the number of places specified by m. The optional keyword LIT (for literal) indicates the value should be printed according to the "C" language format specifier. The C format specification can be found in any programming reference for the C language.

Example 2.7-3 shows some applications of keyword FORMAT.

```
1 ! ex20703
2 ! This program shows how to use keyword FORMAT
3
4 x = 3.14159
5
6 PRINT
7 FORMAT 6.4
8 PRINT "FORMAT 6.4 prints      :", x
9
10 FORMAT 12.2 EXP
11 PRINT "FORMAT 12.2 EXP prints :", x
12
13 FORMAT 6 INT
14 PRINT "FORMAT 6 INT prints   :", x
15
16 FORMAT "%#06i" LIT
17 PRINT "FORMAT %#06i LIT prints:", x
```

Figure 2.7-3 shows the result of the program seen in the pop-up window:



```
1: Text Viewer
Update Settings Print Window
Executing D:\My Macros\CH2\EX20703.ZPL.
FORMAT 6.4 prints      :3.1416
FORMAT 12.2 EXP prints :  3.14E+000
FORMAT 6 INT prints   :    3
FORMAT %#06i LIT prints:000003
```

Fig. 2.7-3 Result of program ex20703.ZPL

Besides the way to type in information from the keyboard, ZPL also supports importing numeric or string information from a text file, with the following keywords and functions:

OPEN, READ, READNEXT, READSTRING, CLOSE, EOFF()

Keyword OPEN is used to open a text file. The syntax is:

OPEN "filename"

or

OPEN filename\$

After the file is open, the needed information can be read with keywords READ, READNEXT or READSTRING.

READ is used to read a whole line and store numeric values into variables followed by READ. The syntax is:

READ x, y, z, ...

When using READ, the number of variables listed in the read command should match the number of columns in the text file, otherwise ZEMAX will assign 0 to extra variables. Numeric data in the file should be delimited by spaces.

Besides READ, keyword READNEXT can also be used to read data. The main difference is READ will read the entire data line from the opened file, up to the newline character, while READNEXT reads only enough characters to fill the number of arguments. The syntax of READNEXT is:

READNEXT x,y,z, ...

If string information is needed to read, keyword READSTRING can be used. The syntax is:

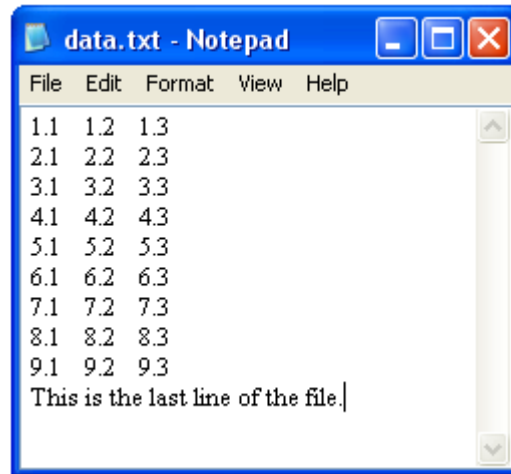
READSTRING textString\$

READSTRING reads the whole line of string information and stores it to the string variable textString\$.

Sometimes we need to know if the end of a file has been reached. ZPL provides a function EOFF() to do so. If the end of the file is reached, the function returns values 1, otherwise it returns value 0. Function EOFF() can only be used after keywords READ, READNEXT or READSTRING.

And remember, keyword CLOSE is needed to close the file after finish reading.

Now let's assume we have a text file "data.txt" with the following content:



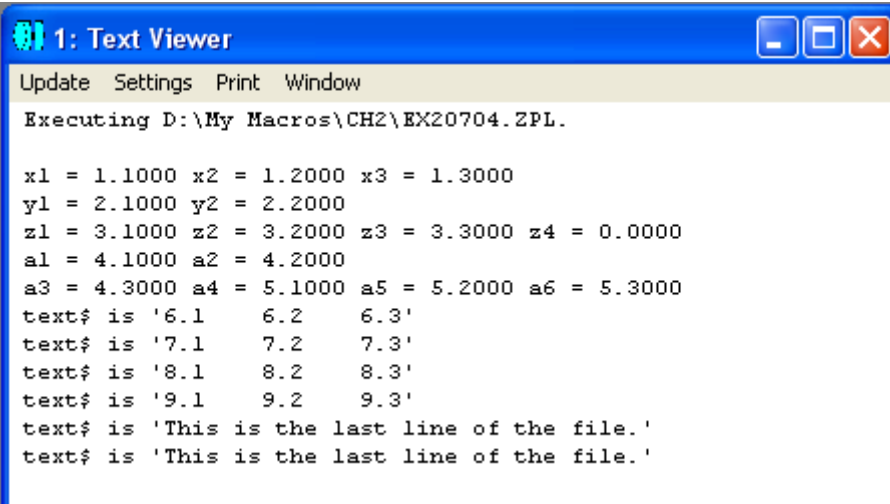
In example 2.7-4, we will use the keywords introduced above to read information from the file "data.txt", as shown below:

```

1 ! ex20704
2 ! This program shows how to use READ series keyword
3
4 filename$ = "D:\My Macros\CH2\data_files\data.txt"
5
6 OPEN filename$
7
8 PRINT
9 READ x1, x2, x3
10 PRINT "x1 = ", x1, " x2 = ", x2, " x3 = ", x3
11
12 READ y1, y2
13 PRINT "y1 = ", y1, " y2 = ", y2
14
15 READ z1, z2, z3, z4
16 PRINT "z1 = ", z1, " z2 = ", z2, " z3 = ", z3, " z4 = ", z4
17
18 READNEXT a1, a2
19 PRINT "a1 = ", a1, " a2 = ", a2
20
21 READNEXT a3, a4, a5, a6
22 PRINT "a3 = ", a3, " a4 = ", a4, " a5 = ", a5, " a6 = ", a6
23
24 LABEL 1
25 IF (EOFF() != 1)
26     READSTRING text$
27     PRINT "text$ is '", text$, "'"
28     GOTO 1
29 ENDIF
30
31 CLOSE

```

The result is shown in figure2.7-4:



```

1: Text Viewer
Update Settings Print Window
Executing D:\My Macros\CH2\EX20704.ZPL.

x1 = 1.1000 x2 = 1.2000 x3 = 1.3000
y1 = 2.1000 y2 = 2.2000
z1 = 3.1000 z2 = 3.2000 z3 = 3.3000 z4 = 0.0000
a1 = 4.1000 a2 = 4.2000
a3 = 4.3000 a4 = 5.1000 a5 = 5.2000 a6 = 5.3000
text$ is '6.1 6.2 6.3'
text$ is '7.1 7.2 7.3'
text$ is '8.1 8.2 8.3'
text$ is '9.1 9.2 9.3'
text$ is 'This is the last line of the file.'
text$ is 'This is the last line of the file.'

```

Fig. 2.7-4 Result of program ex20704.ZPL

(Note in some earlier versions of ZEMAX the last line of the text file was displayed twice on the screen. This might be a bug of ZEMZX and may be fixed in a later version.)

We will discuss more about Input/Output and file operation in later sections.

Chapter 3

ZPL commands in details

ZPL provides a lot of keywords and functions, especially those related to optical design. In Zemax User's Manual, each of the keywords and functions were discussed in alphabetic order. However, sometimes we feel it will be more convenient to classify those keywords and functions and put them in a more organized context. So in this chapter we will introduce many ZPL keywords and functions from a different angle of view, and hope this effort can help Zemax users to be more efficient in learning ZPL. Even to an experienced ZPL programmer, this can be used as a handy tool to find the right keyword or function to use when programming.

From now on, we will no longer separate keywords and functions. We call them as commands in general. The main difference is that keyword doesn't have return values, and function usually has one or more return values.

As Zemax is rapidly being updated every year, some of the older commands we discuss here may be obsolete in later versions of Zemax. In such cases, please refer to Zemax User's Manual for the details.

3.1 Numerical Operation Functions

In last chapter, we already discussed some numerical operation functions in ZPL. In fact, ZPL provides a lot of functions to perform numerical operations. Table 3.1-1 shows numerical operation functions supported by ZPL:

Table 3.1-1 ZPL Numerical Operation Functions

Function	Argument	Return Value
ABSO(x)	Numeric expression	The absolute value of the expression.
ACOS(x)	Numeric expression	Arc cosine in radians.
ASIN(x)	Numeric expression	Arc sine in radians.
ATAN(x)	Numeric expression	Arc tangent in radians.
COSI(x)	Numeric expression in radians	Cosine of the expression.
EXPE(x)	Numeric expression	e to the power of the expression.
EXPT(x)	Numeric expression	10 to the power of the expression.
GAUS(x)	Standard deviation	Returns a random value with a Gaussian distribution, zero mean, and the specified standard deviation.
INTE(x)	Numeric expression	Returns the largest integer not greater than the argument.
LOGE(x)	Positive numeric expression	Log base e of the expression.
LOGT(x)	Positive numeric expression	Log base ten of the expression.
MAGN(x,y)	x and y are any real numbers	Computes the square root of x squared plus y squared.
POWR(x,y)	x and y are any numbers	Computes the absolute value of x to the power of y.
RAND(x)	Positive numeric expression	Random floating point number uniformly distributed between 0 and the expression.
SIGN(x)	Numeric expression	Returns -1 if the argument is less than zero, 0 if the argument is zero, and +1 if the argument is positive.
SINE(x)	Numeric expression in radians	Sine of the expression.
SQRT(x)	Positive numeric expression	Square root of the expression.
TANG(x)	Numeric expression in radians	Tangent of the expression.

3.2 String Functions

We have learned that ZPL provides many functions with string as arguments or return values. Table 3.2-1 shows string functions supported by ZPL:

Table 3.2-1 ZPL String Functions

Function	Description
\$BUFFER()	Returns the current string in the lens buffer. This function is used to extract string data from various ZPL keywords and functions.
\$CALLSTR(i)	Returns the string from the CALLMACRO string buffer at index i.
\$COAT(i)	Returns the coating name for the ith surface.
\$COATINGPATH()	Returns the path name for coating files.
\$COMMENT(i)	Returns the comment string for the ith surface.
\$DATE()	Returns the current date and time string.
\$EXTENSIONPATH()	Returns the path name for Zemax extensions.
\$FILENAME()	Returns the current lens file name, without the path.
\$FILEPATH()	Returns the current lens file name, with the complete path.
\$GETSTRING(A\$, n)	Returns the nth sub-string for the string A\$ using spaces for delimiters. For example, if A\$ = "one two three", then \$GETSTRING(A\$, 2) returns "two".
\$GETSTRINGC(A\$, n)	Returns the nth sub-string for the string A\$ using commas for delimiters. For example, if A\$ = "one,two,three", then \$GETSTRING(A\$, 2) returns "two".
\$GLASS(i)	Returns the glass name of surface number i.
\$GLASSCATALOG(i)	Returns the name of the ith loaded glass catalog for the current lens. If i is less than 1, then the names of all the loaded catalogs separated by spaces are returned in a single string.
\$GLASSPATH()	Returns the path name for glass catalog files.
\$LEFTSTRING(A\$, n)	Returns the left most n characters in the string A\$. If A\$ has fewer than n characters, the remaining spaces will be padded with blanks. This allows formatting of strings with a fixed length.
\$LENSNAME()	Returns the lens title defined in the General System dialog box.

\$MACROPATH()	Returns the path name for macro files.
\$NOTE(line#)	Returns the notes information defined in the General System dialog box. Because the notes may be very long, \$NOTE returns the characters from the notes in groups called lines. A line ends when a newline (carriage return) character is found, or when the total number of consecutive characters on the line reaches 100, whichever comes first. The line# indicates which line in the notes is to be returned. \$NOTE will return a null (empty) string if there are no defined characters in the notes for the specified line.
\$OBJECTPATH()	Returns the path name for NSC object files.
\$PATHNAME()	Returns the path name only for the current lens file. This is useful for determining the folder where the lens file is stored.
\$PROGRAMPATH()	Returns the path name for program files.
\$QUOTE()	Returns the double quote character (").
\$RIGHTSTRING(A\$, n)	Returns the right most n characters in the string A\$. If A\$ has fewer than n characters, the remaining spaces will be padded with blanks. This allows formatting of strings with a fixed length.
\$COM(A\$, B\$)	If the two strings A\$ and B\$ are equal, \$COM returns 0. If A\$ is less than B\$, then \$COM returns a value less than 0; otherwise, a value greater than 0.
\$LEN(A\$)	The number of characters in the string variable A\$.
\$STR(expression)	Returns a string formatted using the format defined by the FORMAT keyword. The numeric expression may be any equation, including combinations of constants, variables, and functions. See function \$VAL(A\$) to convert strings to numbers.
\$VAL(A\$)	String value. Returns a floating point value of the string A\$.
\$TAB()	Returns the tab character (\t).
\$TEMPFILENAME()	Returns the name of a temporary file, with complete path, suitable for temporary storage of text or binary data. See keyword GETTEXTFILE.
\$TOLCOMMENT(operand)	Returns the comment for the specified tolerance operand.
\$TOLOPERAND(operand)	Returns the operand name for the specified tolerance operand.
\$UNITS()	Returns either MM, CM, IN, or M, depending upon the current lens units.

3.3 Setting and Reading Zemax System Properties

Most ZEMAX users know that before you jump into your optical design using Zemax, you need to set some basic system properties for Zemax, such as working wavelength, units, system aperture, etc. ZPL provides a keyword SETSYSTEMPROPERTY (or its short format SYSP) to set various system properties. The syntax is:

SETSYSTEMPROPERTY code, value1, value2

or

SYSP code, value1, value2

In this command, code is an integer which specifies what property is being modified, value1 and value2 are the new values for the specified property, and they may be either text in quotes, a string variable, or a numeric expression. Most codes require only one argument value1, while some other codes require both value1 and value2. The details of each code is shown in table 3.3-1.

Table 3.3-1 SYSP code

Code	Property
4	Adjust Index Data To Environment. Use 0 for off, 1 for on.
10	Aperture Type.
11	Aperture Value.
12	Apodization Type code. Use 0 for uniform, 1 for Gaussian, 2 for cosine cubed.
13	Apodization Factor.
14	Telecentric Object Space. Use 0 for off, 1 for on.
15	Iterate Solves When Updating. Use 0 for off, 1 for on.
16	Lens Title.
17	Lens Notes.
18	Afocal Image Space. Use 0 for off, 1 for on.
21	Global coordinate reference surface.
23	Glass catalog list. Use a string or string variable with the glass catalog name, such as "SCHOTT". To specify multiple catalogs use a single string or string variable containing names separated by spaces, such as "SCHOTT HOYA OHARA".
24	System Temperature in degrees Celsius.
25	System Pressure in atmospheres.

26	Reference OPD method. Use 0 for absolute, 1 for infinity, 2 for exit pupil, and 3 for absolute 2.
30	Lens Units code. Use 0 for mm, 1 for cm, 2 for inches, or 3 for Meters. Changing lens units does not scale or convert the lens data in any way, it only changes how the lens prescription data is interpreted.
31	Source Units Prefix. Use 0 for Femto, 1 for Pico, 2 for Nano, 3 for Micro, 4 for Milli, 5 for None, 6 for Kilo, 7 for Mega, 8 for Giga, and 9 for Tera.
32	Source Units. Use 0 for Watts, 1 for Lumens, and 2 for Joules.
33	Analysis Units Prefix. Use 0 for Femto, 1 for Pico, 2 for Nano, 3 for Micro, 4 for Milli, 5 for None, 6 for Kilo, 7 for Mega, 8 for Giga, and 9 for Tera.
34	Analysis Units “per” Area. Use 0 for square mm, 1 for square cm, 2 for square inches, 3 for square Meters, and 4 for square feet.
35	MTF Units code. Use 0 for cycles per millimeter, or 1 for cycles per milliradian.
40	Coating File name.
41	Scatter Profile name.
42	ABg Data File name.
43	GRADIUM Profile name.
50	NSC Maximum Intersections Per Ray.
51	NSC Maximum Segments Per Ray.
52	NSC Maximum Nested/Touching Objects.
53	NSC Minimum Relative Ray Intensity.
54	NSC Minimum Absolute Ray Intensity.
55	NSC Glue Distance In Lens Units.
56	NSC Missed Ray Draw Distance In Lens Units.
57	NSC Retrace Source Rays Upon File Open. Use 0 for no, 1 for yes.
58	NSC Maximum Source File Rays In Memory.
59	Simple Ray Splitting. Use 0 for no, 1 for yes.
60	Polarization Jx.
61	Polarization Jy.
62	Polarization X-Phase.
63	Polarization Y-Phase.
64	Convert thin film phase to ray equivalent. Use 0 for no, 1 for yes.
65	Unpolarized. Use 0 for no, 1 for yes.
66	Method. Use 0 for X-axis, 1 for Y-axis, and 2 for Z-axis.
70	Ray Aiming. Use 0 for off, 1 for on, 2 for aberrated.

71, 72, 73	Ray aiming pupil shift x, y, and z.
74	Use Ray Aiming Cache. Use 0 for no, 1 for yes.
75	Robust Ray Aiming. Use 0 for no, 1 for yes.
76	Scale Pupil Shift Factors By Field. Use 0 for no, 1 for yes.
77, 78	Ray aiming pupil compress x, y.
100	Field type code.
101	Number of fields.
102, 103	The field number is value1, value2 is the field x, y coordinate.
104	The field number is value1, value2 is the field weight.
105, 106	The field number is value1, value2 is the field vignetting decenter x, decenter y.
107, 108	The field number is value1, value2 is the field vignetting compression x, compression y.
109	The field number is value1, value2 is the field vignetting angle.
110	The field normalization method, value 1 is 0 for radial and 1 for rectangular.
200	Primary wavelength number.
201	Number of wavelengths.
202	The wavelength number is value1, value 2 is the wavelength in micrometers.
203	The wavelength number is value1, value 2 is the wavelength weight.
901	The number of CPU's to use in multi-threaded computations, such as optimization. If the passed value is zero, the number of CPU's will be set to the default value. When testing this value using the function SYPR, this returns the total number of CPU's available as reported by the operating system.

Opposite to system property setting, if we want to read system parameters, we can use function SYPR() provided by ZPL. Most system properties set by keyword SETSYSTEMPROPERTY can be obtained using this function. The syntax of SYPR() is:

returnValue = SYPR(code)

where code is the same as defined for keyword SETSYSTEMPROPERTY in table 3.3-1, and returnValue is a numeric or string value for the corresponding system data. If the result is a string, the content can be read out with string function \$buffer(). It needs to be pointed out that function SYPR() doesn't support two argument properties in SETSYSTEMPROPERTY. Some special functions are needed to read two argument system properties, such as using WAVL(n) to get the value of nth wavelength, and using WWGT(n) to get the weight of nth wavelength, etc.

Besides using SYPR to read system properties, we can also use keyword GETSYSTEMDATA to get most system specific data. The syntax is:

GETSYSTEMDATA vector_expression

where vector_expression is the order of the 4 one-dimension array provided by ZPL (i.e. VEC1, VEC2, VEC3, VEC4). For example, GETSYSTEMDATA 3 means to read system properties and store them in array VEC3. The order of storage is shown in table 3.3-2:

Table 3.3-2 System Properties stored in array

Array Position	System Property
0	The number of system data values in the vector
1	Aperture Value
2	Apodization Factor
3	Apodization Type (0:none, 1:gaussian, 2:tangent)
4	Adjust Index Data To Environment setting (1 if true, 0 if false)
5	Temperature in degrees c (valid only if Use Env Data true)
6	Pressure in ATM (valid only if Use Env Data true)
7	Effective Focal Length
8	Image Space F/#
9	Object Space Numerical Aperture
10	Working F/#
11	Entrance Pupil Diameter
12	Entrance Pupil Position
13	Exit Pupil Diameter
14	Exit Pupil Position
15	Paraxial Image Height
16	Paraxial Magnification
17	Angular Magnification

18	Total Track
19	Ray Aiming (0 for off, 1 for paraxial, and 2 for real)
20	X Pupil Shift
21	Y Pupil Shift
22	Z Pupil Shift
23	Stop Surface Number
24	Global Coordinate Reference Surface Number
25	Telecentric object space (0 for off, 1 for on)
26	The number of configurations
27	The number of multi-configuration operands
28	The number of merit function operands
29	The number of tolerance operands
30	Afocal image space (0 for off, 1 for on)
31	X Pupil Compress
32	Y Pupil Compress

You may have noticed that the same system information may be obtained by using different keywords or function. For example, if we want to get Apodization Type of the system, we can either use function SYPR(12), or use keyword GETSYSTEMDATA 3 to realize. Please notice that numeric values can be directly obtained by functions, but if we use keywords, the numeric values of system information will be stored in one of the predefined arrays in ZPL.

We will give some examples on setting and reading system properties in ZPL.

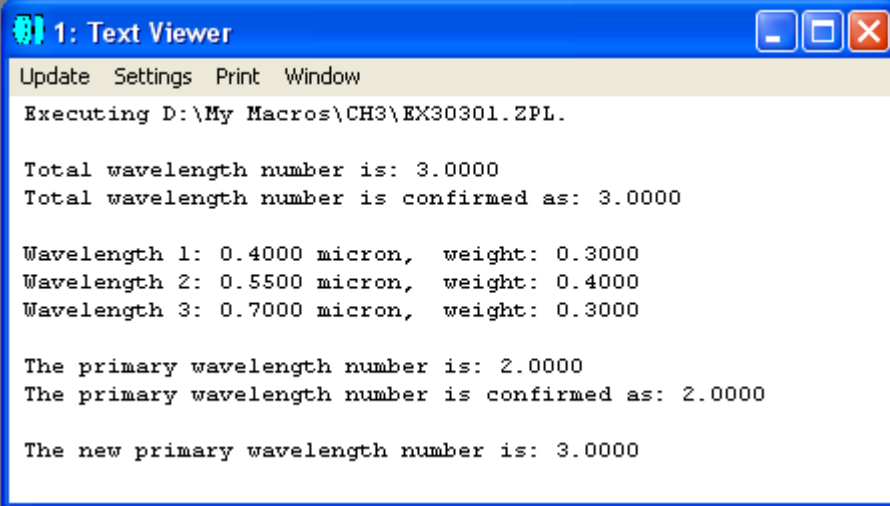
Example 3.3-1: setting and reading of working wavelength

```

1 ! ex30301
2 ! This program shows how to Set and Read working wavelengths
3
4 PRINT
5
6 SYSP 201, 3 # set total wavelength number as 3
7 totalWavelengthNumber = SYPR(201) # read total wavelength number
8 PRINT "Total wavelength number is: ", totalWavelengthNumber
9
10 totalWavelengthNumber = NWAV() # read total wavelength number again
11 PRINT "Total wavelength number is confirmed as: ", totalWavelengthNumber
12
13 SYSP 202, 1, 0.40 # set the 1st wavelength as 0.40 micron
14 SYSP 202, 2, 0.55 # set the 2nd wavelength as 0.55 micron
15 SYSP 202, 3, 0.70 # set the 3rd wavelength as 0.70 micron
16
17 SYSP 203, 1, 0.3 # set the 1st wavelength weight as 0.3
18 SYSP 203, 2, 0.4 # set the 2nd wavelength weight as 0.4
19 SYSP 203, 3, 0.3 # set the 3rd wavelength weight as 0.3
20
21 wavelength1 = WAVL(1) # read the 1st wavelength
22 wavelength2 = WAVL(2) # read the 2nd wavelength
23 wavelength3 = WAVL(3) # read the 3rd wavelength
24
25 wavelengthWeight1 = WWGT(1) # read the 1st weight
26 wavelengthWeight2 = WWGT(2) # read the 2nd weight
27 wavelengthWeight3 = WWGT(3) # read the 3rd weight
28
29 PRINT
30 PRINT "Wavelength 1: ", wavelength1, " micron, weight: ", wavelengthWeight1
31 PRINT "Wavelength 2: ", wavelength2, " micron, weight: ", wavelengthWeight2
32 PRINT "Wavelength 3: ", wavelength3, " micron, weight: ", wavelengthWeight3
33
34 SYSP 200, 2 # set the 2nd wavelength as the primary wavelength
35 primaryNumber = SYPR(200) # read the primary wavelength number
36
37 PRINT
38 PRINT "The primary wavelength number is: ", primaryNumber
39
40 primaryNumber = PWAV() # read the primary wavelength number again
41 PRINT "The primary wavelength number is confirmed as: ", primaryNumber
42
43 PWAV 3 # set the 3rd wavelength as the primary wavelength
44 primaryNumber = PWAV() # read the new primary wavelength number
45
46 PRINT
47 PRINT "The new primary wavelength number is: ", primaryNumber

```


In this example, first we use SYSP 201, 3 to set the total number of working wavelengths as 3 (line 6), and read this property with different methods SYPR(201) and NWAV() (lines 7 and 10), then we use SYSP 202, 1, 0.40 and SYSP 203, 1, 0.3 to set the first wavelength and its weight, and similarly, we set the second and third wavelength and weight (lines 13~19), and read these properties using WAVL() and WWGT() (lines 21~27), and finally, we use two different methods SYSP 200, 2 and PWAV 3 to set system primary wavelength number (lines 34, 43), and read it out using SYPR(200) and PWAV() (lines 35, 44). The result of this program is shown in figure 3.3-1:



```
1: Text Viewer
Update Settings Print Window
Executing D:\My Macros\CH3\EX30301.ZPL.

Total wavelength number is: 3.0000
Total wavelength number is confirmed as: 3.0000

Wavelength 1: 0.4000 micron, weight: 0.3000
Wavelength 2: 0.5500 micron, weight: 0.4000
Wavelength 3: 0.7000 micron, weight: 0.3000

The primary wavelength number is: 2.0000
The primary wavelength number is confirmed as: 2.0000

The new primary wavelength number is: 3.0000
```

Fig. 3.3-1 Result of program ex30301.ZPL

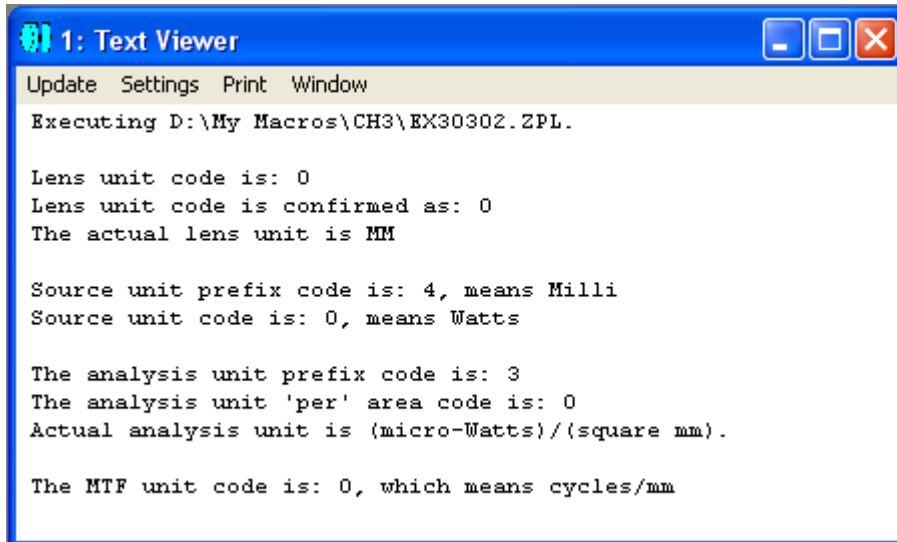
Example 3.3-2: setting and reading of units

```

1 ! ex30302
2 ! This program shows how to Set and Read units
3
4 PRINT
5 FORMAT 1.0
6
7 SYSP 30, 0 # set lens unit as mm
8 lensUnitCode = SYPR(30) # get lens unit code
9 PRINT "Lens unit code is: ", lensUnitCode
10
11 lensUnitCode = UNIT() # get lens unit code with a different method
12 PRINT "Lens unit code is confirmed as: ", lensUnitCode
13
14 lensUnit$ = $UNITS() # get the actual lens unit in string format
15 PRINT "The actual lens unit is ", lensUnit$
16
17 SYSP 31, 4 # set source unit prefix as Milli
18 SYSP 32, 0 # set source unit as Watts
19 sourceUnitPrefix = SYPR(31) # get the source unit prefix code
20 sourceUnit = SYPR(32) # get the source unit code
21 PRINT
22 PRINT "Source unit prefix code is: ", sourceUnitPrefix, ", means Milli"
23 PRINT "Source unit code is: ", sourceUnit, ", means Watts"
24
25 SYSP 33, 3 # set analysis unit prefix as Micro
26 SYSP 34, 0 # set analysis unit "per" area as square mm
27 analysisUnitPrefix = SYPR(33) # get the analysis unit prefix code
28 analysisUnitPerArea = SYPR(34) # get the analysis unit "per" area code
29 PRINT
30 PRINT "The analysis unit prefix code is: ", analysisUnitPrefix
31 PRINT "The analysis unit 'per' area code is: ", analysisUnitPerArea
32 PRINT "Actual analysis unit is (micro-Watts)/(square mm)." # see line 18
33
34 SYSP 35, 0 # set MTF unit
35 mtfUnit = SYPR(35) # get the MTF unit code
36 PRINT
37 PRINT "The MTF unit code is: ", mtfUnit, ", which means cycles/mm"

```

In this example, we first set the lens unit, and read it out with different methods, then we set and read the source unit and prefix, and then set and read analysis unit and prefix as well as analysis unit per area part (analysis unit of source intensity part has been set in line 18), and finally, we set and read the unit of modulation transfer function MTF. The result of program is shown in figure 3.3-2:



```

1: Text Viewer
Update Settings Print Window
Executing D:\My Macros\CH3\EX30302.ZPL.

Lens unit code is: 0
Lens unit code is confirmed as: 0
The actual lens unit is MM

Source unit prefix code is: 4, means Milli
Source unit code is: 0, means Watts

The analysis unit prefix code is: 3
The analysis unit 'per' area code is: 0
Actual analysis unit is (micro-Watts)/(square mm).

The MTF unit code is: 0, which means cycles/mm

```

Fig. 3.3-2 Result of ex30302.ZPL

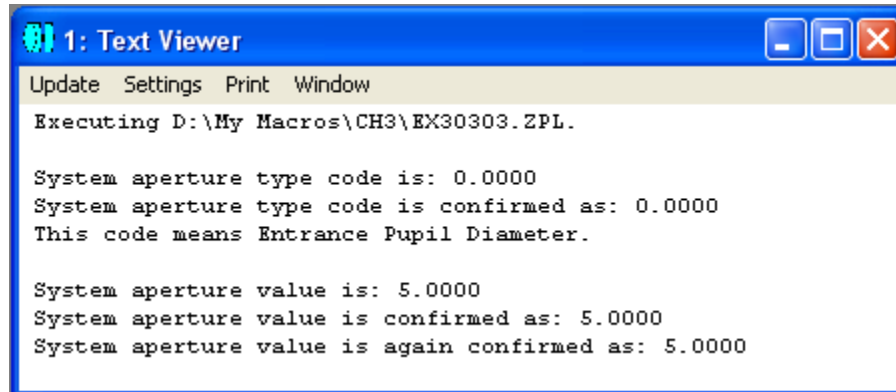
Example 3.3-3: setting and reading of system aperture

```

1 ! ex30303
2 ! This program shows how to Set and Read aperture
3
4 PRINT
5
6 SYSP 10, 0 # set system aperture as Entrance Pupil Diameter
7 apertureType = SYPR(10) # get system aperture type code
8 PRINT "System aperture type code is: ", apertureType
9
10 apertureType = ATYP() # get system aperture type code again
11 PRINT "System aperture type code is confirmed as: ", apertureType
12 PRINT "This code means Entrance Pupil Diameter."
13
14 SYSP 11, 5 # set system aperture value as 5 lens units
15 apertureValue = SYPR(11) # get system aperture value
16 PRINT
17 PRINT "System aperture value is: ", apertureValue
18
19 apertureValue = AVAL() # get system aperture value
20 PRINT "System aperture value is confirmed as: ", apertureValue
21
22 GETSYSTEMDATA 1 # get system information and store them in VEC1
23 apertureValue = VEC1(1) # get system aperture value
24 PRINT "System aperture value is again confirmed as: ", apertureValue

```

In this example, we set the type and value of the system aperture, and read them out with three different methods. Please note that system aperture is different from the aperture of a particular lens surface. We will further discuss the latter one in next section. The result of the program is shown in figure 3.3-3.



```
1: Text Viewer
Update Settings Print Window
Executing D:\My Macros\CH3\EX30303.ZPL.

System aperture type code is: 0.0000
System aperture type code is confirmed as: 0.0000
This code means Entrance Pupil Diameter.

System aperture value is: 5.0000
System aperture value is confirmed as: 5.0000
System aperture value is again confirmed as: 5.0000
```

Fig. 3.3-3 Result of ex30303.ZPL

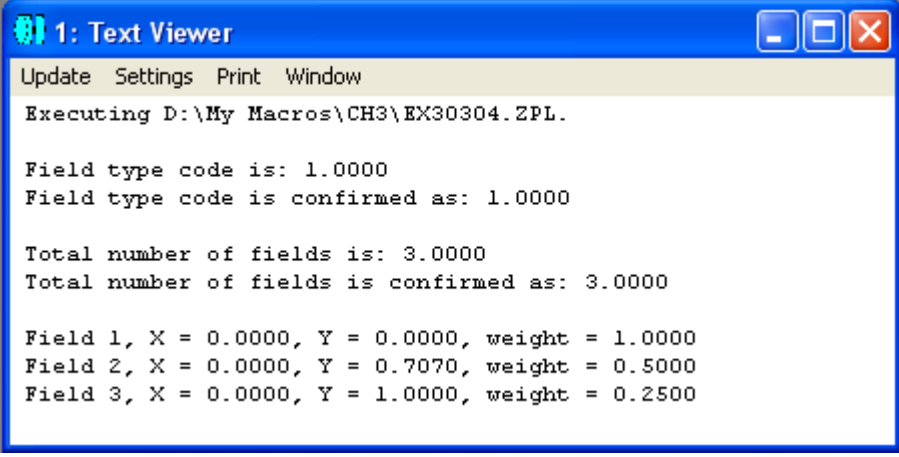
Example 3.3-4: setting and reading of field

```

1 ! ex30304
2 ! This program shows how to set and read field
3
4 PRINT
5
6 SYSP 100, 1 # set field as Object Height
7 fieldType = SYPR(100) # get field type code
8 PRINT "Field type code is: ", fieldType
9
10 fieldType = FTYP() # get field type code
11 PRINT "Field type code is confirmed as: ", fieldType
12
13 SYSP 101, 3 # set total number of fields as 3
14 totalFieldNum = SYPR(101) # get total number of fields
15 PRINT
16 PRINT "Total number of fields is: ", totalFieldNum
17 totalFieldNum = NFLD() # get total number of fields
18 PRINT "Total number of fields is confirmed as: ", totalFieldNum
19
20 SYSP 102, 1, 0 # set x of field 1 as 0
21 SYSP 103, 1, 0 # set y of field 1 as 0
22 SYSP 104, 1, 1 # set weight of field 1 as 1
23 SYSP 102, 2, 0 # set x of field 2 as 0
24 SYSP 103, 2, 0.707 # set y of field 2 as 0.707
25 SYSP 104, 2, 0.5 # set weight of field 2 as 0.5
26 SYSP 102, 3, 0 # set x of field 3 as 0
27 SYSP 103, 3, 1 # set y of field 3 as 1
28 SYSP 104, 3, 0.25 # set weight of field 3 as 0.25
29
30 fld1X = FLDX(1) # get x of field 1
31 fld1Y = FLDY(1) # get y of field 1
32 fldWeight1 = FUGT(1) # get weight of field 1
33 fld2X = FLDX(2) # get x of field 2
34 fld2Y = FLDY(2) # get y of field 2
35 fldWeight2 = FUGT(2) # get weight of field 2
36 fld3X = FLDX(3) # get x of field 3
37 fld3Y = FLDY(3) # get y of field 3
38 fldWeight3 = FUGT(3) # get weight of field 3
39
40 PRINT
41 PRINT "Field 1, X = ", fld1X, ", Y = ", fld1Y, ", weight = ", fldWeight1
42 PRINT "Field 2, X = ", fld2X, ", Y = ", fld2Y, ", weight = ", fldWeight2
43 PRINT "Field 3, X = ", fld3X, ", Y = ", fld3Y, ", weight = ", fldWeight3

```

In this example, we first set the type of field, and read it with two different methods, then we set the total number of field, and read it with two methods, and finally we set and read x, y coordinate and weight of each field. The result is shown below:



```

1: Text Viewer
Update Settings Print Window
Executing D:\My Macros\CH3\EX30304.ZPL.

Field type code is: 1.0000
Field type code is confirmed as: 1.0000

Total number of fields is: 3.0000
Total number of fields is confirmed as: 3.0000

Field 1, X = 0.0000, Y = 0.0000, weight = 1.0000
Field 2, X = 0.0000, Y = 0.7070, weight = 0.5000
Field 3, X = 0.0000, Y = 1.0000, weight = 0.2500

```

Fig. 3.3-4 Result of ex30304.ZPL

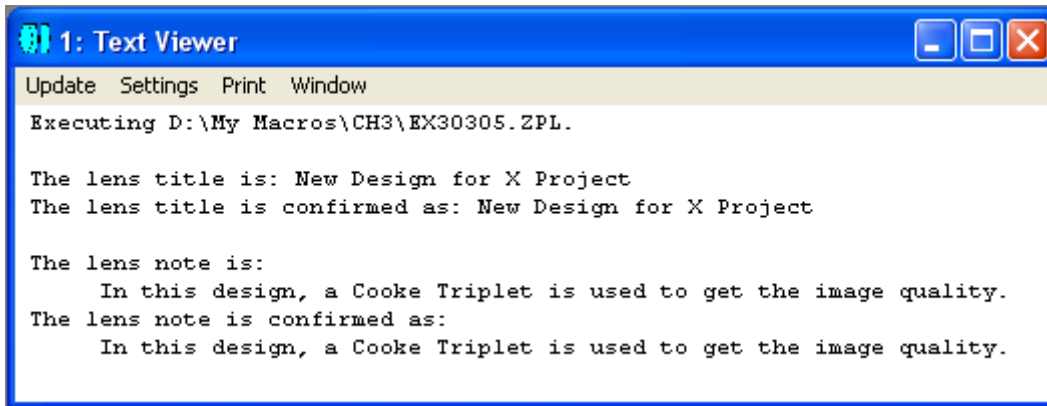
Example 3.3-5: setting and reading of lens title and lens note

```

1 ! ex30305
2 ! This program shows how to set and read lens title and lens note
3
4 PRINT
5
6 SYSP 16, "New Design for X Project" # set lens title
7 SYSP 17, "In this design, a Cooke Triplet is used to get the image quality."
8 ! line 7 sets the lens note
9
10 dummy = SYPR(16) # use a dummy variable to call SYPR() function
11 lensTitle$ = $BUFFER() # the string information is put in a buffer
12 PRINT "The lens title is: ", lensTitle$
13
14 lensTitle$ = $LENSNAME() # get the lens title using a different method
15 PRINT "The lens title is confirmed as: ", lensTitle$
16
17 dummy = SYPR(17) # use a dummy variable to call SYPR() function
18 lensNote$ = $BUFFER() # string information can be obtained from $BUFFER()
19 PRINT
20 PRINT "The lens note is: "
21 PRINT " ", lensNote$
22
23 lensNote$ = $NOTE(1) # get the lens note using a different method
24 PRINT "The lens note is confirmed as: "
25 PRINT " ", lensNote$

```

In this example, we set the lens title and read with two different methods. Similarly, we set lens note and read with two different methods. Please notice that if we want to read string information with function SYPR(), the returned result will be stored in the memory buffer, and needs to be read out using string function \$BUFFER(). The result of this program is shown in figure 3.3-5.



```
1: Text Viewer
Update Settings Print Window
Executing D:\My Macros\CH3\EX30305.ZPL.

The lens title is: New Design for X Project
The lens title is confirmed as: New Design for X Project

The lens note is:
    In this design, a Cooke Triplet is used to get the image quality.
The lens note is confirmed as:
    In this design, a Cooke Triplet is used to get the image quality.
```

Fig. 3.3-5 Result of ex30305.ZPL

In the above shown examples, we discussed the basic process of setting and reading important system properties in ZPL. Please remember that many system properties can be directly set and read in ZEMAX environment instead of through ZPL. The user needs to determine which way is more efficient based on his own case.

Also, we cannot cover all the commands related to system properties here. We will continue to discuss more commands later when needed, however, we strongly encourage readers to dig into Zemax User's Manual for details of different commands.

3.4 Setting and Reading Lens Properties

Lens Data Editor is an important place to do lens design in Zemax. As we know, in sequential ray tracing, Zemax defines an optical system as being made up of various surfaces, and most of the properties related to surfaces are set in Lens Data Editor. In this section, we will discuss how to set and read lens surface properties in ZPL through various examples.

To set lens surface properties, ZPL provides an important keyword SETSURFACEPROPERTY (or SURP in short). The syntax is:

SETSURFACEPROPERTY surface, code, value1, value2

or

SURP surface, code, value1, value2

where *surface* is an expression that evaluates to an integer specifying the surface number. The code may either be an expression that evaluates to an integer or a mnemonic which specifies what property of the surface is being modified. The third and fourth arguments are the new values for the specified property, and they may be either text in quotes, a string variable, or a numeric expression, depending upon the code. For most codes, the property value being modified is defined by the *value1* argument. A few operands require both a *value1* and a *value2*, as described in the table below.

Table 3.4-1

Code	Property
<i>Basic surface data</i>	
0 or TYPE	Surface type. The value should be the name of the object, such as "STANDARD" for the standard surface. The names for each surface type are listed in the Prescription Report in the Surface Data Summary for each surface type currently in the Lens Data Editor.
1 or COMM	Comment.
2 or CURV	Curvature (not radius) in inverse lens units. Use zero for an infinite radius.
3 or THIC	Thickness in lens units.
4 or GLAS	Glass name.
5 or CONI	Conic constant.
6 or SDIA	Semi-diameter. If the value is zero or positive, the semi-diameter solve is set to "Fixed". If the value is negative, the semi-diameter solve is set to "Automatic" and the semi-diameter will be computed with the next UPDATE keyword.

7 or TCE	Thermal coefficient of expansion.
8 or COAT	Coating name. Use a blank string for value1 to remove the coating.
9 or SDLL	User defined surface DLL name.
10 or PARM	Parameter value. Value1 is the new value. Value2 is the parameter number.
11 or EDVA	Extra Data value. Value1 is the new value. Value2 is the extra data number.
12	Surface color, Use 0 for default.
13	Surface opacity.
14	Row color.
15	Surface cannot be hyperhemispheric. Use 1 to avoid surface being hyperhemispheric.
16	Ignore surface. Use 1 to ignore surface, 0 to not ignore surface.
17 or CODE	The integer code for the surface type. The integer code is an alternative to the surface name used by code 0.
18 or GLAN	Glass number. See also Code 4.
Surface aperture data.	
20 or ATYP	Surface aperture type code.
21 or APP1	Surface aperture parameter 1.
22 or APP2	Surface aperture parameter 2.
23 or APDX	Surface aperture decenter x.
24 or APDY	Surface aperture decenter y.
25 or UDA	User Defined Aperture (UDA) file name.
26 or APPU	Surface aperture pick up from surface number. Use 0 for no pickup.
Physical Optics Propagation Settings.	
30	Physical Optics setting "Use Rays To Propagate To Next Surface". Use 1 for true, 0 for false.
31	Physical Optics setting "Do Not Rescale Beam Size Using Ray Data". Use 1 for true, 0 for false.
32	Physical Optics setting "Use Angular Spectrum Propagator". Use 1 for true, 0 for false.
33	Physical Optics setting "Draw ZBF On Shaded Model". Use 1 for true, 0 for false.
34	Physical Optics setting "Recompute Pilot Beam Parameters". Use 1 for true, 0 for false.
35	Physical Optics setting "Resample After Refraction". Use 1 for true, 0 for false.

36	Physical Optics setting "Auto Resample". Use 1 for true, 0 for false.
37	Physical Optics setting "New X Sampling". Use 1 for 32, 2 for 64, etc.
38	Physical Optics setting "New Y Sampling". Use 1 for 32, 2 for 64, etc.
39	Physical Optics setting "New X-Width". New total x direction width of array.
40	Physical Optics setting "New Y-Width". New total y direction width of array.
41	Physical Optics setting "Output Pilot Radius". Use 0 for best fit, 1 for shorter, 2 for longer, 3 for x, 4 for y, 5 for plane, 6 for user.
42, 43	Physical Optics setting "X-Radius" and "Y-Radius", respectively.
44	Physical Optics setting "Use X-axis Reference". Use 1 for true, 0 for false.
Coating Settings.	
50	Use Layer Multipliers and Index Offsets. Use 1 for true, 0 for false.
51	Layer Multiplier value. Value1 is the new value. Value2 is the layer number.
52	Layer Multiplier status. Value 1 is the status, use 0 for fixed, 1 for variable, or n+1 for pickup from layer n. Value2 is the layer number.
53	Layer Index Offset value. Value1 is the new value. Value2 is the layer number.
54	Layer Index Offset status. Value 1 is the status, use 0 for fixed, 1 for variable, or n+1 for pickup from layer n. Value2 is the layer number.
55	Layer Extinction Offset value. Value1 is the new value. Value2 is the layer number.
56	Layer Extinction Offset status. Value 1 is the status, use 0 for fixed, 1 for variable, or n+1 for pickup from layer n. Value2 is the layer number.
Surface Tilt and Decenter Data.	
60 or BOR	Before tilt and decenter order. Use 0 for dec/tilt, 1 for tilt/dec.
61 or BDX	Before decenter x.
62 or BDY	Before decenter y.
63 or BTX	Before tilt about x.
64 or BTY	Before tilt about y.
65 or BTZ	Before tilt about z.
66 or APU	After pick up status: 0 for explicit, 1/2 for pickup/reverse current surface, 3/4 for pickup/reverse current surface minus 1, 5/6 for pickup/reverse current surface minus 2, etc...
70 or AOR	After tilt and decenter order. Use 0 for dec/tilt, 1 for tilt/dec.
71 or ADX	After decenter x.
72 or ADY	After decenter y.

73 or ATX	After tilt about x.
74 or ATY	After tilt about y.
75 or ATZ	After tilt about z.
76	Coordinate Return status. Valid only on Coordinate Break surfaces. Use 0 for None, 1 for Orientation Only, 2 for Orientation XY, and 3 for Orientation XYZ.
77	Coordinate Return To Surface. Valid only on Coordinate Break surfaces.
<i>Surface scatter data.</i>	
80	Sets the scatter code: 0 for none, 1 for Lambertian, 2 for Gaussian, 3 for ABg, 4 for DLL, 5 for BSDF, 6 for ABg File, and 7 for IS Scatter Catalog.
81	Sets the scatter fraction, should be between 0.0 and 1.0.
82	Sets the Gaussian scatter sigma.
83	Sets the ABg file name.
84	Sets the name of the user defined scattering DLL. To set the parameters see Code 181.
85	Sets the name of the data file used by the user defined scattering DLL.
86	Sets the BSDF file name. The value should be the name of the BSDF file with no path (i.e. BrownVinyl.bsdf).
87	Sets the ABg File data file name. The value should be the name of the ABGF file with no path (e.g. SampleABGF.abgf).
88	Sets the IS Scatter Catalog data file name. The value should be the name of the ISX file with no path (e.g. BrownVinyl.ISX).
110	Sets the side for IS Scatter Catalog scattering. Use 0 for front, 1 for back.
111	Sets the sampling for IS Scatter Catalog scattering. Use 0 for 5 degrees, 1 for 2 degrees, and 2 for 1 degree.
<i>Surface draw data.</i>	
91	Sets the “Skip Rays To This Surface” checkbox status: 0 for off, 1 for on.
92	Sets the “Do Not Draw This Surface” checkbox status: 0 for off, 1 for on.
93	Sets the “Do Not Draw Edges From This Surface” checkbox status: 0 for off, 1 for on.
96	Sets the “Draw Edges As” status: 0 for squared, 1 for tapered, 2 for flat.
97	Sets “Mirror Substrate” status: 0 for none, 1 for flat, 2 for curved.
98	Sets the mirror substrate thickness value.
<i>User defined surface scatter DLL parameters.</i>	
181-186	Sets the user defined scatter DLL parameters 1-6.

To delete a certain surface, we can use the keyword DELETE provided by ZPL. The syntax is:

DELETE n

where n is the order number of the surface to be deleted.

To insert a surface at a certain location, we can use the keyword INSERT. The syntax is:

INSERT n

where n is the order number of the surface in front of which the new surface is to be inserted.

To set a surface as the system stop aperture, we can use the keyword STOPSURF. The syntax is:

STOPSURF n

This can set the nth surface as the stop aperture.

It needs to be noticed that after the surface properties were set or modified, keyword UPDATE is usually needed to update the surface properties.

Similar to setting surface properties, ZPL also provides two important functions SPRO() and SPRX() to read the surface properties. The syntax is:

SPRO(surf, code)

and

SPRX(surf, code, value2)

where surf is the order number of the surface to be read, and code value is defined by keyword SURP in table 3.4-1. Please note that code can only be an integer and cannot be mnemonic. The main difference between the two functions is that SPRO() supports commands with one argument in table 3.4-1, and SPRX() supports commands with two arguments in table 3.4.1.

Besides SPRO() and SPRX(), ZPL also provides other important functions to read out surface related properties, as shown in table 3.4-2:

Table 3.4-2 Some important functions used to read out surface properties

Function	Return value
NSUR()	The number of defined surfaces.
APMN(n)	The minimum radius value. For spider apertures, this is the width of the arms. For rectangular and elliptical apertures, it is the x-half width of the aperture.
APMX(n)	The maximum radius value. For spider apertures, this is the number of arms. For rectangular and elliptical apertures, it is the y-half width of the aperture.
APXD(n)	The aperture x-decenter value.
APYD(n)	The aperture y-decenter value.
APTP(n)	An integer code describing the aperture type on the specified surface.
CONI(n)	Conic constant of the surface.
CURV(n)	Curvature of the surface.
EDGE(n)	Edge thickness at the semi-diameter of that surface.
GLCA(n)	Global vertex x vector of the specified surface.
GLCB(n)	Global vertex z vector of the specified surface.
GLCC(n)	Global vertex z vector of the specified surface.
GLCM(n, item)	For item equal to 1-9, the return value is R11, R12, R13, R21, R22, R23, R31, R32, or R33. For item equal to 10-12, the return value is the x, y, or z component of the global offset vector.
GLCX(n)	Global vertex x-coordinate of the specified surface.
GLCY(n)	Global vertex y-coordinate of the specified surface.
GLCZ(n)	Global vertex z-coordinate of the specified surface.
GRIN(n, w, x, y, z)	Returns the index of refraction at the specified x, y, z position on surface n at wavelength number w. Works for gradient and non-gradient media.
PARM(p,n)	Parameter "p" of surface "n".
RADI(n)	Radius of curvature of surface. If the surface has an infinite radius, RADI returns 0.0. This possibility must be considered to avoid potential divide by zero errors.
SDIA(n)	Semi-diameter of surface.
SURC(A\$)	Surface with comment. Returns the first surface number where the comment matches the string A\$. The comparison is case insensitive. If no surface has the matching comment the function returns -1.
THIC(n)	Thickness of the surface.

Among the surface-related properties, the material of the surface is a very important one. For convenience, let's just call all the material as glass. In table 3.4-3 some important surface glass related functions are listed.

Table 3.4-3 Important functions to read surface glass parameters

Function	Return value
GABB(n)	The glass catalog Abbe number of the glass for the specified surface.
GIND(n)	The glass catalog d-light index of the glass for the specified surface.
GNUM(A\$)	If the string A\$ is the name of a valid glass, such as BK7, then GNUM returns the number of the glass in the glass catalog. The glass number can subsequently be used by SETSURFACEPROPERTY to set the glass type on a surface. If A\$ does not correspond to any glass in the catalog, GNUM returns 0. GNUM returns -1 if the string is "MIRROR".
GPAR(n)	The glass catalog partial dispersion coefficient of the glass for the specified surface.
GRIN(n, w, x, y, z)	Returns the index of refraction at the specified x, y, z position on surface n at wavelength number w. Works for gradient and non-gradient media.
INDX(n)	Index of refraction at the primary wavelength. See ISMS.
ISMS(n)	If the surface is an odd mirror, or follows an odd mirror but is not a mirror, the return value will be one, otherwise the return value is 0.
MAXG()	The number of glasses currently loaded.
TMAS()	The total mass in grams of the lens from surface 1 to the image surface.

Besides the functions listed above, ZPL also provides a keyword GETGLASSDATA to read glass data in the current catalogs. The syntax is:

GETGLASSDATA vector_expression, glass_number

where vector_expression is the sequence number of the 4 vectors provided by ZPL (VEC1, VEC2, VEC3, VEC4), glass_number is the sequence number of the glass listed in the glass catalog, and can be read with function GNUM(). The glass parameters are stored in vectors VECn (n is 1, 2, 3 or 4) according to table 3.4-4.

Table 3.4-4 Glass data retrieve command

Array position	Glass data
0	The number of data values in the vector
1	Formula number: The number indicates the formula as follows: 1 = Schott, 2 = Sellmeier 1, 3 = Herzberger, 4 = Sellmeier 2, 5 = Conrady, 6 = Sellmeier 3, 7 = Handbook of Optics 1, 8 = Handbook of Optics 2, 9 = Sellmeier 4, 10 = Extended, 11 = Sellmeier 5, 12 = Extended 2
2	Reference temperature in degrees c
3	Refractive index at d line Nd
4	Abbe number Vd
5	Thermal coefficient of expansion -30 to +70 °C
6	Thermal coefficient of expansion +20 to 300 °C
7	Density in g/cm ³
8	Deviation from normal line P gf
9	Lambda min
10	Lambda max
11~16	Constants of dispersion A0-A5 (meaning depends upon formula)
17~22	Thermal constants of dispersion
23~ (22 + #waves)	Internal transmission coefficient (per mm) alpha, $T = \exp(\alpha * \text{path})$. The alpha for wavelength 1 is stored in 23, wavelength 2 is in 24, etc., up to the number of wavelengths used by the system.
(23 + #waves) ~ (32 + #waves)	Constants of dispersion A0-A9 (meaning depends upon formula)

We will give some examples to show how to set and read important surface related parameters in ZPL.

Example 3.4-1: Construct a doublet.

In this example, we will start from scratch, build a doublet lens in Lens Data Editor (LDE) in Zemax through ZPL. Assume the basic parameters of the doublet is:

Working wavelength: $F = 0.48613270\mu\text{m}$, $d = 0.58756180\mu\text{m}$, $C = 0.65627250\mu\text{m}$

Entrance pupil diameter: 50mm

F/#: F/8

Full field angle: 10°

Boundary constraints: minimum thickness at edge and center is 4mm, maximum thickness at center is 18mm

Glass material: BK7 and F2

After some simple calculation (omitted here), we can get initial parameters of each surface of the doublet. In this example, we will use these parameters to construct the doublet. Later on in other examples we will have opportunities to further analyze and optimize this doublet.

First, from Zemax file menu, choose “new” to get an empty lens data editor that includes an object surface, an image surface, and a lens surface. In our program, we need to set some system parameters such as the type and size of the system aperture, type and size of field, working wavelength, etc. After that, we need to insert enough number of lens surfaces, and input lens data. The program is shown below:


```
1 ! ex30401
2 ! This program shows how to create a doublet from scratch
3
4 ! set system parameters
5 SYSP 30, 0 # set lens unit as mm
6
7 SYSP 10, 0 # set system aperture as Entrance Pupil Diameter
8 SYSP 11, 50 # set system aperture value as 50mm
9
10 SYSP 201, 3 # set total wavelength number as 3
11 SYSP 202, 1, 0.48613270 # set the 1st wavelength as 0.48613270 micron
12 SYSP 202, 2, 0.58756180 # set the 2nd wavelength as 0.58756180 micron
13 SYSP 202, 3, 0.65627250 # set the 3rd wavelength as 0.65627250 micron
14 SYSP 203, 1, 1 # set the 1st wavelength weight as 1
15 SYSP 203, 2, 1 # set the 2nd wavelength weight as 1
16 SYSP 203, 3, 1 # set the 3rd wavelength weight as 1
17
18 SYSP 200, 2 # set the 2nd wavelength as the primary wavelength
19
20 SYSP 100, 0 # set the field type as Angle
21 SYSP 101, 3 # set the total field number as 3
22 SYSP 102, 1, 0 # set field 1 as x = 0 degree
23 SYSP 103, 1, 0 # set field 1 as y = 0 degree
24 SYSP 104, 1, 1 # set field 1 as weight = 1
25 SYSP 102, 2, 0 # set field 2 as x = 0 degree
26 SYSP 103, 2, 3.5 # set field 2 as y = 3.5 degree
27 SYSP 104, 2, 1 # set field 2 as weight = 1
28 SYSP 102, 3, 0 # set field 3 as x = 0 degree
29 SYSP 103, 3, 5 # set field 3 as y = 5 degree
30 SYSP 104, 3, 1 # set field 3 as weight = 1
31
```

```

31
32 ! set surface 1 as stop
33 STOPSURF 1
34
35 ! insert 3 surfaces after stop
36 INSERT 2
37 INSERT 2
38 INSERT 2
39
40 ! set surface parameters
41 SURP 1, THIC, 275 # set surface 1 thickness as 275
42
43 SURP 2, TYPE, "STANDARD" # set surface 2 type as "STANDARD", can be omitted
44 SURP 2, COMM, "front f1" # set surface 2 comment
45 SURP 2, CURV, 1/600 # set surface 2 curvature as 1/600
46 SURP 2, THIC, 18 # set surface 2 thickness as 18
47 SURP 2, GLAS, "BK7" # set surface 2 glass type as BK7
48
49 SURP 3, COMM, "back f1/front f2" # set surface 3 comment
50 SURP 3, CURV, -1/115 # set surface 3 curvature as -1/115
51 SURP 3, THIC, 18 # set surface 3 thickness as 18
52 SURP 3, GLAS, "F2" # set surface 3 glass type as F2
53
54 SURP 4, COMM, "back f2" # set surface 4 comment
55 SURP 4, CURV, -1/243 # set surface 4 curvature as -1/243
56 SURP 4, THIC, 395 # set surface 4 thickness as 395
57
58 UPDATE

```

The first part of the program (lines 5 ~ 30) sets system properties, including lens unit (line 5), system aperture (lines 7 and 8), wavelengths (lines 10 ~ 18), field (lines 20 ~ 30). The second part of the program (lines 32 ~ 58) sets lens data, including choosing stop surface (line 33), inserting 3 new surfaces (lines 36 ~ 38), and defining type, comments, curvature, thickness, material of each surface (lines 41 ~ 56). At the end of the program, we updated the system to assure the data are accepted. Apparently, a lot of data can be directly set in LDE without a ZPL program. This example just shows how to do it through ZPL program. Also, a lot of default setting in LDE (such as "standard surface" in line 51) can be omitted in the program.

