The ASAP Primer

A basic course of study of ASAP software, the most powerful set of tools for optical analysis

Breault Research Organization, Inc.

www.breault.com

Contacting BRO

This Primer is for use with ASAP® 2006V1R1.

Comments on this manual are welcome at: **support@breault.com**

For technical support, information on additional copies of this documentation, or technical information about other BRO products, contact:

Breault Research Organization, Inc.

6400 East Grant Road, Suite 350

Tucson, AZ 85715

US/Canada:1-800-882-5085

Outside US/Canada:+1-520-721-0500

Fax:+1-520-721-9630

E-Mail:

Technical Support:support@breault.com

General Information:info@breault.com

Web Site:http://www.breault.com

bro0955man_primer (October 24, 2006)

This book is dedicated to Rob Hubbard, whose commitment to sharing his excitement for the program's capabilities, and his proven skills as a writer led him to create the *ASAP Primer* for users like you. Rob based the contents on his experience instructing beginning users in numerous classes for the ASAP Introductory Tutorial, as well as helping countless users in his role as a Customer Support Engineer.

Content

ASAP Scripts for Chapter 3 55

Building and Previewing Geometry 57

ASAP Scripts for Chapter 4 85

CHAPTER

PREFACE

The Advanced Systems Analysis Program (ASAP®) is intended to provide scientists and engineers with the most powerful set of basic optical analysis tools available. The *ASAP Primer* gives you a basic understanding of the philosophy and structure of ASAP, along with the basic commands. It's really all that is required to get started and work productively with ASAP.

Using the ASAP Primer

The Primer is intended to be primarily a course of study rather than a reference manual.

As the companion volume to the five-day class, the Introductory ASAP Tutorial (http://www.breault.com/software/k-base.php), the *Primer* covers the essential topics common to all analyses. The *Primer* is intended to be primarily a course of study rather than a reference manual. New tools and procedures are usually not introduced until you need to know about them, and they are discussed to the level necessary only for solving the problem at hand. This approach allows us to introduce complex topics initially in the simplest possible context, and revisit them later as your level of understanding and sophistication increases. The repetition of important information is also a key element in the teaching style we have selected for both the *Primer* and the Introductory ASAP Tutorial.

Learning by Doing

Conducting the ASAP tutorials has shown us that hands-on practice with ASAP is critical to learning and retaining the procedures we teach. For this reason we have included exercises at the end of most chapters in this *Primer*. Many are identical to those developed for the Introductory Tutorial. They have been honed and refined over many years to both complement and supplement the lectures. Most students agree that the tutorial was far more effective with the exercises. We urge all readers to work through these problems.

Two principal methods are available for using ASAP.

1. The Graphical User Interface (GUI)—gives access to most ASAP commands via the ASAP Builder, and a well-organized system of menus and dialog boxes.
2. The command scripting language—in conjunction with the macro language, it turns ASAP into a powerful optical programming language.

ASAP users who plan to work mainly with command scripts will still find their time well spent by working through the early chapters of this book.

While this *Primer* does not specifically address command scripts until near the end, ASAP users who plan to work mainly with command scripts will still find their time well spent by working through the early chapters of this book. Considerable parallelism exists between the terminology and usage of the GUI and command scripts. The procedures employed in both cases are fundamentally the same. Further, many tools and aids were built into the GUI to make the transition to command scripts simple.

In this Primer, whenever the GUI is used to create an ASAP command, a cross-reference to the command script is included; for example,

See Chapter 3 Appendix, "Script 3-1" on page 55.

You can go to the page of the appendix by clicking the part of the sentence with the script and page numbers; for example, ***"Script 3-1" on page 55***. This part of the cross-reference is a link to the script on the designated page. Note that you can easily return to the page you just left, as described in step 3 below.

To follow a link to the associated script:

1. Select the hand tool, a zoom tool, or a selection tool from the Adobe® Acrobat® toolbar.
2. Position the pointer over the linked area on the page until the pointer changes to the hand with a pointing finger. Then click the link.
3. To return to the previous view (the page where the link was referenced), click the arrow on the toolbar with the ToolTip, **Go to Previous View**.

Typographical Conventions

We have used several typographical conventions consistently in this *Primer*. References to the menus, windows, and dialog boxes in ASAP appear in the following typeface and style, **Builder**. Menu commands are referenced from the top level down, with submenus separated by the forward arrow (**>**). For example, **Analysis> Choose Rays> Consider** implies that the **Consider** dialog box is reached from the **Analysis** menu and its **Choose Rays** submenu. Any reference to an actual ASAP command and its options are displayed in the following typeface, reminiscent of the default script editor font: `TRACE PLOT`. Even when we are working with the GUI, we use this typeface to make the point that this is the name of an actual ASAP command.

Command arguments that users need to enter appear as <*name*>. This implies that you should substitute your choice of name in place of the string <*name*> (excluding the <> delimiters). For example, when you see <install directory>, you need to substitute the name of the ASAP directory where the program is stored.

Other ASAP Resources

This *ASAP Primer* is only one of many resources provided to you for learning ASAP. As stated above, it addresses those concepts and procedures common to virtually all ASAP applications.

A comprehensive series of ASAP Technical Guides can carry your learning forward from there. The guides do not depend on one another, and therefore can be read in any order. They are described in the last chapter of this *Primer*, and are frequently referenced in earlier chapters.

A succinct reference-style manual, the *ASAP Reference Guide*, is available to you for viewing or printing. The PDF file, `broman0128_reference.pdf`, is on the distribution CD.

The full content of the *ASAP Reference Guide* is derived from the on-line Help, which is available from within the ASAP program. We describe how to use on-line Help in several places in this *Primer*.

The manual is also one of many information products located on the distribution CD, and a link for installing it is available on the "Documentation" page of CD Autoplay. We encourage you to install documentation files from this page.

> ***TIP*** All ASAP technical publications, with the exception of the *ASAP Reference Guide*, are available in the Knowledge Base:
>
> http://www.breault.com/k-base.php.

More About Optics

An increasing number of ASAP users have little or no formal training in optics. It is likely, in fact, that these successful ASAP users are now in the majority. We attempt, both in this *Primer* and in the Introductory Tutorial, not to presume too much about your knowledge of optics and optical engineering. However, some terms and concepts are likely to be unfamiliar to some readers.

In some cases, we have added a brief ***Note*** or boxed sidebar that can be easily skipped if the topic is already well known to you. Unfortunately, it is not practical to attempt to teach all the basic optics that you might need in order to use every feature in ASAP.

For general information on optics, we recommend

- *Optics,* by Eugene Hecht for an excellent, general grounding in the physics of optics, and
- *Modern Optical Engineering*, by Warren J. Smith, for more about optical engineering and design topics.

Both of these books, in turn, contain extensive bibliographies.

CHAPTER

1

INTRODUCTION

What is ASAP?

ASAP is fundamentally a flexible and efficient optical system-modeling tool. It is able to simulate, by Monte Carlo ray-tracing techniques, the interaction of light with optical and mechanical structures. It does this in a single, global, three-dimensional coordinate system, making no assumptions about symmetry. (See Figure 1.1 on page 22.)

What ASAP is really doing is just a simulation based on the way that real light behaves in the real world.

Rays can automatically split into reflected, refracted, diffracted, polarized, and scattered components as they propagate through the system. The rays proceed independently, following physically realizable paths, encountering objects in any order, as appropriate. This type of ray tracing is often described using terms like “unconstrained” or “non-sequential”. What ASAP is really doing, however, is just a simulation, based on the way that real light behaves in the real world.

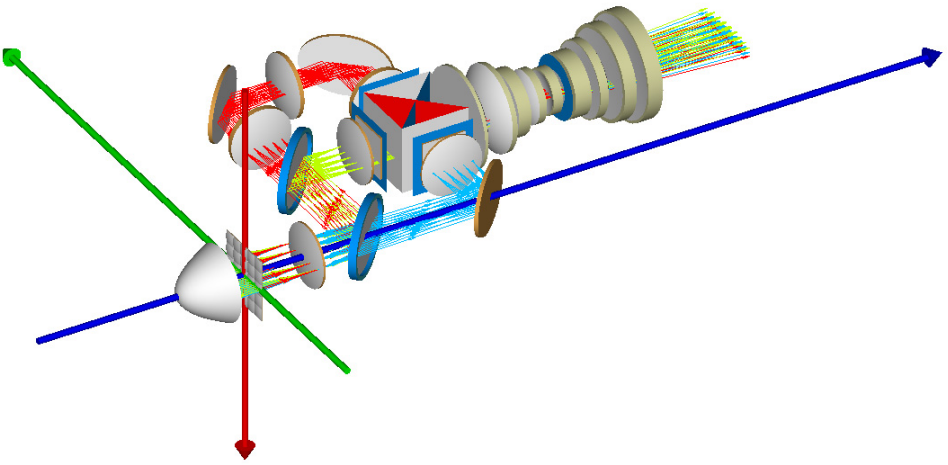


Figure 1.1 ASAP works in a single, global, three-dimensional coordinate system. The user chooses where to put the origin of coordinates and how to orient the system, relative to these traditional Cartesian axes. In this model of an LCD projector, the red, green, and blue arrows represent the x, y, and z axes respectively. The origin was arbitrarily placed at the center of the Kohler lens array.

ASAP can also simulate coherent and diffractive optical systems with a relatively simple but powerful extension of ray tracing methods known as Gaussian beam decomposition. Any complex field can be decomposed into a set of Gaussian beams that in turn can be traced as described above by ray-trace methods. This allows us to work on problems involving coherence, diffraction, interference, coupling efficiency, and other situations where phase matters. (See Figure 1.2 on page 23.)

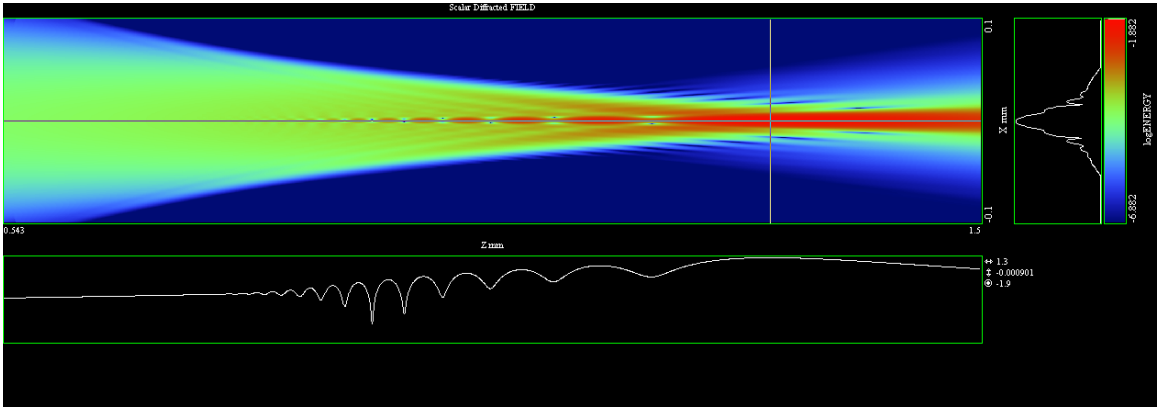


Figure 1.2 A cross-section of a coherent field, propagating from left to right through the focus of a ball lens. This logarithmic cross-section shows the effects of both diffraction and spherical aberration present in the field. ASAP is able to perform this type of work by decomposing an initial field into a set of Gaussian beams, which can be propagated individually by ray-tracing methods. The result shown here is the coherent superposition of the individual beams comprising the field.

While this topic is not covered in the chapters that follow (see, instead, the ASAP Technical Guide, *Wave Optics in ASAP*), most of the methods and techniques described here are also essential to working on "beam" problems in ASAP.

ASAP versus Classical Lens Design Codes

Occasionally, confusion exists about the difference between ASAP and classical lens-design codes, which also are used in optical design and involve ray tracing. How is ASAP different?

The primary function of a lens design code is to evaluate many combinations of lens parameters—like glass type, thickness, and surface curvatures—to provide an optimum solution, based on user-specified criteria or "merit functions". This function can be done efficiently, when the rays are constrained to proceed through the system of optical elements *sequentially*, one surface after another in a prescribed order. (See Figure 1.3 on page 24.)

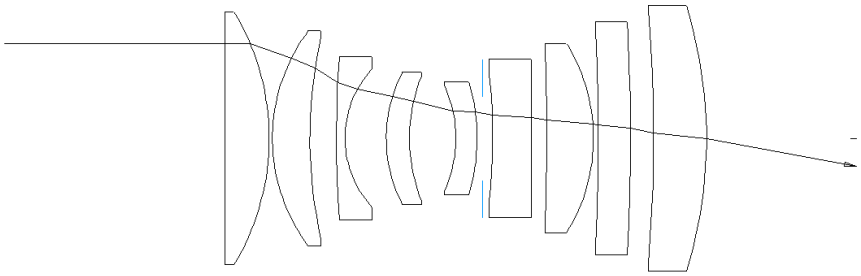


Figure 1.3 Lens-design programs are able to evaluate many possible candidate designs rapidly, when the rays are constrained to move through the lens system elements in a prescribed sequence. The single ray shown here must enter from the left, and encounter each refracting surface in order, until leaving the system to the right.

ASAP can take a basic lens design and fold in Fresnel losses, multiple reflections, scattered light, diffraction, and other non-ideal or more complex behavior.

With this constraint, it is typically possible to find the exit parameters of a ray, simply by feeding the input parameters into a mathematical expression. This process is what allows the rapid evaluation of many design solutions.

The type of analysis performed by lens design programs, while sometimes essential to developing the initial system specification, generally presumes perfect optical surfaces. It neglects Fresnel losses, multiple reflections, scattered light, diffraction, and other non-ideal or more complex behavior. But ASAP can take a basic lens design and fold in these effects, giving a better prediction of the "as-built" performance of the complete opto-mechanical system. (See Figure 1.4.)

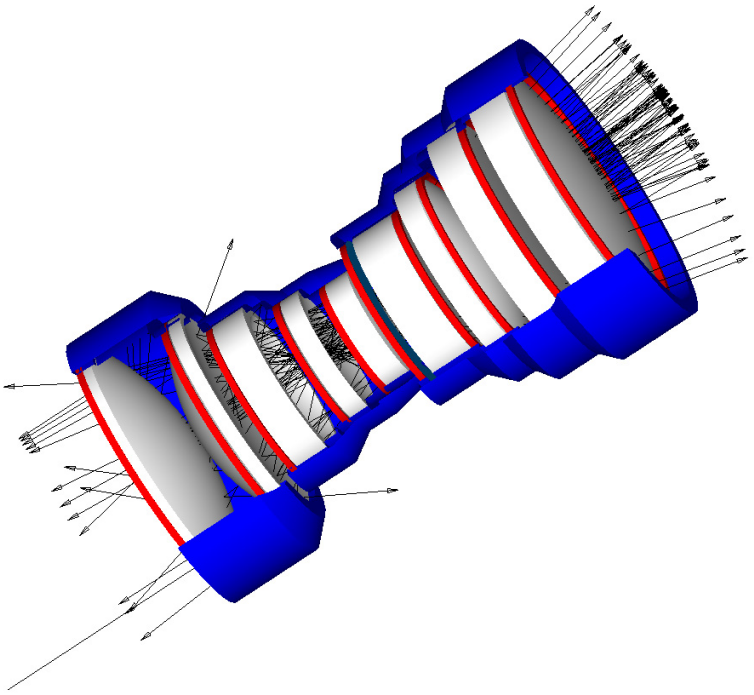


Figure 1.4 A single input ray (lower left) can divide its energy many times as it reflects, refracts, and scatters, interacting with both optical and mechanical components. ASAP performs such analyses without any prior knowledge of the order in which a ray will encounter the various surfaces.

ASAP for Illumination Systems

ASAP is also used extensively to evaluate illumination and other non-imaging systems. The issues are exactly the same: how will the system perform with realistic, manufactured components instead of over-simplified, ideal optics? The ultimate “over-simplified” flashlight model is a point source at the focus of a perfect parabolic reflector. (See Figure 1.5a.)

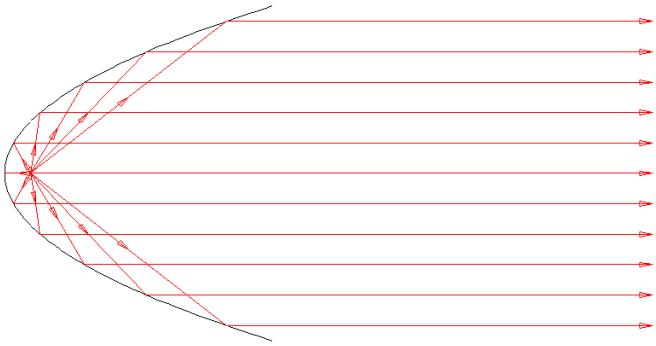


Figure 1.5a A parabolic reflector has the property that any ray starting at its focus will reflect parallel to its axis. Such a reflector could, in principle, produce a symmetrical illumination pattern in front of the reflector.

The result is a perfectly symmetrical illumination pattern every time. Such a flashlight has never been built, however, and probably never will be. A real point source (if there were such a thing) would have some rays coming off the reflector, but would include others that miss the reflector and exit straight through the opening. (See Figure 1.5b.) This problem is no longer strictly "sequential".

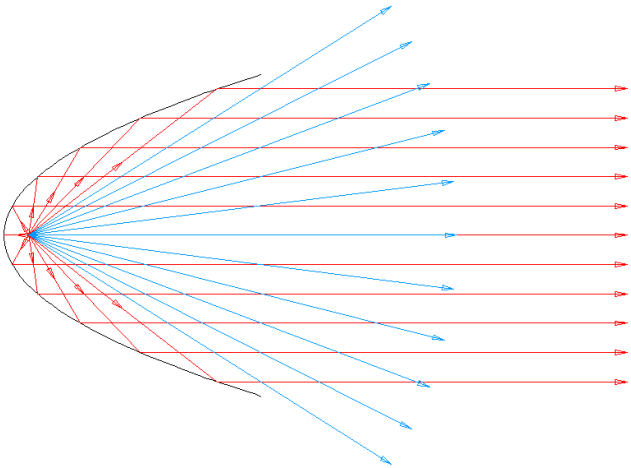


Figure 1.5b A point source located at the focus of a parabolic reflector and radiating in all directions will produce some rays (shown in blue here) that exit this simple system without striking the reflector. This is no longer a sequential ray-tracing problem, since not all rays strike all objects in the same order.

At the least, we need to model an extended source with the appropriate shape and the emission properties of incandescent metal. Next, we can add surface roughness to simulate the vacuum metallized plastic material used in practice for manufacturing the reflector. (See Figure 1.6.)

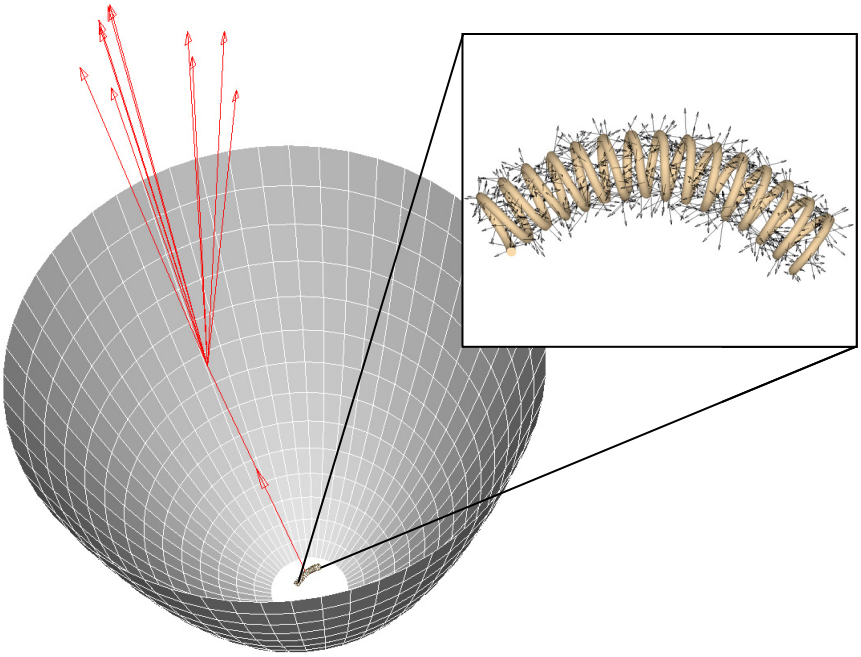


Figure 1.6 A better model of a flashlight includes an extended source, like an incandescent helical filament and appropriate roughness on the reflector, causing some scattering of the rays.

Now the source is a little more realistic, and the rays will "scatter" in directions appropriate to a non-ideal (but realistic) surface. The progression of results, as these features are added to the model, is shown in Figure 1.7a, b, and c.

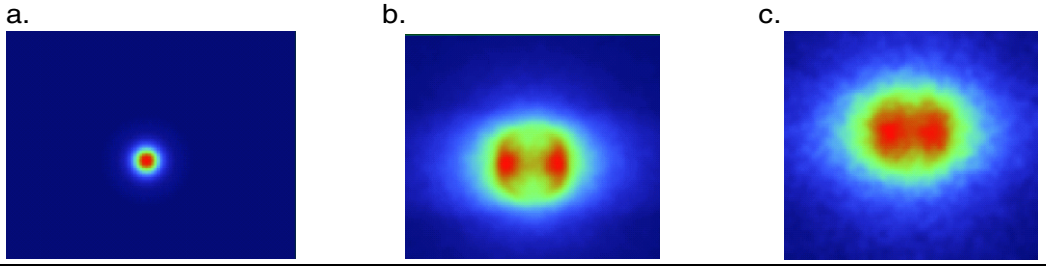


Figure 1.7a A point source at the focus of a perfect parabolic reflector yields a symmetrical, concentrated illumination pattern.

Figure 1.7b When we replace the point source with an extended source, in the shape of a helical filament, the result is a much broader, asymmetric pattern.

Figure 1.7c A real reflector usually has some roughness associated with it. This may be only a result of the manufacturing process, or it may be a design parameter—under the control of the illumination engineer.

For even better results, we can develop a detailed model of the bulb that includes filament support structures, the mounting base, and the glass envelope to cause the appropriate stray reflections, refraction, shadowing, and unexpected lensing effects. An example of such a model is shown in Figure 1.8.

With the appropriate model in place, ASAP can predict the realizable behavior of an illumination design.

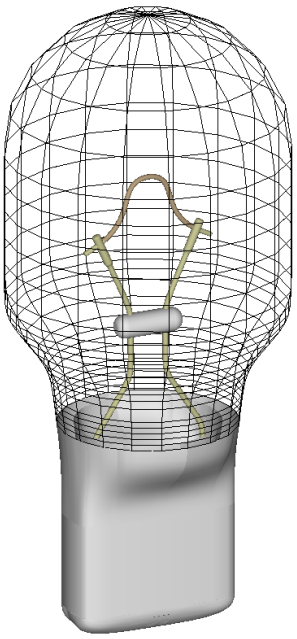


Figure 1.8 For best results, a light source model should include the glass envelope, spacers, electrical posts, and anything else that will cause reflections or shadows. This image represents an ASAP model of a W16W automotive filament lamp.

Accuracy Issues

ASAP excels at providing the right tools to provide state-of-the-art modeling, but the user still has to "build" the model.

People frequently ask, "How accurate is ASAP?" This, of course, is the essential question for any practicing engineer who is trying to predict or explain the behavior of a real system. ASAP does all critical calculations in double precision floating-point arithmetic, but that fact alone does not tell us much. The real answer depends critically on two things:

1. How good are the assumptions built into your system model?

   For example, do you have accurate measurements of the scattering characteristics of your surfaces? Have you included manufacturing tolerances? Is geometrical ray tracing even appropriate for your problem, or should you be using the program's coherent beams or beam propagation methods to simulate coherent or diffractive effects?

   ASAP excels at providing the right tools to provide state-of-the-art modeling, but the user still has to "build" the model. If the best information is not available, you need to at least explore the sensitivity of your model to changes in questionable assumptions.

2 Have you traced enough rays?

As the “Monte Carlo” description implies, “chance” (that is, statistics) is generally an essential element in the ASAP approach. The expected accuracy as a function of the number of rays traced can often be predicted from first principles, given a little grounding in statistical methods, but this is usually not necessary. In practice, we can just double the number of rays and see whether the answer is stable to within the desired precision. If not, trace more rays.

The Four-Step Process

While the foregoing may have motivated you to begin learning ASAP, it is possible that you are a little overwhelmed by the task at hand. The work is easier than in seems because there is a simple four-step process that underlies every ASAP project:

1 Build the system model

Define and verify the geometry and assign optical properties to each component in your system. While this step often requires the maximum amount of attention on the part of the ASAP user, we can often start simply and add complexity later.

2 Create sources (rays)

Define and verify a set of rays that accurately simulates the optical characteristics (spatial, angular, power distribution, and coherence) of your radiation sources. This step is also a critical part of an accurate optical model, and sometimes gets less attention than it deserves. Source modeling is, however, something that ASAP excels at, offering a variety of tools to make realistic sources.

3 Trace the rays

Allow the rays to move through your system. The good news here is that this is a task performed by ASAP, and requires little or no attention from you. In fact, ASAP will probably be doing your longer ray traces while you are eating lunch, or at home for the evening.

4 Perform the analysis

Calculate power, irradiance, intensity, or other performance characteristics at critical places within your system. This step is usually the most interesting and rewarding one. ASAP provides many tools to process, manipulate, and graphically visualize these results.

In the chapters and exercises that follow, we will go through this cycle of steps many times. We will start with simple models and simple analyses initially, and add refinements with each successive pass through this process.

CHAPTER

2

A SHORT TOUR

In this chapter, we describe the basic features of the ASAP™ User Landscape. We give only a general description of many of these features at this time, saving most of the operational details for later.

Overview

After you have installed ASAP on your computer, you can start the program either by selecting it in the Windows® **Start** menu (**Start> ASAPxxxx> ASAP**) or by double-clicking the ASAP icon on your desktop.

Many of the elements in the ASAP landscape can be resized, undocked, moved, or even hidden completely.

The first time you run ASAP, the landscape appears as shown in Figure 2.1. This basic view is highly customizable, however. Many of the elements you see can be resized, undocked, moved, or even hidden completely (toggled) to recover the space they are using. If you make changes, they are not only remembered the next time you load ASAP, but even the next time you load a new version of ASAP. For now, we will leave all these windows in their default positions. A thorough discussion of customizing options will be presented in Chapter 12, “More About the ASAP Landscape”, after you have had more experience with ASAP.

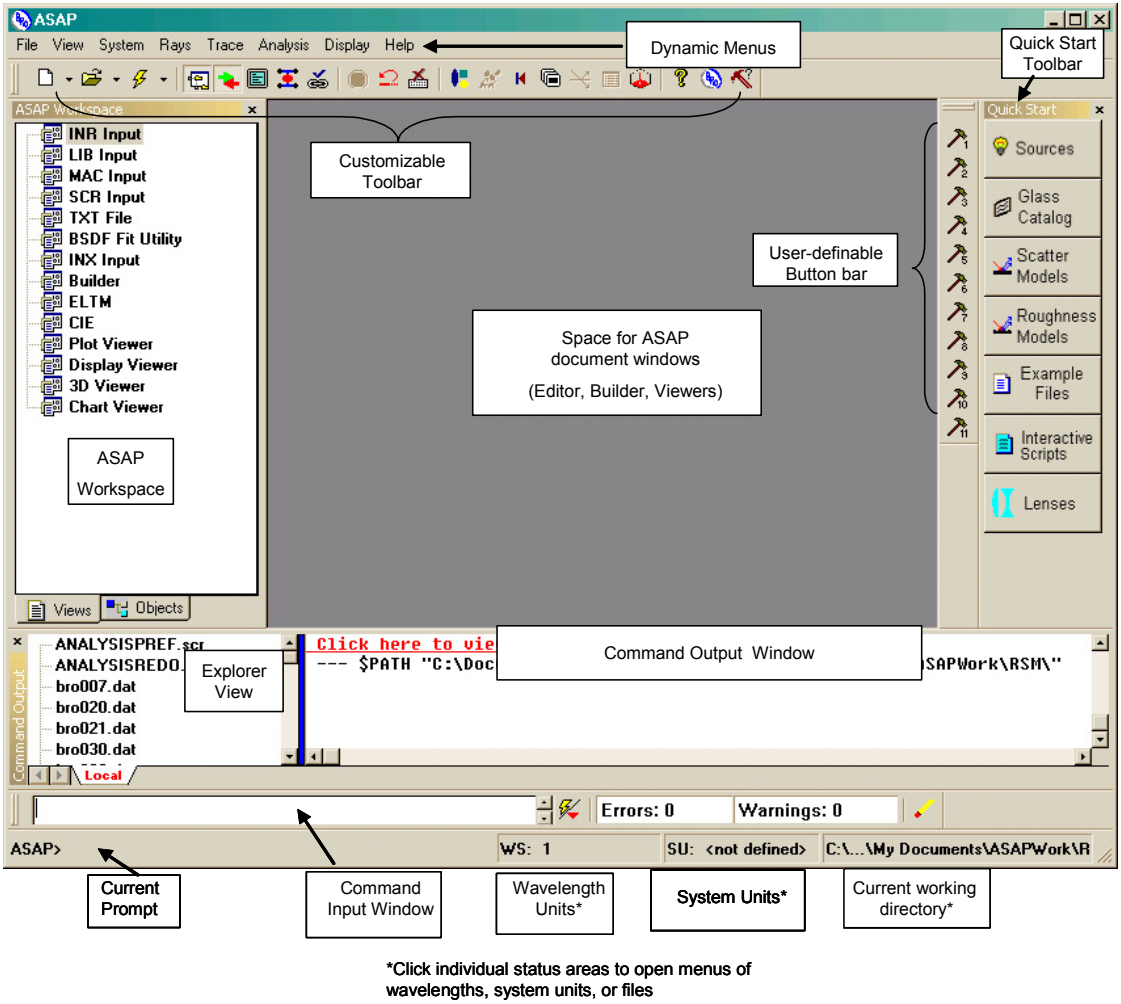


Figure 2.1 The ASAP Landscape, showing the default location of the various elements of the landscape..

ASAP Toolbar

The most commonly used functions in ASAP have buttons assigned to them. You can initiate all but a few of these activities by other means, such as scripts, drop-down menus, or by direct input into the **Command Input** window (see “Command Input Window” on page 40). The buttons you choose to use depend entirely on your style of running this and other Windows® programs. The

function of each button is described for reference in Table 2.1. There is no need to study them in any detail, since we will discuss them as we need them.

Table 2.1 Main Toolbar in ASAP

	Opens a new window. The triangular arrow displays the Editor and other document types.
	Opens files of all types. The triangular arrow displays recently opened files.
	Runs the current ASAP file. The triangular arrow displays a list of previously run files.
	Opens or closes the ASAP Workspace window.
	Opens or closes the **Command Output** window.
	Switches from your preferred screen setting to full screen. Press F11 to return to most recent setting.
	For ASAP/Pro users, Remote spawns multiple ASAP sessions on other computers on your Local Area Network (LAN).
	When selected, you can open a specified plot window from a plot link in the **Command Output** window. To set this feature for future plots, go to the **User Interface/Plot Viewer** tab (**File> Preferences**).
	Stops the current run. When the kernel is active, the button appears red.
	Deletes current system and sources, returning system settings to their default values.
	Restarts the kernel.
	Opens the **Builder**.
	Plots a set of profiles (slices) of the system.
	Rewinds the 3D graphics file (*.vcr).
	Displays the 3D graphics file (*.vcr).
	Traces rays.
	Shows statistics of current ray data.
	Shows an isometric plot of display data.
	Path Explorer Query Builder

Table 2.1 Main Toolbar in ASAP

	Opens the Contents window in ASAP on-line Help.
	Opens information about the currently installed ASAP version.

Custom Toolbar

Since not all users agree on what the "most commonly used functions of ASAP" might be, ASAP also provides 10 predefined buttons and 10 user-definable buttons. The 10 predefined buttons can be redefined by the user. These custom functions can be simple or complex, involving one ASAP command or many. But you need to know some ASAP commands before you start defining your own button functions.

Dynamic Menu Bar

Our initial experiences with ASAP are exclusively with the Graphical User Interface (GUI). This approach continues until Chapter 21, "Analyzing Directional Distributions", when we introduce command scripts. As a result, most of our early interaction with ASAP is by way of the menu bar, located (at least in the default configuration) at the top of the ASAP landscape. It is "dynamic" in the sense that the entries change, depending on which window currently has "focus". In this way, the windows do not need to have menu systems of their own.

> ***Note***: "Focus" is a Windows term. With a multiple-document interface like ASAP, only one of its windows currently has focus. This window is usually the last one created or updated, or it is the last one you clicked to activate. The window that has focus has a different color title bar at its top.

Each of the following ASAP menus is part of an ASAP process: **System**, **Rays**, **Trace**, and **Analysis**. All the other menus are either "standard Windows", (like **File**, **Edit**, and **Help**) or specific to a particular ASAP window that currently has focus.

Status Bar

At the bottom of the ASAP landscape, you will see the status bar. This shows several important things about the state of the ASAP computational engine or "kernel". When you first start ASAP, you see the stationary **ASAP>** prompt on the left, which means that the kernel is idle. This space is also used to display ToolTips when the cursor is placed over any button in the GUI.

The next two items are there to remind us of the unit system in which we have chosen to work. Since we have not declared these yet, we see only, **SU: <not defined>** and **WS: 1**. The wavelength scale factor is at its default value of **1**. These are the first preferences we will change when we begin to define our first system in the Chapter 3, “Builder and Geometrical Preliminaries”.

On the right, you can see what looks like a path name. This is your current working directory. You and ASAP create both temporary and permanent files as you work, and these are all located (by default) in your working directory. Since this path name can be a long string of characters, ASAP often has to truncate the display. But if you place your cursor over this area in the status bar, ASAP displays the full path name. We will show you how to change this path at the beginning of the next chapter.

ASAP Workspace

The **Views** *tab gives us control over what may eventually be dozens of overlapping windows.*

At startup, the **ASAP Workspace** window takes up the left side of the landscape. It has two tabs at the bottom: **Views** and **Objects**. The **Views** tab gives us control over what may eventually be dozens of overlapping windows. Figure 2.2 shows what the **ASAP Workspace Views** tab looks like once we have a few windows open. Any document window can be moved on top of the User Task Space just by clicking its name on the **Views** tab. The windows are sorted by type.

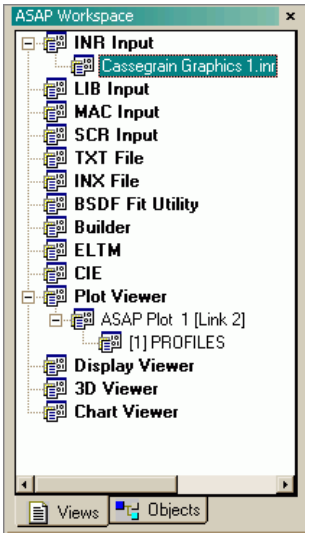


Figure 2.2 The **Views** tab in the **ASAP Workspace** window, when a few windows are open. Use this feature to switch between windows, open new windows, or close windows.

The **Objects** tab displays the names and numbers of the objects that we build in our system. If you choose to use the hierarchical naming conventions in your

definitions, ASAP creates an indented drawing-tree style of list, like that shown in Figure 2.3.

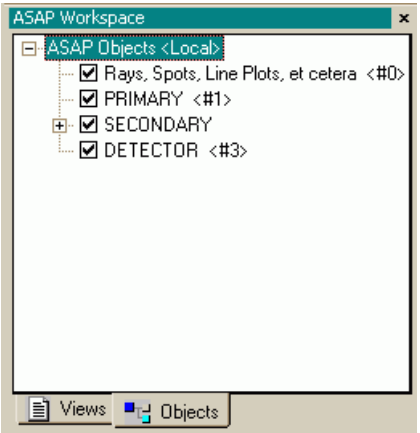


Figure 2.3 Once objects are created, the **Objects** tab in the **ASAP Workspace** window displays them in the style of a drawing tree.

Quick Start Toolbar

The Quick Start toolbar gives you quick access to the following tools in ASAP:

- Light sources, organized by type,
- Glass, media, scatter models, and roughness models for applying to ASAP objects,
- Example scripts, which are INR files of isolated and simple commands that show how the commands can be used,
- Interactive scripts for Bio-Optical Models, Array Enhancers, and Lens Creation. You can also add new scripts you create for future use, and
- Lenses, a catalog of predefined lenses.

Customizing Toolbar Buttons

The toolbars on the main window in ASAP and the Builder window can be customized. You can choose which available buttons to add, remove, or move. Right-click within the empty area of the toolbar (main or Builder) to access the **Customize** dialog box.