The result of program ex30401.ZPL is shown below in figure 3.4-1:

Surf	Type	Comment	Radius	Thickness	Glass	Semi-Diameter	Conic
OBJ	Standard		Infinity	Infinity		Infinity	0.000000
ST0	Standard		Infinity	275.000000		25.000000	0.000000
2	Standard	front f1	600.000000	18.000000	BK7	49.236425	0.000000
3	Standard	back f1/front f2	-115.000000	18.000000	F2	49.378739	0.000000
4	Standard	back f2	-243.000000	395.000000		50.762655	0.000000
IMA	Standard		Infinity	-		34.945937	0.000000

Fig. 3.4-1 The updated content of LDE after program ex30401.ZPL is executed.

For the doublet constructed in program ex30401, we can read various parameters through ZPL program. Some examples are given below.

Example 3.4-2: read lens surface data.

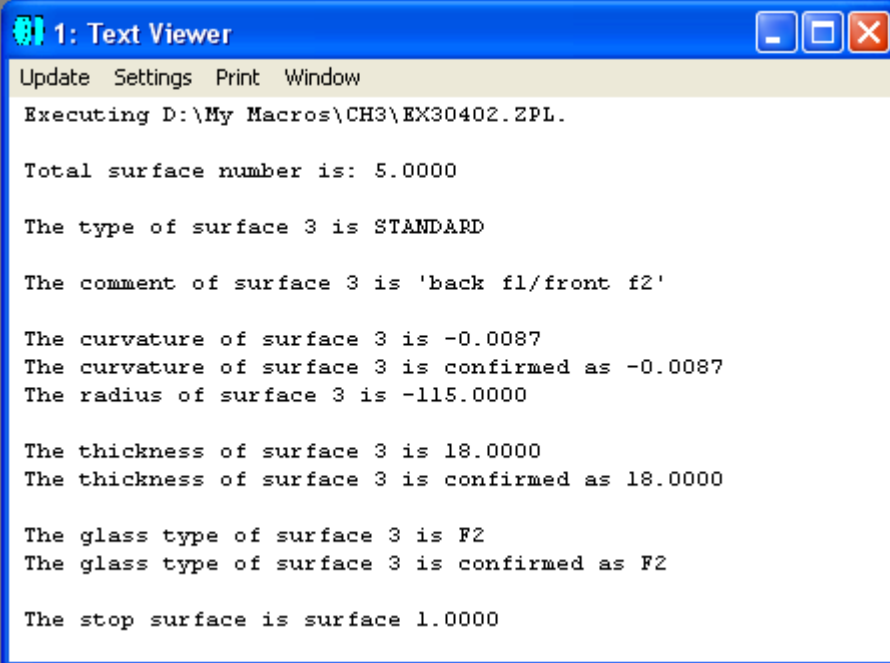
In this program, we first read the total number of surfaces of the the system (line 7), then for surface 3, we read surface type (lines 11 and 12), comments (lines 16 and 17), curvature (lines 21 and 23), radius (line 25), thickness (lines 29 and 31) and material (lines 35, 36 and 38), and finally we read the surface number of the stop (lines 42 and 43). We can see that when using function `SPRO()` to read string information (such as surface type, comments, etc.), the returned data are stored in the buffer, and can be read out with buffer string function `$buffer()`.

```

1 ! ex30402
2 ! This program shows how to read surface data
3 ! Assume the lens is defined in ex30401
4
5 PRINT
6
7 surfTotalNum = NSUR() # get the total surface number
8 PRINT "Total surface number is: ", surfTotalNum
9 PRINT
10
11 dummy = SPRO(3,0) # get the type of surface 3
12 surfaceType$ = $buffer() # read the type of surface 3 from buffer
13 PRINT "The type of surface 3 is ", surfaceType$
14 PRINT
15
16 dummy = SPRO(3,1) # get the comment of surface 3
17 surfaceComment$ = $buffer() # read the comment from buffer
18 PRINT "The comment of surface 3 is '", surfaceComment$, "'"
19 PRINT
20
21 curvOfSurf = SPRO(3,2) # get the curvature of surface 3
22 PRINT "The curvature of surface 3 is ", curvOfSurf
23 curvOfSurf = CURV(3) # get the curvature by a different way
24 PRINT "The curvature of surface 3 is confirmed as ", curvOfSurf
25 radiOfSurf = RADI(3) # get the radius of surface 3
26 PRINT "The radius of surface 3 is ", radiOfSurf
27 PRINT
28
29 thickness = SPRO(3,3) # get the thickness of surface 3
30 PRINT "The thickness of surface 3 is ", thickness
31 thickness = THIC(3) # get the thickness by a different way
32 PRINT "The thickness of surface 3 is confirmed as ", thickness
33 PRINT
34
35 dummy = SPRO(3,4) # get the glass type of surface 3
36 glassType$ = $buffer() # read the glass type from buffer
37 PRINT "The glass type of surface 3 is ", glassType$
38 glassType$ = $GLASS(3) # get the glass type by a different way
39 PRINT "The glass type of surface 3 is confirmed as ", glassType$
40 PRINT
41
42 GETSYSTEMDATA 1 # get system information and save into VEC1
43 stopNum = VEC1(23) # read the stop surface number from VEC1
44 PRINT "The stop surface is surface ", stopNum
45

```

The result of program ex30402.ZPL is shown in figure 3.4-2:



1: Text Viewer

Update Settings Print Window

```
Executing D:\My Macros\CH3\EX30402.ZPL.  
  
Total surface number is: 5.0000  
  
The type of surface 3 is STANDARD  
  
The comment of surface 3 is 'back f1/front f2'  
  
The curvature of surface 3 is -0.0087  
The curvature of surface 3 is confirmed as -0.0087  
The radius of surface 3 is -115.0000  
  
The thickness of surface 3 is 18.0000  
The thickness of surface 3 is confirmed as 18.0000  
  
The glass type of surface 3 is F2  
The glass type of surface 3 is confirmed as F2  
  
The stop surface is surface 1.0000
```

Fig. 3.4-2 Result of program ex30402.ZPL

Example 3.4-3: read glass material parameters:

In this example, various functions and keywords are used to read material parameters.

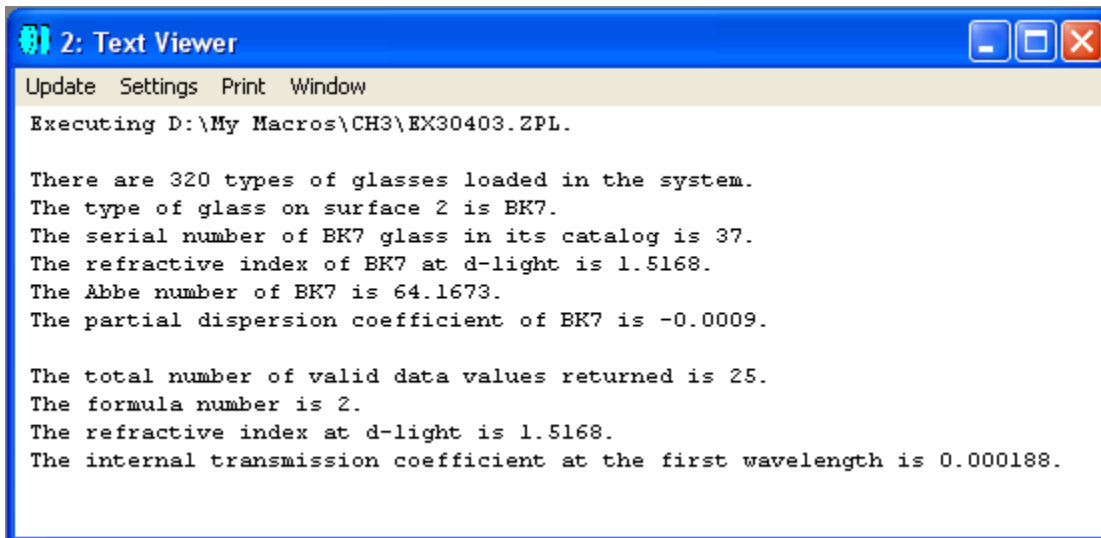
```

1 ! ex30403
2 ! This program shows how to read glass data
3 ! Assume the lens system is defined in ex30401
4
5 PRINT
6
7 ! get the total number of glass types currently loaded in the system
8 FORMAT 1 INT
9 PRINT "There are ", MAXG(), " types of glasses loaded in the system."
10
11 ! get the glass name
12 surfaceNum = 2
13 glassName$ = $GLASS(surfaceNum)
14 PRINT "The type of glass on surface ", surfaceNum, " is ", glassName$, "."
15
16 ! get the glass number
17 PRINT "The serial number of ", glassName$, " glass in its catalog ",
18 PRINT "is ", GNUM(glassName$), "."
19
20 ! get the refractive index of glass at d-light
21 PRINT "The refractive index of ", glassName$, " at d-light is ",
22 FORMAT 5.4
23 PRINT GIND(surfaceNum), "."
24
25 ! get the Abbe number of glass
26 PRINT "The Abbe number of ", glassName$, " is ", GABB(surfaceNum), "."
27
28 ! get the partial dispersion coefficient of glass
29 PRINT "The partial dispersion coefficient of ", glassName$, " is ",
30 PRINT GPAR(surfaceNum), "."
31
32 ! use keyword GETGLASSDATA to get glass data
33 GETGLASSDATA 1, GNUM(glassName$)
34 PRINT
35 FORMAT 1 INT
36 PRINT "The total number of valid data values returned is ", VEC1(0), "."
37 PRINT "The formula number is ", VEC1(1), "."
38 FORMAT 5.4
39 PRINT "The refractive index at d-light is ", VEC1(3), "."
40 PRINT "The internal transmission coefficient at the first wavelength is ",
41 FORMAT 7.6
42 PRINT VEC1(23), "."

```

In the program, function MAXG() was used in line 9 to read the total number of glasses loaded in the system, and function \$GLASS() was used in lines 12 ~ 14 to read the glass type of a given surface. Please note that different from directly calling numeric function in line 9, since PRINT command cannot directly

call string function, the program needs to store the result in a string variable in line 13 first, and then print out the result in line 14. Function GNUM() was used in lines 17 ~ 18 to read the order number of the glass material in the catalog, and function GIND() was used in lines 21 ~ 23 to read the refractive index of the glass at d line. Function GABB() in line 26 read the Abbe number for a given surface. Function GPAR() in lines 29 ~ 30 was used to read partial dispersion of given surface material. As mentioned before, besides using functions to read glass data, keyword GETGLASSDATA can also be used to obtain the information. Lines 33 ~ 42 in the program shows how to do it. The information read with keyword is stored in the default array VEC1, and can be easily read out. The result of the program is shown in figure 3.4-3:



```
2: Text Viewer
Update Settings Print Window
Executing D:\My Macros\CH3\EX30403.ZPL.

There are 320 types of glasses loaded in the system.
The type of glass on surface 2 is BK7.
The serial number of BK7 glass in its catalog is 37.
The refractive index of BK7 at d-light is 1.5168.
The Abbe number of BK7 is 64.1673.
The partial dispersion coefficient of BK7 is -0.0009.

The total number of valid data values returned is 25.
The formula number is 2.
The refractive index at d-light is 1.5168.
The internal transmission coefficient at the first wavelength is 0.000188.
```

Fig. 3.4-3 Result of program ex30403.ZPL

3.5 Merit Function

When using Zemax to design and optimize an optical system, merit function is often used. It is a numerical value defined by the user and used to evaluate the deviation of an optical system performance relative to a series of design targets. In Zemax there is a merit function editor that includes different operands. Each operand is used to evaluate a certain system constraint or design target. The whole merit function is composed with various operands in the merit function editor with different weights.

ZPL provided various keywords and functions to set and read merit function.

Keyword DEFAULTMERIT is used to generate a default merit function. The syntax is:

DEFAULTMERIT type, data, reference, method, rings, arms, grid, delete, axial, lateral, start, xweight, oweight, pup_obsc

(Note: pup_obsc is used in newer versions of Zemax. In some older versions, this parameter is not included, as in the examples below.)

This keyword generates a default merit function in the Merit Function Editor. Any existing default merit function will be deleted. The values are as follows:

Table 3.5-1: parameters of keyword DEFAULTMERIT

Parameter	Description
type	use 0 for RMS, 1, for PTV.
data	use 0 for wavefront, 1 for spot radius, 2 for spot x, 3 for spot y, 4 for spot x + y.
reference	use 0 for centroid, 1 for chief, 2 for unreferenced.
method	use 1 for Gaussian quadrature, 2 for rectangular array.
rings	the number of annular rings (Gaussian quadrature only).
arms	the number of radial arms (Gaussian quadrature only). The number of arms must be even and no less than 6.

grid	the size of the grid. Use an integer, such as 8, for an 8 x 8 grid. n must be even and no less than 4.
delete	use 0 to not delete vignetted rays, 1 to delete vignetted rays.
axial	use -1 for automatic, which will use symmetry only if the system is axial symmetric. Use 1 to assume axial symmetry, 0 to not assume axial symmetry.
lateral	use 1 to ignore lateral color, 0 otherwise.
start	use -1 for automatic, which will add the default merit function after any existing DMFS operand. Otherwise use the operand number at which to add the default merit function. Any existing operands above the specified operand number will be retained.
xweight, oweight	the x direction weigh and overall weight for the merit function. Only the data "spot x + y" uses the xweight value.
pup_obsc	the pupil obscuration ratio.

If we want to delete an operand in the merit function editor, we can use keyword DELETEMFO to do so. The syntax is:

DELETEMFO row

or

DELETEMFO ALL

where row is the line number of the operand to be deleted, and it needs to be an integer expression larger than 0 and smaller than the total number of operands. If "All" is used, then all the operands will be deleted.

If we want to insert an operand in the merit function editor, we can use keyword INSERTMFO. The syntax is:

INSERTMFO row

where row is the line number of the operand to be inserted, and it needs to be an integer expression larger than 0 and smaller than the total number of operands. After the operation, original operands in the line with number row and higher will be shifted one number higher. The newly inserted operand is an empty operand. Its content needs to be defined using other keywords such as SETOPERAND.

If we need to set or update an operand in the merit function editor, we can use keyword SETOPERAND to do so. The syntax is:

SETOPERAND row, col, value

where row is the line number of the operand to be edited, col is the column number, and the value is for the position determined by row and col. The meaning of col depends on the operand. In general, 1 is for operand type, 2 for Int1, 3 for Int2, 4~7 for data1~data4, 8 for target, and 9 for weight. Figure 3.5-1 shows a dialog box popped out when directly setting operand in Zemax, where Operand is the type of the operand. To operand CNAX, Surf and Wave correspond to Int1 and Int2 mentioned above, Hx, Hy, Pol and Samp correspond to data1~data4 mentioned above.

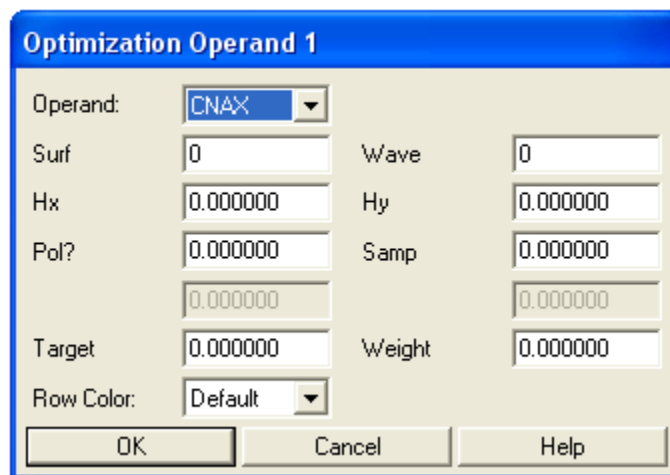


Fig. 3.5-1 Dialog box of operand setting

When setting operand type with col = 1, “value” should be an integer associated with the operand, such as 1 for “ACOS”, 2 for “ABS0”, 4 for “DENC”, 367 for “CNAX”, etc. The integer associated with each operand can be determined by the return value of function ONUM(A\$), where A\$ stands for the string of various operands.

Besides the method described above (col = 1), we can also use col = 11 to set the type of operand, and set “value” as the string that represents the type, such as:

```
SETOPERAND 1, 11, "CNAX"
```

or

```
A$ = "CNAX"
```

```
SETOPERAND 1, 11, A$
```

If we want to read the content of a certain operand in the merit function editor, we can use function OPER(row,col), where row is the row number in the editor, col is the column number in the editor with 1 for operand type, 2 for Int1, 3 for Int2, 4~7 for data1~data4, 8 for target value, 9 for weight, 10 for operand value, and 11 for the percentage contribution to the total merit function. Please note that function OPER() only reads the current content of the merit function, but will not change it.

If we want to calculate the total value of the merit function, we can use function MFCN(). This function will update the lens data, evaluate the validity of the merit function, calculate its value, and return the final result.

Now we will give some examples to show how to set and read parameters in the merit function editor.

Example 3.5-1: set and read default merit function. In this example, we assume the optical is the one defined in program ex30401. If we want to define a merit function to evaluate the image quality of this optical system, i.e. to evaluate the overall quality of the light beam on the image plane formed by rays coming from various field, a direct and most common way is to use the default merit function provided by Zemax. This example shows how to do it with ZPL.

```

1 ! ex30501
2 ! This program shows how to set default merit function
3 ! Assume the lens system is defined in ex30401
4
5 tp = 0 # type is RMS
6 data = 1 # spot radius
7 ref = 0 # centroid reference
8 method = 1 # Gaussian quadrature
9 ring = 3
10 arm = 6
11 grid = 4
12 delOrNot = 0 # not delete vignettted rays
13 ax = 1 # assume axial symmetry
14 lat = 1 # ignore lateral color
15 st = 1 # put the default merit function start at the beginning
16 ow = 1 # no xweight, overall weight = 1
17
18 DEFAULTMERIT tp,data,ref,method,ring,arm,grid,delOrNot,ax,lat,st,ow
19
20 PRINT
21 PRINT "The final merit function value is ", MFCN{}

```

After execution, the content of the merit function editor is shown in figure 3.5-2:

Oper #	Type							
1	DMFS	DMFS						
2	BLNK	BLNK	Default merit function: RMS spot radius centroid CQ 3 rings 6 arms Overall Wgt = 0.0000					
3	BLNK	BLNK	No default air thickness boundary constraints.					
4	BLNK	BLNK	No default glass thickness boundary constraints.					
5	BLNK	BLNK	Operands for field 1.					
6	PRIM	PRIM	1					
7	TRAC	TRAC	1	0.000000	0.000000	0.335711	0.000000	
8	TRAC	TRAC	1	0.000000	0.000000	0.707107	0.000000	
9	TRAC	TRAC	1	0.000000	0.000000	0.941965	0.000000	
10	PRIM	PRIM	2					
11	TRAC	TRAC	2	0.000000	0.000000	0.335711	0.000000	
12	TRAC	TRAC	2	0.000000	0.000000	0.707107	0.000000	
13	TRAC	TRAC	2	0.000000	0.000000	0.941965	0.000000	
14	PRIM	PRIM	3					
15	TRAC	TRAC	3	0.000000	0.000000	0.335711	0.000000	
16	TRAC	TRAC	3	0.000000	0.000000	0.707107	0.000000	
17	TRAC	TRAC	3	0.000000	0.000000	0.941965	0.000000	
18	BLNK	BLNK	Operands for field 2.					
19	PRIM	PRIM	1					
20	TRAC	TRAC	1	0.000000	0.700000	0.167855	0.290734	

Fig. 3.5-2 Merit function generated by program ex30501.ZPL

In the mean time, the total value of the merit function obtained by function MFCN() is displayed in the text message window, as shown in figure 3.5-3.

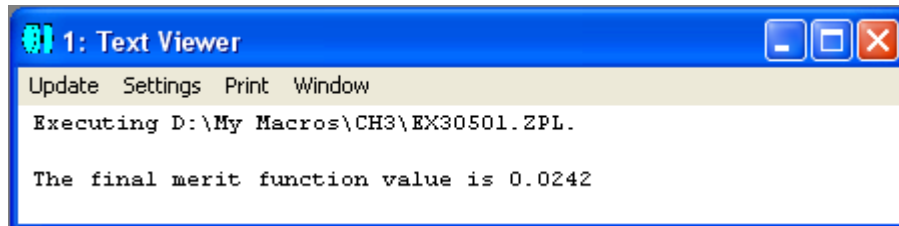


Fig. 3.5-3 The merit function resulted from program ex30501.ZPL

The powerful optimization function of Zemax relies on changing the value of different variable to minimize the value of the merit function. We will discuss how to optimize an optical system with ZPL program in latter sections.

Besides the default Zemax merit function discussed above, often times the user needs to define his own merit function to meet his special design target. We will show how to do this in the following example.

Example 3.5-2: Self-defined merit function

How to define the merit function depends on the design target of an optical system. In this example, we still assume that the optical system is the doublet defined in program ex30401. Further, we assume that due to the limitation of the detector size, the image height of view field 3 ($\theta_y = 5^\circ$) on the image plane needs to be 30 lens units. So we need to modify the default merit function and construct a user-defined merit function. To realize this, we can start from the default merit function defined in last example, and insert a new operand CENY with a target value as 30. The program is shown below:

```

1 ! ex30502
2 ! This program shows how to set user defined merit function
3 ! Assume the lens system is defined in ex30401
4 ! Also assume the default merit function was first created as in ex30501
5
6 INSERTMFO 1           # insert a blank row in the merit function editor
7 INSERTMFO 1           # insert another blank row
8 SETOPERAND 1, 11, "CENY" # set operand type
9 SETOPERAND 1, 2, 0     # set surface number, 0 for image surface
10 SETOPERAND 1, 3, 0    # set wavelength number, 0 for polychromatic
11 SETOPERAND 1, 4, 3    # set field number
12 SETOPERAND 1, 8, 30   # set target
13 SETOPERAND 1, 9, 1    # set weight
14
15 PRINT
16 PRINT "The final merit function value is ", MFCN()

```

In this example, we inserted two empty rows in the merit function editor, set the operand in the first row as CENY, and then set corresponding parameters such as surface number, wavelength number, field number, target, weight, etc., and finally output the value of the current merit function, as shown in figure 3.5-4.

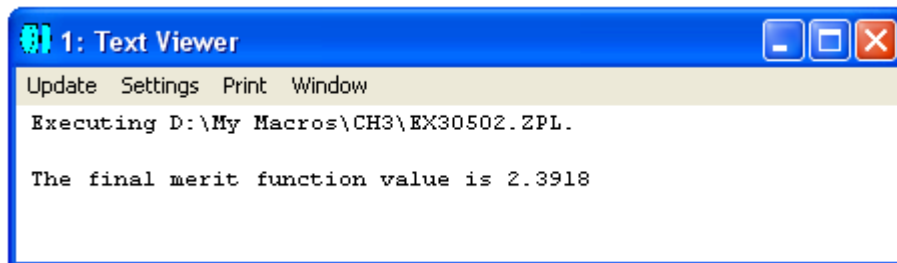


Fig. 3.5-4 The value of merit function after running program ex30502.ZPL

If we set surfaces of the doublet (surface 2, 3 and 4) and the distance of its back surface to the image plane as variables, we can easily optimize this optical system with the merit function we just defined. The value of the merit function after optimization can be 0.02. You can try it out if interested.

3.6 Solve

Zemax is very powerful on solving and optimizing existing optical systems. To allow user to access this feature in programs, ZPL provided many related keywords and functions. In this and next section, we will discuss how to handle solve and optimization in ZPL programs.

A commonly used keyword for setting and modifying arguments for solve is SOLVETYPE. The syntax is:

SOLVETYPE surf, CODE, arg1, arg2, arg3, arg4

In this command, surf is the surface number to be set, and the range of which should be between 0 and the maximum number of total surfaces. CODE is a mnemonic as listed in table 3.6-1. The arg1 - arg4 expressions evaluate to the first - fourth solve parameters. Please note that different solve type requires different number and types of arguments.

Table 3.6-1 arguments used in keyword SOLVETYPE

Solve Type	CODE	arg1	arg2	arg3	arg4
Curvature: Fixed (turn solve off)	CF				
Curvature: Variable	CV				
Curvature: Marginal Ray	CM	Angle			
Curvature: Chief Ray	CC	Angle			
Curvature: Pickup	CP	Surface #	Ratio		Column #
Curvature: Marginal Ray Normal	CN				
Curvature: Chief Ray Normal	CO				
Curvature: Aplanatic	CA				
Curvature: Element Power	CE	Power			
Curvature: Concentric With Surface	CQ	Concentric surface #			
Curvature: Concentric With Radius	CR	Concentric with			

		radius surface #			
Curvature: F/#	CG	Paraxial F/#			
Curvature: ZPL Macro	CZ	Macro name			
Thickness: Fixed (turn solve off)	TF				
Thickness: Variable	TV				
Thickness: Marginal Ray Height	TM	Height	Aperture area		
Thickness: Chief Ray Height	TC	Height			
Thickness: Edge Thickness	TE	Thickness	Radical height (0 for diameter)		
Thickness: Pickup	TP	Surface #	Ratio	Offset	Column #
Thickness: Optical Path Difference	TO	Optical Path Difference	Aperture area		
Thickness: Position	TL	Surface #	Self defined surface length		
Thickness: Compensator	TX	Surface #	Total surface thickness		
Thickness: Center Of Curvature	TY	Surface # at center of curvature			
Thickness: Pupil Position	TU				
Thickness: ZPL Macro	TZ	Macro name			
Glass: Fixed (turn solve off)	GF				
Glass: Model	GM	D line refractive	Abbe number Vd	Partial dispersion	

		index Nd		$\Delta P_{g,F}$	
Glass: Pickup	GP	Surface #			
Glass: Substitute	GS	Catalog name			
Glass: Offset	GO	D line refractive index Nd offset	Abbe number Vd offset		
Semi-Diameter: Automatic	SA				
Semi-Diameter: User Defined	SU				
Semi-Diameter: Pickup	SP	Surface #	Ratio		Column #
Semi-Diameter: Maximum	SM				
Semi-Diameter: ZPL Macro	SZ	Macro name			
Conic: Fixed (turn solve off)	KF				
Conic: Pickup	KP	Surface number	Ratio		Column #
Conic: ZPL Macro	KZ	Macro name			
Parameter: Fixed (turn solve off). Replace “p” with the parameter number in the code.	PF_p				
Parameter: Pickup. Replace “p” with the parameter number in the code.	PP_p	Surface #	Ratio	Offset	Column #
Parameter: Chief Ray. Replace “p” with the parameter number in the code.	PC_p	Field #	Wavelength #		
Parameter: ZPL Macro. Replace “p” with the	PZ_p	Macro			

parameter number in the code.		name			
Thermal Coefficient of Expansion: Fixed (turn solve off)	HF				
Thermal Coefficient of Expansion: Pickup	HP				
Extra Data Value: Fixed (turn solve off). Replace “e” with the extra data number in the code.	EF_e				
Extra Data Value: Pickup. Replace “e” with the extra data number in the code.	EP_e	Surface #	Ratio	Offset	Column #
Extra Data Value: ZPL Macro. Replace “e” with the extra data number in the code.	EZ_e	Macro name			
Non-Sequential Component Pickup X, Y, Z, Tilt-X, Tilt-Y, Tilt-Z, Material. Replace “o” with the object number in the code.	NSC_PX_o, NSC_PY_o, NSC_PZ_o, NSC_PTX_o, NSC_PTY_o, NSC_PTZ_o, NSC_PMAT_o				
Non-Sequential Component Material is fixed, model glass, pick up, or offset. Replace “o” with the object number in the code.	NSC_MATF_o, NSC_MATM_o, NSC_MATP_o, NSC_MOFF_o				
Non-Sequential Component ZPL Macro solve on X, Y, Z, Tilt-X, Tilt-Y, Tilt-Z. Replace “o” with the object number in the code.	NSC_ZX_o, NSC_ZY_o, NSC_ZZ_o, NSC_ZTX_o, NSC_ZTY_o, NSC_ZTZ_o				
Non-Sequential	NSC_PP_o_p				

Component Parameter Pickup. Replace “o” with the object number and “p” with the parameter number in the code.					
Non-Sequential Component ZPL Macro solve. Replace “o” with the object number and “p” with the parameter number in the code.	NSC_ZP_o_p				

* In this table, column # is defined below:

0: current column;

1~4: radius, thickness, conic constant, semi-diameter;

5~17: parameters 1~12;

18~259: extra data 1~242.

Among the solve types listed above, ZPL macro is used in many places. This allows user to call self-defined macros in solve. The only difference between this macro to other ZPL macros is that it requires keyword SOLVERETURN to return the result to the editor that calls the macro. The syntax is:

SOLVERETURN value

where value is the return value.

When more than one keyword SOLVERETURN is used in the program, only the last SOLVERETURN command will be executed, and all the other SOLVERETURN commands will be ignored. If there is error in the program, the SOLVERETURN command will not be executed, that means there are issues in the optical system.

We need to keep in mind that although solve with ZPL program is very flexible, it is also very easy to cause infinite loops or abnormal interruptions. Therefore, we need to be cautious when using solve in ZPL.

Besides using keyword to set and modify solve parameters, ZPL also provides a function SOLV() to read solve parameters. The syntax is:

$$\text{returnValue} = \text{SOLV}(\text{surf}, \text{code}, \text{param})$$

In this function, surf is the surface number. Code is 0 for curvature, 1 for thickness, 2 for glass, 3 for conic, 4 for semi-diameter, and 5 for thermal expansion coefficient TCE. For parameter data, the code is 100 plus the parameter number. For extra data, the code is 300 plus the extra data number. Param is an integer between 0 and 4, inclusive. The return value is data about the solve type for the specified surface and data. If param is zero, then an integer corresponding to the solve type is returned. For param between 1 and 3, the data is the solve parameters. For param 4, the data is the pickup column number. String values may be extracted using the \$buffer() function after calling this function with a code that returns string data.

We will give some examples below.

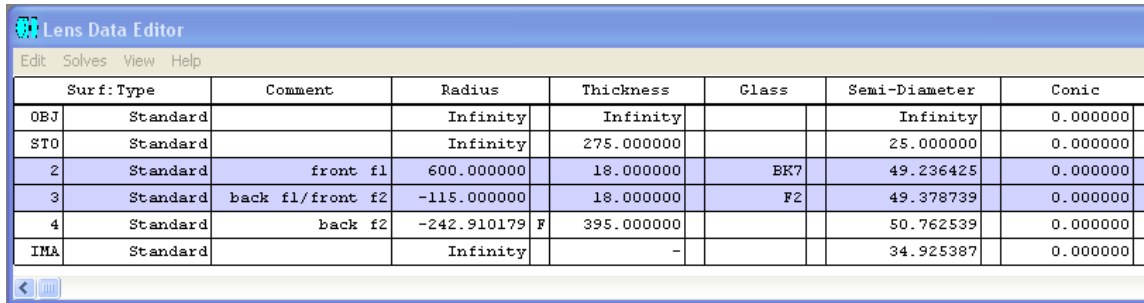
Example 3.6-1: set solve parameters. Assume we have constructed the doublet as discussed in example 3.4-1. In that system, the radius of surface 4 is -243, and the paraxial F/# is not quite equal to design target 8. We can change the curvature of surface 4 using solve, and force the paraxial F/# approximating 8 quickly. The program is shown below:

```

1 ! ex30601
2 ! This program shows how to set solve parameters
3 ! Assume the lens system is defined in ex30401
4
5 surf = 4
6 CODE$ = "CG" # solve type is "Curvature: F/#"
7 arg1 = 8 # F/# = 8
8
9 SOLVETYPE surf, CODE$, arg1 # only 1 argument is needed
10 UPDATE # UPDATE is needed to get the solve result

```

In this program, we set surface 4 as solve type "curvature: F/#", and let F/# = 8. After running the program, we can see that the radius of surface 4 in the lens data editor has been changed, and letter F appears at right side of the radius value, as shown in figure 3.6-1:



Surf	Type	Comment	Radius	Thickness	Class	Semi-Diameter	Conic
OBJ	Standard		Infinity	Infinity		Infinity	0.000000
STO	Standard		Infinity	275.000000		25.000000	0.000000
2	Standard	front f1	600.000000	18.000000	BK7	49.236425	0.000000
3	Standard	back f1/front f2	-115.000000	18.000000	F2	49.378739	0.000000
4	Standard	back f2	-242.910179	395.000000	F	50.762539	0.000000
IMA	Standard		Infinity	-		34.925387	0.000000

Fig. 3.6-1 Result of program ex30601.ZPL

Example 3.6-2: read solve parameters. Assume we need to read solve parameters from last example. We can use function SOLV() to do so, as shown below:

```

1 ! ex30602
2 ! This program shows how to read solve parameters
3 ! Assume the solve type is defined in ex30601
4
5 surf = 4
6 code = 0 # solve type is "Curvature"
7 param = 1 # want to know the first argument, i.e. F/#
8
9 returnValue = SOLV(surf, code, param)
10
11 PRINT
12 PRINT "The solve parameter is ", returnValue

```

Please note that code here is a numerical variable, whereas CODE\$ in last example is a string variable. The result is shown below:

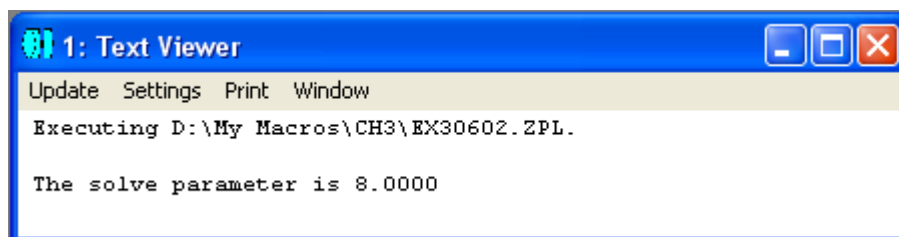


Fig. 3.6-2 Result of program ex30602.ZPL

It showed that the solve parameter we obtained in last example is $F/\# = 8$.

Example 3.6-3: Application of ZPL solve. In example 3.6-1, we assumed the refractive index of air is 1, and got radius of surface 4 as -242.910179. Now we want to use actual refractive index of air $n = 1.0008$ and relation $n_1C_1 = n_2C_2$ (n is refractive index, C is curvature) to fine tune the radius value to get a more realistic result. We can use ZPL solve to realize this. The parameter setting program and solve program is shown in ex30603.ZPL and ex30603M.ZPL, respectively:

```

1 ! ex30603
2 ! This program shows how to set ZPL macro solve
3 ! Assume the optical system is defined in ex30601
4
5 surf = 4
6 CODE$ = "C2" # solve type is "Curvature: ZPL macro"
7 arg1$ = "\ch3\ex30603M.zpl" # macro name
8
9 SOLVETYPE surf, CODE$, arg1$ # only 1 argument is needed
10 UPDATE # UPDATE is needed to get the solve result

```

```

1 ! ex30603M
2 ! This program shows an example of ZPL macro
3 ! It will be called by program "ex30603.zpl"
4
5 nVacuum = 1
6 nAir = 1.0008
7
8 surf = 4
9 culvatureVacuum = CURV(surf) # get the curvature of surface 4
10
11 culvatureAir = culvatureVacuum*nVacuum/nAir # modify the culvature
12
13 temp = 1/123.456 # create a number
14 SOLVEReturn temp # try to add more than one SOLVEReturn
15
16 SOLVEReturn culvatureAir # return the culvature, not the radius!!

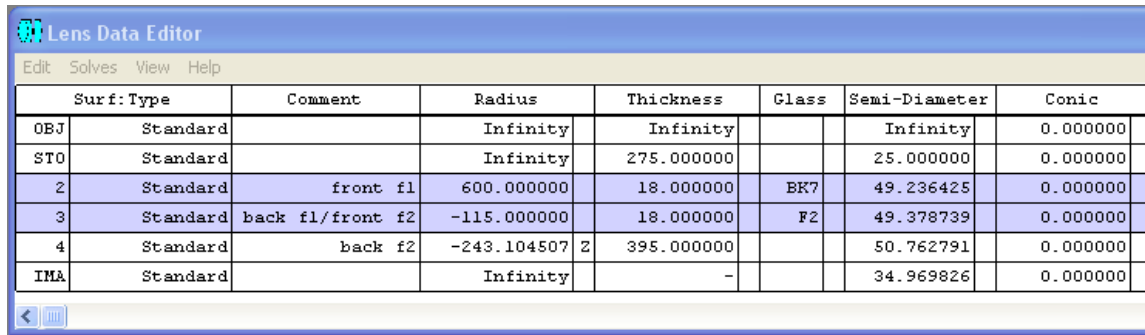
```

Please note that in line 7 of program ex30603, we added a path name “\ch3\” in front of macro name “ex30603M.zpl”. It is because when running program ex30603.zpl, Zemax only searches solve macro under the macro folder. Any further directory structure in the macro folder needs to be specifically called out.

We also need to note that the calculated value in program ex30603M is curvature. After returning this curvature, Zemax will automatically calculate radius value.

Also, line 13 and 14 don't have any real meaning, and can be deleted from the program. We add them in the program to demonstrate that if there are more than one keyword SOLVERRETURN in the program, only the last SOLVERRETURN is valid.

The result of program ex30603 is shown below:



Surf	Type	Comment	Radius	Thickness	Class	Semi-Diameter	Conic
OBJ	Standard		Infinity	Infinity		Infinity	0.000000
ST0	Standard		Infinity	275.000000		25.000000	0.000000
2	Standard	front f1	600.000000	18.000000	BK7	49.236425	0.000000
3	Standard	back f1/front f2	-115.000000	18.000000	F2	49.378739	0.000000
4	Standard	back f2	-243.104507	395.000000	Z	50.762791	0.000000
IMA	Standard		Infinity	-		34.969826	0.000000

Fig. 3.6-3 Result of ex30603.ZPL

We can see that the new radius value of surface 4 is -243.104507, and letter Z appeared at the right side of this value, indicating that the solve type is ZPL macro.

Similar to automatic solve, ZPL also provides a keyword QUICKFOCUS to quickly adjust the focus of the optical system. The syntax is:

QUICKFOCUS mode, centroid

where mode is 0, 1, 2 or 3 for root mean square (RMS) beam radius, beam x value, beam y value and wave front light path difference, centroid is 0 or 1 for RMS relative to chief ray or image center.

3.7 Optimization

Optimization gives Zemax its great power in optical design. In this section we will discuss how to use optimization commands in ZPL. For general optimization in Zemax, please refer to Zemax User's Manual.

We all know that optimizing an optical system in Zemax is essentially setting certain lens parameters as variables and allowing Zemax to automatically change the value of the variables to minimize the merit function. To do this in ZPL, a keyword SETVAR is provided to set and change variables for optimization. The syntax is:

SETVAR surf, code, status, objectNum

or

SETVAR config, M, status, operand

where surf is the lens surface order number, config is the configuration order number in the multi-configuration editor, code is one of the strings listed in table 3.7-1. If the value of status is 0, then the variable status is removed, otherwise the value associated with code is made variable. If the code is Nn or On, the object number must be provided, otherwise it should be omitted. If the code is M, then the syntax for this command is as shown above as the multi-configuration one, and the operand needs to be provided.

Table 3.7-1: Code associated to keyword SETVAR

Code	Description
R	radius of curvature
T	thickness
C	conic
G	glass
I	glass index
J	glass Abbe
K	glass dpgf
Pn	parameter n
D	thermal coefficient of expansion
En	extra data value n
M	multi-configuration data
Nn	non-sequential component position data, 1-6 for x, y, z, tx, ty, tz
On	non-sequential component parameter data, where n is the parameter number

If we want to remove all the variables, we can use keyword REMOVEVARIABLES to change all the variables as fixed value.

If we want to read a variable we set, we can use keyword GETVARDATA. The syntax is:

GETVARDATA vector

where vector = 1~4 is one of the 4 vector array variables provided by ZPL (either VEC1, VEC2, VEC3 or VEC4). The data is stored in the specified VECn array variable in the format described in table 3.7-2.

Table 3.7-2: Storage format of data obtained with keyword GETVARDATA

Array Position	Description
0	n, the number of variables
1	The type code for the first variable
2	Surface number for the first variable
3	Parameter number for the first variable
4	Object number for the first variable
5	The value of the first variable
5*q-4	The type code for the qth variable
5*q-3	Surface number for the qth variable
5*q-2	Parameter number for the qth variable
5*q-1	Object number for the qth variable
5*q	The value of the qth variable

The type code for variables is as described in the following table. The surface number, parameter number, and object number may or may not have meaning depending upon the type of variable.

Table 3.7-3: GETVARDATA type and ID codes

Variable type	Type Code	Surface	Parameter	Object
Curvature	1	surface #	-	-
Thickness	2	surface #	-	-
Conic	3	surface #	-	-
Index Nd	4	surface #	-	-
Abbe Vd	5	surface #	-	-
Partial Dispersion $\Delta P_{g \square \square F}$	6	surface #	-	-
TCE	7	surface #	-	-
Parameter Values	8	surface #	parameter #	-
Extra Data Values	9	surface #	extra data #	-
Multi-configuration Operand Values	10	oper #	config #	-
Non-Sequential Object Position X	11	surface #	-	object #
Non-Sequential Object Position Y	12	surface #	-	object #
Non-Sequential Object Position Z	13	surface #	-	object #
Non-Sequential Object Tilt X	14	surface #	-	object #
Non-Sequential Object Tilt Y	15	surface #	-	object #
Non-Sequential Object Tilt Z	16	surface #	-	object #
Non-Sequential Object Parameters	17	surface #	parameter #	object #

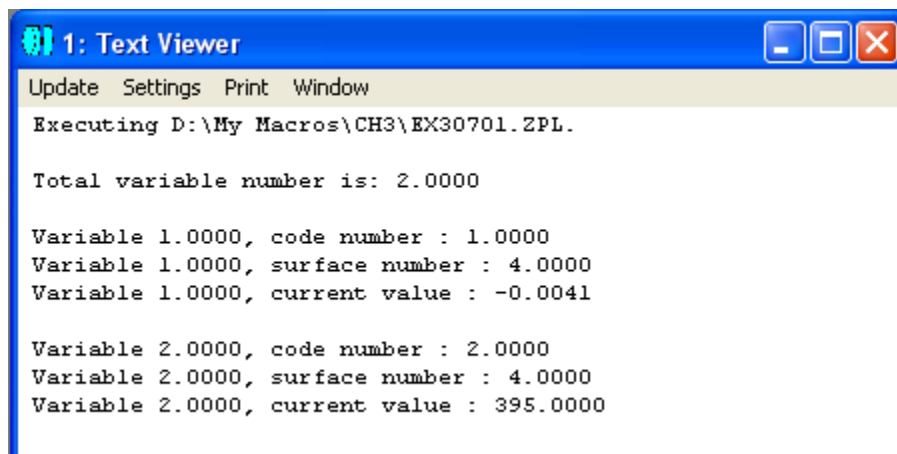
Example 3.7-1 gives an example of setting and reading variables in ZPL. In this example, we assume the optical system is the doublet defined in ex30401.ZPL. We will set the curvature radius and thickness of surface 4 as variables, and read back the value of the variables. The program is shown below:

```

1 ! ex30701
2 ! This program shows how to set and read variables for optimization
3 ! Assume the lens system is defined in ex30401
4
5 surf = 4
6 CODE1$ = "R" # Radius
7 CODE2$ = "T" # Thickness
8 status = 1 # set variable
9
10 SETVAR surf, CODE1$, status # variable 1, only 3 arguments
11 SETVAR surf, CODE2$, status # variable 2, only 3 arguments
12
13 GETVARDATA 1 # read variable data and put in VEC1
14 totalVarNum = VEC1(0)
15
16 PRINT
17 PRINT "Total variable number is: ", totalVarNum
18
19 for q = 1, totalVarNum, 1
20 PRINT
21 PRINT "Variable ", q, ", code number : ", VEC1(5*q-4)
22 PRINT "Variable ", q, ", surface number : ", VEC1(5*q-3)
23 PRINT "Variable ", q, ", current value : ", VEC1(5*q)
24 NEXT

```

After running the program, we can see that, in the lens data editor, letter “V” appears after the values of radius and thickness of surface 4. It means that the two parameters are set as variables. Also, we can see the read out values of the variables in the text window, as shown in figure 3.7-1.



```

1: Text Viewer
Update Settings Print Window
Executing D:\My Macros\CH3\EX30701.ZPL.

Total variable number is: 2.0000

Variable 1.0000, code number : 1.0000
Variable 1.0000, surface number : 4.0000
Variable 1.0000, current value : -0.0041

Variable 2.0000, code number : 2.0000
Variable 2.0000, surface number : 4.0000
Variable 2.0000, current value : 395.0000

```

Fig. 3.7-1 Result of program ex30701.ZPL

We notice that the value of variable 1 is the curvature of surface 4, which is a reciprocal of the radius value shown in the lens data editor.

After we set the variable and merit function, it's straightforward to do the optimization using keywords OPTIMIZE and HAMMER provided by ZPL. The syntax is:

OPTIMIZE

OPTIMIZE number_of_cycles

OPTIMIZE number_of_cycles, algorithm

and

HAMMER

HAMMER number_of_cycles

HAMMER number_of_cycles, algorithm

The argument *number_of_cycles* is an integer between 1 and 99 for the number of cycles the optimization algorithm will run. For the Optimize command, if *number_of_cycles* evaluates to zero, or there is no argument, then the optimization runs in "Automatic" mode, stopping when the algorithm detects the process has converged. For the Hammer command, if there is no argument provided, then the Hammer optimization runs 1 cycle using Damped Least Squares. For the algorithm argument, use 0 for Damped Least Squares (the default) and 1 for Orthogonal Descent.

Sometimes if we want to directly calculate the values of certain operand variables without putting them in the merit function, we can use functions OPEV or OPEW provided by ZPL. The two functions are similar, but have different arguments. The syntax is:

OPEV(code, int1, int2, data1, data2, data3, data4)

and

OPEW(code, int1, int2, data1, data2, data3, data4, data5, data6)

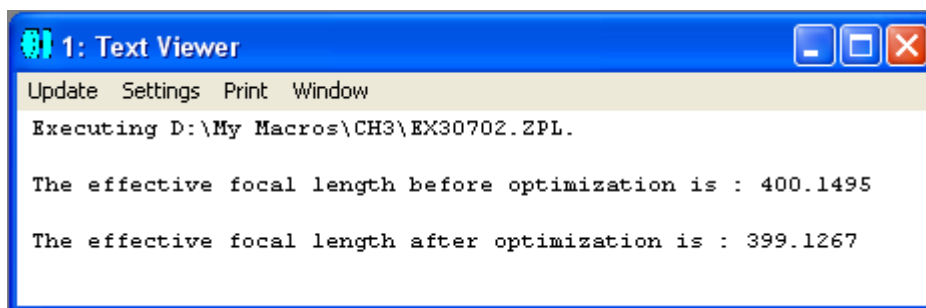
where *code* is the optimization operand code, *int1*~*int2* and *data1*-*data4* are the defining values for the operand. In general, the operand code is returned by function OCOD(A\$), where A\$ is the string associated to the optimization operand, such as "EFFL", etc.

Example 3.7-2 gives an example of optimization in ZPL. In this example, we assume the optical system is the doublet defined in program ex30401.ZPL, the variables are set as in program ex30701.ZPL, and the

merit function is set as in program ex30501.ZPL. Our target is to optimize this optical system, and calculate the effective focal length of the system. The program is shown below:

```
1 ! ex30702
2 ! This program shows how to do optimization and calculate operand
3 ! Assume the lens system is defined in ex30401
4 ! Assume the variables are defined in ex30701
5 ! Assume the merit function is defined in ex30501
6
7 code = OCOD("EFFL") # get the code of operand "EFFL"
8 wavelengthNum = 2
9 efflValue = OPEV(code, 0, wavelengthNum, 0, 0, 0, 0) # only int2 is needed
10
11 PRINT
12 PRINT "The effective focal length before optimization is : ", efflValue
13
14 OPTIMIZE # use automatic optimization, default algorithm
15
16 efflValue = OPEV(code, 0, wavelengthNum, 0, 0, 0, 0) # only int2 is needed
17
18 PRINT
19 PRINT "The effective focal length after optimization is : ", efflValue
20
```

In the program, we use function OCOD() to read out the code of the operand for effective focal length "EFFL", and directly calculate the effective focal length. We also use keyword OPTIMIZE to optimize the system, and compare the values of effective focal length before and after optimization. The result is shown below:



```
1: Text Viewer
Update Settings Print Window
Executing D:\My Macros\CH3\EX30702.ZPL.

The effective focal length before optimization is : 400.1495

The effective focal length after optimization is : 399.1267
```

Fig. 3.7-2 Result of program ex30702.ZPL

ZPL also provided a keyword similar to automatic optimization: TESTPLATEFIT. This keyword can be used in optical design to call the test plate fitting routine to fit the test plate library provided by lens vendors. The syntax is:

TESTPLATEFIT tpd_file, log_file, method, number_cycles

where *tpd_file* is the test plate data file, *log_file* is the name of a file for the output log, *method* is an integer between 0 and 4, inclusive, for try all methods, best to worst, worst to best, long to short, and short to long, respectively. The integer *number_cycles* is 0 for automatic or the maximum number of optimization cycles to execute. Note the *tpd_file* name should NOT include the path, since all test plate files are in a fixed folder, while the path should be included for the log file.

3.8 Ray Tracing

Most of Zemax calculations are based on ray tracing. Therefore, ray tracing is a key function in Zemax. ZPL provided two keywords RAYTRACE and RAYTRACEX to support ray tracing in sequential systems. As for ray tracing in non-sequential systems, we will discuss in section 10.

Keyword RAYTRACE calls the Zemax ray tracing routines to trace a particular ray through the current system. The syntax is:

RAYTRACE hx, hy, px, py, wavelength

where hx and hy are normalized field coordinates with values between -1 and +1; px and py are normalized pupil coordinates with values between -1 and +1; wavelength is optional working wavelength, defaulting to the primary wavelength.

Keyword RAYTRACEX calls the Zemax ray tracing routines to trace a particular ray from any starting surface through the current system. The syntax is:

RAYTRACEX x, y, z, l, m, n, surf, wavelength

where x, y, z are the input ray position in the local coordinates of the starting surface, l, m, n are direction cosines in the local coordinates of the starting surface, surf is an integer between 0 and the number of surfaces minus one, inclusive, and wavelength is optional working wavelength, defaulting to the primary wavelength.

After ray tracing using keyword RAYTRACE or RAYTRACEX, the result can be read by various functions, as shown in table 3.8-1:

Table 3.8-1: functions used to read back ray tracing result

Functions	Description
RAYX(n), RAYY(n), RAYZ(n)	The x-coordinate, y-coordinate, and z-coordinate of the ray intercept. n is the surface number.
RAGX(n), RAGY(n), RAGZ(n)	The global x, y and z coordinate of the ray intercept. n is the surface number.
RAYL(n), RAYM(n), RAYN(n)	The x-direction cosine, y-direction cosine, and z-direction cosine of the ray following the surface. n is the surface number.
RANX(n), RANY(n), RANZ(n)	The x-direction cosine, y-direction cosine, and z-direction cosine of the surface normal. n is the surface number.
RAYT(n)	The ray path length from the previous surface to the specified

	surface. The path length may be negative. n is the surface number.
RAYO(n)	The ray optical path length from the previous surface to the specified surface. The optical path length is the path length times the index of refraction, either or both of which may be negative. For rays inside a non-sequential surface, RAYO returns the sum of the path length times the index of refraction of all objects the ray passed through. n is the surface number.
RAYV()	0 if ray was not vignetted, else vignetted surface number.
RAYE()	The ray-trace error flag, 0 if no error.

Additionally, ZPL provided the following keywords related to ray tracing: SCATTER, SETAIM, and SETAIMDATA.

SCATTER is used to control whether sequential surface scattering is done while tracing rays. The syntax is:

SCATTER ON

or

SCATTER OFF

The default condition at the start of a ZPL program is SCATTER OFF, and all rays will be traced deterministically. If SCATTER ON is executed, sequential surface scattering will be enabled for all subsequent RAYTRACE commands.

SETAIM is used to set the state of the ray aiming function. The syntax is:

SETAIM state

where state is 0 for ray aiming off and 1 for ray aiming on.

SETAIMDATA is used to set various data for the ray aiming function. The syntax is:

SETAIMDATA code, value

where code and value are used according to table 3.8-2.

Table 3.8-2: code and value for keyword SETAIMDATA

Code	Property
1	Sets "Use Ray Aiming Cache" to true if value is 1, or false if value is 0.
2	Sets "Robust Ray Aiming" to true if value is 1, or false if value is 0.
3	Sets "Scale Pupil Shift Factors by Field" to true if value is 1, or false if value is 0.
4, 5, 6	Sets the value of the x, y, and z pupil shift, respectively.
7, 8	Sets the value of the x and y pupil compress, respectively.