User Task Space

The large User Task Space is the area where your various document windows appear. You can enlarge a window to fill the entire user task space, or even your entire screen.

Command Output Window

*Any entry that begins with three dashes (for example, **`---$PATH`**) is an ASAP command.*

The **Command Output** window is how the ASAP kernel communicates with you. We direct the kernel by sending it commands. We generate these commands with the menu system, the **Builder**, command scripts, or directly via the **Command Input** window. The ASAP kernel echoes these commands to us as it runs the command. Any entry that begins with three dashes (for example, **`---$PATH`**) is an ASAP command. If you are eager to start learning the command language right away, just watch the command output window as these commands fly by. We will discuss the ASAP command language in detail in Chapter 22, “Writing Command Scripts”.

The **Command Output** window is also where the kernel writes most information that we have requested, like the results of calculations, lists of coatings or glass types, and so forth. ASAP writes other information here that it thinks might be of interest to us, like the number of rays generated by, and total flux assigned to a source we are creating.

Error messages also appear in the **Command Output** window, displayed in red. These error messages are perhaps the most important reason to pay attention to this window. If you did not see the result you expected during an ASAP operation, an explanation may be offered in this window.

Note that the **Command Output** window is, in effect, keeping a sequential history of what has recently occurred during the current ASAP session. This history is often useful when you are trying to determine why your current result is different from something you saw a while ago. All the steps used to create both results are a matter of record, still visible in the **Command Output** window by scrolling upward.

Command Input Window

You will occasionally find it useful to communicate directly with the ASAP kernel by typing single commands into the **Command Input** *window.*

After you have learned a few ASAP commands, you will occasionally find it useful to communicate directly with the ASAP kernel by typing single commands into this window. ASAP maintains a history list so that you can access recently issued commands, even from past ASAP sessions. The **Command Input** window also shares space with the **Error** and **Warning** counter, and a button to reset these.

CHAPTER

3

BUILDER AND GEOMETRICAL PRELIMINARIES

In this chapter, we introduce the ASAP Builder, and begin preliminary work on our first optical system. We start with the groundwork necessary to build a simple model of a three-element lens system, known as a Cooke Triplet.

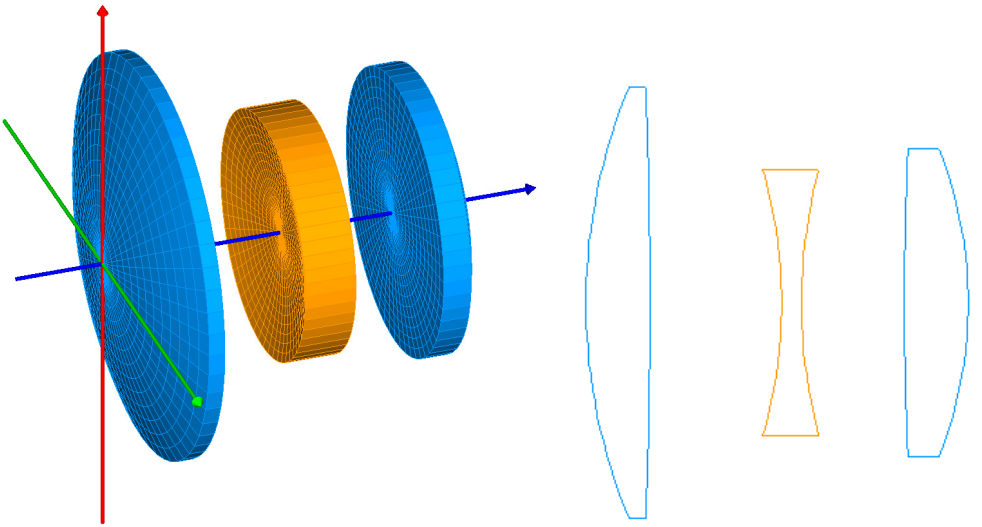


Figure 3.1 Cooke triplet—a classic lens design.

In ASAP, our job is to take the designer's specifications and turn them into an ASAP model with realistic physical assumptions that control how the light interacts with various surfaces.

The Cooke triplet is important because its configuration has just enough degrees of freedom to allow the lens designer to correct the primary aberrations completely. In ASAP, our job is not to design such things, but to take the designer's specifications and turn them into an ASAP model, with realistic physical assumptions that control how the light interacts with various surfaces.

If your computer is at hand, we strongly recommend that you follow along during this and the next chapter. We step through the first element of the Cooke triplet, and leave the creation of the other two lenses for "Exercise 1: Completing the Cooke Triplet" on page 83.

Setting the Working Directory

The first step in starting any new ASAP task is deciding where in your computer's directory system you want to store the results. As we noted during the discussion of the task bar in the last chapter, various files are created while ASAP is running, and these are placed in the working directory. You can set the working directory from the ASAP menus: **File> Set Working Directory** (near the bottom). The resulting dialog box (see 2) allows you to browse to the place you want to store your work, creating new folders as you go. You may find it convenient to place all work that you perform while reading the *Primer* in a single working directory, with a name like **ASAP Primer**.

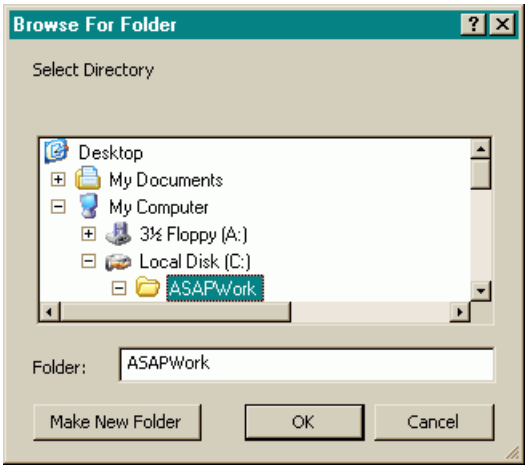


Figure 3.2 When you start an ASAP project, the first step is generally to choose a place on your computer to store the files. The **Browse For Folder** dialog box allows you to do this, and to create new folders as needed.

When you complete this process by clicking **OK**, the ASAP status bar reflects the change. Next time you start ASAP, the program automatically returns to this working directory. When you place your cursor over this area in the status bar,

you can view a list of recently used working directories. You can reselect one of the recently used directories by clicking on the name.

Basic Procedure

Now you are ready to build your first system, the Cooke triplet. We will follow a logical six-step procedure, outlined below.

1. Define system settings:

   Units,

   Wavelengths.

2. Define optical materials and coatings:

   Refractive indices,

   Reflection and transmission coefficients.

3. Define a geometrical shape for each object.
4. Assign optical properties to each object.
5. Shift and rotate each object to the desired location and orientation.
6. Graphically verify the shape, location, and orientation of each object, as it is constructed.

Options for Building Geometry

Several different ways exist to accomplish these steps in ASAP. We can:

- Use the Builder, which is a spreadsheet-style tool, built into ASAP for creating geometry and sources and doing analysis.
- Build the geometry, using a computer-aided-design (CAD) program, and import the shapes into ASAP. (To do this, your ASAP license must be CAD+.) Using a CAD program, you accomplish step 4 above as a manual procedure during the import process.
- Use the command language to write an ASAP command script, type the same commands directly into the **Command Input** window, or combine the **Builder** and scripts by entering ASAP commands into a script file, using the **Mini Builder**; and
- Import lens designs directly from Zemax®, SYNOPSYS™, OSLO®, or Code V®, using the appropriate translators available with ASAP Optical+ with:

1. ASAP CAD+, import SolidWorks® geometry files,
2. ASAP COMPLETE, import CATIA files via the CAA V5 plug-in.

- With the CAD module, import SolidWorks® geometry files.
- With CATIA/CAD, import CATIA files via the CAA V5 plug-in.

In the early chapters of this *Primer*, we concentrate on the first option, the Builder**.** This environment is the best one to use when you are first learning ASAP, because of the labeled column headings and built-in previewing available with this tool. Later in the *Primer*, we also cover the command language method. If you eventually expect to use the command language, learning the Builder is still time well spent. The Builder generally uses the same naming conventions and syntax as the corresponding script command, so making the transition will be easy. As pointed out in Chapter A, “Preface”, whenever we use the user interface in this *Primer* to create an ASAP command, a cross-reference to the chapter’s appendix, which contains the corresponding command script, is included.

Builder Introduction and Basics

You can start the Builder either by clicking the Builder button or from **System> Builder**. You can expect to find most of the ASAP tools for building and verifying your system on the **System** menu or one of its sub-menus. The Builder window opens in the user task space.

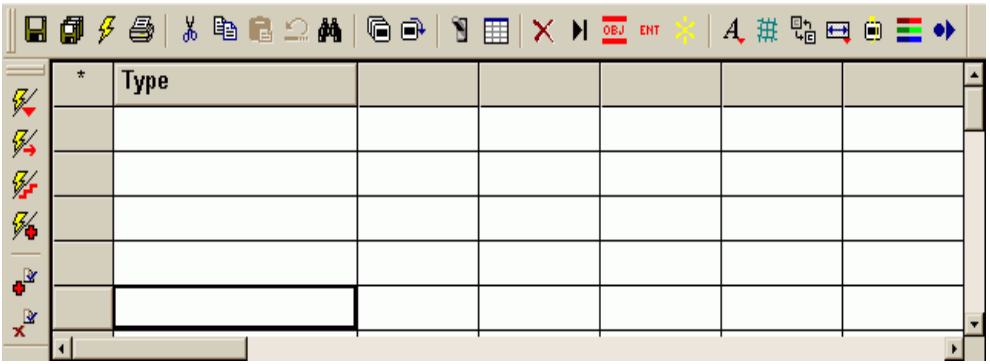


Figure 3.3 Builder Window with toolbars

Since this is probably the only window you have open at the moment, you may as well use all the space available by clicking the maximize button at the top right corner of the window. You can also use the **F11** key to fill your entire screen with the Builder, but remember to press **F11** again to return to the ASAP landscape when you are finished.

You may also have noticed that the **ASAP Workspace Views** tab now has an entry called “Builder 1” under “Builder”, and two new menus (**Preview** and **Builder**) have appeared among the other menus. These two menus persist as long as the Builder has focus.

A first look at the Builder reveals a spreadsheet-style document with its own scroll bars and toolbar. The meaning of each button is described for reference in 2, but as before, we will introduce these functions individually, as you need them. While the Builder has little in common with an actual mathematical spreadsheet, we refer to the individual boxes in the window as “cells”.

Table 3.1 Toolbar for the **Builder**

	**Save** this Builder file.
	**Save as**...
	**Run** this Builder file.
	**Print** this Builder file.
	**Cut** to Clipboard.
	**Copy** to Clipboard.
	**Paste** from Clipboard.
	**Undo** the last change.
	Opens the **Find Command** dialog box from an empty Builder line.
	**Preview all** current geometry.
	**Preview only** selected Builder lines.
	**Delete** the current line.
	Opens the dialog box for perturbation of tolerancing data.
	Opens the **Tolerancing** dialog box for setting up the distribution for the analysis.
	**Insert** a line.
	**Switch line status** to ignore or “comment out” a command.
	Changes a geometry definition to an “**Entity**”.
	Explode a lens specification.
	Change **font** size in the Builder.

Table 3.1 Toolbar for the **Builder**

	Change the default **faceting** level in previews.
	If selected, **append** new geometry (that is, do not delete all previous geometry definitions).
	Change the default system **units**.
	If selected, all new geometry has "**entity**" rather than "object status.
	If selected enables **color** coding of Builder lines.
	If selected, opens the Macro Builder dialog box to create or edit a user-defined macro.

The Debug Bar facilitates running from and to the cursor, stepping through the file, and running to a breakpoint. It includes the buttons described in 2.

Table 3.2 Debug Toolbar in the Builder

	Runs the current Builder (.enx, enz) file from the point at which the pointer is currently located within the file.
	Runs the current .enx or .enz file from the beginning of the file to the line above the pointer point.
	Runs the current command from the point at which the pointer is currently located, and then moves the pointer to the next line.
	Runs the current .enx or .enz file to the next breakpoint.
	Inserts or removes one breakpoint in the .enx or .enz file
	Removes all breakpoints in the .enx or .enz file.

Although we have not actually built anything yet, this would be a good time to save the new Builder file, thus giving it a better name than the default "Builder 1". You can do this from **File> Save As** (as long as the Builder window currently has focus), or by clicking the **Save As** button in the Builder's toolbar. "Cooke Triplet" is a good choice for a file name.

Units

You can access all the commands available in the Builder by double-clicking a cell in the **Type** column (the first column) of any line. This action produces the pop-up menu shown in 2.

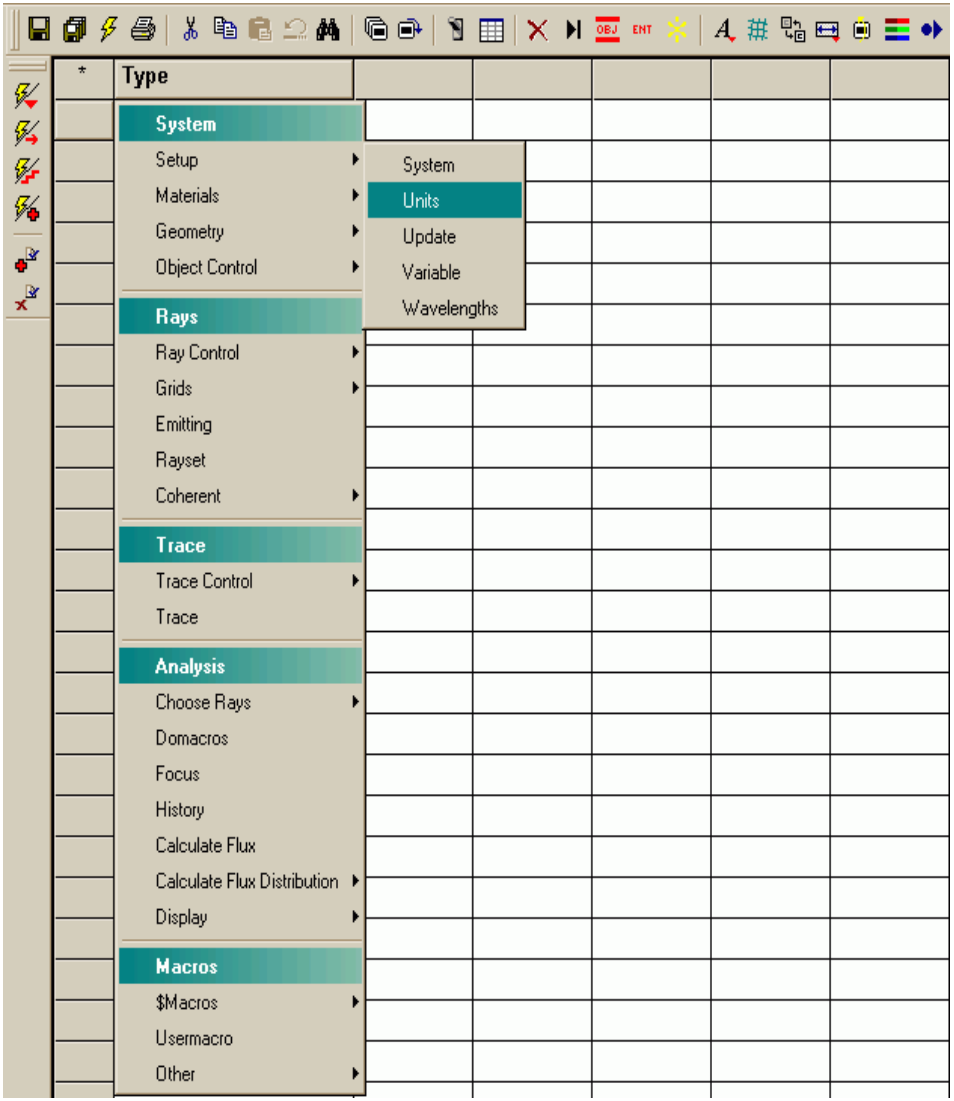


Figure 3.4 All commands available within the Builder can be found in one system of menus. You can access this menu by double-clicking any cell in the first column, labeled **Type**. For quick access to commonly used Builder commands, you can "tear-off" (detach) a commonly used section of the menu and dock or float it in the Builder by right-clicking any one of the colored menu bars.

To start our system definitions, double-click the cell in the upper left corner, and select **Setup> Units**. The first three cells (left to right) now have column headings. Double-click the cell under the second **Type** column, and a drop-down list appears. Select **Millimeters**, which becomes your system units for all subsequent definitions throughout this ASAP model. The completed line should appear as shown below (2).

*	Type	Type	Flux Label					
SYS	Units	MilliMeters						

Figure 3.5 First completed line in the Builder.

The third column, **Flux Label**, is a string that you type to let ASAP know what flux unit label you want to appear on your graphs during the analysis phase of the project. You might select units like Watts, or Lumens, but this is only a label. Since we are not initially defining sources in this model, we will leave that column blank for now. We have more to say about this later when we define sources and set their absolute flux (see Chapter 9, “Verifying Sources”).

See Chapter 3 Appendix, “Script 3-1” on page 55.

Note: When we have finished entering the information for the `UNITS` command, the ASAP Status Bar at the bottom of your screen (visible as long as you are not in full-screen mode) still lists the system units as **SU: <not defined>**. *The ASAP Kernel is not reading what you type as you enter commands into the Builder.* As we shall see in Chapter 5 “Running and Verifying Geometry”, you must “run” the Builder to actually create the system you are defining.

Wavelengths

Next, double-click the cell in the **Type** column on the next line in the Builder. This time, select **System> Setup> Wavelengths**. The second cell in this case has no headings at the top of the Builder. This cell is not used in order to retain alignment with other wavelength-related commands to follow.

Cells three through seven allow us to specify up to five wavelengths that interest us in this model. We will use only one wavelength for now. Set the column labeled as **Wave 1** to **550**. The last labeled column is the **Units** column. We are not required to express wavelengths in system units, since the wavelength of the light is generally much smaller than the scale of the system geometry. If you double-click this column, another drop-down menu appears. Select **Nanometers** for this example.

The completed **Wavelengths** line should appear as in 2.

*	Type		Wave 1	Wave 2	Wave 3	Wave 4	Wave 5	Units
SYS	Wavelength		550					Nanometer

Figure 3.6 Setting Wavelengths in the Builder.

See Chapter 3 Appendix, "Script 3-2" on page 55.

> ***Note***: The order in which you place the commands in the Builder does matter in many cases. When we run this file, it will go from the top down. Since **WAVELENGTH** actually sets a scale factor relative to the system units currently in place, the **UNITS** command must precede it. If the order is switched, an error message results when you run the file.

Media

Media information, along with knowledge of Snell's law, tells ASAP everything it needs to know to bend or refract rays reaching an object made of the specified material.

Next, we need to define some media. This definition gives us the opportunity to describe any glass types we plan to use in this model, in particular their indices of refraction. This information, along with knowledge of Snell's law, tells ASAP everything it needs to know to bend or refract rays reaching an object made of this material. Shortly, we will assign this and other properties to specific surfaces in our system. Two of the lenses in this Cooke Triplet are made of the Schott glass SK4, and one is Schott F15. ASAP has several built-in glass catalogs, including French, Hikari, Hoya, Ohara, Schott, and Sumita glass types. For this exercise, however, we define the glass type ourselves, in case you ever need to define a special media. We will demonstrate the use of the glass catalogs when we discuss Lens Entities in Chapter 4.

To define a medium, double-click another cell under the **Type**-column in the Builder, and select **System> Materials> Media**. The entry that ASAP creates allows us to name the medium, and assign up to five different indices of refraction, labeled **Index 1**, **Index 2**, and so forth.

> ***Note***: The default column width is too narrow to show the complete heading in this case. If you hover your cursor over the heading, a pop-up window shows you the complete label. If you prefer, you can change the width of the entire spreadsheet column by left-clicking and dragging the divider between cells in the header line. You may have to "sneak up" on the divider from below to keep the pop-up window from interfering with this process.

These five indices correspond to the five wavelengths above in the **WAVELENGTH** command. This correspondence allows us to model dispersive media—glasses that change index as a function of wavelength. When we begin defining sources, the rays can have any wavelength within this total range. ASAP interpolates to

obtain the appropriate index for a wavelength that lies between two specified values in this table. While the Builder allows only up to five wavelengths in the dispersion table, the limit increases to 25 when you use the command language version of the **WAVELENGTH** command.

Name your first medium, **SK4**, and give it an index of refraction of **1.613** in **Index 1**. The name can be up to 16 characters long. Later, when we assign this medium to various objects in our model, we will refer to it by this name.

Since the Cooke Triplet uses two glass types, create a second **MEDIA** command on the next line. Name the new medium **F15**, and set **Index 1** to **1.606**. When you are finished, these last two lines should appear as shown in 2.

*	Type	Name	Cmd	Media #	**Index 1**	Index 2
CMD	Media	SK4	Media		1.613	
CMD	Media	F15	Media		1.606	

Figure 3.7 Setting the index of refraction in the **Builder**.

See Chapter 3 Appendix, "Script 3-3" on page 55.

Coatings

The refractive index of a material alone does not tell ASAP everything it needs to know when a ray arrives at an object. In general, a bare glass surface refracts (transmits) most of the light, but some is also reflected. When an anti-reflection coating has been evaporated onto the surface, the performance is quite different. When a reflecting coating, like a thin layer of silver or aluminum, is deposited on the surface, virtually all of the light may be reflected.

ASAP allows us to specify coatings in a variety of ways. The program is quite capable of computing reflection and transmission coefficients as a function of angle of incidence, including phase shifts when doing wave optics. For now, we are content with simple, "ideal" coatings that transmit and reflect a constant amount of flux, independent of angle of incidence of the ray.

We specify the particular behavior of an ideal coating with the **COATING PROPERTIES** command. Somewhere below the last **MEDIA** command, double-click in a **Type**-column cell and select **System> Materials> Coatings> Properties**. Name the coating **Transmit**. For each coating definition, we are permitted to enter up to five reflection and transmission coefficient pairs, one for each of the wavelengths specified above in the **WAVELENGTH** command. Since we have only one wavelength, we need to fill in only one of these pairs. Set **Reflect 1** to **0** and **Transmit 1** to **1**. This causes 100% of the flux of each arriving ray to be transmitted, and none reflected—a truly ideal coating. We can leave the other cells blank.

Note: We could have specified a non-zero value for both reflection and transmission. This would cause any ray arriving at such a coating to split into two rays: one reflected and one transmitted. We could make an ideal beam splitter by setting both **Reflect** and **Transmit** to 0.5, for example. ASAP does not require us to conserve energy. We could have had some optical energy "disappear" at this coating, turning into heat or some other form of energy that is not relevant to our analysis.

Next, define a second coating named **Absorb**, with **Reflect 1** and **Transmit 1** both set to **0**. ASAP treats this "zero-zero" coating as a special case. It does not, as you might expect, cause all the ray flux to vanish from the system. Instead, the ray with all its flux simply stops on such a coating with all its flux intact. We will use this any time we want to collect rays for analysis on some particular surface. The `COATING PROPERTIES` definitions should appear as shown in 2

*	Type	Name	Coating Number	Cmd	Angle of Incidence	Reflect 1	Transmit 1
CMD	Coating	Transmit		Properties		0	1
CMD	Coating	Absorb		Properties		0	0

Figure 3.8 Setting coating properties in the **Builder**.

See Chapter 3 Appendix, "Script 3-4" on page 56.

Save your work!

If you have not already done so, this is a good time to save your work. You can do this with the **Save** button on the Builder, or from **File> Save** (while the Builder has focus).

Summary

In this chapter, we have discussed the following new commands:

ASAP Commands	Menu	Description
UNITS	Builder: **System> Setup> Units**	UNITS declares the system of units for subsequent definitions in the project.
WAVELENGTHS	Builder: **System> Setup> Wavelengths**	WAVELENGTHS declares one or more wavelengths corresponding to the interface information that follows in the MEDIA and COATING PROPERTIES commands.
MEDIA	Builder: **System> Materials> Media**	Defines media (glass types) in terms of their index of refraction.
COATING PROPERTIES	Builder: **System> Materials> Coatings**	Defines the behavior of the surface in terms of the fraction of the flux that reflects and transmits.

This section completes the “preliminaries” necessary for defining a new ASAP system. We have completed the first two of the six steps outlined at the beginning of this chapter:

1. √ Define system settings:
   - √ Units,
   - √ Wavelengths.
2. √ Define optical materials and coatings:
   - √ Refractive indices,
   - √ Reflection and Transmission coefficients.

   The remaining steps to complete are:
3. Define a geometrical shape for each object.
4. Assign optical properties to each object.
5. Shift and rotate each object to the desired location and orientation.
6. Graphically verify the shape, location, and orientation of each object as it is constructed.

The completed Builder, up to this point, is shown in 2.

*	Type	Name	Coating Nu	Cmd	Angle of In	Reflect 1	Transmit 1	Reflect 2
SYS	Units	Millimeters						
SYS	Wavelengths		550					Nanometer
CMD	Media	SK4	Media		1.613			
CMD	Media	F15	Media		1.606			
CMD	Coating	Transmit		Properties		0	1	
CMD	Coating	Absorb		Properties		0	0	

Figure 3.9 Preliminary declarations and definitions necessary for our Cooke Triplet model.

We can insert more media and coatings any time during the process. The only rule is that they must be defined before they are referenced.

Beyond setting up a system of units and a wavelength, we have built what amounts to a database of materials and coatings. These latter items are now available to you as you build the optical (and later mechanical) elements of a system. While it appears that you are required to "anticipate" such needs in advance, this should not be a concern. We can insert more media and coatings any time during the process. The only rule is that they must be defined before they are referenced.

Note that we have added some blank lines in our version of the basic definitions, shown in 2. This is just a matter of style. ASAP ignores these lines. You may find that adding lines is convenient as your Builder file grows, since it makes it easier for you to quickly scan the spreadsheet and locate logical groups of commands.

You have also probably noticed by now that the headings at the top of the Builder change to correspond to the line in which you are currently working. This dynamic adjustment is usually a great help, but sometimes causes confusion. When you want to see the headings specific to a given command, be sure to click any cell within that line in the Builder to activates them.

APPENDIX

3A

ASAP SCRIPTS FOR CHAPTER 3

The following ASAP scripts are referenced in “Builder and Geometrical Preliminaries”.

Script 3-1

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY THE BUILDER.
!!SEE THE REFERENCE MANUAL FOR OTHER SYSTEM UNITS CHOICES.
UNITS MM
```

Script 3-2

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY THE BUILDER.
!!SEE THE REFERENCE MANUAL FOR OTHER WAVELENGTH UNITS CHOICES.
WAVELENGTHS 550 NANOMETERS
```

Script 3-3

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY THE BUILDER.
MEDIA; 1.613  'SK4'
MEDIA; 1.606  'F15'

!!THIS SCRIPT IS REWRITTEN HERE IN THE FOLLOWING PREFERRED
!!MANNER IN ORDER TO IMPROVE READABILITY.
!!THE COMMENTS ARE ADDED FOR EXPLANATION.
MEDIA
  1.613  'SK4'  !!A GLASS MEDIUM SK4 HAS AN INDEX OF REFRACTION = 1.613
  1.606  'F15'  !!A GLASS MEDIUM F15 HAS AN INDEX OF REFRACTION = 1.606
```

Script 3-4

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY THE BUILDER            .
COATING PROPERTIES; 0 1  'TRANSMIT'
COATING PROPERTIES; 0 0  'ABSORB'

!!ONCE AGAIN THIS SCRIPT IS WRITTEN HERE IN THE FOLLOWING
!!PREFERRED MANNER IN ORDER TO IMPROVE READABILITY.
!!COMMENTS ARE ADDED AFTER THE COATING DEFINITIONS.
COATING PROPERTIES
  0 1  'TRANSMIT'   !!REFLECTIVITY = 0%; TRANSMISSIVITY = 100%
  0 0  'ABSORB'     !!REFLECTIVITY = 0%; TRANSMISSIVITY = 0%
```

CHAPTER

4

BUILDING AND PREVIEWING GEOMETRY

In this chapter, we continue to build the first element of the Cooke triplet, adding the actual geometry, surface by surface. While there are easier ways to build a lens in ASAP, this is the most general approach to defining ASAP geometry.

Once you master the concepts introduced here, you will know the basic principles behind assembling groups of surfaces, however complex, into any type of geometrical structure.

"Shell" Concept

We naturally think of a structure like a lens as a "solid". A lens, after all, is a volume of glass bounded by a front surface, a back surface, and a cylindrical edge. (See Figure 4.1a.)

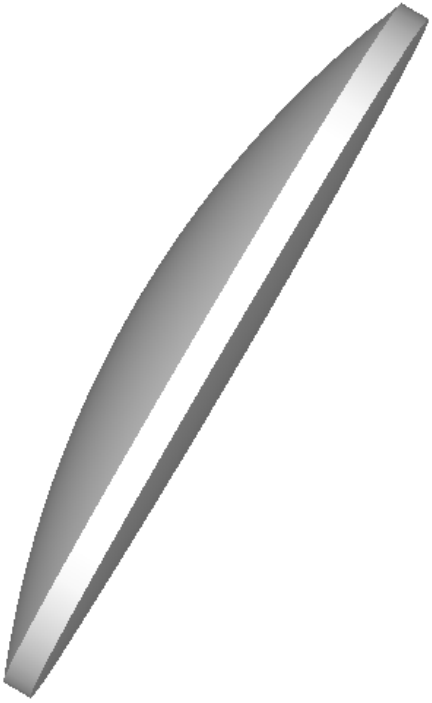


Figure 4.1a We think of a lens as a solid. It is a a single piece of glass, formed into the desired shape.

A lens is generally manufactured by starting with a solid block of glass and removing material to obtain the desired form. Many computer-aided design programs have recognized this, and moved to solid-based geometry definitions.

When we model a lens in ASAP, however, it is not treated as a solid. Instead ASAP views this structure as three independent surfaces that form elements of a shell around the volume. (See Figure 4.1b.)

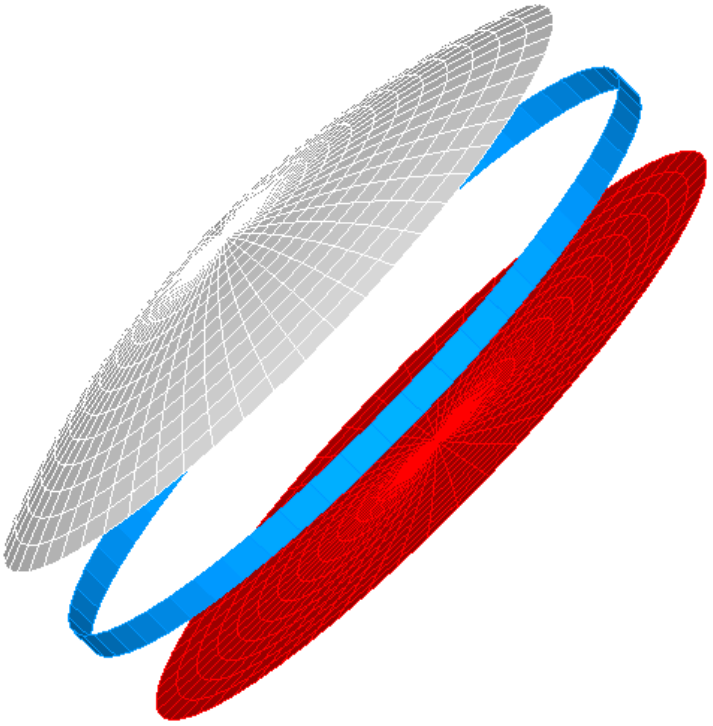


Figure 4.1b ASAP views the lens as three independent surfaces: front (white), back (red), and edge (blue). This view allows us to assign different optical properties to each element of the shell structure.

The three surfaces are shifted away from each other in the figure for clarity. Normally, they form a sealed volume. Several reasons exist for this approach:

1. In optics, the surface is usually where interesting things happen. The change in direction of a ray, due to refraction, happens at the surface. A ray proceeds through the rest of the volume unperturbed, changing direction again only at the exit surface. Scattering often occurs at a surface because it is not perfectly polished or cleaned. Special anti-reflection or beam-splitting coatings are placed on surfaces. When developing models of real optical devices, it is generally more logical and efficient to concentrate on surfaces rather than volumes, because so many optical phenomena happen at surfaces rather than in volumes.

Note: Some important volume-based optical phenomena exist, of course. For example, gradient index (GRIN) materials change index of refraction as a function of position within a volume. Volume scattering from suspended particles and volume absorption in a material are other examples of volume effects that

we might want to model. ASAP allows us to model all three of these optical phenomena as simple exceptions to the surface rule.

2 While our lens may be a solid, the individual surfaces often have different optical properties. The edge of a lens is usually not polished like the front and back. It is typically "edged" with a grinding wheel. It needs to be modeled with the scattering properties of ground glass. We also do not want to give up the flexibility of placing different optical properties on the front and the back surfaces, since the front may be dirty from exposure to the elements, while the back may be in a hermetically sealed environment. As a result, even if we were to model the lens as a solid, we would still need to reference its various surfaces independently.

3 Because ASAP rays do not trace sequentially, independent surfaces are again the more logical way to organize geometry. The rays typically proceed from surface to surface in no prescribed order. As we will see, when we interrupt a ray trace, we never find a ray "in transit" inside a volume. When we get to the analysis phase of an ASAP project, any surface can be a place where rays accumulate for detailed analysis.

Learning ASAP will accelerate if you temporarily abandon the idea of solids, and try to think of your geometry the way ASAP does: a series of independent surfaces.

For all these reasons and more, your learning of ASAP will accelerate if you temporarily abandon the idea of solids, and try to think of your geometry the way ASAP does: a series of independent surfaces. As you will see, it is not necessary to completely abandon the volume concept. ASAP provides tools to move between the world of solids and surfaces. The CAD translation process in ASAP automatically converts CAD-based solids into independent surfaces, if you choose to define your geometry with a CAD program. We will also see that objects can be grouped together when appropriate, and manipulated as a single, solid object when shifting and rotating. (See the `GROUP` command in "Grouping Geometry" on page 302 of Chapter 15).

"Surface" Entity Type

If your Cooke Triplet file is not already open in ASAP, recover the model we started in the previous chapter at this time. You can do this from the main menu bar using **File> Open**, selecting **Files of type: Builder Files (*.enx, *.enz)**, and browsing to the file location. It presumably is in your current working directory.

You can also accomplish this with the **Open** button on the ASAP Toolbar. The folder icon has the result as the dialog box described above. The arrow on the right produces a drop-down list of your most recently accessed files of all types.

We access all the geometry building tools available to us in the ASAP Builder from the pop-up menu (Figure 4.2).

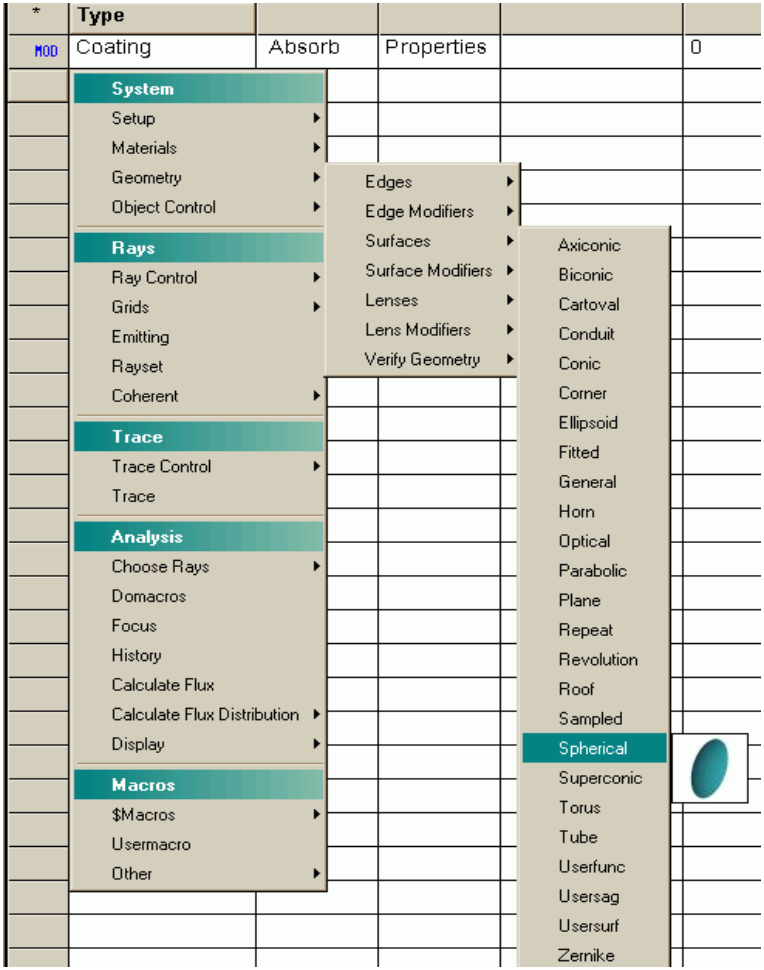


Figure 4.2 The ASAP geometry-building commands are in the second group of the popup menu. You access it by double-clicking a cell in the **Type** column.