We will give an example of ray tracing in ZPL program. In this example, we assume the optical system is the doublet defined in program ex30401.ZPL. We will trace a marginal ray, a chief ray and an arbitrary ray, and read back the interception information of the rays and different surfaces. The program is shown below:

```

1 ! ex30801
2 ! This program shows how to do ray tracing
3 ! Assume the lens system is defined in ex30401
4
5 ! define the marginal ray
6 hx = 0
7 hy = 0
8 px = 0
9 py = 1
10 wavelength = 2
11
12 RAYTRACE hx, hy, px, py, wavelength # trace the marginal ray
13
14 PRINT
15 FOR n = 3, NSUR(), 1 # go through the last 3 surfaces
16   PRINT "Marginal ray, Surface ", n
17   PRINT " cross position: ", RAYX(n), ", ", RAYY(n), ", ", RAYZ(n)
18   PRINT " ray direction: ", RAYL(n), ", ", RAYM(n), ", ", RAYN(n)
19   PRINT " surface normal: ", RANX(n), ", ", RANY(n), ", ", RANZ(n)
20 NEXT n
21
22 ! define the chief ray
23 hx = 0
24 hy = 1
25 px = 0
26 py = 0
27
28 RAYTRACE hx, hy, px, py # trace the chief ray, use primary wavelength
29
30 PRINT
31 FOR n = 4, NSUR(), 1 # go through the last 2 surfaces
32   PRINT "Chief ray, Surface ", n
33   PRINT " cross position: ", RAGX(n), ", ", RAGY(n), ", ", RAGZ(n)
34   PRINT " ray direction: ", RAYL(n), ", ", RAYM(n), ", ", RAYN(n)
35 NEXT n
36

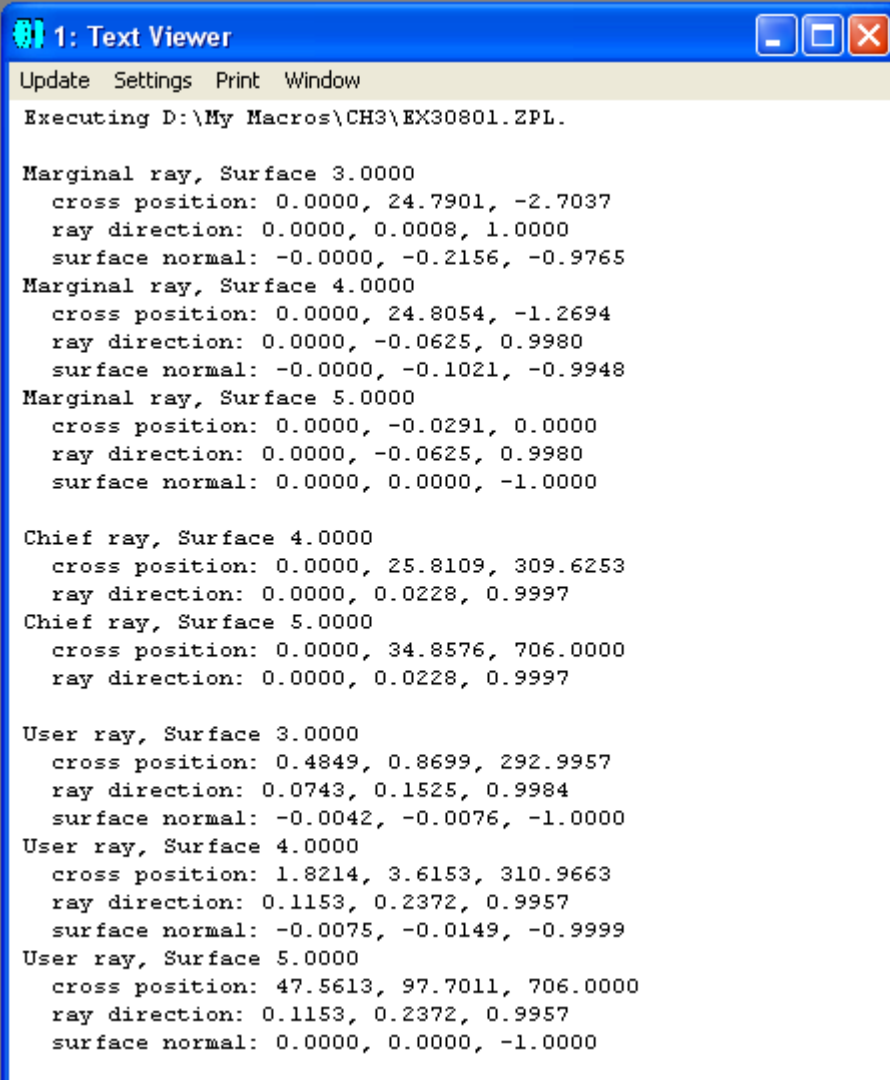
```

```

37 ! define an arbitrary ray on surface 2
38 x = 0.2
39 y = 0.3
40 z = -0.5
41 l = 0.1
42 m = 0.2
43 z = 0.9
44 surf = 2
45 wavelength = 3
46
47 RAYTRACEX x, y, z, l, m, n, surf, wavelength
48 PRINT
49 FOR n = 3, 5, 1 # go through surfaces 3, 4 and 5
50   PRINT "User ray, Surface ", n
51   PRINT " cross position: ", RAGX(n), ", ", RAGY(n), ", ", RAGZ(n)
52   PRINT " ray direction: ", RAYL(n), ", ", RAYM(n), ", ", RAYN(n)
53   PRINT " surface normal: ", RANX(n), ", ", RANY(n), ", ", RANZ(n)
54 NEXT n

```

In this program, we defined a marginal ray, a chief ray, and an arbitrary ray, traced the rays, and read back ray tracing result with various functions. Please note that when we read the interception position of rays and surfaces, for marginal ray, the coordinates we read are local ones relative to the surface apex (line 17 in the program), and for chief ray, the coordinates are global ones (line 33 in the program). The result of the program is shown below:



```
1: Text Viewer
Update Settings Print Window
Executing D:\My Macros\CH3\EX30801.ZPL.

Marginal ray, Surface 3.0000
cross position: 0.0000, 24.7901, -2.7037
ray direction: 0.0000, 0.0008, 1.0000
surface normal: -0.0000, -0.2156, -0.9765
Marginal ray, Surface 4.0000
cross position: 0.0000, 24.8054, -1.2694
ray direction: 0.0000, -0.0625, 0.9980
surface normal: -0.0000, -0.1021, -0.9948
Marginal ray, Surface 5.0000
cross position: 0.0000, -0.0291, 0.0000
ray direction: 0.0000, -0.0625, 0.9980
surface normal: 0.0000, 0.0000, -1.0000

Chief ray, Surface 4.0000
cross position: 0.0000, 25.8109, 309.6253
ray direction: 0.0000, 0.0228, 0.9997
Chief ray, Surface 5.0000
cross position: 0.0000, 34.8576, 706.0000
ray direction: 0.0000, 0.0228, 0.9997

User ray, Surface 3.0000
cross position: 0.4849, 0.8699, 292.9957
ray direction: 0.0743, 0.1525, 0.9984
surface normal: -0.0042, -0.0076, -1.0000
User ray, Surface 4.0000
cross position: 1.8214, 3.6153, 310.9663
ray direction: 0.1153, 0.2372, 0.9957
surface normal: -0.0075, -0.0149, -0.9999
User ray, Surface 5.0000
cross position: 47.5613, 97.7011, 706.0000
ray direction: 0.1153, 0.2372, 0.9957
surface normal: 0.0000, 0.0000, -1.0000
```

Fig. 3.8-1 Result of program ex30801.ZPL

Besides the ray tracing commands mentioned above, ZPL also provided two keywords POLDEFINE and POLTRACE specifically for polarized light ray tracing. POLDEFINE is used to set initial polarization state, and POLTRACE is used to do polarization ray tracing. The syntax for POLDEFINE is:

POLDEFINE Jx, Jy, PhaX, PhaY

where Jx and Jy are Jones electric field magnitudes, PhaX and PhaY are phases in degrees. The input values are automatically normalized to have unity magnitude. The default values are 0, 1, 0, and 0, respectively. Once the polarization state is defined, it remains the same until changed.

The syntax for POLTRACE is:

POLTRACE Hx, Hy, Px, Py, wavelength, vec, surf

where Hx and Hy are normalized object coordinates with values between -1 and 1; Px and Py are pupil coordinates with values also between -1 and 1; wavelength is the wavelength order number between 1 and the the maximum number of defined wavelengths; vec is an integer number between 1 and 4 for the 4 default ZPL vector arrays; surf is the surface number between 1 and the number of total surfaces, inclusive. The input polarization state of the ray is defined by the POLDEFINE keyword.

Once the ray is traced, the polarization data for the ray is placed in the vector variable specified by the vec expression. The data storage format is shown in table 3.8-3:

Table 3.8-3: format of polarization ray tracing result storage

Array Position	Description
0	n, the number of data entries in the vector. 0 means there is an error.
1	The ray intensity after the surface
2	E-Field X component, real
3	E-Field Y component, real
4	E-Field Z component, real
5	E-Field X component, imaginary
6	E-Field Y component, imaginary
7	E-Field Z component, imaginary
8	S-Polarization field amplitude reflection, real
9	S-Polarization field amplitude reflection, imaginary
10	S-Polarization field amplitude transmission, real
11	S-Polarization field amplitude transmission, imaginary
12	P-Polarization field amplitude reflection, real
13	P-Polarization field amplitude reflection, imaginary
14	P-Polarization field amplitude transmission, real

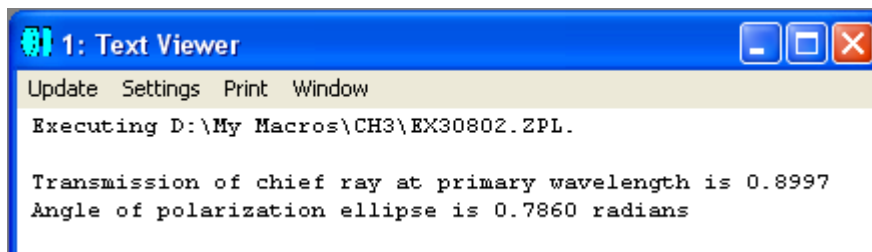
15	P-Polarization field amplitude transmission, imaginary
16	E-Field X direction phase Px
17	E-Field Y direction phase Py
18	E-Field Z direction phase Pz
19	Major axis length of polarization ellipse
20	Minor axis length of polarization ellipse
21	Angle of polarization ellipse in radians
22	The surface number at which the ray was vignetted or zero if not vignetted
23	S-Polarization ray amplitude reflection, real
24	S-Polarization ray amplitude reflection, imaginary
25	S-Polarization ray amplitude transmission, real
26	S-Polarization ray amplitude transmission, imaginary
27	P-Polarization ray amplitude reflection, real
28	P-Polarization ray amplitude reflection, imaginary
29	P-Polarization ray amplitude transmission, real
30	P-Polarization ray amplitude transmission, imaginary

We now give an example to discuss the usage of POLDEFINE and POLTRACE in ZPL.

Example 3.8-2: Polarization ray tracing

```
1 ! ex30802
2 ! This program shows how to do polarization ray tracing
3 ! Assume the lens system is defined in ex30401
4
5 ! define the initial polarization
6 Jx = 1
7 Jy = 1
8 PhaiX = 0
9 PhaiY = 0
10 POLDEFINE Jx, Jy, PhaiX, PhaiY
11
12 ! define the chief ray
13 hx = 0
14 hy = 1
15 px = 0
16 py = 0
17
18 ! choose the vector number
19 vec = 1
20
21 ! ray tracing
22 POLTRACE hx, hy, px, py, pwav(), vec, nsur()
23
24 PRINT
25 PRINT "Transmission of chief ray at primary wavelength is ", vec1(1)
26 PRINT "Angle of polarization ellipse is ", vec1(21), " radians"
```

In this example, we assume the optical is the doublet defined in ex30401.ZPL. We first set the initial polarization state (lines 6~10) and defined a chief ray (lines 13~16), selected default vector vec1 for data storage, and then started polarization ray tracing. The result is shown below:



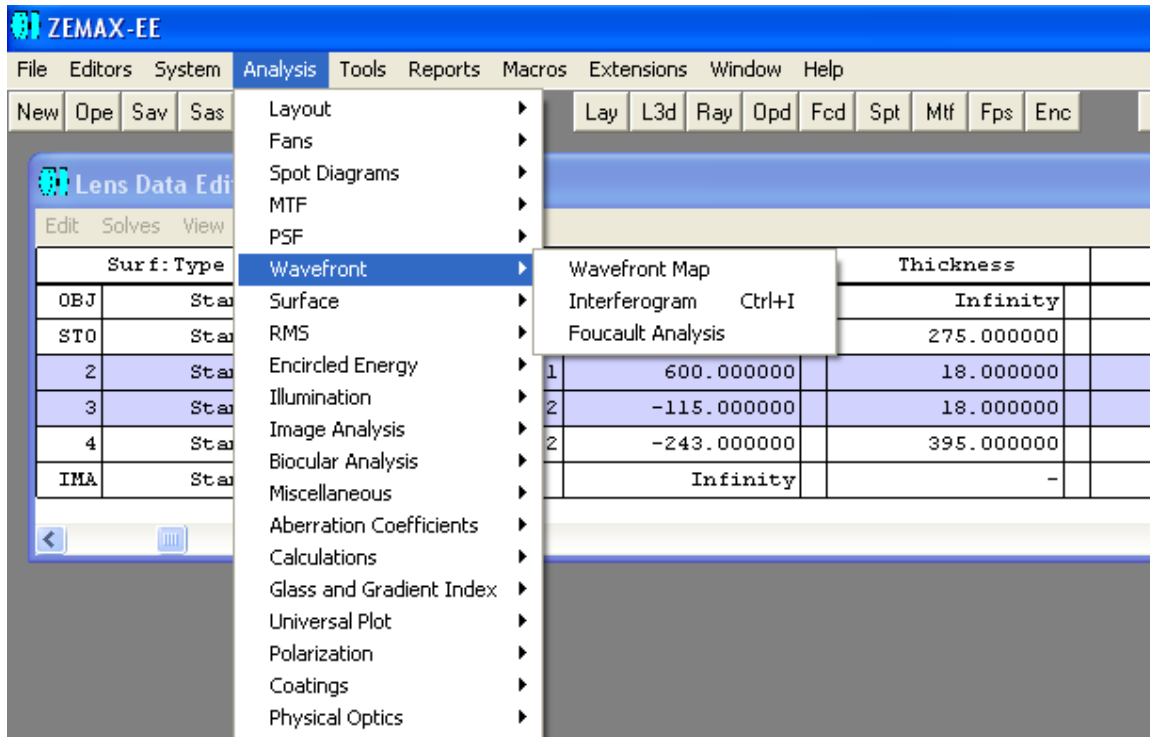
```
1: Text Viewer
Update Settings Print Window
Executing D:\My Macros\CH3\EX30802.ZPL.

Transmission of chief ray at primary wavelength is 0.8997
Angle of polarization ellipse is 0.7860 radians
```

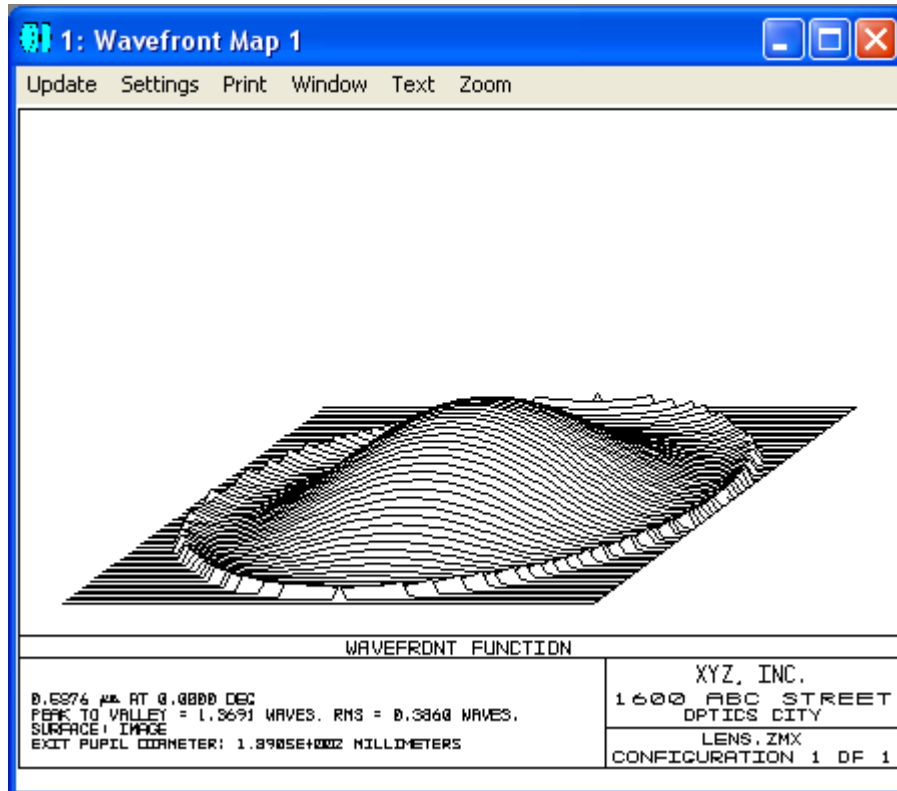
Fig. 3.8-2 Result of program ex30802.ZPL

3.9 System Analysis

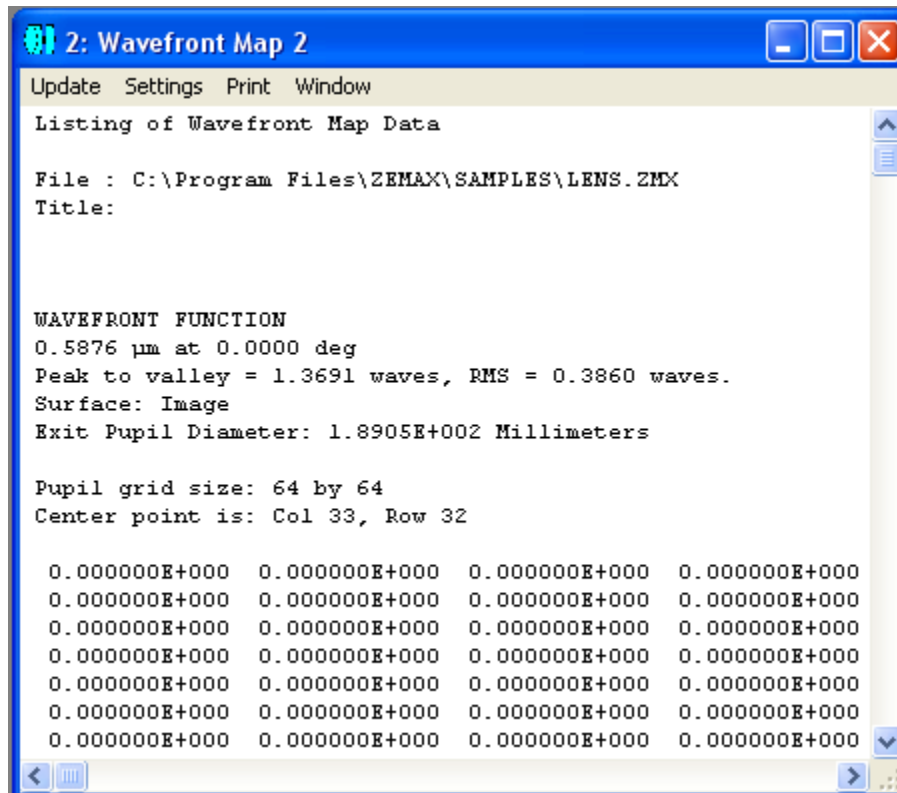
Zemax provided a lot of analysis tools to evaluate the performance of an optical system, with many of them providing text output. For example, the menu option Wavefront Map in figure 3.9-1(a) can display the wavefront map at a given surface, as shown in figure 3.9-1(b), as well as text information of the wavefront map shown in figure 3.9-1(c). In this example we assume the optical system is the doublet defined in example ex30401.ZPL.



(a) ZEMAX analysis tool menu option



(b) graphical output of the analysis result



(c) text output of the analysis result

Fig. 3.9-1: ZEMAX analysis options and different output of the result

For the text output, ZPL provided a keyword GETTEXTFILE to read related information and store the result in a text file. The syntax is:

GETTEXTFILE textfilename\$, type, settingsfilename\$, flag

The textfilename argument is a string for the target file name, including the full path and extension of the file name. The string function \$TEMPFILENAME can be used to define a suitable temporary file name. The type argument is a 3 character string code that indicates the type of analysis to be performed, as shown in table 3.9-1. The string codes are identical to those used for the button bar in Zemax. A list of string codes may be found on the “Buttons” tab of the File, Preferences dialog box. If no type is provided or recognized, a standard ray trace will be generated.

The settingsfilename\$ argument is a string for using or saving the settings, depending on the value of the flag parameter. If the flag value is 0, then the default settings will be used. If the lens file has its own default settings, then those will be used; these are the settings stored in the “lensfilename.cfg” file. If no

lens specific default settings exist, then the default settings for all Zemax files, stored in the file “Zemax.CFG” will be used, if any. If no previous settings have been saved for this or any other lens, then the default settings used are the “factory” defaults used by Zemax. If the flag value is 1, then the settings provided in the settings file, if valid, will be used to generate the file. If the data in the settings file is in anyway invalid, then the default settings will be used to generate the file. The only valid settings files are those generated by Zemax, then renamed and saved to a new user defined file name. If the flag value is 2, then the settings provided in the settings file, if valid, will be used and the settings box for the requested feature will be displayed. After the user makes any changes to the settings the file will then be generated using the new settings. The dialog boxes used to change the analysis settings use the data from the lens currently in the Lens Data Editor.

No matter what the flag value is, if a valid file name is provided for the settingsfilename, the settings used will be written to the settings file, overwriting any data in the file. To modify the settings defined within an existing settings file, use keyword MODIFYSETTINGS.

Please note that only text, and not graphic files, are supported by GETTEXTFILE.

Table 3.9-1: type code used in keyword GETTEXTFILE

Code	Description
Bfv	Beam File Viewer
Caa	Coating Abs. vs Angle
Car	Cardinal Points
Caw	Coating Abs. vs Wavelength
Cda	Coating Diattenuation vs Angle
Cdw	Coating Diatten. vs Wavelength
Cfs	Chromatic Focal Shift
Chk	System Check
ClS	Coating List
Cna	Coating Retardation vs Angle
Cnw	Coating Retardation vs Waveleng
Con	Conugate Surface Analysis
Cpa	Coating Phase vs Angle
Cpw	Coating Phase vs Wavelength

Cra	Coating Refl. vs Angle
Crw	Coating Refl. vs Wavelength
Cta	Coating Tran. vs Angle
Ctw	Coating Tran. vs Wavelength
Dim	Diffraction Image Analysis
Dip	Dipvergence/Convergence Data
Dis	Dispersion Diagram
Enc	Diff Encircled Energy
Fcd	Field Curvature/ Distortion
Fcl	Fiber Coupling
Fmm	FFT MTF Map
Foa	Foucault Analysis
Foo	Footprint Analysis
Fps	FFT PSF
Gbp	Paraxial Gaussian Beam
Gbs	Skew Gaussian Beam
Gee	Geom Encircled Energy
Gip	Grin Profile
Gmm	Geometric MTF map
Gmp	Glass Map
Gpr	Gradium Profile
Grd	Grid Distortion
Gtf	Geometric MTF
Gvf	Geometric MTF vs Field
Hcs	Huygens PSF Cross Section
Hmf	Huygens MTF
Hps	Huygens PSF
Hsm	Huygens Surface MTF

Htf	Huygens Through Focus MTF
Ibm	Geometric Bitmap Ima. Analysis
Ilf	Illumination Surface
Ils	Illumination XY Scan
Ima	Geometric Energy Analysis
Imv	IMA/BIM File Viewer
Int	Interferogram
Lat	Lateral Color
Lin	Geom Line/Edge Spread
Lon	Longitudinal Aberration
Lsf	FFT Line/Edge Spread
Mfl	Merit Function List
Mtf	FFT MTF
Mth	FFT MTF vs field
Opd	Opd Fan
Pab	Pupil Aberration Fan
Pal	Power Field Map
Pcs	FFT PSF cross section
Pha	Polarization Phase Abberation
Pmp	Polarization Pupil Map
Pol	Polarization Ray Trace
Pop	Physical Optics Propagation
Ppm	Power Pupil Map
Pre	Prescription Data
Ptf	Polarization Transmission Fan
Ray	Ray Fan
Rel	Relative Illumination
Rfm	RMS Field Map

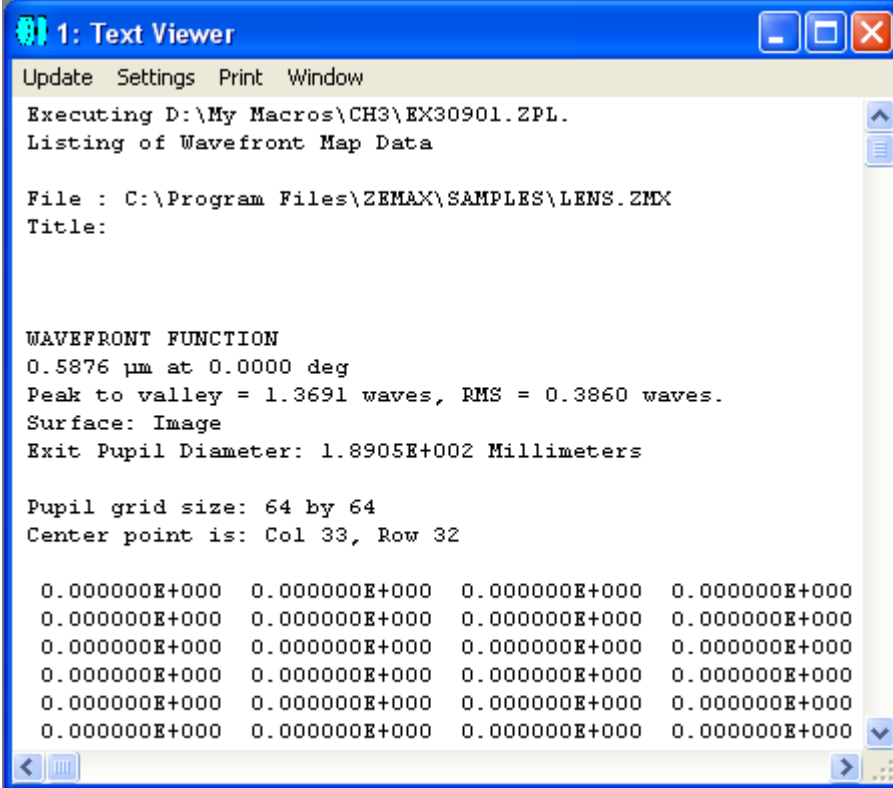
Rmf	RMS vs. Focus
Rms	RMS vs. Field
Rmw	RMS vs. Wavelength
Rtr	Ray Trace
Sag	Sag Table
Sei	Seidel Coefficients
Smf	Surface MTF
Spt	Spot Diagram
Srp	Surface Phase
Srs	Surface Sag
Sur	Surface Data
Sys	System Data
Tfg	Geometric through focus MTF
Tfm	FFT Through Focus MTF
Tls	Tolerance List
Tpl	Test Plate List
Tra	Polarization Transmission Data
Trw	Internal Transmission vs Lambda
Tsm	Tolerance Data Summary
Uni	Universal Plot – 1D
Un2	Universal Plot – 2D
Vig	Vignetting Plot
Wfm	Wavefront Map
Xdi	Extended Diffra Image Analysis
Xse	Extended source encircled energ
Yni	YNI Contribution
Yyb	Y-Ybar
Zat	Zernike Annular Coefficients

Zfr	Zernike Fringe Coefficients
Zst	Zernike Standard Coefficients

In example 3.9-1, we will show how to get the analysis result as in figure 3.9-1(c).

```
1 ! ex30901
2 ! This program shows how to get analysis information
3 ! Assume the lens system is defined in ex30401
4
5 ! Get a temporary file name
6 A$ = $TEMPFILENAME()
7
8 ! Compute the data and place in the temp file
9 GETTEXTFILE A$, Wfm
10
11 ! Open the temp file and print it out
12 OPEN A$
13 LABEL 1
14 READSTRING B$
15 IF (!EOFF())
16   PRINT B$
17   GOTO 1
18 ENDIF
19 CLOSE
```

In this program, we created a temporary file through function \$TEMPFILENAME(), then used keyword GETTEXTFILE to read out the analysis result of wavefront and save it to the temporary file. After that, we only need to open the temporary file, read the content line by line and display it on the screen, and we can get the text information of the wavefront as shown in figure 3.9-2:



```

1: Text Viewer
Update Settings Print Window
Executing D:\My Macros\CH3\EX30901.ZPL.
Listing of Wavefront Map Data

File : C:\Program Files\ZEMAX\SAMPLES\LENS.ZMX
Title:

WAVEFRONT FUNCTION
0.5876 µm at 0.0000 deg
Peak to valley = 1.3691 waves, RMS = 0.3860 waves.
Surface: Image
Exit Pupil Diameter: 1.8905E+002 Millimeters

Pupil grid size: 64 by 64
Center point is: Col 33, Row 32

0.000000E+000 0.000000E+000 0.000000E+000 0.000000E+000
0.000000E+000 0.000000E+000 0.000000E+000 0.000000E+000
0.000000E+000 0.000000E+000 0.000000E+000 0.000000E+000
0.000000E+000 0.000000E+000 0.000000E+000 0.000000E+000
0.000000E+000 0.000000E+000 0.000000E+000 0.000000E+000
0.000000E+000 0.000000E+000 0.000000E+000 0.000000E+000

```

Fig. 3.9-2: The analysis result of wavefront read by ZPL program.

If we compare figure 3.9-1 and 3.9-2, we can find that they are actually the same.

When we use keyword GETTEXTFILE to analyze and read information, if we want to modify the setting file, we can use keyword MODIFYSETTINGS. The syntax is:

MODIFYSETTINGS settingsfilename\$, type, value

In this command, the settingsfilename must be in quotes, or be a string variable name, and include the full path, name, and extension for the file to be modified. The type argument is a text mnemonic that indicates which setting within the file is to be modified. The supported values for the type argument are listed in the table below. Because there are many different types of analysis windows, and each has many different settings available, the list of types does not include all possible settings. After the command is executed, the old setting file will be updated.

Table 3.9-2: type codes supported by MODIFYSETTINGS

Feature	Available type codes
2D Layout	LAY_RAYS: The number of rays.
Detector Viewer	DVW_SURFACE: The surface number. Use 1 for Non-Sequential mode. DVW_DETECTOR: The detector number. DVW_SHOW: The “show as” setting. The meaning depends upon the type of window displayed: For Graphics Windows: Use 0 for grey scale, 1 for inverted grey scale, 2 for false color, 3 for inverted false color, 4 for cross section row, and 5 for cross section column. For Text Windows: Use 0 for full pixel data, 1 for cross section row, and 2 for cross section column. DVW_ROWCOL: The row or column number for cross section plots. DVW_ZPLANE: The Z-Plane number for detector volumes. DVW_SCALE: The scale mode. Use 0 for linear, 1 for Log -5, 2 for Log -10, and 3 for Log -15. DVW_SMOOTHING: The integer smoothing value. DVW_DATA: Use 0 for incoherent irradiance, 1 for coherent irradiance, 2 for coherent phase, 3 for radiant intensity, 4 for radiance (position space), and 5 for radiance (angle space). DVW_ZRD: The ray data base name, or null for none. DVW_FILTER: The filter string. DVW_MAXPLOT: The maximum plot scale. DVW_MINPLOT: The minimum plot scale. DVW_OUTPUTFILE: The output file name.
Extended Diffraction Image Analysis	EXD_DISPLAYSIZE: The display size. EXD_FIELD: The field number. EXD_FILESIZE: The file size. EXD_WAVE: The wavelength number.
FFT Line/Edge Spread	LSF_COHERENT: Use 0 for incoherent, 1 for coherent LSF_TYPE: Use 0-9 for X-Linear, Y-Linear, X-Log, Y-Log, X-Phase, Y-Phase, XReal, Y-Real, X-Imaginary, or Y-Imaginary, respectively. LSF_SAMP: The sampling, use 1 for 32 x 32, 2 for 64 x 32, etc. LSF_SPREAD: Use 0 for line, 1 for edge. LSF_WAVE: The wavelength number, use 0 for polychromatic (incoherent only) LSF_FIELD: The field number. LSF_POLARIZATION: Use 0 for unpolarized, 1 for polarized. LSF_PLOTSCALE: The plot scale.
FFT PSF	PSF_TYPE: Use 0-4 for Linear, Log, Phase, Real, or Imaginary, respectively. PSF_SAMP: The sampling, use 1 for 32 x 32, 2 for 64 x 32, etc. PSF_WAVE: The wavelength number, use 0 for polychromatic. PSF_FIELD: The field number. PSF_SURFACE: The surface number, use 0 for image. PSF_POLARIZATION: Use 0 for unpolarized, 1 for polarized. PSF_NORMALIZE: Use 0 for unnormalized, 1 for unity normalization. PSF_IMAGEDELTA: The image point spacing in micrometers.
FFT PSF Cross	PSF_TYPE: Use 0-9 for X-Linear, Y-Linear, X-Log, Y-Log, X-Phase, Y-Phase,

Section	<p>XReal, Y-Real, X-Imaginary, or Y-Imaginary, respectively. PSF_ROW: The row number (if doing an X scan) or column number (if doing a Y scan). Use 0 for center. PSF_SAMP: The sampling, use 1 for 32 x 32, 2 for 64 x 32, etc. PSF_WAVE: The wavelength number, use 0 for polychromatic. PSF_FIELD: The field number. PSF_POLARIZATION: Use 0 for unpolarized, 1 for polarized. PSF_NORMALIZE: Use 0 for unnormalized, 1 for unity normalization. PSF_PLOTSCALE: The plot scale.</p>
Footprint Diagram	<p>FOO_RAYDENSITY: The ray density. Use 0 for ring, 1 for 10, 2 for 15, 3 for 20 etc. FOO_SURFACE: The surface number. FOO_FIELD: The field number. FOO_WAVELENGTH: The wavelength number. FOO_DELETEVIGNETTED: Delete vignetted, use 0 for no, 1 for yes.</p>
Geometric Bitmap Image Analysis	<p>GBM_FIELDSIZE: The field Y size. GBM_RAYS: The number of rays per source pixel. GBM_XPIX: The number of X pixels. GBM_YPIX: The number of Y pixels. GBM_XSIZ: The X pixel size. GBM_YSIZ: The Y pixel size. GBM_INPUT: The input file name GBM_OUTPUT: The output file name GBM_SURFACE: The surface number GBM_ROTATION: The rotation setting</p>
Geometric Image Analysis	<p>IMA_FIELD: The field size. IMA_IMAGESIZE: The image size. IMA_IMANAME: The image file name. IMA_KRAYS: The number of rays x 1000. IMA_NA: The numerical aperture. IMA_OUTNAME: The output file name. IMA_SURFACE: The surface number. IMA_PIXELS: The number of pixels.</p>
FFT Through Focus MTF	<p>TFM_SAMP: The sampling. Use 1 for 32x32, 2 for 64x64, etc. TFM_DELTAFOC: The delta focus. TFM_FREQ: The spatial frequency for which the data is plotted. TFM_STEPS: The number of focal plane steps. TFM_WAVE: The wavelength number. Use 0 for all. TFM_FIELD: The field number. Use 0 for all. TFM_TYPE: The data type. Use 0 for modulation, 1 for real, 2 for imaginary, 3 for phase, or 4 for square wave. TFM_POLAR: Use polarization. Use 0 for no, 1 for yes. TFM_DASH: Use dashes. Use 0 for no, 1 for yes.</p>
Huygens MTF	<p>HMF_PUPILSAMP: The pupil sampling. Use 1 for 32x32, 2 for 64x64, etc. HMF_IMAGESAMP: The image sampling. Use 1 for 32x32, 2 for 64x64, etc. HMF_IMAGEDELTA: The image point spacing in micrometers. HMF_CONFIG: The configuration number. Use 0 for all, 1 for current, etc. HMF_WAVE: The wavelength number. Use 0 for polychromatic. HMF_FIELD: The field number. Use 0 for all. HMF_TYPE: The data type. Currently only modulation (0) is supported.</p>

	<p>HMF_MAXF: The maximum spatial frequency. HMF_POLAR: Use polarization. Use 0 for no, 1 for yes. HMF_DASH: Use dashes. Use 0 for no, 1 for yes.</p>
Huygens Through Focus MTF	<p>HTF_PUPILSAMP: The pupil sampling. Use 1 for 32x32, 2 for 64x64, etc. HTF_IMAGESAMP: The image sampling. Use 1 for 32x32, 2 for 64x64, etc. HTF_IMAGEDELTA: The image point spacing in micrometers. HTF_CONFIG: The configuration number. Use 0 for all, 1 for current, etc. HTF_FREQ: The spatial frequency for which data is plotted. HTF_WAVE: The wavelength number. Use 0 for all. HTF_FIELD: The field number. Use 0 for all. HTF_TYPE: The data type. Currently only modulation (0) is supported. HTF_DELTAFOC: The delta focus. HTF_STEPS: The number of focal plane steps. HTF_POLAR: Use polarization. Use 0 for no, 1 for yes. HTF_DASH: Use dashes. Use 0 for no, 1 for yes.</p>
Huygens MTF vs. Field	<p>HMH_SAMP: The sampling. Use 1 for 32x32, 2 for 64x64, etc. HMH_SCANTYPE: The field scan type. Use 0 for +Y, 1 for +X, etc. HMH_WAVE: The wavelength. Use 0 for all. HMH_FIELDDENSITY: The field density. HMH_FREQ1: Spatial frequency 1. HMH_FREQ2: Spatial frequency 2. HMH_FREQ3: Spatial frequency 3. HMH_FREQ4: Spatial frequency 4. HMH_FREQ5: Spatial frequency 5. HMH_FREQ6: Spatial frequency 6. HMH_POLAR: Use polarization. Use 0 for no, 1 for yes. HMH_DASH: Use dashes. Use 0 for no, 1 for yes. HMH_REMOVEVIGNETTING: Remove vignetting factors. Use 0 for no, 1 for yes.</p>
Huygens PSF	<p>HPS_PUPILSAMP: The pupil sampling, use 1 for 32 x 32, 2 for 64 x 64, etc. HPS_IMAGESAMP: The image sampling, use 1 for 32 x 32, 2 for 64 x 64, etc. HPS_WAVE: The wavelength number, use 0 for polychromatic. HPS_FIELD: The field number. HPS_IMAGEDELTA: The image point spacing in micrometers. HPS_TYPE: The data type. Use 0-8 for Linear, Log -1, Log -2, Log -3, Log -4, Log -5, Real, Imaginary, or Phase, respectively.</p>
Huygens PSF Cross Section	<p>HPC_PUPILSAMP: The pupil sampling, use 1 for 32 x 32, 2 for 64 x 64, etc. HPC_IMAGESAMP: The image sampling, use 1 for 32 x 32, 2 for 64 x 64, etc. HPC_WAVE: The wavelength number, use 0 for polychromatic. HPC_FIELD: The field number. HPC_IMAGEDELTA: The image point spacing in micrometers. HPC_TYPE: The data type. Use 0-9 for X-Linear, X-Log, Y-Linear, Y-Log, X-Real, Y-Real, X-Imaginary, Y-Imaginary, X-Phase, or Y-Phase, respectively.</p>
Illumination XY Scan	<p>ILL_SOURCE: The source size. ILL_SMOOTH: The smoothing value to use. ILL_DETSIZE: The detector size. ILL_SURFACE: The surface number.</p>
Image Simulation	<p>ISM_INPUTFILE: The input file name. This should be specified without a path. ISM_FIELDHEIGHT: The Y field height. ISM_OVERSAMPLING: Oversample value. Use 0 for none, 1 for 2X, 2 for 4x,</p>

	<p>etc.</p> <p>ISM_GUARDBAND: Guard band value. Use 0 for none, 1 for 2X, 2 for 4x, etc.</p> <p>ISM_FLIP: Flip Source. Use 0 for none, 1 for TB, 2 for LR, 3 for TB&LR.</p> <p>ISM_ROTATE: Rotate Source: Use 0 for none, 1 for 90, 2 for 180, 3 for 270.</p> <p>ISM_WAVE: Wavelength. Use 0 for RGB, 1 for 1+2+3, 2 for wave #1, 3 for wave #2, etc.</p> <p>ISM_FIELD: Field number.</p> <p>ISM_PSAMP: Pupil Sampling. Use 1 for 32x32, 2 for 64x64, etc.</p> <p>ISM_ISAMP: Image Sampling. Use 1 for 32x32, 2 for 64x64, etc.</p> <p>ISM_PSFY, ISM_PSFY: The number of PSF grid points.</p> <p>ISM_ABERRATIONS: Use 0 for none, 1 for geometric, 2 for diffraction.</p> <p>ISM_POLARIZATION: Use 0 for no, 1 for yes.</p> <p>ISM_FIXEDAPERTURES: Use 0 for no, 1 for yes.</p> <p>ISM_USERI: Use 0 for no, 1 for yes.</p> <p>ISM_SHOWAS: Use 0 for Simulated Image, 1 for Source Bitmap, and 2 for PSF Grid.</p> <p>ISM_REFERENCE: Use 0 for chief ray, 1 for vertex, 2 for primary chief ray.</p> <p>ISM_SUPPRESS: Use 0 for no, 1 for yes.</p> <p>ISM_PIXELSIZE: Use 0 for default or the size in lens units.</p> <p>ISM_XSIZE, ISM_YSIZE: Use 0 for default or the number of pixels.</p> <p>ISM_FLIPIMAGE: Use 0 for none, 1 for top-bottom, etc.</p> <p>ISM_OUTPUTFILE: The output file name or empty string for no output file.</p>
MTF - FFT	<p>MTF_SAMP: The pupil sampling, use 1 for 32, 2 for 64, etc.</p> <p>MTF_WAVE: The wavelength number, use 0 for all.</p> <p>MTF_FIELD: The field number, use 0 for all.</p> <p>MTF_TYPE: Use 0 for modulation, 1 for real, 2 for imaginary, 3 for phase, 4 for square wave.</p> <p>MTF_SURF: The surface number, use 0 for image.</p> <p>MTF_MAXF: The maximum frequency, use 0 for default.</p> <p>MTF_SDLI: Show diffraction limit, 0 for no, 1 for yes.</p> <p>MTF_POLAR: Polarization, 0 for no, 1 for yes.</p> <p>MTF_DASH: Use dashes, 0 for no, 1 for yes.</p>
NSC Object Viewer	<p>SHA_ROTX: The x rotation in degrees.</p> <p>SHA_ROTY: The y rotation in degrees.</p> <p>SHA_ROTZ: The z rotation in degrees.</p>
NSC Shaded Model	<p>SHA_ROTX: The x rotation in degrees.</p> <p>SHA_ROTY: The y rotation in degrees.</p> <p>SHA_ROTZ: The z rotation in degrees.</p>
Partially Coherent Image Analysis	<p>PCI_FIELD: The field number.</p> <p>PCI_FILESIZE: The file size.</p> <p>PCI_WAVE: The wavelength number.</p> <p>PCI_RESAMPLE: The resample image setting, 0 for no 1 for yes.</p> <p>PCI_RSNX: The resample number x</p> <p>PCI_RSNY: The resample number y</p> <p>PCI_RSDCX: The resample decenter x</p> <p>PCI_RSDCY: The resample decenter y</p> <p>PCI_RSDLX: The resample delta x</p> <p>PCI_RSDLY: The resample delta y</p>
Polarization Pupil Map	<p>PPM_SAMP: The sampling, use 0 for 3x3, 1 for 5x5, 2 for 7x7, etc.</p> <p>PPM_FIELD: The field number.</p>

	<p>PPM_WAVE: The wavelength number. PPM_SURFACE: The surface number. PPM_JX: The Jx amplitude. PPM_JY: The Jy amplitude. PPM_PX: The Px phase. PPM_PY: The Py phase. PPM_ADDCONFIG: The add configs string. PPM_SUBCONFIGS: The subtract configs string.</p>
Physical Optics Propagation - General Tab	<p>POP_END: The end surface. POP_FIELD: The field number. POP_START: The starting surface. POP_WAVE: The wavelength number.</p>
Physical Optics Propagation - Beam Definition Tab	<p>POP_AUTO: Simulates the pressing of the “auto” button which chooses appropriate X and Y beam widths based upon the sampling and other settings. POP_BEAMTYPE: Selects the beam type. Use 0 for Gaussian Waist, 1 for Gaussian Angle, 2 for Gaussian Size + Angle, 3 for Top Hat, 4 for File, 5 for DLL and 6 for Multimode. POP_PARAMn: Sets beam parameter n, for example, use POP_PARAM3 to set parameter3. POP_PEAKIRRAD: Sets the normalization by peak irradiance. POP_POWER: Sets the normalization by total beam power. POP_SAMPX: The X direction sampling, use 1 for 32, 2 for 64, etc. POP_SAMPY: The Y direction sampling, use 1 for 32, 2 for 64, etc. POP_SOURCEFILE: The file name if the starting beam is defined by a ZBF file, DLL, or multimode file. POP_WIDEX: The X direction width. POP_WIDEY: The Y direction width.</p>
Physical Optics Propagation - Fiber Data Tab	<p>POP_COMPUTE: Use 1 to check the fiber coupling integral on, 0 to check it off. POP_FIBERFILE: The file name if the fiber mode is defined by a ZBF or DLL. POP_FIBERTYPE: Use the same values as POP_BEAMTYPE above, except for multimode which is not yet supported. POP_FPARAMn: Sets fiber parameter n, for example, use POP_FPARAM3 to set fiber parameter3. POP_IGNOREPOL: Use 1 to ignore polarization, 0 to consider polarization. POP_POSITION: Fiber position setting. Use 0 for chief ray, 1 for surface vertex. POP_TILTX: The X-Tilt. POP_TILTY: The Y-Tilt.</p>
Relative Illumination	<p>REL_RAYDENSITY: The number of rays. REL_FIELDDENSITY: The number of field points. REL_WAVE: The wavelength number, use 0 for all. REL_POLAR: Use 1 to use polarization, 0 to ignore polarization REL_LOG: Use 1 for a log scale, 0 for linear. REL_REMOVEVIGNETTING: Use 1 to remove vignetting factors, otherwise 0. REL_SCANTYPE: Use 0 for +y, 1 for +x, 2 for -y, or 3 for -x scan direction.</p>
Shaded Model	<p>SHA_ROTX: The x rotation in degrees. SHA_ROTY: The y rotation in degrees. SHA_ROTZ: The z rotation in degrees.</p>

Spot Diagram	SPT_RAYS: The ray density.
Surface Sag	SRS_SAMP: The sampling. Use 1 for 33x33, 2 for 65x65, etc. SRS_SURF: The surface number.
Universal Plot 1D	UN1_CATEGORY: Use 0 for surface, 1 for system, 2 for config. UN1_PARAMETER: Use 0 for first option, 1 for second option, etc. UN1_SURFACE: The surface or configuration number. UN1_STARTVAL: The start value for the independent variable. UN1_STOPVAL: The stop value for the independent variable. UN1_STEPS: The number of steps between start and stop. UN1_OPERAND: The optimization operand name. UN1_MFLINE: The optimization operand line number. Use 0 for MF value. UN1_PAR1: Operand parameter 1. UN1_PAR2: Operand parameter 2. UN1_PAR3: Operand parameter 3. UN1_PAR4: Operand parameter 4. UN1_PAR5: Operand parameter 5. UN1_PAR6: Operand parameter 6. UN1_PAR7: Operand parameter 7. UN1_PAR8: Operand parameter 8. UN1_PLOTMIN: The minimum plot value for the dependent variable. UN1_PLOTMAX: The maximum plot value for the dependent variable. UN1_TITLE: The plot title.
Universal Plot 2D	UN2_CATEGORYX: Use 0 for surface, 1 for system, 2 for config. UN2_PARAMETERX: Use 0 for first option, 1 for second option, etc. UN2_SURFACEX: The surface or configuration number. UN2_STARTVALX: The start value for the independent variable. UN2_STOPVALX: The stop value for the independent variable. UN2_STEPSX: The number of steps between start and stop. UN2_CATEGORYY: Use 0 for surface, 1 for system, 2 for config. UN2_PARAMETERY: Use 0 for first option, 1 for second option, etc. UN2_SURFACEY: The surface or configuration number. UN2_STARTVALY: The start value for the independent variable. UN2_STOPVALY: The stop value for the independent variable. UN2_STEPSY: The number of steps between start and stop. UN2_OPERAND: The optimization operand name. UN2_MFLINE: The optimization operand line number. Use 0 for MF value. UN2_PAR1: Operand parameter 1. UN2_PAR2: Operand parameter 2. UN2_PAR3: Operand parameter 3. UN2_PAR4: Operand parameter 4. UN2_PAR5: Operand parameter 5. UN2_PAR6: Operand parameter 6. UN2_PAR7: Operand parameter 7. UN2_PAR8: Operand parameter 8. UN2_SHOWAS: Data display. Use 0 for surface, 1 for contour, etc. UN2_CONTOURFORMAT: Contour format string. UN2_PLOTMIN: The minimum plot value for the dependent variable. UN2_PLOTMAX: The maximum plot value for the dependent variable. UN2_TITLE: The plot title.

Wavefront Map	<p>WFM_SAMP: The sampling, use 1 for 32, 2 for 64, etc. WFM_FIELD: The field number. WFM_WAVE: The wavelength number. WFM_SUBSR: The sub aperture radius. WFM_SUBSX: The sub aperture X decenter. WFM_SUBSY: The sub aperture Y decenter.</p>
---------------	--

Example 3.9-2 shows how to modify settings using keyword MODIFYSETTINGS in the program. This program is almost the same as example 3.9-1, with only lines to modify settings added at the very beginning (lines 5 and 6), as shown below:

```

1 ! ex30902
2 ! This program shows how to modify setting file for analysis
3 ! Assume the lens system is defined in ex30401
4
5 settingFile$ = "C:\Program Files\ZEMAX\Samples\LENS.CFG"
6 MODIFYSETTINGS settingFile$, WFM_SAMP, 3
7
8 ! Get a temporary file name
9 A$ = $TEMPFILENAME()
10
11 ! Compute the data and place in the temp file
12 GETTEXTFILE A$, Wfm
13
14 ! Open the temp file and print it out
15 OPEN A$
16 LABEL 1
17 READSTRING B$
18 IF (!EOFF())
19     PRINT B$
20     GOTO 1
21 ENDIF
22 CLOSE

```

Please note that the parameter setting file is placed in the same folder as the lens file. In this program, we set the wavefront sampling value as 3, i.e. 128x128. The result of the program is shown in figure 3.9-3:

```

1: Text Viewer
Update Settings Print Window
Executing D:\My Macros\CH3\EX30902.ZPL.
Listing of Wavefront Map Data

File : C:\Program Files\ZEMAX\SAMPLES\LENS.ZMX
Title:

WAVEFRONT FUNCTION
0.5876 μm at 0.0000 deg
Peak to valley = 1.3691 waves, RMS = 0.3852 waves.
Surface: Image
Exit Pupil Diameter: 1.8905E+002 Millimeters

Pupil grid size: 128 by 128
Center point is: Col 65, Row 64

0.000000E+000 0.000000E+000 0.000000E+000 0.000000E+000
0.000000E+000 0.000000E+000 0.000000E+000 0.000000E+000
0.000000E+000 0.000000E+000 0.000000E+000 0.000000E+000
0.000000E+000 0.000000E+000 0.000000E+000 0.000000E+000
0.000000E+000 0.000000E+000 0.000000E+000 0.000000E+000
0.000000E+000 0.000000E+000 0.000000E+000 0.000000E+000

```

Fig. 3.9-3: Analysis result of the wavefront read by ZPL program after changing setting.

If we compare this result to that shown in figure 3.9-2, we can see that the sampling value has been changed to “Pupil grid size: 128 by 128”.

Besides the method discussed above, ZPL also provided many functions and keywords to read some commonly used analysis information directly, such as IMAE(), OPDC(), OPTH(), GETLSF, GETMTF, GETPSF, GETZERNIKE, POP, XDIFFIA, GETT(), etc.

Function IMAE(seed) is used to read the geometric image analysis efficiency. If seed is 0, each time when the function is called, the same random number will be used, otherwise, a different random number will be used.

Function OPDC() is used to calculate The optical path difference, and is only valid after executing RAYTRACE command. OPDC will not return valid data if the chief ray cannot be traced. Please note that no matter what the lens unit is, the returned unit from this function is always mm. An example is given below for the application of this function.

```

1 ! ex30904
2 ! This program shows how to use function OPDC()
3 ! Assume the lens system is defined in ex30401
4
5 ! first define the marginal ray
6 hx = 0
7 hy = 0
8 px = 0
9 py = 1
10 wavelength = 2
11
12 RAYTRACE hx, hy, px, py, wavelength # trace the marginal ray
13
14 ! then calculate the optical path difference
15 ! between the marginal ray and the chief ray
16
17 opticalPathDiff = OPDC()
18
19 PRINT
20 PRINT "The optical path difference is ", opticalPathDiff, " mm"

```

We assume the optical system is the doublet defined in program ex30401.ZPL. In this program, we first defined an arbitrary ray. For the sake of discussion, we use marginal ray here. Then, we did ray tracing using RAYTRACE command, and finally we called function OPDC() to compare the optical path difference between the marginal ray and the chief ray. The result is shown below:

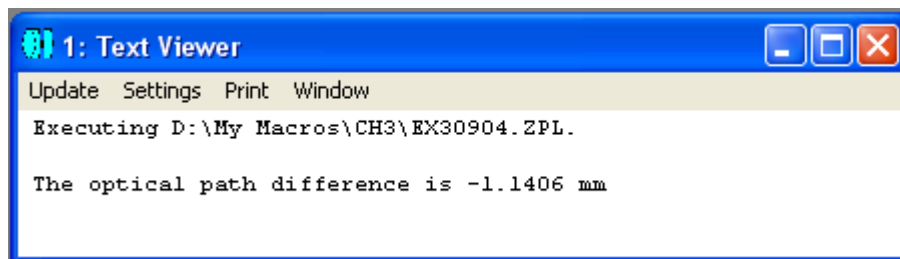


Fig. 3.9-5: Result of program ex30904.ZPL

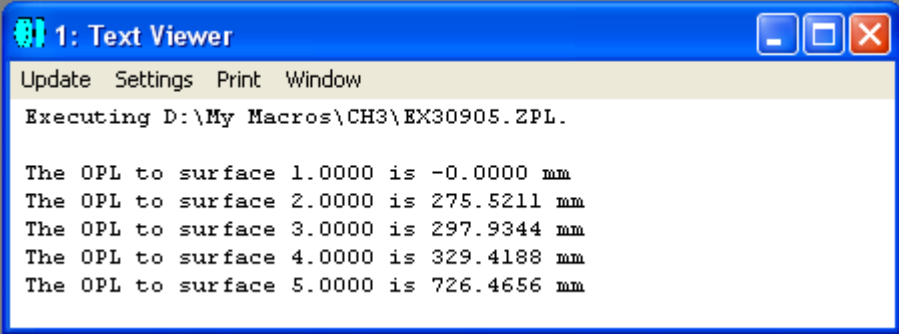
Function OPTH(n) is used to calculate the total optical path length along the ray to the specified surface. It considers the phase added by diffractive surfaces such as gratings, holograms, and binary optics. It is valid only after a RAYTRACE call. OPTH will not return valid data if the chief ray cannot be traced. Please note that different from function OPDC(), the unit of the return value of function OPTH(n) is current lens unit. Example 3.9-5 shows how to use this function in ZPL program.


```

1 ! ex30905
2 ! This program shows how to use function OPTH()
3 ! Assume the lens system is defined in ex30401
4
5 ! first define the ray (we use marginal ray here)
6 hx = 0
7 hy = 0
8 px = 0
9 py = 1
10 wavelength = 2
11
12 RAYTRACE hx, hy, px, py, wavelength # trace the ray
13
14 ! find out the lens unit
15 u = UNIT()
16 IF u == 0 THEN lensUnit$ = "mm"
17 IF u == 1 THEN lensUnit$ = "cm"
18 IF u == 2 THEN lensUnit$ = "inch"
19 IF u == 3 THEN lensUnit$ = "m"
20
21 ! then calculate the optical path length (OPL) to a given surface
22
23 PRINT
24 FOR sn, 1, 5, 1
25   opticalPath = OPTH(sn)
26   PRINT "The OPL to surface ", sn, " is ", opticalPath, " ", lensUnit$
27 NEXT

```

We assume the optical system is the doublet defined in program ex30401.ZPL. In this program, we first defined an arbitrary ray (the marginal ray), then, we did ray tracing using RAYTRACE command, and finally we called function OPTH() to calculate the total light path of the defined ray. In the program we also used function UNIT() to read current lens unit. The result is shown below:



```

1: Text Viewer
Update Settings Print Window
Executing D:\My Macros\CH3\EX30905.ZPL.

The OPL to surface 1.0000 is -0.0000 mm
The OPL to surface 2.0000 is 275.5211 mm
The OPL to surface 3.0000 is 297.9344 mm
The OPL to surface 4.0000 is 329.4188 mm
The OPL to surface 5.0000 is 726.4656 mm

```

Fig. 3.9-6: result of program ex30905.ZPL.

Keyword GETLSF is used to calculate the geometric edge and line response functions. The syntax is:

GETLSF wave, field, sampling, vector, maxradius, use_polarization

Wave is an integer corresponding to the wavelength number to use for the calculation. A value of zero indicates a polychromatic calculation. Field must be an integer between 1 and the maximum number of fields. The value indicates which field position to use. Sampling may be 1 (32 x 32), 2 (64 x 64), 3 (128 x 128), etc... up to 2048 x 2048. The vector argument must be an integer value between 1 and 4, and specifies which vector array the data should be placed in. The maxradius argument is the maximum radial coordinate of the edge and line spread functions; this is the half-width of the data range. Use 0 for a default width. If any of the arguments fall outside the valid ranges, then the nearest acceptable value is used instead. The data is returned as an array of values in the specified vector. Vector position 0-3 will hold the number of points "N", the starting x coordinate (this is the negative of the half width of the data range), the delta coordinate, and the offset (defined below), respectively. The offset is the first position in the vector that holds the edge or line spread data. Starting at the offset, the first N value are the tangential LSF response. The next N values are the sagittal LSF response. The tangential and sagittal ERF values are in the next two groups of N data values. If the current vector size is not large enough, Zemax will automatically increase the size of the vectors to hold the LSF data in the manner described in SETVECSIZE.

Example 3.9-6 shows how to use this keyword in ZPL program:

```
1 ! ex30906
2 ! This program shows how to use key word GETLSF
3 ! Assume the lens system is defined in ex30401
4
5 wave = 2
6 field = 1
7 sampling = 2
8 vector = 1
9 maxradius = 10
10 use_polarization = 0
11
12 GETLSF wave, field, sampling, vector, maxradius, use_polarization
13
14 FOR n, 0, 1000, 1
15     PRINT "VEC1(", n, ") = ", VEC1(n)
16 NEXT
```

Assume the optical system is the doublet defined in program ex30401.ZPL. We calculated the geometric edge and line response functions of the system using keyword GETLSF, and displayed the result stored in vector VEC1. The result is shown below:

```

1: Text Viewer
Update Settings Print Window
Executing D:\My Macros\CH3\EX30906.ZPL.
VEC1(0.0000) = 101.0000
VEC1(1.0000) = -10.0000
VEC1(2.0000) = 0.2000
VEC1(3.0000) = 10.0000
VEC1(4.0000) = 0.0000
VEC1(5.0000) = 0.0000
VEC1(6.0000) = 0.0000
VEC1(7.0000) = 0.0000
VEC1(8.0000) = 0.0000
VEC1(9.0000) = 0.0000
VEC1(10.0000) = 0.3421
VEC1(11.0000) = 0.4210
VEC1(12.0000) = 0.3684
VEC1(13.0000) = 0.4473
VEC1(14.0000) = 0.2631
VEC1(15.0000) = 0.4737
VEC1(16.0000) = 0.2894
VEC1(17.0000) = 0.2894
VEC1(18.0000) = 0.4473
VEC1(19.0000) = 0.2368
VEC1(20.0000) = 0.2895
VEC1(21.0000) = 0.3158

1: Text Viewer
Update Settings Print Window
VEC1(401.0000) = 0.7223
VEC1(402.0000) = 0.7269
VEC1(403.0000) = 0.7306
VEC1(404.0000) = 0.7354
VEC1(405.0000) = 0.7403
VEC1(406.0000) = 0.7446
VEC1(407.0000) = 0.7491
VEC1(408.0000) = 0.7538
VEC1(409.0000) = 0.7573
VEC1(410.0000) = 0.7596
VEC1(411.0000) = 0.7632
VEC1(412.0000) = 0.7678
VEC1(413.0000) = 0.7735
VEC1(414.0000) = 0.0000
VEC1(415.0000) = 0.0000
VEC1(416.0000) = 0.0000
VEC1(417.0000) = 0.0000
VEC1(418.0000) = 0.0000
VEC1(419.0000) = 0.0000
VEC1(420.0000) = 0.0000
VEC1(421.0000) = 0.0000
VEC1(422.0000) = 0.0000
VEC1(423.0000) = 0.0000

```

Fig. 3.9-7: result of program ex30906.ZPL

From the result we can see that the index of array VEC1 starts from 0, which is different from user defined vector array that starts from 1. We have mentioned this before when we discuss array in last chapter. VEC1(0) ~ VEC1(3) store number of data points $N = 101$, coordinate $x = -10$ of starting point, step size 0.2 and offset 10. The actual data is determined by the offset. Starting from VEC1(10), the first group of $N = 101$ data is the tangential LSF response, the next N values are the sagittal LSF response. The tangential and sagittal ERF values are in the next two groups of N data values. The data ends at VEC1(413), as shown in figure 3.9-7.

Keyword GETMTF is used to calculate tangential and sagittal MTF, real part, imaginary part, phase, or square wave response data for the currently loaded lens file, and places the data in one of the vector arrays (either VEC1, VEC2, VEC3, or VEC4). The syntax is:

GETMTF freq, wave, field, sampling, vector, type

The freq argument is the desired spatial frequency in MTF Units. If the frequency is less than zero, or greater than the cutoff frequency, GETMTF returns zero. Wave is an integer corresponding to the wavelength number to use for the calculation. A value of zero indicates a polychromatic calculation. Field must be an integer between 1 and the maximum number of fields. The value indicates which field position to use. Sampling may be 1 (32 x 32), 2 (64 x 64), 3 (128 x 128), etc... up to 2048 x 2048. The vector argument must be an integer value between 1 and 4, and specifies which vector array the data should be placed in. The type argument refers to the data type: 1 for MTF, 2 for real part, 3 for imaginary part, 4 for phase in radians, 5 for square wave MTF. If any of the arguments fall outside the valid ranges, then the nearest acceptable value is used instead. This calculation uses the FFT MTF method. The data is returned in one of the vector arrays with the following format: Vector position 0: tangential response; Vector position 1: sagittal response.

Example 3.9-7 shows how to use this keyword in ZPL program:

```
1 ! ex30907
2 ! This program shows how to use key word GETMTF
3 ! Assume the lens system is defined in ex30401
4
5 freq = 30
6 wave = 2
7 field = 1
8 sampling = 2
9 vector = 1
10 type = 1
11
12 GETMTF freq, wave, field, sampling, vector, type
13
14 PRINT
15 PRINT "Tangential response: ", vec1(0)
16 PRINT "Sagittal response: ", vec1(1)
```

Assume the optical system is the doublet defined in program ex30401.ZPL. We calculated the tangential and sagittal MTF of the system using keyword GETMTF, and displayed the result stored in vector VEC1. The result is shown below:

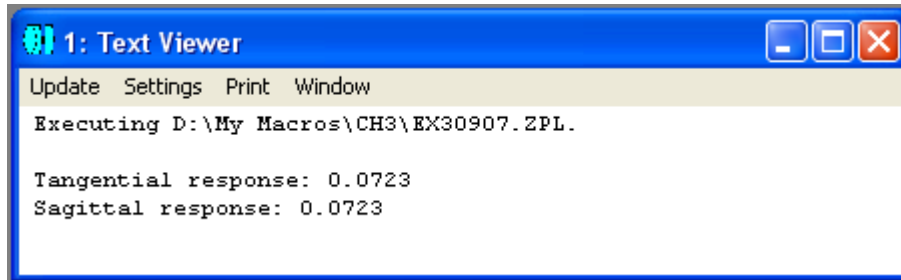


Fig. 3.9-8: result of program ex30907.ZPL

Keyword GETPSF is used to calculate the diffraction point spread function (PSF) using the FFT algorithm and places the data in one of the vector arrays (either VEC1, VEC2, VEC3, or VEC4). The syntax is:

GETPSF wave, field, sampling, vector, unnormalized, phaseflag, imagedelta

In this command, wave is an integer corresponding to the wavelength number to use for the calculation. A value of zero indicates a polychromatic calculation. Field must be an integer between 1 and the maximum number of fields. The value indicates which field position to use. Sampling may be 1 (32 x 32), 2 (64 x 64), 3 (128 x 128), etc... up to 2048 x 2048. The vector argument must be an integer value between 1 and 4, and specifies which vector array the data should be placed in. The unnormalized flag is zero if the data should be normalized to a peak of 1.0, if the unnormalized value is 1, then the data is returned unnormalized. If phase flag is zero, the data returned is intensity, if 1, then the phase in degrees is returned. The imagedelta value is the spacing between PSF points in micrometers; use zero for the default spacing. The wavelength must be monochromatic to compute phase data. If any of the arguments fall outside the valid ranges, then the nearest acceptable value is used instead. The data is returned in one of the vector arrays with the following format:

Vector position 0: the total number of PSF data points in the vector array. Usually, this number will be $4*n*n$ where n is the sampling size (32, 64, etc.). For example, if the sampling density is 2, the pupil sampling will be 64 x 64, and there will be 128 x 128 or 16,384 values in the array. This will require 8 bytes per number, or a total of 131 kb. A sampling density of 1024 will require at least 8 Mb just for the array; another 64 Mb or more to compute the PSF. Position 0 also returns other values as error codes. If position 0 is zero, then the computation was aborted. If -1, then the vector array is not large enough to hold all the data. Use SETVECSIZE to make the array bigger. If -2, then there is not enough system RAM to compute the PSF data. If -3, a general error occurred while computing the PSF.

Vector position 1 through $4*n*n$ holds the PSF data intensity. The first $2n$ values are the first row, going left to right from $-x$ to $+x$, then each subsequent block of $2n$ values is another row, going from $-y$ to $+y$. Vector position $4*n*n+1$ holds the spacing between data values in micrometers.

Example 3.9-8 shows how to use this keyword in ZPL program.

```

1 ! ex30908
2 ! This program shows how to use key word GETPSF
3 ! Assume the lens system is defined in ex30401
4
5 wave = 0 # use polychromatic light
6 field = 1
7 sampling = 2
8 vector = 3 # result saved in VEC3
9 unnormalized = 1
10 phaseflag = 0 # no phase data
11 imagedelta = 0 # default image delta
12
13 vecSize = 1000 # this is the default vector size
14
15 PRINT
16 FORMAT 5.0
17 PRINT "The original vector size is ", vecSize
18
19 label RESIZE
20 vecSize = vecSize+100
21 SETVECSIZE vecSize
22
23 GETPSF wave, field, sampling, vector, unnormalized, phaseflag, imagedelta
24 pointNum = VEC3(0)
25 IF pointNum == -1 THEN GOTO RESIZE
26
27 PRINT "The final vector size is ", vecSize
28 PRINT "There are ", pointNum, " data points."
29 FORMAT 3.4
30 PRINT "They spaced ", VEC3(pointNum+1), " micrometers apart."

```

Assume the optical system is the doublet defined in program ex30401.ZPL. We calculated the diffraction point spread function of the system using keyword GETPSF, and displayed the result stored in vector VEC3. Please note that the default length of vector VEC3 is 1000, too small for the result, so the return value of VEC3(0) is -1. We added a conditional statement in line 25. If result of -1 is detected, then go to line 19 to increase the vector length, and redo the calculation, until positive result is obtained. The result is shown below:

```

1: Text Viewer
Update Settings Print Window
Executing D:\My Macros\CH3\EX30908.ZPL.

The original vector size is 1000
The final vector size is 16400
There are 16384 data points.
They spaced 1.3310 micrometers apart.

```

Fig. 3.9-9: result of program ex30908.ZPL

Keyword GETZERNIKE is used to calculate Zernike Fringe, Standard, or Annular coefficients for the currently loaded lens file, and places them in one of the vector arrays (either VEC1, VEC2, VEC3, or VEC4). The syntax is:

GETZERNIKE maxorder, wave, field, sampling, vector, zerntype, epsilon, reference

The maxorder argument is any number between 1 and 37 for Fringe or between 1 and 231 for Standard or Annular coefficients, and corresponds to the highest Zernike term desired. Wave and field are the integer values for the wavelength and field number respectively. The value for sampling determines the size of the grid used to fit the coefficients. Sampling may be 1 (32 x 32), 2 (64 x 64), etc.... up to 2048 x 2048. The vector argument must be an integer value between 1 and 4, and specifies which vector array the data should be placed in. The zerntype is 0 for “fringe” Zernike terms, 1 for “Standard” Zernike terms, and 2 for “Annular” Zernike terms. For Annular Zernike Coefficients epsilon is the annular ratio; this value is ignored for other Zernike types. To reference the OPD to the chief ray, the reference value should be zero or omitted; use 1 to reference to the surface vertex. If any of the arguments fall outside the valid ranges, then the nearest acceptable value is used instead.

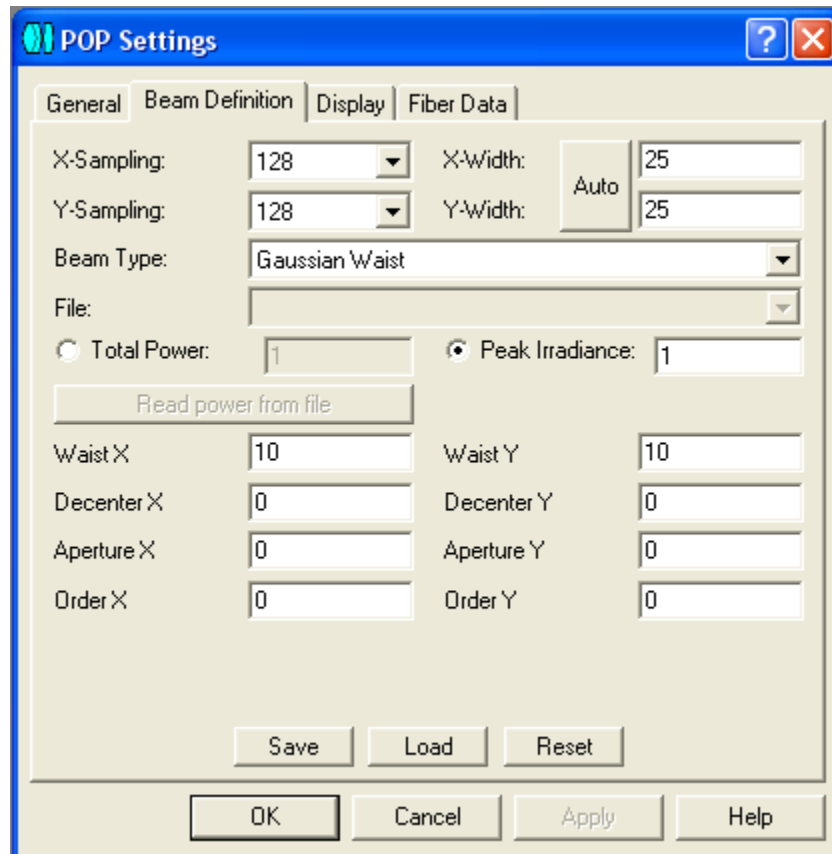
The data is returned in one of the vector arrays with the following format: Vector position 1: Peak to valley in waves; Vector position 2: RMS to the zero OPD line in waves (this value is not physically meaningful but is provided for reference); Vector position 3: RMS to the chief ray in waves; Vector position 4: RMS to the image centroid in waves (this is the most physically meaningful number related to image quality); Vector position 5: Variance in waves; Vector position 6: Strehl ratio; Vector position 7: RMS fit error in waves; Vector position 8: Maximum fit error (at any one point) in waves. The remaining vector positions contain the actual Zernike coefficient data. For example, Zernike term number 1 is in vector position 9, Zernike term 2 is in position 10, and so on.

Keyword POP is used to compute the Physical Optics Propagation (POP) of a beam through the optical system and saves the surface by surface results to ZBF files. The syntax is:

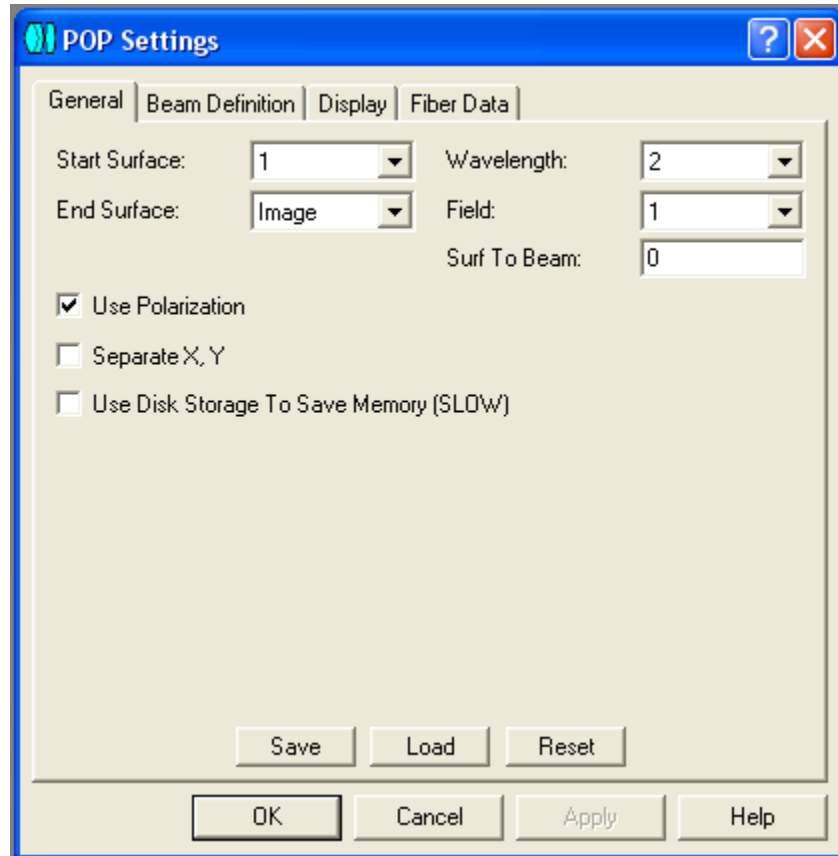
POP outfilename\$, lastsurface

This keyword requires the name of the output ZBF file, an expression that evaluates to the last surface to propagate to, and optionally the name of a settings file. The filename must be enclosed in quotes if any blank or other special characters are used. The created ZBF files will be placed in the <pop> folder. No paths should be provided with the file names. The settings for the POP feature will be those settings previously saved for the current lens, unless a settings file name is provided. The settings file name must include the full path, name, and extension. To make adjustments to the settings, open a POP window, choose the appropriate settings, then press "Save". By default, all subsequent calls to POP within ZPL will use the saved settings. The exceptions are the output file name, which is specified as the first argument after the POP keyword, and the last surface number, which is optionally specified as the second argument after the POP keyword.

We will discuss more on ZBF file related commands in section 14. Here we just give an example to show how to use keyword POP in ZPL program. First we assume the optical system is the doublet defined in program ex30401.ZPL, and also we assume the POP settings are as shown in figure 3.9-10:



(a)



(b)

Fig. 3.9-10(a)(b): POP settings for program ex30910.ZPL

The program is shown below:

```

1 ! ex30910
2 ! This program shows how to use key word POP
3 ! Assume the lens system is defined in ex30401
4
5 outfilename1$ = "ex30910a.ZBF"
6 outfilename2$ = "ex30910b.ZBF"
7
8 lastsurface = 4
9 POP outfilename1$, lastsurface
10
11 lastsurface = 5
12 POP outfilename2$, lastsurface

```

In this program, we defined two different file names, and saved POP calculation results of surface 4 and 5 in those two files, respectively. Please note that there is no path name in the program, because all the ZBF files will be stored in the folder "...\\POP\\Beamfiles". We will use those two files in section 14.

Keyword XDIFFIA is used to compute the Extended Diffraction Image Analysis feature and saves the result to a ZBF file. The syntax is:

XDIFFIA outfilename\$, infilename\$

This keyword requires the name of the output ZBF file, and optionally, the name of the input IMA or BIM file. If the extension to the outfilename is not provided, the extension ZBF will be appended. The extension must be provided on the infilename. The filenames must be enclosed in quotes if any blank or other special characters are used. The outfilename will be placed in the <pop> folder. The infilename must be placed in the <data>\\<images> folder. No paths should be provided with the file names.

The settings for the Extended Diffraction Image Analysis feature will be those settings previously saved for the current lens. To make adjustments to the settings, open an Extended Diffraction Image Analysis window, choose the appropriate settings, then press "Save". All subsequent calls to XDIFFIA will use the saved settings. The exceptions are the output file name, which is specified as the first argument after the XDIFFIA keyword, and the input source file, which is optionally specified as the second argument after the XDIFFIA keyword.

Function GETT(window_num, line, column) is used to read the value defined by given line and column of any open text window defined by window_num. Each column is delimited by spaces.

Please refer to Zemax User's Manual for further discussion on system analysis related keywords and functions.

3.10 Non-Sequential Components

As we mentioned in Chapter 1, in many cases, the analysis of an optical system can only be done through non-sequential model, such as illumination system analysis and stray light analysis. For this reason, ZEMAX developed powerful non-sequential analysis tools, and ZPL also provided many related keywords and functions to utilize those tools.

Non-Sequential Component Editor is an important place to define and modify non-sequential optical system. We can add and delete various optical components and modify their parameters here.

If we want to add a component in the Non-Sequential Component Editor, we can use keyword INSERTOBJECT. The syntax is:

INSERTOBJECT surf, object

where surf is the surface number of the non-sequential component, 1 for non-sequential mode; object is the location of the new null object to be placed with value between 1 and current total number of objects + 1, inclusive. If there are other objects after the new object, their object number will be re-ordered.

If we want to delete a component from the Non-Sequential Component Editor, we can use keyword DELETEOBJECT. The syntax is:

DELETEOBJECT surf, object

where surf is the surface number of the non-sequential component, 1 for non-sequential mode; object is the location of the object to be deleted. If there are other objects after it, their object number will be re-ordered.

Usually, after a new component is added, we need to define its space position and other properties. We can use keyword SETNSCPOSITION, SETNSCPROPERTY and SETNSCPARAMETER to do so.

SETNSCPOSITION is used to define space position and tilt of a non-sequential object. The syntax is:

SETNSCPOSITION surface, object, code, value

where surf is the surface number of the non-sequential component, 1 for non-sequential mode; object is the location of the object; code is 1 ~ 6 for x, y, z, tilt-x, tilt-y, and tilt-z, respectively; value is the new value for the specified position.

SETNSCPROPERTY is used to define properties of NSC objects. The syntax is:

SETNSCPROPERTY surface, object, code, face, value

where surf is the surface number of the non-sequential component, 1 for non-sequential mode; object is the location of the object; code, as defined in the table below, is used to specify what property of the object is being modified; face is the face number, 0 if not applicable; value is the new value of the property.

Table 3.10-1 Code for keyword SETNSCPROPERTY

Code	Property
<i>The following codes set values on the NSC Editor.</i>	
1	Sets the object comment.
2	Sets the reference object number.
3	Sets the “inside of” object number.
4	Sets the object material.
<i>The following codes set values on the Type tab of the Object Properties dialog.</i>	
0	Sets the object type. The value should be the name of the object, such as “NSC_SLEN” for the standard lens. The names for each object type are listed in the Prescription Report for each object type in the NSC editor. All NSC object names start with “NSC_”.
13	Sets User Defined Aperture, use 1 for checked, 0 for unchecked.
14	Sets the User Defined Aperture file name.
15	Sets the “Use Global XYZ Rotation Order” checkbox, use 1 for checked, 0 for unchecked.
16	Sets the “Rays Ignore This Object” combo box, use 0 for Never, 1 for Always, and 2 for On Launch.
17	Sets the “Object Is A Detector” checkbox, use 1 for checked, 0 for unchecked.
18	Sets the “Consider Objects” list. The argument should be a string listing the object numbers to consider delimited by spaces, such as “2 5 14”.
19	Sets the “Ignore Objects” list. The argument should be a string listing the object numbers to ignore delimited by spaces, such as “1 3 7”.
20	Sets the “Use Pixel Interpolation” checkbox, use 1 for checked, 0 for unchecked.
30	Sets the “Use Consider/Ignore Objects When Splitting” checkbox, use 1 for checked, 0 for unchecked.
<i>The following codes set values on the Coat/Scatter tab of the Object Properties dialog.</i>	
5	Sets the coating name for the specified face.
6	Sets the profile name for the specified face.
7	Sets the scatter mode for the specified face: 0 = none, 1 = Lambertian, 2 = Gaussian, 3 = ABg, 4 = User Defined, 5 = BSDF, 6 = ABg File, 7 = IS Scatter Catalog.
8	Sets the scatter fraction for the specified face.

9	Sets the number of scatter rays for the specified face.
10	Sets the Gaussian sigma (Gaussian scatter model) or the sample orientation angle (BSDF or IS Scatter Catalog scatter models) for the specified face.
11	Sets the reflect ABg data name for the specified face.
12	Sets the transmit ABg data name for the specified face.
27	Sets the name of the user defined scattering DLL.
21-26	Sets parameter values on the user defined scattering DLL.
28	Sets the name of the user defined scattering data file.
29	Sets the "Face Is" property for the specified face. Use 0 for "Object Default", 1 for "Reflective", and 2 for "Absorbing".
31	Sets the reflect BSDF data file for the specified face. The value should be the name of the BSDF file with no path (i.e. BrownVinyl.bsdf).
32	Sets the transmit BSDF data file for the specified face. The value should be the name of the BSDF file with no path (i.e. BrownVinyl.bsdf).
33	Sets the reflect ABg File data file for the specified face. The value should be the name of the ABGF file with no path (e.g. SampleABGF.abgf).
34	Sets the transmit ABg File data file for the specified face. The value should be the name of the ABGF file with no path (e.g. SampleABGF.abgf).
35	Sets the reflect IS Scatter Catalog data file for the specified face. The value should be the name of the ISX file with no path (e.g. BrownVinyl.ISX).
36	Sets the transmit IS Scatter Catalog data file for the specified face. The value should be the name of the ISX file with no path (e.g. BrownVinyl.ISX).
37	Sets the Thin Window Scattering option for the specified face. Use 0 to turn the option off (i.e. unchecked option in checkbox) and 1 to turn the option on (i.e. checked option in checkbox).
38	Sets the sample side R for IS Scatter Catalog scattering. Use 0 for front and 1 for back.
39	Sets the sample side T for IS Scatter Catalog scattering. Use 0 for front and 1 for back.
40	Sets the sampling R for IS Scatter Catalog scattering. Use 0 for 5 degrees, 1 for 2 degrees, and 2 for 1 degree.
41	Sets the sampling T for IS Scatter Catalog scattering. Use 0 for 5 degrees, 1 for 2 degrees, and 2 for 1 degree.
<i>The following codes set values on the Bulk Scattering tab of the Object Properties dialog.</i>	
81	Sets the "Model" value on the bulk scattering tab. Use 0 for "No Bulk Scattering", 1 for "Angle Scattering", and 2 for "DLL Defined Scattering".
82	Sets the mean free path to use for bulk scattering.
83	Sets the angle to use for bulk scattering.
84	Sets the name of the DLL to use for bulk scattering.
85	Sets the parameter value to pass to the DLL, where the face value is used to specify which parameter is being defined. The first parameter is 1, the second is 2, etc.
86	Sets the wavelength shift string.

The following codes set values on the Diffraction tab of the Object Properties dialog.	
91	Sets the "Split" value on the diffraction tab. Use 0 for "Don't Split By Order", 1 for "Split By Table Below", and 2 for "Split By DLL Function".
92	Sets the name of the DLL to use for diffraction splitting.
93	Sets the Start Order value.
94	Sets the Stop Order value.
95, 96	Sets the parameter values on the diffraction tab. These are the parameters passed to the diffraction splitting DLL as well as the order efficiency values used by the "split by table below" option. The face value is used to specify which parameter is being defined. The first parameter is 1, the second is 2, etc. The code 95 is used for reflection properties, and 96 for transmission.
<i>The following codes set values on the Sources tab of the Object Properties dialog.</i>	
101	Sets the source object random polarization. Use 1 for checked, 0 for unchecked.
102	Sets the source object reverse rays option. Use 1 for checked, 0 for unchecked.
103	Sets the source object Jones X value.
104	Sets the source object Jones Y value.
105	Sets the source object Phase X value.
106	Sets the source object Phase Y value.
107	Sets the source object initial phase in degrees value.
108	Sets the source object coherence length value.
109	Sets the source object pre-propagation value.
110	Sets the source object sampling method; 0 for random, 1 for Sobol sampling.
111	Sets the source object bulk scatter method; 0 for many, 1 for once, 2 for never.
112	Sets the array mode; 0 for none, 1 for rectangular, 2 for circular, 3 for hexapolar, and 4 for hexagonal.
113	Sets the source color mode. For a complete list of the available modes, see "Defining the color and spectral content of sources" on page 403. The source color modes are numbered starting with 0 for the System Wavelengths, and then from 1 through the last model listed in the dialog box control.
114-116	Sets the number of spectrum steps, start wavelength, and end wavelength, respectively.
117	Sets the name of the spectrum file.
161-162	Sets the array mode integer arguments 1 and 2.
165-166	Sets the array mode double precision arguments 1 and 2.
181-183	Sets the source color mode arguments, for example, the XYZ values of the Tristimulus.
<i>The following codes set values on the Grin tab of the Object Properties dialog.</i>	
121	The following codes set values on the Grin tab of the Object Properties dialog.
122	Sets the Maximum Step Size value.
123	Sets the DLL name.
124	Sets the Grin DLL parameters. These are the parameters passed to the DLL. The face value is used to specify which parameter is being defined. The first parameter

	is 1, the second is 2, etc.
<i>The following codes set values on the Draw tab of the Object Properties dialog.</i>	
141	Sets the do not draw object checkbox. Use 1 for checked, 0 for unchecked.
142	Sets the object opacity. Use 0 for 100%, 1 for 90%, 2 for 80%, etc.
<i>The following codes set values on the Scatter To tab of the Object Properties dialog.</i>	
151	Sets the scatter to method. Use 0 for scatter to list, and 1 for importance sampling.
152	Sets the Importance Sampling target data. The argument should be a string listing the ray number, the object number, the size, and the limit value, all separated by spaces.
153	Sets the "Scatter To List" values. The argument should be a string listing the object numbers to scatter to delimited by spaces, such as "4 6 19".
<i>The following codes set values on the Birefringence tab of the Object Properties dialog.</i>	
171	Sets the Birefringent Media checkbox. Use 0 for unchecked, and 1 for checked.
172	Sets the Birefringent Media Mode. Use 0 for Trace ordinary and extraordinary rays, 1 for Trace only ordinary rays, 2 for Trace only extraordinary rays, and 3 for Waveplate mode.
173	Sets the Birefringent Media Reflections status. Use 0 for Trace reflected and refracted rays, 1 for Trace only refracted rays, and 2 for Trace only reflected rays.
174-176	Sets the Ax, Ay, and Az values.
177	Sets the Axis Length.
<i>The following codes do not set values, but are included here to return values for the function NPRO.</i>	
200	Used by function NPRO to determine the index of refraction of an object. The syntax is NPRO(surface, object, 200, wavenumber)
201-203	Used by function NPRO to determine the nd (201), vd (202), and dpgf (203) parameters of an object using a model glass. The syntax is NPRO(surface, object, 201, 0)

SETNSCPARAMETER is used to set the parameter values of any object in the NSC editor. The syntax is:

SETNSCPARAMETER surface, object, parameter, value

This keyword requires 3 numeric expressions that evaluate to integers specifying the non-sequential component's surface number (1 for total non-sequential system), the object number, and the parameter number. The fourth argument is the new value for the specified parameter.

ZPL also defined a series of functions to read various parameters of NSC components.

If we want to know the total number of non-sequential components in a certain surface, we can use function `NOBJ(surface)`, where `surface` is the surface number. The return value is the total number of NSC components in the given surface.

If we want to read the position and tilt of a certain object in a surface, we can use function `NPOS(surf, object, code)`, where `surf` is the surface number, `object` is the object number, and `code` is 1~6 for x, y, z, tilt x, tilt y, tilt z, respectively. The return value is the value responsible for the code.

Function `NPRO(surf, object, code, face)` is used to read the properties of a given NSC component, where `surf` is the surface number, `object` is the object number, `code` is as described in table 3.10-1, `face` is the face number on a component. The return value is the property associated to the code, and can be either numerical value or string. If the return value is string, we can use function `$buffer()` to read the string from the return value.

Function `NPAR(surf, object, param)` is used to read the parameter column value in the NSC editor, where `surf` is the surface number, `object` is the object number, and `param` is the parameter number.

We will give some examples to show the applications of some keywords and functions discussed above.

Example 3.10-1 shows how to add, delete object and set parameters in the NSC editor.

```

1 ! ex31001
2 ! This program shows how to Set NSC parameters
3 ! Assume the mode is total Non-Sequential Mode
4
5 ! this part is used to clean the NSC editor
6 totalObjNum = NOBJ(1)
7 for i, totalObjNum, 1, -1
8   DELETEOBJECT 1, i      # delete all the objects
9 next                    # there is still a null object in the editor at last
10                       # because the editor cannot be empty
11
12 ! add two more objects in the editor, so the total is 3
13 INSERTOBJECT 1, 1
14 INSERTOBJECT 1, 1
15
16 ! define the type of the first object
17 surface = 1
18 object = 1
19 code = 0
20 face = 0
21 value$ = "NSC_SLEN"
22 SETNSCPROPERTY surface, object, code, face, value$
23
24 ! define the position of the first object
25 SETNSCPOSITION 1, 1, 1, 0      # x
26 SETNSCPOSITION 1, 1, 2, 0      # y
27 SETNSCPOSITION 1, 1, 3, 0      # z
28 SETNSCPOSITION 1, 1, 4, 0      # x tilt
29 SETNSCPOSITION 1, 1, 5, 0      # y tilt
30 SETNSCPOSITION 1, 1, 6, 0      # z tilt
31
32 ! define the comment of the first object
33 SETNSCPROPERTY 1, 1, 1, 0, "first lens"
34
35 ! define the material of the first object
36 SETNSCPROPERTY 1, 1, 4, 0, "BK7"
37
38 ! define curvature, clear aperture, edge and thickness of the first object
39 SETNSCPARAMETER 1, 1, 1, 10      # radius 1
40 SETNSCPARAMETER 1, 1, 3, 1.8     # clear 1
41 SETNSCPARAMETER 1, 1, 4, 2      # edge 1
42 SETNSCPARAMETER 1, 1, 5, 1      # thickness
43 SETNSCPARAMETER 1, 1, 6, -10     # radius 2
44 SETNSCPARAMETER 1, 1, 8, 1.8     # clear 2
45 SETNSCPARAMETER 1, 1, 9, 2      # edge 2
46

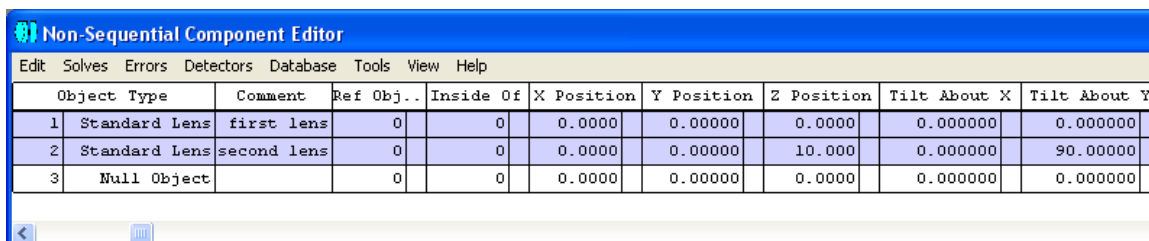
```

```

46
47 ! define the type of the second object
48 SETNSCPROPERTY 1, 2, 0, 0, "NSC_SLEN"
49
50 ! define the comment of the second object
51 SETNSCPROPERTY 1, 2, 1, 0, "second lens"
52
53 ! define the position of the second object
54 SETNSCPOSITION 1, 2, 3, 10      # z position
55 SETNSCPOSITION 1, 2, 5, 90     # rotate 90 degrees around y
56
57 ! define the material of the second object
58 SETNSCPROPERTY 1, 2, 4, 0, "BK7"
59
60 ! define curvature, clear aperture, edge and thickness of the second object
61 SETNSCPARAMETER 1, 2, 1, 10     # radius 1
62 SETNSCPARAMETER 1, 2, 3, 1.8   # clear 1
63 SETNSCPARAMETER 1, 2, 4, 2     # edge 1
64 SETNSCPARAMETER 1, 2, 5, 1     # thickness
65 SETNSCPARAMETER 1, 2, 6, -10   # radius 2
66 SETNSCPARAMETER 1, 2, 8, 1.8   # clear 2
67 SETNSCPARAMETER 1, 2, 9, 2     # edge 2
68

```

In this example, we first cleared the NSC editor by deleting objects (lines 6~9). Please note that after clearance, there is still a null object in the editor. Then we inserted two null objects in the editor (lines 13~14), so the total number of objects in the editor is 3. We then defined the first object as standard lens (lines 17~22), set its position and tilt (lines 17~22), give the comment (line 33), and set the material (line 36). We then set the parameters of the lens (lines 39~45), i.e. first surface curvature radius, first surface effective half diameter, first surface edge half diameter, thickness, second surface curvature radius, second surface effective half diameter, and second surface edge half diameter. Similarly, we defined the second object (lines 47~67). In fact, after defining the object type, ZEMAX will automatically generate some default properties, so we only need to modify those different from the default values. For example, we only defined z position and y tilt of the second object, and omitted x position, y position, x tilt, and z tilt. After we run program ex31001.ZPL, the content in the NSC editor will be updated, as shown in figure 3.10-1:



Object	Type	Comment	Ref Obj.	Inside Of	X Position	Y Position	Z Position	Tilt About X	Tilt About Y
1	Standard Lens	first lens	0	0	0.0000	0.00000	0.0000	0.000000	0.000000
2	Standard Lens	second lens	0	0	0.0000	0.00000	10.000	0.000000	90.00000
3	Null Object		0	0	0.0000	0.00000	0.0000	0.000000	0.000000

Fig. 3.10-1: content of NSC editor after running program ex31001.ZPL

If we open 3D Layout window, we can see the two lenses we defined. Their tilt is different, as shown in figure 3.10-2:

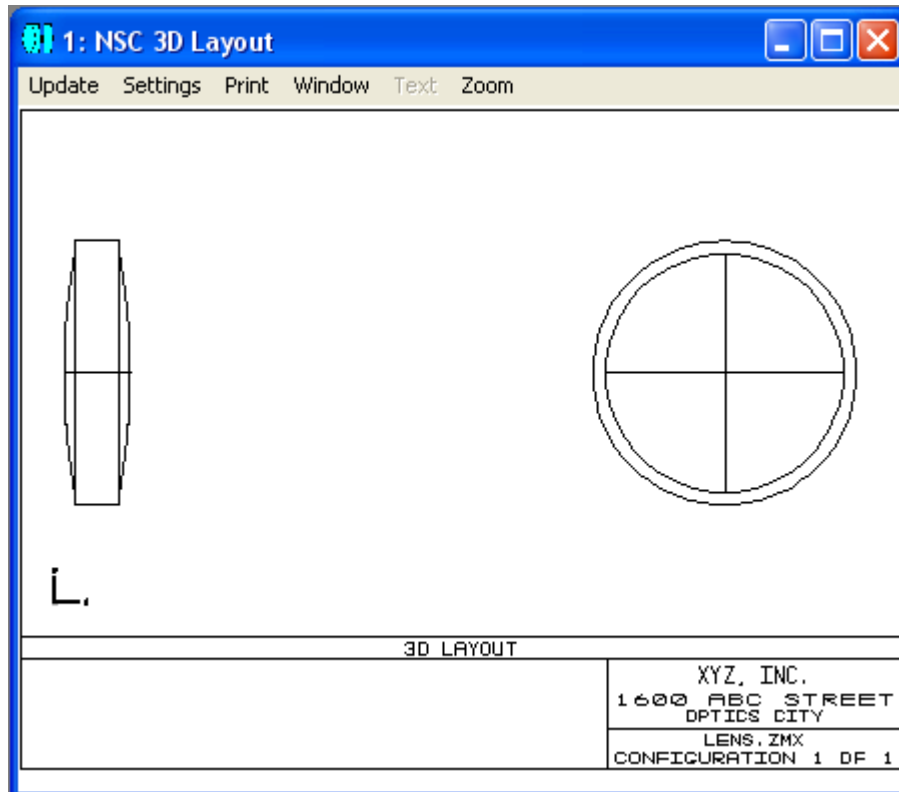


Fig. 3.10-2: content of 3D Layout window after running program ex31001.ZPL

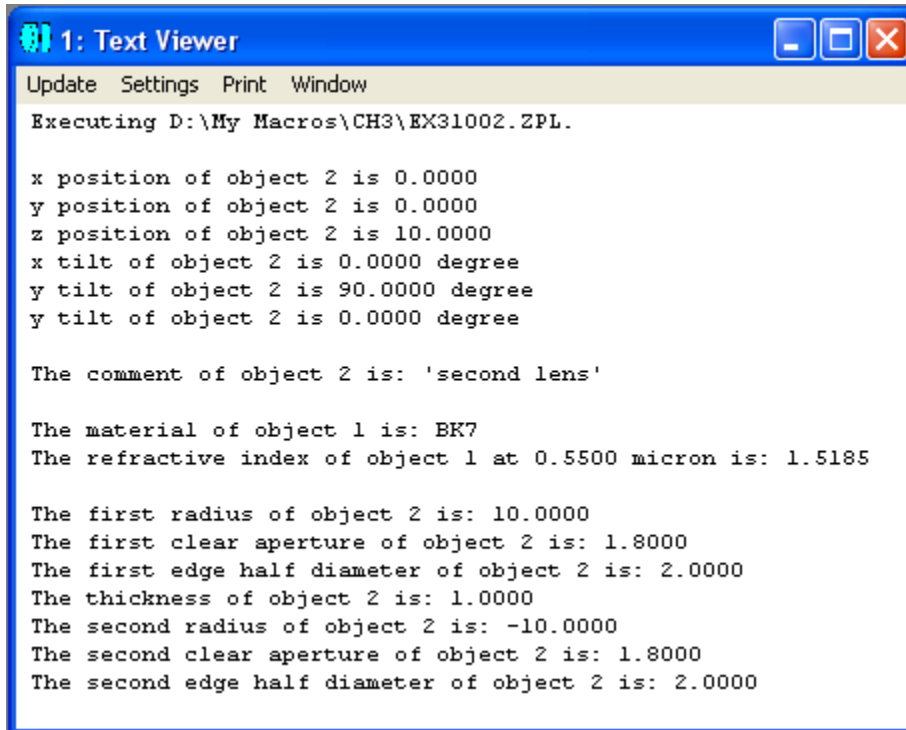
Example 3.10-2 shows how to read parameters of NSC objects.

```

1 ! ex31002
2 ! This program shows how to Read NSC parameters
3 ! Assume the objects are defined in ex31001
4
5 ! read the position
6 x = NPOS(1, 2, 1)
7 PRINT
8 PRINT "x position of object 2 is ", x
9 PRINT "y position of object 2 is ", NPOS(1, 2, 2)
10 PRINT "z position of object 2 is ", NPOS(1, 2, 3)
11 PRINT "x tilt of object 2 is ", NPOS(1, 2, 4), " degree"
12 PRINT "y tilt of object 2 is ", NPOS(1, 2, 5), " degree"
13 PRINT "y tilt of object 2 is ", NPOS(1, 2, 6), " degree"
14
15 ! read the comment
16 dummy = NPRO(1, 2, 1, 0)
17 a$ = $buffer()
18 PRINT
19 PRINT "The comment of object 2 is: '", a$, "'"
20
21 ! read the material
22 dummy = NPRO(1, 2, 4, 0)
23 a$ = $buffer()
24 PRINT
25 PRINT "The material of object 1 is: ", a$
26 PRINT "The refractive index of object 1 at ",
27 PRINT WAVL(1), " micron is: ", NPRO(1,1,200,1)
28
29 ! read the parameters
30 PRINT
31 PRINT "The first radius of object 2 is: ", NPAR(1,2,1)
32 PRINT "The first clear aperture of object 2 is: ", NPAR(1,2,3)
33 PRINT "The first edge half diameter of object 2 is: ", NPAR(1,2,4)
34 PRINT "The thickness of object 2 is: ", NPAR(1,2,5)
35 PRINT "The second radius of object 2 is: ", NPAR(1,2,6)
36 PRINT "The second clear aperture of object 2 is: ", NPAR(1,2,8)
37 PRINT "The second edge half diameter of object 2 is: ", NPAR(1,2,9)

```

In this program, we assume the optical system is the two lens system defined in example 3.10-1. The position and tilt of an object can be read through function NPOS(), as shown in lines 6~13 in the program. Please note that we can either assign the return value to a middle variable, and use the variable when needed, as shown in lines 6 and 8, or directly obtain the return value by calling the function, as shown in lines 9~13. In the program, the object properties were also read through function NPRO(), and object parameters were read through function NPAR(). Please note that when using function NPRO(), if the return value is string, the result can be saved in a temporary variable (line 16), and then read out through function \$buffer() (line 17). The result of the program is shown in figure 3.10-3:



```
1: Text Viewer
Update Settings Print Window
Executing D:\My Macros\CH3\EX31002.ZPL.

x position of object 2 is 0.0000
y position of object 2 is 0.0000
z position of object 2 is 10.0000
x tilt of object 2 is 0.0000 degree
y tilt of object 2 is 90.0000 degree
y tilt of object 2 is 0.0000 degree

The comment of object 2 is: 'second lens'

The material of object 1 is: BK7
The refractive index of object 1 at 0.5500 micron is: 1.5185

The first radius of object 2 is: 10.0000
The first clear aperture of object 2 is: 1.8000
The first edge half diameter of object 2 is: 2.0000
The thickness of object 2 is: 1.0000
The second radius of object 2 is: -10.0000
The second clear aperture of object 2 is: 1.8000
The second edge half diameter of object 2 is: 2.0000
```

Fig. 3.10-3: result of program ex31002.ZPL

In non-sequential system, light source and detector are two very important objects. Example 3.10-3 shows how to set source and detector in ZPL program.

```
1 ! ex31003
2 ! This program shows how to set NSC sources and detectors
3
4 ! this part is used to clean the NSC editor
5 for i, NOBJ(1), 1, -1
6   DELETEOBJECT 1, i      # delete all the objects
7 next                    # there is still a null object in the editor at last
8
9 ! add two more objects in the editor, so the total is 3
10 INSERTOBJECT 1, 1
11 INSERTOBJECT 1, 1
12
13 ! define the type of the first object as source ellipse (circular)
14 SETNSCPROPERTY 1, 1, 0, 0, "NSC_SRCE"
15
16 ! define the position
17   # this part is omitted. Use the default setting.
18
19 ! define parameters of the source
20 SETNSCPARAMETER 1, 1, 1, 100      # number of layout rays
21 SETNSCPARAMETER 1, 1, 2, 10000   # number of analyze rays
22 SETNSCPARAMETER 1, 1, 3, 1      # power in Watts
23 SETNSCPARAMETER 1, 1, 6, 1      # x half width
24 SETNSCPARAMETER 1, 1, 7, 1      # y half width
25 SETNSCPARAMETER 1, 1, 8, 0      # source distance, 0 for collimated
26
```

```

26
27 ! define the type of the second object as a standard lens
28 SETNSCPROPERTY 1, 2, 0, 0, "NSC_SLEN"
29
30 ! define the position
31 SETNSCPOSITION 1, 2, 3, 10      # z position
32
33 ! define the material
34 SETNSCPROPERTY 1, 2, 4, 0, "BK7"
35
36 ! define curvature, clear aperture, edge and thickness of the lens
37 SETNSCPARAMETER 1, 2, 1, 10     # radius 1
38 SETNSCPARAMETER 1, 2, 3, 1.8   # clear 1
39 SETNSCPARAMETER 1, 2, 4, 2     # edge 1
40 SETNSCPARAMETER 1, 2, 5, 1     # thickness
41 SETNSCPARAMETER 1, 2, 6, -10   # radius 2
42 SETNSCPARAMETER 1, 2, 8, 1.8   # clear 2
43 SETNSCPARAMETER 1, 2, 9, 2     # edge 2
44
45 ! define the type of the third object as a detector rect
46 SETNSCPROPERTY 1, 3, 0, 0, "NSC_DETE"
47
48 ! define the position
49 SETNSCPOSITION 1, 3, 3, 20     # z position
50
51 ! define the material
52 SETNSCPROPERTY 1, 3, 4, 0, "ABSORB" # assuming the detector is absorbing
53
54 ! define parameters of the detector
55 SETNSCPARAMETER 1, 3, 1, 1     # x half width
56 SETNSCPARAMETER 1, 3, 2, 1     # y half width
57 SETNSCPARAMETER 1, 3, 3, 100   # number of x pixels
58 SETNSCPARAMETER 1, 3, 4, 100   # number of y pixels

```

In this program, we first cleared the NSC editor (lines 5~7), and inserted two null objects (lines 10~11), so the total objects in the editor is 3. We set the first object as source ellipse (line 14). Its position is default (0, 0, 0). We then defined its parameters (lines 20~25). After that, we set the second object as standard lens (line 28), and defined its z position (line 31), material (line 34), and other parameters (lines 37~43). Similarly, we set the third object as detector and defined its properties and parameters (lines 46~58).

After running program ex31003.ZPL, if we open 3D Layout window, we can see the three objects defined in the program, and can also see the display rays, as shown in figure 3.10-4:

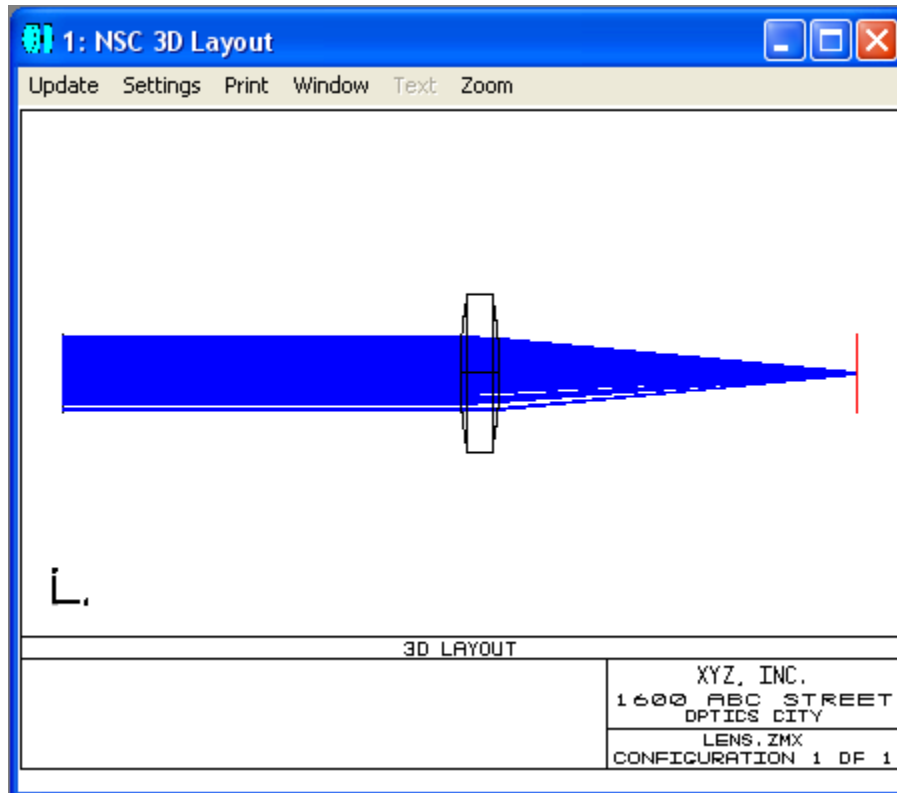


Fig. 3.10-4: content of 3D Layout window after running program ex31003.ZPL

Sometimes after modifying and analyzing old NSC objects, we need to recover old objects. This can be done with keyword RELOADOBJECTS. The syntax is:

RELOADOBJECTS surf, object

where surf is surface number, 1 for NSC mode; object is object number, 0 for reloading all objects.

Similar to sequential system, in non-sequential system, ray tracing is a very important function. ZPL provided keyword NSTR and function NSDC() and NSDD() to perform ray tracing and read out detector result.

The syntax of keyword NSTR is:

NSTR surf, source, split, scatter, usepol, ignore_err, rand_seed, save, filename\$, filter

where surf is surface number, 1 for NSC mode; source is the object number for the source used in ray tracing, 0 for all the sources in the current system; split is non-zero for splitting on, 0 for off; scatter is non-zero for scattering on, 0 for off; usepol is non-zero for polarization on, 0 for off; ignore_err is non-zero for ignoring error, 0 for terminating ray-tracing and reporting error; rand_seed is non-zero integer for seeding the random generator with the given integer each time, 0 for seeding the random generator with a random number; save is zero for not saving the result, in which case the arguments after it can be omitted, otherwise if save is non-zero, the ray-tracing result will be saved in a ZRD file in the same folder as lens file, with name defined by filename\$ (not including path) and extension name as ZRD; filter\$ is optional with different ray filters defined in table 3.10-2. In newer versions of ZEMAX, another argument zrd_format is required after filter. Please refer to ZEMAX User Manual for details.

NSTR always calls UPDATE before tracing rays to make certain all objects are correctly loaded and updated.

Table 3.10-2: Filter String Flags

Flag	Description
<code>_n</code>	Filters that start with an underscore “_” followed by an integer code are defined and used by the Path Analysis feature;
<code>~nnmm...[#]</code>	Ray path filter. This filter selects ray branches whose segments follow an explicit path. The first object number defined is the source object, followed by each object the rays interact with, in order. Each object number must be defined by a three digit integer, with leading zeros added if required. A ray that leaves source object 7, then hits objects 18 and then 9 and is then terminated or hits no other object can be selected using the filter “~007018009”. Multiple consecutive object numbers are not redundantly defined; for example, if a ray hits the front face of lens 9 and then hits the back face of the same object 9 need only be listed as a single 009 in the filter definition. Optionally, the filter may be terminated with the # symbol, which indicates any segment that initially follows this path is selected. This allows rays which hit different objects, or no objects at all, after the last object listed are still selected. The maximum number of objects in any one filter is 50.
<code>\$nnmm...[#]</code>	Ray path filter alternate form. This is identical to the “~” filter above, except two digit codes are used instead of three digit codes for the object numbers. This is a more convenient form if the number of objects is less than 100.

Bn	Ray bulk scattered inside of object n. If the n value is 0, then bulk scattered segments from any object will return true for this test.
Dn	Ray diffracted after striking object n. See En.
En	Diffracted from parent segment's object n. This flag only gets set for ray segments split from diffractive elements, for order numbers other than zero, when ray splitting is on.
Fn	Scattered from parent segment's object n. This flag only gets set for ray segments split from scattering surfaces when ray splitting is on. The specular segment does not get this flag, only scattered segments. If the n value is 0, then scattered segments from any object will return true for this test.
Gn	Ghost reflected from parent segment's object n. This flag only gets set for ray segments reflected from refractive objects when ray splitting is on. If the n value is 0, then ghost segments from any object will return true for this test.
Hn	Ray hit object n. To test whether a ray hit an object, the flag is of the form Hn. For example, to test if a ray hit object 5, the flag would be H5. See Ln.
Jn	Similar to Gn, except that all segments prior to the ghost reflection point are set to have zero intensity. This allows Detector Viewers to look only at ghost energy, not direct incident energy, even if the ray later ghosted off another object. The zero intensity values will only affect the Detector Viewer, not the ray database viewer or layouts.
Ln	Ray hit object n last. To test whether the last segment of a ray branch hit an object, the flag is of the form Ln. For example, to test if the last segment of a ray branch hit object 5, the flag would be L5. See Hn.
Mn	Ray missed object n. To test whether a ray missed an object, the flag is of the form Mn. For example, to test if a ray missed object 15, the flag would be M15.
On	Ray originated at source number n. O0 (that is "O" as in Origin and "0" as number zero) will select all sources.
Rn	Ray reflected after striking object n. The flag R7 would test if the ray reflected after striking object 7. See Gn.
Sn	Ray scattered after striking object n. This tests the "S" flag as listed in the ZRD file, which refers to scattering at the point a ray strikes an object. See also Fn and X_SCATTER.
Tn	Ray transmitted (refracted) in to or out of object n. The flag T4 would test if the ray refracted in or out of object 4 after striking the object.
Wn	Ray uses wavelength n. If the n value is 0, then rays with any wavelength will return true for this test. Note this filter only tests the initial wavelength for the ray as it leaves the source. If wavelength shifting is used, the wavelength may change during propagation.
X_AXYG(n,v)	Ray has incident angle (in degrees) on object n in the local x-y plane greater than v. The angle is measured with respect to the +y direction without regard to the direction of propagation. If the ray never strikes object n, this flag is false.
X_AXYL(n,v)	Ray has incident angle (in degrees) on object n in the local x-y plane less than v. The angle is measured with respect to the +y direction without regard to the direction of propagation. If the ray never strikes object n, this flag is false.

X_AXZG(n,v)	Ray has incident angle (in degrees) on object n in the local x-z plane greater than v. The angle is measured with respect to the +z direction without regard to the direction of propagation. If the ray never strikes object n, this flag is false.
X_AXZL(n,v)	Ray has incident angle (in degrees) on object n in the local x-z plane less than v. The angle is measured with respect to the +z direction without regard to the direction of propagation. If the ray never strikes object n, this flag is false.
X_AYZG(n,v)	Ray has incident angle (in degrees) on object n in the local y-z plane greater than v. The angle is measured with respect to the +z direction without regard to the direction of propagation. If the ray never strikes object n, this flag is false.
X_AYZL(n,v)	Ray has incident angle (in degrees) on object n in the local y-z plane less than v. The angle is measured with respect to the +z direction without regard to the direction of propagation. If the ray never strikes object n, this flag is false.
X_EXT(n,b)	Ray segment is an extraordinary ray generated from a birefringent interface after the parent ray has hit object n exactly b times. To apply this filter, a search is made for the parent segment that hit object n exactly b times, and only the children of that particular parent segment are considered. If no parent segment hit object n exactly b times, the filter returns false. See also X_ORD.
X_GHOST(n,b)	Ray segment has ghosted exactly b times, and has hit object n at least once. If n is zero, any ray segment that has ghosted b times will pass the test. For example, to consider only all second generation ghosts (ghost rays from ghost parents), use X_GHOST(0, 2). X_GHOST does not consider ghost ray segments that end in a TIR condition; although rays that TIR are considered ghosts. For example, if a third generation ghost ray leaves one surface, strikes another surface, and then TIR's from this second surface, X_GHOST(0, 3) will not include this segment because the segment ended in a TIR and not a ray termination (the ray reflected and continued). This same segment will however be included in the filter X_GHOST(0, 4) because the ray ghosted a fourth time (at the TIR point). This is an artifact of how Zemax defines segments and counts ghost rays. In all cases, all ghost rays can be found if sufficiently high values of b are tested. Note rays which TIR from refractive surfaces are considered ghosts, but rays reflected from mirror surfaces are not. See also Gn.
X_HIT(n,b)	Ray segment has hit object n exactly b times. See also Hn, X_HITS, X_HITFACE, and X_HITFACE2.
X_HITS(n1,n2)	If n2 is zero: Ray has n1 or more hits on any object(s). If n2 is not zero, then Ray has between n1 and n2 hits, inclusive.
X_HITFACE(n,f)	Ray segment has hit object n on face f. See also Hn, X_HIT, and X_HITFACE2.
X_HITFACE(n,f)	Ray segment has hit object n on face f. See also Hn, X_HIT, and X_HITFACE2.
X_HITFACE2(n,f,b)	Ray segment has hit object n on face f exactly b times. See also Hn and X_HIT.
X_IAGT(n,v)	Ray has absolute intensity greater than value v on object n. If the ray never strikes object n, this flag is false.
X_IALT(n,v)	Ray has absolute intensity less than value v on object n. If the ray never strikes

	object n, this flag is false.
X_IRGT(n,v)	Ray has intensity relative to initial intensity greater than value v on object n. If the ray never strikes object n, this flag is false.
X_IRLT(n,v)	Ray has intensity relative to initial intensity less than value v on object n. If the ray never strikes object n, this flag is false.
X_LGT(n,v)	Ray has local incident x ray direction cosine greater than value v at point on object n. If the ray never strikes object n, this flag is false.
X_LLT(n,v)	Ray has local incident x ray direction cosine less than value v at point on object n. If the ray never strikes object n, this flag is false.
X_MGT(n,v)	Ray has local incident y ray direction cosine greater than value v at point on object n. If the ray never strikes object n, this flag is false.
X_MLT(n,v)	Ray has local incident y ray direction cosine less than value v at point on object n. If the ray never strikes object n, this flag is false.
X_NGT(n,v)	Ray has local incident z ray direction cosine greater than value v at point on object n. If the ray never strikes object n, this flag is false.
X_NLT(n,v)	Ray has local incident z ray direction cosine less than value v at point on object n. If the ray never strikes object n, this flag is false.
X_ORD(n,b)	Ray segment is an ordinary ray generated from a birefringent interface after the parent ray has hit object n exactly b times. To apply this filter, a search is made for the parent segment that hit object n exactly b times, and only the children of that particular parent segment are considered. If no parent segment hit object n exactly b times, the filter returns false. See also X_EXT.
X_SCATTER(n,b)	Ray segment has scattered from parent exactly b times, and has hit object n at least once. If n is zero, any child ray segment split off from the parent ray that has scattered b times will pass the test. For example, to consider only first generation scatter rays, use X_SCATTER(0, 1). This filter tests only the scatter from parent or "F" flag as listed in the ZRD. See also Sn and X_SCATTERF.
X_SCATTERF(n,b)	Ray segment has scattered from object n after the parent of the segment hit object n exactly b times. To apply this filter, a search is made for the parent segment that hit object n exactly b times, and only that particular parent segment is considered. If no parent segment hit object n exactly b times, the filter returns false. For example, to consider only scattered rays that branch off from the parent ray after the third hit on object 5 (that is, the ray leaving the source has twice before hit this same object), use X_SCATTERF(5, 3). See also Fn and X_SCATTER.
X_SEGMENTS(n1,n2)	If n2 is zero: Ray has n1 or more segments. If n2 is not zero, then Ray has between n1 and n2 segments, inclusive.
X_WAVERANGE(n, a, b)	Ray has hit object n and has a wavelength between a and b micrometers, inclusive.
X_WAVESHIFT(i,j)	Ray has wave shifted during a bulk scatter event from wavelength i to wavelength j.
X_XGT(n,v)	Ray has local x coordinate greater than value v at point on object n. If the ray never strikes object n, this flag is false.
X_XLT(n,v)	Ray has local x coordinate less than value v at point on object n. If the ray never strikes object n, this flag is false.