So far, all our previous commands have come from the **System** group. We are now ready to move one group down to where the basic geometry definitions reside.

There are actually three different entity types listed in this group: **Lenses**, **Surfaces**, and **Edges**. Even though we are making a lens, we will resist using the `LENS` entities for the time being. We will be working exclusively with the `SURFACE` entities for this Cooke triplet model.

The differences and uses of all three entity types are explained in the sidebar, “Three ASAP Entity Types” on page 62.

Three ASAP Entity Types

ASAP objects are constructed from three different entity types:

***Surface Entities**—These are smooth and relatively simple shapes that are defined mathematically as implicit polynomials. In many cases, a ray intersection can be calculated by simply evaluating a mathematical expression. Originally, this was the only kind of geometry supported by ASAP, because it is ideally suited for ray-tracing applications.*

Examples: plane, sphere, paraboloid, tube, cone, axicon.

***Edge Entities**—Edges, also known as "Curves" in ASAP and elsewhere, are generally points in space connected with straight or curved lines. The curves are then swept to form surfaces (not to be confused with ASAP "Surface" entities). Mathematically, the curves are represented as parameterized Bezier polynomials. This geometry type is compatible with CAD programs, and was, in fact, originally added to ASAP just to allow the importing of geometry from CAD environments. Because these are defined in terms of parameterized equations, ray intersections have to be determined iteratively. While this slows down ray tracing somewhat, the ASAP Edge allows us to model structures that would be difficult or impossible to define in terms of implicit polynomials. ASAP provides tools to build this type of geometry within the program. Edge-based geometry can also be imported from CAD programs via the ASAP IGES translator, smartIGES™. Importing from CAD programs is discussed in the Technical Guide,* IGES Import. *Edge entities are discussed in Chapter Chapter 13, "Edges", with additional information found in the Technical Guide,* More about Edges.

Examples: faceted reflectors, spiral filaments, automotive headlamp lenses.

***Lens Entities**—The name "lens entity" is the source of some confusion since we can use these to make far more than just lenses. Further, these entities are not the only, or even the preferred way to define lenses in ASAP. They are standard conic surfaces used in classical imaging systems, and are defined using the language and conventions of lens designers and classical lens-design codes. Lens entities also have a sequential nature: rays are constrained to move through the surfaces of a lens entity in a prescribed order. This constraint can give them a considerable ray-trace speed advantage, especially when a lens entity consists of many surfaces. Because they are limited to conics (spheres, parabolas, hyperbolas, ellipses), they can be converted exactly to equivalent edge-based entities, since these have exact Bezier representations. This conversion is handy when exporting them back to a CAD program. Lens entities is the topic of Chapter 14, "Lens Entities", and is covered in more detail in the Technical Guide,* Lens Entities.

Examples: singlets, doublets, ideal lenses, simple telescopes.

Surfaces are often the most efficient entity type for ray-tracing since, in many cases, ASAP can calculate ray intersections simply by evaluating a mathematical expression.

The name given to this group of geometrical entities can be misleading, since all three types are used to produce surfaces or "shells", as discussed above. The reason for this name is largely historical. At one time, this group was the only type of geometry available in ASAP, so it was logical to name it "Surfaces". Surfaces are often the most efficient entity type for ray-tracing since, in many cases, ASAP can calculate ray intersections simply by evaluating a mathematical expression. This approach usually leads to the most rapid result for a given accuracy level. Because we sometimes need to trace many millions of rays through systems consisting of many surfaces, the speed advantage gained by using **SURFACE** entities can be significant.

Spherical Surface

The specifications for the first lens of the Cooke triplet are shown in Figure 4.3.

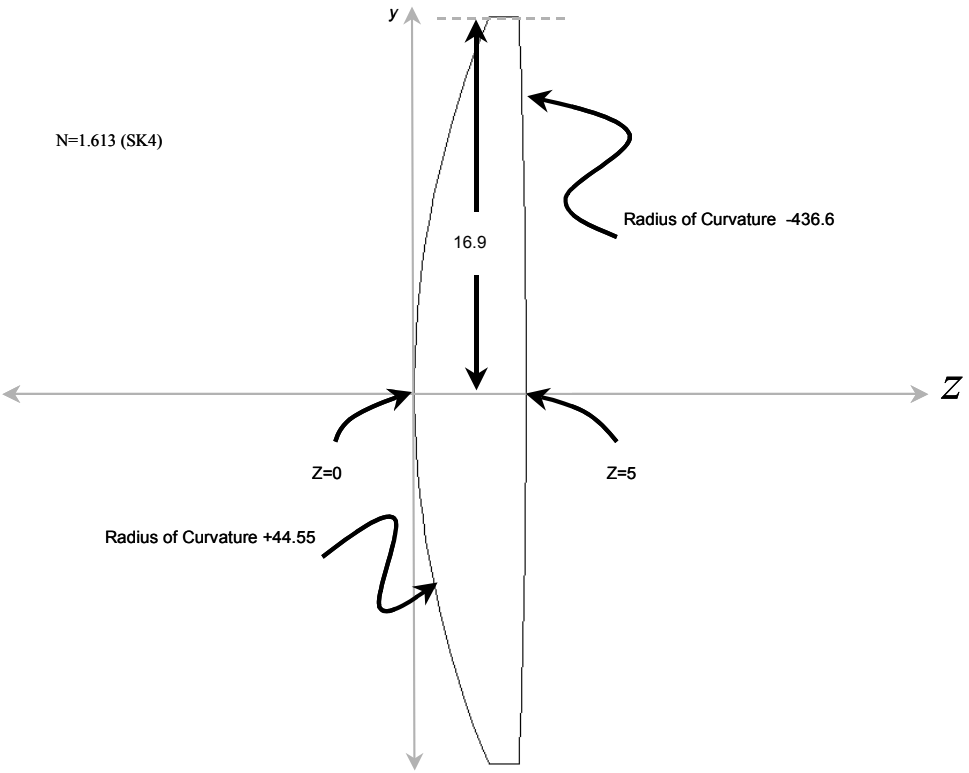


Figure 4.3 Specification for lens 1 of the Cooke triplet (units in millimeters)

We start by building the front surface. It is spherical, so we begin by selecting **System> Geometry> Surfaces> Spherical** from the pop-up menu. Give it the name **L1.FRONT**. This name can be typed in upper, lower, or mixed case; the ASAP Builder is, in fact, case insensitive. The "dot" in the name is important. It allows you to produce indented lists of assemblies and subassemblies reminiscent of a drawing tree. The number of levels permitted is constrained only by the 344-character limit on the overall object name.

The next entry is the **Axis** of the surface. This is the axis to which the spherical surface is perpendicular at its vertex. Since this surface type is axially symmetric, it is also the symmetry axis. By double-clicking this cell, you can see that you have six choices: the three Cartesian axes of our global coordinate system, or the negative-going axis of each. It is common practice (but by no means a requirement) in optical design to choose the Z axis, so choose **+Z**.

> ***Note:*** You get exactly the same surface whether you choose the positive or negative Z axis. The only difference between the two is generally just a matter of mathematical formality and convention. But in a few special cases—for example, if we were to turn this object into a random ray emitter (see Chapter 16, "Extended Sources"—the sign of the surface normal would determine the default direction of the rays covering the surface.

If we were modeling a lens that was not perpendicular to one of the global coordinate axes, we would apply rotations and shifts after creating it.

The **Location** entry is the position in a global coordinate system along the global Z axis of the vertex of this spherical surface. We arbitrarily place this object at the origin, $z=0$.

The **Radius of Curvature** entry applies to the spherical surface. By convention, a positive value implies that the center of curvature is in the $+z$ direction relative to the vertex, as shown in Figure 4.4.

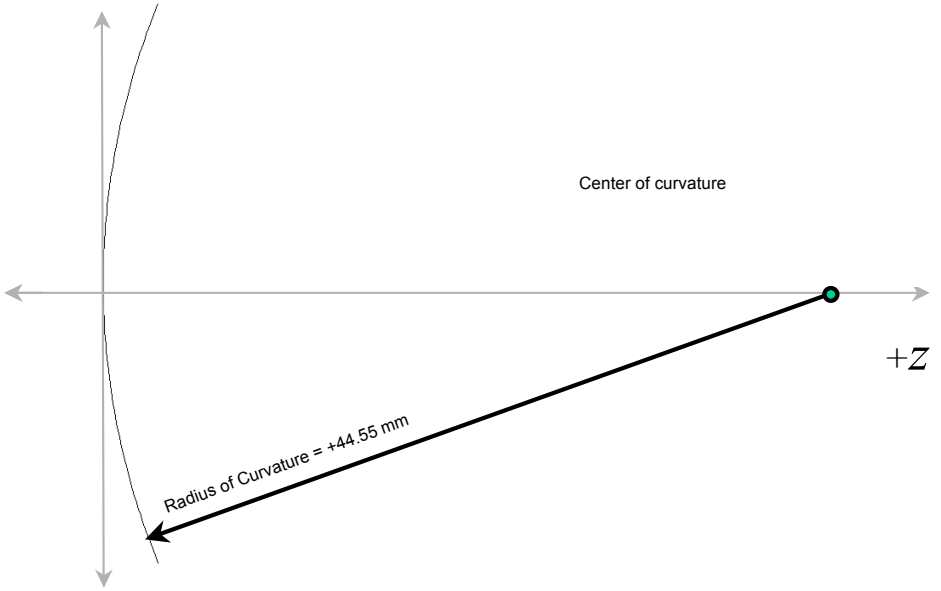


Figure 4.4 By convention, a surface has a positive radius of curvature when the center of curvature is in the positive direction (as shown on right), relative to the surface.

The **Aperture** entry describes the shape of the outside boundary of the surface. When you double-click this cell, you see that your choices are **Ellipse**, **Rectangle** and **Hexagonal**. We want a circular aperture, so choose **Ellipse**, which is a circle when both dimensions are the same.

Finally, the **Semiwidth X** and **Semiwidth Y** values are the dimensions of the ellipse or rectangle defined above. Note that these are half dimensions, as is the convention in most optical applications. Set these both to **16.9,** as specified in Figure 4.3, "Specification for lens 1 of the Cooke triplet (units in millimeters)". The result is a circular lens with a *diameter* of twice this value, or 33.8 mm. If your optical elements look disproportionately large, you probably forgot to divide by 2.

Virtually every geometrical entity in ASAP is specified in terms of semi- or half-aperture dimensions.

A common error for new ASAP users who are unfamiliar with optical terminology is to express these specifications in terms of an object's diameter or total "width", since this is what is normally specified in catalogs. Virtually every geometrical entity in ASAP is specified in terms of these semi- or half-aperture dimensions, however. Many of the exercises in this *Primer* give you aperture diameters in an attempt to drive this point home.

Three additional entries exist for the **Spherical** surface. The **Obs Ratio** is the so-called "obscuration ratio". It is a method of putting a hole in the center of the aperture. An **Obs Ratio** of 0.5 would place a circular hole with a half diameter of 8.45 mm. The **Offset X'** and **Offset Y'** values allow us to shift the entire spherical

surface (hole and all) by the amounts specified. None of this is necessary for **L1.FRONT**; we can leave all three entries blank since they are optional parameters.

> ***Note:*** Any Builder cells that are empty when you initially select the command from the menu are most likely optional parameters. They can remain blank.

When you are finished, this line in the Builder should appear as shown here:

*	Type	Name	Axis	Location	Radius of Curvature	Aperture	Semiwidth	Semiwidth
OBJ	Spherical	L1.FRONT	Z	0	44.55	Ellipse	16.9	16.9

Figure 4.5 Completed Spherical line in the **Builder**.

See Chapter 4 Appendix, "Script 4-1" on page 85.

Previewing in the BRO 3D Viewer

We can easily preview objects as we make them in the Builder. In fact, this is one of several features of the ASAP Builder. Once you have finished with the definition of the **Spherical** surface, try clicking the **Preview All** button on the Builder Toolbar.

You will notice some brief activity in the **Command Output** window before the **3D Viewer** window opens (Figure 4.6). You can maximize this window to fill the user task space, or press the **F11** key on our keyboard to fill the entire screen.

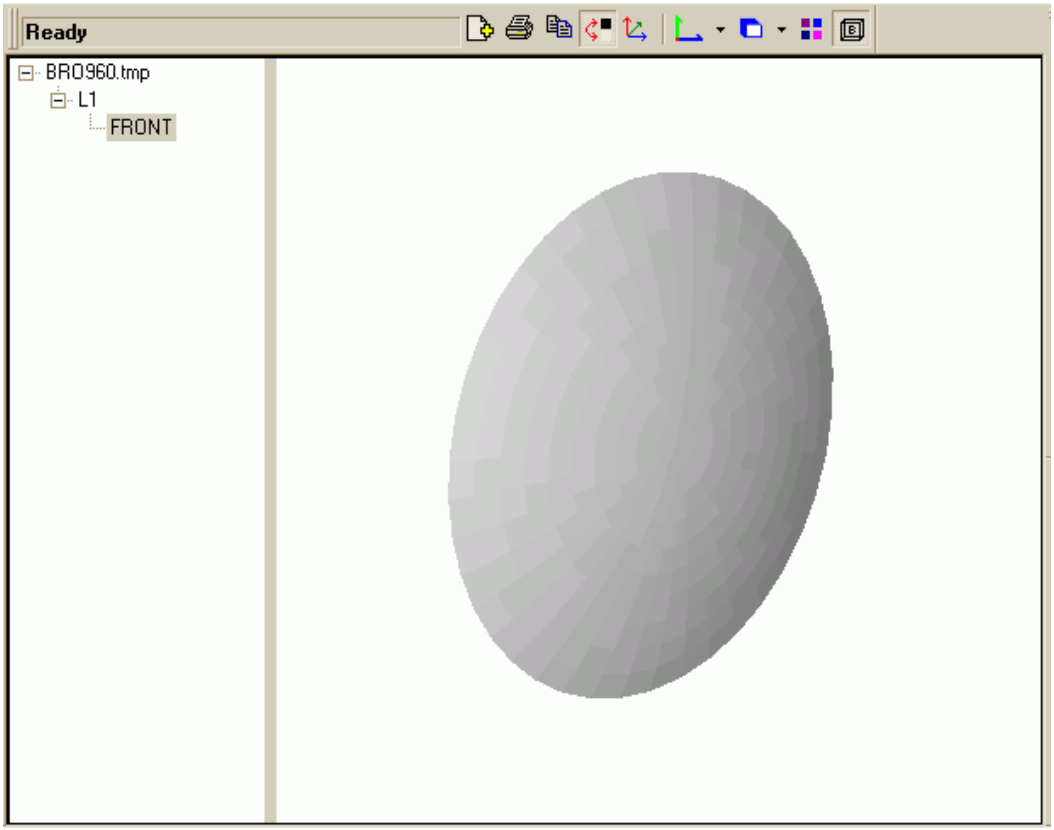


Figure 4.6 Initial 3D view of **L1.FRONT** with the tree view expanded

Start by looking in the white pane on the extreme left of the window. This pane is called the **Tree View**. The temporary file, **bro030.dat** contains the three-dimensional graphical information necessary to create this view. To reveal the beginnings of the drawing tree, click the plus (+) sign to the left of this file name, and then the + sign next to **L1**. Though a little simple now, this pane becomes increasingly interesting as we add objects to the system.

The pane on the right is called the **Object View**. ASAP made this view by approximating the actual spherical surface with a series of flat polygons, although it is probably not apparent in this case. This simplification of the geometry allows us to zoom and rotate the view rapidly and efficiently. Try holding down the right mouse button, and moving the mouse to rotate the view. You can shift the view by adding the **Shift** key while right-clicking and moving. You can also zoom in and out by pressing the **Ctrl** key instead of **Shift**. If you have a mouse wheel, you can also zoom the view with it. If you click the object in the **Object View**, the object's name and number appear in a ToolTip window,

and the object is highlighted in the **Tree View**. This technique becomes increasingly useful as you add objects to the view (Figure 4.7).

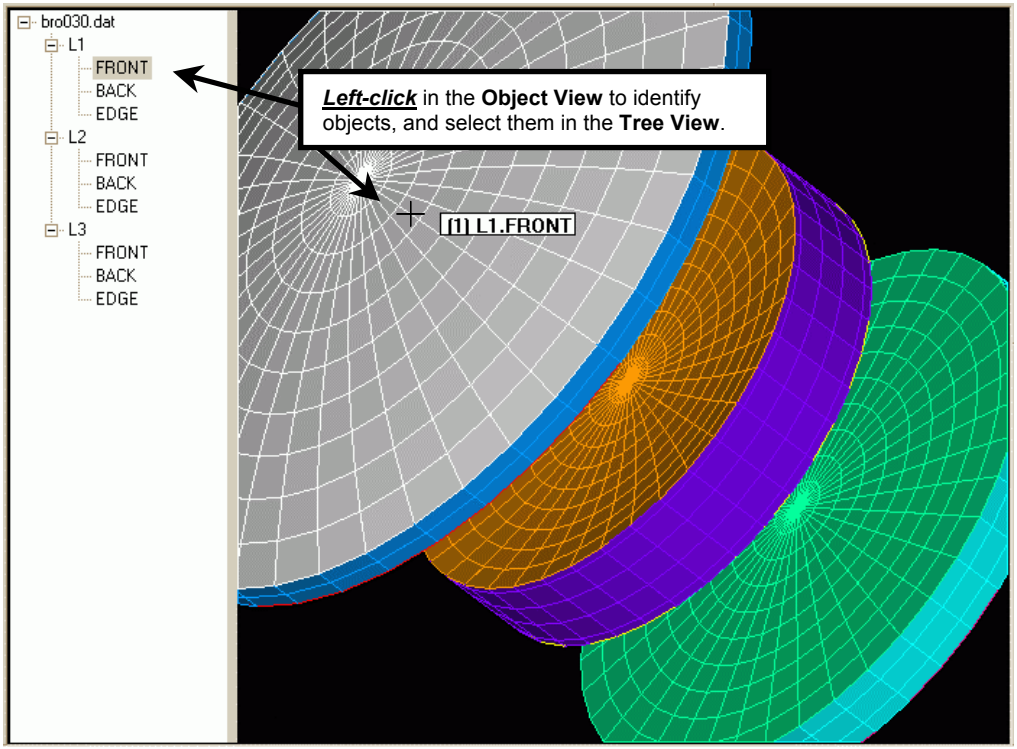


Figure 4.7 The ASAP **3D Viewer** can help identify objects and find them in the drawing tree. Place the mouse over the object of interest, and click. A ToolTip appears [**(1) L1.FRONT** in this case], including the ASAP assigned object and its complete name. The same object is also selected in the drawing tree.

Many viewing options are available in the BRO 3D Viewer. Two pop-up menus control most of these functions. One is the **3D Viewer** menu that appears on the ASAP menu bar, when the 3D Viewer window has focus (Figure 4.8a). Right-clicking at any level of the drawing tree accesses the other menu (Figure 4.8b). "Hot key" assignments are circled.

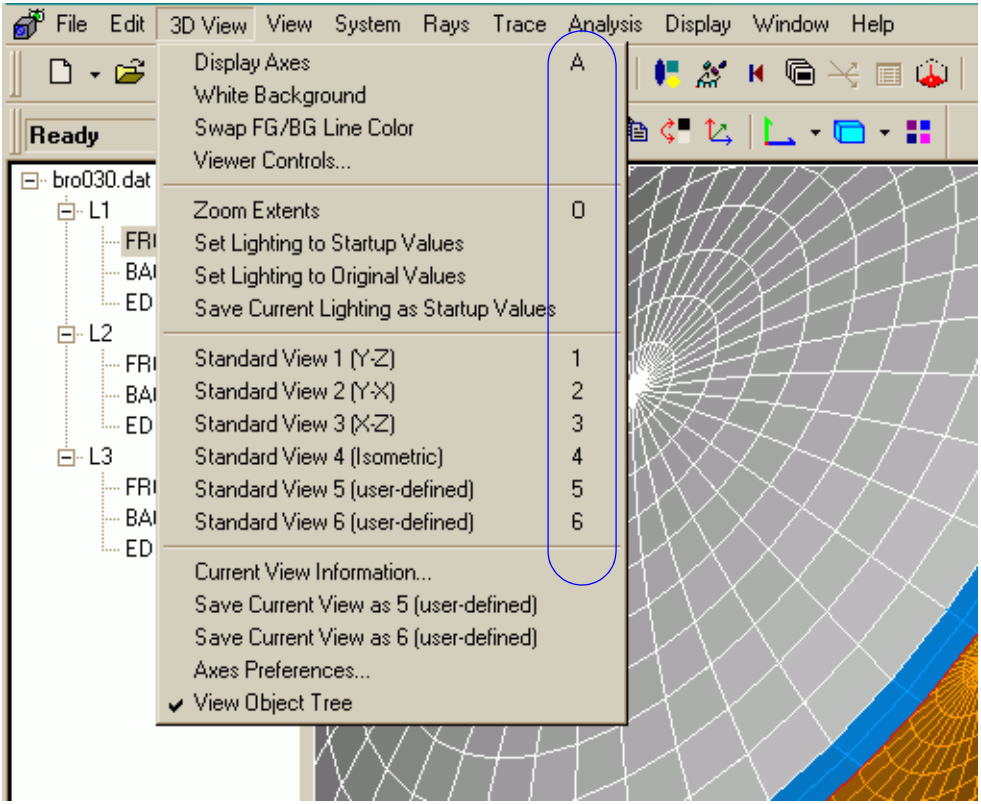


Figure 4.8a One set of menu commands controlling the **3D Viewer** is on the main menu bar, whenever the this view has focus. These items control general features of the view.

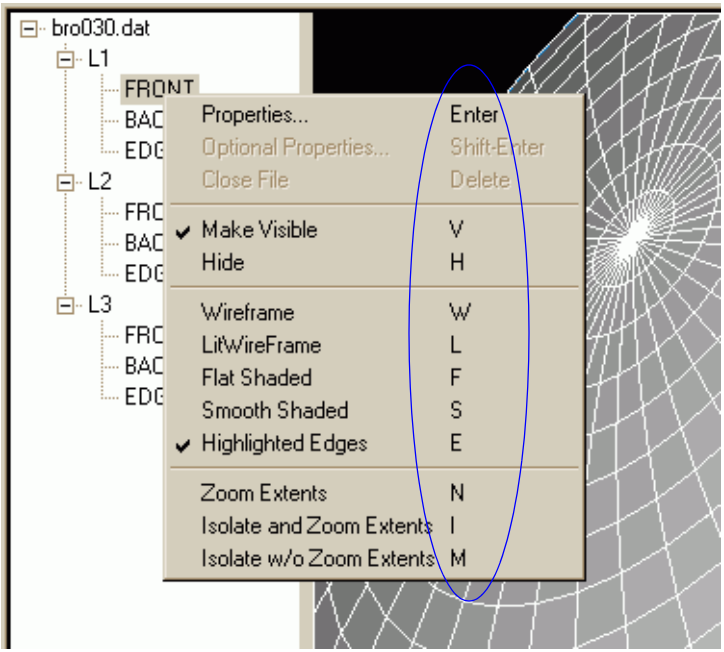


Figure 4.8b The second set of menus is accessed by right-clicking in any level of the drawing tree. These items control options that are specific to the selected level in the tree view.

In general, the main **3D Viewer** menu applies to the **Object View** window as a whole, while the pop-up menu (accessible from the drawing tree) applies only to the individual objects or assemblies selected. Try right-clicking the **Front** surface listed in the **Tree View** and selecting **Highlighted Edges** from the third group of options. The appearance of **L1.FRONT** changes from the default (**Smooth Shaded**) to a combination of **Wireframe** and **Flat Shaded**. Now you can see the individual flat polygons making up the actual view.

> ***Note:*** This approximation of the spherical surface is used for only this 3D view. During an actual ray trace, the ray intersections are calculated for the real mathematical version of the sphere.

You can experiment with the various drawing modes. Note that many of the menu commands also have “hot keys” associated with them. You may find it convenient to memorize the hot-key shortcuts for the commands you use frequently.

Many commonly used 3D Viewer commands also have buttons. Table 4.2 summarizes these. Whether you use menus, hot keys, or buttons is a matter of style, preference and, eventually, habit.

Table 4.2 Frequently used menu commands with associated buttons.

Add files.

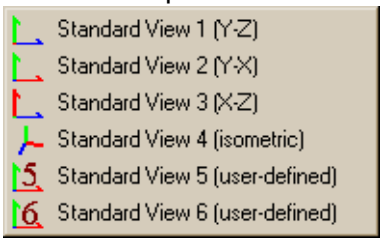
Print this file.

Copy to Clipboard.

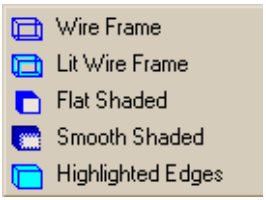
Toggle (switches) background between white and black.

Rotational axis. Toggles (switches) axes on and off. Axes can also be toggled on and off with the **A** key on your keyboard. Note the color order of RGB corresponding to XYZ axes.

Standard/user-defined view. Go to selected view angle. The triangular arrow on the button opens a menu of other views.



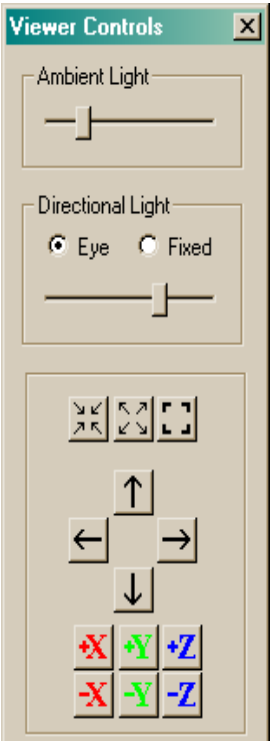
View mode. Change the appearance of objects or assemblies. The triangular arrow on the button can be used to select a different mode.



(continued)

Viewer Controls.

Change lighting, zoom, shift, or rotate.m



Bounding Box

Display bounding box when performing a rotate, shift, or zoom with the mouse.

Perhaps the best way to learn more about the 3D Viewer is by exploring and experimenting on your own. When the 3D Viewer has focus, you can consult the on-line Help from **Help> 3D Viewer Help**, for more information and guidance. When you have finished, press **F11** again to return to the normal ASAP landscape (if you have entered full-screen mode). The 3D view does not automatically update as new objects are added to the system, so close this **3D Viewer** window. You can click the **Preview All** button at any time to inspect new objects as you place them in the Builder file.

Interface Command

If a ray arrives at a surface in your system, what should it do? Should it reflect? Refract? Stop?

Once we have defined a new object, the very next step should always be to assign optical properties to it. Adding the **INTERFACE** command directly below the object definition in the Builder accomplishes this. You will find it in the Builder's pop-up menu under **Object Control> Object Modifiers> Interface**. Three cells need information that you supply. The first is **Coating**. ASAP uses its predefined **Bare** coating by default, which would be the right choice for an uncoated piece of glass. This coating, however, would cause ray splitting since some flux would be reflected and some transmitted, according to Fresnel's equations. To keep things simple for now, we substitute the **Transmit** coating. Double-click the **Coating** cell to edit, and type **Transmit** in place of **Bare**. This name must be spelled exactly as you did in the **Name** cell of the **COATING PROPERTIES** command.

You also need to supply two media on the **INTERFACE** line. Most surfaces in your model represent an "interface" (boundary) between two media. This surface, **L1.FRONT**, has air on one side, and SK4 on the other, so enter these as **Air** and **SK4** in the last two cells labeled **Media 1** and **Media 2**. The order does not matter. See sidebar, "How does ASAP know?" on page 74.

You already defined **SK4** on the media line, and **Air** (index of refraction equal to exactly 1) is predefined within ASAP. Because we have identified this surface as transmitting, ASAP knows to apply Snell's law at the interface to bend an arriving ray by the appropriate amount. When you have finished, the entry should appear as shown in Figure 4.9.

*	Type	Option	Cmd	Coating	Media 1	Media 2	List
MOD	Interface	Coating	COATING	Transmit	Air	SK4	

Figure 4.9 Interface entry in the **Builder**.

See Chapter 4 Appendix,"Script 4-2" on page 85.

How does ASAP know?

You do not need to concern yourself about the order of the media listed on the interface command. If you think about it, you will realize that this is due to a direct consequence of the realistic or "non-sequential" way that ASAP traces rays. We are not constrained in the way sequential ray tracing codes are. There is no "left-to-right" presumption. So, how does a ray know whether it is entering or leaving the new medium?

The answer is simple. The ray keeps track of what medium it is in at all times during a ray trace. This tracking is part of a large table of information about the ray that is updated as it moves through your system.

When a new ray is created, it is presumed to be in "Air" (unless we specify otherwise with the **IMMERSE** *command). When it arrives at* **L1.FRONT** *during a ray trace, it checks the interface assigned to that surface and finds that its choices are Air and SK4. One of these must match the ray's current medium (air, in our case). Hence, the ray must be passing into the other medium, SK4. It updates its table of parameters to specify that it is now in SK4, and moves on. This approach always works as long as we have specified our interfaces correctly, and have not allowed any "leaks" in the structures we have created.*

If the ray does not find its current medium to be one of the choices when it arrives at a surface, it stops and issues a warning, called a "wrong-side error". Other rays continue tracing. This and other "ray cessation" warnings are discussed in Chapter 17, "Ray Cessation—Stopping Rays Prematurely".

Lens Back

Since the next surface, **L1.BACK**, is similar enough to **L1.FRONT,** we can save some time by copying both the **Spherical** and the **Interface** lines, pasting them into new lines, and making the few necessary changes.

> ***Note:*** **Copy** and **Paste** are done by any of the usual Windows methods. First, select the line where **L1.FRONT** is defined by clicking the small **OBJ** label to the left of the first cell. Do the same for the **Interface** line while pressing the **Ctrl** key. Dragging and holding the mouse button accomplishes the same thing. This should leave both lines highlighted and ready to copy to the **Clipboard**. You can now use the keyboard (**Ctrl+C**), the menus (**Edit> Copy**), or the **Copy** button in the Builder window to copy these two lines to the **Clipboard**. Paste these commands in empty Builder lines further down.

After pasting the copies, change the name of the second **SPHERICAL** entry to **L1.BACK**. Change the **Location** to 5, and the **Radius** (of curvature) to –436.6. The negative curvature value indicates that, this time, the center of curvature is in the negative *z* direction (left, in Figure 4.3, “Specification for lens 1 of the Cooke triplet (units in millimeters)” on page 63).

The interface command requires no changes at all. It can remain the same as the one used for **L1.FRONT**. (The order of the media is NOT important!) When you are finished, the two new lines should look like Figure 4.10.

*	Type	Name	Axis	Location	Radius of Curvature	Aperture	Semiwidth X	Semiwidth Y
OBJ	Spherical	L1.BACK	Z	5.0	-436.6	Ellipse	16.9	16.9
MOD	Interface	Coating	COATING	Transmit	Air	SK4		

Figure 4.10 Two new lines for lens back in the **Builder**

See Chapter 4 Appendix “Script 4-3” on page 86.

Tube Surface

The third part of the lens is the cylindrical outer edge that connects the lens front and back. The tube surface is the right choice for this piece of geometry. Begin the definition by selecting **Geometry> Surfaces> Tube** from the Builder menu. We can use this command to make a general “tube” of the type shown in Figure 4.11.

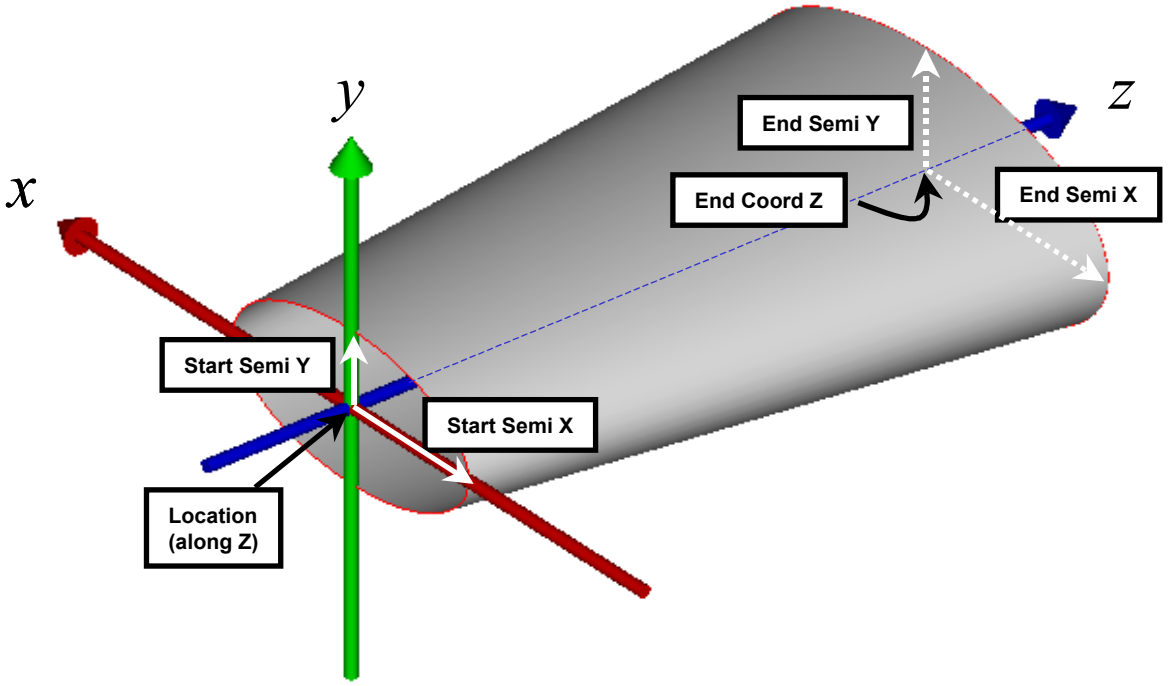


Figure 4.11 The ASAP **TUBE** is general, allowing us to define tapered tubes with elliptical cross sections.

Our version is much simpler, however, because we need only a uniform circular cylinder. Still, we are confronted with 11 parameters that we can change. Give the object the name **L1.EDGE**. The **Axis** is again **Z**. All four starting and ending semi-diameters are **16.9**. The **Start Q Factor** and the **End Q Factor** can stay at their default value of zero, and the **Cylinder Option** column should be blank.

> ***Note:*** If you are curious about these additional entries, try pressing the **F1** key while you are on this line of the Builder. The on-line Help explains their use.

Now, only two cells require attention: **Location** and **End Coord Z**. Ideally, these should be the starting and ending z coordinate of the tube, but these values are not given in the lens specification. Of course, we could calculate these points from geometrical principles (using the sag equation for a sphere), but that is not necessary. ASAP is able to do this calculation for us. Our strategy is to make the tube just a little longer than it needs to be, and let ASAP trim away the unwanted parts using the `BOUNDS` command. Set **Location** to **0**, and **End Coord Z** to **5.** These correspond to the vertex positions of the front and back spherical surfaces, which were convenient in this case. Any rough guess would have been acceptable, however. For the sake of ray tracing efficiency, and to avoid

problems in the 3D graphics, try to keep the guess close (within a few millimeters at each end). When you finish, the tube command should appear as shown in Figure 4.12, and the **Preview** should appear as shown in Figure 4.13a.

*	Type	Name	Option	Axis	Location	Start Semi X	Start Semi Y	End Coord Z	End Semi X	End Semi Y	Cylinder Opt
OBJ	Tube	L1.EDGE	Axis	Z	0.0	16.9	16.9	5	16.9	16.9	

Figure 4.12 Tube entry in the Builder

See Chapter 4 Appendix, "Script 4-4" on page 86.

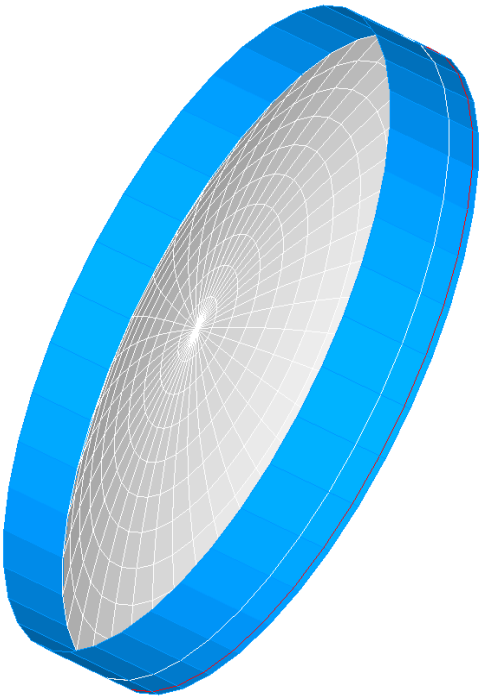


Figure 4.13a Previewing the results in the **3D Viewer**: Since the lens specification did not include the starting and ending coordinates of the edge, we have initially made it slightly too large on both ends. The next step is to use the `BOUNDS` object modifier to trim away the unwanted portions.