X_YGT(n,v)	Ray has local y coordinate greater than value v at point on object n. If the ray never strikes object n, this flag is false.
X_YLT(n,v)	Ray has local y coordinate less than value v at point on object n. If the ray never strikes object n, this flag is false.
X_ZGT(n,v)	Ray has local z coordinate greater than value v at point on object n. If the ray never strikes object n, this flag is false.
X_ZLT(n,v)	Ray has local z coordinate less than value v at point on object n. If the ray never strikes object n, this flag is false.
Z	Ray has fatal error.

The syntax of function NSDC() is:

$$\text{returnValue} = \text{NSDC}(\text{surf}, \text{object}, \text{pixel}, \text{data})$$

where surf is NSC surface number, 1 for total NSC mode; object is the object number of the detector; pixel is the pixel number on the detector, 0 for the summation of all the pixels; data is 0 for real, 1 for imaginary, 2 for the amplitude, and 3 for the coherent intensity. The returned result is saved in variable returnValue.

The syntax of function NSDD () is:

$$\text{returnValue} = \text{NSDD}(\text{surf}, \text{object}, \text{pixel}, \text{data})$$

where surf is NSC surface number, 1 for total NSC mode; object is the object number of the detector; pixel is the pixel number on the detector, 0 for the summation of all the pixels; data is 0 for flux, 1 for flux/area, 2 for flux/solid angle pixel, and 3 for normalized flux. If the object number is zero, then all detectors are cleared and the function returns zero. If the object number is less than zero, then the detector defined by the absolute value of the object number is cleared and the function returns zero. If the object number corresponds to a detector rectangle, surface, or volume object, then the incoherent intensity data from the specified pixel is returned. For a full discussion of the pixel and data arguments, please refer to Zemax User's Manual.

Sometimes if we want to set or modify the coherent or incoherent intensity data of a pixel on a rectangle detector, we can use keyword SETDETECTOR to realize. The syntax is:

$$\text{SETDETECTOR surf}, \text{object}, \text{pixel}, \text{datatype}, \text{value}$$

where surf is NSC surface number, 1 for total NSC mode; object is the object number of the rectangle detector; pixel is the pixel number on the detector, between 1 and the maximum number of pixels; Datatype is 0 for incoherent intensity, 1 for incoherent intensity in angle space, 2 for coherent real part, 3 for coherent imaginary part, and 4 for coherent amplitude 0; value is the value to be set.

Now let's give some examples to show how to do non-sequential ray tracing in ZPL programs.

Example 3.10-4: Ray tracing in non-sequential system

```
1 ! ex31004
2 ! This program shows how to do ray tracing in NSC mode
3 ! Assume the lens system is defined in ex31003
4
5 surf = 1
6 source = 1
7 split = 1
8 scatt = 1
9 pol = 1
10 ignore_err = 1
11 random_seed = 1
12 save = 0
13 object = 0
14 obj = 3
15 pix = 0
16 data = 0
17
18 temp = NSDD(surf, object, pix, data)      # clear the detector
19 NSTR surf, source, split, scatt, pol, ignore_err, random_seed, save
20
21 PRINT
22 PRINT "Reading on detector is:", NSDD(surf, obj, pix, data)
23
24 NSTR surf, source, split, scatt, pol, ignore_err, random_seed, save
25 PRINT "Reading on detector without clearing: ", NSDD(surf, obj, pix, data)
26
27 temp = NSDD(1, 0, 0, 0)                  # clear the detector
28 NSTR surf, source, split, scatt, pol, ignore_err, random_seed, save
29 PRINT "Reading on detector should be: ", NSDD(surf, obj, pix, data)
```

In this example, detector is cleaned in line 18, ray tracing is done in line 19, and light intensity on the detector is read and printed out in line 22. The whole ray tracing process is this simple. However, it needs to be noticed that if the previous values on the detector are not cleared, the new values on the detector after ray tracing will include previous value. Lines 24~29 in the program compared the light intensity value on the detector after ray tracing, without and with clearing the detector. The result of the program is shown in figure 3.10-5:

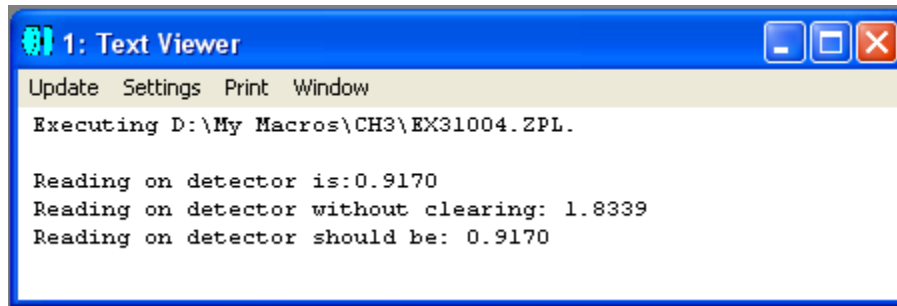


Fig. 3.10-5: Result of program ex31004.ZPL

If we open the detector viewer, we can see the light distribution on the detector, as shown in figure 3.10-6:

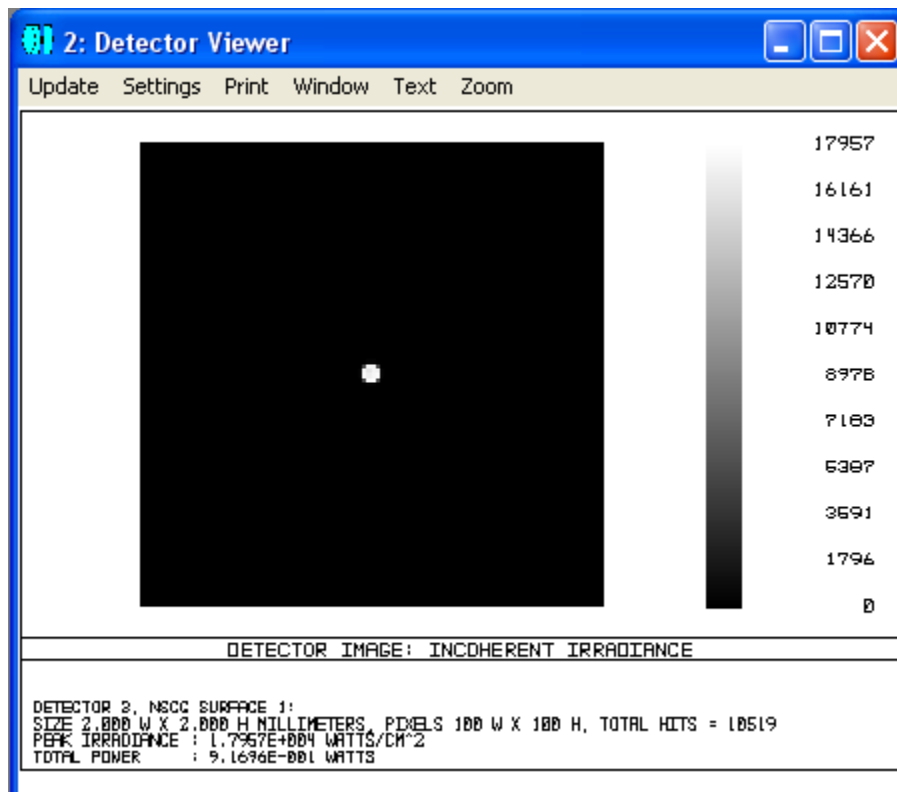


Fig. 3.10-6: Light intensity distribution seen on the detector after ray tracing.

Example 3.10-5: Modify the values on the detector in a non-sequential system.

```

1 ! ex31005
2 ! This program shows how to use SETDETECTOR key word
3 ! Assume the lens system is defined in ex31003
4
5 surf = 1
6 object = 3
7 datatype = 0
8
9 temp = NSDD(1, 0, 0, 0) # clear the detector
10 NSTR 1, 1, 1, 1, 1, 1, 1, 0 # ray tracing, as in ex31004
11
12 PRINT
13 PRINT "Reading on detector: ", NSDD(surf, object, 0, datatype)
14
15 radius = 25
16 width = 2
17 value = 0.05
18
19 FOR i, 1, 100, 1
20   FOR j, 1, 100, 1
21     r2 = (i-50)*(i-50)+(j-50)*(j-50)
22     ra = (radius+width/2)
23     rb = (radius-width/2)
24     IF (r2<ra*ra) & (r2>rb*rb)
25       pix = i*100+j
26       SETDETECTOR surf, object, pix, datatype, value
27     ENDIF
28   NEXT
29 NEXT
30
31 PRINT "Modified reading on detector: ", NSDD(surf, object, 0, datatype)

```

In this example, we assume the system is the same as previous two examples. The detector is cleared in line 9, ray tracing is done in line 10. The reading on the detector is the same as that in example 3.10-4. Then, we added a circular ring around the center of the detector by comparing the distance of each pixel to the center. If a pixel is on the circumference of the ring, then its value is modified. The result of the program is shown in figure 3.10-7:

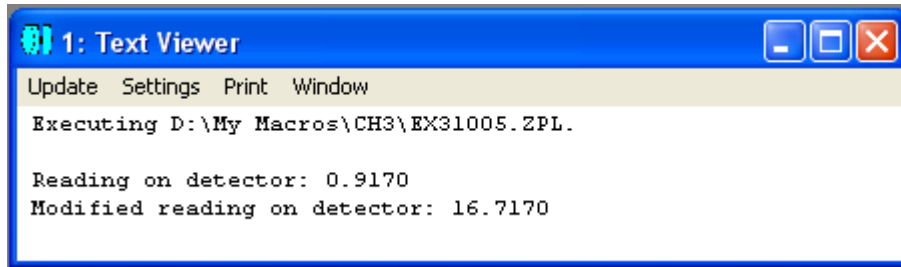


Figure 3.10-7: result of program ex31005.ZPL

The result shows that the values on the detector have been modified. If we open the detector viewer, we can see the circle we added, as shown in figure 3.10-8:

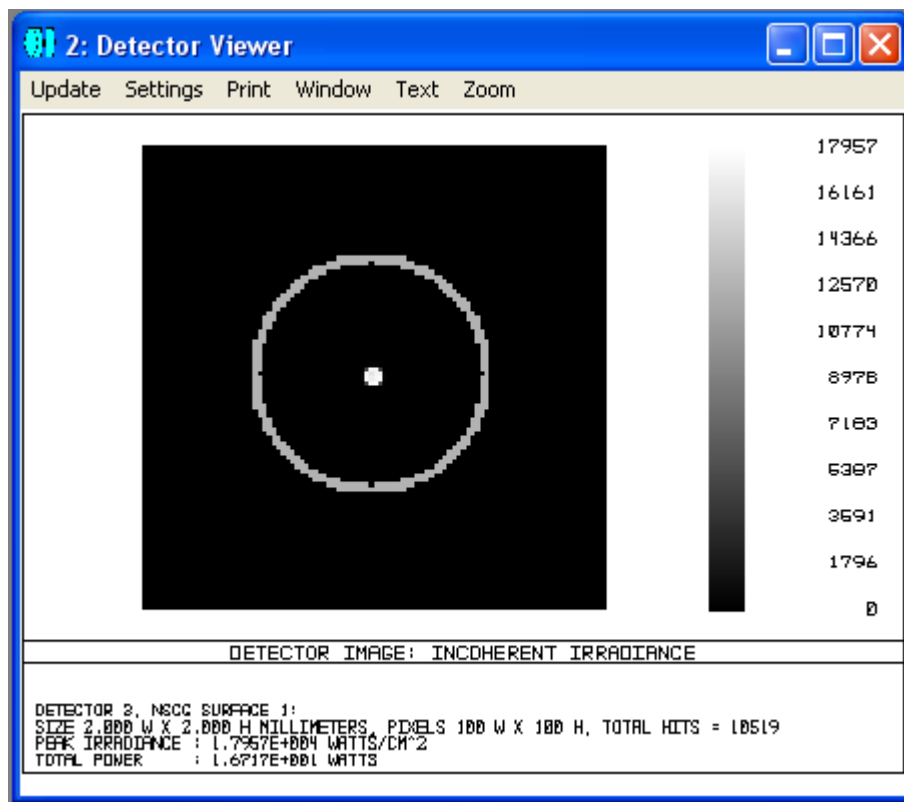


Fig. 3.10-8: : Light intensity distribution seen on the detector after modification.

3.11 Multi-Configuration

In optical design, often times we will need to add, delete or modify some components or parameters based on different working environment, such as designing focal length adjustable lenses, optimizing optical system at different wavelengths, etc. Multi-configuration supported by ZEMAX can be widely used in those cases. In this section, we will discuss multi-configuration related commands in ZPL. For details on multi-configuration, please refer to Zemax User's Manual.

We know that multi-configuration editor is the place to set and modify multi-configuration in Zemax. If we want to add, delete or modify configurations or operands in multi-configuration editor, we can use ZPL keywords INSERTCONFIG, INSERTMCO, DELETECONFIG, and DELETEMCO.

INSERTCONFIG is used to add a configuration in the multi-configuration editor. The syntax is:

INSERTCONFIG config

where config is an integer greater than 0 and smaller than or equal to the current number of configurations plus 1.

INSERTMCO is used to insert a new multi-configuration operand in the multi-configuration editor. The syntax is:

INSERTMCO row

where row is an integer greater than 0 and smaller than or equal to the current number of operands plus 1.

DELETECONFIG is used to delete a configuration from current multi-configuration editor. The syntax is:

DELETECONFIG config

where config is an integer greater than 0 and smaller than or equal to the number of current configurations.

DELETEMCO is used to delete an existing operand in the multi-configuration editor. The syntax is:

DELETEMCO row

where row is an integer greater than 0 and smaller than or equal to current number of operands.

If we want to set or modify the parameters in the multi-configuration editor, we can use keyword SETCONFIG and SETMCOPERAND.

SETCONFIG is used to set the current configuration for multi-configuration (zoom) systems. The syntax is:

SETCONFIG config

where config is an integer greater than 0 and smaller than or equal to current number of configurations.

SETMCOPERAND is used to set any row or configuration of the Multi-Configuration Editor to any numeric value. The syntax is:

SETMCOPERAND row, config, value, datatype

where row and config are used to specify the row and configuration of the Multi-Configuration Editor.

If the config number is 0, then the value is interpreted as follows:

datatype = 0, value is a string literal or variable that specifies the name of the operand.

datatype = 1, 2, or 3, value is the number 1, 2, or 3 value used as part of the multi-configuration operand

definition.

If the config number corresponds to a defined configuration then the value is interpreted as follows:

datatype = 0, value is the value of the operand.

datatype = 1, value is the pickup offset of the operand.

datatype = 2, value is the pickup scale of the operand.

datatype = 3, value is the status of the operand, 0 for fixed, 1 for variable, 2 for pickup, 3 for thermal pickup.

datatype = 4, value is the pickup configuration number.

datatype = 5, value is the pickup row number.

The table below listed codes and arguments for various operands:

Table 3.11-1 SUMMARY OF MULTI-CONFIGURATION OPERANDS

Type	Numbers 1,2,3	Description
AFOC	Ignored	Afocal Image Space mode.
AICN	Surface, Object	iPartFactory Number for the Autodesk Inventor part.
APDF	Ignored	System apodization factor. See also APDT.
APDT	Ignored	System apodization type. Use 0 for none, 1 for Gaussian, 2 for cosine cubed. See also APDF.
APDX	Surface #	Surface aperture X- decenter. The surface must have a defined aperture (NOT semi-diameter).
APDY	Surface #	Surface aperture Y- decenter. The surface must have a defined aperture (NOT semi-diameter).
APER	Ignored	System aperture value. If the system aperture type is float by stop size, this is the semi-diameter of the stop surface. See also SATP.
APMN	Surface #	Surface aperture minimum value. The surface must have a defined aperture (NOT semi-diameter). This same operand also works to control the first parameter of all surface aperture types, such as the X-Half Width on rectangular and elliptical apertures.
APMX	Surface #	Surface aperture maximum value. The surface must have a defined aperture (NOT semi-diameter). This same operand also works to control the second parameter of all surface aperture types, such as the Y-Half Width on rectangular and elliptical apertures.
APTP	Surface #	Surface aperture type. The integer values indicating the aperture type are 0-10 for none, circular aperture, circular obscuration, spider, rectangular aperture, rectangular obscuration, elliptical aperture, elliptical obscuration, user aperture, user obscuration, and floating aperture; respectively.
CADX	Surface #	Surface Tilt/Decenter after surface decenter x.
CADY	Surface #	Surface Tilt/Decenter after surface decenter y.
CATX	Surface #	Surface Tilt/Decenter after surface tilt x.
CATY	Surface #	Surface Tilt/Decenter after surface tilt y.
CATZ	Surface #	Surface Tilt/Decenter after surface tilt z.
CAOR	Surface #	Surface Tilt/Decenter after surface order. Use 0 for Decenter then Tilt, or 1 for Tilt then Decenter.
CBDX	Surface #	Surface Tilt/Decenter before surface decenter x.
CBDY	Surface #	Surface Tilt/Decenter before surface decenter y.
CBTX	Surface #	Surface Tilt/Decenter before surface tilt x.
CBTY	Surface #	Surface Tilt/Decenter before surface tilt y.

CBTZ	Surface #	Surface Tilt/Decenter before surface tilt z.
CBOR	Surface #	Surface Tilt/Decenter before surface order. Use 0 for Decenter then Tilt, or 1 for Tilt then Decenter.
CONN	Surface #	Conic constant.
COTN	Surface #	The name of the coating, if any, to be applied to the surface.
CPCN	Surface, Object	Family Table Instance Number for the Creo Parametric part.
CROR	Surface #	Coordinate Return Orientation. Use 0 for none, 1 for Orientation only, 2 for Orientation XY, and 3 for Orientation XYZ.
CRSR	Surface #	Coordinate Return Surface.
CRVT	Surface #	Curvature of surface.
CSP1	Surface #	Curvature solve parameter 1.
CSP2	Surface #	Curvature solve parameter 2.
CWGT	Ignored	The overall weight for the configuration. This number only has meaning relative to the weights in other configurations.
EDVA	Surface, Extra Data Number	The EDVA operand is used to assign multiple values to the extra data values. This operand requires 2 numerical arguments: the surface number and the extra data value number.
FLTP	Ignored	Field type. Use 0 for angle in degrees, 1 for object height, 2 for paraxial image height, 3 for real image height.
FLWT	Field #	Field weight.
FVAN	Field #	Vignetting factor VAN.
FVCX	Field #	Vignetting factor VCX.
FVCY	Field #	Vignetting factor VCY.
FVDX	Field #	Vignetting factor VDX.
FVDY	Field #	Vignetting factor VDY.
GCRS	Ignored	The global coordinate reference surface.
GLSS	Surface #	Glass.
GPEX, GPEY	obsolete	obsolete
GPJX	Ignored	Global Jones polarization vector component Jx.
GPJY	Ignored	Global Jones polarization vector component Jy.
GPIU	Ignored	Global polarization state "is unpolarized", 1 if polarization state is unpolarized, otherwise state is polarized.
GPPX	Ignored	Global polarization state phase x.
GPPY	Ignored	Global polarization state phase y.
G QPO	Ignored	Obscuration value used for Gaussian Quadrature pupil sampling in the default merit function.
HOLD	Ignored	Holds data in the multi-configuration buffer, but has no other effect. Useful for temporarily turning off one operand without losing the associated data.
IGNR	Surface #	Ignore This Surface status. Use 0 to consider the surface, and 1 to ignore the surface. If IGNR

		and IGMN operands are defined for the same surface, the one listed second will take precedence.
IGNM	First Surface, Last Surface	Sets Ignore This Surface status on a range of surfaces. Use 0 to consider the surfaces, and 1 to ignore the surfaces. If IGNR and IGMN operands are defined for the same surface, the one listed second will take precedence.
LTTL	Ignored	Lens title. The string length is limited to 32 characters.
MABB	Surface #	Model glass Abbe.
MCOM	Surface #	Surface comment.
MDPG	Surface #	Model glass dPgF.
MIND	Surface #	Model glass index.
MOFF	Ignored	An unused operand, may be used for entering comments.
MTFU	Ignored	MTF units. Use 0 for cycles/millimeter or 1 for cycles/milliradian.
NCOM	Surface, Object	Modifies the comment for non-sequential objects in the NSC Editor. The string value is limited to 32 characters.
NCOT	Surface, Object, Face #	Modifies the coating on each face for non-sequential objects in the NSC Editor.
NGLS	Surface, Object	The material type for non-sequential objects in the NSC Editor.
NPAR	Surface, Object, Parameter	Modifies the parameter columns for non-sequential objects in the NSC Editor.
NPOS	Surface, Object, Position	Modifies the x, y, z, tilt x, tilt y, and tilt z position values for nonsequential objects in the NSC Editor. The position flag is an integer between 1 and 6 for x, y, z, tilt x, tilt y, and tilt z, respectively.
NPRO	Surface, Object, Property	Modifies various properties of NSC objects. Property is an integer value indicating what data is controlled: 1 - Inside of object number 2 - Reference object number 3 - Do Not Draw Object (0 = no, 1 = yes) 4 - Rays Ignore Object (0 = never, 1 = always, 2 = on launch) 5 - Use Pixel Interpolation (0 = no, 1 = yes) 201-212 - User defined gradient index parameters 301-312 - User defined diffraction parameters for reflection 351-362 - User defined diffraction parameters for transmission 401-416 - User defined bulk scatter parameters 481, 482 - Bulk scatter mean free path and angle arguments. 500 - Media is birefringent. Use 0 for false and 1 for true, 501 - Birefringent mode. use 0-3 for ordinary and extraordinary rays, ordinary rays only, extraordinary rays only, and waveplate mode, respectively. 502 - Birefringent Reflections. Use 0 for refracted and reflected

		rays, 1 for refracted rays only, and 2 for reflected rays only. 503-505 - Birefringent crystal axis orientation x, y, and z.
PAR1	Surface #	Parameter 1. Obsolete, use PRAM instead.
PAR2	Surface #	Parameter 2. Obsolete, use PRAM instead.
PAR3	Surface #	Parameter 3. Obsolete, use PRAM instead.
PAR4	Surface #	Parameter 4. Obsolete, use PRAM instead.
PAR5	Surface #	Parameter 5. Obsolete, use PRAM instead.
PAR6	Surface #	Parameter 6. Obsolete, use PRAM instead.
PAR7	Surface #	Parameter 7. Obsolete, use PRAM instead.
PAR8	Surface #	Parameter 8. Obsolete, use PRAM instead.
PRAM	Surface, Parameter	Parameter value. This operand controls any of the parameters.
PRES	Ignored	Air pressure in atmospheres. Zero means vacuum, 1 means normal air pressure.
PRWV	Ignored	Primary wavelength number.
PSCX	Ignored	X Pupil Compress. Used for ray aiming.
PSCY	Ignored	Y Pupil Compress. Used for ray aiming.
PSHX	Ignored	X Pupil Shift. Used for ray aiming.
PSHY	Ignored	Y Pupil Shift. Used for ray aiming.
PSHZ	Ignored	Z Pupil Shift. Used for ray aiming.
PSP1	Surface #	Parameter solve parameter 1 (the pickup surface). This operand requires 2 numerical arguments: the surface number and the parameter number.
PSP2	Surface #	Parameter solve parameter 2 (the scale factor). This operand requires 2 numerical arguments: the surface number and the parameter number.
PSP3	Surface #	Parameter solve parameter 3 (the offset). This operand requires 2 numerical arguments: the surface number and the parameter number.
PUCN	Ignored	Used for picking up a range of values from a previous configuration. If a positive integer configuration number is provided, then all values below the PUCN operand will be picked up from the configuration number specified. If the configuration value is negative, then a negative pickup will be used. If the configuration number is zero, then the values below the PUCN operand will not have pickup solves applied. Note two PUCN operands can be used to define the beginning and end of a range of values to be picked up. All specified configuration numbers must be less than the configuration the PUCN data is provided for.
PXAR	Surface #	Physical optics setting "Use X-axis Reference". Use 0 for no, 1 for yes.

RAAM	Ignored	Ray aiming. Use 0 for off, 1 for paraxial, and 2 for real.
SATP	Ignored	System aperture type. Use 0 for Entrance Pupil Diameter, 1 for Image Space F/#, 2 for Object Space NA, 3 for Float By Stop Size, 4 for Paraxial Working F/#, 5 Object Cone Angle. See also APER.
SDIA	Surface #	Semi-diameter.
SDRW	Surface #	Modifies the do not draw this surface flag. Use 0 to draw and 1 to not draw.
STPS	Ignored	Stop surface number. The stop can be moved to any valid surface number (excluding the object and image surfaces) by specifying an integer argument for each configuration.
SWCN	Surface, Object	Configuration number for the SolidWorks part.
TCEX	Surface #	Thermal coefficient of expansion.
TELE	Ignored	Telecentric in object space, 0 for no, 1 for yes.
TEMP	Ignored	Temperature in degrees Celsius.
THIC	Surface #	Thickness of surface.
TSP1	Surface #	Thickness solve parameter 1.
TSP2	Surface #	Thickness solve parameter 2.
TSP3	Surface #	Thickness solve parameter 3.
UDAF	Surface #	User defined aperture file. Surface must use either a user defined aperture or user defined obscuration aperture type.
WAVE	Wave #	Wavelength.
WLWT	Wave #	Wavelength weight.
XFIE	Field #	X-field value.
YFIE	Field #	Y-field value.

Now we will give some examples to show how to define and modify multi-configurations in ZPL program.

Example 3.11-1: Define and modify multi-configuration system.

```
1 ! ex31101
2 ! This program shows how to create a multiconfiguration system from scratch
3 ! Before start this program, make sure a new lens file is created
4
5 ! add 5 more surfaces
6 FOR i, 1, 5, 1
7     INSERT 1
8 NEXT
9
10 ! set system parameters
11 SYSP 30, 0 # set lens unit as mm
12 SYSP 10, 0 # set system aperture as Entrance Pupil Diameter
13 SYSP 11, 15 # set system aperture value as 15mm
14
15 SYSP 201, 1 # set total wavelength number as 1
16 SYSP 202, 1, 0.55 # set the wavelength as 0.55 micron
17
18 SYSP 100, 0 # set the field type as Angle
19 SYSP 101, 3 # set the total field number as 3
20 SYSP 102, 1, 0 # set field 1 as x = 0 degree
21 SYSP 103, 1, 0 # set field 1 as y = 0 degree
22 SYSP 104, 1, 1 # set field 1 as weight = 1
23 SYSP 102, 2, 0 # set field 2 as x = 0 degree
24 SYSP 103, 2, 0.5 # set field 2 as y = 0.5 degree
25 SYSP 104, 2, 1 # set field 2 as weight = 1
26 SYSP 102, 3, 0 # set field 3 as x = 0 degree
27 SYSP 103, 3, 1 # set field 3 as y = 1 degree
28 SYSP 104, 3, 1 # set field 3 as weight = 1
29
```

```

29
30 ! set surface parameters
31 SURF 1, THIC, -37 # set surface 1 thickness as -37
32 SURF 2, THIC, 50 # set surface 2 thickness as 50
33
34 SURF 3, CURV, -1/65.8 # set surface 3 curvature as -1/65.8
35 SURF 3, THIC, -12.34 # set surface 3 thickness as -12.34
36 SURF 3, GLAS, "MIRROR" # set surface 3 glass type as MIRROR
37
38 SURF 4, CURV, -1/124.6 # set surface 4 curvature as -1/124.6
39 SURF 4, THIC, 60.62 # set surface 4 thickness as 60.62
40 SURF 4, GLAS, "MIRROR" # set surface 4 glass type as MIRROR
41
42 SURF 5, CURV, -1/21.8 # set surface 5 curvature as -1/21.8
43 SURF 5, THIC, -7.01 # set surface 5 thickness as -7.01
44 SURF 5, GLAS, "MIRROR" # set surface 5 glass type as MIRROR
45
46 SURF 6, CURV, -1/25.7 # set surface 6 curvature as -1/25.7
47 SURF 6, THIC, 48.02 # set surface 6 thickness as 48.02
48 SURF 6, GLAS, "MIRROR" # set surface 6 glass type as MIRROR
49
50 ! set surface 3 as stop
51 STOPSURF 3
52
53 ! insert 4 more configurations
54 FOR i, 1, 4, 1
55     INSERTCONFIG 1
56 NEXT
57
58 ! insert 4 more operand rows in the multi-configuration editor
59 FOR i, 1, 4, 1
60     INSERTMCO 1
61 NEXT
62
63 ! set the operand types in different rows
64 row = 1
65 config = 0
66 datatype = 0
67 value$ = "THIC"
68 value1 = 1
69 datatype1 = 1
70 SETMCOOPERAND row, config, value$, datatype # set operand type
71 SETMCOOPERAND row, config, value1, datatype1 # set operand surface #
72 SETMCOOPERAND 2, 0, "APMN", 0 # set operand type
73 SETMCOOPERAND 2, 0, 1, 1 # set operand surface #
74 SETMCOOPERAND 3, 0, "THIC", 0 # set operand type
75 SETMCOOPERAND 3, 0, 3, 1 # set operand surface #
76 SETMCOOPERAND 4, 0, "THIC", 0 # set operand type
77 SETMCOOPERAND 4, 0, 4, 1 # set operand surface #
78 SETMCOOPERAND 5, 0, "THIC", 0 # set operand type
79 SETMCOOPERAND 5, 0, 5, 1 # set operand surface #
80

```

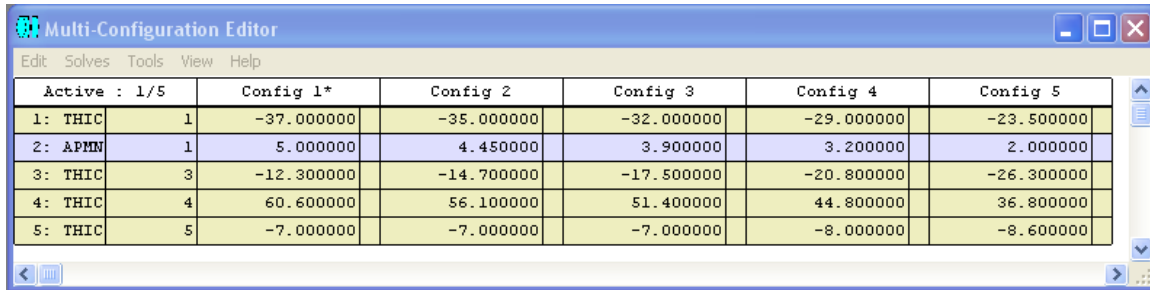
```

80
81 ! set the operand values in different configurations
82 SETHCOPERAND 1, 1, -37, 0 # set the first operand in each configuration
83 SETHCOPERAND 1, 2, -35, 0
84 SETHCOPERAND 1, 3, -32, 0
85 SETHCOPERAND 1, 4, -29, 0
86 SETHCOPERAND 1, 5, -23.5, 0
87 SETHCOPERAND 2, 1, 5, 0 # set the second operand in each configuration
88 SETHCOPERAND 2, 2, 4.45, 0
89 SETHCOPERAND 2, 3, 3.9, 0
90 SETHCOPERAND 2, 4, 3.2, 0
91 SETHCOPERAND 2, 5, 2, 0
92 SETHCOPERAND 3, 1, -12.3, 0 # set the third operand in each configuration
93 SETHCOPERAND 3, 2, -14.7, 0
94 SETHCOPERAND 3, 3, -17.5, 0
95 SETHCOPERAND 3, 4, -20.8, 0
96 SETHCOPERAND 3, 5, -26.3, 0
97 SETHCOPERAND 4, 1, 60.6, 0 # set the fourth operand in each configuration
98 SETHCOPERAND 4, 2, 56.1, 0
99 SETHCOPERAND 4, 3, 51.4, 0
100 SETHCOPERAND 4, 4, 44.8, 0
101 SETHCOPERAND 4, 5, 36.8, 0
102 SETHCOPERAND 5, 1, -7, 0 # set the fifth operand in each configuration
103 SETHCOPERAND 5, 2, -7, 0
104 SETHCOPERAND 5, 3, -7, 0
105 SETHCOPERAND 5, 4, -8, 0
106 SETHCOPERAND 5, 5, -8.6, 0
107
108 UPDATE

```

In this example, we want to build a multi-configuration system from scratch, so we need first create a new lens system from Zemax main window file menu. In the new system, we can see three surfaces in the lens editor. 5 more surfaces are inserted in lines 6 ~ 8 of the program, so there are total 8 surfaces, including object and image surfaces. Lines 11 ~ 28 set some basic system parameters, including lens unit (line 11), entrance pupil type and size (lines 12 and 13), wavelength number and value (lines 15 and 16), and object field (lines 18 ~ 28). Lines 31 ~ 48 define parameters of each surface, line 51 defines the stop surface. After basic optical system is set up, lines 54 ~ 56 insert 4 configurations in the multi-configuration editor, so there are total 5 configurations. Lines 59 ~ 61 insert 4 operand rows in the multi-configuration editor, so there are total 5 rows. Lines 64 ~ 79 define the type of each operand in the multi-configuration editor, and lines 82 ~ 106 set the value of each operand in different configurations. The last line in the program updates the optical system using keyword UPDATE to assure each parameter of the system is the newest value.

After running program ex31101.ZPL, if we open multi-configuration editor, the content can be seen as shown in figure 3.11-1:



Active : 1/5	Config 1*	Config 2	Config 3	Config 4	Config 5	
1: THIC	1	-37.000000	-35.000000	-32.000000	-29.000000	-23.500000
2: APMN	1	5.000000	4.450000	3.900000	3.200000	2.000000
3: THIC	3	-12.300000	-14.700000	-17.500000	-20.800000	-26.300000
4: THIC	4	60.600000	56.100000	51.400000	44.800000	36.800000
5: THIC	5	-7.000000	-7.000000	-7.000000	-8.000000	-8.600000

Fig. 3.11-1: Content in the multi-configuration editor after running program ex31101.ZPL

If we open 3D Layout window, we can see each configuration as shown in figure 3.11-2:

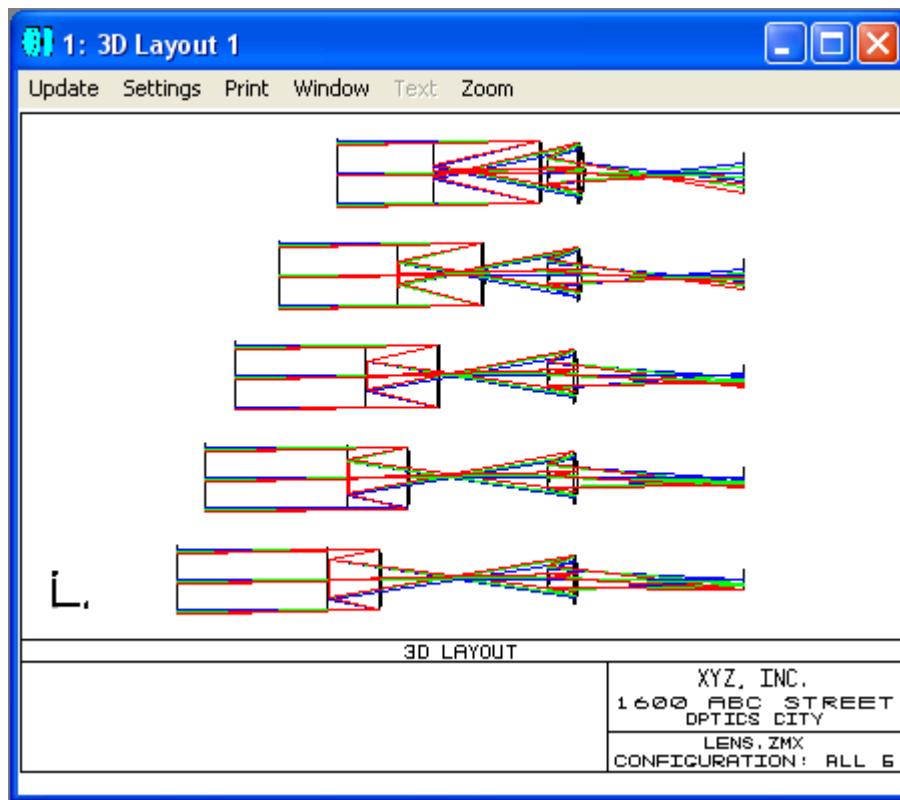


Fig. 3.11-2: Content of the 3D Layout window after running program ex31101.ZPL

Of course, this multi-configuration system is not optimized yet. If needed, we can choose proper variables and merit function to further optimize this system, depending on the design target.

Besides setting parameters of multi-configuration system, we can also read various parameters through ZPL functions such as `NCON()`, `CONF()`, `MCOP()`, `MCON()`, etc.

Function `NCON()` is used to read the number of configurations. The syntax is:

```
returnValue = NCON( )
```

Function `CONF()` is used to read the current configuration number. The syntax is:

```
returnValue = CONF ( )
```

Function `MCOP()` is used to read the data of given row (operand) in the given configuration. The syntax is:

```
returnValue = MCOP(row, config)
```

where row is the row number of the operand, config is the configuration number. If config is 0, then the current configuration is chosen.

Function `MCON()` is used to extract data from any row and configuration of the Multi-Configuration Editor. This function is similar to `MCOP` with extended capabilities for extracting data. The syntax is:

```
MCON(row, config, data)
```

where row is the row number (operand number), config is the configuration number, and data is the data value to extract from the Multi-Configuration Editor. If the row and config number are both zero, `MCON` returns either the number of operands, the number of configurations, or the active configuration number for data = 0, 1, and 2, respectively. If the row number is between 1 and the number of multi-config operands, and the config number is zero, `MCON` returns the operand type, integer 1, integer 2, integer 3, and string flag for that specified row, for data = 0 through 4, respectively. The 3 integer values are used for various purposes for different operands, such as surface and wavelength numbers. The string flag is 1 if the operand data is a string value, such as a glass name, or 0 for numerical data. If the row number is between 1 and the number of multi-config operands, and the config number is valid, `MCON` returns either the numerical value or the string data for that operand.

Note that all string data returned by `MCON` must be extracted with the `$buffer` command after the call to `MCON`. For example, the following code will place the name of the operand on row 1 in a\$:

```
osphotonics.wordpress.com
```

```
dummy = MCON(1, 0, 0)
```

```
a$ = $buffer()
```

Example 3.11-2: Extract data from multi-configuration system.

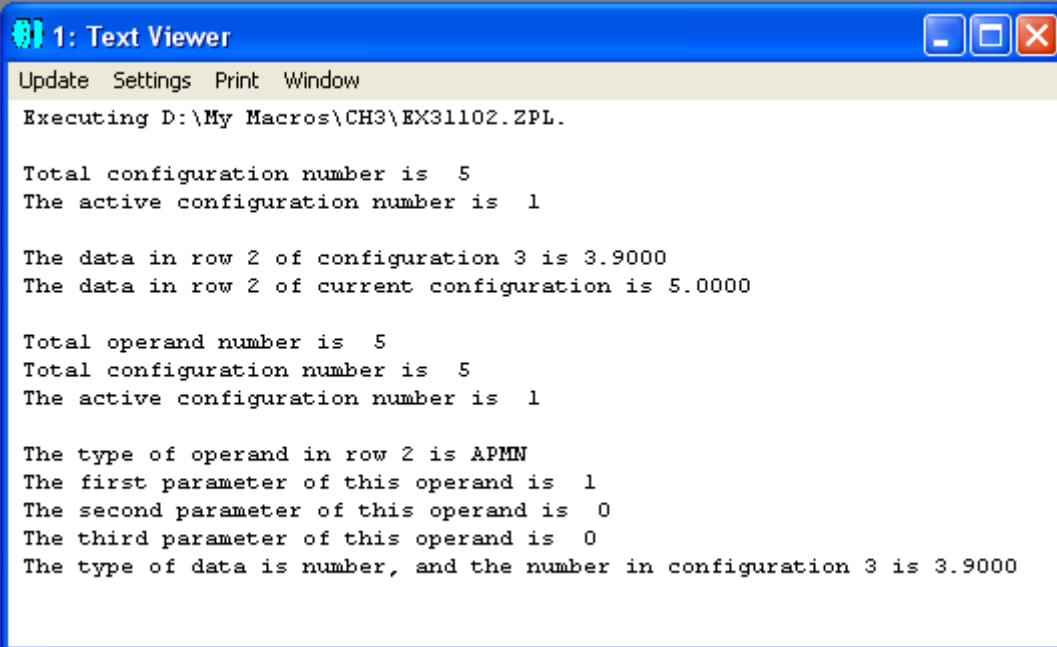
```

1 ! ex31102
2 ! This program shows how to read data from multiconfiguration editor
3 ! Assume the multi-configuration system is defined in ex31101
4
5 PRINT
6 FORMAT 2.0
7 PRINT "Total configuration number is ", NCON()
8 PRINT "The active configuration number is ", CONF()
9
10 FORMAT 6.4
11 PRINT
12 PRINT "The data in row 2 of configuration 3 is ", MCOP(2,3)
13 PRINT "The data in row 2 of current configuration is ", MCOP(2,0)
14
15 FORMAT 2.0
16 PRINT
17 PRINT "Total operand number is ", MCON(0,0,0)
18 PRINT "Total configuration number is ", MCON(0,0,1)
19 PRINT "The active configuration number is ", MCON(0,0,2)
20
21 dummy = MCON(2,0,0)
22 a$ = $buffer()
23 PRINT
24 PRINT "The type of operand in row 2 is ", a$
25 PRINT "The first parameter of this operand is ", MCON(2,0,1)
26 PRINT "The second parameter of this operand is ", MCON(2,0,2)
27 PRINT "The third parameter of this operand is ", MCON(2,0,3)
28 IF MCON(2,0,4) == 1 # if data in this operand is string
29   PRINT "The type of data is string, ",
30   dummy = MCON(2,3,0)
31   a$ = $buffer()
32   PRINT "and the string in configuration 3 is ", a$
33 ELSE # if data in this operand is number
34   FORMAT 6.4
35   PRINT "The type of data is number, ",
36   PRINT "and the number in configuration 3 is ", MCON(2,3,0)
37 ENDIF

```

In this example, we assume the multi-configuration system is the one defined in example 3.11-1. Line 7 reads the total number of configurations with function NCON(); line 8 read the current configuration number through function CONF(); lines 12 and 13 read data of row 2 in configuration 3 and row 2 of current configuration, respectively, through function MCOP; lines 17 ~ 37 read various data through function MCON, such as total number of operands (line 17), total number of configurations (line 18), current configuration number (line 19), type of operand in row 2 of multi-configuration editor (lines 21 ~

24) and its corresponding 1st, 2nd and 3rd parameters (lines 25 ~ 27). In line 28, the type of the operand data is first determined to be string or value. If it is string, then the buffer string function is used to extract the data (lines 29 ~ 32), otherwise the value is directly read (line 36). The result of the program is shown in figure 3.11-3.



```
1: Text Viewer
Update Settings Print Window
Executing D:\My Macros\CH3\EX31102.ZPL.

Total configuration number is 5
The active configuration number is 1

The data in row 2 of configuration 3 is 3.9000
The data in row 2 of current configuration is 5.0000

Total operand number is 5
Total configuration number is 5
The active configuration number is 1

The type of operand in row 2 is APMN
The first parameter of this operand is 1
The second parameter of this operand is 0
The third parameter of this operand is 0
The type of data is number, and the number in configuration 3 is 3.9000
```

Fig. 3.11-3: result of program ex31102.ZPL

From the result we can see that both function MCOP() and function MCON() can be used to read the same data, such as total number of configurations (line 7 and line 18), current configuration number (line 8 and line 19), or data of the given row in the given configuration (line 12 and line 36), and the results are the same. However, function MCON() can be used to extract more data, such as operand type (lines 21 ~ 24) and parameters (lines 25 ~ 27), etc. In this example, since the operand only has one parameter, the return values of the second and third parameters are 0. Please also note that when using function MCON() to read the operand value, if the type of the return value is unknown, we can read the string flag to determine the type, and then read the data properly, as shown in lines 28 ~ 37.

3.12 Display

Screen is the most common terminal device in a computer system. ZPL provided a lot of keywords and functions to control and read/write screen information. Actually, we have already introduced some keywords and functions related to screen and other terminals, such as INPUT for allowing user to input numerical or string information through keyboard, OUTPUT for allowing user to output result to windows or files, FORMAT for controlling the format of numerical values, PRINT for allowing user to display result to text windows on the screen or output to a file, etc. In this section, we will continue to introduce some other commands, focusing on graphic display on the screen. Please remember that Zemax continuously adds new commands, so we suggest users to refer to Zemax User's Manual for the update.

In chapter 1 we mentioned that graphic window is a very important tool for Zemax to output data. ZPL provided a keyword GRAPHICS to open a standard graphic window to allow designers to draw their own graphs. The syntax of GRAPHICS is:

GRAPHICS

GRAPHICS NOFRAME

GRAPHICS OFF

If GRAPHICS is specified alone, then a standard Zemax graphics window will be created. If the optional argument NOFRAME is supplied, then the standard frame for the graph title will be suppressed. All subsequent graphics commands will be sent to this newly created window. GRAPHICS OFF will close any existing open graphics windows, and then display the closed window.

If we want to add a title in the graphics window, we can use keyword GTITLE. The syntax is:

GTITLE user_title\$

where user_title\$ is the title string user defined, and the text will appear centered in the title bar on the graphics display.

If we want to display string at given location and orientation in the graphics window, we can use keyword GTEXT. The syntax is:

GTEXT x, y, angle, user_text\$

where x, y are the coordinates refer to the left edge of where the text string user_text will appear. "user_text" may be either a constant string in quotes or a string variable name. Angle specifies how the text is rotated with respect to the graphics frame, and defaults to 0 degrees (horizontal).

If we want to display centered string in the graphics windows, we can use keyword GTEXTCENT. The syntax is:

GTEXTCENT y, user_text\$

where the coordinate y refers to the vertical position of the text string user_text.

When displaying string, we can also use keyword SETTEXTSIZE to set its size. The syntax is:

SETTEXTSIZE xsize, ysize

where the arguments refer to the fraction of the graphic screen width that each character represents. For example, the default text size is 70 40. This means each character is 1/70 of the graphic screen width, and 1/40 of the screen height. An argument of zero restores the text size to the default.

Also current date can be displayed in the graphics windows with keyword GDATE, and the format is determined by the Zemax main menu File → Preference option.

Some versions of Zemax can display the title of the lens file on the graphics window using keyword GLENSNAME so the graphics window user created looks like other Zemax windows. But in general the standard graphics windows created with GRAPHICS already include this content. Please refer to Zemax User's Manual for details.

In the graphics window user created, user can use keywords LINE and PIXEL to draw line segments and pixels.

The syntax of keyword LINE is:

LINE oldx, oldy, newx, newy

where oldx and oldy are the coordinates of the starting point of the line segment, newx and newy are those of the end point of the line segment. They should be contained within the current graphics frame defined by XMIN, YMIN, XMAX, and YMAX. The coordinates can be real values, but Zemax will round them to the nearest integers.

The syntax of keyword PIXEL is:

PIXEL xcoord, ycoord

where xcoord and ycoord are the coordinates of the pixel in the current graphics window.

Keyword COLOR can be used to control the color of the pen when drawing text, pixels or line segments. The syntax is:

COLOR n

The value n is an integer between 0 ~ 24 for different colors. 0 is for black, and the other colors are as defined in Zemax main menu File → Preferences option.

In ZPL display, functions XMIN(), XMAX(), YMIN() and YMAX() are often used to read the minimum and maximum coordinates of the current graphics window. Please note that the origin of the coordinates is located at the upper left corner of the graphics window, with x increases from left to right, and y increases from up to down. Also, the height-width ratio of current graphics device can be read through function ASPR(). The functions mentioned here don't need any arguments.

We can lock an opened window with keyword LOCKWINDOW. The syntax is:

LOCKWINDOW winnum

where winnum is the window number. If winnum is 0, then all the windows will be locked, and if it is -1, then the current window will be locked after the program is executed.

If we want to unlock a window, we can use keyword UNLOCKWINDOW. The syntax is:

UNLOCKWINDOW winnum

where winnum is the window number. If winnum is 0, then all the windows will be unlocked, and if it is -1, then the current window will be unlocked after the program is executed.

If we want to close a window, we can use keyword CLOSEWINDOW. The syntax is:

CLOSEWINDOW

or

CLOSEWINDOW winnum

If CLOSEWINDOW is used alone with no argument "n" provided, it will run the ZPL macro in "quiet" mode. The text window normally displayed at the end of the macro execution will not be displayed if the CLOSEWINDOW keyword is included at any line in the macro. CLOSEWINDOW has no other effect on macro execution.

If CLOSEWINDOW is used with an integer argument "n" provided, it will close analysis window number n.

When we discuss keywords PRINT and OUTPUT in section 2.7, we mentioned that we can output the result to a file using keyword OUTPUT. But how do we display the result saved to the file? We can do it with keyword SHOWFILE. The syntax is:

SHOWFILE filename\$, saveflag

It displays a text file to the screen using the Zemax file viewer. The filename must be a valid file name. The file must be a text file (as would be created by OUTPUT and PRINT commands in ZPL) and must be in the current folder (determined by Zemax main menu File → Preferences → Directories option). Once the file is displayed, it may be scrolled up and down and printed like any other text file. The ability to scroll and print the data is the primary advantage of using OUTPUT and SHOWFILE instead of PRINT commands. SHOWFILE also closes the file if no CLOSE command has been executed. If the saveflag is zero or omitted, then the file is erased when the window is closed. If saveflag is any value other than zero, then the file remains even after the window is closed.

Besides displaying text files, ZPL can also display image files. For example, with keyword IMASHOW, image files with IMA or BIM format can be displayed in a graphics window. The syntax is:

IMASHOW filename\$

This keyword requires the name of the IMA or BIM file. The extension must be included. The filename may be enclosed in quotes if any blank or other special characters are used. The file must be located in the <data>\<images> folder. This command will open a new window to display the file.

Now let's give an example to show how to use the functions and keywords related to screen display.

Example 3.12-1: Screen display

```

1 ! ex31201
2 ! This program shows how to use display-related keywords and functions
3
4 ! open a graphic window
5 GRAPHICS
6
7 ! get the coordinates
8 xmx = XMAX()
9 xmn = XMIN()
10 ymx = YMAX()
11 ymn = YMIN()
12 xWidth = xmx-xmn
13 yWidth = ymx-ymn
14 xLeft = xmn + ( 0.1 * xWidth )
15 xRight = xmn + ( 0.9 * xWidth )
16 yTopp = ymn + ( 0.1 * yWidth )
17 yBott = ymn + ( 0.7 * yWidth )
18
19 ! draw a frame
20 LINE xLeft,yTopp,xRight,yTopp
21 LINE xRight,yTopp,xRight,yBott
22 LINE xRight,yBott,xLeft,yBott
23 LINE xLeft,yBott,xLeft,yTopp
24
25 ! add some text
26 SETTEXTSIZE 80, 80
27 GTEXT xmx*0.2,ymx*0.75,0, " X axis"
28 SETTEXTSIZE 50, 50
29 GTEXTCENT ymx*0.75, " X axis"
30 SETTEXTSIZE 30, 30
31 GTEXT xmx*0.6,ymx*0.75,0, " X axis"
32 SETTEXTSIZE 0, 0
33 GTEXT xmx*0.05,ymx*0.45,90, " Y axis"
34 GDATE # add data/time
35

```

```
35
36 ! draw a curve
37 pi = 3.1416
38 FOR theta, 1, 3600, 1
39   r = yWidth/4/3600*theta
40   x = r*COSI(theta*pi/180)+xmn+0.5*xWidth
41   y = r*SINE(theta*pi/180)+ymn+0.4*yWidth
42   COLOR theta - INTE(theta/25)*25
43   PIXEL x,y
44 NEXT
45
46 ! draw title
47 COLOR 0
48 myTitle$ = "This is a user created graphic window"
49 GTITLE myTitle$
50
51 PRINT "This line will not be displayed!"
52 CLOSEWINDOW
53 GRAPHICS OFF
```

In this program, we first open a graphics window using keyword GRAPHICS, and then read the basic coordinates of the window (lines 8 ~ 11). After that, we defined the coordinates of 4 corners and draw a rectangle (lines 12 ~ 23), and then output some text information (lines 26 ~ 34). In the text output, we set the size (lines 26, 28, 30, 32), center (line 29), rotation (line 33), and output the date (line 34). After that, we draw a spiral curve in the graphics window (lines 37 ~ 44) with color control (line 42). At the end of the program, we add a title to the graphics window (lines 47 ~ 49). Although we add a print command on line 51, the result will not be seen because we choose to run the program in “quiet” mode, and the window used to display the text message is closed.

After program is executed, the graphical result is seen as in figure 3.12-1:

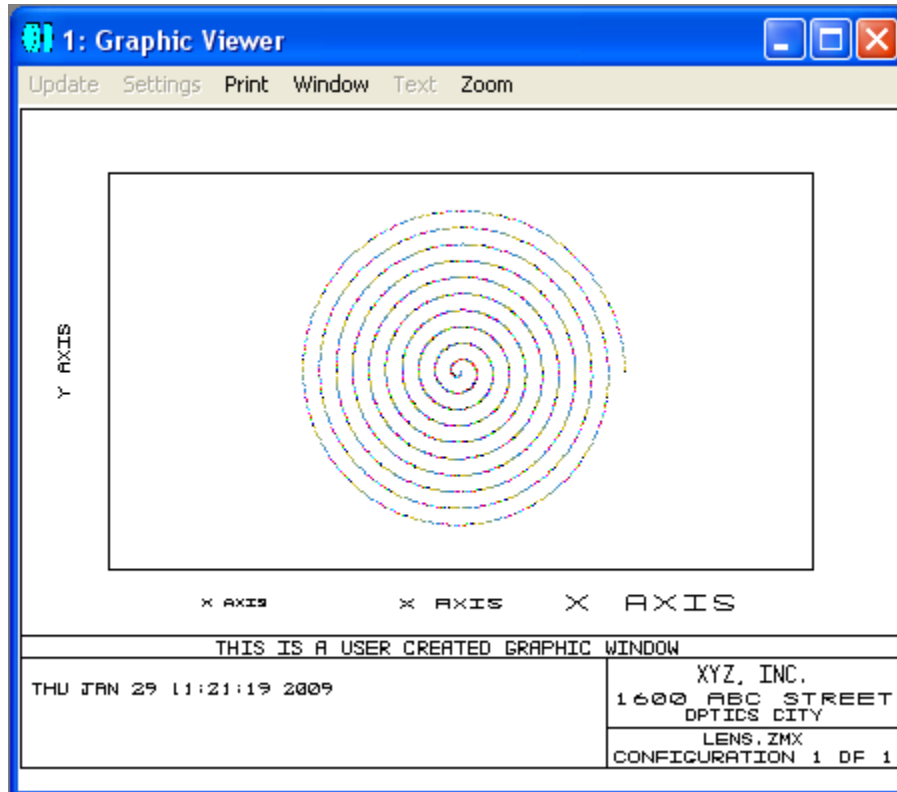


Fig. 3.12-1: result of program ex31201.ZPL

3.13 File Operation

During optical design with Zemax, file operations are often needed. We discussed some basic file operation commands in section 7 of chapter 2. In this section, we will continue to discuss more file operation commands.

When we do optical design, often times we don't need to start from scratch. Instead, we can load existing lens files, or re-load lens files when running ZPL programs. ZPL provided a keyword `LOADLENS` to do this. The syntax is:

LOADLENS filename\$, appendflag, session

where `filename$` is the lens file. If the filename contains the complete path, then the specified file will be loaded. If the path is left off, then the default folder for lenses defined by Zemax main menu File → Preferences → Directories will be used. If the `appendflag` is zero or absent, then `LOADLENS` loads the file. If the `appendflag` is greater than zero, then the file is appended to the current lens starting at the surface specified by the value of the `appendflag`. The `appendflag` should only be used when appending one sequential system to another. Appending non-sequential systems isn't currently supported. If the `session` flag is non-zero, any associated session file will be loaded with the lens and all windows will be updated, otherwise, the lens session file is ignored.

When lenses are loaded, any associated glass catalogs and data files, including the `COATING.DAT` file, are automatically loaded if they are not already loaded. However, if these catalogs have been modified, then the `LOADCATALOG` keyword may be used to force a reload of the catalogs. Use of this keyword is not required unless the `COATING.DAT` or glass AGF catalog files have been modified since the start of the current Zemax session. When using this keyword, no arguments are needed.

If we want to re-load merit function file, we can use keyword `LOADMERIT`. The syntax is:

LOADMERIT filename\$

where `filename$` is the merit function file. If the filename contains the complete path, then the specified file will be loaded. If the path is left off, then the default folder defined by Zemax main menu File → Preferences → Directories will be used.

A similar keyword `IMPORTEXTRADATA` can be used to import data into the extra data editor from a file. The syntax is:

IMPORTEXTRADATA surface, filename\$

where `surface` is the surface number, `filename$` is the extra data file to be loaded. Filename should include full path.

Opposite to loading lens files, if we want to save a lens file currently in memory, we can use keyword `SAVELENS`. The syntax is:

SAVELENS filename\$, session

This command will save the current lens file to the specified file name. The name of the current lens in memory will also be changed. If the file name is absent, then the lens data is stored in the current file name. If the session argument evaluates to anything other than zero, the session file will also be saved.

If we want to save current merit function, we can use keyword `SAVEMERIT`. The syntax is:

SAVEMERIT filename\$

This command will save the current merit function to a file. If the filename contains the complete path, then the specified path will be used. If the path is left off, then the default folder defined by Zemax main menu `File → Preferences → Directories` will be used.

Another keyword to save file is `SAVEWINDOW`. It is used to save the text from any text window to a file. The syntax is:

SAVEWINDOW winnum, filename\$

where `winnum` is the text window number that should be saved to a file, and `filename$` is the target file name that may include a full path name or use the default path. Zemax numbers windows sequentially as they are opened, starting with 1. Any closed windows are deleted from the window list, without renumbering the windows which remain. Any windows opened after another window has been closed will use the lowest window number available.

File operation often requires to copy, rename, delete and search a file. ZPL provided keywords `COPYFILE`, `RENAMEFILE`, `DELETEFILE` and `FINDFILE` for this.

Keyword COPYFILE is used to copy a source file to a target file. The syntax is:

COPYFILE sourcefilename\$, newfilename\$

where sourcefilename\$ is the source file name, and newfilename\$ is the target file name. The file names can include path, otherwise the default path will be used. If the target file exists, it will be overwritten.

Keyword RENAMEFILE is used to modify the name of a file. The syntax is:

RENAMEFILE oldfilename\$, newfilename\$

Keyword DELETEFILE is used to delete a file. The syntax is:

DELETEFILE filename\$

Keyword FINDFILE is used to find names of files. The syntax is:

FINDFILE TEMPNAME\$, FILTER\$

This keyword requires two expressions, one to specify the string variable name to store the file name in, and another string variable which contains a "filter" string. The filter string usually specifies a path name and wildcards appropriate to the desired file type.

FINDFILE is useful for listing all files of a certain type in a folder, or for analyzing large numbers of similar lens files. To reset FINDFILE back to the first file of any type, just call FINDFILE with a different filter, then call FINDFILE again with the original filter name. Each time FINDFILE is called with a new filter, it resets back to the first file that meets the filter specifications.

We will now give some examples to show how to use the commands we discussed above. Before doing that, we assume the lens file and the related session file already exist, and they are "ex31301.ZMX" and "ex31301.SES", respectively. If the two files don't exist, we can run program "ex30401.ZPL" in Zemax, then save current optical system to a file, and name it as "ex31301".

Example 3.13-1: File operation

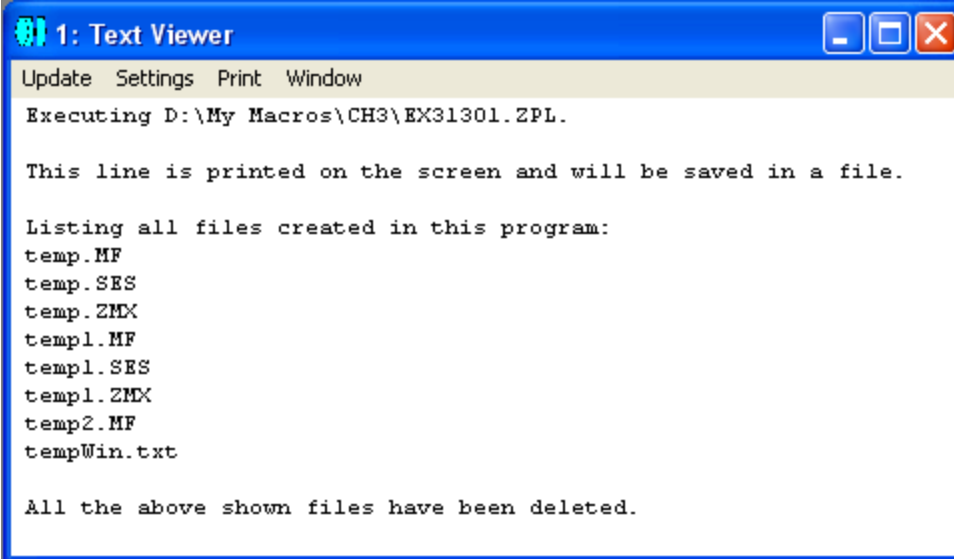
```

1 ! ex31301
2 ! This program shows file operation
3 ! Assume files "ex31301.ZMX" and "ex31301.SES" have already been created,
4 ! otherwise run "ex30401.ZPL" to build the lens system and save files.
5
6 LOADLENS "D:\My Macros\ch3\ex31301.ZMX"
7 LOADLENS "D:\My Macros\ch3\ex31301.ZMX", 6
8
9 SAVELENS "D:\My Macros\ch3\temp.ZMX", 1
10 SAVEMERIT "D:\My Macros\ch3\temp.MF"
11
12 PRINT
13 PRINT "This line is printed on the screen and will be saved in a file."
14 SAVEWINDOW 1, "D:\My Macros\ch3\tempWin.txt"
15
16 COPYFILE "D:\My Macros\ch3\temp.ZMX", "D:\My Macros\ch3\temp1.ZMX"
17 COPYFILE "D:\My Macros\ch3\temp.SES", "D:\My Macros\ch3\temp1.SES"
18 COPYFILE "D:\My Macros\ch3\temp.MF", "D:\My Macros\ch3\temp1.MF"
19
20 RENAMEFILE "D:\My Macros\ch3\temp.MF", "D:\My Macros\ch3\temp2.MF"
21
22 path$ = "D:\My Macros\ch3\"
23 FILTER$ = path$ + "temp*.*"
24 PRINT
25 PRINT "Listing all files created in this program:"
26 FINDFILE TEMPFILE$, FILTER$
27 LABEL 1
28 IF (SLEN(TEMPFILE$))
29 PRINT TEMPFILE$
30 FINDFILE TEMPFILE$, FILTER$
31 GOTO 1
32 ENDIF
33
34 FILTER1$ = path$ + "e*.*"
35 FINDFILE TEMPFILE$, FILTER1$
36 FINDFILE TEMPFILE$, FILTER$
37 LABEL 2
38 IF (SLEN(TEMPFILE$))
39 TEMPFILE$ = path$ + TEMPFILE$
40 DELETEFILE TEMPFILE$
41 FINDFILE TEMPFILE$, FILTER$
42 GOTO 2
43 ENDIF
44 PRINT
45 PRINT "All the above shown files have been deleted."

```

In this program, line 6 loads existing lens file, line 7 re-loads the same file and appends it to the current optical system. Line 9 saves the new current lens system to files "temp.ZMX" and "temp.SES", and line

10 saves current merit function to file "temp.MF". Line 13 outputs a text message to a text window, line 14 saves the content of the text window to file "tempWin.txt" (assume the number of the text window is 1). Lines 16 ~ 18 copy files, line 20 renames a file. Lines 22 ~ 23 set the search condition, line 26 does the first search, line 28 evaluate the search result, if not empty then print file name (line 29), line 30 searches again, and go back to loop for evaluation, until all the files meeting the search condition are printed out. Line 34 sets a new search condition for line 35 to do the search again, in order to re-locate the first search file. Lines 36 ~ 43 actually repeat the same search as in lines 22 ~ 32, and delete the searched files. The result is shown in figure 3.13-1:



```
1: Text Viewer
Update Settings Print Window
Executing D:\My Macros\CH3\EX31301.ZPL.

This line is printed on the screen and will be saved in a file.

Listing all files created in this program:
temp.MF
temp.SES
temp.ZMX
templ.MF
templ.SES
templ.ZMX
temp2.MF
tempWin.txt

All the above shown files have been deleted.
```

Fig. 3.13-1: result of ex31301.ZPL

We can see that a series of files were generated during the execution of the program, however, all those files are deleted at last, so if we try to use windows explorer to view those files after running the program, the files cannot be found.

The function PRINT we discussed before actually outputs the result to either the screen or a file. However, if we want to output the result to a printer, we can use keyword PRINTFILE or PRINTWINDOW.

Keyword PRINTFILE is used to print a text file to the printer. The syntax is:

PRINTFILE filename\$

where filename\$ is the file name that may include path name or use the current path. The file should be a text file. PRINTFILE also closes the file if no CLOSE command has been executed.

Keyword PRINTWINDOW is used to output any open graphic or text window to the printer. The syntax is:

PRINTWINDOW winnum

where winnum is the window number.

If we want to save the content of a graphic window to a file, depending on the file format, we can use keyword EXPORTBMP, EXPORTJPG, or EXPORTWMF.

Keyword EXPORTBMP is used to save a graphic window to a BMP file. The syntax is:

EXPORTBMP winnum, filename\$, delay

In this command, the integer winnum corresponds to the graphic window number that should be saved to a file, the filename is the full file name including the path, but with no extension. Zemax will automatically add the BMP extension. The optional delay parameter specifies a time delay in milliseconds. For some complex graphics, a delay is required to allow the graphic to be completely redrawn and the screen capture to complete. If the BMP files appear incomplete, try a delay value of 500 ~ 2500 milliseconds. It needs to be pointed out that the content of the graphic window is obtained by screen capture. If Zemax is running in the background (e.g. other programs overlap Zemax graphic window), the result of the output file may not be as desired.

Keyword EXPORTJPG is used to save a graphic window to a JPG file. The syntax is:

EXPORTJPG winnum, filename\$, delay

where winnum is the graphic window number, filename\$ is the target file name including the path but no extension, and delay is in the unit of milliseconds.

Keyword EXPORTWMF is used to save a graphic window to a WMF file. The syntax is:

EXPORTWMF winnum, filename\$

where winnum is the graphic window number, filename\$ is the target file name including the path. Different from EXPORTBMP and EXPORTJPG, extension should be included in the file name.

Besides outputting graphic window to a file, ZPL can also export the whole lens system to IGES, STEP, SAT or STL format file for other CAD software to use. Keyword EXPORTCAD is designed to do so. The syntax is:

EXPORTCAD filename\$

where filename\$ is the target file name including the path. The format of the file is determined by a series of parameters that are put in the default vector VEC1. The details of the elements in the vector is listed in table 3.13-1:

Table 3.13-1 Parameters of EXPORTCAD output file

Element	Description
VEC1(1)	The File type. Use 0 for IGES, 1 for STEP, 2 for SAT, 3 for STL.
VEC1(2)	The number of spline points to use (if required on certain entity types). Use 16, 32, 64, 128, 256, or 512.
VEC1(3)	The First surface to export. In NSC Mode, this is the first object to export.
VEC1(4)	The last surface to export. In NSC Mode, this is the last object to export.
VEC1(5)	The layer to place ray data on.
VEC1(6)	The layer to place lens data on.
VEC1(7)	Use 1 to export dummy surfaces, otherwise use 0.
VEC1(8)	Use 1 to export surfaces as solids, otherwise use 0.
VEC1(9)	Ray pattern. Use 0 for XY, 1 for X, 2 for Y, 3 for ring, 4 for list, 5 for none, 6 for grid, and 7 for solid beams.
VEC1(10)	The number of rays.
VEC1(11)	The wave number. Use 0 for all.
VEC1(12)	The field number. Use 0 for all.
VEC1(13)	Use 1 to delete vignetted rays, otherwise use 0.
VEC1(14)	The dummy surface thickness in lens units.
VEC1(15)	Use 1 to split rays from NSC sources, otherwise use 0.
VEC1(16)	Use 1 to scatter rays from NSC sources, otherwise use 0.
VEC1(17)	Use 1 to use polarization when tracing NSC rays, otherwise use 0. Polarization is automatically selected if splitting is specified.
VEC1(18)	Use 0 for the current configuration, 1 to n for a specific configuration where n is the total number of configurations, n+1 to export "All By File", n+2 to export "All By Layer", and n+3 for "All At Once".
VEC1(19)	Tolerance setting. Use 0 for 1.0E-4, 2 for 1.0E-05, 3 for 1.0E-06, and 4 for 1.0E-07.
VEC1(20) ~ VEC1(21)	If the program mode is sequential, and the range of surfaces includes a non-sequential components surface, these values allow a range of objects to be exported. VEC1(20) is the first object to export, and VEC1(21) is the last object to export. If both values are zero or out of range all objects are exported.

Example 3.13-2 shows how to use EXPORTCAD:

```
1 ! ex31302
2 ! This program shows how to use EXPORTCAD key word
3 ! Assume the optical system is defined in "ex31301.zpl"
4
5 VEC1(1) = 1      # define STEP file
6 VEC1(3) = 1      # start surface
7 VEC1(4) = 5      # end surface
8 VEC1(7) = 1      # export dummy surfaces if any
9 VEC1(8) = 1      # export solid model
10 VEC1(18) = 0     # current configuration,
11
12 EXPORTCAD "D:\My Macros\ch3\ex31302.stp"
```

The program assumes the current optical system is defined in example 3.13-1, and the goal is to export the doublet defined between surface 1 ~ surface 5 to a STEP file. After running the program, a file "ex31302.stp" is generated. If we use a CAD program to open this file, we can see the doublet as shown in figure 3.13-2:

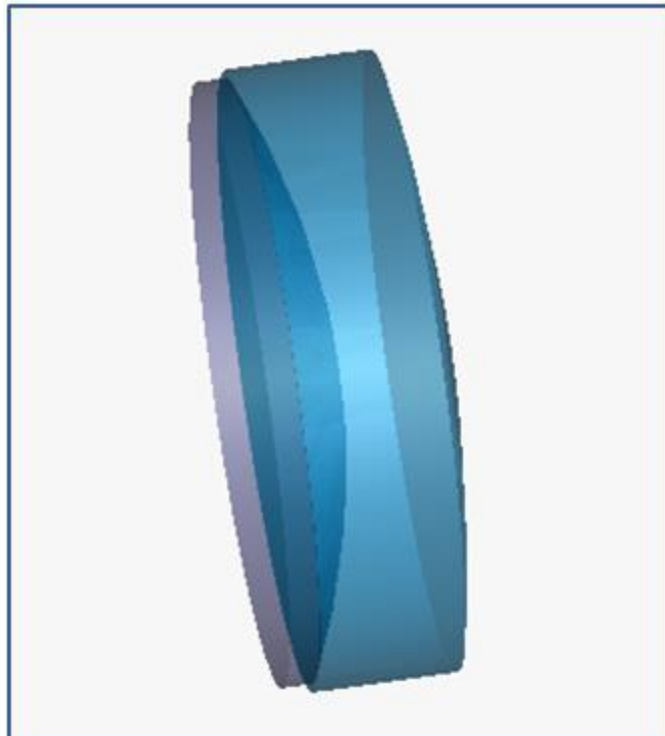


Fig. 3.13-2: the doublet exported by program ex31302.ZPL

3.14 ZBF File

In section 9 of this chapter, we discussed keyword POP for analyzing physical optics propagation. Zemax beam file (ZBF) is used to save the analysis result. Since ZBF is very important to physical optics propagation analysis, ZPL provided a series of related keywords, including ZBFCLR, ZBFMULT, ZBFPROPERTIES, ZBFREAD, ZBFRESAMPLE, ZBFSHOW, ZBFSUM, ZBFTILT, ZBFWRITE, etc. We will discuss them in details in this section. Please note that Zemax saves all the ZBF files into the folder ““...\POP\Beamfiles\”. Although the extension name can be anything, we suggest use “.ZBF” for consistency and clarification.

Keyword ZBFCLR is used to clear the complex amplitude data in a ZBF file. The syntax is:

ZBFCLR filename\$

where filename\$ is the file name.

Keyword ZBFMULT is used to multiply the complex amplitude data in a ZBF file by a complex factor. The syntax is:

ZBFMULT filename\$, Ax, Bx, Ay, By

where filename\$ is the name of the ZBF file, A and B are the real part and imaginary part of the complex number to multiply every point in the ZBF file by, x and y are different polarized light directions. The resulting data is written back to the same file name.

Keyword ZBFPROPERTIES is used to open the specified ZBF file and place various data about the beam in a vector variable. The syntax is:

ZBFPROPERTIES filename\$, vector

where filename\$ is the name of the ZBF file, and vector is 1~4 for the 4 default vectors provided by Zemax. After this command executes, the following beam data will be placed in the specified vector: nx, ny, dx, dy, waist_x, waist_y, position_x, position_y, rayleigh_x, rayleigh_y, wavelength (in lens units), total power, peak irradiance (power per area), the is_polarized flag (0 for no, 1 for yes), and the media index; the values are placed in vector positions 1 through 15.

Keyword ZBFREAD is used to open ZBF file and place the electric field and beam property data in two user-defined array variables. The syntax is:

ZBFREAD filename\$, beamname, propertyname

where filename\$ is the ZBF file name, beamname is a 3 dimensional array of minimum size (nx, ny, 2) for an unpolarized beam and minimum size (nx, ny, 4) for a polarized beam, propertyname is a one dimensional array of minimum size 14. After this command executes, the following beam data will be placed in the specified propertyname array: nx, ny, dx, dy, waist_x, waist_y, position_x, position_y, rayleigh_x, rayleigh_y, wavelength (in lens units), total power, peak irradiance (power per area), the is_polarized flag (0 for no, 1 for yes), and the media index; the values are placed in array positions 1 through 15. The electric field data will be placed in the beamname array. The third dimension of the beamname array is 1 for Ex Real, 2 for Ex Imaginary, and if the beam is polarized, 3 for Ey Real, and 4 for Ey Imaginary.

Keyword ZBFRESAMPLE is used to re-sample a ZBF file to a new width and point spacing. The syntax is:

ZBFRESAMPLE filename\$, nx, ny, wx, wy, decenterx, decentery

where filename\$ is the name of the ZBF file. The beam will be resampled and interpolated as required to create a new beam file with nx and ny points, of total width wx and wy, in the x and y directions, respectively. The nx and ny values must be powers of 2, such as 32, 64, 128, etc. The decenterx and decentery values may be provided to optionally decenter the new beam relative to the old beam. If either nx or ny is zero, no change is made to the existing beam sampling. If either wx or wy is zero, no change is made to the existing beam width. The length units in the ZBF file are converted automatically to the current lens units. The resulting data is written back to the same file name.

Keyword ZBFSHOW is used to display a ZBF file in a viewer window. The syntax is:

ZBFSHOW filename\$

where filename\$ is the ZBF file name. This command will open a new viewer window, and display the ZBF file.

Keyword ZBFSUM is used to sum either coherently or incoherently the data in two ZBF files and places the resulting data in a third ZBF file. The syntax is:

ZBFSUM coherent, filename1\$, filename2\$, outfile\$

where coherent is 0 for incoherent or other integer for coherent summation, filename1\$, filename2\$, outfile\$ are the names of two source files and the target file. If an incoherent sum is performed, the output data will be real valued only. If the two source files do not have the same number of data points, point spacing, and reference radii in both x and y directions, then the second source file listed is first scaled and interpolated, and the reference radii is adjusted to match the first file before the

summation is performed. The length units in the ZBF files are converted automatically to the current lens units. The outfilename may be the same as one of the source file names, in which case the original file is overwritten.

Keyword ZBFTILT is used to multiply the data in a ZBF file by a complex phase factor to introduce phase tilt to the beam. The syntax is:

ZBFTILT filename\$, cx, cy, tx, ty

where filename\$ is the ZBF file name, cx and cy are the center of the phase tilt, tx and ty are the slopes of the tilt in units of radians per lens unit length. The coordinates x and y refer to positions within the beam file, with the center coordinate (x = 0, y = 0) being at the point (nx/2 + 1, ny/2 + 1) where nx and ny are the number of points in the x and y directions. The length units in the ZBF file are converted automatically to the current lens units. The resulting data is written back to the same file name.

Keyword ZBFWRITE is used to write electric field and beam property data arrays to a ZBF file. The syntax is:

ZBFWRITE filename\$, beamname, propertyname

where filename\$ is the ZBF file name, beamname and propertyname are two arrays defined by a previous call to DECLARE. The beamname must be a 3 dimensional array, of minimum size (nx, ny, 2) for an unpolarized beam and minimum size (nx, ny, 4) for a polarized beam. The propertyname array must be a one dimensional array of minimum size 14. The following beam data must be placed in the specified propertyname array: nx, ny, dx, dy, waist_x, waist_y, position_x, position_y, rayleigh_x, rayleigh_y, wavelength (in lens units), total power, peak irradiance (power per area), the is_polarized flag (0 for no, 1 for yes), and the media index. The values are placed in array positions 1 through 15. The electric field data must be placed in the beamname array. The third dimension of the beamname array is 1 for Ex Real, 2 for Ex Imaginary, and if the beam is polarized, 3 for Ey Real, and 4 for Ey Imaginary.

Now we will show how to use ZBF related commands in ZPL with an example. Assume the optical system is the doublet defined in example 3.04-1, as shown in figure 3.14-1. Since we will use previously save file in this example, we don't have any special requirements to the optical system.

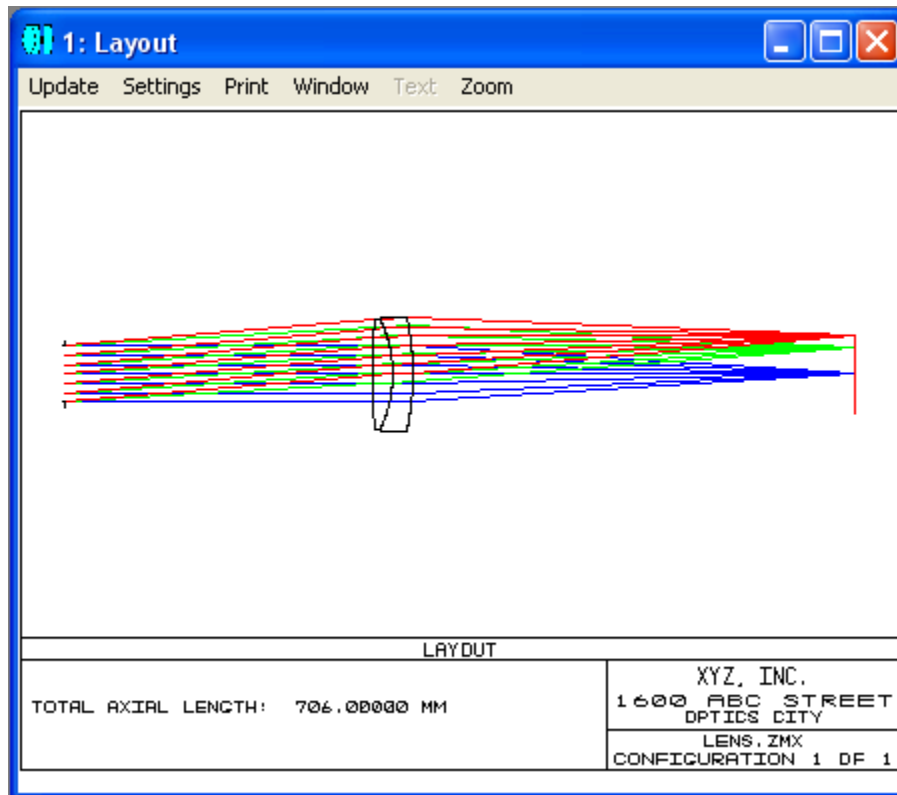


Fig. 3.14-1: Optical system associated to file "ex30910b.ZBF"

When we discuss POP in section 9 of this chapter, we saved file "ex30910b.ZBF". The data saved are the beam data of the 0 degree incident light when it reaches the last surface. The example below will use this file. Please note that the ZBF file was saved in folder "...\\POP\\Beamfiles".

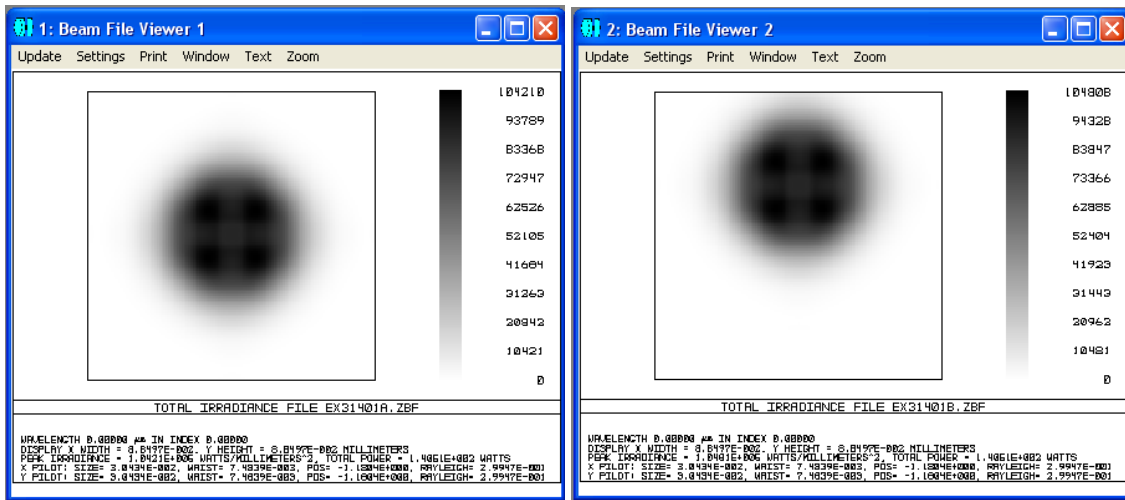
Example 3.14-1: Usage of ZBF related keywords.

```

1 ! ex31401
2 ! This program shows application of some ZBF related key words.
3 ! Assume file "ex30910b.ZBF" already exists.
4
5 oldFile$ = "ex30910b.ZBF"
6 newFile1$ = "ex31401a.ZBF"
7 newFile2$ = "ex31401b.ZBF"
8 newFile3$ = "ex31401c.ZBF"
9 newFile4$ = "ex31401d.ZBF"
10
11 ZBFPROPERTIES oldFile$, 1
12 nx = vec1(1)
13 ny = vec1(2)
14 ip = vec1(14)          # check if is polarization
15
16 ! Allocate enough memory to hold the beam data
17 IF (ip == 0) THEN DECLARE beamArray, DOUBLE, 3, nx, ny, 2
18 IF (ip == 1) THEN DECLARE beamArray, DOUBLE, 3, nx, ny, 4
19 DECLARE propertyArray, DOUBLE, 1, 15
20
21 ZBFREAD oldFile$, beamArray, propertyArray
22 ZBFWRITE newFile1$, beamArray, propertyArray
23 ZBFWRITE newFile2$, beamArray, propertyArray
24 ZBFSHOW newFile1$
25
26 decenterx = 0
27 decentery = 0.015
28 ZBFRESAMPLE newFile2$, 0, 0, 0, 0, decenterx, decentery
29 ZBFSHOW newFile2$
30
31 ZBFSUM 1, newFile1$, newFile2$, newFile3$      ! coherent summation
32 ZBFSHOW newFile3$
33
34 ZBFSUM 0, newFile1$, newFile2$, newFile4$     ! incoherent summation
35 ZBFSHOW newFile4$
36
37 CLOSEWINDOW          # run in "quiet" mode

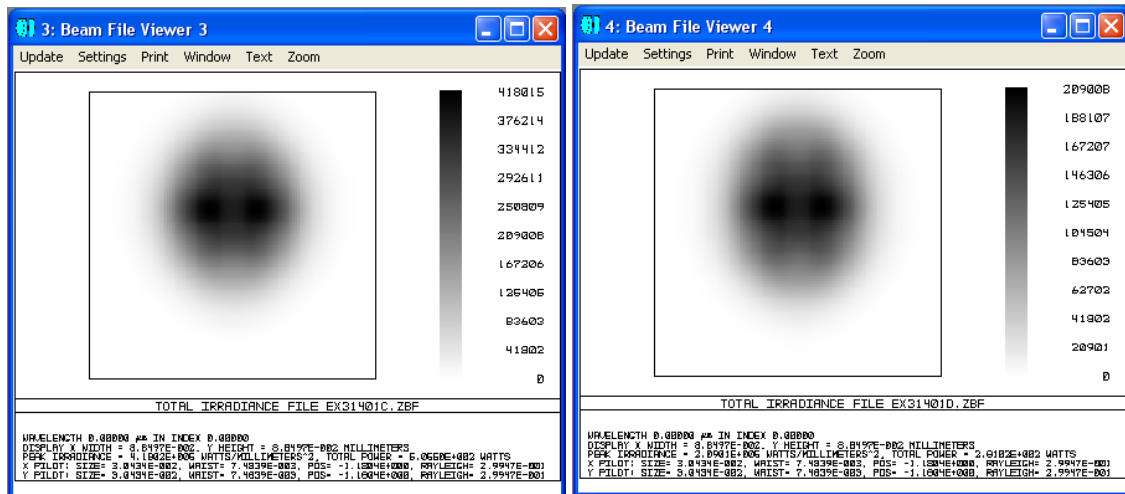
```

In this program, we use keyword ZBFPROPERTIES to read the beam properties in the source file, and defined two arrays beamArray and propertyArray (lines 11 ~ 19). Line 21 reads the beam data in the source file, lines 22 and 23 save the data into two new files for later use. After that, we use keyword ZBFRESAMPLE to shift original beam in Y direction for 0.015 lens unit (lines 26 ~ 28), then do the coherent summation (line 31) and incoherent summation (line 34) using the two beams, and save the result into two different files. We also use keyword ZBFSHOW to open a viewer window to view different files. The results are shown below:



(a)

(b)



(c)

(d)

Fig. 3.14-2: The content in the viewer window after program execution.

(a) ~ (d) are original beam, shifted beam, coherent sum beam, and incoherent sum beam.

Chapter 4

ZPL Application Examples

In this chapter, we will give some ZPL examples for real applications. From these examples we can see that with the aid of ZPL, a lot of tedious routine work can be finished by computer, and our work efficiency can be greatly improved. As an optical engineer, if one can master the tool of ZPL, he can do his design much faster and more flexible.

4.1 Sequential Optical Systems

The examples given in this section involve only sequential optical system.

Example 4.1-1: Basic ray-tracing parameters.

In this example, we will let the user input H_x , H_y , P_x , P_y to define a light ray, and calculate the coordinates, incident angle and exit angle at the intersection point of the light ray and each surface in the optical system. The program is shown below:

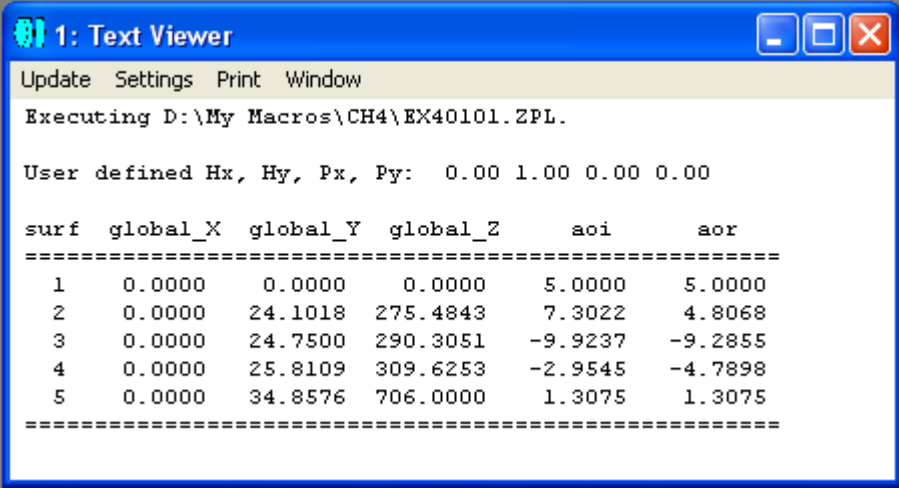
```

1 ! ex40101
2 ! This program traces a given ray and generate the table of ray data.
3
4 ! first, let user define the ray
5 INPUT "Please enter the value of Hx: ", hx
6 INPUT "Please enter the value of Hy: ", hy
7 INPUT "Please enter the value of Px: ", px
8 INPUT "Please enter the value of Py: ", py
9
10 ! print the top frame
11 PRINT
12 FORMAT 5.2
13 PRINT "User defined Hx, Hy, Px, Py: ", hx, hy, px, py
14 PRINT
15 PRINT "surf  global_X  global_Y  global_Z    aoi    aor"
16 PRINT "-----"
17
18 ! trace the ray for entire system:
19 RAYTRACE hx, hy, px, py
20
21 ! calculate ray data at each surface
22 FOR s = 1, NSUR(), 1
23   normal = 57.29577951*ATAN(RANY(s)/RANZ(s))
24   slope  = 57.29577951*ATAN(RAYM(s-1)/RAYN(s-1))
25   aoi = (slope - normal) # in degrees
26   slope  = 57.29577951*ATAN(RAYM(s)/RAYN(s))
27   aor = (slope - normal) # in degrees
28   FORMAT 2 INT
29   PRINT " ", s,
30   FORMAT 10.4
31   PRINT RAGX(s), RAGY(s), RAGZ(s), aoi, aor
32 NEXT
33
34 ! print the bottom frame
35 PRINT "-----"

```

In this program, lines 5 ~ 8 ask the user to define the light ray, line 19 traces the ray, lines 22 ~ 32 calculate the parameters of each intersection points, wherein line 23 calculates the normal direction of each surface, line 24 calculates the ray direction before it hits each surface, and line 26 calculates the ray direction after it leaves each surface.

This program is a general program and works for different optical systems. If we assume the optical system is the doublet defined in example 3.4-1, and the user-defined ray is $H_x = 0$, $H_y = 1$, $P_x = 0$, and $P_y = 0$, then the result after execution is:



```

1: Text Viewer
Update Settings Print Window
Executing D:\My Macros\CH4\EX40101.ZPL.

User defined Hx, Hy, Px, Py:  0.00 1.00 0.00 0.00

surf  global_X  global_Y  global_Z    aoi    aor
=====
  1    0.0000    0.0000    0.0000    5.0000    5.0000
  2    0.0000    24.1018   275.4843    7.3022    4.8068
  3    0.0000    24.7500   290.3051   -9.9237   -9.2855
  4    0.0000    25.8109   309.6253   -2.9545   -4.7898
  5    0.0000    34.8576   706.0000    1.3075    1.3075
=====

```

Fig. 4.1-1: Result of program ex40101.ZPL

Example 4.1-2: Light spot near focal plan.

In this example, we move the image plan around the focal plan by changing the thickness of the surface just before the image surface, and observe the change of the size of the light spot on the image plan. The program is shown below:


```

1 ! ex40102
2 ! This program shows how to modify lens data and update graphic window.
3 ! It can be used to create animation.
4
5 surface = NSUR()-1      # serial number of the second to the last surface
6 z0 = THIC(surface)     # the original thickness of that surface
7
8 str1$ = "Please enter the shift from focus (0 ~ " + $STR(z0) + "):"
9 str2$ = "Please enter the total steps (positive integer): "
10
11 ! let user define the offset and total steps
12 LABEL 1
13 INPUT str1$, shift
14 IF ((shift < 0)|(shift > z0)) THEN GOTO 1
15
16 LABEL 2
17 INPUT str2$, stepNum
18 IF (stepNum < 1) THEN GOTO 2
19
20 stepNum = INTE(stepNUM)
21
22 startZ = z0 - shift
23 endZ = z0 + shift
24 stepSize = shift*2/stepNum
25
26 prefix$ = "D:\My Macros\ch4\ex40102 shift "
27 ext$ = ".WMF"
28
29 FOR z = startZ, endZ, stepSize
30   FORMAT 3.1
31   shift$ = $STR(z-z0)
32   fileName$ = prefix$ + shift$ + ext$
33   value = z
34   SURP surface, THIC, value
35   UPDATE 1      # assume serial number of the graphic window is 1
36   EXPORTBMP 1, fileName$
37 NEXT
38
39 value = z0      # put back the original thickness
40 SURP surface, THIC, value
41 UPDATE ALL
42
43 CLOSEWINDOW

```

This program is also a general program. As an illustration, we assume the optical is the doublet defined in example 3.4-1. Before running the program, open the Spot Diagram window in Zemax, and set the system according to figure 4.1-2:

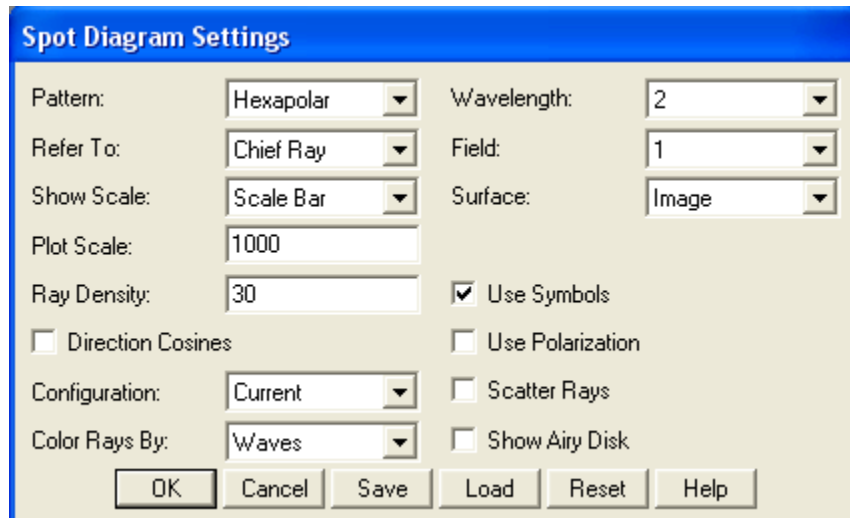


Fig. 4.1-2: Spot Diagram settings

In this program, the user is first asked to input the range and step number of the image plan shift (lines 12 ~ 18), and then the program will evaluate the user input, requiring shift range be positive but not larger than the thickness of the surface just before the image plan, and step number be positive, too. If the input cannot meet the requirement, then the user will be asked to input again. Line 20 assures the step number is a positive integer. After that, the program calculates the starting and stopping Z coordinates and the step size (lines 22 ~ 24), and in each loop, generates a new file name based on the shift value of the image plan (line 32), modifies the image plan position (line 34), updates the Spot Diagram window (line 35), and outputs the content of the window to the target file (line 36). Lines 39 ~ 41 restore the original image plan location to assure the original optical system is not impacted by the program.

During the execution of the program, we can see that the light spot size in the Spot Diagram is changed. After the execution, the following files are generated in the given folder (assume our inputs are shift = 5, step = 10):

```
ex40102 shift -5.0.BMP  
ex40102 shift -4.0.BMP  
ex40102 shift -3.0.BMP  
ex40102 shift -2.0.BMP  
ex40102 shift -1.0.BMP  
ex40102 shift 0.0.BMP
```

ex40102 shift 1.0.BMP

ex40102 shift 2.0.BMP

ex40102 shift 3.0.BMP

ex40102 shift 4.0.BMP

ex40102 shift 5.0.BMP

wherein the content of the 1st, 3rd and 6th file is shown in figure 4.1-3:

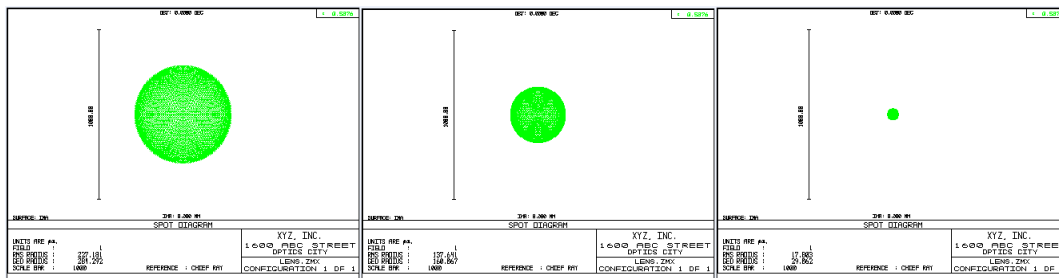


Fig. 4.1-3: Content of some of the files generated from program ex40102.ZPL

With the files generated from this program, it's not hard to make an animation to demonstrate how the light spot size on the image plan changes with the shift of the plan.

Example 4.1-3: Geometrical beam and Gaussian beam comparison

In this example, we will discuss how to read Zemax analysis data by comparing the beam size at each surface (Y coordinate of the intersection of the light ray and the surface) obtained from geometrical beam method and Gaussian beam method. Assume the optical system is the doublet defined in example 3.4-1. Since Zemax has a default beam waist radius of 0.05 lens unit when calculating Gaussian beam, we need to modify this number according to our system. Create a new lens file in Zemax, run program EX30401.ZPL, and open a paraxial Gaussian beam analysis window from menu Analysis → Physical Optics → Paraxial Gaussian Beam, press the right button of the mouse to pop out the setting dialog, change the beam waist size to be 25, as shown in figure 4.1-4, and then save the settings.

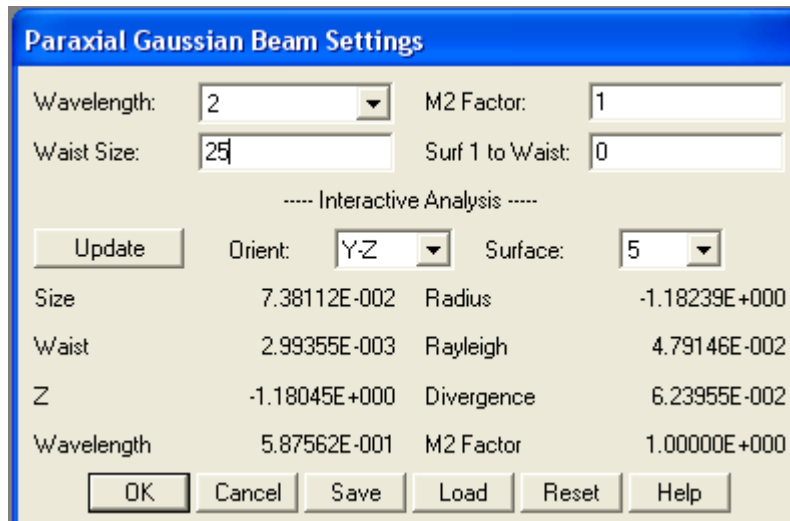


Fig. 4.1-3: Paraxial Gaussian Beam Settings for example Ex4.1-3

```

1 ! ex40103
2 ! This program compares the Geometrical beam size and Gaussian beam size
3 ! Assume the lens system is defined in ex30401
4
5 PI = 3.14159265
6
7 ! define a marginal ray
8 hx = 0
9 hy = 0
10 px = 0
11 py = 1
12
13 ! trace the ray for entire system:
14 RAYTRACE hx, hy, px, py
15
16 ! obtain Gaussian beam data
17 A$ = $TEMPFILENAME() # get a temporary file to store Gaussian beam data
18 GETTEXTFILE A$, Gbp # store Gaussian beam data in the temp file
19 OPEN A$ # open the newly created temp file
20 row = 0 # line number in the file
21 s = 0 # surface number
22
23 LABEL 1
24
25 READSTRING B$ # read a line in the file
26 row = row + 1 # line number
27 IF row < 28 THEN GOTO 1 # line 28 is the data of the first surface
28 s = s + 1 # surface number
29 temp$ = $GETSTRING(B$, 2) # the second sub-string
30 VEC1(s) = SVAL(temp$) # convert to number, and stored in an array
31 temp$ = $GETSTRING(B$, 1) # the first sub-string
32 IF (temp$ $! = "IMA") THEN GOTO 1 # repeat until reaching IMA surface
33 CLOSE # close file
34 DELETEFILE A$ # delete the temp file
35

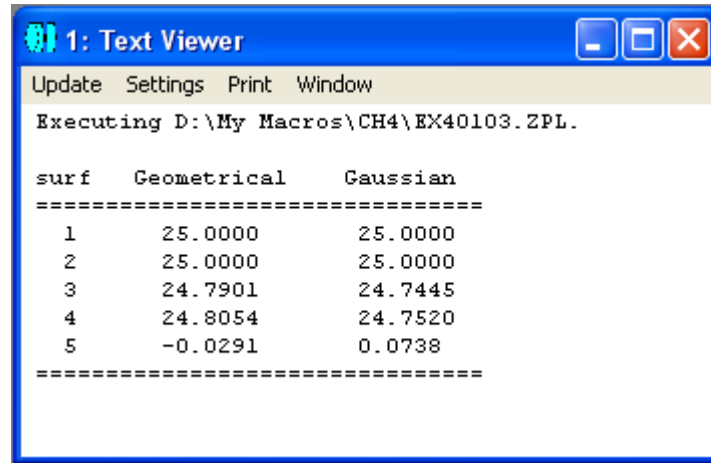
```

In this program, line 14 does the ray-tracing to obtain the geometrical beam size, i.e. the Y coordinate of the intersection of the marginal ray and the surface. Lines 17 ~ 34 obtain the Gaussian beam data, wherein line 17 generates a temporary file, line 18 stores the content of the paraxial Gaussian beam analysis window into the temporary file, line 19 opens the temporary file to read the data, and line 25 reads a whole line in the file. Since the file format is fixed, and we know the 28th line in the file is the data of the first surface, so if the line number is smaller than 28, the program jumps back to label 1 to continue to read next line, until it reaches line 28. Lines 29 and 30 of the program convert the 2nd string (Gaussian beam radius) into a number, and store it in the default vector VEC1. Lines 31 and 32 evaluate the first string of the read data and see if it is "IMA", if yes, it means the data of the last surface has been reached, otherwise the program goes back to label 1 and continues to read next line. After the data of the last surface is read, line 33 closes the temporary file, and line 34 deletes the file.

```
35
36 ! print header
37 PRINT
38 PRINT "surf   Geometrical   Gaussian"
39 PRINT "===== "
40
41 FOR s = 1, NSUR(), 1
42   FORMAT 1 INT
43   PRINT " ", s,
44   FORMAT 6.4
45   PRINT " ", RAGY(s), " ", VEC1(s)
46 NEXT
47
48 PRINT "===== "
49
```

Lines 37 ~ 48 display the result on the screen, wherein function RAGY() is the result of the geometrical ray tracing, and array VEC1() is the Gaussian beam result read from the temporary file.

The result of the program is shown in figure 4.1-5:



```
1: Text Viewer
Update Settings Print Window
Executing D:\My Macros\CH4\EX40103.ZPL.

surf   Geometrical   Gaussian
=====
  1     25.0000     25.0000
  2     25.0000     25.0000
  3     24.7901     24.7445
  4     24.8054     24.7520
  5     -0.0291     0.0738
=====
```

Fig. 4.1-5: Result of program ex40103.ZPL

Example 4.1-4: Comparison of transmission property of different glass materials

In optical design, sometimes we need to search for glass materials with special transmission properties. For example, we may want to find glass that has highest transmission in the visible wavelength range (400nm ~ 700nm), but has biggest absorption at UV and infrared wavelengths. In this example, we will search for such glass material among the thousands of glasses in Zemax database. Assume our interested wavelength range is 380 ~ 1000nm, and we hope to get lowest transmission in 380 ~ 400nm and 700 ~ 1000nm range, but highest transmission in 400 ~ 700nm range. We will set the thickness of the glass to be 1mm, calculate transmission at different wavelength, convert the transmission value to absorption value in the range of 380 ~ 400nm and 700 ~ 1000nm, and define a merit value as the root mean square of all those values across the whole interested wavelength range. Each glass material will have one merit value. After sorting all the merit values, we can choose those with highest merit values as our candidates.

```
1 ! ex40104
2 ! This program compares different glasses to find the best profile.
3 ! Assume the target is to get lowest transmission in 380~400nm, 700~1000nm
4 ! and get highest transmission in 400~700nm
5
6 ! define some parameters
7 startWav = 0.38      # start wavelength, 380 nm
8 stopWav = 1         # stop wavelength, 1000 nm
9 stepNum = 500       # total steps between start and stop wavelengths
10 gm = 0             # initial total glass number of all the catalogs
11
12 saveName$ = "D:\My Macros\ch4\ex40104.txt"
13
14 SETVECSIZE 10000
15
16 GOSUB calculate      # calculate the merit value for each glass
17 GOSUB sortResult    # sort the results based on merit value
18 GOSUB reportResult  # print result on screen and save in a file
19
20 END
21
22
```

Lines 1 ~ 20 is the main program, wherein lines 7 ~ 10 define some basic parameters, and line 12 gives the output file name. We will use the default Zemax vectors to store glass data. Since the total number of glasses is bigger than 1000, line 14 increases the vector size to 10000. Line 16 calls sub-program “calculate” to calculate merit value of each glass material, line 17 calls sub-program “sortResult” to sort the merit values, and line 18 calls sub-program “reportResult” to output the final result.

```

22
23 SUB calculate
24
25 originalWave = WAVL(1)      # store the original first wavelength
26
27 FOR cNum = 1, 16, 1         # go through the catalogs
28   GOSUB getCatalog         # set catalog
29   g = 1                    # initial glass number
30   LABEL 1
31   merit = 0                # initial merit
32   FOR s = 1, stepNum+1, 1   # go through the wavelengths
33     w = startWav+(s-1)*(stopWav-startWav)/stepNum
34     SYSP 202, 1, w          # set the wavelength
35     GETGLASSDATA 1, g       # store glass data in VEC1
36     alpha = VEC1(23)        # Internal transmission coefficient
37     IF alpha < 0 THEN alpha = -1*alpha    # correct some catalog errors
38     t = EXPE(-1*alpha)      # transmission, assume 1 mm length
39     IF ((w < 0.4) || (w > 0.7)) THEN t = 1-t    # need absorption
40     merit = merit+t*t       # updated merit
41   NEXT s
42   gm = gm + 1              # total glass number increase by 1
43   VEC2 (gm) = cNum         # store catalog number
44   VEC3 (gm) = g            # store glass number of the catalog
45   VEC4 (gm) = SQR(merit)   # store merit value
46   FORMAT 1 INT
47   REWIND
48   PRINT gm, " processed. ", "Now processing glass ", g, " of ", cName$
49   g = g + 1                # glass number of the catalog increase by 1
50   GETGLASSDATA 1, g       # store new glass data
51   IF VEC1(1) > 0 THEN GOTO 1 # if formula is valid, go back to process
52 NEXT cNum
53
54 SYSP 202, 1, originalWave  # recover the original first wavelength
55
56 RETURN
57

```

Lines 23 ~ 56 is the sub-program “calculate” used to calculate the merit value of a glass material. Since we need to change the first wavelength value in the system settings, we save the original wavelength value in line 25, and after calculation, we restore the original wavelength value in line 54, so the original optical system remains impacted. The loop in lines 27 ~ 52 evaluates total 16 glass catalogs, wherein line 28 calls sub-program “getCatalog” to load glass catalogs in serial, and the loop in lines 32 ~ 41 calculates transmission or absorption at each wavelength, and calculates the merit value (note the square root hasn’t been calculated yet). We found that in the glass catalog data provided by Zemax, sometimes the transmission coefficient alpha is positive, and sometimes negative, so we change them to be positive in line 37, and calculate the transmission based on positive coefficient in line 38. Line 39 evaluates the wavelength, and if it is out of visible range, then use absorption instead of transmission to calculate the merit value. After finish calculating the merit value of a glass type, line 42 increases the total glass number, and lines 43 ~ 45 store the corresponding catalog number, glass number in the catalog, and merit value into default vectors VEC2, VEC3 and VEC4, respectively. Since the calculation needs some time, lines 46 ~ 48 display the current progress on the screen. Keyword REWIND is used to

assure the display is always on the same line. Figure 4.1-6 shows the screen content at a certain moment during the program execution. After that, line 49 increases the glass number in the catalog by 1, line 50 reads the new glass parameters, line 51 evaluates if the glass is valid (We use the method of judging the code of dispersion equation type. The code is a positive integer for a valid glass type, 0 for an invalid glass type), if yes then program jumps to label 1 to process the new glass, otherwise, the end of a catalog is reached, so the program proceeds to the next glass catalog in the loop.

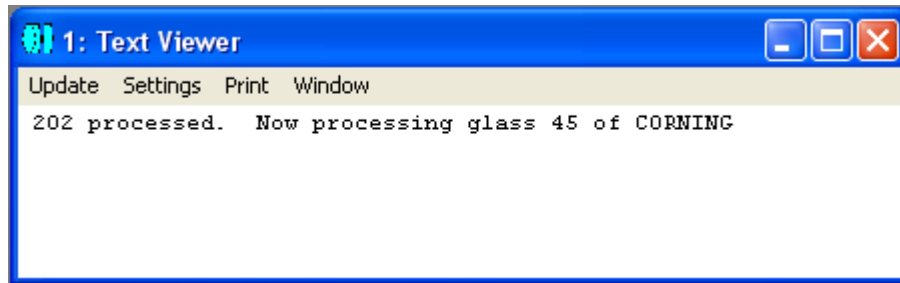


Fig. 4.1-6 The progress shown on the screen during the execution of ex40104.ZPL.

```

57
58 SUB getCatalog
59
60 IF cNUM == 1 THEN cName$ = "ARCHER"
61 IF cNUM == 2 THEN cName$ = "CDGM"
62 IF cNUM == 3 THEN cName$ = "CORNING"
63 IF cNUM == 4 THEN cName$ = "HERAEUS"
64 IF cNUM == 5 THEN cName$ = "HIKARI"
65 IF cNUM == 6 THEN cName$ = "HOYA"
66 IF cNUM == 7 THEN cName$ = "LIGHTPATH"
67 IF cNUM == 8 THEN cName$ = "LZOS"
68 IF cNUM == 9 THEN cName$ = "OHARA"
69 IF cNUM == 10 THEN cName$ = "PILKINGTON"
70 IF cNUM == 11 THEN cName$ = "RPO"
71 IF cNUM == 12 THEN cName$ = "SCHOTT"
72 IF cNUM == 13 THEN cName$ = "SUMITA"
73 IF cNUM == 14 THEN cName$ = "TOPAS"
74 IF cNUM == 15 THEN cName$ = "UMICORE"
75 IF cNUM == 16 THEN cName$ = "ZEON"
76
77 SYSP 23, cName$
78 LOADCATALOG
79
80 RETURN
81
82

```

Lines 58 ~ 80 is the sub-program “getCatalog”. Its function is to assign the glass catalog name according to the catalog number, and load the catalog.

```

82
83 SUB sortResult
84
85 FOR i = 1, gm, 1           # go through all the stored numbers
86   FOR j = i, 1, -1       # go through all processed numbers
87     IF j > 1             # if j is not the first
88       IF VEC4(j) > VEC4(j-1) # if current merit is larger
89         temp = VEC4(j-1)
90         VEC4(j-1) = VEC4(j)
91         VEC4(j) = temp
92         temp = VEC3(j-1)
93         VEC3(j-1) = VEC3(j)
94         VEC3(j) = temp
95         temp = VEC2(j-1)
96         VEC2(j-1) = VEC2(j)
97         VEC2(j) = temp
98       ENDIF
99     ENDIF
100   NEXT j
101   FORMAT 1 INT
102   REWIND
103   PRINT i, " of total ", gm, " glasses sorted"
104 NEXT i
105
106 RETURN
107
108

```

Lines 83 ~ 106 is the sub-program “sortResult”. Its function is to sort the glasses stored in the default vectors according to the merit values stored in VEC4 using a common bubble sorting method. Similarly, lines 101 ~ 103 displays the progress during sorting, as shown in figure 4.1-7.