Bounding the Tube and Adding the Interface

The `BOUNDS` command in ASAP is a powerful tool that we can use to trim away portions of surfaces that we do not want. To do this ASAP needs to know two things:

1. Which geometrical entities do the trimming? These must be pieces of geometry that were defined prior to our tube definition. This is why we opted to build both the front and back surfaces before defining the tube.
2. Once ASAP determines the intersection of the object and the trimmers, which side of the trimming boundary should be kept, and which side discarded?

In this case, we want to trim away those parts of the tube that extend beyond the bounds of the front and back surface of the lens (Figure 4.13b).

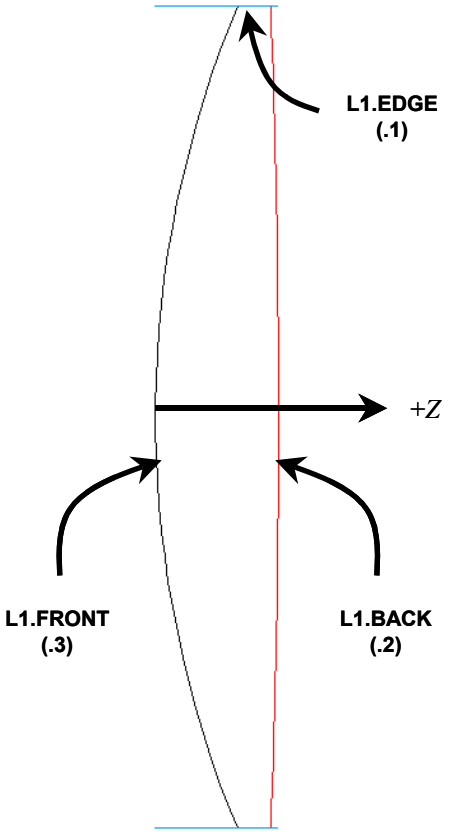
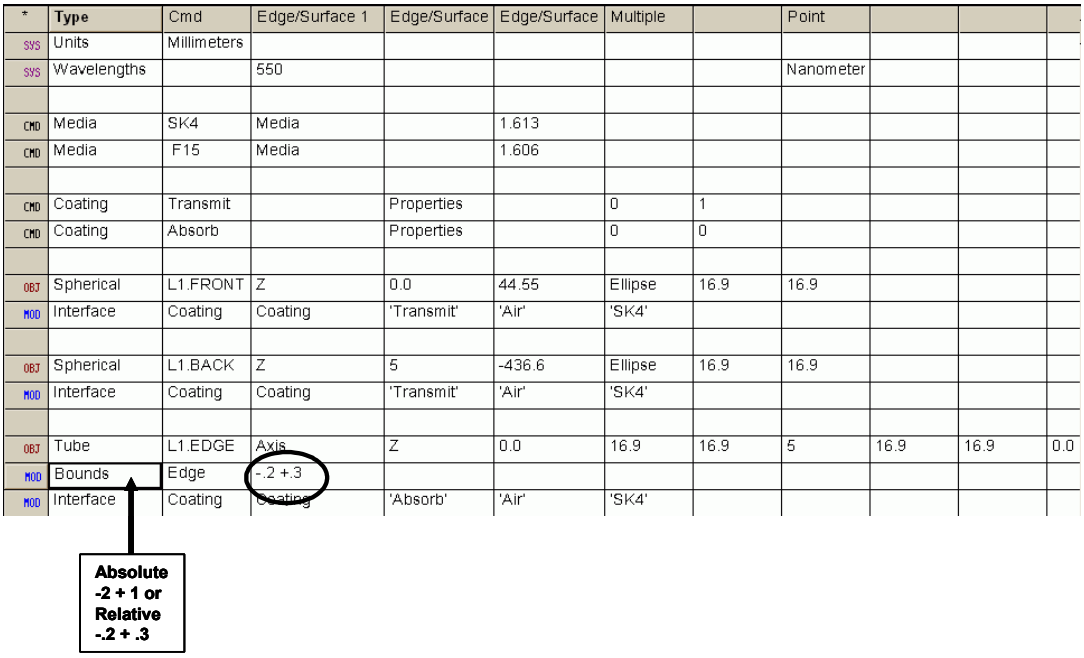


Figure 4.13b Defining the BOUNDS command. We need to give this command the part or parts of the object to keep. We want that portion of the tube (**L1.EDGE**) that is on the positive (+Z) side of **L1.FRONT** (+.3) and on the negative side of **L1.BACK** (-.2). ASAP keeps any part of the tube that meets *both* these conditions.

On the Builder line directly under the TUBE command, select **Object Control> Object Modifiers> Bounds**. *The cell labeled **EdgeSurface 1** is where we answer both questions posed above.*

We specify the geometry that does the trimming by number. Two choices exist: absolute or relative referencing. In absolute referencing, you count from the top of your file downward, noting each piece of geometry in top-down order (Figure 4.14).

With relative referencing, you count from your current position (the **BOUNDS** command) upward. This choice is almost always the better one, since your bounding geometry is generally near by. We will use this method here. Start at the **Bounds** line and proceed upward, counting only pieces of geometry. Specify relative referencing by placing a "dot" (the period character) in front of the integer.



*	Type	Cmd	Edge/Surface 1	Edge/Surface	Edge/Surface	Multiple		Point			
SYS	Units	Millimeters									
SYS	Wavelengths		550					Nanometer			
CMD	Media	SK4	Media		1.613						
CMD	Media	F15	Media		1.606						
CMD	Coating	Transmit		Properties		0	1				
CMD	Coating	Absorb		Properties		0	0				
OBJ	Spherical	L1.FRONT	Z	0.0	44.55	Ellipse	16.9	16.9			
MOD	Interface	Coating	Coating	'Transmit'	'Air'	'SK4'					
OBJ	Spherical	L1.BACK	Z	5	-436.6	Ellipse	16.9	16.9			
MOD	Interface	Coating	Coating	'Transmit'	'Air'	'SK4'					
OBJ	Tube	L1.EDGE	Axis	Z	0.0	16.9	16.9	5	16.9	16.9	0.0
MOD	Bounds	Edge	-.2 +.3								
MOD	Interface	Coating	Coating	'Absorb'	'Air'	'SK4'					

Figure 4.14 Methods for referencing commands. When an ASAP command references other commands in a file, we have the option to specify them in terms of the absolute position (counting from the top down), or the relative position (counting from the bottom up). Generally, it is more convenient (and safer) to use the relative referencing method. We accomplish this by placing a "dot" (.) in front of the integer. ASAP can tell from the context that the dot is not intended to represent a decimal point. The **BOUNDS** command counts only geometry definitions—other commands are not counted.

Counting from the bottom up, the first trimmer that we come to is **L1.BACK**, which is **.2**. We want to keep what is on the negative side of this (-*z* direction), so enter **-.2** with no spaces. The next trimmer is **L1.FRONT**, which is **.3**. We want to keep what is on the positive side (+*z* direction), so enter **+.3** in the same cell. The plus sign is optional—it is the default. Be sure to leave at least one space between the two numbers, otherwise ASAP automatically adds the two values together! Figure 4.14 also shows the completed **Bounds** line.

See Chapter 4 Appendix, "Script 4-5" on page 86.

Note: While some new ASAP users find this bounding process logical and easy, others find it to be one of the most difficult concepts that we introduce. We offer one useful suggestion: if you are not sure which sign to use, just pick one. If that does not work, pick the other one. You will always get it in two tries or less. For more information about the **BOUNDS** command, see the Technical Guide, *Arrays and Bounds*.

The last step in defining the tube is to add the **INTERFACE** specification, as we did with both spherical surfaces. Recall that this was also an object modifier, found on **Object Control> Object Modifiers> Interface**. Since most lenses do not have a polished edge, we use the **Absorb** coating in this case. Recall, too, that any ray that reaches this place stops on this type of coating. Until we are ready to define scattering surfaces (the appropriate modification for a ground surface), we will settle for this solution.

Figure 4.15a shows the complete Builder file for the first element of the Cooke triplet.

*	Type										
SYS	Units	Millimeters									
SYS	Wavelengths		550					Nanometer			
CMD	Media	SK4	Media		1.613						
CMD	Media	F15	Media		1.606						
CMD	Coating	Transmit		Properties		0	1				
CMD	Coating	Absorb		Properties		0	0				
OBJ	Spherical	L1.FRONT	Z	0.0	44.55	Ellipse	16.9	16.9			
MOD	Interface	Coating	Coating	'Transmit'	'Air'	'SK4'					
OBJ	Spherical	L1.BACK	Z	5	-436.6	Ellipse	16.9	16.9			
MOD	Interface	Coating	Coating	'Transmit'	'Air'	'SK4'					
OBJ	Tube	L1.EDGE	Axis	Z	0.0	16.9	16.9	5	16.9	16.9	0.0
MOD	Bounds	Edge	-.2 +.3								
MOD	Interface	Coating	Coating	'Absorb'	'Air'	'SK4'					

Figure 4.15a Builder file for element 1 of the Cooke triplet.

Your result, when previewed, should look like Figure 4.15b.

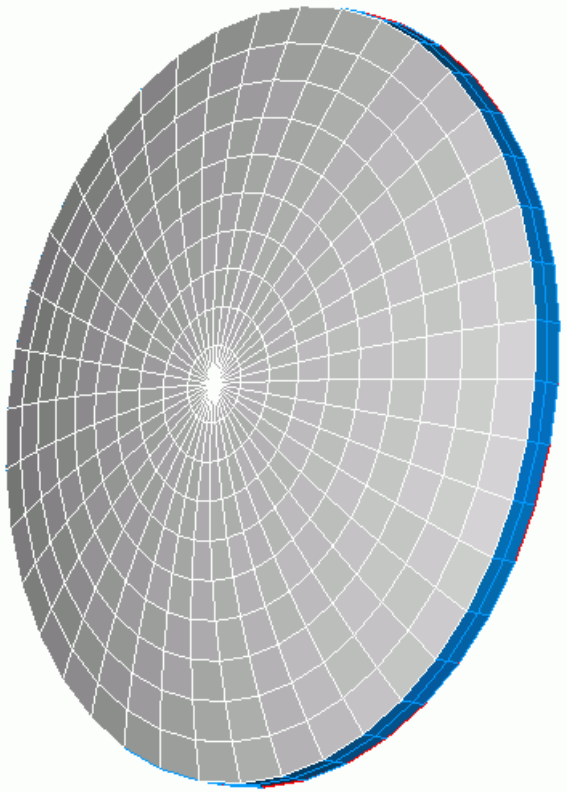


Figure 4.15b Preview of the completed lens 1 of the Cooke Triplet.

The next two lenses in the triplet are part of the exercise at the end of this chapter.

Summary

In this chapter, we have discussed the following new commands:

ASAP Commands	Builder Menu	Description
SURFACE; OPTICAL*	**System> Geometry> Surfaces> Spherical**	Creates a spherical surface.
SURFACE; TUBE	**System> Geometry> Surfaces> Tube**	Creates a tube.
INTERFACE	**Object Control> Object Modifiers> Interface**	Defines behavior of surface in terms of the fraction of the flux that reflects and transmits.
BOUNDS	**Object Control> Object Modifiers> Bounds**	Trims away unwanted pieces of objects.

*The surface generated from **Geometry> Surfaces> Spherical** is based on a more general ASAP command named OPTICAL. With this command, we can model any conic surface, including spheres, ellipses, parabolas and hyperbolas. The OPTICAL command is discussed in more detail in Chapter 6, “Cassegrain Telescope Model”.

We have made progress on the six geometry-building steps introduced at the beginning of Chapter 3, “Builder and Geometrical Preliminaries”. A quick review of this process (below) shows that we have accomplished all but one of these: Step 5. We did not need to perform shifting and rotating of objects on any of the objects in the Cooke triplet.

1. √ Define system settings:
   - √ Units,
   - √ Wavelengths.
2. √ Define optical materials and coatings:
   - √ Refractive indices,
   - √ Reflection and Transmission coefficients.
3. √ Define a geometrical shape for each object.
4. √ Assign optical properties to each object.
5. Shift and rotate each object to the desired location and orientation.
6. √ Graphically verify the shape, location, and orientation of each object as it is constructed.

Exercise 1: Completing the Cooke Triplet

1 Starting with the ASAP Builder file that you began in this chapter and the previous one (Figure 4.15a, “Builder file for element 1 of the Cooke triplet.” on page 80), complete the geometrical model of the Cooke triplet. The specification is outlined in Table 4.2 and illustrated in Figure 4.16, taken from Warren J. Smith, *Modern Lens Design*, page 130.

Table 4.2 Specification for a Cooke Triplet

Radius	Thickness*	Material	Index	Semi-Diameter
44.55	5.000	SK4	1.613	16.9
-436.60	10.310	Air	1.000	16.9
-38.61	1.600	F15	1.606	10.5
42.62	8.040	Air	1.000	10.5
250.97	5.000	SK4	1.613	12.1
-32.67	89.103	Air	1.000	12.1

*For an explanation of “Thickness”, see the first bullet under the section “Hints”

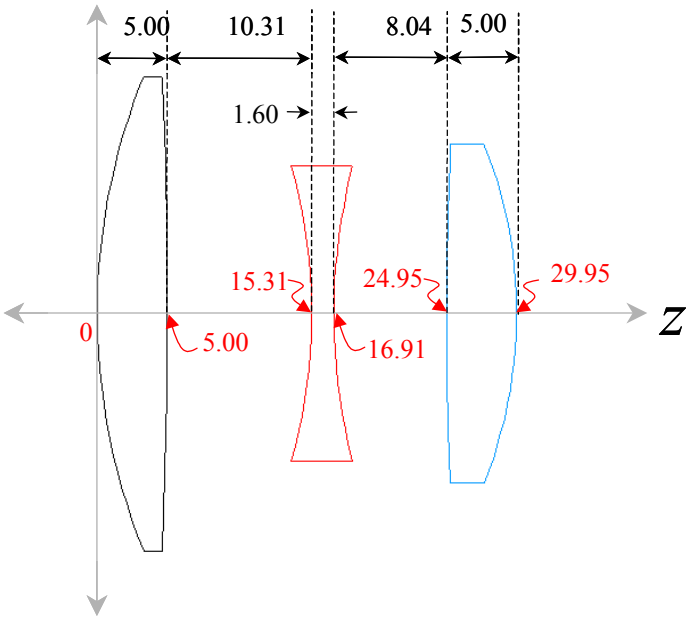


Figure 4.16 Illustrated Cooke Triplet. Red numbers along the z axis are the coordinates of the various surface vertices. Black numbers, near the top, are “thicknesses” commonly used in lens specifications.

2 Verify the model using the **Preview** tool in the Builder. Confirm that all nine surfaces are present.

3 Within the 3D Viewer, turn on the axes (press the **A** key), and change the drawing mode of all objects to **Highlighted Edges**.

4 Click the **Object** view to determine the object number that ASAP has given to **L2.EDGE**.

Hints

- The table expresses the spacing between the six lens surfaces in terms of thickness, rather than in absolute global coordinates. Using thickness is common practice in lens design. The thickness values need to be summed cumulatively to convert them to *z* coordinates. This conversion was done for you in Figure 4.16 on page 83.
- Using the vertex *z* value of the spherical surfaces in the second lens does not work for initial estimates of the tube lengths as it did for the first. Remember, the tube must be made slightly too *large* before applying the bounds command.
- The final thickness is the distance to the focus of the triplet. You do not need to model this.

APPENDIX

4A

ASAP SCRIPTS FOR CHAPTER 4

The following ASAP scripts are referenced in “Building and Previewing Geometry” on page 57.

Script 4-1

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY THE BUILDER             ..
!!THIS IS CALLED THE "UGLY" FORM.
ENT OBJECT;OPTICAL Z 0.0 44.55  0 ELLIPSE 16.9 16.9  'L1.FRONT'
!!IN THE BUILDER THE DEFAULT FORM OF A SURFACE IS AN OBJECT.
!!IN THE COMMAND MODE THE DEFAULT FORM OF A SURFACE IS AN ENTITY.
!!SO IN CREATING SCRIPT FROM THE BUILDER AN OBJECT COMMAND IS ADDED.

!!THE FOLLOWING EDITED SCRIPT IS THE PREFERRED FORM OR "GOOD" FORM.
SURFACE
  OPTICAL Z 0.0 44.55  0 ELLIPSE 16.9 16.9
  OBJECT 'L1.FRONT'
!!NOTE THAT THE SCRIPT USES THE OPTICAL COMMAND.
!!SPHERICAL IS A FORM OF THE OPTICAL COMMAND SHOWN ONLY IN THE BUILDER.
!!THE VALUE 0 BEFORE THE WORD ELLIPSE IS THE CONIC CONSTANT.
!!THE CONIC CONSTANT FOR A SPHERICAL SURFACE IS 0.
```

Script 4-2

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY THE BUILDER.
INTERFACE COATING TRANSMIT AIR SK4
```

Script 4-3

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY THE BUILDER.
ENT OBJECT;OPTICAL Z 5 -436.6  0 ELLIPSE 16.9 16.9  'L1.BACK'
INTERFACE COATING TRANSMIT AIR SK4

!!THE FOLLOWING EDITED SCRIPT IS THE PREFERRED FORM.
SURFACE
  OPTICAL Z 5.0 -436.6 0 ELLIPSE 16.9 16.9
OBJECT 'L1.BACK'
  INTERFACE COATING TRANSMIT AIR SK4
!!NOTE THAT THE SCRIPT USES THE OPTICAL COMMAND.
!!SPHERICAL IS A FORM OF THE OPTICAL COMMAND SHOWN ONLY IN THE BUILDER.
!!THE VALUE 0 BEFORE THE WORD ELLIPSE IS THE CONIC CONSTANT.
!!THE CONIC CONSTANT FOR A SPHERICAL SURFACE IS 0.
```

Script 4-4

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY THE BUILDER.
ENT OBJECT;TUBE Z 0.0 16.9 16.9 5 16.9 16.9 0.0 0.0  'L1.EDGE'

!!THE FOLLOWING EDITED SCRIPT IS THE PREFERRED FORM.
SURFACE
  TUBE Z 0.0 16.9 16.9 5 16.9 16.9 0 0
OBJECT 'L1.EDGE'
```

Script 4-5

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY THE BUILDER.
  BOUNDS -.2 +.3
```

CHAPTER

5

RUNNING AND VERIFYING GEOMETRY

In this chapter, you will "run" Builder files for the first time so that the ASAP kernel can become aware of your system. Once this is done, you will have access to additional tools that you can use to further verify that you have modeled the geometry in your system correctly. For most of us, this is a critical step that is worth some effort before blindly proceeding with ray tracing and analysis. The key word is "blindly". This geometry verification step would be tedious indeed without excellent graphical and interactive tools. Several of these are introduced in this chapter.

We are continuing to use the Cooke Triplet model, developed in Chapter 3 and Chapter 4, to illustrate these new concepts and tools.

ASAP Builder and the Kernel

The kernel becomes aware of your system when you "run" the Builder file.

The ASAP kernel is the computational engine that performs all the ray tracing and analysis done by the program. It also maintains a database of all the media, coatings, and geometrical entities we have defined using the Builder, or other methods. The kernel, however, is not active while you type into Builder cells, or even when you complete a line. The kernel becomes aware of your system only when you "run" the Builder file. Information is then passed from the Builder to the kernel, where your various definitions become part of the ASAP knowledge base.

Running the Builder

When you are satisfied with the previews of your geometry, it is time to run the Builder file and get this information into the ASAP system database. You will normally begin the process by clicking the **End** button on the ASAP Toolbar.

Clicking **End** *avoids getting multiple copies of the same geometry into the system database.*

This step throws away all existing geometry (if you have any). Of course, this is not necessary if you have not run a Builder file or ASAP script during the current ASAP session. However, clicking **End** is a habit you need to learn early to avoid getting multiple copies of the same geometry into the system database. When you click **End**, the output window responds with the following:

```
--- END
    System database and settings have been reinitialized
```

Note: You can configure the ASAP Builder so that it automatically runs the **END** command each time the Builder is run. Click the **Append Geometry** button in the Builder, or deselect **Builder> Append Geometry** on the Builder menu. Most people prefer to manually run the **END** command, however, giving them the flexibility to use multiple Builder and script files containing different elements of their system.

If you have not already done so, open your Cooke Triplet Builder file (**Cooke Triplet.enx**), and run the file.The **Run** button is on the Toolbar of the Builder window, and looks like a lightning bolt. When you click it, every line in the Builder is sent to the kernel and run in sequence. You will see each command echoed in the Command Output window. This process takes only a fraction of a second on most computers.

The ASAP kernel is the computational engine that performs all the ray tracing and analysis done by the program. It also maintains a database of all the media, coatings, and geometrical entities we have defined using the Builder, or other methods.

After running the Builder, two interesting changes have taken place in the ASAP landscape. First, you see that the wavelength scale (**WS**) and system units (**SU**) are now defined in the **Status Bar** at the bottom of the screen. Second, the nine surfaces of the Cooke triplet now appear on the **Object** tab in the ASAP Workspace window. For the Cooke triplet, this window should appear as shown in Figure 5.1. (Click the **+** symbols to expand all levels of the drawing tree to see the exact drawing tree shown in the figure.) Both the **Status Bar** and **Objects** tab are indications of the current state of the ASAP system database, and reassure us that our geometry definitions were run successfully.

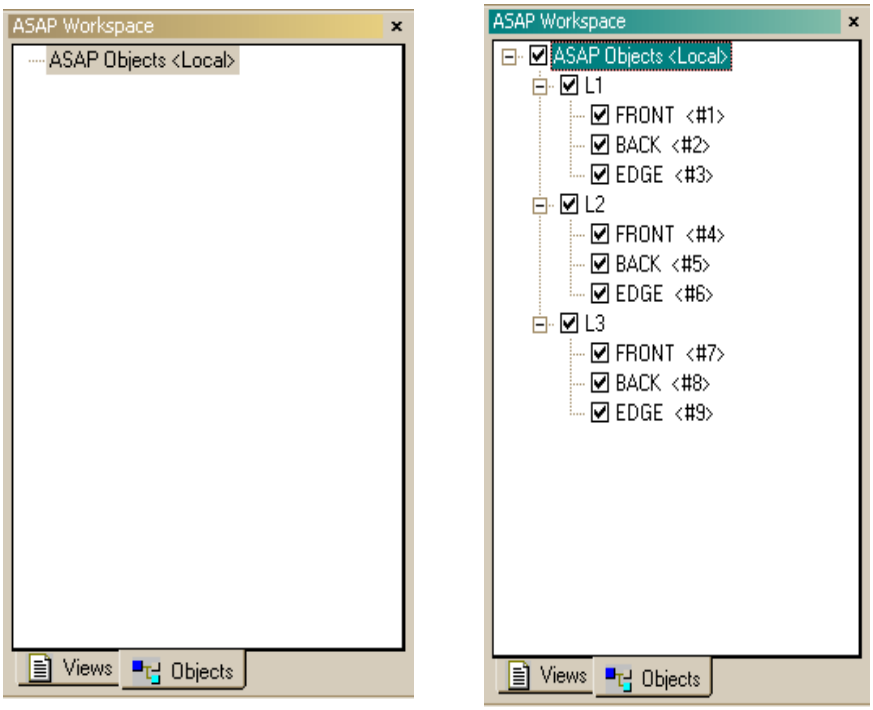


Figure 5.1 (Left) The **Objects** tab is in focus on **ASAP Workspace**, after clicking the **End** button. (Right) The same window is in focus after running the Builder file and expanding the drawing tree. The checked boxes show that all objects are currently "considered". By deleting the check mark, we can temporarily cause ASAP to ignore one or more objects while producing graphics, tracing rays, or performing analysis.

Further, when you go to the **System** menu and click **List Media** or **List Coatings**, you see (in the Command Output window) that your glass types and coating definitions are also known to ASAP now. This information remains part of the ASAP knowledge base until next time you click **End** or exit ASAP.

Profiles Command

One feature that is not available in the **Preview** function within the Builder is the ability to *quantitatively* verify the size and location of the surfaces in our systems. The 3D Viewer can tell us whether the results look right—no pieces missing and everything in generally the right place—but no actual measurements can be made from those views. However, now that we have objects defined within the system database, new tools designed specifically to work from this internally stored data are available to us.

The first new graphics command we will look at is **PROFILES**. This command allows us to display one or more flat slices through the geometrical elements of a system. The **PROFILES** command is useful for making interactive measurements of geometrical dimensions and positions. Later, we will use it in conjunction with ray-trace graphics to show rays moving through optical systems.

Try selecting **System> Profiles** from the main ASAP menus, or click the **Profiles** button on the ASAP Toolbar.

Using either method, you will see the multi-tab dialog box shown in Figure 5.2a and Figure 5.2b.

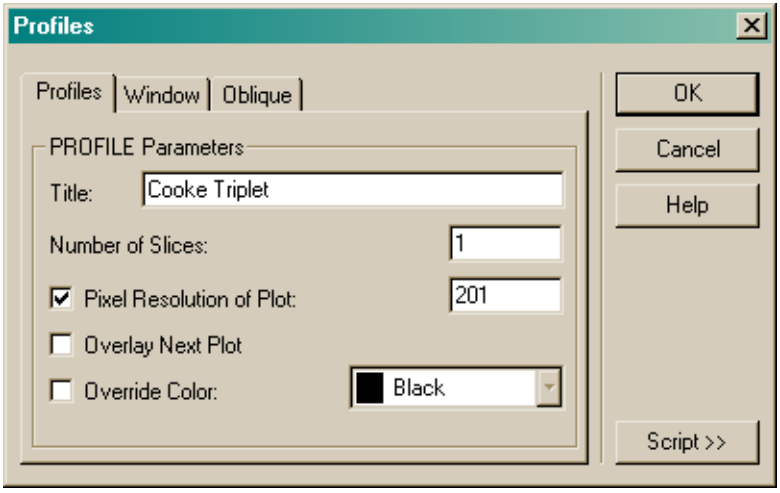


Figure 5.2a The **Profiles** tab on the **Profiles** dialog box. We can place a title on the resulting graphic, select the number of slices, resolution, option to overlay one or more subsequent plots on top of this one, and override the default object colors. The **Script** button expands the box to show the actual ASAP commands that will be issued when you click **OK**.

On the **Profiles** tab, you can type an optional title that will appear on the graph. We named ours “Cooke Triplet”. The default resolution is 201 pixels, which means that the vertical dimension of the plot that ASAP draws will be divided into 201 equal resolution elements. This resolution is usually more than enough. You can leave the number of slices set to 1 (the default), which will give you a single slice through the objects, drawing a cross section or “profile” of the system.

But what plane will do the slicing? You can specify the plane on the **Window** tab.

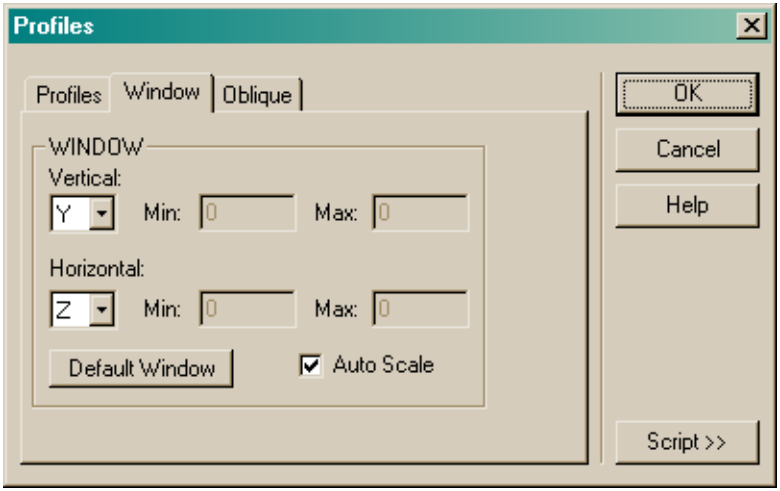


Figure 5.2b The **Window** tab on the **Profiles** dialog box. We can select the projection plane for the resulting view. Note that the first coordinate (Y here) is vertical, and the second (Z) is horizontal. We can specify the window size, or allow ASAP to auto scale to fit all the geometry into the window.

The default on the **Window** tab is the *Y-Z* plane—a good choice in our case, since our optical axis is *Z*. We also want ASAP to auto scale the window (the default selection), making it just large enough to show all nine objects in our system. The other default options will also work well for us. In many situations we never need to visit the **Window** tab at all.

The third tab, **Oblique**, can also be ignored for now. This tab is sometimes used to make an oblique view of your objects (including no perspective), which is occasionally useful.

When you click **OK**, ASAP opens a Plot Viewer window and displays a profile of the system. In the following section, we will give a quick tour of the features available in this window.

See Chapter 5 Appendix, "Script 5-1" on page 111.

Plot Viewer Window

ASAP uses the Plot Viewer to display the results of the `PROFILES` command, and many other graphical verification and analysis tools found in ASAP. Like the **3D Viewer** discussed in the last chapter, it is also worth spending some time exploring and experimenting with this window.

Figure 5.3 on page 92 highlights a few important features of the Plot Viewer window. The origin of your coordinate system may or may not be visible in the view, depending on the choices you made at the **Window** tab of the dialog, and whether or not any objects have been created near to the global origin. But

whenever the origin is included in the view, its position is indicated by the small tick marks at the edge of the plot area. The corners of the window frame are also marked, along with their coordinates.

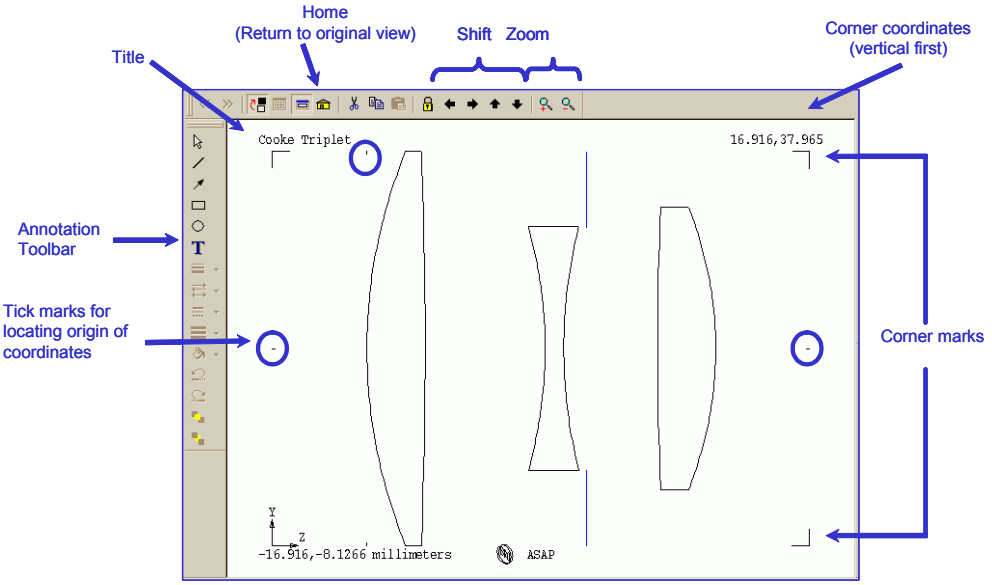


Figure 5.3 A single slice, or cross section, taken through the center of the Cooke Triplet using **Profiles** in a Y-Z window. The origin of coordinates (if present) can be found by connecting the vertical and horizontal tick marks. ASAP also prints the coordinates of the window's corners at the lower left and upper right of the plot.

Once ASAP has drawn the default view, we can zoom in on an area of interest. ASAP provides **Zoom** and **Shift** buttons at the top of the window. It is usually more convenient to select the area by dragging the mouse to define a new window (Figure 5.4). If you get lost zooming and shifting, pressing HOME on your keyboard or selecting the Home button on the Viewer menu bar always returns you to the original scaling and centering.

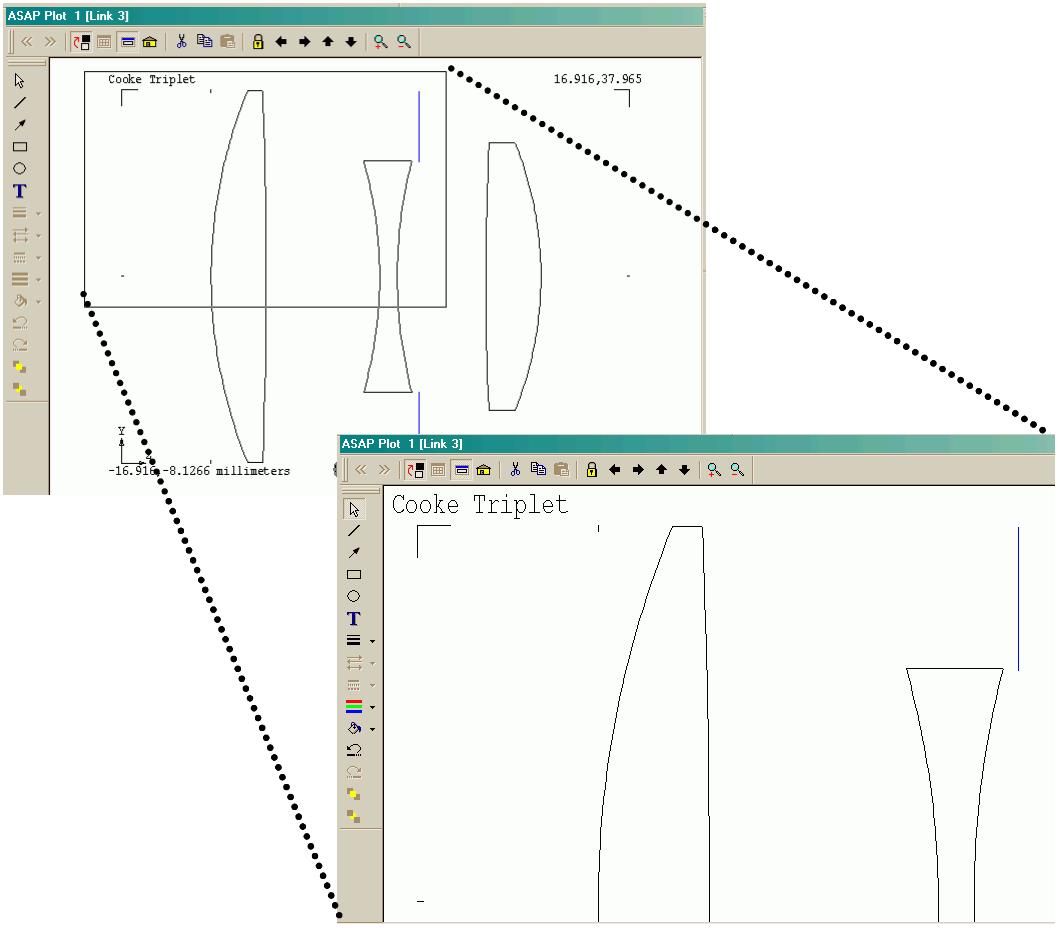


Figure 5.4 (Top) You can zoom in on any part of the window by dragging the mouse to define a rectangular area of interest. (Bottom) Clicking the **Home** button returns the view to its original scale and centering.

One of the most useful interactive features of the Plot Viewer window is the ability to dynamically read out coordinates. Try holding down the SHIFT key and moving the mouse within the Plot Viewer. A pop-up window displays the z and y coordinates of the mouse location within the window (Figure 5.5). This feature could be an excellent way of estimating the starting and stopping z coordinates of the tubes used to model the lens edges prior to bounding in the previous chapter.

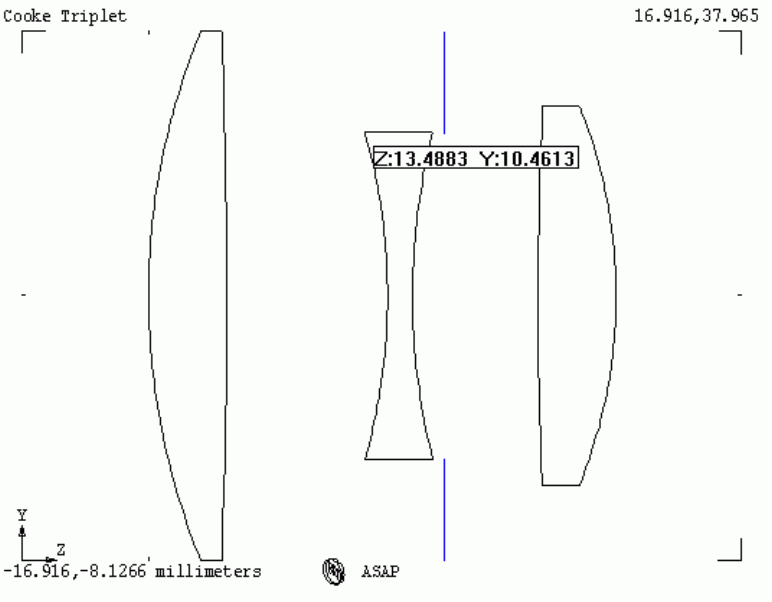


Figure 5.5 By holding down the SHIFT key and moving the mouse, a cursor appears along with the position coordinates. You can use the coordinates in conjunction with zooming the view to make detailed measurements of your geometry.

We could have run the incomplete Builder file after making the front and back surfaces of the lens. Next, we would use **System> Profiles** (or the button) to plot a cross section, and use the mouse to read out the numbers we needed, just a little bigger than the edge of the lens. Once finished, the **End** button would remove all signs of this preliminary attempt to create geometry.

As we did with the 3D Viewer, other details of the Plot Viewer remain (for now) for your personal exploration and experimentation. The Plot Viewer buttons include the most frequently used functions and options, which are listed for reference in Table 5.1. These and other features also appear on the **Plot**, **Edit**, and **File** menus (present only while the Plot Viewer has focus), or on a pop-up menu activated by right-clicking in the **Plot Viewer** window.

You can change the default behavior of the Plot Viewer window in the **User Interface Preferences** dialog box (**File> Preferences**).

Table 5.1 Functions of the various **Plot Viewer** buttons.

	Show **Previous** plot.(By default, the 12 previous plots are saved.)
	Show **Next** plot.
	Toggle (switches) **background** between white and black.
	**Tile** all plots.
	Toggle **Isotropic**. (equal scaling vertically and horizontally) on and off. Default is “on”.
	Return to original or **“home”** scale and centering after zoom and shifts.
	**Cut** (delete) this plot information and copy it to the Windows **Clipboard.**
	**Copy** this plot information to the **Clipboard.**
	**Paste** this plot information to the **Clipboard**.
	**Lock** this window to prevent subsequent plots from overwriting it.
	**Shift** plot left, right, up, or down.
	**Zoom** in or out.

Vector Files and the 3D Viewer

Sometimes, making sense out of complicated three-dimensional information is difficult when it is projected onto a two-dimensional window, like that of the Plot Viewer. For example, try selecting **System> Profiles** again, but this time choose 21 slices instead of just one. The result appears as shown in Figure 5.6a.

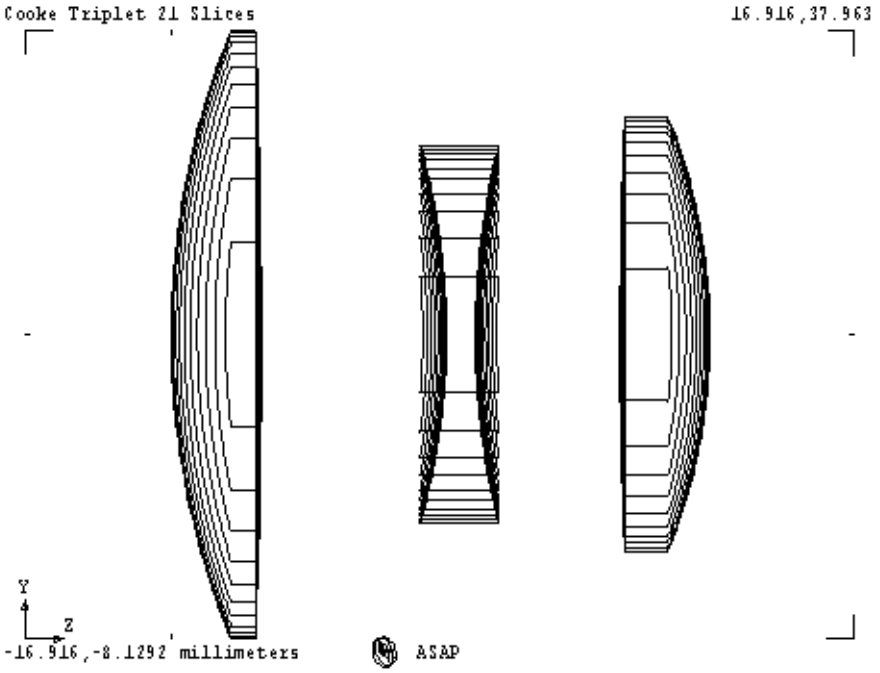


Figure 5.6a From the **Profiles** dialog box (**System> Profiles**), when you run 21 slices, the result is the intersection of 21 evenly spaced planes, slicing through each element of your geometry. The plot is projected onto the flat Y-Z view.

Do not be concerned if you are not entirely sure what you are looking at. This is the projection of 21 slices, distributed evenly over each of the objects in the system database.

Along with the two-dimensional plot presented in the Plot Viewer window, ASAP also creates a three-dimensional version of the data in a "vector file". This version is done automatically with **Profiles** (and many other graphics commands), and requires no additional effort on your part until you are ready to look at it. After running **Profiles** with 21 slices, try clicking the **3D View** button on the ASAP Toolbar. It looks similar to the **Preview** button in the Builder.

The result appears in Figure 5.6b. The 3D Viewer window that opens is the same tool that you used for previewing objects in the Builder, so some of its functions should already be familiar. Try right-clicking and dragging to rotate the view. Now, the meaning of "21 slices" may be clearer.

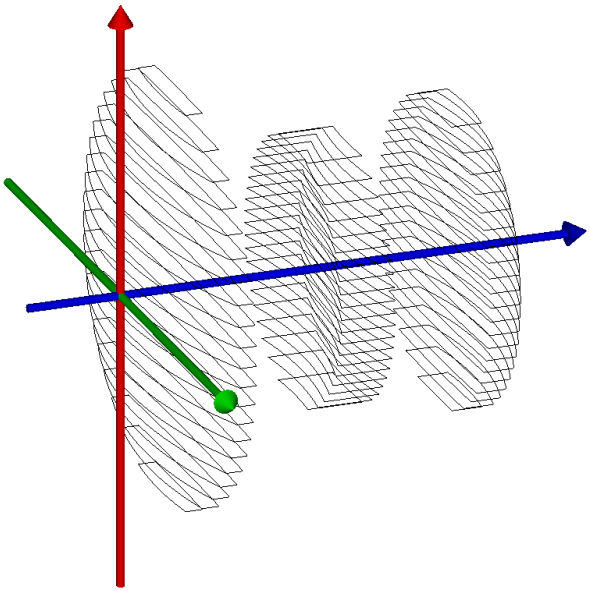


Figure 5.6b ASAP automatically creates a three-dimensional version of **PROFILES** data for viewing in the **3D Viewer**. Our ability to quickly and smoothly rotate this graphic is often an invaluable aid in visualizing a complicated view in projection.

As you continue learning about and working in ASAP and its user interface, you will see that an amazing amount of information accumulates in this vector file, and other files compatible with the 3D Viewer. Eventually, you will be able to generate three-dimensional views of your model showing geometry, ray trails, spot diagrams, and other analysis results (Figure 5.7).

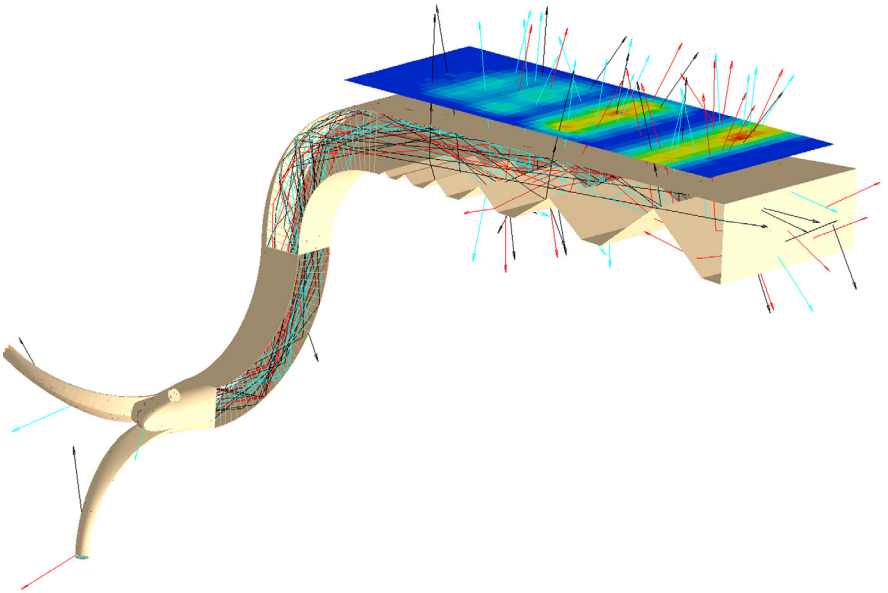


Figure 5.7 This plot is a light-pipe design with some surfaces temporarily hidden, so we can see ray paths within the volume. The top plane is a pseudo-color rending of the irradiance distribution. Because ASAP can make a three-dimensional version of most graphical results, we can make elaborate illustrations, which combine geometry information, ray trace details, and analysis results.

Plot Facets

Another useful ASAP tool is **PLOT FACETS**. It accomplishes the first step in making three-dimensional views of your geometry, like those used in the Builder's **Preview** function. This command approximates (for graphical purposes only) all surfaces in the system database by a series of flat polygons. This process is called "faceting" or sometimes "meshing" a surface.

When you choose **System> Plot Facets** from the menu, another dialog box with three tabs appears. The **Window** and **Oblique** tabs are identical to those you saw in the **Profiles** dialog box. On the **Plot Facets** tab, however, we are confronted with some faceting options. This dialog box is where we control the number of polygons that we want to use to render our objects. For now, leave the inter-edge and intra-edge values set to five and the **Maximum Facets** blank. We will have more to say about these values shortly.

After clicking **OK** in the dialog box, you should see the result that appears in Figure 5.8.

See Chapter 5 Appendix, "Script 5-2" on page 111.

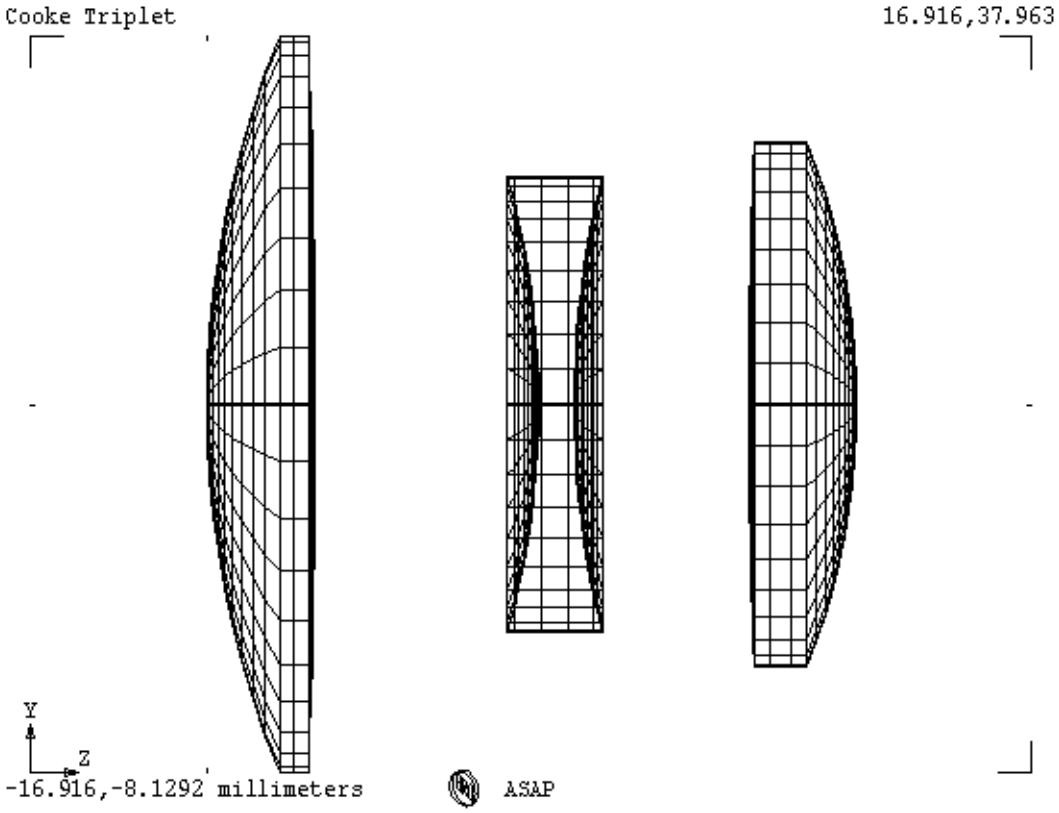


Figure 5.8 Plot Facets renders surfaces in the system database by representing them as a series of flat polygons. When viewed in the **Plot Viewer**, we see the boundaries of these "facets" projected into the selected window (the Y-Z plane here).

From this view, the usual features of the Plot Viewer are available, including zooming, shifting, and the interactive measurements discussed in the previous section. So far, however, the **PLOT FACETS** command does not really appear to have significant advantages over **PROFILES**. If all we are doing is using the cursor to make some measurements, the task could be accomplished with a slice through the center, and with less clutter in the graphic. The real virtue of **PLOT FACETS** lies in the vector (three-dimensional) version of the data that is also created, but not automatically displayed. By clicking the **3D View** button, this information is displayed. If you have not cleared the **PROFILES** information created earlier, you will see the **PROFILES** slices superimposed on the **PLOT FACETS** results, as in Figure 5.9a on page 100.

If you want to see the results of only the **PLOT FACETS** in the 3D Viewer, you can clear the **PROFILES** information from the vector data before using the **PLOT FACETS** command. To do this, click the **Vector Rewind** button, just to the left of the **3D View** button on the ASAP toolbar.

Vector rewind removes all previous data from the three-dimensional vector file. The result of only our most recent plotting command will be present, so that you see the **PLOT FACETS** vector data alone (Figure 5.9b).

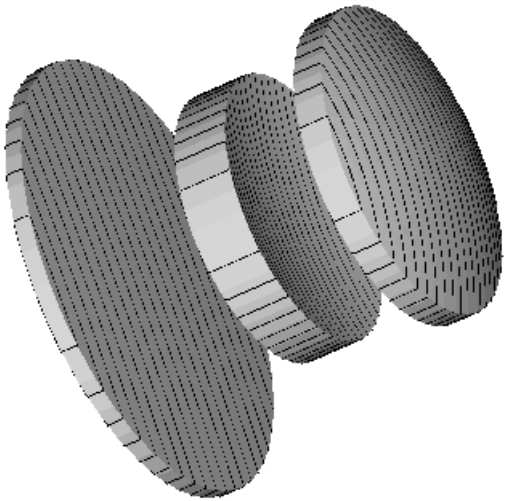


Figure 5.9a The vector file accumulates all plot information since the last **Vector Rewind** or the last **End**. The figure shows the combination of the previous **Profiles** graphic on top of the smooth-shaded 3D information created by **Plot Facets**. The drawing tree view in the **3D Viewer** lists every object twice.

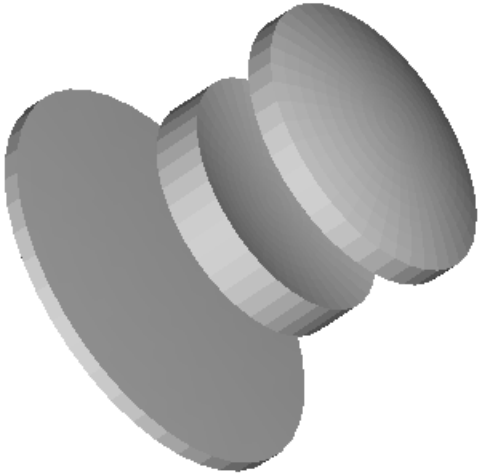


Figure 5.9b By clicking **Vector Rewind**, we can "clean out" the vector file prior to running **Plot Facets**. At this point, only this last result appears in the **3D Viewer**.

After clicking the **3D View** button, you see a three-dimensional view that is not only reminiscent of the **Preview** we have seen before, but is, in fact, exactly the

same thing. We now have the means of manually creating that three-dimensional view—even as an afterthought, if we did not learn everything we had anticipated from the information presented in the Plot Viewer. We will see that it also gives us more flexibility when combining various graphical products into a single 3D view, like the one shown previously in Figure 5.7 on page 98.

How Many Facets?

We need to return briefly to the issue of the inter-edge and intra-edge parameters and the **Maximum Facets** in the **Plot Facets** dialog box. We left them set to five and blank, and got a reasonably good figure both in the Plot Viewer and the 3D Viewer windows. But what do these numbers mean?

Faceting or meshing ASAP surface-based objects is not a simple process because of the mathematics of this entity type (they are represented internally as implicit polynomials). ASAP is using a "smart faceting" algorithm that brings considerable intelligence to the problem. As a result, it is sometimes difficult to fathom exactly what is going on by looking at the results. Simply stated, more is usually better but slower to compute, and harder (slower) to manipulate in the 3D Viewer. Too few will result in poor detail, or even missing objects. This point is illustrated in Figure 5.10a and Figure 5.10b.

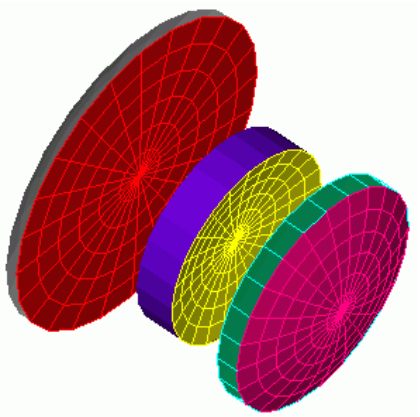


Figure 5.10a `PLOT FACETS 1 1`. We have forced ASAP to facet at this level. Note that **L1.EDGE** has vanished completely (too thin), and the circular objects have flat, faceted edges.

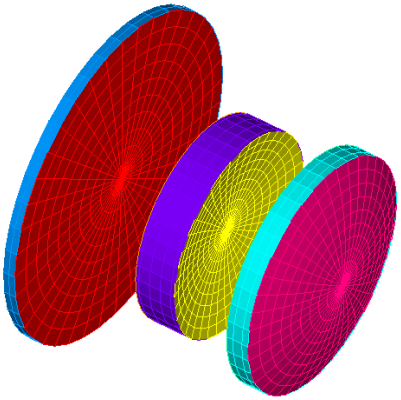


Figure 5.10b `PLOT FACETS 5 5`. For simple objects, this is a reasonably good faceting choice. While the flat facets are still apparent, particularly on the tubes, they become far less obvious if you select the **Smooth Shaded** viewing mode.

The above figures illustrate how you can tell if the problem is real or "just faceting". We will return to this issue again in Chapter 13, "Edges" on page 233 when we discuss edge entities.

Another characteristic of the "smart faceting" algorithm is to allow a maximum number of facets (the individual flat polygons making up the surface) for an object's surface. If the **Maximum Facets** box is left blank, ASAP allows only 4096 facets per object to be created with the **PLOT FACETS** command. If a 0 is placed in this box, ASAP creates as many facets as would be created with the given values for the inter-edge and intra-edge parameters. Any other number placed in this box dictates an approximate maximum number of facets per object.

The answer to the question of how many facets to use, however, is simple: use as many as it takes to get a nice picture. If you are using **PLOT FACETS** to verify geometry and you see something that does not look right, increase the number of facets and see if the view improves. That is how you can tell if the problem is real, or "just faceting". We will return to this issue again in Chapter 13, "Edges" when we discuss edge entities.

It is worth emphasizing again that these faceting issues are relevant only to the artwork, and generally have nothing to do with what the object really looks like to a ray. The rays are working strictly with the mathematical representation of the surfaces, not the faceted approximations. (If you are particularly mathematically inclined and would like to know more about the detailed representation of surface-based entities in ASAP, see the sidebar, "System Database" on page 108.

Managing Multiple Plots

By now you have probably run several **PROFILES** and **PLOT FACETS** commands. You have likely noticed that ASAP has placed all these graphics in a single **Plot Viewer** window. ASAP has not permanently overwritten the older results, however. The last 12 plots (by default) remain available to you for review. The **Views** tab of the **ASAP Workspace** window allows us to select a specific plot for review (Figure 5.11). The most recent plot is always at the bottom. You can also page through the plots using the **Next Plot** and **Previous Plot** buttons on the top left of the **Plot Viewer** window.

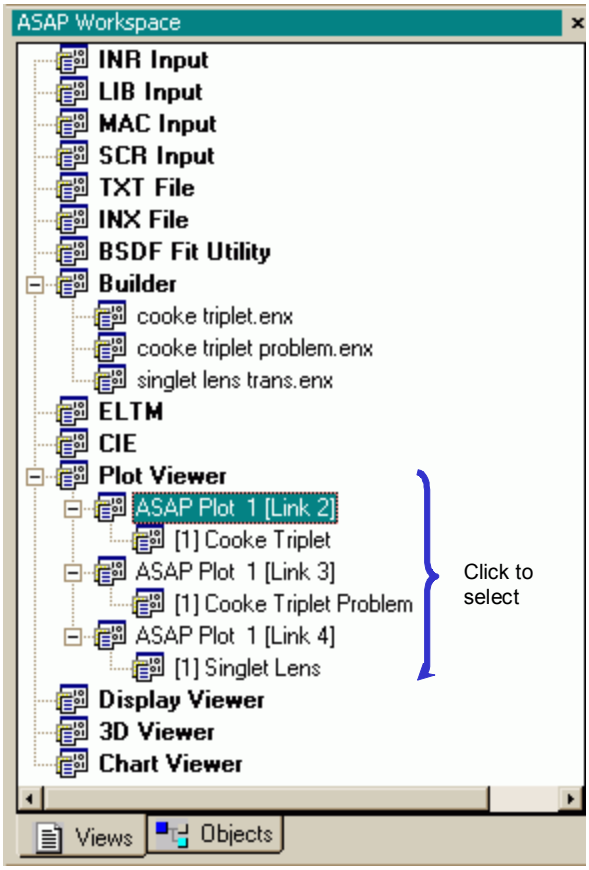


Figure 5.11 While ASAP appears to be writing all plots into a single window, up to 12 plot windows are saved before older information is overwritten. You can switch between these by clicking any of the plots on the **Views** tab of **ASAP Workspace**, below **ASAP Plot 1**. This view shows only 3 plots.

Sometimes, you may need to make a side-by-side comparison of plots. We have several options available for this. Perhaps the simplest is the **Tile** button in the **Plot Viewer** window, which displays all plots (up to 12) in a tiled format. To carefully compare just two plots, a second plot window can be temporarily opened, and one of the plots placed within it. Here is a procedure:

1. Click one of the plots in the **ASAP Workspace** window under **ASAP Plot 1**.
2. From the menus, select **Plot> Copy**, or use the copy button in the **Plot Viewer** window.
3. Right-click **Plot Viewer** in the **ASAP Workspace** window, and select **New**.
4. With this new window in focus, select **Plot> Paste**, or use the paste button in the **Plot Viewer** window.
5. Move and size the windows as needed for your side-by-side comparison.

When you finish, you can close this new window by right-clicking **Plot 2** in the **ASAP Workspace** window and selecting **Close**. If you prefer, leave it open for future use. Individual plot frames can be copied, cut, and pasted between these windows at any time.

We have been describing the default behavior of the **Plot Viewer** function. You can change this and other plotting functions using the **File> Preferences** dialog box, selecting the **Plot Viewer** tab (Figure 5.12). From here, you can change the default number of display windows, and the number of plots per display window.

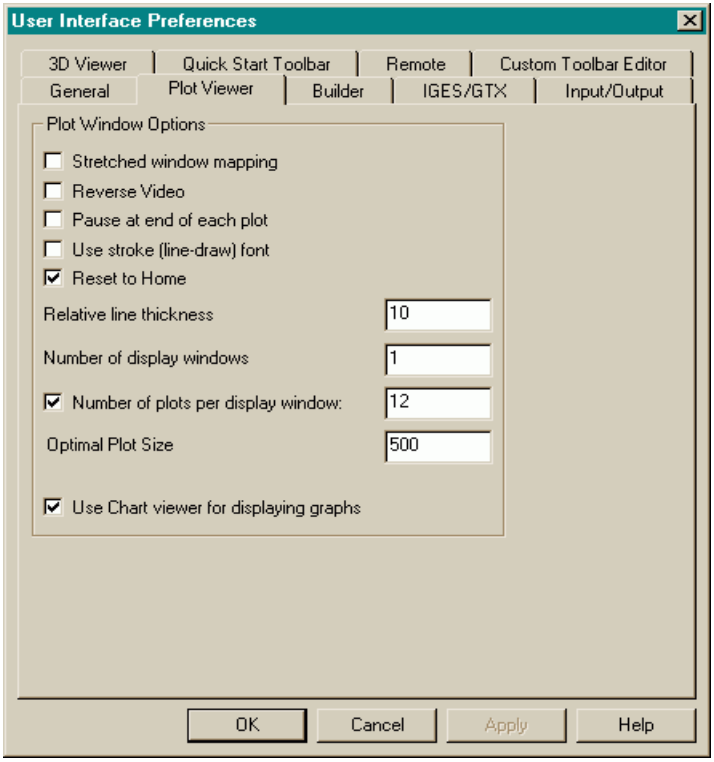


Figure 5.12 You can control many properties of the user interface by selecting **File> Preferences**. From the **Plot Viewer** tab, you can adjust the number of display windows and number of plots allowed in each window, along with several other features of the **Plot Viewer** window.

When you choose to have more than one display window, ASAP distributes all future plots into these windows in a rotating fashion. You can experiment with these settings to see what works best for you.

Summary

In this chapter, we have discussed the following new commands:

ASAP Commands	ASAP Menu	Description
WINDOW PROFILES	System> Profiles	PROFILES draws one or more slices through the system geometry in the Plot Viewer. WINDOW controls the viewing plane. The same information can also be viewed in the 3D Viewer.
WINDOW PLOT FACETS	System> Plot Facets	PLOT FACETS makes a 3D view of the system geometry in the Plot Viewer by approximating (for graphical purposes only) as flat polygons all the surfaces of the system geometry. WINDOW controls the viewing plane. The same information can also be viewed in the 3D Viewer (equivalent to Preview).

We ran a Builder file for the first time. This caused the ASAP kernel to interpret the commands in the Builder and, in so doing, to build a "system database". This file includes detailed information about our media, coatings, and the geometry—a key concept in understanding how ASAP works. (If you would like to know more about the detailed representation of surface-based entities in ASAP, see "System Database" on page 108.) Once built, the system database will remain in place until you quit ASAP, or click the **END** button to discard all previous work. When we introduce rays into the system, they will use the information in this database to find their way through the geometry. If we defined the interfaces correctly, rays will show the expected optical behavior (reflect, refract, and so on) at each object.

Once you run the Builder, ASAP knows about your media, coatings, and the various geometrical objects you are defining. The tools in the **System** menu are now available to you for listing, plotting, and verifying elements of your system. You can always tell if you have run the Builder by checking the **Objects** tab on the **ASAP Workspace** window. No objects are listed until you have performed this critical step.

We looked closely at the **PROFILES** and **PLOT FACETS** commands, which allowed us to explore the features of the ASAP **Plot Viewer**. You will also be using the **Plot Viewer** during ray tracing and for many analysis graphics as well. We will be revisiting this important tool frequently throughout the *Primer*.

We also found that most plot functions are producing a three-dimensional or "vector" version of the plot information. This version is not automatically displayed, but can be viewed at any time by launching the **3D Viewer**.

It is a deeply rooted part of the ASAP philosophy that optical ray tracing of the sort performed by ASAP is an inherently mathematical rather than a graphical problem. For the most part, however, this distinction is not important to the end user. Good graphical tools in the user interface will convert the native, fundamentally mathematical representation of your geometry that ASAP prefers into forms that most of us find more useful and intuitive. The mathematics is still fundamentally what ASAP uses, so it may help you to see the actual data with which ASAP works.

You can get a glance into the world of the ASAP system database by returning to the **Objects** *tab on the* **ASAP Workspace** *window. Expand the tree to the level of individual objects, and right click L1.FRONT. Click* **Properties** *in the pop-up menu, and ASAP runs the* `PRINT` *command, printing the database information for Object 1. The information appears in the* **Command Output** *window, as illustrated below:*

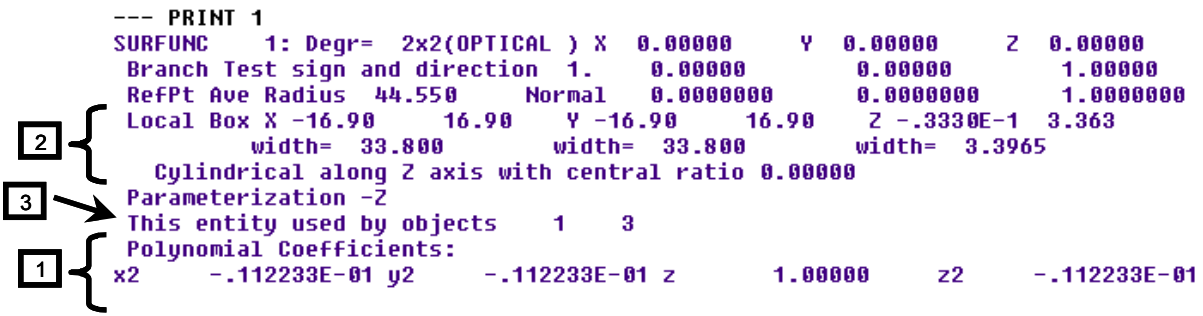
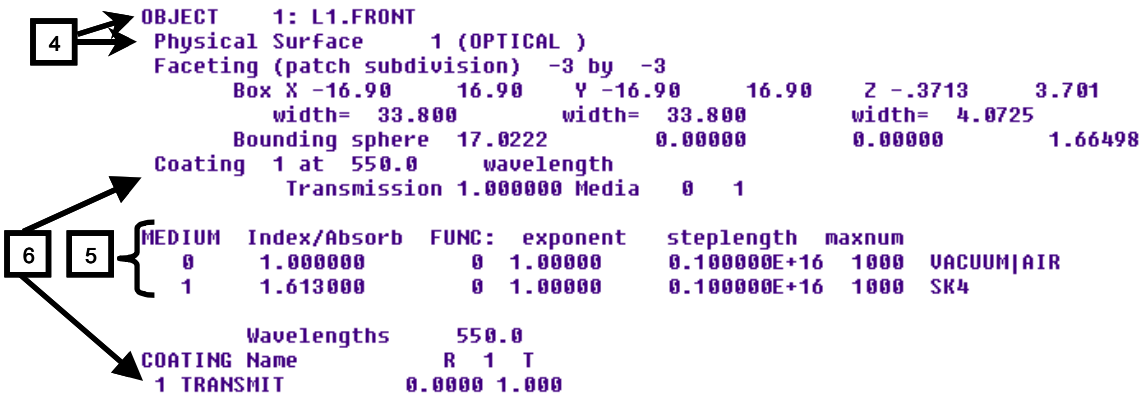


Figure 5.13 Printed database for the mathematical entity



Sidebar Figure 5.14 Printed database for the object

(Continued)

System Database (concluded)

The basic mathematical entities (just one in this case) used to construct Object 1 (a sphere) are described first:

1. The mathematical entity that defines Object 1 is the locus of points that satisfy the equation:

$$-0.0112233y^2 - 0.0112233x^2 + z - 0.0112233z^2 = 0$$

2. The only part of this surface that we are interested in is within the cylindrical pillbox of radius 16.90, extending from

$$z = -0.03330$$

to

$$z = 3.363$$

3. This entity is also used by Object 3 (the tube that we bounded with L1.FRONT and L1.BACK).

The object data comes next:

4. Object 1 is named L1.FRONT and it is based on entity 1 (described above).
5. It is the boundary between Vacuum|Air and SK4.
6. It has been assigned coating number one, which is defined at the bottom for reference.

We found that most plot functions are also producing a three-dimensional "vector" version of the plot information. This version is not automatically displayed, but can be viewed at any time by launching the 3D Viewer.

APPENDIX

5A

ASAP SCRIPTS FOR CHAPTER 5

The following ASAP scripts are referenced in “Running and Verifying Geometry” on page 87.

Script 5-1