Fig. 4.1-7 Progress display during sorting of glasses.

```

108
109 SUB reportResult
110
111 INSERT 1                # insert a surface temporarily
112
113 REWIND
114 PRINT
115 PRINT "Catalog          Glass          Merit          "
116 PRINT "-----"
117 FORMAT 6.5
118 FOR i = 1, gm, 1
119   cNum = VEC2(i)        # get the catalog number
120   GOSUB getCatalog     # load the catalog
121   g = VEC3(i)          # get the glass number in the catalog
122   SURP 1, GLAN, g      # set the glass material for the surface
123   glassName$ = $GLASS(1) # read back the glass name
124   PRINT cName$, "      ", glassName$, "      ", VEC4(i)
125 NEXT
126
127 DELETE 1              # delete the temporary surface
128
129 ! save the result
130 SAVEWINDOW 1, saveName$ # assume there is no other windows open
131
132 RETURN
133

```

Lines 109 ~ 132 is sub-program “reportResult”. It’s used to output calculation result. In this sub-program, lines 113 ~ 116 display a table head on the screen, 118 ~ 125 use a FOR loop to print all the sorted glass catalog names, glass names and merit values. Since we only know the glass number, in order to print the glass name, we need to insert a temporary surface (line 111) in the lens data editor, set the surface material type as the glass with known number, and read back the glass name using function \$GLASS() in line 123. After all the glasses are displayed, line 127 deletes the temporary surface, and restore the original optical system. Line 130 saves the content displayed in the text viewer window to a target file.

Figure 4.1-8 shows partial of the result of the program displayed in the text viewer window. It needs to be pointed out that our program assumes the data in Zemax glass catalogs are all correct. Sometimes this assumption may not be true, and the designer needs to make his or her own judgement on the result of the program.

Catalog	Glass	Merit
HOYA	CF6	21.08480
HOYA	FL6	21.08473
HOYA	M-NBF1	21.08413
HOYA	M-BACD12	21.08299
HOYA	ADF10	21.08280
HOYA	C3	21.08245
HOYA	ADF50	21.08218
HOYA	BACD12	21.08082
HOYA	FF5	21.08043
HOYA	M-LAC130	21.07990
HOYA	BACED20	21.07885
HOYA	TAF5	21.07863
HOYA	ADC1	21.07848
HOYA	M-NBFD82	21.07559
HOYA	M-LAF81	21.07436
HOYA	BAF10	21.07379
HOYA	M-NBFD13	21.07379
HOYA	E-FDS1	21.02132
RPO	E-FDS1_MOLD	21.02132
OHARA	L-TIM28	20.87823
RPO	L-TIM28_MOLD	20.87823
OHARA	PBL1	20.12615
OHARA	BSL21	20.12582
OHARA	PBL2	20.12579
OHARA	PBL25	20.12555
OHARA	PBL6	20.12509
OHARA	PBL21	20.12445
OHARA	BSM16C	20.12442
RPO	BSM16C_MOLD	20.12442

Fig. 4.1-8 Partial content of the result of program ex40104.ZPL

displayed in the text viewer window

Finally, as a verification, we compare three glasses CF6, BK3 and SF5 in the sorting list with merit value from high to low, and display their transmission curves in figure 4.1-9. From the plot we can see that our sorting result is reasonable. The transmission data shown in figure 4.1-9 come from Zemax glass catalog. We will describe how to get those data in next example.

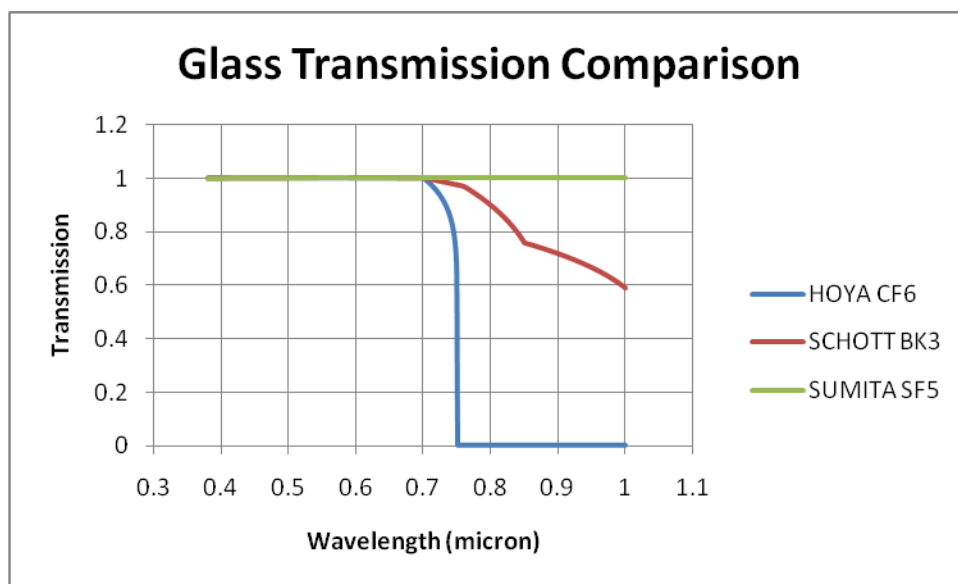


Fig. 4.1-9 Comparison of internal transmission of 1mm-thick glass plate

A question to readers: in real application, it might be better to combine two different glasses to get desired transmission property. So how to do it in ZPL program?

Example 4.1-5: Read refractive index and transmission data of catalog glass.

Zemax catalogs provides many glass property data such as refractive index, transmission, etc, and they can be very helpful in real applications. We wrote this simple program to show that we can easily get those data through ZPL program.

Lines 8 ~ 12 of the program ask the user to input the glass material name, thickness, start wavelength, stop wavelength, and step number. Line 15 set wavelength 1 as the primary wavelength, so line 26 can read the refractive index of that wavelength. In the loop of lines 21 ~ 34, line 22 calculates each wavelength value, line 23 sets the glass material of surface 1 as the user defined glass, line 24 sets wavelength 1 (i.e. the primary wavelength) as the calculation wavelength, line 25 updates the system settings, line 26 reads the refractive index associated to the primary wavelength, lines 27 ~ 29 read the transmission coefficient of glass, and as in last example, line 30 evaluates the coefficient, and treat it as a positive value, and then line 31 calculates the transmission of the given thickness glass, line 33 displays the result to the screen as a table. Finally, line 36 saves the content of the text viewer window to a target file. In figure 4.1-9 of last example, the data we displayed were read with this program.

Before we run this program, we can create a new lens file, and we don't need to save the lens file after we run the program.

```

1 ! ex40105
2 ! This program lists refractive index and internal transmission of glass
3 ! The glass type and wavelength range is given by user
4
5 saveName$ = "D:\My Macros\ch4\ex40105.txt"
6
7 ! let user define the glass type and wavelength range
8 INPUT "Please enter the glass type", glassType$
9 INPUT "Please enter the glass thickness (mm)", glassThickness
10 INPUT "Please enter the starting wavelength (um): ", startWave
11 INPUT "Please enter the stopwavelength (um): ", stopWave
12 INPUT "Please enter the number of steps: ", stepNum
13
14 ! set the first wavelength as primary wavelength
15 PWAV 1
16
17 PRINT
18 PRINT "Wavelength(um)      Refractive Index      Internal Transmission"
19 PRINT "=====
20
21 FOR i = 1, stepNum+1, 1
22   w = startWave+(i-1)*(stopWave-startWave)/stepNum # calculate wavelength
23   SURF 1, GLAS, glassType$ # set the glass material for surface 1
24   SYSP 202, 1, w # set the new wavelength for the primary
25   UPDATE
26   index = INDX(1) # get the refractive index of primary wave at surface 1
27   glassType$ = $GLASS(1)
28   GETGLASSDATA 1, GNUM(glassType$) # store glass data in VEC1
29   alpha = VEC1(23) # Internal transmission coefficient
30   IF alpha < 0 THEN alpha = -1*alpha # correct some catalog errors
31   t = EXPE(-1*alpha*glassThickness) # transmission, length unit is mm
32   FORMAT 8.7
33   PRINT " ", w, " ", index, " ", t
34 NEXT
35
36 savewindow 1, saveName$

```

4.2 Non-Sequential Optical System

The examples given in this section involve only non-sequential optical system.

Example 4.2-1: Light Pipe.

In optical design, we often use light pipe to guide the light. The simplest light pipe is a cylinder that is made of glass or plastic. Light can enter from one end of the pipe, and come out from the other end. The cross section of the light pipe can be round or other shape such as triangle, rectangle or hexagon, etc. When designing light pipe with different lengths and different cross sections, we need to consider light coupling ratio, light spot uniformity, and other factors. In this example, we will compare the maximum output light intensity, total flux and spot uniformity of light pipes with different length and shape.

First, we assume the light source is a circular disk with Lambertian light distribution. The location of the light source is at the origin, the normal of the source is along +Z direction, and the size is smaller than the light pipe cross section. We also assume the area of the cross section of the light pipe is fixed, the shape can be equilateral polygon with 3 to 8 sides, and the length can be 10, 40, 160 or 640 lens unit. The light pipe also starts at origin $Z = 0$, with its length along +Z direction. The material of the pipe is Acrylic. Finally, we assume a rectangular detector is put at the end of the light pipe to collect the output light from the pipe. The size of the detector is slightly larger than the circumference of the light pipe. The whole system is shown in figure 4.2-1:

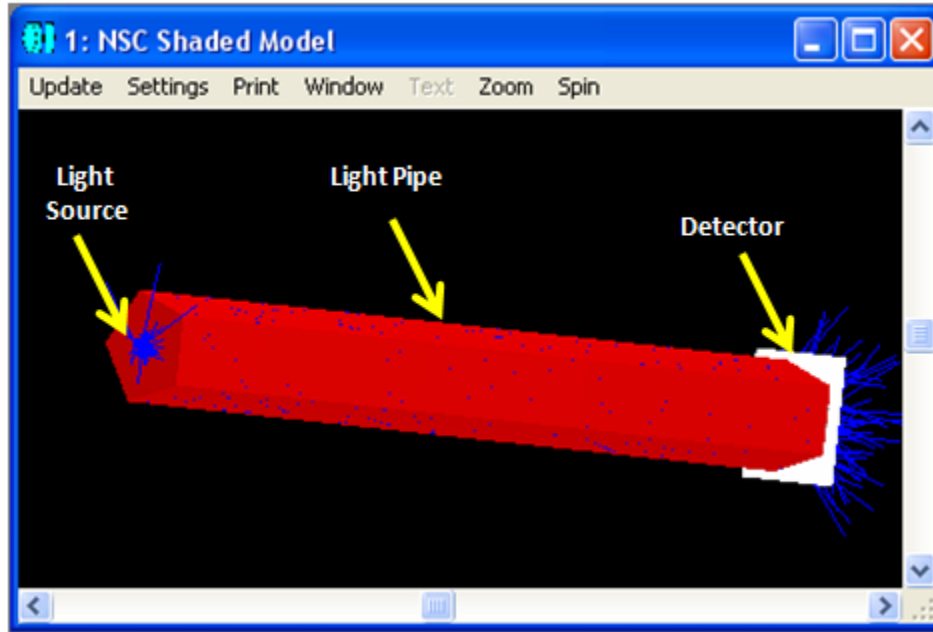


Fig. 4.2-1: the optical system in example ex40201.ZPL

Since this optical system is very simple, we can easily construct the whole system from scratch in the ZPL program, change the system parameters in a loop, and do the ray tracing to get the final result.

```

1 ! ex40201
2 ! This program compares output of different shape and length light pipes.
3 ! Assume the mode is total Non-Sequential Mode.
4
5 ! define constant and parameters
6 PI = 3.14159265
7 objPath$ = "D:\My Macros\ch4\"
8 fileName$ = "polygon.txt"
9 outputName$ = objPath$ + fileName$           # polygon UDA file
10 crossArea = 1                               # the area of the cross section of polygon
11 sourceRadius = 0.1                          # the radius of a circular source
12 pNum = 100*100                              # total pixel number
13 rayNum = 1000000                            # number of rays to be traced
14
15 ! define result as a 3-dimension array
16 lengthNum = 4                               # assume lengths are 10, 40, 160, 640
17 sideNum = 8                                 # assume sides are 3, 4, 5, 6, 7, 8
18 paraNum = 3                                 # assume to analyze maxFlux, totalFlux, nonUniformity
19 DECLARE result, DOUBLE, 3, lengthNum, sideNum, paraNum
20
21 ! clean and prepare the NSC editor
22 GOSUB prepareEditor
23
24 ! define the first object as Source Ellipse
25 GOSUB defineSource
26
27 ! go through different polygons
28 FOR n = 3, 8, 1                             # different n sided polygons
29     FOR m = 1, lengthNum, 1                 # go through different lengths
30         GOSUB createPolygon
31         GOSUB createDetector
32         GOSUB analyze
33     NEXT m
34 NEXT n
35
36 ! print result
37 GOSUB reportResult
38
39 END

```

Lines 1 ~ 39 are the main program. Among them, lines 6 ~ 13 define some basic constants and parameters, lines 16 ~ 19 define a 3 dimensional array to store the calculation result. After that, line 22 calls sub-program “prepareEditor” to set the non-sequential component editor, line 25 calls sub-program “defineSource” to set up the light source, lines 28 ~ 34 form a loop to do ray tracing and analysis by varying the side number of the polygon of the light pipe cross section and the length of the pipe. Particularly, line 30 calls sub-program “createPolygon” to set up the light pipe, line 31 calls sub-program “createDetector” to set up the detector, and line 32 calls sub-program “analyze” to trace rays and analyze result. After that, line 37 calls sub-program “reportResult” to output result to the screen. In this program, we use modularized code design, and put detailed settings into various sub-programs. This gives us a simple and clear main program, and makes the whole program easier to understand and to debug.

```

40
41
42 SUB prepareEditor
43
44 totalObjNum = NOBJ(1)
45 FOR i, totalObjNum, 1, -1
46   DELETEOBJECT 1, i      # delete all the objects
47 NEXT                    # there is still a null object in the editor at last
48
49 ! add two more objects in the editor, so the total is 3
50 INSERTOBJECT 1, 1
51 INSERTOBJECT 1, 1
52
53 RETURN prepareEditor
54
55

```

Lines 42 ~ 53 are the sub-program “prepareEditor”. Its function is to first delete all the objects in the non-sequential component editor (leaves only a null object at last), and then insert two more null objects so there are total 3 null objects in the editor. In line 53, we add the name of the sub-program “prepareEditor” in order to make the code clearer and easier to read. In fact, Zemax treats the RETURN command the same with or without the name of the sub-program.

```

55
56 SUB defineSource
57
58 ! define the source type
59 SETNSCPROPERTY 1, 1, 0, 0, "NSC_SRCE"
60
61 ! define the position
62 SETNSCPOSITION 1, 1, 3, -0.1      # z
63
64 ! define the source parameters
65 SETNSCPARAMETER 1, 1, 1, 100      # Layout Rays
66 SETNSCPARAMETER 1, 1, 2, rayNum   # Analysis Rays
67 SETNSCPARAMETER 1, 1, 3, 1       # Power
68 SETNSCPARAMETER 1, 1, 6, sourceRadius # X Half Width
69 SETNSCPARAMETER 1, 1, 7, sourceRadius # Y Half Width
70 SETNSCPARAMETER 1, 1, 9, 1       # Cosine Exponent, assume Lambertian
71
72 RETURN defineSource
73
74

```

Lines 56 ~ 72 are the sub-program “defineSource”. Its function is to define the first object in the non-sequential component editor as a circular source, and set it up.

```

73
74
75 SUB createPolygon
76
77 ! calculate r1(from center to a side) and r2(from center to a vertex)
78 r1 = SQRT(crossArea/(n*TANG(PI/n)))
79 r2 = r1/COSI(PI/n)
80
81 ! write the polygon UDA file
82 OUTPUT outputName$
83 PRINT "POL 0 0 ",
84 FORMAT 8.7
85 PRINT r2,
86 FORMAT 1.0
87 PRINT " ", n, " 0"
88 PRINT "BRK"
89 OUTPUT SCREEN
90
91 ! set the second object as extruded
92 SETNSCPROPERTY 1, 2, 0, 0, "NSC_EXTR"
93
94 ! define the filename in the comment
95 SETNSCPROPERTY 1, 2, 1, 0, fileName$      # notice the path is not needed
96
97 ! define the material
98 SETNSCPROPERTY 1, 2, 4, 0, "ACRYLIC"
99
100 ! define the polygon parameters
101 length = 10*POWR(4, m-1)
102 SETNSCPARAMETER 1, 2, 1, length      # length
103 SETNSCPARAMETER 1, 2, 2, 1          # front x scale
104 SETNSCPARAMETER 1, 2, 3, 1          # front y scale
105 SETNSCPARAMETER 1, 2, 4, 1          # rear x scale
106 SETNSCPARAMETER 1, 2, 5, 1          # rear y scale
107
108 RETURN createPolygon
109
110

```

Lines 75 ~ 108 are sub-program “createPolygon”. Its function is to define the second object in the non-sequential component editor as a polygon light pipe, and set it up. The method we choose to create a polygon light pipe is to use Extruded type object in Zemax, generate equilateral polygon through user-defined aperture (UDA) file, and extrude it to a light pipe. In the program, line 78 calculates the radius of the inscribed circle of the polygon, line 79 calculates the radius of the circumcircle of the polygon, lines 82 ~ 89 define the UDA file. Please note that line 82 sets the output to a file, line 89 sets the output back to the screen, so it is guaranteed that the output in the rest of the program uses the default

setting, i.e. the screen. Line 92 sets up the object type, line 95 defines the UDA file name, where the path name is not needed, and the file should be stored in the objects folder (defined by Zemax main menu, File → Preferences → Directories), which is “D:\My Macros\ch4” in this example. Lines 98 ~ 106 set up the material and other parameters.

```
109
110
111 SUB createDetector
112
113 ! define the type of the third object as Detector Rect
114 SETNSCPROPERTY 1, 3, 0, 0, "NSC_DETE"
115
116 ! define the position of the second object
117 z = length + 0.01
118 SETNSCPOSITION 1, 3, 3, z      # z
119
120 ! define the detector parameters
121 halfWidth = r2*1.1      # set the detector slightly larger than the pipe
122 SETNSCPARAMETER 1, 3, 1, halfWidth      # X Half Width
123 SETNSCPARAMETER 1, 3, 2, halfWidth      # Y Half Width
124 SETNSCPARAMETER 1, 3, 3, 100           # Number of X Pixels
125 SETNSCPARAMETER 1, 3, 4, 100           # Number of Y Pixels
126
127 RETURN createDetector
128
129
```

Lines 111 ~ 127 are the sub-program “createDetector”, defining the type, position, size and pixel numbers.

```

128
129
130 SUB analyze
131
132 temp = NSDD(1, 0, 0, 0) # clear the detector
133 NSTR 1, 1, 1, 1, 1, 1, 1, 0 # ray tracing
134
135 maxFlux = 1E-8
136 sumFlux = 0
137 sumSquare = 0
138 totalCount = 0
139 FOR pix = 1, pNum, 1
140   pFlux = NSDD(1, 3, pix, 0) # check light flux on each pixel
141   IF maxFlux < pFlux THEN maxFlux = pFlux
142   IF pFlux > 1E-8
143     sumFlux = sumFlux + pFlux
144     sumSquare = sumSquare + pFlux*pFlux
145     totalCount = totalCount + 1 # only count those illuminated pixels
146   ENDIF
147 NEXT
148
149 temp = totalCount*maxFlux*maxFlux-2*maxFlux*sumFlux+sumSquare
150 ripple = Sqrt(temp)/(maxFlux*Sqrt(totalCount))
151 result(m, n, 1) = maxFlux/(halfWidth*halfWidth*4/pNum) # intensity on pixel
152 result(m, n, 2) = sumFlux
153 result(m, n, 3) = ripple
154
155 RETURN analyze
156
157

```

Lines 130 ~ 155 are sub-program “analyze”. Line 132 clears the detector, line 133 does the ray-tracing, lines 139 ~ 147 read the light flux on each pixel (line 140), evaluate if the light flux value on the current pixel is larger than the maximum flux value (line 141), if yes then replace the max flux value with the current pixel flux value. The goal is to find out the maximum flux value of all the pixels. Also, if the current pixel flux value is larger than 0 (line 142), then add it into the total light flux “sumFlux” and summation of square of the light flux “sumSquare”, and increase effective pixel number by 1 (lines 143 ~ 145). In this sub-program, we need to calculate the following values, and save the result to the array “result”: maximum light intensity on a single pixel, i.e. the maximum light flux on a single pixel divided by the area of each pixel (line 151); total light flux (line 152); ripple or non-uniformity (line 150), defined as the root mean square of the difference of light flux on each pixel and the maximum light flux “maxFlux”, using the relation of $\sum_i (pFlux_i - maxFlux)^2 = \sum_i (pFlux_i)^2 - 2 * \sum_i (pFlux_i) * maxFlux + \sum_i (maxFlux)^2 = sumSquare - 2 * sumFlux * maxFlux + totalCount * sumSquare$.

```
157
158 SUB reportResult
159
160 PRINT
161 PRINT "length side max-Flux total-Flux non-uniformity"
162 PRINT "-----"
163 FOR m = 1, lengthNum, 1
164     FOR n = 3, sideNum, 1
165         length = 10*POWR(4, m-1)
166         FORMAT 3.0
167         PRINT " ", length, " ", n, " ",
168         FORMAT 11.8
169         PRINT result(m, n, 1), " ", result(m, n, 2), " ", result(m, n, 3)
170     NEXT n
171     PRINT "-----"
172 NEXT m
173
174 RETURN reportResult
175
176
```

Lines 158 ~ 174 are sub-program "reportResult". Its function is to output the result stored in array "result" to the screen.

The final result of the whole program is as shown in figure 4.2-2:

```

1: Text Viewer
Update Settings Print Window
Executing D:\My Macros\CH4\EX40201.ZPL.

length side max-Flux total-Flux non-uniformity
=====
10 3 0.94890384 0.83360122 0.47665470
10 4 0.98323281 0.83754776 0.43453269
10 5 1.03984896 0.83606445 0.43815834
10 6 0.98345285 0.83673602 0.42302788
10 7 1.19124229 0.83645425 0.48791640
10 8 1.01125039 0.83681343 0.42774033
-----
40 3 0.94141587 0.83331077 0.47507123
40 4 0.97470743 0.83714738 0.43872912
40 5 0.99361068 0.83589607 0.43236967
40 6 1.00724617 0.83633069 0.43288871
40 7 1.06363062 0.83609228 0.44942962
40 8 0.98955022 0.83634466 0.42173299
-----
160 3 0.94366142 0.83319181 0.47089640
160 4 0.96346162 0.83663494 0.43271867
160 5 0.98934567 0.83544672 0.42888529
160 6 0.98361326 0.83577082 0.42446393
160 7 1.02188224 0.83569761 0.43432436
160 8 0.99222065 0.83583068 0.42282320
-----
640 3 0.92571785 0.83175683 0.46277315
640 4 0.97548388 0.83518598 0.42666194
640 5 0.98219971 0.83409849 0.42370388
640 6 0.98067980 0.83431529 0.41773588
640 7 0.99645881 0.83435704 0.42007982
640 8 1.01831913 0.83447147 0.42445987
-----

```

Fig. 4.2-2: Result of program ex40201.ZPL

Please note that when the light pipe is long, the number of total internal reflections of each ray in the pipe is large. Therefore, the setting of “Maximum Intersections Per Ray” and “Maximum Segments Per Ray” in Zemax (System → General → Non-Sequential) might need to be adjusted, otherwise many rays will be lost due to surpassing the maximum limit, and the proper result cannot be obtained.

From the result shown in figure 4.2-2, when the light pipe is long enough, its output light flux becomes stable. This is easy to understand, because the transmission of light in the pipe is through total internal reflection, so the loss is negligible.

People also investigated the relation between cross section of the light pipe and the output uniformity, and believe that if the total plan can be covered by multiple cross section shape (such as hexagon), the output light uniformity will be better. The worst uniformity comes from round cross section light pipe. This is beyond our discussion in this book. Interested readers may want to investigate it further.

If we modify the program a little bit, we can also output to the screen the shape of the light spot seen on the detector, as shown in figure 4.2-3. We will skip the details here.

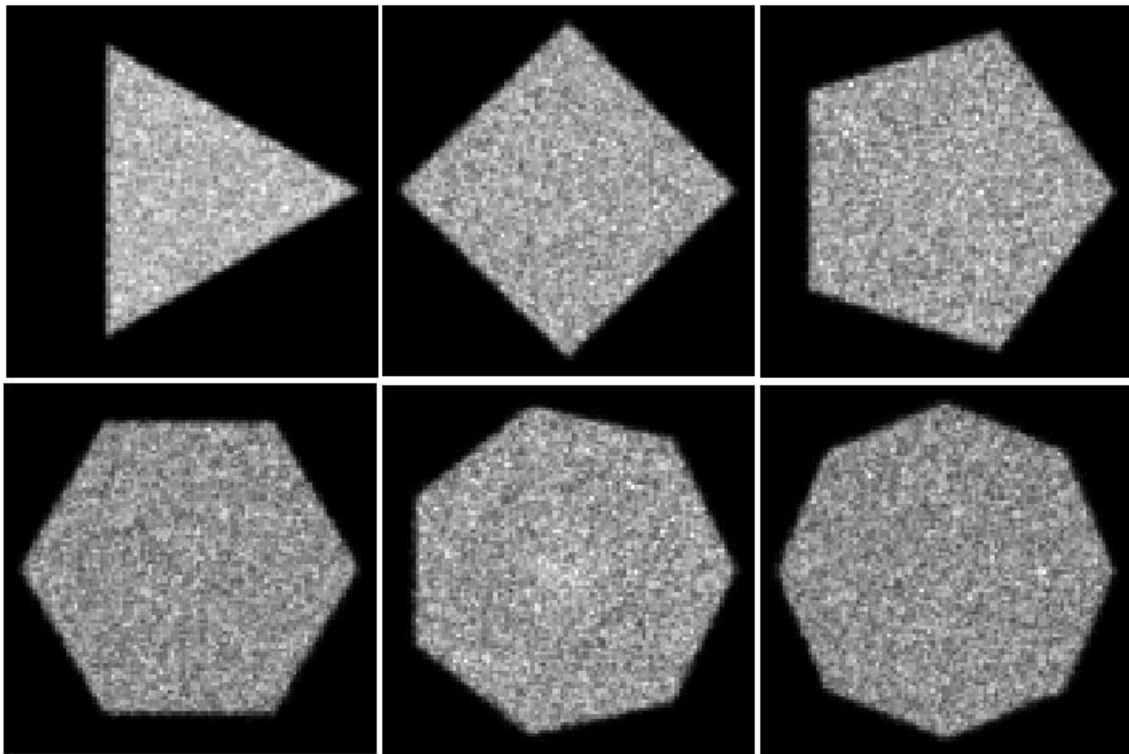


Fig. 4.2-3: The light spot seen on the detector after running program ex40201.ZPL.

Example 4.2-2: Cosine Fourth Rule.

We know in an optical system, even the exit pupil is uniformly illuminated and there is no vignetting, the illuminance between the center and the edge of the image plan is different, and it follows the cosine fourth rule, i.e. point H on the image plan with an off-axis angle Θ has only an illuminance of $\cos^4(\Theta)$ compared to that of a point A on the axis, as shown in figure 4.2-4:

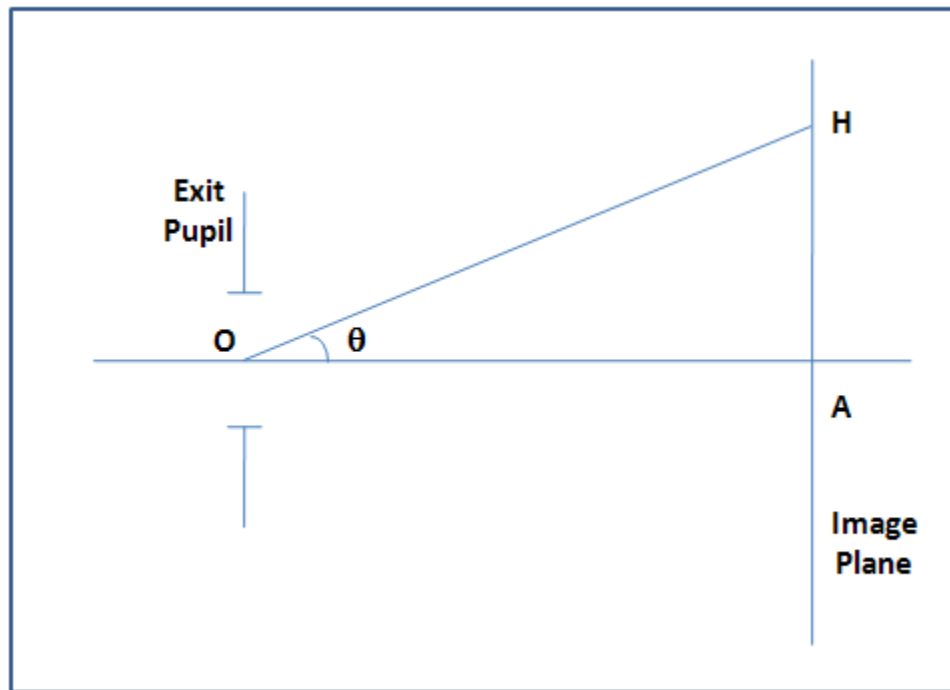


Fig. 4.2-4: optical system following cosine fourth rule

In this example, we use a round Lambertian light source to simulate the exit pupil with uniform illumination, and put a rectangular detector on the image plan, with its Y coordinate determined by angle Θ . We will investigate the variation of light flux at point H by changing angle Θ .

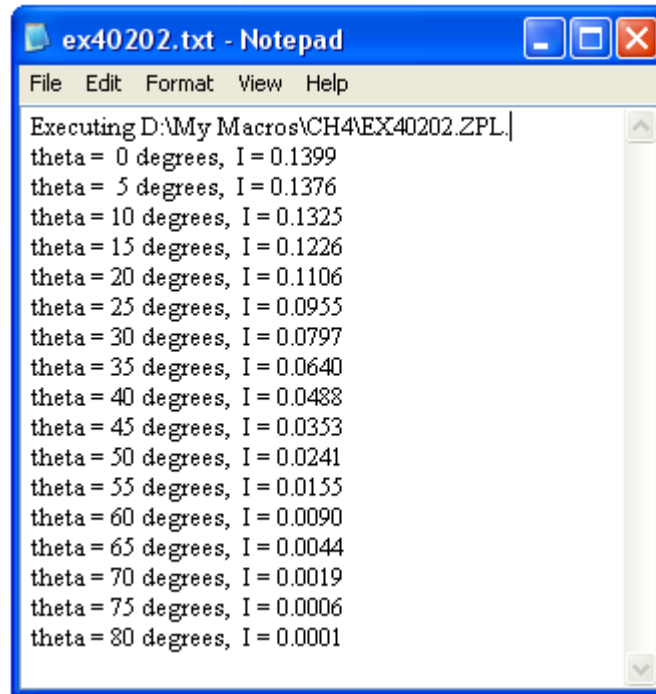
```

1 ! ex40202
2 ! this program is used to check cosine fourth rule
3
4 ! define some constant and parameters
5 PI = 3.14159
6 sourceR = 1          # source radius
7 detectorHW = 1      # detector half width
8 d0 = 15             # on axis distance
9 rayNum = 10000000   # total rays to be traced
10 saveName$ = "D:\My Macros\ch4\ex40202.txt"
11
12 ! create optical system
13 INSERTOBJECT 1, 1   # so the total object number is 2
14
15 ! define the first object
16 SETNSCPROPERTY 1, 1, 0, 0, "NSC_SRCE" # elliptical source
17 SETNSCPARAMETER 1, 1, 1, 100         # Layout Rays
18 SETNSCPARAMETER 1, 1, 2, rayNum      # Analysis Rays
19 SETNSCPARAMETER 1, 1, 3, 1          # Power
20 SETNSCPARAMETER 1, 1, 6, sourceR     # X Half Width
21 SETNSCPARAMETER 1, 1, 7, sourceR     # Y Half Width
22 SETNSCPARAMETER 1, 1, 9, 1          # Cosine Exponent, assume Lambertian
23
24 ! define the second object
25 SETNSCPROPERTY 1, 2, 0, 0, "NSC_DETE" # detector rectangular
26 SETNSCPOSITION 1, 2, 3, d0           # z position
27 SETNSCPARAMETER 1, 2, 1, detectorHW  # X Half Width
28 SETNSCPARAMETER 1, 2, 2, detectorHW  # Y Half Width
29 SETNSCPARAMETER 1, 2, 3, 1          # Number of X Pixels
30 SETNSCPARAMETER 1, 2, 4, 1          # Number of Y Pixels
31
32 ! trace the rays
33 FOR theta = 0, 80, 5                 # in degrees
34   y = d0*TANG(theta*PI/180)
35   SETNSCPOSITION 1, 2, 2, y         # y position
36   temp = NSDD(1, 0, 0, 0)           # clear the detector
37   NSTR 1, 1, 1, 1, 1, 1, 1, 0      # ray tracing
38   FORMAT 2 INT                       # same as FORMAT 2.0
39   PRINT "theta = ", theta,
40   FORMAT 6.4
41   PRINT " degrees, I = ", NSDD(1,2,0,1)
42 NEXT theta
43
44 ! save the result
45 SAVEWINDOW 1, saveName$             # assume there is no other windows open

```

Create a new non-sequential optical system, and open a new Non-Sequential Component (NSC) editor. Line 13 inserts a null object, so there are totally two null objects in the system. Lines 16 ~ 22 set the first object as a round light source, and lines 25 ~ 30 set the second object as a rectangular detector. We assume the sizes of the light source and the detector are far smaller than the distance between them. Lines 33 ~ 42 modify the value of angle theta, and thus change the Y position of the detector, trace the

rays, read the total light flux on the detector and output to the screen. At the end of the program, save the result displayed on the screen (assume text window 1) to a target file. Figure 4.2-5 shows the content of the saved file.



```
ex40202.txt - Notepad
File Edit Format View Help
Executing D:\My Macros\CH4\EX40202.ZPL
theta = 0 degrees, I = 0.1399
theta = 5 degrees, I = 0.1376
theta = 10 degrees, I = 0.1325
theta = 15 degrees, I = 0.1226
theta = 20 degrees, I = 0.1106
theta = 25 degrees, I = 0.0955
theta = 30 degrees, I = 0.0797
theta = 35 degrees, I = 0.0640
theta = 40 degrees, I = 0.0488
theta = 45 degrees, I = 0.0353
theta = 50 degrees, I = 0.0241
theta = 55 degrees, I = 0.0155
theta = 60 degrees, I = 0.0090
theta = 65 degrees, I = 0.0044
theta = 70 degrees, I = 0.0019
theta = 75 degrees, I = 0.0006
theta = 80 degrees, I = 0.0001
```

Fig. 4.2-5: The content of the saved file after running program ex40202.ZPL

The saved file is a text file so it's easy to further process it with other software. Figure 4.2-6 shows the normalized result comparing to $\cos^4(\theta)$. It's clear to see that the simulated result follows the cosine fourth rule. When the distance between the light source and the detector is short, such relation may not be true. Readers can investigate further by themselves.

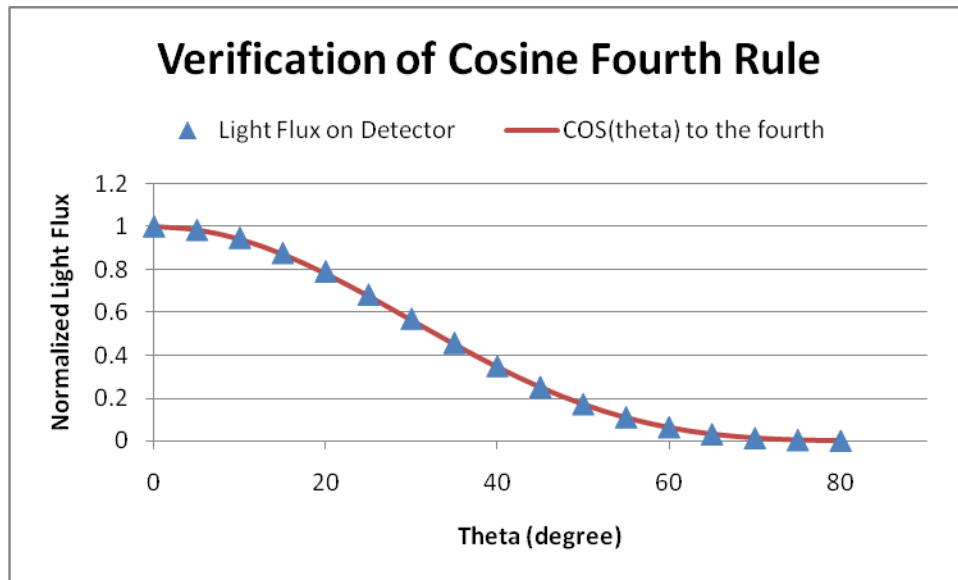


Fig. 4.2-6: Comparison of the result of program ex40202.ZPL and the cosine fourth function.

Example 4.2-3: Importance sampling.

When doing scattering analysis, we often face such a problem: the sample scatters light towards all the directions in the space, so the detector can only collect a very small part of the scattered light. In such a case, if we want to do ray-tracing analysis, we need to trace huge amount of rays, and this is usually unrealistic. In order to solve this problem, Zemax provided an importance sampling method to increase the number of rays scattered towards the target without impacting the actual light energy distribution in the space. This greatly increased the efficiency of the analysis, and is often used in optical system design. We will discuss in detail the programming of importance sampling in this example.

As shown in figure 4.2-7, assume a collimated light beam is sent from the light source onto a Lambertian sample, and the sample scatters incident light towards different directions in the space. We put 8 different detectors on the same plan in the space to measure the intensity of the scattered light. The distance between the sample and each detector is the same. We want to investigate how the light intensity varies on each detector when the sample tilts.

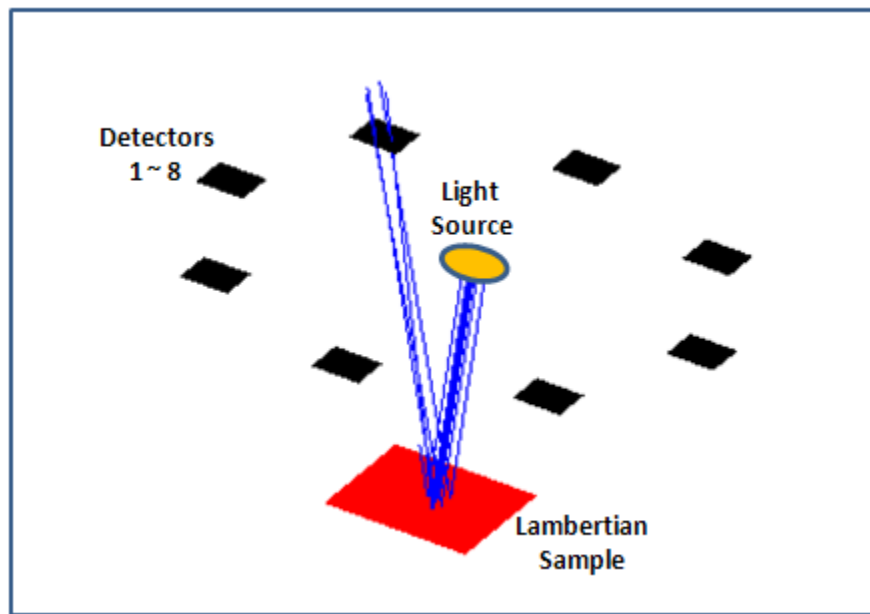


Fig. 4.2-7: Optical system in program ex40203.ZPL.

```

1 ! ex40203
2 ! This program shows the application of importance sampling.
3 ! Assume the mode is total Non-Sequential Mode.
4
5 ! define constant and parameters
6 PI = 3.14159265
7 saveName$ = "D:\My Macros\ch4\ex40203.txt"
8 sourceRadius = 1           # the radius of a circular source
9 sampleHW = 3               # the half width of the sample
10 detectorHW = 1            # the half width of the detector
11 sourceY = 10              # the Y position of source
12 detectorY = 10            # the Y position of detector
13 rayNum = 1000000         # number of rays to be traced
14
15 ! define result as a 2-dimension array
16 detectorNum = 8           # assume 8 total detectors
17 tiltAngleNum = 10        # assume rotate 10 different angles, 1~10 degrees
18 DECLARE result, DOUBLE, 2, detectorNum, tiltAngleNum
19
20 ! clean and prepare the NSC editor
21 GOSUB prepareEditor
22
23 ! define the first object as Source Ellipse
24 GOSUB defineSource
25
26 ! define the second object as Rectangle sample
27 GOSUB defineSample
28
29 ! define the third to the tenth object as Detector Rectangle
30 GOSUB defineDetectors
31
32 ! analyzing
33 FOR theta = 0, tiltAngleNum-1, 1
34   GOSUB analyze
35 NEXT
36
37 ! print and output result
38 GOSUB reportResult
39
40 END
41
42

```

Lines 1 ~ 40 are the main program. Among them, lines 6 ~ 13 defines a two-dimensional array to save the result; line 21 calls sub-program “prepareEditor” to clear NSC editor, and adds total 10 null objects; line 24 calls sub-program “defineSource” to set the first object as the light source; line 27 calls sub-program “defineSample” to set the second object as the Lambertian sample; line 30 calls sub-program “defineDetectors” to set the 3rd to 10th object as detectors; lines 33 ~ 35 modify the tilt angle of the sample, and call sub-program “analyze” to set importance sampling, do ray-tracing, and analyze the result; at last, line 38 calls sub-program “reportResult” to output result.

```

42
43 SUB prepareEditor
44
45 totalObjNum = NOBJ(1)
46 FOR i, totalObjNum, 1, -1
47   DELETEOBJECT 1, i      # delete all the objects
48 NEXT                    # there is still a null object in the editor at last
49
50 ! add 9 more objects in the editor, so the total is 10
51 FOR i = 1, 9, 1
52   INSERTOBJECT 1, 1
53 NEXT
54
55 RETURN prepareEditor
56
57

```

Lines 43 ~ 55 are sub-program “prepareEditor”. It’s used to clear NSC editor and add total 10 null objects.

```

57
58 SUB defineSource
59
60 ! define the source type
61 SETNSCPROPERTY 1, 1, 0, 0, "NSC_SRCE"
62
63 ! define the position
64 SETNSCPOSITION 1, 1, 2, 10      # Y position
65 SETNSCPOSITION 1, 1, 4, 90     # rotate about X by 90 degrees
66
67 ! define the source parameters
68 SETNSCPARAMETER 1, 1, 1, 100    # Layout Rays
69 SETNSCPARAMETER 1, 1, 2, rayNum # Analysis Rays
70 SETNSCPARAMETER 1, 1, 3, 1     # Power
71 SETNSCPARAMETER 1, 1, 6, sourceRadius # X Half Width
72 SETNSCPARAMETER 1, 1, 7, sourceRadius # Y Half Width
73 SETNSCPARAMETER 1, 1, 8, 0     # Source distance, assume collimated
74
75 RETURN defineSource
76
77

```

Lines 58 ~ 75 are sub-program “defineSource”. It’s used to set the first object as a round light source. Particularly, line 73 sets the light beam from the source as collimated light.


```
76
77
78 SUB defineSample
79
80 ! define the sample type
81 SETNSCPROPERTY 1, 2, 0, 0, "NSC_RECT"      # rectangle
82 SETNSCPROPERTY 1, 2, 4, 0, "MIRROR"      # reflective
83 SETNSCPROPERTY 1, 2, 7, 0, 1            # Lambertian
84 SETNSCPROPERTY 1, 2, 8, 0, 1            # Scatter fraction
85 SETNSCPROPERTY 1, 2, 9, 0, 1            # number of scatter rays
86
87 ! define the position
88 SETNSCPOSITION 1, 2, 2, 0                # Y position
89 SETNSCPOSITION 1, 2, 4, 90              # rotate about X by 90 degrees
90
91 ! define the sample size
92 SETNSCPARAMETER 1, 2, 1, sampleHW       # X Half Width
93 SETNSCPARAMETER 1, 2, 2, sampleHW       # Y Half Width
94
95 RETURN defineSample
96
97
```

Lines 78 ~ 95 are sub-program “defineSample”. It’s used to set the 2nd object as a Lambertian scattering sample, and set the scatter fraction as 1 (line 84), the total ray number as 1 (line 85).

```
97
98 SUB defineDetectors
99
100 FOR i = 3, 10, 1
101   ! define the detector type as detector rectangle
102   SETNSCPROPERTY 1, i, 0, 0, "NSC_DETE"
103
104   ! define the position of the detector
105   theta = (i-3)*45*PI/180           # in radians
106   x = detectorY*COSI(theta)         # calculate X coordinate
107   z = detectorY*SINE(theta)         # calculate Z coordinate
108   SETNSCPOSITION 1, i, 1, x         # X position
109   SETNSCPOSITION 1, i, 2, detectorY # Y position
110   SETNSCPOSITION 1, i, 3, z         # Z position
111   SETNSCPOSITION 1, i, 4, 90        # rotate about X by 90 degrees
112
113   ! define the detector parameters
114   SETNSCPARAMETER 1, i, 1, detectorHW # X Half Width
115   SETNSCPARAMETER 1, i, 2, detectorHW # Y Half Width
116   SETNSCPARAMETER 1, i, 3, 1         # Number of X Pixels
117   SETNSCPARAMETER 1, i, 4, 1         # Number of Y Pixels
118 NEXT
119
120 RETURN defineDetectors
121
122
```

Lines 98 ~ 120 are sub-program “defineDetectors”. It’s used to set 3rd ~ 10th object as detectors. Since we only need to read the total light intensity on the detector, we set the pixel number of each detector as 1 (lines 116 and 117).

```
122
123 SUB analyze
124
125 ! tilt sample
126 SETNSCPOSITION 1, 2, 4, 90+theta # rotate about X by another theta degrees
127
128 FOR i = 3, 10, 1 # going through different detectors
129 ! set the importance sampling towards the detector
130 SETNSCPROPERTY 1, 2, 151, 0, 1 # sample at importance sampling mode
131 FORMAT 1 INT
132 targetObject$ = $STR(INTE(i))
133 FORMAT 4.2
134 targetSize$ = $STR(detectorHW*2)
135 targetData$ = "1 " + targetObject$ + " " + targetSize$ + " 0.6"
136 SETNSCPROPERTY 1, 2, 152, 0, targetData$ # importance sampling setting
137
138 ! ray tracing
139 temp = NSDD(1, 0, 0, 0) # clear the detector
140 NSTR 1, 1, 1, 1, 1, 1, 0 # ray tracing
141 result(i-2, theta+1) = NSDD(1, i, 0, 0) # record detector reading
142 NEXT
143
144 ! recover sample
145 SETNSCPOSITION 1, 2, 4, 90 # recover original rotation
146
147 RETURN analyze
148
149
```

Lines 123 ~ 147 are sub-program “analyze”. It’s used to define the tilt angle of the sample (line 126), set importance sampling, and do ray-tracing and analysis. Among them, line 130 sets the scattering mode as importance sampling, lines 131 ~ 135 generate the target data string based on different detector, line 136 sets importance sampling target data, line 139 clears the detector, line 140 does the ray-tracing, line 141 reads the values on the detector to the “result” array, and line 145 restores the original sample rotation angle.

```

149
150 SUB reportResult
151
152 PRINT
153 PRINT "theta      D1      D2      D3      D4      D5      D6      D7      D8"
154 PRINT "=====  =====  =====  =====  =====  =====  =====  =====  ====="
155 FOR theta = 0, tiltAngleNum-1, 1
156   FORMAT 2 INT
157   PRINT " ", theta, " ",
158   FORMAT 5.4
159   FOR i = 1, detectorNum, 1
160     PRINT result(i, theta+1)/result(i, 1), " ",
161     IF i == detectorNum THEN PRINT " "
162   NEXT i
163 NEXT theta
164
165 ! save the result
166 SAVEWINDOW 1, saveName$           # assume there is no other windows open
167
168 RETURN reportResult

```

Lines 150 ~ 168 are sub-program “reportResult”. It’s used to output result to the screen, and save the content of the Text Viewer window to the target file. Please note that there is a comma at the end of line 157 and also line 160, meaning no line change after printing, and line 161 evaluates if all the detector values have been printed, then print a space and change to a new line. Further, since we only want to know the relative readings on the detectors, the result printed by line 160 is the ratio between the current value on the detector and the value on the detector with 0 tilt angle.

```

Update Settings Print Window
Executing D:\My Macros\CH4\EX40203.ZPL.

theta      D1      D2      D3      D4      D5      D6      D7      D8
=====  =====  =====  =====  =====  =====  =====  =====  =====
  0      1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000
  1      0.9998  1.0120  1.0170  1.0120  0.9999  0.9876  0.9826  0.9877
  2      0.9993  1.0237  1.0337  1.0236  0.9994  0.9750  0.9650  0.9751
  3      0.9985  1.0350  1.0501  1.0349  0.9987  0.9621  0.9471  0.9622
  4      0.9974  1.0461  1.0661  1.0460  0.9976  0.9490  0.9289  0.9490
  5      0.9961  1.0568  1.0819  1.0566  0.9962  0.9355  0.9104  0.9355
  6      0.9944  1.0673  1.0973  1.0669  0.9945  0.9218  0.8916  0.9217
  7      0.9923  1.0774  1.1124  1.0770  0.9925  0.9077  0.8726  0.9077
  8      0.9900  1.0871  1.1270  1.0867  0.9902  0.8934  0.8532  0.8934
  9      0.9874  1.0965  1.1413  1.0961  0.9876  0.8788  0.8336  0.8788

```

Fig. 4.2-8: Result of program ex40203.ZPL

The result saved in the target file can be processed by other software such as Excel, and we can get the final result as shown in figure 4.2-9. It can be seen that when the sample is tilted, the value on each detector will change accordingly, and the result follows a cosine function.

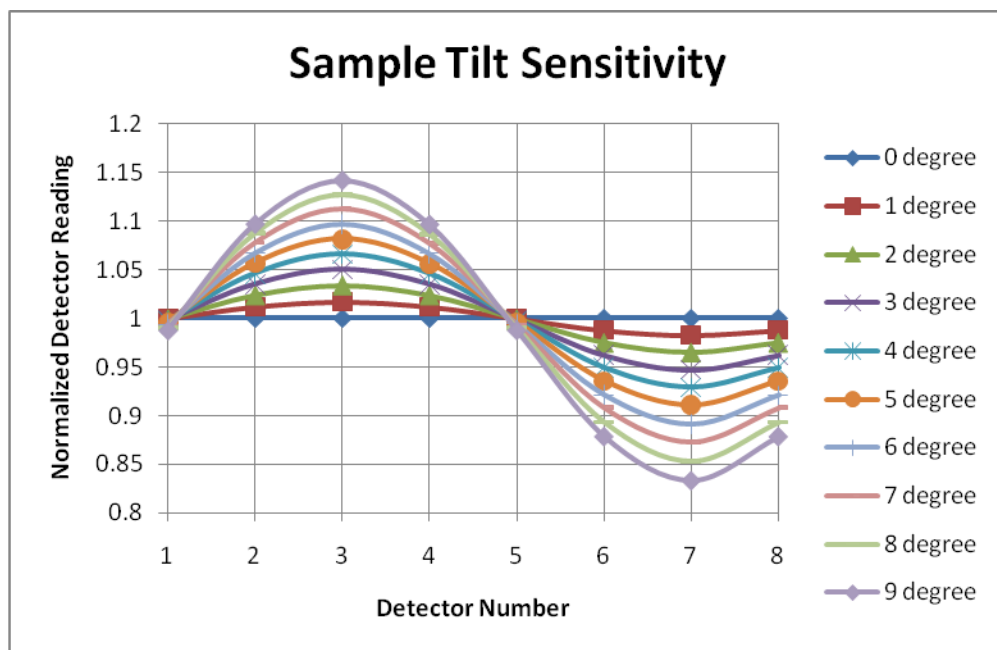


Fig. 4.2-9: Relation between detector reading and sample tilt angle from program ex40203.ZPL

Example 4.2-4: Interference fringes.

The rectangular detector provided in ZEMAX non-sequential mode is very useful. It not only can detect light flux, but also can be used to do interference analysis. In this example, we will discuss the simplest interference system, and observe interference fringes using rectangular detector.

As shown in figure 4.2-10, assume a collimated laser beam was sent onto a glass plate, so the reflected light from top and bottom surfaces will interfere with each other. If the bottom surface of the glass plate is a standard plan, and the top surface has a small curvature, we can then see interference fringes on the detector. Here we assume the size of the detector is smaller than the cross section of the light beam, so the impact from the edge of the beam can be neglected.

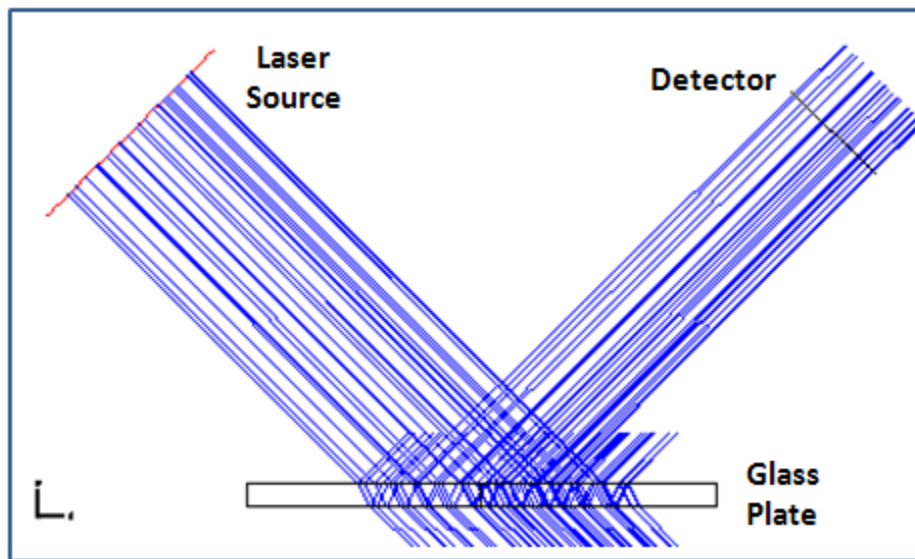


Fig. 4.2-10: Interference between light reflected from the top and the bottom surface of the glass plate.

Program ex40203.ZPL is used to change the curvature of the top surface of the glass plate, observe the change of the interference fringes accordingly, and output the result to a series of files.

Before running the program, we need to do some preparation. First, create a new lens file under non-sequential mode. Set the default null object in the NSC editor as type “Detector Rect”, and open a detector viewer, change its show data type to “Coherent Irradiance”, as shown in figure 4.2-11:

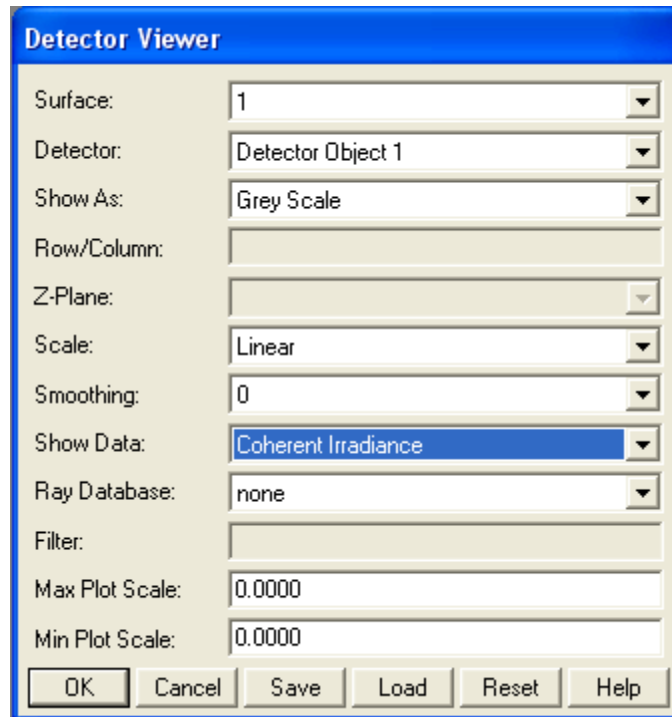


Fig. 4.2-11: set the Show Data type of the Detector Viewer as “Coherent Irradiance”

After setting the detector viewer, the following program can be executed. Lines 1 ~ 34 are the main program, wherein line 6 ~ 17 define some parameters needed in the program, line 20 calls sub-program “prepareEditor” to insert two more null objects, line 23 calls sub-program “defineSource” to set the first object as the light source, line 26 calls sub-program “defineGlassPlate” to set the second object as the glass plate, line 29 calls sub-program “defineDetector” to set the third object as the rectangular detector, line 32 calls sub-program “analyze” to do the ray-tracing and output the interference fringes on the detector to a series of files.

```

1 ! ex40204
2 ! This program shows the observation of interference fringes.
3 ! Some pre-setting of Detector Viewer is needed.
4
5 ! define constant and parameters
6 sourceRadius = 1           # the radius of a circular source
7 detectorHW = 0.5          # the half width of the detector
8 sourceY = 3               # the Y position of source
9 sourceZ = -3              # the Z position of source
10 detectorY = 3             # the Y position of detector
11 detectorZ = 3             # the Z position of detector
12 plateThickness = 0.2     # lens unit, assume mm
13 r0 = 100                 # initial radius of top surface of glass plate
14 rayNum = 1000000        # number of rays to be traced
15
16 prefix$ = "D:\My Macros\ch4\ex40204 R"
17 ext$ = ".WHF"
18
19 ! prepare the NSC editor
20 GOSUB prepareEditor
21
22 ! define the first object as Source Ellipse
23 GOSUB defineSource
24
25 ! define the second object as glass plate
26 GOSUB defineGlassPlate
27
28 ! define the third object as detector
29 GOSUB defineDetector
30
31 ! analyze
32 GOSUB analyze
33
34 END
35
36

```

The following are the sub-programs.

```

36
37 SUB prepareEditor
38
39 ! add 2 more objects in the editor, so the total is 3
40 FOR i = 1, 2, 1
41     INSERTOBJECT 1, 1
42 NEXT
43
44 RETURN prepareEditor
45
46

```


Lines 37 ~ 44 are the sub-program “prepareEditor”. It’s used to insert two null objects.

```

46
47 SUB defineSource
48
49 ! define the source type
50 SETNSCPROPERTY 1, 1, 0, 0, "NSC_SRCE"
51
52 ! define the position
53 SETNSCPOSITION 1, 1, 2, sourceY      # Y position
54 SETNSCPOSITION 1, 1, 3, sourceZ      # Z position
55 SETNSCPOSITION 1, 1, 4, 45          # rotate about X by 45 degrees
56
57 ! define the source parameters
58 SETNSCPARAMETER 1, 1, 1, 100         # Layout Rays
59 SETNSCPARAMETER 1, 1, 2, rayNum      # Analysis Rays
60 SETNSCPARAMETER 1, 1, 3, 1          # Power
61 SETNSCPARAMETER 1, 1, 6, sourceRadius # X Half Width
62 SETNSCPARAMETER 1, 1, 7, sourceRadius # Y Half Width
63 SETNSCPARAMETER 1, 1, 8, 0          # Source Distance, assume collimated
64
65 RETURN defineSource
66
67

```

Lines 47 ~ 65 are the sub-program “defineSource”. It’s used to set the first object as a collimated light source.

```

67
68 SUB defineGlassPlate
69
70 ! define the object type and material
71 SETNSCPROPERTY 1, 2, 0, 0, "NSC_SLEN"
72 SETNSCPROPERTY 1, 2, 4, 0, "BK7"
73
74 ! define the position
75 SETNSCPOSITION 1, 2, 4, 90          # rotate about X by 90 degrees
76
77 ! define the glass plate parameters
78 SETNSCPARAMETER 1, 2, 1, 0          # Radius 1
79 SETNSCPARAMETER 1, 2, 3, 2          # Clear 1
80 SETNSCPARAMETER 1, 2, 4, 2          # Edge 1
81 SETNSCPARAMETER 1, 2, 5, plateThickness # Thickness
82 SETNSCPARAMETER 1, 2, 6, 0          # Radius 2
83 SETNSCPARAMETER 1, 2, 8, 2          # Clear 2
84 SETNSCPARAMETER 1, 2, 9, 2          # Edge 2
85
86 RETURN defineGlassPlate
87
88

```

Lines 68 ~ 86 are the sub-program “defineGlassPlate”. It’s used to set the second object as a glass plate. The type of the glass plate is set as standard lens, so the curvature of its top surface can be easily modified.

```

88
89 SUB defineDetector
90
91 ! define the detector type as detector rectangle
92 SETNSCPROPERTY 1, 3, 0, 0, "NSC_DETE"
93
94 ! define the position of the detector
95 SETNSCPOSITION 1, 3, 2, detectorY # Y position
96 SETNSCPOSITION 1, 3, 3, detectorZ # Z position
97 SETNSCPOSITION 1, 3, 4, -45 # rotate about X by -45 degrees
98
99 ! define the detector parameters
100 SETNSCPARAMETER 1, 3, 1, detectorHW # X Half Width
101 SETNSCPARAMETER 1, 3, 2, detectorHW # Y Half Width
102 SETNSCPARAMETER 1, 3, 3, 100 # Number of X Pixels
103 SETNSCPARAMETER 1, 3, 4, 100 # Number of Y Pixels
104
105 RETURN defineDetector
106
107

```

Lines 89 ~ 105 are the sub-program “defineDetector”. It’s used to set the third object as a rectangular detector.

```

107
108 SUB analyze
109
110 FOR i = 1, 10, 1
111   radius1 = r0 + POWR(2, i-1)
112   SETNSCPARAMETER 1, 2, 1, radius1 # change Radius 1
113   temp = NSDD(1, 0, 0, 0) # clear the detector
114   NSTR 1, 1, 1, 1, 1, 1, 1, 0 # ray tracing
115   FORMAT 1 INT
116   r$ = $STR(radius1)
117   fileName$ = prefix$ + r$ + ext$
118   UPDATE 1 # assume serial number of the graphic window is 1
119   EXPORTBMP 1, fileName$
120 NEXT
121
122 RETURN analyze
123

```

Lines 108 ~ 122 are the sub-program “analyze”. It’s used to modify the curvature of the top surface of the glass plate, do ray-tracing, generate file names according to the curvature of the top surface of the glass plate, and output the content of the detector viewer to the target files.

After running the program, we can see the following files are added in the target folder:

ex40204 R101.BMP

ex40204 R102.BMP

ex40204 R104.BMP

ex40204 R108.BMP

ex40204 R116.BMP

ex40204 R132.BMP

ex40204 R164.BMP

ex40204 R228.BMP

ex40204 R356.BMP

ex40204 R612.BMP

Among them, the content of the 1st, 5th, and 9th files are shown below in figure 4.2-12:

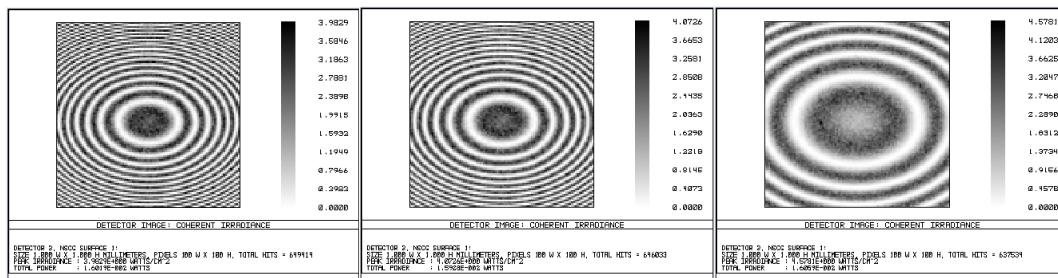


Fig. 4.2-12: The content of some files generated from program ex40204.ZPL.

Example 4.2-5: Impact of entrance size and detector size to the efficiency of the integrating sphere.

Integrating sphere is a common device used in optical measurement. One type of structure has two perpendicular ports on the sphere, one for light source, and the other for detector, as shown in figure 4.2-13. As we know, both the size of the light entrance port and the size of the detector have impact on the integrating efficiency, because light can get lost from those two places. In this program, we will change the size of the light source and the detector, and investigate its impact on the integrating efficiency.

First, we will use the sphere object in Zemax to simulate an integrating sphere. Assume the radius is 1 (we are only interested in the relative value, not the absolute value). We then put a round light source at location $Z = 1$, and put a round detector at location $Y = 1$. The actual locations require some simple calculation to assure they are tangent to the sphere. Further, to assure there is no overlapping during ray-tracing, we move the light source and the detector slightly towards the sphere center. We set the detector material as "ABSORB" to simulate the loss of light when entering it. Since we cannot set the light source as absorbing material, we put another round disk with the same size of the light source between the light source and the sphere vertex, close to the light source. This is to simulate the light loss when it comes back to the entrance port.

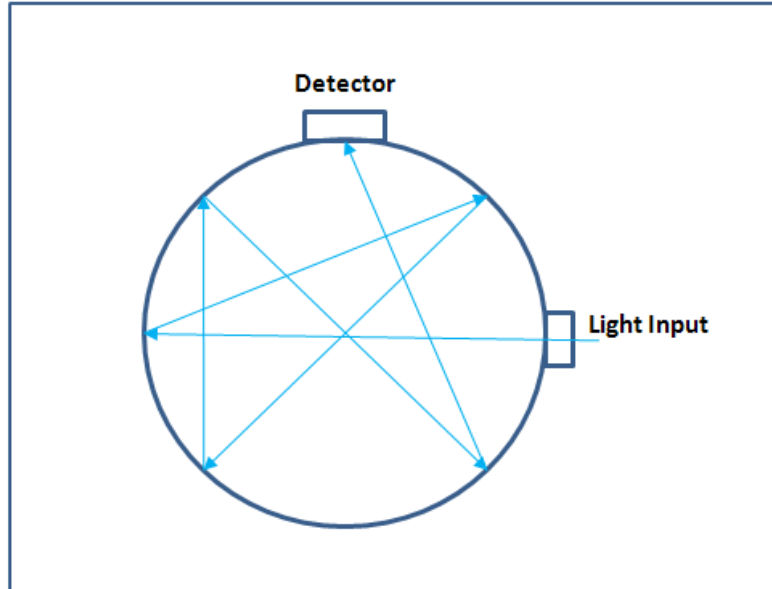


Fig. 4.2-13: Diagram of a common integrating sphere.

Create a new lens file in Zemax, and then we can run our program shown below.

```

1 ! ex40205
2 ! This program shows how to simulate an integrating sphere
3
4 ! define constant and parameters
5 sphereR = 1           # the radius of the integrating sphere
6 gap = 0.001          # a gap from the sphere
7 startSouR = 0        # start source radius, will skip the first one
8 stopSouR = 0.8       # stop source radius
9 stepSouNum = 20      # steps between start and stop source radius
10 startDetR = 0        # start detector radius, will skip the first one
11 stopDetR = 0.5       # stop detector radius
12 stepDetNum = 10     # steps between start and stop detector radius
13 rayNum = 100000     # number of rays to be traced
14
15 saveName$ = "D:\My Macros\ch4\ex40205.txt"
16
17 ! define result as a 2-dimension array
18 DECLARE result, DOUBLE, 2, stepSouNum, stepDetNum
19
20 ! prepare the NSC editor and define some system parameters
21 GOSUB prepareEditor
22
23 ! define the first object as Source Ellipse
24 sourceR = 0.01       # this number is just an innitial value, will change
25 GOSUB defineSource
26
27 ! define the second object as a sphere shell
28 GOSUB defineSphere
29
30 ! define the third object as detector
31 detectorR = 0.01     # this number is just an innitial value, will change
32 GOSUB defineDetector
33
34 ! define the fourth object as the output port
35 GOSUB defineOutputPort
36
37 ! analyze
38 GOSUB analyze
39
40 ! export result to a file
41 GOSUB exportToFile
42
43 END
44
45

```

Lines 1 ~ 43 are the main program, wherein lines 5 ~ 15 set the parameters needed in the program, line 18 is used to define a two dimensional array to store the result, and the rest is similar to the examples

we discussed before, i.e. calling different sub-programs to set up system, do the ray-tracing and analysis, and output the result. In this example, we will use nested sub-programs, i.e. calling other sub-programs from a sub-program.

```

45
46 SUB prepareEditor
47
48 ! add 2 more objects in the editor, so the total is 3
49 FOR i = 1, 3, 1
50     INSERTOBJECT 1, 1
51 NEXT
52
53 SYSP 50, 4000           # maximum intersection per ray
54 SYSP 51, 50000        # maximum segments per ray
55 SYSP 53, 1E-6         # minimum relative energy
56
57 RETURN prepareEditor
58
59

```

Lines 46 ~ 57 are sub-program “prepareEditor”. It’s used to create total 3 objects in the NSC editor, and set the maximum intersection per ray, maximum segments per ray and minimum relative energy.

```

59
60 SUB defineSource
61
62 ! define the source type
63 SETNSCPROPERTY 1, 1, 0, 0, "NSC_SRCE"
64 SETNSCPROPERTY 1, 1, 4, 0, "ABSORB"
65
66 ! define the position
67 sourceZ = SQRT(sphereR*sphereR-sourceR*sourceR)
68 SETNSCPOSITION 1, 1, 3, sourceZ-gap      # Z position
69 SETNSCPOSITION 1, 1, 4, 180             # rotate about X by 180 degrees
70
71 ! define the source parameters
72 SETNSCPARAMETER 1, 1, 1, 100            # Layout Rays
73 SETNSCPARAMETER 1, 1, 2, rayNum        # Analysis Rays
74 SETNSCPARAMETER 1, 1, 3, 1            # Power
75 SETNSCPARAMETER 1, 1, 6, sourceR       # X Half Width
76 SETNSCPARAMETER 1, 1, 7, sourceR       # Y Half Width
77 SETNSCPARAMETER 1, 1, 8, 0            # Source Distance, assume collimated
78
79 RETURN defineSource
80
81

```

Lines 60~ 79 are sub-program “defineSource”. It’s used to set up the light source. Among them, line 67 calculates the Z position of the light source, line 68 sets the light source position with a small shift. In this sub-program, the radius of the light source “sourceR” is not fixed, and it needs to be determined before calling this sub-program. This allows us the easily analyze different size light sources.

```
81
82 SUB defineSphere
83
84 ! define the object type as Lambertian reflective
85 SETNSCPROPERTY 1, 2, 0, 0, "NSC_SPHE" # sphere
86 SETNSCPROPERTY 1, 2, 4, 0, "MIRROR" # reflective
87 SETNSCPROPERTY 1, 2, 7, 0, 1 # Lambertian
88 SETNSCPROPERTY 1, 2, 8, 0, 1 # Scatter fraction
89 SETNSCPROPERTY 1, 2, 9, 0, 1 # number of scatter rays
90
91 ! define the sphere parameters
92 SETNSCPARAMETER 1, 2, 1, sphereR # radius
93 SETNSCPARAMETER 1, 2, 2, 0 # a shell, not a volume
94
95 RETURN defineSphere
96
97
```

Lines 82~ 95 are sub-program “defineSphere”. It’s used to set up the integrating sphere. We set the sphere object type as a shell (line 93), and its optical property as Lambertian scattering (lines 86 ~ 89).

```

97
98 SUB defineDetector
99
100 ! define the detector type as detector surface
101 SETNSCPROPERTY 1, 3, 0, 0, "NSC_DETS"
102 SETNSCPROPERTY 1, 3, 4, 0, "ABSORB"
103
104 ! define the position of the detector
105 detectorY = SQRT(sphereR*sphereR-detectorR*detectorR)
106 SETNSCPOSITION 1, 3, 2, detectorY-gap # Y position
107 SETNSCPOSITION 1, 3, 4, 90 # rotate about X by 90 degrees
108
109 ! define the detector parameters
110 SETNSCPARAMETER 1, 3, 1, 0 # Radius = 0, flat
111 SETNSCPARAMETER 1, 3, 3, detectorR # max aperture
112 SETNSCPARAMETER 1, 3, 4, 0 # min aperture
113
114 RETURN defineDetector
115
116

```

Lines 98~ 114 are sub-program “defineDetector”. It’s used to set up the round detector. Just like the light source, the size of the detector is not fixed, and it needs to be determined before calling this sub-program. The position of the detector also needs to be calculated (line 105).

```

116
117 SUB defineOutputPort
118
119 ! define the output port type as ellipse
120 SETNSCPROPERTY 1, 4, 0, 0, "NSC_ELLI"
121 SETNSCPROPERTY 1, 4, 4, 0, "ABSORB"
122
123 ! define the position
124 outputPortZ = SQRT(sphereR*sphereR-sourceR*sourceR)
125 SETNSCPOSITION 1, 4, 3, outputPortZ-gap/2 # Z position
126 SETNSCPOSITION 1, 4, 4, 90 # rotate about X by 90 degrees
127
128 ! define the output port parameters
129 SETNSCPARAMETER 1, 4, 1, sourceR # X Half Width, same as source
130 SETNSCPARAMETER 1, 4, 2, sourceR # Y Half Width, same as source
131
132 RETURN defineOutputPort
133
134

```

Lines 117~ 132 are sub-program “defineOutputPort”. It’s used to put an absorbing object after the light source, and simulate the light loss from here. Similarly, its size and position need to be calculated.


```

134
135 SUB analyze
136
137 ! print header
138 GOSUB printHeader
139
140 ! ray tracing and print result
141 FOR i = 1, stepSouNum, 1
142   sourceR = startSouR+i*(stopSouR-startSouR)/stepSouNum
143   FORMAT 4.2
144   PRINT sourceR, " ",
145   FOR j = 1, stepDetNum, 1
146     detectorR = startDetR+j*(stopDetR-startDetR)/stepDetNum
147     GOSUB defineSource           # modify source setting
148     GOSUB defineDetector        # modify detector setting
149     GOSUB defineOutputPort      # modify output port setting
150     temp = NSDD(1, 0, 0, 0)     # clear the detector
151     NSTR 1, 1, 1, 1, 1, 1, 1, 0 # ray tracing
152     result(i, j) = NSDD(1, 3, 0, 0) # record detector reading
153     FORMAT 6.5
154     PRINT NSDD(1, 3, 0, 0), " ", # print result
155   NEXT j
156   PRINT
157 NEXT i
158
159 RETURN analyze
160
161

```

Lines 135~ 159 are sub-program “analyze”. It’s used to modify the size of the light source and the detector, do ray-tracing, and record the result. In this sub-program, we list result of each ray-tracing on the screen in a table, and save the result into array “result”. Line 138 calls another sub-program “printHeader” to print table header on the screen. The outer loop in lines 141 ~ 157 modifies the size of the light source, the inner loop in lines 145 ~ 155 modifies the size of the detector. Lines 142 and 146 calculate the size of the light source and the detector, respectively, lines 147, 148 and 149 call different sub-programs to set the light source, the detector, and the absorber alongside the light source, and then do the ray-tracing, record and output data. Please note that we can call the same sub-programs from either the main program or from this sub-program.

```
161
162 SUB printHeader
163
164 PRINT
165 PRINT " S ",
166 FOR i = 1, stepDetNum, 1
167   FORMAT 1 INT
168   PRINT " D", i, " ",
169 NEXT
170 PRINT
171 PRINT " ",
172 FOR i = 1, stepDetNum, 1
173   detectorR = startDetR+i*(stopDetR-startDetR)/stepDetNum
174   FORMAT 4.2
175   PRINT " ", detectorR, " ",
176 NEXT
177 PRINT
178 PRINT "=====",
179 FOR i = 1, stepDetNum, 1
180   PRINT "=====",
181 NEXT
182 PRINT " "
183
184 RETURN printHeader
185
186
```

Lines 162~ 184 are sub-program “printHeader”. It’s used to print the table header. We use the default output target screen here.

```

186
187
188 SUB exportToFile
189
190 OUTPUT saveName$
191 FORMAT 6.5
192
193 PRINT "First row is detector radius, ",
194 PRINT "first column is source and output port radius,"
195 PRINT "and the rest data are corresponding detector readings."
196 PRINT
197
198 PRINT 0, " ",
199 FOR i = 1, stepDetNum, 1
200     detectorR = startDetR+i*(stopDetR-startDetR)/stepDetNum
201     PRINT detectorR, " ",
202 NEXT
203 PRINT
204 FOR i = 1, stepSouNum, 1
205     sourceR = startSouR+i*(stopSouR-startSouR)/stepSouNum
206     PRINT SourceR, " ",
207     FOR j = 1, stepDetNum, 1
208         PRINT result(i, j), " ",
209     NEXT j
210     PRINT
211 NEXT i
212
213 OUTPUT SCREEN
214
215 RETURN exportToFile
216

```

Lines 188~ 215 are sub-program “exportToFile”. It’s used to output the result stored in array “result” to the target file with given format. Of course we can directly output the table printed on the screen to the target file, as we did in example 4.2-3, but here we show a different output method.

After running the program, we can see on the screen the result shown in figure 4.2-14:

1: Text Viewer

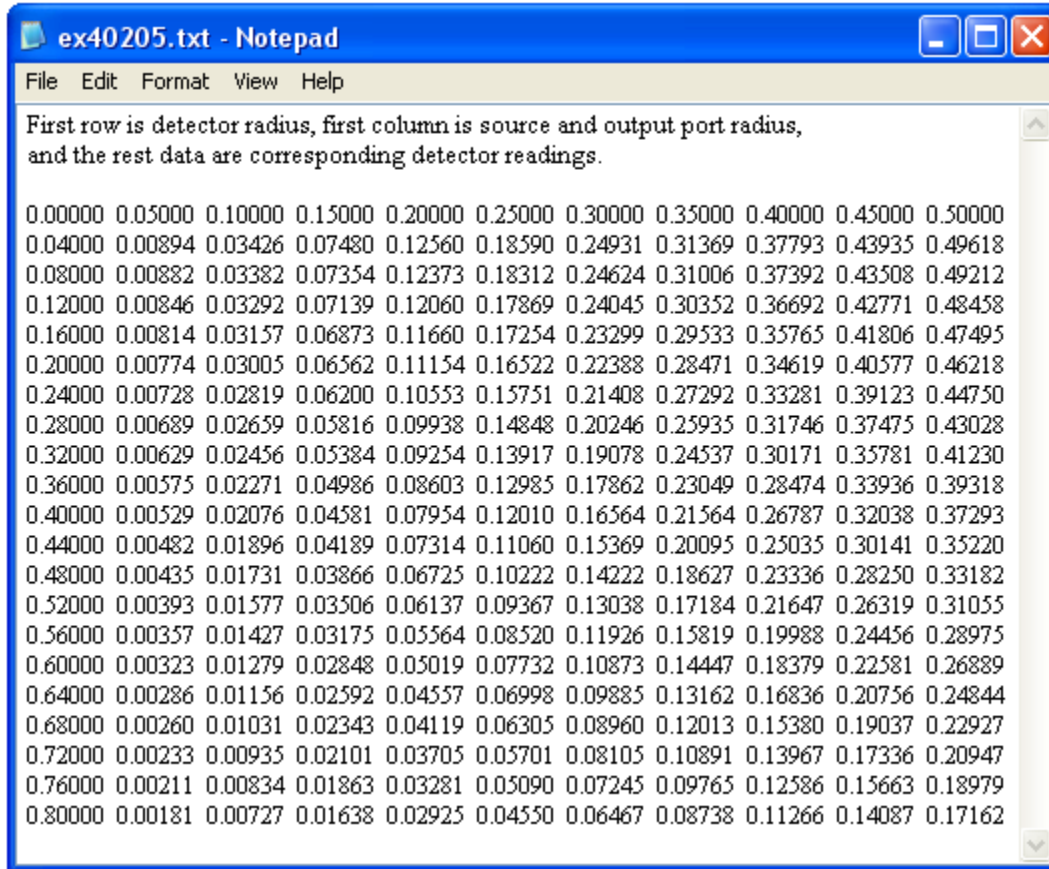
Update Settings Print Window

Executing D:\My Macros\CH4\EX40205.ZPL.

S	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50
0.04	0.00894	0.03426	0.07480	0.12560	0.18590	0.24931	0.31369	0.37793	0.43935	0.49618
0.08	0.00882	0.03382	0.07354	0.12373	0.18312	0.24624	0.31006	0.37392	0.43508	0.49212
0.12	0.00846	0.03292	0.07139	0.12060	0.17869	0.24045	0.30352	0.36692	0.42771	0.48458
0.16	0.00814	0.03157	0.06873	0.11660	0.17254	0.23299	0.29533	0.35765	0.41806	0.47495
0.20	0.00774	0.03005	0.06562	0.11154	0.16522	0.22388	0.28471	0.34619	0.40577	0.46218
0.24	0.00728	0.02819	0.06200	0.10553	0.15751	0.21408	0.27292	0.33281	0.39123	0.44750
0.28	0.00689	0.02659	0.05816	0.09938	0.14848	0.20246	0.25935	0.31746	0.37475	0.43028
0.32	0.00629	0.02456	0.05384	0.09254	0.13917	0.19078	0.24537	0.30171	0.35781	0.41230
0.36	0.00575	0.02271	0.04986	0.08603	0.12985	0.17862	0.23049	0.28474	0.33936	0.39318
0.40	0.00529	0.02076	0.04581	0.07954	0.12010	0.16564	0.21564	0.26787	0.32038	0.37293
0.44	0.00482	0.01896	0.04189	0.07314	0.11060	0.15369	0.20095	0.25035	0.30141	0.35220
0.48	0.00435	0.01731	0.03866	0.06725	0.10222	0.14222	0.18627	0.23336	0.28250	0.33182
0.52	0.00393	0.01577	0.03506	0.06137	0.09367	0.13038	0.17184	0.21647	0.26319	0.31055
0.56	0.00357	0.01427	0.03175	0.05564	0.08520	0.11926	0.15819	0.19988	0.24456	0.28975
0.60	0.00323	0.01279	0.02848	0.05019	0.07732	0.10873	0.14447	0.18379	0.22581	0.26889
0.64	0.00286	0.01156	0.02592	0.04557	0.06998	0.09885	0.13162	0.16836	0.20756	0.24844
0.68	0.00260	0.01031	0.02343	0.04119	0.06305	0.08960	0.12013	0.15380	0.19037	0.22927
0.72	0.00233	0.00935	0.02101	0.03705	0.05701	0.08105	0.10891	0.13967	0.17336	0.20947
0.76	0.00211	0.00834	0.01863	0.03281	0.05090	0.07245	0.09765	0.12586	0.15663	0.18979
0.80	0.00181	0.00727	0.01638	0.02925	0.04550	0.06467	0.08738	0.11266	0.14087	0.17162

Fig. 4.2-14: Result shown on the screen after running program ex40205.ZPL

In the mean time, in the target folder given in the program, we can see a target file with content shown in figure 4.2-15:



```

File Edit Format View Help
First row is detector radius, first column is source and output port radius,
and the rest data are corresponding detector readings.

0.00000 0.05000 0.10000 0.15000 0.20000 0.25000 0.30000 0.35000 0.40000 0.45000 0.50000
0.04000 0.00894 0.03426 0.07480 0.12560 0.18590 0.24931 0.31369 0.37793 0.43935 0.49618
0.08000 0.00882 0.03382 0.07354 0.12373 0.18312 0.24624 0.31006 0.37392 0.43508 0.49212
0.12000 0.00846 0.03292 0.07139 0.12060 0.17869 0.24045 0.30352 0.36692 0.42771 0.48458
0.16000 0.00814 0.03157 0.06873 0.11660 0.17254 0.23299 0.29533 0.35765 0.41806 0.47495
0.20000 0.00774 0.03005 0.06562 0.11154 0.16522 0.22388 0.28471 0.34619 0.40577 0.46218
0.24000 0.00728 0.02819 0.06200 0.10553 0.15751 0.21408 0.27292 0.33281 0.39123 0.44750
0.28000 0.00689 0.02659 0.05816 0.09938 0.14848 0.20246 0.25935 0.31746 0.37475 0.43028
0.32000 0.00629 0.02456 0.05384 0.09254 0.13917 0.19078 0.24537 0.30171 0.35781 0.41230
0.36000 0.00575 0.02271 0.04986 0.08603 0.12985 0.17862 0.23049 0.28474 0.33936 0.39318
0.40000 0.00529 0.02076 0.04581 0.07954 0.12010 0.16564 0.21564 0.26787 0.32038 0.37293
0.44000 0.00482 0.01896 0.04189 0.07314 0.11060 0.15369 0.20095 0.25035 0.30141 0.35220
0.48000 0.00435 0.01731 0.03866 0.06725 0.10222 0.14222 0.18627 0.23336 0.28250 0.33182
0.52000 0.00393 0.01577 0.03506 0.06137 0.09367 0.13038 0.17184 0.21647 0.26319 0.31055
0.56000 0.00357 0.01427 0.03175 0.05564 0.08520 0.11926 0.15819 0.19988 0.24456 0.28975
0.60000 0.00323 0.01279 0.02848 0.05019 0.07732 0.10873 0.14447 0.18379 0.22581 0.26889
0.64000 0.00286 0.01156 0.02592 0.04557 0.06998 0.09885 0.13162 0.16836 0.20756 0.24844
0.68000 0.00260 0.01031 0.02343 0.04119 0.06305 0.08960 0.12013 0.15380 0.19037 0.22927
0.72000 0.00233 0.00935 0.02101 0.03705 0.05701 0.08105 0.10891 0.13967 0.17336 0.20947
0.76000 0.00211 0.00834 0.01863 0.03281 0.05090 0.07245 0.09765 0.12586 0.15663 0.18979
0.80000 0.00181 0.00727 0.01638 0.02925 0.04550 0.06467 0.08738 0.11266 0.14087 0.17162

```

Fig. 4.2-15: The content of the target file saved after running program ex40205.ZPL

The result saved in the text file can be easily processed with other software. Shown in figure 4.2-16 is the relation between the integrating efficiency and the size of the light source and the detector, processed in Excel.

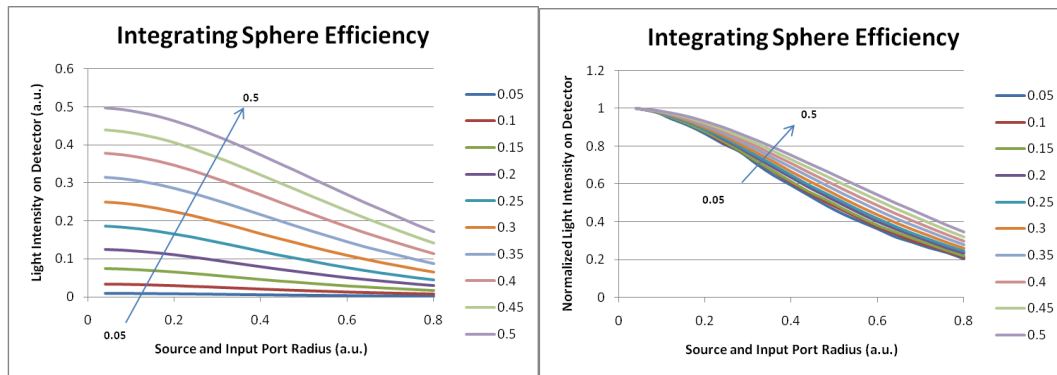


Fig. 4.2-16: Integrating efficiency obtained from the result of program ex40205.ZPL

Example 4.2-6: Three-dimensional light intensity distribution obtained with volume detector.

Detector Volume is a powerful tool Zemax provided. It is a rectangular volume with an arbitrary number of voxels. Voxels is a name derived from "volume pixels". A voxel is a 3D rectangle block that defines some portion of the total volume occupied by imported solids. The detector volume may be nested within or straddle any other object. Multiple detector volumes may also be superimposed and all will be illuminated by rays passing through the individual voxels. With the help of Detector Volume we can easily obtain the intensity distribution of incident light or absorbed light in 3-D space.

In this example, we will discuss how to obtain the 3-D intensity distribution of incident light, and display this distribution in 3-D plot. The optical system is an LED model provided by Zemax, and can be obtained from Zemax installation folder (default as "C:\Program Files\ZEMAX\Samples\Non-sequential\Sources\"), file name "led_model.ZMX". We resave the file to "ex40206.ZMX". In this optical system, there is a simple LED model, as well as a plan detector, as shown in figure 4.2-17:

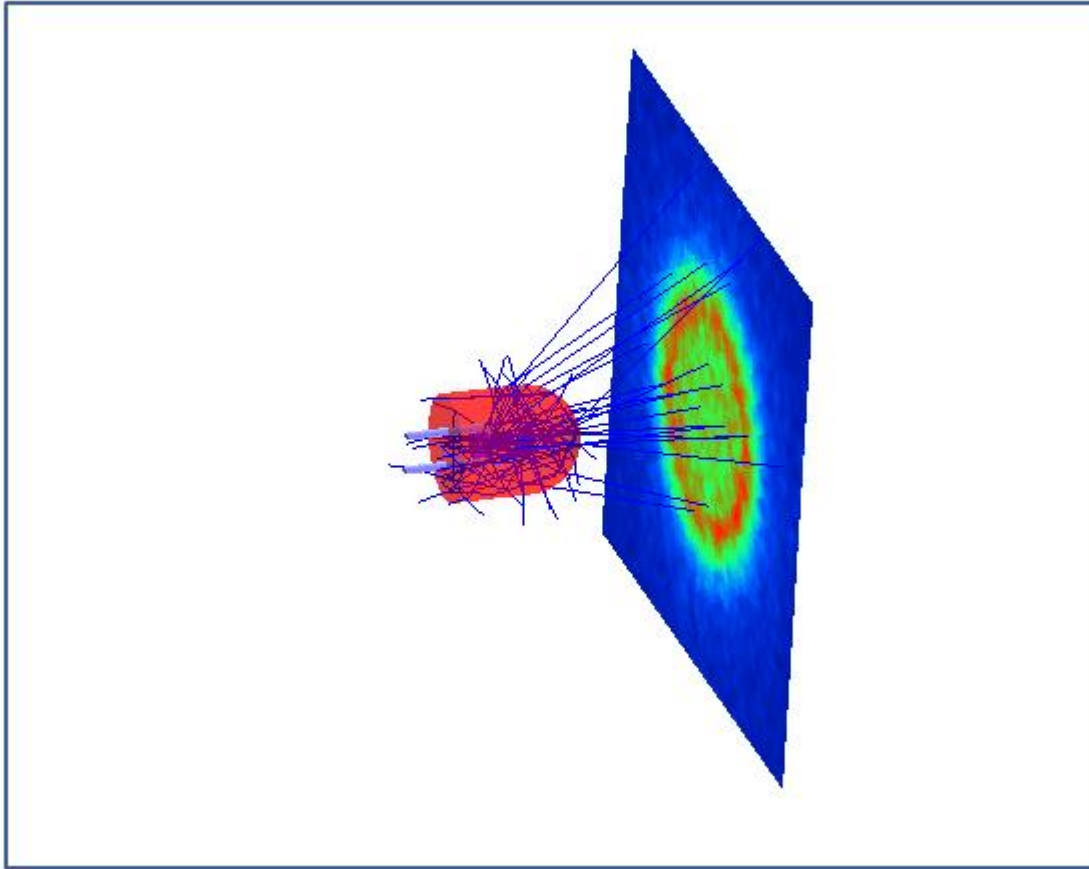


Fig. 4.2-17: Diagram of optical system ex40206.ZMX

We will slightly modify this system, delete the plan detector, and add a volume detector, as shown in figure 4.2-18. We can do this either directly in the NSC editor, or with ZPL program. In this example we will do it with ZPL program.

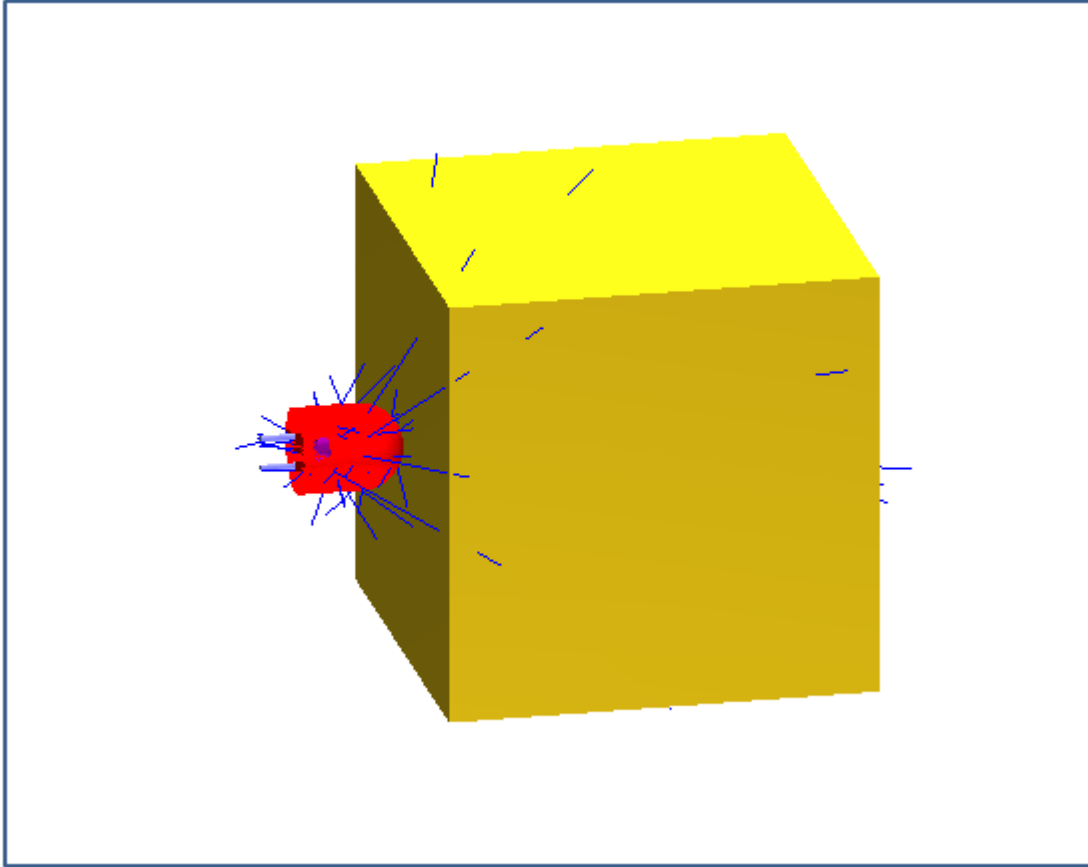


Fig. 4.2-18: Diagram of modified optical system ex40206.ZMX.


```
1 ! ex40206
2 ! shows how to use detector volume to analyze light distribution
3 ! Assume the optical system is defined in ex40206.ZMX
4
5 ! initializing
6 hwx = 10           # x half width of voxel
7 hwy = 10           # y half width of voxel
8 hwz = 10           # z half width of voxel
9 nvx = 20           # pixel number in x
10 nvy = 20          # pixel number in y
11 nvz = 20          # pixel number in z
12
13 DECLARE intensity, DOUBLE, 1, nvx*nvy*nvz
14 DECLARE voxelColor, INTEGER, 1, nvx*nvy*nvz
15 DECLARE threshold, DOUBLE, 1, 4
16 threshold(1) = 0.5
17 threshold(2) = 0.2
18 threshold(3) = 0.05
19 threshold(4) = 0.01
20
21 ! revise model
22 GOSUB changeModel
23
24 ! record light intensity
25 GOSUB recordLight
26
27 ! assign color to each voxel
28 GOSUB setColor
29
30 ! create 3D model to visualize light intensity distribution
31 GOSUB visualize
32
33 END
34
35
```

Lines 1 ~ 33 are the main program, wherein lines 6 ~ 11 define the size of the volume detector and the number of voxels, line 13 defines a one dimensional array “intensity” to store the incident light intensity, line 14 defines a one dimensional array “voxelColor” to store the color group of each voxel (determined by the relative light intensity on the voxel), line 15 ~ 19 define the light intensity threshold, line 22 calls sub-program “changeModel” to slightly modify the optical system, i.e. delete the plan detector and add the volume detector as mentioned before, line 25 calls sub-program “recordLight” to do the ray-tracing and record the incident light intensity of each voxel, line 28 calls sub-program “setColor” to normalize the light intensity of each voxel, and assign different color number according to the intensity value, and finally, line 31 calls sub-program “visualize” to create a 3-D model of colors to represent the light intensity distribution in the space, and import the 3-D objects to the current optical system.

```

35
36 SUB changeModel
37
38 objNum = NOBJ(1)           # find out total object number
39
40 FOR i = objNum, 1, -1
41   temp = NPRO(1, i, 0, 0)
42   A$ = $buffer()          # find out the object type
43   IF A$ $== "NSC_DETE" THEN DELETEOBJECT 1, i   # delete the detector
44 NEXT
45
46 INSERTOBJECT 1, 1
47 SETNSCPROPERTY 1, 1, 0, 0, "NSC_DETV"          # type as detector volume
48 SETNSCPARAMETER 1, 1, 1, hwX                  # x half width
49 SETNSCPARAMETER 1, 1, 2, hwY                  # y half width
50 SETNSCPARAMETER 1, 1, 3, hwZ                  # z half width
51 SETNSCPARAMETER 1, 1, 4, nvX                  # number of x pixels
52 SETNSCPARAMETER 1, 1, 5, nvY                  # number of y pixels
53 SETNSCPARAMETER 1, 1, 6, nvZ                  # number of z pixels
54 SETNSCPOSITION 1, 1, 3, hwZ+5                 # put volume in front of LED
55 SETNSCPROPERTY 1, 1, 142, 0, 9                # opacity as 10%
56
57 UPDATE ALL
58
59 RETURN
60
61

```

Lines 36 ~ 59 are sub-program "". It's used to modify original optical system, i.e. delete the plan detector, and add a volume detector. Lines 40 ~ 44 check each object, and delete it if its type is plan detector (NSC_DETE). Lines 46 ~ 55 insert a new object, define its type and other parameters. Line 57 updates the system.

```
61
62 SUB recordLight
63
64 temp = NSDD(1, 0, 0, 0)           # clear detector
65
66 PRINT
67 PRINT "Ray tracing ..."
68 NSTR 1, 0, 1, 1, 1, 1, 0       # trace rays
69
70 IMax = 0
71 FOR ix = 1, nvx, 1
72   FOR iy = 1, nvy, 1
73     FOR iz = 1, nvz, 1
74       n = ix+(iy-1)*nvx+(iz-1)*nvx*nvy   # voxel number
75       intensity(n) = NSDD(1,1,n,0)       # read incident flux
76       IF IMax < intensity(n) THEN IMax = intensity(n) # update IMax
77     NEXT iz
78   NEXT iy
79 NEXT ix
80
81 RETURN
82
83
```

Lines 62 ~ 81 are sub-program "recordLight". It's used to do the ray-tracing and record the intensity of the light incident to each voxel of the volume detector. Particularly, line 64 clears the detector, line 68 does the ray-tracing, lines 71 ~ 79 use a loop to record the light intensity of each pixel, find the maximum value, and store the value in variable IMax.

```
83
84 SUB setColor
85
86 FOR n = 1, nvx*nvy*nvz, 1
87   intensity(n) = intensity(n)/IMax   # normalize intensity of each voxel
88   IF intensity(n) > threshold(1)
89     voxelColor(n) = 1               # red color
90   ELSE
91     IF intensity(n) > threshold(2)
92       voxelColor(n) = 2             # yellow color
93     ELSE
94       IF intensity(n) > threshold(3)
95         voxelColor(n) = 3           # green color
96       ELSE
97         IF intensity(n) > threshold(4)
98           voxelColor(n) = 4         # blue color
99         ELSE
100          voxelColor(n) = 0          # below the lowest threshold, then no color
101        ENDIF
102      ENDIF
103    ENDIF
104  ENDIF
105 NEXT n
106
107 RETURN
108
109
```

Lines 84 ~ 107 are sub-program “setColor”. It’s used to normalize the light intensity on each voxel, and assign each voxel to different color group according to the threshold. The basic idea is to set the voxels with highest intensity as red, then yellow, then green, and then blue. If the light intensity is smaller than then minimum threshold, it will be omitted.

```
109
110 SUB visualize
111
112 objPath$ = $OBJECTPATH()           # get current object path name
113 lensFileName$ = $FILENAME()       # get current lens file name
114
115 FOR colorNum = 1, 4, 1
116     GOSUB createObj
117 NEXT
118
119 FOR colorNum = 1, 4, 1
120     GOSUB import
121 NEXT
122
123 RETURN
124
125
```

Lines 110 ~ 123 are sub-program “visualize”. It’s used to create 3-D model of the voxels in each color group, i.e. use a small cube to represent each voxel, and load the 3-D model of each color group into the current optical system. Line 112 and 113 read the current object path name and lens file name, respectively. The loop in lines 115 ~ 117 calls sub-program “createObj” to create a 3-D object for each color group. Finally, the loop in lines 119 ~ 121 calls sub-program “import” to import the 3-D object in each color group to the current optical system.

```

125
126 SUB createObj
127
128 objTotal = NOBJ(1)           # current total object number
129 objNum = objTotal
130
131 FOR n = 1, nvx*nvy*nvz, 1
132   IF voxelColor(n) == colorNum
133     objNum = objNum+1
134     REWIND
135     FORMAT 1 INT
136     PRINT "Inserting voxel ", n, " for group ", colorNum, " ..."
137
138     INSERTOBJECT 1, objNum    # insert one object at the end of NSC editor
139     SETNSCPROPERTY 1, objNum, 0, 0, "NSC_RBLK"    # type as rect block
140     SETNSCPARAMETER 1, objNum, 1, hwx/nvx        # x1 half width
141     SETNSCPARAMETER 1, objNum, 2, hwy/nvy        # y1 half width
142     SETNSCPARAMETER 1, objNum, 3, 2*hwz/nvz      # z length
143     SETNSCPARAMETER 1, objNum, 4, hwx/nvx        # x2 half width
144     SETNSCPARAMETER 1, objNum, 5, hwy/nvy        # y2 half width
145     SETNSCPROPERTY 1, objNum, 16, 0, 1          # rays ignore this object
146
147     nz = INTE((n-1)/(nvx*nvy))+1
148     ny = INTE((n-1-(nz-1)*nvx*nvy)/nvx)+1
149     nx = n-(nz-1)*nvx*nvy-(ny-1)*nvx
150     x = (nx-(nvx+1)/2)*2*hwx/nvx                # calculate x position
151     y = (ny-(nvy+1)/2)*2*hwy/nvy                # calculate y position
152     z = (nz-1-nvz/2)*2*hwz/nvz+hwz+5           # calculate z position
153     SETNSCPOSITION 1, objNum, 1, x
154     SETNSCPOSITION 1, objNum, 2, y
155     SETNSCPOSITION 1, objNum, 3, z
156   ENDIF
157 NEXT n
158
159 REWIND
160 PRINT "Updating object group ", colorNum, " ..."
161 UPDATE ALL
162
163 exportStartNum = objTotal+1
164 exportStopNum = objNum
165 GOSUB export
166
167 RETURN
168
169

```

Lines 126 ~ 167 are sub-program “createObj”. It’s used to create 3-D model for the voxels in each color group. Among them, line 128 reads the total number of objects in the current system, save it as objTotal; line 129 assign the number to variable objNum; lines 131 ~ 157 check each voxel with a loop, and if a voxel belongs to a particular color group (line 132), then insert a new object (line 138) at the end of the NSC editor (determined by the value of objNum+1), and set corresponding parameters (lines 139 ~ 145); lines 147 ~ 152 calculate the position of each voxel; lines 153 ~ 155 set the voxel position

accordingly; line 161 updates the current system after finishing the loop of each color group; lines 163 ~ 165 indicate the start number and stop number of each new object in the NSC editor, and call sub-program “export” to export those objects to the target file.

```

169
170 SUB export
171
172 ! setting for export
173 VEC1(1) = 3           # CAD file type, choose STL
174 VEC2(2) = 8           # spline segments
175 VEC1(3) = exportStartNum # start object
176 VEC1(4) = exportStopNum # end object
177 VEC1(5) = 1           # ray data layer
178 VEC1(6) = 0           # lens data layer
179 VEC1(7) = 0           # do not export dummy surfaces
180 VEC1(8) = 1           # export surfaces as solids
181 VEC1(9) = 5           # ray pattern NONE
182 VEC1(10) = 0          # number of rays = 0
183 VEC1(11) = 0          # wavenumber
184 VEC1(12) = 0          # field number
185 VEC1(13) = 0          # do not 'delete vignetted'
186 VEC1(14) = 1          # dummy thickness
187 VEC1(15) = 0          # do not split rays
188 VEC1(16) = 0          # do not scatter rays
189 VEC1(17) = 0          # do not use polarization
190 VEC1(18) = 0          # use the current configuration
191
192 FORMAT 1 INT
193 temp$ = $STR(colorNum)
194 comment$ = lensFileName$+"Color"+temp$+".STL"
195 exportFileName$ = objPath$+"\ "+comment$
196
197 REWIND
198 PRINT "Exporting object group ", temp$, " ..."
199 EXPORTCAD exportFileName$
200
201 FOR i = exportStopNum, exportStartNum, -1
202     REWIND
203     PRINT "Deleting object ", i, " ..."
204     DELETEOBJECT 1, i
205 NEXT
206
207 RETURN
208
209

```

Lines 170 ~ 207 are sub-program “export”. It’s used to export the objects to a target CAD file. Lines 173 ~ 190 define some parameters related to exporting file (some default values don’t need to be specified

in the program). We choose the export file type as STL (line 173). The range of the objects to be exported is determined by exportStartNum and exportStopNum (lines 175 and 176), and the objects are exported as solids (line 180). Lines 192 ~ 195 generate the target file name according to the color group, line 199 exports the object to the target file, and lines 201 ~ 205 delete all the exported objects from the NSC editor.

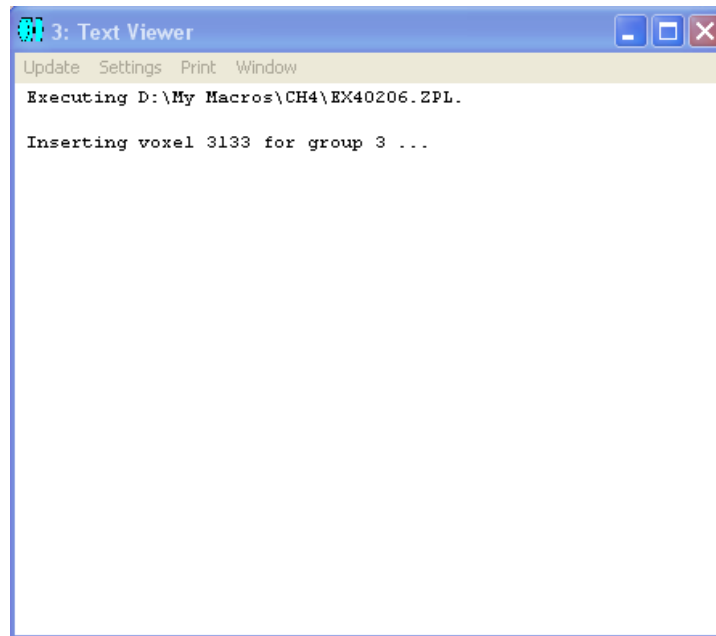
```

209
210 SUB import
211
212 FORMAT 1 INT
213 temp$ = $STR(colorNum)
214 comment$ = lensFileName$+"Color"+temp$+".STL"
215
216 newObjNum = NOBJ(1)+1
217
218 REWIND
219 PRINT "Importing object group ", temp$, " ..."
220
221 INSERTOBJECT 1, newObjNum
222 SETNSCPROPERTY 1, newObjNum, 0, 0, "NSC_STLO" # object type
223 SETNSCPROPERTY 1, newObjNum, 1, 0, comment$ # comment
224 SETNSCPROPERTY 1, newObjNum, 2, 0, 0 # reference object
225 SETNSCPROPERTY 1, newObjNum, 16, 0, 1 # rays ignore this object
226 SETNSCPROPERTY 1, newObjNum, 142, 0, 9 # opacity as 10%
227 SETNSCPARAMETER 1, newObjNum, 1, 1 # Scale be 1
228 SETNSCPARAMETER 1, newObjNum, 2, 1 # is volume
229
230 UPDATE ALL
231
232 REWIND
233 PRINT ""
234
235 RETURN
236

```

Lines 210 ~ 235 are sub-program "import". It's used to import the solid model of each color group to the current optical system for visualization. Among them, lines 212 ~ 214 generate file name according to the color group value, and please note no path name is included here, so the file must be put in the default object folder; line 221 inserts a null object at the end of the NSC editor; line 222 sets the object type; line 223 adds the comment, representing the import file name; lines 224 ~ 228 set the parameters of the imported object; and finally, line 230 updates the current optical system.

Since the running time of this program is long, we added some prompt messages in the Text Viewer window on the screen to indicate the progress of the program, as shown in line 67, lines 134 ~ 136, lines 159 ~ 160, lines 197 ~ 198, and lines 202 ~ 203. Figure 4.2-19 shows some prompt messages during the program execution.

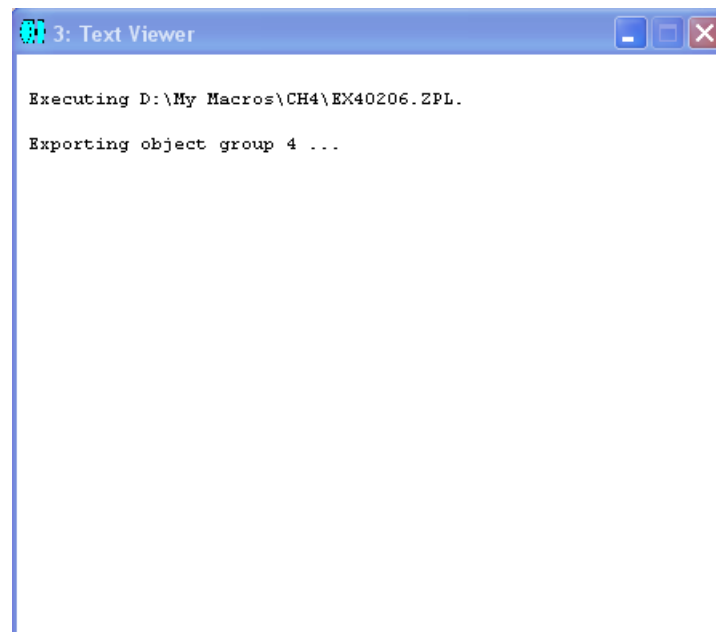


3: Text Viewer

Update Settings Print Window

```
Executing D:\My Macros\CH4\EX40206.ZPL.  
Inserting voxel 3133 for group 3 ...
```

(a)



3: Text Viewer

```
Executing D:\My Macros\CH4\EX40206.ZPL.  
Exporting object group 4 ...
```

(b)

Fig. 4.2-19: Prompt messages shown in the Text Viewer when running program ex40206.ZPL

After finish running the program, we can set the properties such as opacity and color of the imported solid model that represents the light intensity distribution in the space. This can help to visualize the distribution. Since the solid models are saved in related files, we can also import the models into other CAD software for further process. Figure 4.2-20 are the distribution of light intensity observed in other CAD software. For the purpose of publication, only gray-level graphs are plotted. It will be clearer if color is added.

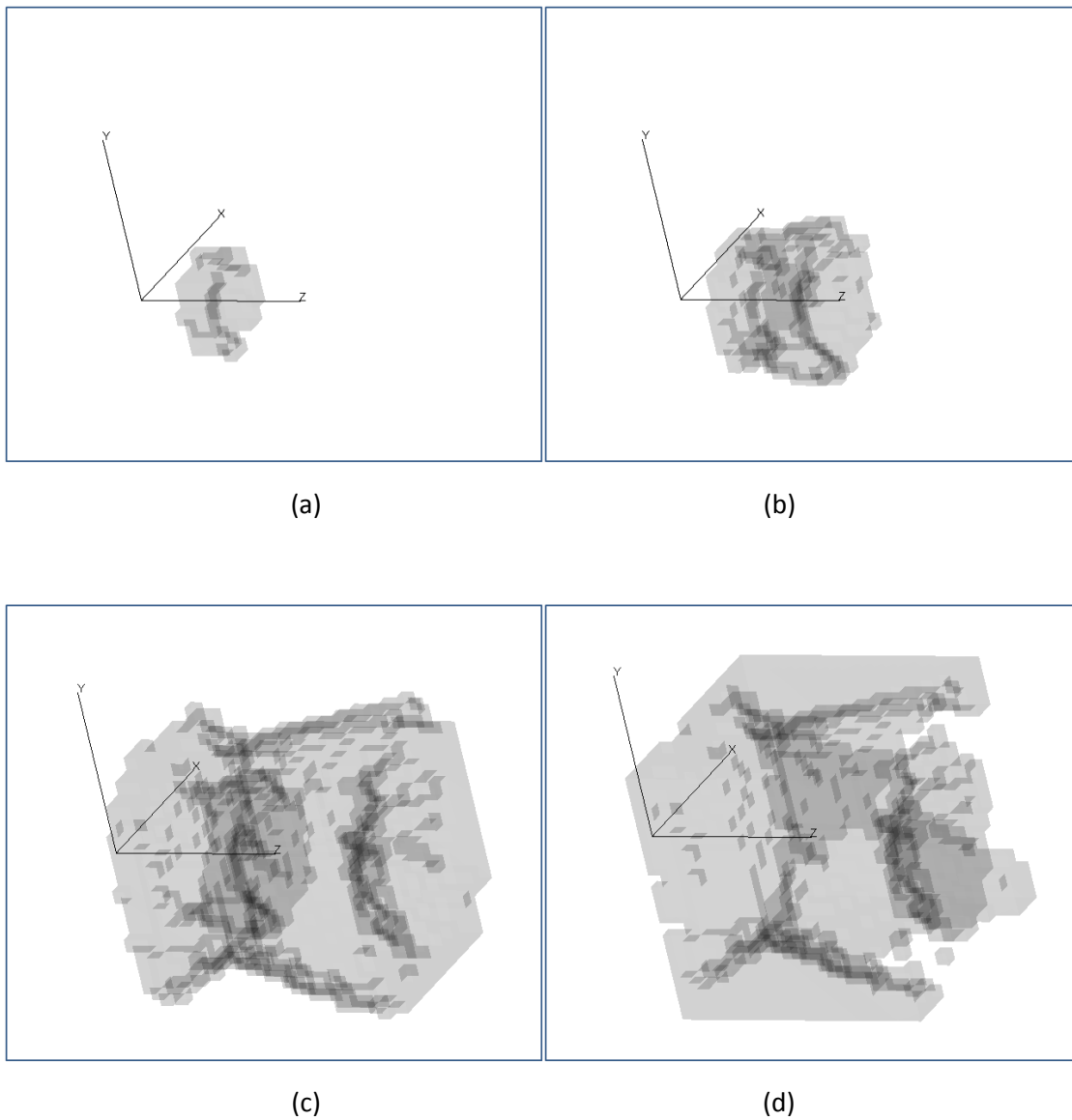


Fig. 4.2-20: Light intensity distribution in the space obtained from program ex40206.ZPL.

(a) ~ (d) Intensity from high to low.