```
!!THIS IS THE COMMAND SCRIPT PRODUCED FROM THE PULL-DOWN MENU.
WINDOW Y Z
OBLIQUE OFF
PROFILES 0 0 -1 PIXELS 201

!!THE FOLLOWING IS THE SCRIPT YOU ARE LIKELY TO CREATE.
WINDOW Y Z
PROFILES 0 0 -1 PIXELS 201
```

Script 5-2

```
!!THIS IS THE COMMAND SCRIPT PRODUCED FROM THE PULL-DOWN MENU.
WINDOW Y Z
OBLIQUE OFF
PLOT FACETS 5 5

!!THE FOLLOWING IS THE SCRIPT YOU ARE LIKELY TO CREATE.
WINDOW Y Z
PLOT FACETS 5 5
```

CHAPTER

6

CASSEGRAIN TELESCOPE MODEL

In this chapter, we leave the Cooke triplet for now, and begin building a classical Cassegrain telescope. We can simulate the essential features of this design with just three surfaces, shown in profile in Figure 6.1. The primary mirror is in the shape of a parabola, and the secondary mirror is hyperbolic.

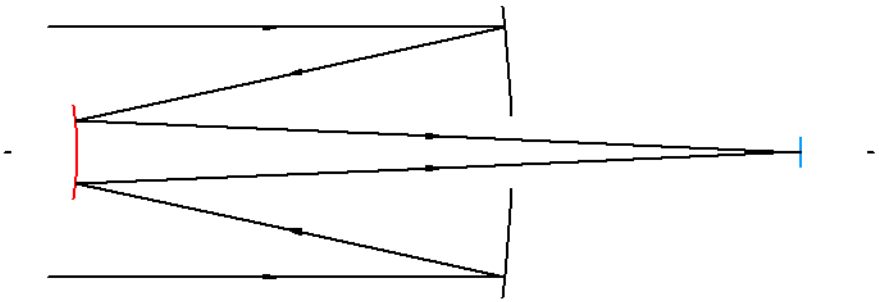


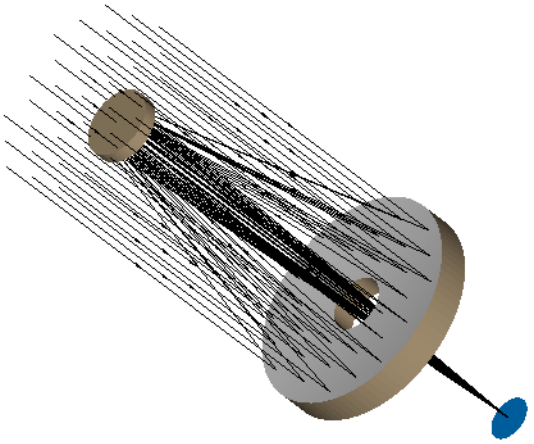
Figure 6.1 A classical Cassegrain telescope. Light enters from the left, strikes the concave primary mirror, and reflects a second time at a convex secondary mirror. Rays exit the system through a hole in the primary.

This design is capable of giving virtually perfect on-axis images, though aberrations increase rapidly away from the axis. Other features of this design are discussed in the sidebar,"What is a Cassegrain Telescope?" on page 114.

What is a Cassegrain Telescope?

Most large telescopes use a mirror, rather than a lens, to gather light. Because light must pass through the lens, it may be supported only around the outer edge. This limits "refracting" telescopes to diameters up to one meter, since larger lenses will sag significantly under their own weight. Gravity deforms their shapes (and hence, the image quality) in a way that depends on where in the sky they are pointed.

Reflecting telescopes can be made very large because the objective mirror is supported by a rigid structure behind it. The only problem is, the image formed by a concave mirror is in front of it. This is generally an inconvenient and impractical location because of its height above the floor, and its position "in the way" of the incoming light.



One way of fixing this was proposed by L. Cassegrain in 1672. He added a convex mirror called a "secondary" some distance in front of the focus of the primary mirror. The light then exits the system through a hole in the primary mirror as shown in the figure. While the secondary mirror is in the way of the incoming light, the only effect it has on the focused image is to remove a small amount of the incoming light, and add some additional (usually minor) diffraction effects.

The Cassegrain telescope is a compact design. A simple one-mirror system with the same effective focal length would have an over-all length approximately four times larger than in the Cassegrain configuration. Now a telescope with a relatively long focal length can be built into a short tube or lattice structure. This, in turn, allows the dome that houses the telescope to be smaller and cheaper as well.

Another important feature of the Cassegrain design is that the focal plane is now located at the rear of the telescope. This is a convenient place for mounting large, heavy instruments. By placing the combined mass of the primary mirror and the focal plane instrumentation close together, it is far easier to balance the system mechanically. For these reasons most modern astronomical telescopes have this basic Cassegrain configuration, although there are many variations.

The Cooke triplet modeled in the previous chapters required only spherical surfaces and tubes. What surfaces are available in ASAP to model planes, parabolas, and hyperbolas? We will need to introduce only two new surfaces to develop this model: the **PLANE** and the **OPTICAL**. Most of the real work is in the exercise at the end of this chapter. This model is used once more in Chapter 10 on Page 179.

> ***Note***: If you are planning to use ASAP to study illumination or other non-imaging systems, the time you spend on these examples is time well spent. Both imaging and non-imaging problems share the same ray tracing and analysis techniques. The early examples in this *Primer* focus on imaging problems only because the basic sources are simpler, allowing us to progress more rapidly. We will concentrate on non-imaging applications beginning with Chapter 16, when we introduce extended sources.

Plane Surface

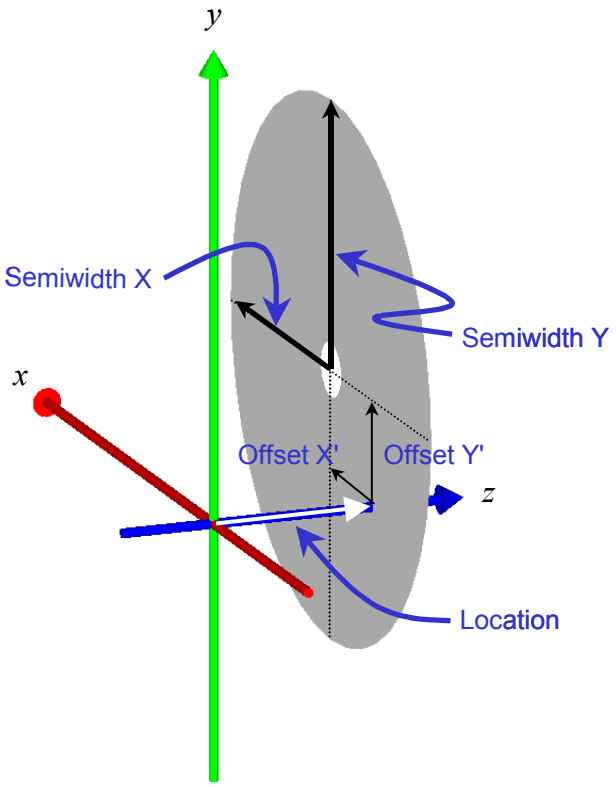
You will need a planar surface to model the focal plane of the Cassegrain telescope. You could model it as a sphere with an infinite (or at least an arbitrarily large) focal length, but this would not be as efficient. The **PLANE** surface is located with the other surface-based entities in the Builder. While three versions of the plane command exist, by far the most commonly used is **Geometry> Surfaces> Plane**. The default Builder entry appears as shown here.

*	Type	Name	Axis	Location	Aperture	Semiwidth	Semiwidth	Obs Ratio	Offset X'	Offset Y'
OBJ	Plane	Detector	Z	3.0	ELLIPSE	15.0	5.0	0.1	2	2

Figure 6.2 Default Builder entry for the **PLANE** Surface

This entry allows us to define a plane perpendicular to any of the three global coordinate axes. We can even define skew planes with this version of the **PLANE** command once we learn to shift and rotate objects (Chapter 15, “Repositioning Geometry and Rays”). For this reason, the other two forms of **PLANE** are rarely used.

The aperture shape, semi widths, obscuration ratio, and offset have exactly the same meanings as they did for the spherical surface used in Chapter 4. The default example does not need most of these. It will create a circular plane located at the origin, normal to the z axis, and having a half-diameter (semi width) of 1. The other fields are optional, though it is always a good idea to name objects. A complete example that shows the use of the obscuration ratio and offsets appears in Figure 6.3.



*	Type	Name	Axis	Location	Aperture	Semiwidth	Semiwidth	Obs Ratio	Offset X'	Offset Y'
OBJ	Plane		Z	0.0	ELLIPSE	1.0				

Figure 6.3 The **PLANE** command, in its most general form, allows for a hole in the center (always with the same shape as the **Aperture** parameter), and an offset of the entire object, away from the specified **Axis**.

Optical Surface

The only other surface command you will need to model for the Cassegrain telescope is **Geometry> Surfaces> Optical**. This one command allows us to model any conic: paraboloids, ellipsoids (including spheroids), and hyperboloids. Further, we can add axially symmetric (even) aspheric terms up to twentieth order to any of these, making it the command of choice for most common optical surfaces used in imaging systems.

With all this flexibility in one command, it may not be surprising that the Builder line for an **OPTICAL** surface has many options—22 entries in all, though many are optional. It will likely require some horizontal scrolling for you to see

all of these. We will discuss this long Builder line in three pieces, starting with the default appearance of the first six cells on the left.

*	Type	Name	Cmd	Axis	Location	Radius of Curvature	Conic Constant
OBJ	Optical		Axis	Z	0	2	0

Figure 6.4 Beginning portion of the Builder line for the `OPTICAL` surface

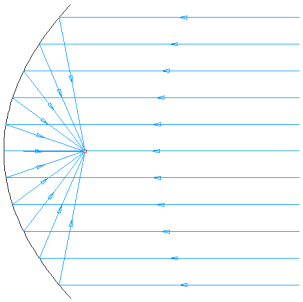
Most of these should look familiar, reminiscent of the entry for **Geometry> Surfaces> Spherical**. Only the conic constant is new. A common convention used in analytical geometry (and optics) is to specify the type of conic we want to create with a single, continuous parameter. See the sidebar, "The Mathematics of Conics" on page 118. The table below summarizes its meaning.

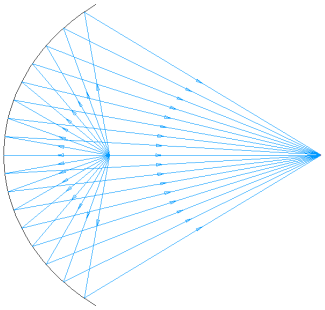
Table 6.1 Conic Constant

Conic Form	Conic Constant (k)
Hyperbola	$\kappa < -1$
Parabola	$\kappa = -1$
Ellipse	$-1 < \kappa$
Circle	$\kappa = 0$

The Mathematics of Conics

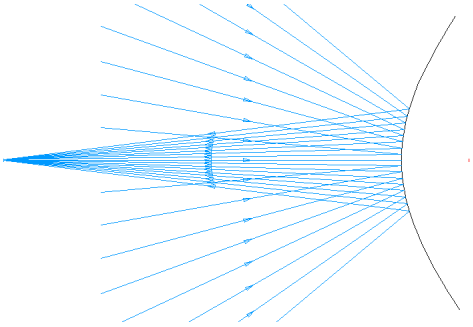
Conic sections (the circle, ellipse, parabola, and hyperbola) are important to optics because each has unique mathematical properties that we can exploit to control light. For example, all rays parallel to the axis of symmetry of a parabolic mirror will focus at a single point, its focus. This makes it useful for both imaging infinitely distant point sources (like stars), and collimating pseudo point sources.





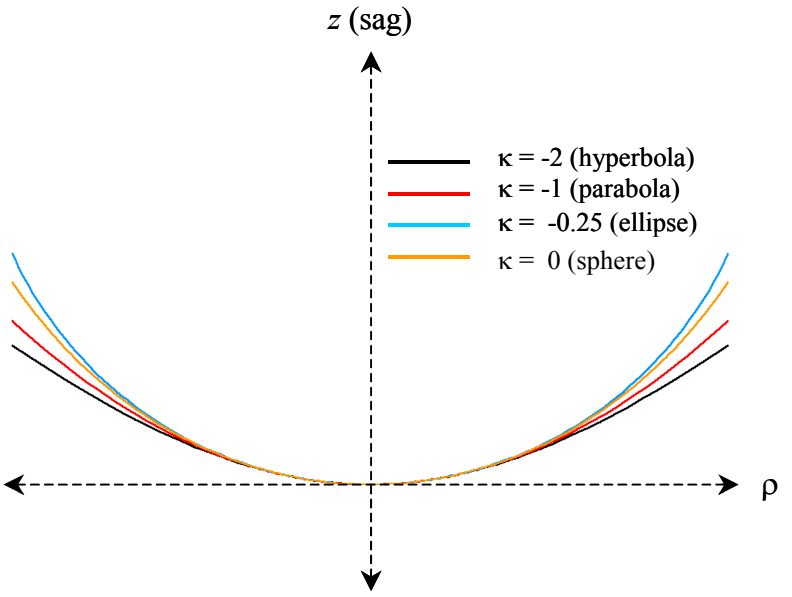
An elliptical mirror has two foci. Any ray coming from one focus will reflect and pass through the other. This property is often used to concentrate light from a small source located at one focus of the ellipse onto the entrance face of a light pipe located at the other.

Hyperbolas, like ellipses, have two foci, and behave much like an ellipse in reverse. Any rays directed toward one of the foci will be reflected off of the convex side of the hyperbola and pass through the other focus. This is why the hyperbola was selected as the secondary of the classical Cassegrain telescope. The result is that all on-axis rays hitting the parabolic primary will be intercepted short of the "prime focus" and converge instead at the second focus of the hyperbolic secondary.



Spheres show up everywhere in optics, not because they have such grand optical properties, but because they are easy to fabricate. They are the only conics that can be generated simply by randomly rubbing one piece of glass against another with an abrasive material in between. Designers can create excellent low-cost systems of spherical surfaces that, in ensemble, compensate for the aberrations of the individual spherical elements.

Continued



Any conic surface can be written as an implicit second-degree polynomial,

$$\rho^2 - 2rz + (\kappa - 1)z^2 = 0,$$

where

ρ is the radial coordinate $\sqrt{x^2 + y^2}$

r is the vertex radius of curvature

z is the sagitta or "sag" of the surface

κ is the conic constant

This can be solved for Z to give the sag equation:

$$z = sag(\rho) = \frac{\frac{\rho^2}{r}}{1 + \sqrt{1 - (1+\kappa)\left(\frac{\rho}{r}\right)^2}}$$

ASAP uses this basic mathematical representation to calculate ray intersections with surfaces defined with **OPTICAL** *(as long as no aspheric deformation coefficients are included). See the ASAP online help for details on the* **OPTICAL** *command when deformations are specified.*

All parabolas have a conic constant of –1, and all spheres (a special case of the ellipse) have a conic constant of 0. If someone asks you to model an elliptical or hyperbolic surface (as we will do shortly), you will need the value of κ as part of the specification for the surface. If no conic constant is specified (as is the case in the default Builder line for **OPTICAL**), a value of 0 (a circle) is assumed.

The meaning of **Radius of Curvature** is obvious if we are talking about a circular section (as with a spherical mirror), but less clear for the other three conics. The curvature of these varies continuously depending on where we are on the curve. We actually need to specify the *vertex* radius of curvature. This, along with the conic constant, completely and uniquely specifies the conic.

The middle section of the **OPTICAL** command includes the aspheric terms, and the **Expand** option.

Figure 6.5 Middle portion of the Builder line for the **OPTICAL** surface

These terms are all optional and not needed for pure conics, like those needed to model a classical Cassegrain telescope. They can all remain blank.

The final parameters for **OPTICAL**, at the far right of the Builder line, appear below.

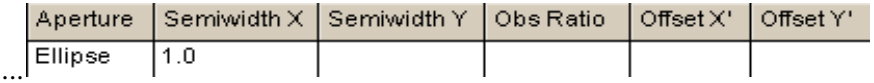


Figure 6.6 End portion of the Builder line for the **OPTICAL** surface

These should be familiar parameters again. They are the same specifications that were required for both the spherical and planar surfaces already discussed, and have the same meaning here.

Before leaving the **OPTICAL** command, we should mention some apparent overlap of this ASAP command with two other surfaces that can be modeled using the ASAP Builder: **Geometry> Surfaces> Spherical** and **Geometry> Surfaces> Parabolic**. Why do we need these, since spheres and parabolas can be modeled using the **OPTICAL** command? The answer is, “We do not.” In fact, the ASAP kernel does not know anything about the “Spherical” and “Parabolic” commands. If you carefully examine the Command Output window after running a Builder file that uses **SPHERICAL** or **PARABOLIC**, you will find that the actual command echoed there is **OPTICAL**. Because the **OPTICAL** command is so long, and because the aspheric terms are often not necessary, these “short-cut” versions of the command have been included in the Builder for our convenience.

Summary

In this chapter, we have discussed the following new commands:

ASAP Commands	Menu	Description
`SURFACE; PLANE`	Builder: **Geometry> Surface> Plane**	Creates a rectangular or elliptical plane.
`SURFACE; OPTICAL`	Builder: **Geometry> Surface> Optical**	Creates classical optical surfaces, including hyperbolas, parabolas, ellipses, and spheres. Any of these surfaces can include aspheric terms, up to the twelfth order.

Recall from Chapter 1 that every ASAP project consists of four basic steps:

1. √ Building the system model
2. Creating sources (rays)
3. Tracing the rays
4. Performing the analysis

With the addition of these last two surface types, and the experience of building and verifying the Cassegrain telescope model (“Exercise 2: Cassegrain Telescope” on page 122), you will have completed step one, and be ready to move on to step 2, creating sources.

Exercise 2: Cassegrain Telescope

1 Construct a model of the classical Cassegrain telescope described by the table and figures below. Assume that the mirror surfaces are 100% reflecting.

	Primary	Secondary
Diameter	10 cm	3.2 cm
Central hole diameter	2.5 cm	none
Conic constant	-1 (parabola)	-2.25 (hyperbola)
Vertex radius of curvature	40 cm	12.5 cm

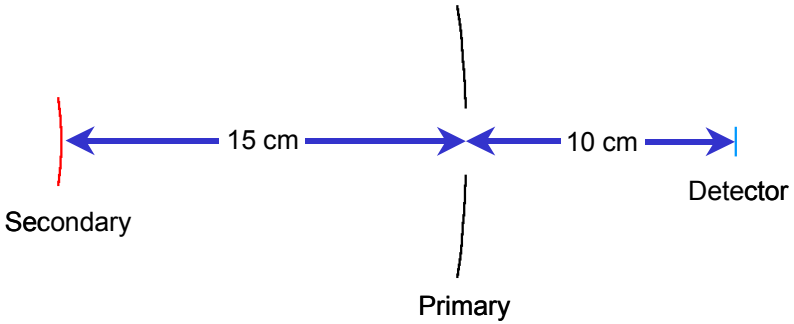


Figure 6.7 Exercise 2: Location of the elements of the Cassegrain telescope.

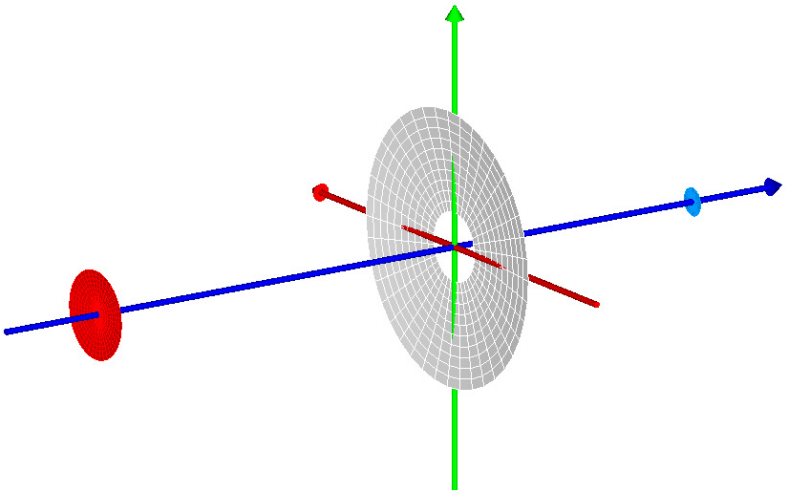


Figure 6.8 Exercise 2: Elements of the Cassegrain telescope produced by **PLOT FACETS** and shown in the **3D Viewer**.

2 Make two profiles of the system: one with a single slice and one with nine slices. View the profiles in the 3D Viewer.

3 Use the cursor in the Plot Viewer to estimate the sagitta or “sag” at the edge of the primary mirror. Sag is the difference in *z* values between the vertex and edge of the optic, presuming that you have chosen your optical axis to be *z*, as shown in the figure below.

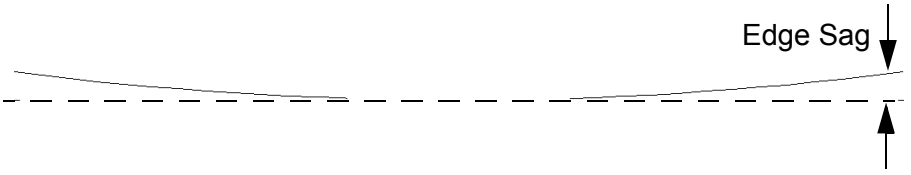


Figure 6.9 Exercise 2: Meaning of the “sag” for the primary mirror.

4 Compare your estimate with the value calculated using the sag formula in the sidebar “The Mathematics of Conics” on page 118.

5 Make two **PLOT FACETS** pictures of the system. One with the default facet setting (5 5), and one with **FACETS 11 11**. View each in the 3D Viewer.

Hints

- We have not specified the sign on the radii of curvature for either mirror. Should they be positive or negative in your coordinate system?
- We have not specified dimensions for the detector. Choose any reasonable values. You can always change the model later if some rays miss.
- Do not forget to rewind the vector file before making new plots, if you intend to view the results in the 3D Viewer.
- When estimating the sag in 3, remember that you already know exactly where the vertex of the primary mirror would be if there were no hole there.

CHAPTER

7

BASIC RAY TRACING CONCEPTS

In this chapter, we set the groundwork for defining sources and tracing rays in ASAP. Learning a few basic concepts will make the process of creating sources and tracing rays much easier to understand.

The ray set needs to have the correct characteristics to allow you to draw accurate conclusions from your work.

Once the system is defined, the next step in any ASAP simulation is the creation of some rays. Creating a source that is appropriate to your problem is just as important as modeling the geometry accurately, and assigning the appropriate optical properties. The ray set that you use needs to have the correct characteristics to allow you to draw accurate conclusions from your work.

Rays and Sources

A ray can simulate the local flow of electromagnetic energy (energy per unit time, or "flux").

The concept of a geometric ray has been important to optics since at least the time of Newton. A ray is often described as the normal to the wavefront of an electromagnetic wave. For our purposes, it is important only to know that a ray can simulate the local flow of electromagnetic energy (energy per unit time, or "flux"). A ray is not a photon, carrying a quantum of energy. A ray represents the continuous delivery of power. As a result, all ray trace results represent a steady-state situation; there is presumed to be no time dependence. As long as we trace enough rays, and as long as the spatial scales in our optical system are large compared to the wavelength of the light they simulate, we can use rays to get highly accurate predictions of the behavior of a real optical system.

An ASAP source is basically a collection of rays that have been created by a single ASAP source command. Our goal is to do this in such a way that we accurately simulate both the spatial and the angular behavior of the source. In other words, the right amount of flux needs to come from the right places on the source, and be pointed in the right directions.

An important point to remember is that it may take more than one ASAP source to simulate what you might think of as your complete light source. For example,

Figure 7.1 shows five rectangular surface emitters being used to simulate the die of a light-emitting diode. While you may think of this as one source, ASAP allows you to model it as five. This flexibility can be extremely useful since we can perform analysis on the various sources separately, and learn more about how the structure of the whole source is affecting performance.

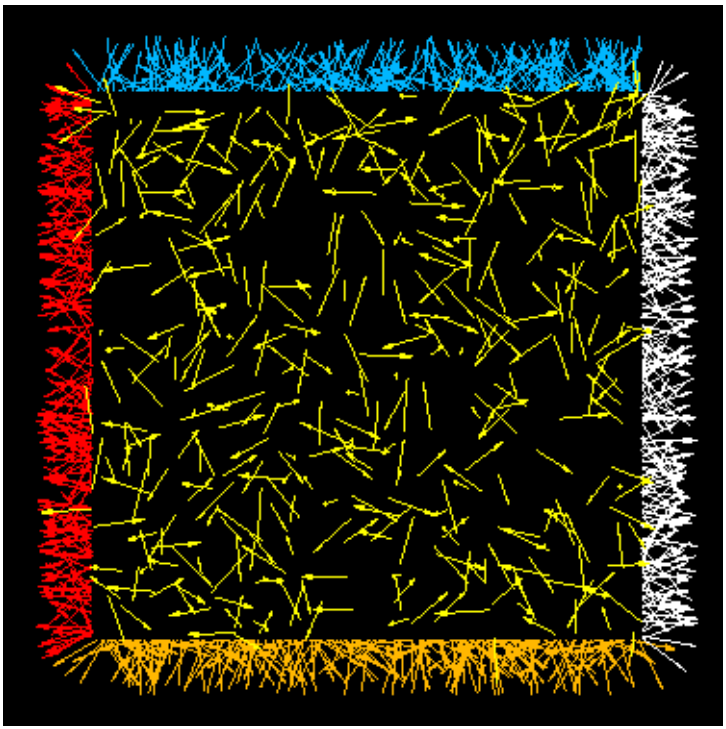


Figure 7.1 At times, several ASAP sources may be needed to model the light in your ASAP simulation. In this case, five emitting rectangles are used to model the die in a light-emitting diode. Each ray is represented by an arrow. The colors in this case are arbitrary, since all five sources have the same wavelength.

Another common situation where more than one ASAP source is used concerns projects involving multiple wavelengths of light. Since each ASAP source can have only one wavelength, a polychromatic source is necessarily modeled by more than one monochromatic ASAP source.

These various ray subsets can be useful to us later in the analysis stage of a project, because ASAP gives us the ability to isolate rays according to their source of origin. This and other isolation methods available in ASAP can often give us important insights into how to improve the performance of our designs.

In some ray tracing programs, defining and tracing rays are done in one step. As soon as the rays are defined, they launch themselves into the system, stopping only when the ray trace is finished.

In some ray tracing programs, defining and tracing rays are done in one step. In ASAP, creating sources and tracing rays are two separate steps.

In ASAP, creating sources and tracing rays are two separate steps. ASAP is capable of modeling sources in considerable detail. After all, we are attempting to generate rays that accurately model the physical properties of the real sources used in our systems. We have to get the rays in the right place, and they need to be pointed in the right direction. Each ray must carry a flux (energy per unit time) appropriate to where it is, and where it is pointed. In many ways, creating ASAP sources is like modeling the geometry. It deserves careful attention, and is important enough to have its own set of graphical verification tools (described in Chapter 9). Only when we are satisfied with both the geometrical and source models do we proceed to trace rays through the system.

How do we go about creating such realistic source models? Do all ASAP users have to be experts on the physics of light sources? Usually they do not. The appropriate physics for incandescent emitters, for example, is built into ASAP. We can model high-intensity discharge lamps (arc lamps) by modifying a simple ray set to match observations, even without understanding the physics of a plasma discharge. Even tools for creating coherent point-source models and Hermite-Gauss laser modes are built into ASAP.

ASAP provides four basic methods for creating sources:

1. Grid sources are used to model simple point sources and parallel rays (point sources at infinity).
2. Emitters allow us to model most types of incoherent extended sources

   Both of these source types allow custom modification of ray flux to match observational data. It is common for the manufacturer of a source, or for other laboratories with the appropriate measuring instruments, to provide basic graphical intensity distributions. You can then modify basic ASAP sources so that they reproduce the observed behavior. This process is called ray apodization.
3. The BRO Source Library includes arc sources, LEDs, CCFLs (cold-cathode florescent lamps), and incandescent sources commonly used in automotive illumination.
4. Custom rays can be imported in text format. If you have an external program that can generate an appropriate ray set, it is easy to read these directly into ASAP from text files.

Only the first two methods are described in this *Primer*. The other topics (including source apodization) are included in the Technical Guide, *Sources*.

ASAP Rays

To understand what is going on during source creation, ray tracing, and analysis, it is important to understand what a ray is in the ASAP context.

In ASAP, rays are points in space with associated directions and flux.

Figure 7.2 shows a ray trace through a simple plano-convex lens. Any textbook on the subject of ray tracing will describe the blue lines with arrowheads on them as "rays". While you may prefer to cling to this concept, you will make much faster progress with ASAP if you take a somewhat different view: *Rays are points in space with associated directions and flux*. The lines in the figure are just the *paths* followed by these points as they progress through an optical system.

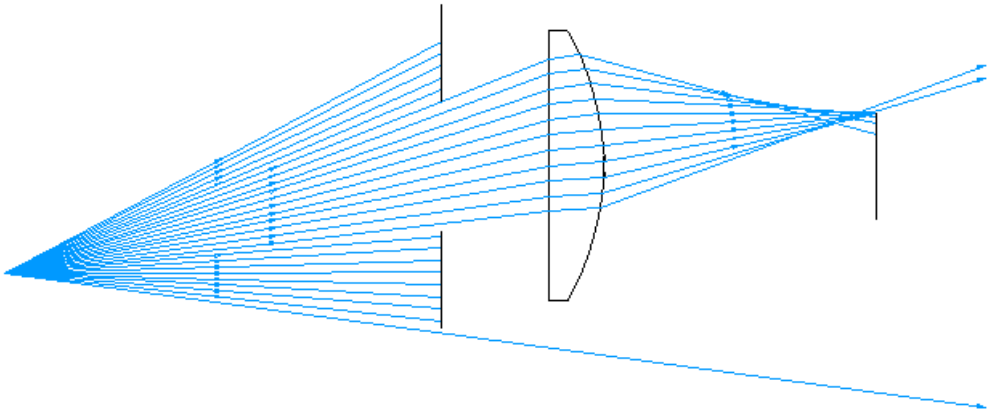


Figure 7.2 A simple fan of rays is shown as being traced through an aperture stop, a singlet lens, and finally to a detector plane. Several basic ASAP ray-tracing concepts are easier to understand if you resist believing that the blue lines in this figure are the rays. In the ASAP context, think of the rays as points in space that move from object to object as appropriate. The blue lines are the *paths* that the rays followed.

For several reasons, it is important to think of an ASAP ray as a point in space with a direction and other attributes. When we create a ray, we have to say where the ray is located initially, as if it were geometry. You will see that you can shift it or rotate it in much the same way you would move any piece of your geometry. This action makes little sense if the ray is a line that does not yet exist, but makes perfect sense if the ray is a point in space with an associated direction vector.

At any given time, every ASAP ray is always in just one place. In general, a given ray does not have a memory of everywhere it has been.

Later, you will see that it is also these points upon which we perform the analysis. We can do the analysis at any time—before the ray trace, part way through a trace, and at the end of the process, when all the rays have progressed as far as they can. In ASAP, it always makes sense to ask, "Where are the rays now?" The lines representing the ray paths in Figure 7.2 may span much of the system, but *at any given time, every ASAP ray is always in just one place*. In general, a given ray does not have a memory of everywhere it has been. The lines in the figure were drawn during the ray trace as the ray moved through the system, but ASAP does not keep (at least by default) all the information necessary to draw that picture again.

ASAP describes a ray in terms of three Cartesian coordinates, a direction vector, and a flux value.

So what is an ASAP ray? ASAP describes it in terms of three Cartesian coordinates (*x, y,* and *z*) in the same global coordinate system you used to describe your geometry. The ray also has a direction vector associated with it. Within a homogeneous media (for example, in a vacuum, inside a glass or plastic object), the rays move along their direction vectors. At an interface between media, or on a reflecting surface, the direction vector changes. It does this based on the information given in the **INTERFACE** commands assigned to each object. It also has a flux value that may change as the ray progresses, again depending on the interfaces assigned to the objects. These are not the only features of a ray that are tracked by ASAP, but they are the most fundamental. The sidebar located at the end of this chapter, "ASAP Ray Details" on page 133, lists other information that is maintained in the ASAP ray table.

Rays Belong to Objects

Another important ASAP ray-tracing concept is that rays always belong to objects. You will see shortly that it is possible to interrupt a ray trace (or any other kernel process) that is in progress by clicking the **Abort** button in the main ASAP Toolbar. If you were to do this, you would never find a ray "in transit" between two objects. That is not how ASAP traces rays. When you first create rays, they are assigned to a fictitious object called "Object 0". When the ray trace begins, ASAP uses an elaborate algorithm to quickly determine which object is next along the ray's direction vector. Then, a careful, detailed calculation is performed to determine the exact coordinates of the intersection. Once this intersection point is determined, the ray table is updated with new coordinates for this particular ray. The ray was, in effect, *instantaneously* moved from one object to another. ASAP drew the blue lines in Figure 7.2 on page 128 by drawing a line between the old and new ray coordinates.

After the rays are traced, they always belong to one of the objects in your system, whichever one they intersected last. Some may still belong to Object 0, if they never intersected anything.

Figure 7.3 on page 130 shows where the rays reside after the ray trace we performed in the previous figure.

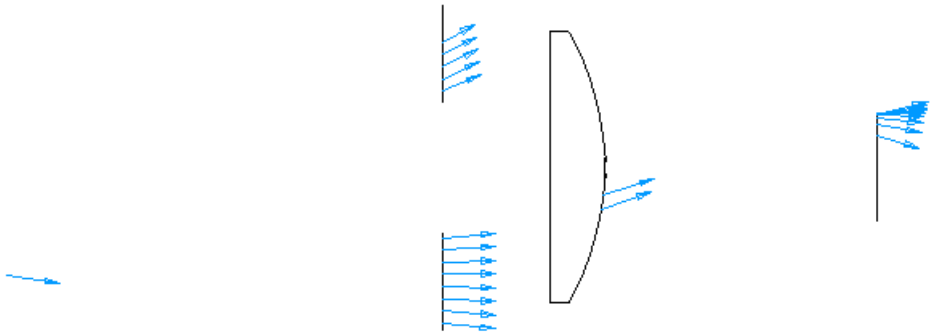


Figure 7.3 The blue arrows show the final location of every ray—*after* the ray trace shown in Figure 7.2. The directions of the arrows show their final direction vector, with a length proportional to the flux of the ray. After the ray trace, the rays remain on the last object they intersected. Even though the previous figure drew arrows showing three of these rays leaving the system, they actually do not leave. In this case, one remained on the source (Object 0), and two remained on the back of the lens. The rest of the rays stopped on surfaces with absorbing interfaces: 13 rays stuck to the aperture, and 9 made it to the detector.

Some of the rays are now on the aperture, some on the back of the lens, and some are on the detector. One ray is still at its original location where the source was defined. The 13 rays on the aperture stopped because that object had an absorbing interface assigned to it (`INTERFACE COATING ABSORB`…). This interface, you may recall, causes the rays to stop on that object with all their flux intact. They now "belong" to that object. The nine rays on the detector stopped for the same reason.

What is the explanation for the two rays "stuck" on the back of the lens? This is a transparent object, so they should have left that surface and traced on. The answer is that there were no other objects along their direction vectors. ASAP could have removed them from the ray list as "lost light", but this action would have given us no way to investigate them. By leaving them on the last object they intersected, we can perform a detailed analysis to learn what went wrong. In this case, the detector plane was not big enough to intercept them.

In the same way, the single ray still located at the position of the source missed everything, and was never moved. It still belongs to "Object 0".

Rays intersect objects in the physically correct order

It should not be surprising that you do not need to define objects in a specific order to get the rays to behave correctly as they move through the system. This is the essence of the ASAP style of physical ray tracing. The rays will find their own way through the system, encounter objects in whatever order is physically appropriate. The order in which you defined the objects is completely irrelevant when you are performing a non-sequential ray trace.

Sources can be traced only once

One last important ray trace concept: you can trace a source of rays only once. Many new users expect that if they ask the program to trace again, ASAP will know enough to "load up" the original rays and repeat the trace from the beginning. This approach would be useful when we have made some changes or corrections to the source or geometry definitions, and want to trace the same rays again. Instead, we have to discard the old rays and run the source-definition commands again to create a new (identical) set of rays back at the original source location. Only then can we repeat the trace from the beginning.

This important feature will become clearer as you gain experience with ASAP. You will also grow to appreciate it, and count it among the strengths of ASAP. The program makes few assumptions about what we are doing, and what we intend to do next. You are always in complete control of the process. You could, for example, add new objects to the system *after* a ray trace (without clicking the **End** button), and ask ASAP to trace again. The original rays will carry on from their last positions, intersecting the new objects if they can do so. You can also define a new, different source of rays, and trace them through the system without discarding the old rays. When you begin your analysis, rays from both sources can be included. This gives you the flexibility needed to interactively solve problems caused by errors in your model, or shortcomings of the design you are analyzing.

Still, the flexibility that ASAP offers us is sometimes a source of confusion to new users. The key to understanding and taking advantage of this important ASAP feature is to remain aware of the current state of ASAP—both the system and ray databases. They are always just the way you left them after the last operation. The program is waiting to be told what to do next, making no attempt to read your mind.

Summary

In this chapter, we have introduced five important ray-tracing concepts:

1. Creating sources and tracing rays are two separate steps in ASAP. While some ray-trace software launch the rays immediately after creation, ASAP waits, allowing you to apply analysis and verification tools to help satisfy yourself that the source model is correct.
2. ASAP rays are best thought of as points in space with associated directions and flux. It always makes sense to ask, "Where is the ray (the point) now?"
3. Rays belong to objects. When they are created, they are assigned to a fictitious "Object 0". As they trace, they are transferred from object to object. When the trace is finished, they belong to the last object they touched.

4. Rays intersect objects in the physically correct order. You do not need to worry about the order in which you define objects. This has no effect on the ray trace.
5. ASAP rays can be traced only once from their initial source positions. If you want to see the trace again, you must discard the old rays, define new ones, and trace again.

If any of these points remain confusing, do not worry. As you gain experience with creating sources and tracing rays, these concepts and their significance will become increasingly clear. We will repeat them often in forthcoming chapters.

ASAP Ray Details

In ASAP, a ray is an entry in a table. The table is stored in binary form in a disk file named virtual.pgs, the "virtual pages" file. This file can become extremely large if you are tracing millions of rays. In fact, the number of rays that can be traced by ASAP depends on how large this file is able to get before it fills all available disk space on the drive containing your ASAP working directory. Pieces (or "pages") of this file are swapped in and out of memory as required during ray tracing and analysis — hence the name, "pages".

The virtual.pgs file contains highly compressed information about each ray, from which the following quantities can be derived:

- Global X,Y,Z coordinates of the ray
- Direction cosines of the ray
- Wavelength
- Flux
- Medium in which the ray is currently located
- Object upon which the ray currently resides
- Previous object
- Total number of surfaces the ray has hit
- Source number from which the ray originated
- Optical path length for the ray
- Parametric coordinates of ray position (when current object is an **EDGE** entity)
- Number of the ray from which this ray was split (its parent)
- Number of times the ray has been scattered
- Number of times ray has been split (due to Fresnel splitting, grating orders, or Objects at which the ray and all progenitors split.

ASAP tracks additional information when it is being used to trace coherent beams, or when polarized ray or beam traces are being performed. The list above is only the basic data for unpolarized geometric rays. You can see by studying this list that it is not possible from this information alone to reconstruct the complete history of a ray. ASAP keeps track of only how many objects the ray intersected, the object it is on now, and the object it intersected previous to this. While it is possible to keep the entire history, this can produce a prohibitively large amount of information for large ray traces. (See **SAVE** *in the online help or the ASAP Reference Guide for more about this option.)*

When ray splitting occurs due to scattering, Fresnel reflections, diffractive orders or birefringence, ASAP keeps details about the ray's parents, and other progenitors. This sort of information is often invaluable when we try to sort out the paths taken by rays through the system. Even though we don't have a detailed history, this information — along with current object, previous object, and the total number of hits — is usually sufficient to sort the rays into unique paths.

It is rare that you will need to know such detailed information about a single ray. Most ASAP analysis commands look at these data for many rays to draw general conclusions about the behavior of your system. You will, however, see how to access this information in Chapter 9, Verifying Sources.

CHAPTER

8

GRID SOURCES

We commonly use the grid source family in both imaging and non-imaging problems, when it is useful to see the response of a system to a simple, ideal point source.

In this chapter, we introduce the ASAP grid source. This source allows us to define a set of rays that generally has a regular distribution in space and direction. The grid source type is suited for modeling point sources, either at finite or infinite distances. In other words, grids can be used to create sets of converging, diverging, or parallel rays. We commonly use this source family in both imaging and non-imaging problems, when it is useful to see the response of a system to a simple, ideal point source. For example, we might want to see how ghost images or stray-light problems affect the point-spread function of an imaging system. These grid sources are also the basis for modeling plane and spherical waves (see the Technical Guide, *Wave Optics in ASAP*).

In the previous chapter, we noted that a single ASAP source is monochromatic—all the rays in the source have the same wavelength. The wavelength assigned to the source is the same as that specified in the **WAVELENGTH** command, near the top of the Builder file. While it is not difficult to simulate broadband light using multiple source models in ASAP, that topic is addressed in the Technical Guide, *Sources*. All the examples in this *Primer* presume a single wavelength and no wavelength-dependent behavior in the system.

Steps to Defining a Source

Defining a source in ASAP requires that we specify two things:

1. The initial location of each ray.
2. The initial direction of each ray.

If necessary, we can also set the absolute flux of the rays in a source, creating an optional third specification:

3 The flux of each ray.

ASAP provides high-level commands that automatically assign appropriate values to the individual rays, based on a general description of the source.

We, of course, do not have to do this ray by ray. If we define millions of rays (which is not uncommon when simulating illumination systems), this task would be tedious. ASAP provides high-level commands that automatically assign appropriate values to the individual rays, based on a general description of the source.

For grid sources, we can accomplish the entire task with just three commands, one for each of the steps above. Figure 8.1 shows a typical example of these commands in a Builder file.

*	Type	Option	Axis	Position	X Min	X Max	Y Min	Y Max	Number of X Rays	Number of Y Rays	Aperture
ENT	Grid	Rect	Z	-10	-1	1	-1	1	10	10	10
MOD	Source	Direction		0	0	1					
CMD	Flux	Total	TOTAL	59							

Figure 8.1 Three **Builder** lines are needed to completely specify an ASAP grid source. We are initializing the fundamental data in the ray table described in the last chapter: ray position coordinates, directions, and flux.

See Chapter 8 Appendix, "Script 8-1" on page 157.

The **Grid** line tells ASAP where we want the rays to be located, and the **Source** line allows ASAP to calculate and assign directions to them. The **Flux** line (if present) overrides the default flux values to give us control over the total ray flux in this source. We will describe each of these steps in detail in the following sections.

We will use the ASAP Builder to create these sources. The definition process used is not different from the way we created system and geometry definitions in the Builder file.

> ***Note:*** Your source definitions can be either in the same Builder file used to define your geometry, or in their own Builder file. You may use as many Builder files as you like, but only the first one you run (usually your system and geometry definitions) should specify units and other definitions that are common to the whole project. For now, it is probably best to keep everything in one file. In Chapter 12, "More About the ASAP Landscape", we will describe how to use the **Project Workspace** option to manage multiple files in a larger ASAP project.

Rectangular Grid of Parallel Rays

The simplest and perhaps the most common point-source model in ASAP is a grid of parallel rays, simulating a point source at infinity. ASAP allows us to make grids with rectangular or elliptical boundaries, coming from any direction. We will start with the simplest case, a rectangular grid parallel to one of the global coordinate axes. This approach would be appropriate for a star or any other well-collimated source entering a system straight down its optical axis. We need to learn two commands to accomplish this: `GRID RECT` and `SOURCE DIR`.

The goal of the `GRID RECT` command is to create a set of rays whose initial location is in the shape of a regular grid with rectangular boundaries. We normally use this command any time we need to send rays into a system with a square or rectangular entrance aperture (see Figure 8.2).

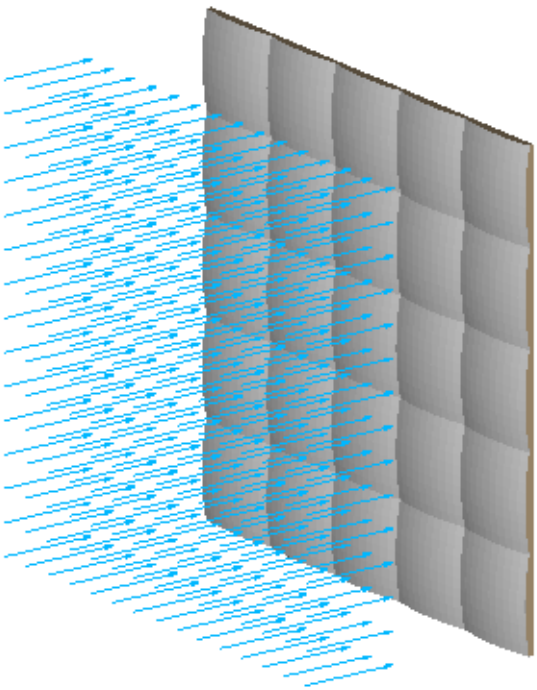


Figure 8.2 A rectangular grid can be used to couple rays into an optical system limited by a square or rectangular shape, like this array of pillow lenses. Note that these rays (in blue) already have direction vectors associated with them, which were added for clarity. The `GRID` command establishes only the location of the rays.

All the `GRID` commands are accessed within the Builder in the usual way: double-click in the first available cell of a new Builder line. The familiar menu appears. This time, we are looking for **Grids> Grid**, and select **Rect** from the

drop-down box in the **Options** column. All dimensions and positions used in the GRID definitions are in your system units, as defined by the UNITS command.

A sample Builder line and the resulting grid of initial ray positions is shown in Figure 8.3.

*	Type	Option	Axis	Position	X Min	X Max	Y Min	Y Max	Number of X Rays	Number of Y Rays	Aperture
ENT	Grid	Rect	Z	-10	-1	1	-1	1	11	11	10

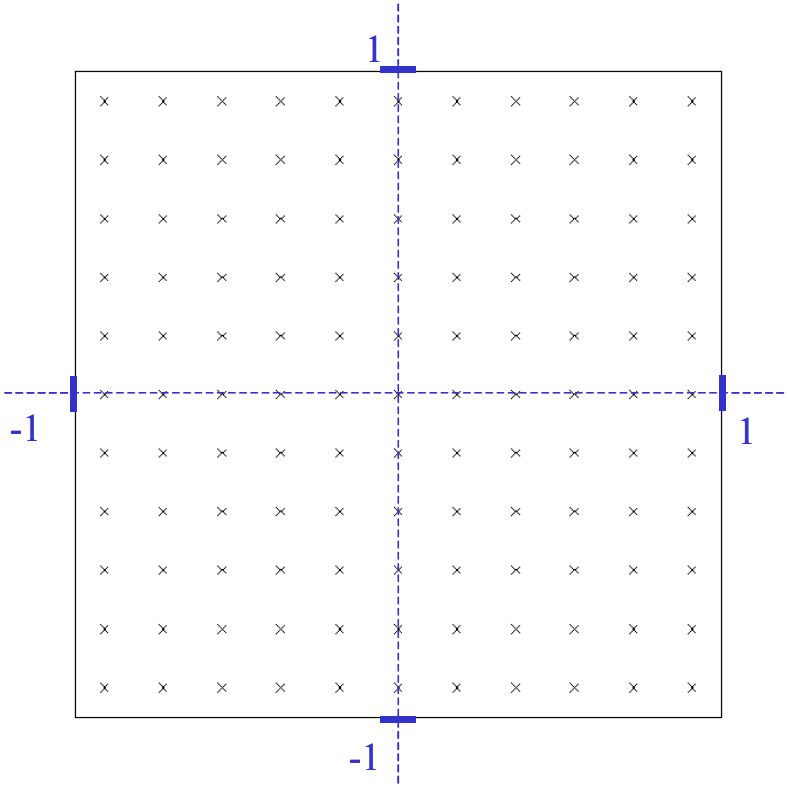
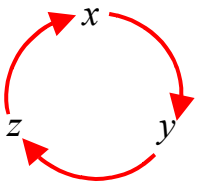


Figure 8.3 This example of a GRID RECT is in the z=0 plane. The X symbols mark the initial positions of the rays in this source. In this case, the source is defined between +/- in both x and y. There are 11X11 rays in the grid.

The first choice we have in the GRID command is the **Axis** specification, which is the plane in which the rays will be created. The convention here is the same as we used when defining geometry: we are designating the plane that is normal to the specified axis. The *z*-axis grid in the example generates rays with different *x* and *y* values, but constant *z* coordinates. Next, the **Position** parameter allows us to say where along this axis we want this plane to be located. In the example, we have placed the rays in the $z = 0$ plane. Then, we specify the minimum and maximum extent of the grid in each of the other two axes.

We would normally use values that match the semidiameter of our entrance aperture. The convention here (as in most of ASAP) is the right-handed rule, as illustrated in the drawing and table below. If we specify the *z* axis, the major axis becomes *x*, and the minor axis is *y*.

Table 8.1 Right-handed rule (on left) and Grid Axes (on right)



Grid Plane	Major Axis	Minor Axis
Z	X	Y
X	Y	Z
Y	Z	X

The next parameters we must set are the number of rays along the major and minor axes. As Figure 8.3 on page 138 shows, ASAP does not actually place rays at the extremes of the rectangle you specify. What ASAP is really doing is dividing the rectangular area into sub-rectangles, based on the number of rays specified and the number of rays along each axis. This sub-rectangle size defines the local grid spacing. A ray is placed at the center of each sub-rectangle.

> ***Note:*** If you want a ray at the center of the distribution, be sure to select an odd number of rays. Since this is often the optical axis of your system, having a ray right down the middle is usually desirable. That is why we changed the defaults from 10 to 11 rays.

The remaining parameters in the `GRID RECT` command are optional. The **Aperture** parameter allows you to specify a rectangular "obscuration" in the center of the ray distribution. No rays will be created here. The value entered in this cell is the ratio of the obscuration size to the overall size of the grid. If you select a value of 0.5, rays will be missing from an area with dimensions that are half of the overall dimension of the grid.

If the **Random** option is selected (by double-clicking that cell and selecting **Random**), ASAP randomizes the location of the rays over a region described by the last parameter. This **Region** parameter defines the maximum size of the box over which each ray's position is permitted to roam. The box size is equal to the **Region** parameter times the local grid spacing. The box is centered on the position the ray would have if the **Region** parameter were 0. A value of **1** therefore, allows each ray to be displaced as far as the edge of the sub-rectangle (the local grid spacing), while a value of **2** allows each ray to be displaced as far as the center of the neighboring ray's box. Figure 8.4 shows the result of varying the **Region** parameter on the placement of the rays.

> ***Note:*** If you want to randomize the ray positions, you must specify a **Region** value in the last cell of the command. The default value is zero, resulting in no randomization at all.

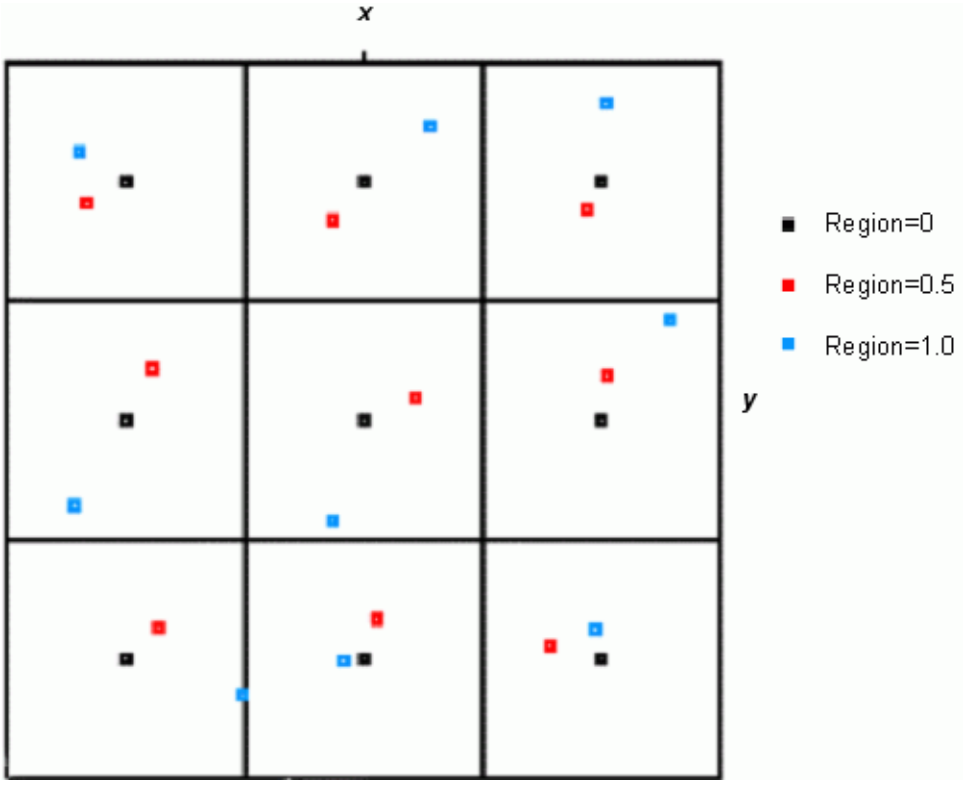


Figure 8.4 Randomizing the ray position within the sub-rectangle, as defined by the local grid spacing by means of the **Region** parameter.

Once the ray positions are set, the next step is giving a direction to each ray in the grid. There is no default ray direction, so ray creation is not complete until we perform this task.

ASAP provides us with **Grids> Source** with the **Direction** option to make parallel rays. Stated another way, all rays in the source will have exactly the same direction vectors. The only parameters we need to set in this command are **Vector A**, **Vector B**, and **Vector C**. These are the direction cosines for the rays.

For anyone unfamiliar with direction cosines, they are the cosine of the angle that the unit direction vector makes with the each of the Cartesian axes, *X*, *Y,* and *Z (*see the sidebar, “Direction Cosines” on page 156).

If we want a grid of rays pointed along the +*X* axis, the direction cosines are (1, 0, 0). For rays pointing in the opposite (–*x*) direction, choose (–1, 0, 0). For the systems we have been building so far (the Cooke triplet or the Cassegrain telescope), we want rays parallel to the +*Z* axis (Figure 8.5).

*	Type	Cmd	Option	Vector A	Vector B	Vector C	Vector A'	Vector B'	Vector C'	Vector A''
ENT	Grid	Rect	Z	-10	-1	1	-1	1	11	11
MOD	Source	Direction		0	0	1				

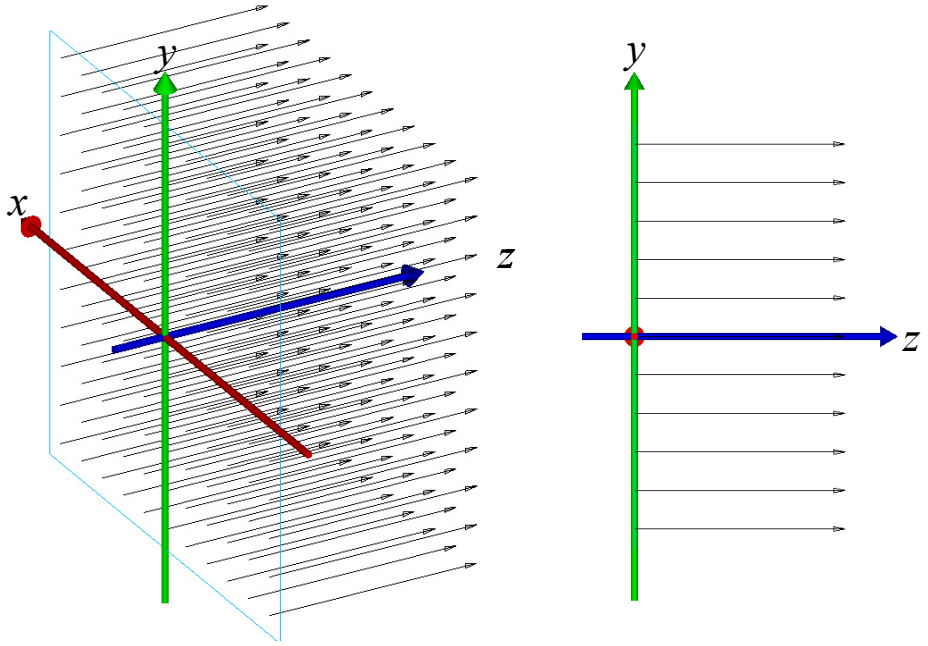


Figure 8.5 The `SOURCE DIRECTION` command is used to make parallel ray grids. The blue rectangle shows the location of the imaginary plane on which the rays are located. The black arrows show the direction vectors.

These rays have direction cosines (0, 0, 1). These systems, of course, do not have square (or rectangular) symmetry, however. Before we can send rays into those models, we need to learn other grid options that are more appropriate for circular apertures.

GRID ELLIPTIC

The **Grids> Grid** with the **Elliptic** option is similar to **Grids> Grid** with the **Rect** option, but more appropriate for use with a circular or elliptical entrance aperture (Figure 8.6).

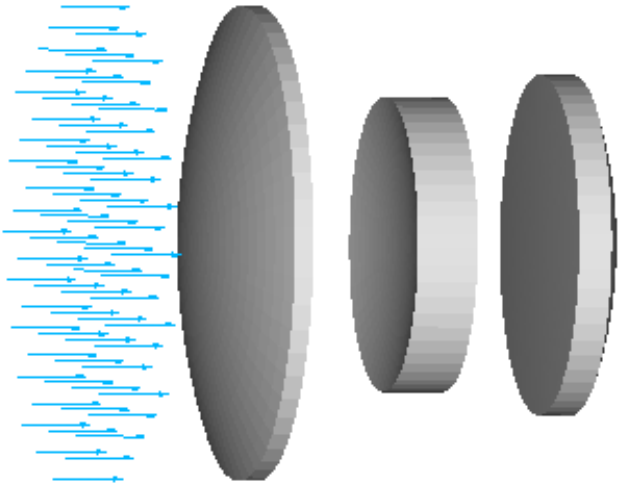


Figure 8.6 An “elliptic” grid is often the correct choice when coupling rays into an optical system with rotational symmetry. The rays are the same as those used in the rectangular grid, but the rays outside of the inscribed circle (or ellipse) are eliminated.

Using exactly the same parameters as we specified for `GRID RECT`, ASAP creates the same set of rays, and then discards any that are outside of the ellipse boundary (Figure 8.7 on page 143).

When an aperture parameter is specified, the “hole” in the ray distribution is elliptical rather than rectangular in this case.

*	Type	Option	Axis	Position	X Min	X Max	Y Min	Y Max	Number of X Rays	Number of Y Rays
ENT	Grid	Elliptic	Z	-10	-1	1	-1	1	11	11

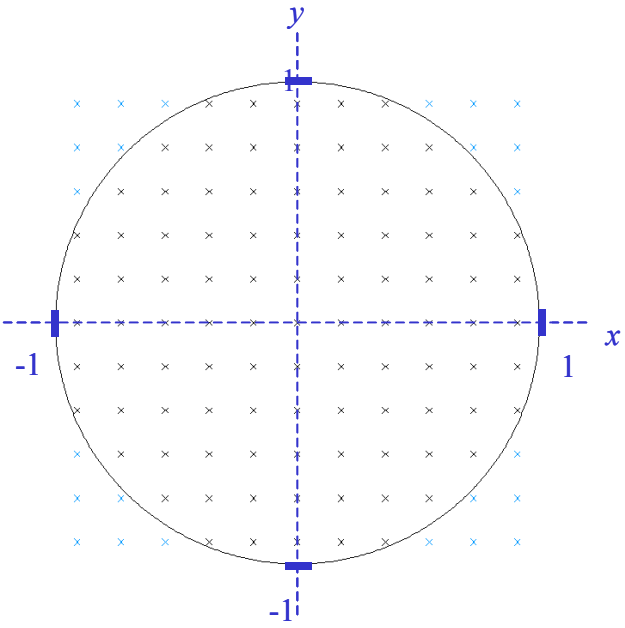


Figure 8.7 The `GRID ELLIPTIC` command generates exactly the same set of rays as `GRID RECT`, but the rays outside the elliptical boundary (shown in blue here) are discarded.

See Chapter 8 Appendix, “Script 8-2” on page 157.

GRID POLAR

The polar grid is an alternative for making grids with circular boundaries. This version of GRID places the rays on concentric rings, rather than fixed spacing in a rectangular coordinate system.

A typical set of rays generated by **Grids> Polar** is shown in Figure 8.8 on page 144.

*	Type	Option	Option	Axis	Location	Radial Max	Rings	Random
EMT	Grid	Polar	Short Form	Z	-10	1	7	

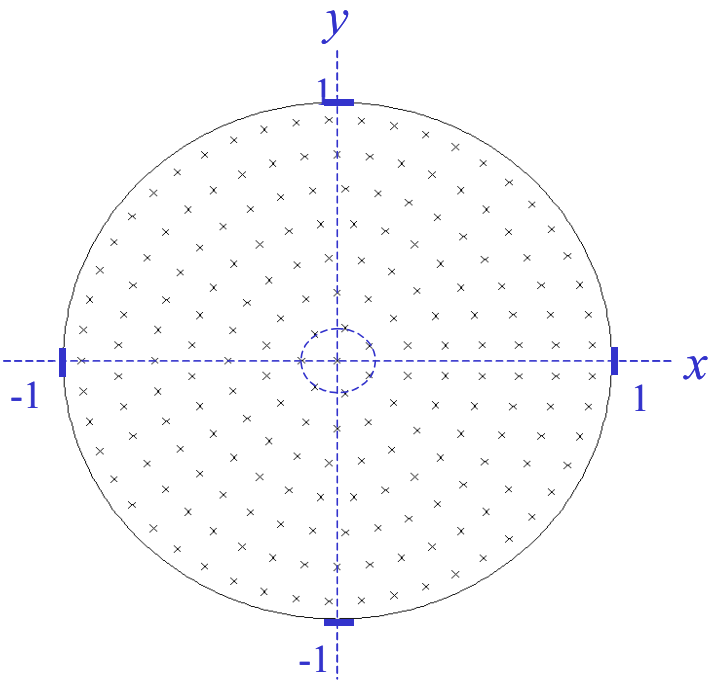


Figure 8.8 The GRID POLAR command allows us to generate a set of rays on concentric rings, which is useful for sampling geometry with radially symmetric structures, like Fresnel lenses. We specify the number of rays in the first (innermost) ring, and the total number of rings (not counting the central ray, if one is present).

See Chapter 8 Appendix, "Script 8-3" on page 157.

GRID POLAR is used somewhat less often than GRID ELLIPTIC, perhaps because the additional flexibility it offers leads to slightly more complexity in the command syntax. It is, however, useful when sampling optical components with radial symmetry, like circular Fresnel lenses (Figure 8.9 on page 145).

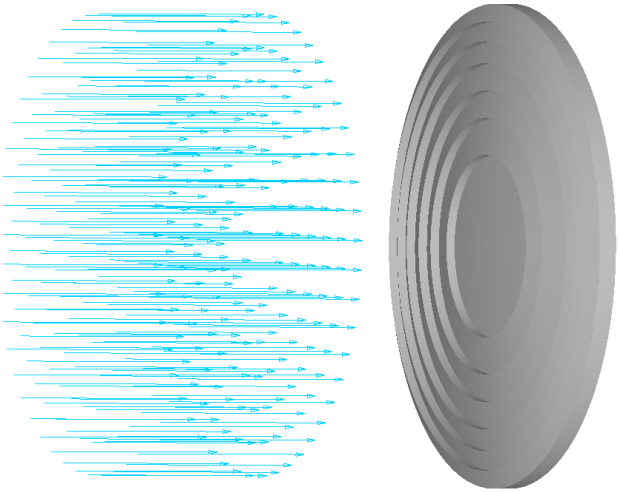


Figure 8.9 The **GRID POLAR** command creates rays in concentric, circular rings. This command may be a better choice than **GRID ELLIPTIC** when sampling optics with circularly symmetric structures, like this Fresnel lens.

The **Axis** and **Position** parameters are defined in exactly the same way as they were for the **Grids** with **Rect** and **Grids** with **Elliptic**. The next two parameters, **Radial Min** and **Radial Max**, define the range of radial distances over which rays will be defined. If **Radial Min** is zero, a ray will appear at the center. To avoid creating a central ray, you can set the **Radial Min** parameter to a small but finite value, like the default 0.01.

The next pair allows us to define the range of angles over which rays will be defined. For example, 0 to 180 produce a semicircle of rays. The typical values are 0 and 360, a full polar grid.

The next three parameters, **Radial Rays**, **Ring 0 Rays** and **Weight** set the actual ray positions. **Radial Rays** defines the number of rings in the source, *excluding the central ray* (if present). The **Ring 0 Rays** parameter sets the number of rays in the first ring, again ignoring the central ray if one is present. The number of rays and relative spacing of subsequent rings is controlled by the **Weight** parameter. By far the most common usage is to leave the weight at the default value of **1**. This value causes the rings to be evenly spaced with an increasing number of rays in each ring. In this case, each ray represents an equal area in the grid. Other weights will still maintain constant area per ray, but the rings will not be evenly spaced. A weight of 0.5 will generate straight radial “spokes” with a constant number of rays per ring, with steadily decreasing ring separation toward the outside of the grid area.

Off-Axis Parallel Rays

Any of these grid definitions can be combined with a variety of other variations on the SOURCE command to model other types of point sources. The most obvious of these uses **Grids> Gridd> Source** and select **Direction** from the drop-down menu to model an off-axis point source at infinity. Figure 8.10 shows a grid of rays making a 30° angle to the *z* axis in the *y*-*z* plane. It also illustrates the use of the trigonometric functions in the Builder. These functions will operate on arguments expressed in either degrees or radians.

*	Type	Cmd	Option	Vector A	Vector B	Vector C	Vector A'	Vector B'	Vector C'	Vector A''	Vector B''
ENT	Grid	Rect	Z	-10	-1	1	-1	1	11	11	10
MOD	Source	Direction		0	SIN[30]	COS[30]					

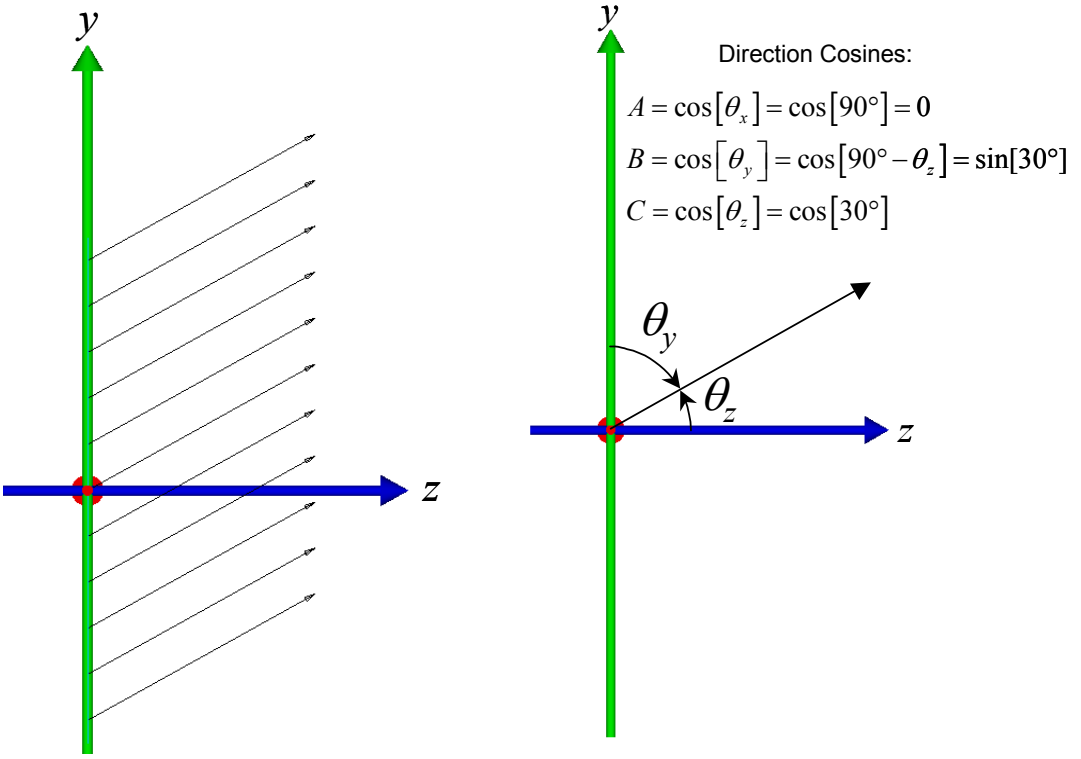


Figure 8.10 Off-axis parallel sources are also made with the SOURCE DIRECTION command. If the optical system is aligned down the Z axis, this source could be used to study the response to a point source 30 degrees off the optical axis.

If you want to express your angles in degrees, the argument should be enclosed in square brackets as shown in the **SOURCE** command that appears in Figure 8.10. Parentheses are used for radians.

Sometimes, it is more convenient to express directions in terms of spherical angles rather than direction cosines. Once a polar axis is defined, any direction in space (or point on the unit direction sphere) can be specified by two angles (Figure 8.11).

*	Type	Cmd	Option	Vector A	Vector B	Vector C	Vector A'	Vector B'	Vector C'	Vector A''	Vector B''
EMT	Grid	Rect	Z	-10	-1	1	-1	1	11	11	10
MOD	Source	Direction		30	90	1					

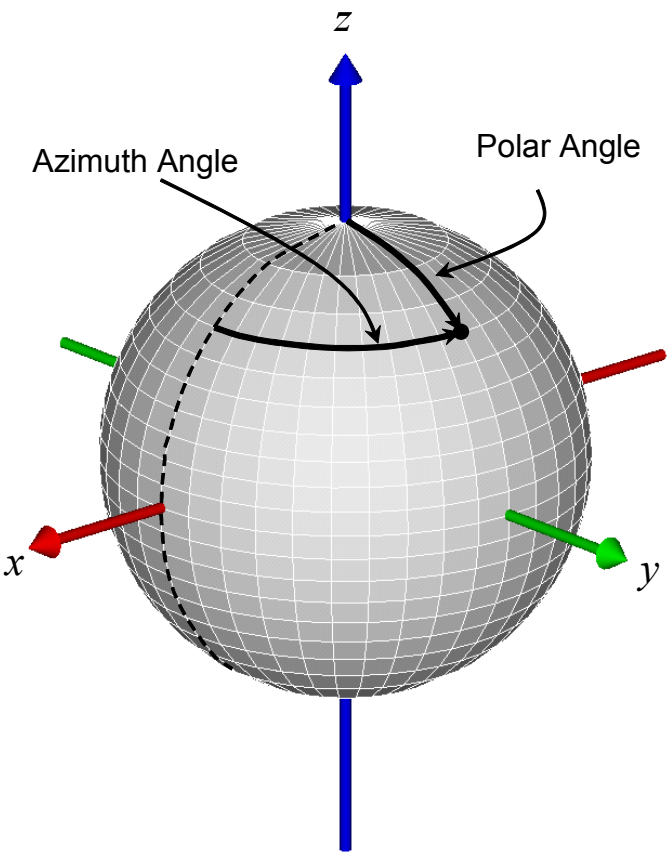


Figure 8.11 As an alternative to direction cosines, ASAP generally accepts directions in terms of a polar angle and azimuth angle (in degrees) relative to a specified polar axis. ASAP interprets directions in this way any time the Vector A parameter is an uppercase X, Y, or Z, rather than a numerical value. The Builder lines at the top of the figure make exactly the same source as shown in Figure 8.9.

See Chapter 8 Appendix, "Script 8-4"and "Script 8-5" on page 158.

The first angle is the polar angle (sometimes called the *zenith distance*), measured down from the polar axis. It can take on values between only 0° and 180°. The second angle (azimuth) is measured around the polar axis in a right-handed sense. If you prefer to specify directions in this way, use the **Vector A** cell to specify the polar axis of your coordinate system (either **X**, **Y**, or **Z**). Then **Vector B** becomes the polar angle, and **Vector C** is the angle around the pole, both measured in degrees. ASAP automatically converts these spherical angles to direction cosines internally.

> ***Note:*** If the polar axis is defined to be the *Z* axis, the azimuth angle is measured from the positive *x* axis toward the positive *y* axis. The other two coordinate systems are defined in rotating, right-side order, as shown in the following table. An azimuth value of 90 will place the direction vector in the Y-Z plane, as illustrated in Figure 8.11 on page 148.

Table 8.2 Polar Axis

For Polar Axis,	measure azimuth from:	toward:
X	Y axis	Z axis
Y	Z axis	X axis
Z	X axis	Y axis

Source Position

What occurs if we want to model a point source at a position other than infinity? The rays will no longer be parallel, but will diverge from a specified point. We could specify the point, and have ASAP generate a set of rays radiating in all directions from that point. This approach is often not efficient, however, since we rarely have an optical system that will capture all these rays. We are interested in only the ones that couple into our optical system. (If you do need such a source, an emitting spheroid with zero dimensions would be a better choice for modeling this than a grid source. See Chapter 16, "Extended Sources".)

It is possible, however, to define a grid of rays close to the entrance aperture of a system where all the rays have directions *as though* they had come from a specific point. This situation is shown in Figure 8.12 on page 150. Now we are able to make a grid of rays of just the right size to couple into the system. We don't waste time tracing rays that will not contribute to the final results.

*	Type									
ENT	Grid	Elliptic	Z	-5	-15	15	-15	15	1	9
MOD	Source	Position	0	0	-66					

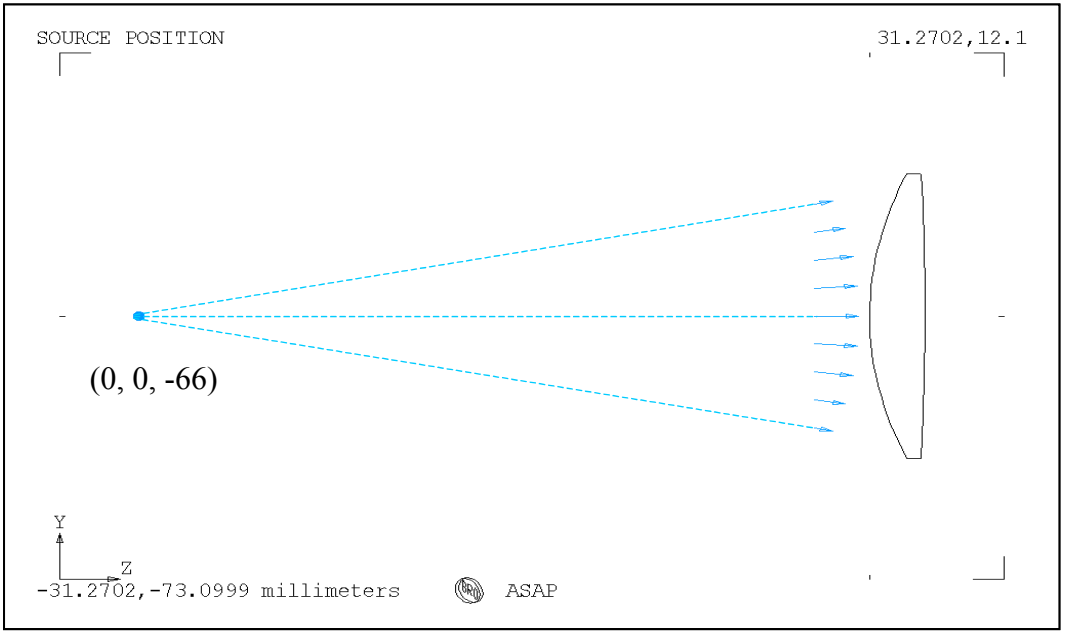


Figure 8.12 A grid of rays can be created directly in front of an optical element, and given directions so that the rays appear to have come from a specified point. This process is done with the **SOURCE POSITION** command. In this case, the rays are diverging from a point at the front focal point of the lens. If we trace these rays, they should become collimated after passage through the lens.

You need to know that the rays are not actually traced from this point. They are created in the plane specified by the **GRID** command, and given directions radiating from that point in space. ASAP does not modify the directions of the rays to compensate for any optical elements that may be in between the grid of rays, and the point.

Note: Occasionally, we might want the rays to actually start at the specified point. This might be necessary if there are additional optical surfaces between the point and the grid that will change their directions before reaching the position of the grid. You might also need to do this only to plot the actual ray paths through space to make an illustration. ASAP provides the **MOVE** command to actually move the rays to a point. It is available as a source modifier in the Builder, as well as from the **Rays** menu and the **Analysis** menu. We will discuss **MOVE** in greater detail in Chapter 15, “Repositioning Geometry and Rays”.

Source Focus

The **SOURCE FOCUS** command is similar to **SOURCE POSITION**, but now the rays are converging toward a specific point in front of them, rather than diverging away from a point to the rear. This convergence is illustrated in Figure 8.13. Again, please note that ASAP does not compensate for optical surfaces between the ray grid and the specified point. The rays are simply given trajectories directed toward the point in space.

*	Type	Cmd	X	Y	Z	X'	Y'	Z'	X''	Y''
EMT	Grid	Elliptic	Z	8	-11	11	-11	11	1	9
MOD	Source	Focus	0	0	45.79032					

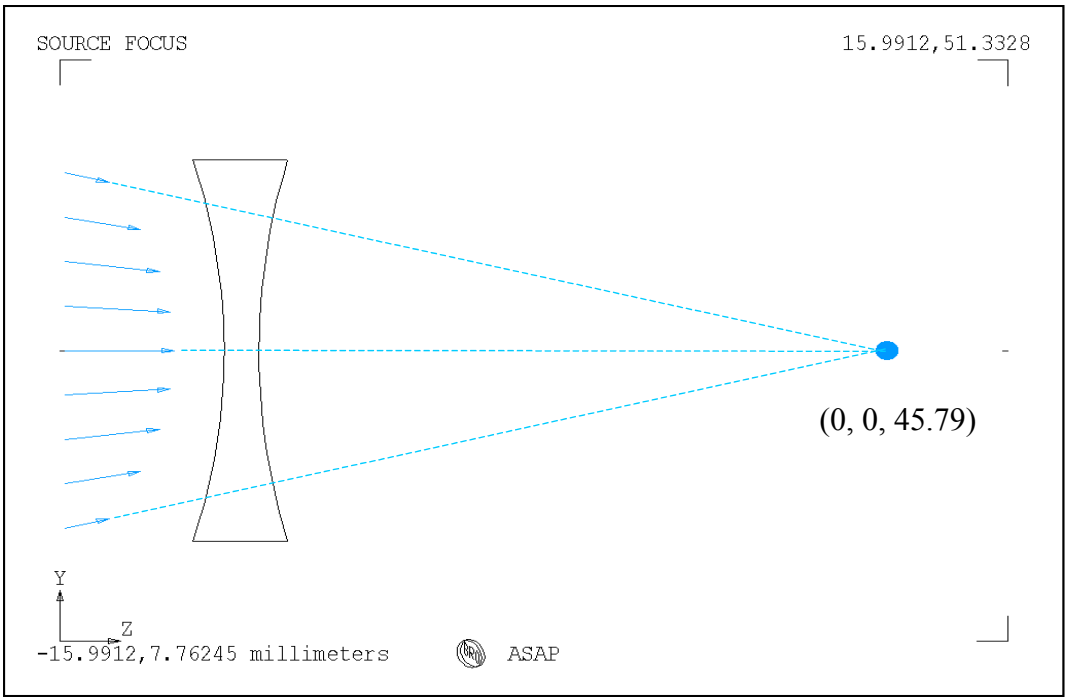


Figure 8.13 A grid of rays can be created directly in front of an optical element, and given directions so that the rays appear to be converging toward a specified point. This process is done with the **SOURCE FOCUS** command. In this case, we have directed the rays toward the virtual focus of a negative lens. When the rays are actually traced through the lens, they should become collimated.

Setting Ray Flux

One last optional step remains to define an ASAP source, and that is setting the total flux. Situations may arise where you do not care about total flux; you may be interested in only a relative look at illumination uniformity or power

distribution in an image. But if your task is to determine an absolute power scattered into the image plane of a telescope or camera system, setting the absolute flux of the source becomes essential. Can the stray light be neglected given the expected flux of your in-field sources, or will you have to make design modifications to meet the performance requirements of the system?

All ASAP sources have default rules for assigning the initial flux of the rays. For example, a set of parallel rays created with the **GRID** and **SOURCE DIRECTION** commands will be scaled so that the source has approximately unit irradiance (unit flux per unit area). A point source located a finite distance away is presumed to have unit irradiance on the surface of a sphere. Since the rays are defined in a plane and not on a sphere, ASAP will adjust the individual ray fluxes automatically to account for both projection effects and the inverse square distance law. ASAP is doing the best it can to scale the flux in a way that is appropriate to the physics of the situation. We are left with only the task of assigning absolute flux to the source as a whole.

> ***Note:*** One common error made by new ASAP users (and more than a few of the old ones) is to assume that changing the number of rays defined within a given ASAP source will change the total flux. This is not true. Recall that for **GRID** sources, ASAP is adjusting the individual ray fluxes to maintain *unit flux per unit area* in a plane or on the surface of a sphere. If two sources defined using the **GRID** and **SOURCE DIRECTION** commands are specified to cover the same total areas, changing the number of rays in the source will change only the flux of the individual rays, but will have no effect whatever on the total default flux of the source.

ASAP provides us with the **FLUX TOTAL** command to set the flux of rays and sources. The most common version of the command is found in the Builder by double clicking in a left-most cell, and selecting **Ray Control> Ray Modifiers> Flux Total**. The command line appears as shown in Figure 8.14.

*	Type	Cmd	Cmd	Factor	Cmd
CMD	Flux	Total	TOTAL	100	

Figure 8.14 Builder line for **FLUX TOTAL**

This command should be placed *below* the source definition commands (**GRID** and **SOURCE**) because it is applied to all existing rays when the command is encountered. Remember, when we click the **Run** button, ASAP executes the commands in the Builder in their order, from the top down.

In the example above, we have entered the value **100** in the **Amount** cell. ASAP retains the relative flux of the default rays, but rescales them so that the sum of all ray fluxes is exactly 100. But what units are implied? Watts? Lumens? ASAP doesn't really care. Recall that you had the option of adding a **Flux Label** to your **UNITS** command. If you declared the flux units there to be Watts, all subsequent analysis results are labeled with this character string when appropriate. It is only a label, however.

The **FLUX TOTAL** command, while described as a source modifier in the Builder menu, behaves a little differently than the object modifiers we used while defining our system geometry. Object modifiers, like the **INTERFACE** and **BOUNDS** commands, apply only to the most recently defined object. They cannot go anywhere in the Builder file, but must be placed in a list directly below the object definition. The **FLUX** command can be used at any time after the source is created. By default, it changes the flux of *all* rays that exist when the command is executed, not just the most recently defined source. What should we do if we have more than one source, and we want to scale the flux of each source independently? The answer lies in the two optional cells in the flux command. If you double-click the **Cmd** cell and select **Source**, you can specify a **Source #** in the last cell (Figure 8.15).

*	Type	Cmd	Cmd	Factor	Cmd	Source
CMD	Flux	Total	TOTAL	100	SOU	2

Figure 8.15 Builder line using **FLUX**

This specification allows us to scale the sources individually. The sources are addressed in the order in which they were defined. The first source defined (highest in the Builder) is 1, the second is 2, and so on. Of course, if you have only one source in your system, these last two cells may be left blank (see Figure 8.16).

> ***Note:*** A second, more fundamental (but usually less convenient) way to set the flux of the rays in your system is with the **FLUX** command, and is available in the Builder menu using **Ray Controls> Ray Modifiers> Flux**.

*	Type	Cmd	Addition	Factor	Start Ray Number	End Ray Number
CMD	Flux	Selected	1	0		

Figure 8.16 Builder line using **FLUX**

This command requires two arguments: an additive correction and a scale factor. Each ray in your system has its flux altered in this way:

New flux value = Addition + (Scale Factor x Old flux value).

This equation is often used to force every ray in the system to have a flux of exactly **1**, as shown in Figure 8.16. The starting ray number and ending ray number are optional. In the next chapter, we will learn ways to determine the ray numbers associated with individual rays and sources.

Summary

In this chapter, we have discussed the following new commands:

ASAP Commands	Builder Menu	Description
GRID RECT	**Grids> Grid, select Rect**	Define the positions of a rectangular grid of rays.
GRID ELLIPTIC	**Grids> Grid, select Elliptic**	Same as GRID RECT, but rays outside an elliptical boundary are discarded.
GRID POLAR	**Grids> Grid> select Polar**	Define the positions of a circular grid of rays on concentric rings.
SOURCE DIRECTION	**Grids> Source select Direction**	Define the directions of a grid of rays so that all rays are parallel.
SOURCE POSITION	**Grids> Source, select Position**	Define the directions of a grid of rays so that all rays appear to be diverging from a point.
SOURCE FOCUS	**Grids> Source, select Focus**	Define the directions of a grid of rays so that all rays appear to be converging toward a point
FLUX TOTAL	**Ray Control> Ray Modifiers> Flux Total**	Set the total flux of one or all sources.

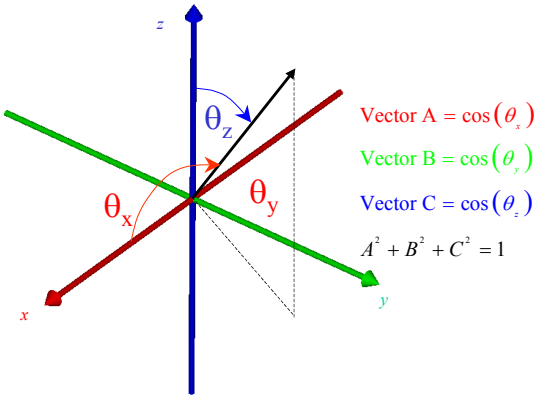
In this chapter, we introduced the basic tools necessary for defining grids of rays. You saw that grid sources required three steps to create them. The various versions of the GRID command allow us to specify where the rays will be located

initially. The **SOURCE** commands set the initial directions of the rays, and **FLUX TOTAL** rescales the flux of all rays so that they sum to a specific total.

The grid sources are normally used to model point sources. A second family of ASAP sources, known as *emitters,* is better suited for defining extended sources like incandescent filaments, high intensity discharge lamps or light-emitting diodes. ASAP uses a random number generator to specify both the positions and directions of the rays in emitters, within constraints specified by the user. These sources will be discussed in Chapter 16.

Direction Cosines

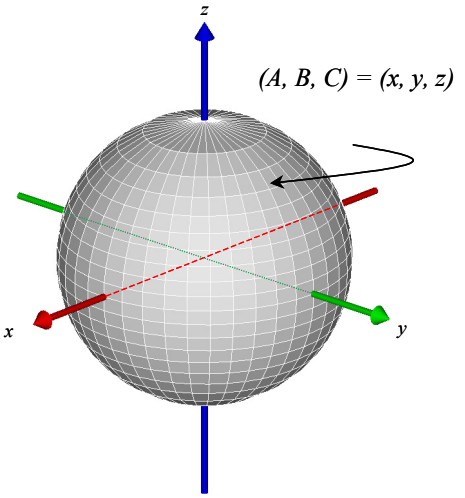
Direction cosines are a common means of specifying directions in mathematics and physics. Any direction in three-dimensional space is uniquely specified by the cosine of the angle that the unit direction vector makes with each of the three Cartesian coordinate axes. Since we are dealing with a unit vector, the sum of the squares of the three direction cosines always is one. This means that just two of the direction cosines, and the sign of the third will uniquely specify any direction in space.



This method of specifying directions is the default method used throughout ASAP, and is the means by which ASAP stores directions internally, in most cases. Direction cosines have the virtue that all three Cartesian axes are treated the same. This is in contrast with the spherical-polar coordinate system where we must designate one axis as a special "polar" axis. Many seemingly complex physical phenomena become mathematically simple and elegant when directions are expressed in terms of direction cosines. The grating equation is just one example of this. Physics, it would seem, loves direction cosines.

If you are new to direction cosines, the figure below illustrates a useful method for visualizing their meaning.

The figure shows a unit sphere. The direction vector can be thought of as an arrow drawn from the origin to a point on the unit sphere. Any direction in space will have a corresponding point on the sphere. The direction cosines are just the Cartesian coordinates of that point on the unit sphere.



APPENDIX

8A

SCRIPTS FOR CHAPTER 8

The following ASAP scripts are referenced in Chapter 8, Grid Sources.

Script 8-1

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY THE BUILDER.
GRID RECT Z -10 -1 1 -1 1 10 10
  SOURCE DIR 0 0 1
FLUX TOTAL 59 SOU 1
!!THESE COMMANDS CREATE A GRID OF RAYS, GIVE EACH RAY AN
!!INITIAL DIRECTION, AND ASSIGN A FLUX VALUE TO EACH RAY.
!!THESE 3 COMMANDS WILL BE EXPLAINED IN THE FOLLOWING PAGES.
```

Script 8-2

```
!!COMMAND SCRIPT PRODUCED BY THE BUILDER FOR AN ELLIPTICAL GRID
GRID ELLIPTIC Z -10 -1 1 -1 1 10 10
```

Script 8-3

```
!!COMMAND SCRIPT PRODUCED BY THE BUILDER FOR A POLAR GRID
GRID POLAR Z -10 .01 1 0 360 3 5 1
```

Script 8-4

```
!!COMMAND SCRIPT PRODUCED BY THE BUILDER FOR AN OFF-AXIS PARALLEL SOURCE
GRID RECT Z -10 -1 1 -1 1 11 11
  SOURCE DIR 0 SIN(30) COS(30)
```

Script 8-5

```
!!COMMAND SCRIPT PRODUCED BY THE BUILDER FOR AN OFF-AXIS PARALLEL SOURCE
!!THE DIRECTION OF THE RAYS IS SPECIFIED WITH SPHERICAL POLAR ANGLES,
!!RATHER THAN WITH DIRECTION COSINES AS DONE ABOVE.
GRID RECT Z -10 -1 1 -1 1 11 11
  SOURCE DIR Z 30 90
```

CHAPTER

9

VERIFYING SOURCES

In this chapter, we will explore some of the tools available within ASAP for verifying that you have defined your source correctly, before tracing the rays. Remember that after creating a new source, the rays can be thought of as points in space, each with a direction vector and a flux associated with it.

If your model includes multiple sources feeding the system from a variety of positions and directions, you can check your definitions graphically.

When do you need to verify sources? If you are confident that you have placed the source in the right place, and set the initial ray directions correctly, perhaps this step is not necessary. But if your model includes multiple sources feeding the system from a variety of positions and directions, it becomes increasingly important to check the definitions graphically.

To help you verify your sources, ASAP can plot these ray positions and directions either in two dimensions (using the **Plot Viewer** or **Chart Viewer**) or in three dimensions (using the **3D Viewer**). You will also learn how to combine the geometry and source verification tools to see the initial ray positions in the same view with the geometry. These graphical tools were the basis for most of the figures in the preceding chapter.

> ***Note:*** We access all these source verification tools from the **Rays** menu in the ASAP menu bar. Many of these functions are duplicated in the **Analysis** menu selections in one form or another. What does verifying sources have to do with analyzing the performance of an optical system? In both cases, we are just looking at the rays, and they are points in space. We can plot, list, and sort them, or perform statistical calculations on the ray set as a whole. As far as ASAP is concerned, these activities can be carried out at any time, no matter where the rays are currently located. Any time we look at rays, we are performing analysis.

Plot Positions 2D and View Positions 3D

Sometimes, we just want to see the positions of the rays in a grid (or any other source) plotted in two dimensions. This view can be particularly useful for polar grids, since the eye is often the best tool for deciding if the grid we have created is giving us the uniform sampling needed. This tool is available as soon as you have defined some rays. Try opening a new Builder file, and creating the polar grid shown here.

*	Type	Option	Option	Axis	Location	Radial Min	Radial Max	Angular Min	Angular Max	Radial Rays	Ring 0 Rays	Weight
SYS	Units	Millimeters										
ENT	Grid	Polar	Long Form	Z	-10	0.01	1	0	360	7	7	1
MOD	Source	Direction		0	0	1						

Figure 9.1 Creating the polar grid in the Builder

Remember to click the **End** button to reinitialize the ASAP system database and remove any old rays. Now, run the Builder file to create these rays. If your definition is the same as the one above, the **Command Output** window shows that 307 rays were created:

```
--- UNITS MILLIMETERS
--- GRID POLAR Z -10 0.01 1 0 360 7 7 1
     307 rays created in a POLAR grid for a total of 307
```

Figure 9.2 Command output for `GRID POLAR`

See Chapter 9 Appendix, "Script 9-1" on page 175.

The tool for plotting ray positions is available from the main menu **Rays> Graphics> Plot Positions 2D**. When you select it, the dialog box shown in Figure 9.3 is displayed.

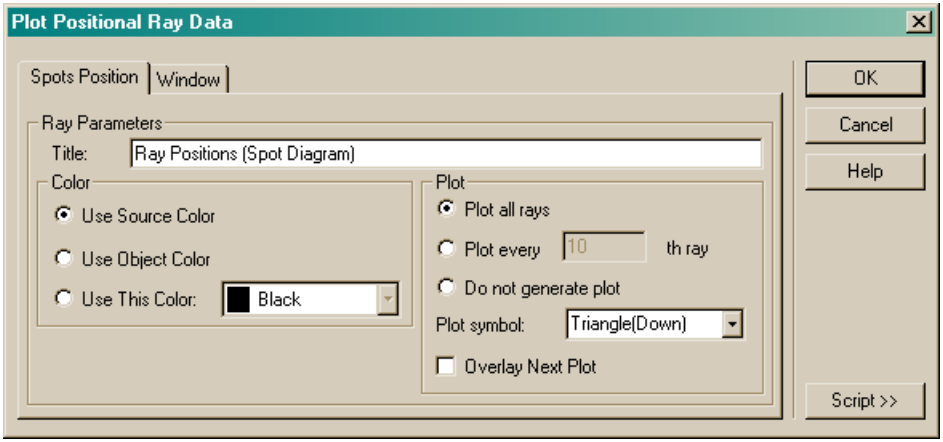


Figure 9.3 Dialog box for **Plot Positional Ray Data**

You can choose from among several plot symbols available to mark the location of each ray. As for geometry, we must also specify a projection window. Since our source is defined in the plane perpendicular to the *z* axis, the appropriate choice would be **Y** (vertical) and **X** (horizontal). To set this, click the **Window** tab on this dialog box, and change the horizontal axis to **Z**. Then, return to the **Spots Position** tab, and select the **Triangle (Down)** plot symbol. Click **OK**, and the plot of the ray positions appears in Figure 9.4 on page 162.

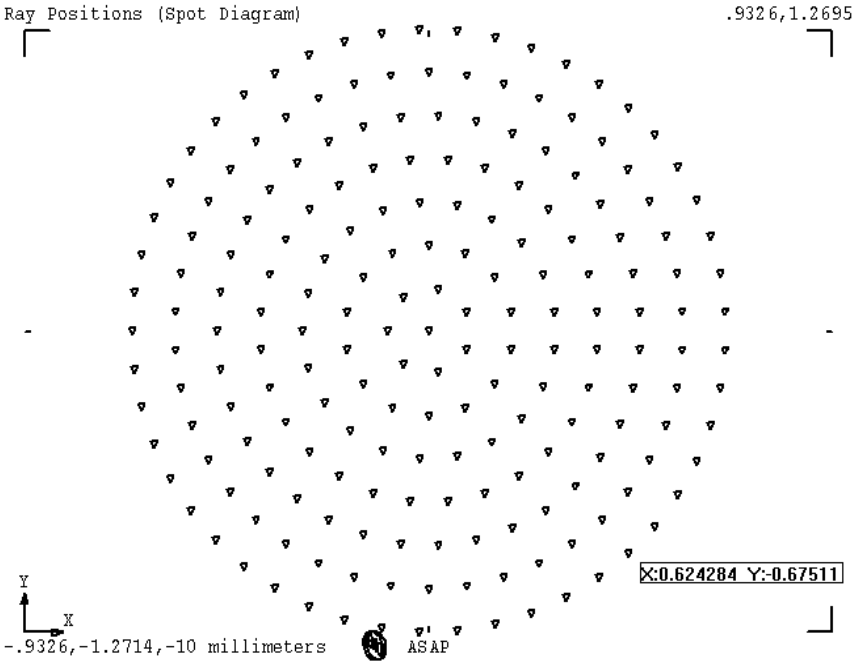


Figure 9.4 **Plot Positions 2D** can be used to verify the initial location of the rays, or to plot current positions at any time rays exist. They are plotted in the ASAP **Plot Viewer**, where you can zoom shift, or read out cursor coordinates (see Chapter 5). In this figure, we are reading the approximate **X** and **Y** coordinates of one of the rays.

See Chapter 9 Appendix, "Script 9-2" on page 175.

You may want to see the same information in the **3D Viewer**. You can accomplish this from the main menu with **Rays> Graphics> View Positions 3D**. This command is particularly useful when combined with a view of the geometry. We will discuss this shortly.

> ***Note:*** For both **Plot Positions 2D** and **View Positions 3D**, the basic ASAP command being used is `SPOTS POSITION`. This important ASAP analysis command will be discussed at some length in Chapter 11, "Basic Analysis", and Chapter 19, "Analyzing Total Flux and Positional Flux Distributions".

Plot Rays 2D and View Rays 3D

While **Plot Positions 2D** shows us the location of the rays, it tells us nothing about their directions. We have verified the `GRID` command, but not the `SOURCE` command. To do this, ASAP provides us with **Rays> Graphics> Plot Rays 2D**,

and **Rays> Graphics> View Rays 3D**. The dialog boxes, **Plot Rays 2D** (Figure 9.5a) and **View Rays 3D** (Figure 9.5b)are used to plot the ray direction vectors.

The critical item in both of these dialog boxes is the area labeled **Scale Factor for Ray Plot Vector**. The number you enter here will be the length (in system units) of the vector representing the ray carrying the most flux. Other ray lengths in the plot will be scaled down in proportion to their flux, relative to the flux of the ray with the maximum flux. This value should be selected so that the rays have a useful length relative to the size of the source.

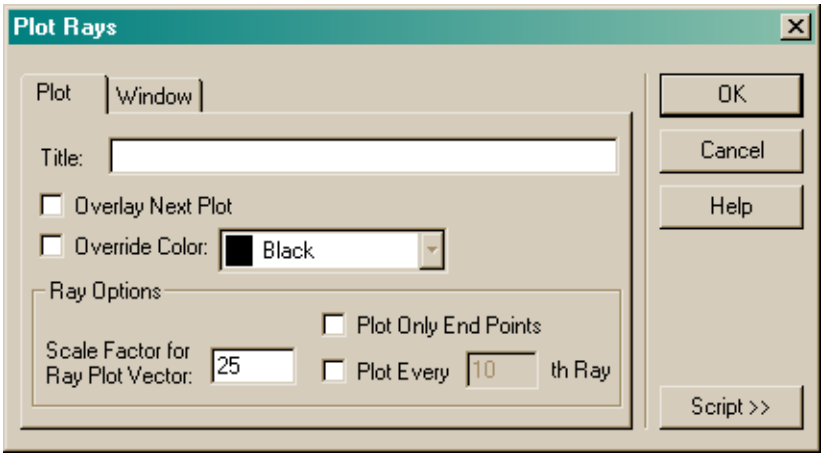


Figure 9.5a **Plot Rays** dialog box for plotting ray direction vectors

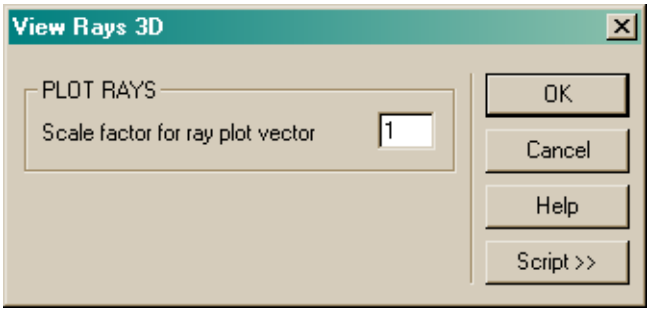


Figure 9.5b **View Rays 3D** dialog box for viewing ray direction vectors

The **Scale factor** in both of these dialog boxes allows us to control the length of the vector. You should normally set it to a value similar to the dimensions of the grid.

See Chapter 9 Appendix, "Script 9-3" on page 176.

The result for a ray grid created using `SOURCE POSITION` is shown in Figure 9.6.

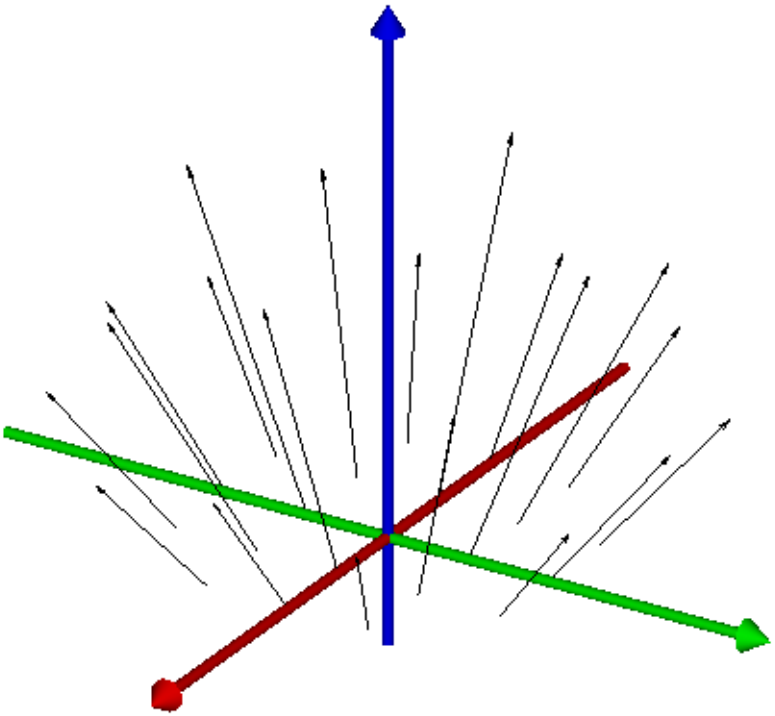


Figure 9.6 View Rays 3D plots a direction vector, the origin of which is at the ray position. ASAP allows you to select the length (in system units) of the ray with the greatest flux. All other vector lengths are then scaled according to their flux relative to that ray. These rays were created with `GRID ELLIPTIC`, and given directions with `SOURCE POSITION`. The rays have varying lengths because ASAP is trying to simulate a nearby point source. The rays, however, are located on the surface of a plane, not on the surface of a sphere, where all fluxes would be the same.

See Chapter 9 Appendix,"Script 9-4" on page 176.

Combining Ray and Geometry Graphics

Plotting the ray positions and direction vectors is often more beneficial if we can see geometry in the same view. The following procedure will allow you to do this.

1. Run the desired Builder file (or files) to create the system geometry and rays.
2. Click the **Vector Rewind** button on the ASAP toolbar to make sure there is no old three-dimensional information in the vector file (see Chapter 5, "Running and Verifying Geometry").

3 From the ASAP menu bar, select **System> Plot Facets** to see the **Plot Facets** dialog box.

4 Set the desired options at the **Plot Facets** tab, as you did in Chapter 5.

5 Select the box labeled **Overlay Next Plot**. Selecting this option adds the ray plot to the same **Plot Viewer** window.

6 Add the necessary information at the **Window** tab.

In this case, **Y Z** is a good window choice. You can allow ASAP to autoscale the plot for you, but it may be necessary to set the limits yourself. Remember that this window must be large enough to show the location of both the geometry and the rays.

7 Click **OK** to create the plot.

8 From the **Rays** menu, select **Rays> Graphics> Plot Rays 2D**.

9 The only piece of information that must be provided in the **Plot** tab is the **Scale Factor for Ray Plot Vector**, as described in the previous section. The **Title** and **Window** information should not be changed, since we are adding this plot to the previous one.

10 On the **Window** tab, deselect **Autoscale** to force ASAP to use the same scale, as with **Plot Facets** above.

11 Click **OK** to create the plot. You should now see the two-dimensional **Plot Viewer** version of the graphic (see Figure 9.7).

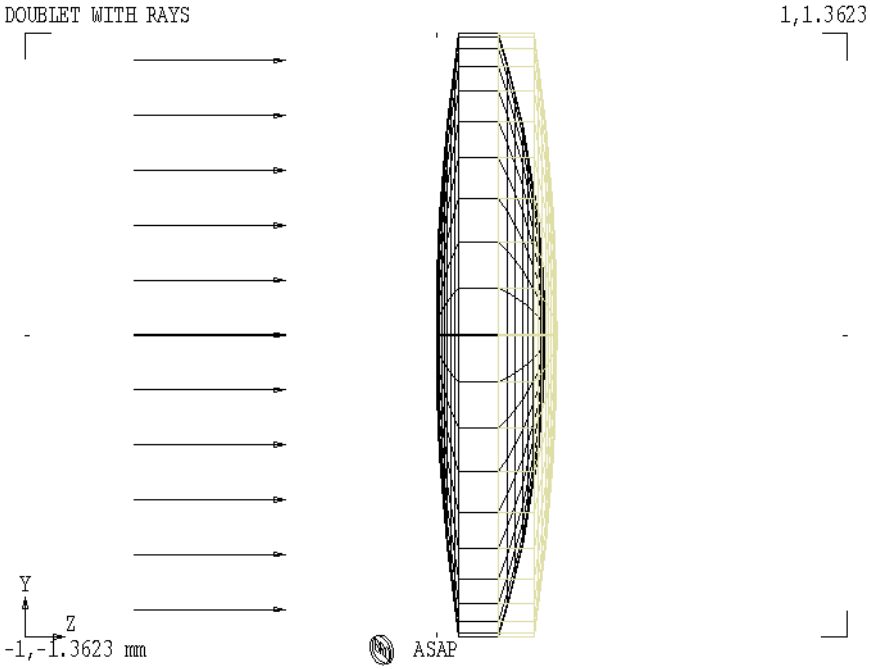


Figure 9.7 The **Plot Viewer** window shows a faceted version of a cemented doublet lens, combined with a plot of the initial ray vectors. This plot was created with the **Overlay** option in the **Plot Facets** dialog box, followed by **Plot Rays 2D**, using a window with the same dimensions.

12 Click the **3D Viewer** button to see the three-dimensional version, which ASAP created from your work in the previous steps (Figure 9.8).

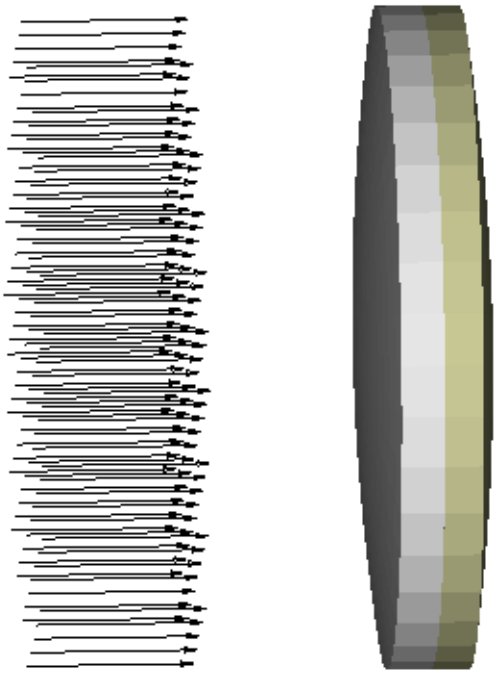


Figure 9.8 The plots shown in Figure 9.7 automatically generated a three-dimensional version of the same graphic, which can be viewed by clicking the 3D Viewer button on the toolbar.

See Chapter 9 Appendix, "Script 9-5" on page 176.

While this procedure may seem long and involved, it is worth understanding and practicing. It is completely general in the sense that it allows you to combine many graphical products into a single view. Virtually any graphical representation of your geometry, rays, ray-trace plots, and eventually even your analysis results can be added together in this way. In a few cases (like plotting a ray trace on top of the geometry), ASAP provides a single menu command and associated dialog box to perform these steps for you (see Chapter 10, "Tracing Rays"). It is impossible, however, to anticipate all the graphical needs of an imaginative optical engineer. You will grow to appreciate the flexibility that this basic procedure gives you.

Note: If you choose to combine **System> Plot Facets** and **Rays> Graphics> Plot Positions 2D** to see only ray positions without vectors, be sure to select the **Dots** plot symbol option in the **Plot Positions 2D** dialog box. This symbol is the only one that will appear in the **3D Viewer**.

Numerical Ray Information

ASAP also provides us with tools to access the numerical ray parameters, showing us the fundamental data straight out of the ray table (the binary file named **virtual.pgs**). While this is rarely necessary, it does help us to remember the true nature of an ASAP ray: a point in space with an associated direction, flux, and other parameters relevant to tracing the ray through a system model. As we will see later (Chapter 11, “Basic Analysis”), this numerical information is the source used by ASAP to generate virtually all analysis results that we request of it.

Try creating the following simple source.

*	Type	Option	Axis	Position	Major Axis	Major Axis	Minor Axis Min	Minor Axis Max	Major Axis Rays	Minor Axis Rays
ENT	Grid	Rect	Z	10	1	1	1	1	3	3
MOD	Source	Direction		0	0	1				

Figure 9.9 **Builder** lines for creating a source

This line is basically the default `GRID RECT` and `SOURCE DIR` commands provided by the Builder, although we have reduced the number of rays in the grid to only 3×3.

1. Click the **End** button, and run this Builder file to create the rays.
2. Select **Rays> Ray Information> List Ray Data** from the main menu bar.
3. Choose **Positions and Directions** from the options provided, and click **OK**.

The following information appears in the **Command Output** window.

```
!     X            Y            Z            A            B            C            F            S            D        ;$FAST 9 _
 -.666667     -.666667     -10.0000     0.00000      0.00000      1.00000      0.444444     0.333333     0.00000      !        1
 -.666667     0.298023E-07 -10.0000     0.00000      0.00000      1.00000      0.444444     0.333333     0.00000      !        2
 -.666667     0.666667     -10.0000     0.00000      0.00000      1.00000      0.444444     0.333333     0.00000      !        3
 0.298023E-07 0.666667     -10.0000     0.00000      0.00000      1.00000      0.444444     0.333333     0.00000      !        4
 0.298023E-07 -.298023E-07 -10.0000     0.00000      0.00000      1.00000      0.444444     0.333333     0.00000      !        5
 0.298023E-07 -.666667     -10.0000     0.00000      0.00000      1.00000      0.444444     0.333333     0.00000      !        6
 0.666667     -.666667     -10.0000     0.00000      0.00000      1.00000      0.444444     0.333333     0.00000      !        7
 0.666667     0.298023E-07 -10.0000     0.00000      0.00000      1.00000      0.444444     0.333333     0.00000      !        8
 0.666667     0.666667     -10.0000     0.00000      0.00000      1.00000      0.444444     0.333333     0.00000      !        9
```

Figure 9.10 Command window output for `GRID RECT`

See Chapter 9 Appendix, “Script 9-6” on page 177.

This output is some of the fundamental data maintained by ASAP about each ray. In our case, we have only nine rays. The first three columns are their *x*, *y*, and *z* positions. The next three columns are the corresponding direction cosines. The seventh column is the flux of the ray. The eighth and ninth columns (labeled S and D) are additional ray parameters, which are generally important only when ASAP is being used to do wave optics. The last column on the extreme right is the ray number. Several ASAP commands allow us to address individual rays by this number, as you will see next.

This information reflects only a small fraction of what ASAP maintains about each ray. Even more detail about any particular ray is available to us, if we need it.

1. Select **Rays> Ray Information> Show Ray Details** from the menu bar.
2. In the **Select Rays** area of the dialog box, next to **Data from Ray Number** enter **5**.
3. Select only **Basic Ray Data** at the bottom, and click **OK**.

ASAP again places the results in the **Command Output** window. We see additional information specific to ray number 5 (Figure 9.11).

```
A0: X_DIR_B                    0
B0: Y_DIR_B                    0
C0: Z_DIR_B                    1
D0: OPL                        0
F0: FLUX                       .444444477558136
J0: SOURCE                     1
K0: CURR_OBJ                   0
M0: MEDIUM                     0
U0: UPARAMB                    0
V0: VPARAMB                    0
W0: WAVELEN                    0
X0: X_POS_B                    2.980232238769531E-8
Y0: Y_POS_B                    -2.980232238769531E-8
Z0: Z_POS_B                    -10
```

Figure 9.11 Command output of **Basic Ray Data** for ray number 5

See Chapter 9 Appendix, "Script 9-7" on page 177.

Note: The basic ASAP command performing this operation is **GET**. This command always retrieves all the available information about the ray, and loads it into various ASAP register variables. The dialog box for **Show Ray Details** then assists us in printing the subsets of these data that we have selected. The location and names of these register variables are what we see in the two columns to the left of the numerical values. (You will have your first introduction to register variables in Chapter 15, "Repositioning Geometry and Rays". A more thorough discussion of variables and expressions is included in Chapter 24, "Other Command Script Features").

Again, we see the direction cosines at the top (A0, B0, and C0), and the ray coordinates at the bottom (X0, Y0, and Z0). We also see the optical path length of the ray (which will remain zero until we have traced the ray), the current flux, and the source number to which this ray belongs. Recall that rays always belong

to objects. The current object for these rays is the fictitious “Object 0”, the source. We have also emphasized that the ray must always know what medium it is currently in, to determine whether it is entering or leaving a lens or other transmissive object. The medium in which this ray is currently located is `MEDIUM 0`, the default, “vacuum-air” medium. The U0 and V0 parameters are another way of expressing the location of a ray when it is currently located on an edge entity. W0 is the wavelength of the ray.

All these values will change as the rays are traced through an optical system. It is important to realize that to ASAP, a ray is a table of parameters.

Summary

In this chapter, we have discussed the following new commands:

ASAP Commands	Menu	Description
`SPOTS POSITION`	**Rays> Graphics> Plot Positions 2D**	Plot the positions of rays in the **Plot Viewer**.
`SPOTS POSITION`	**Rays> Graphics> View Positions 3D**	View the positions of the rays in the **3D Viewer**.
`PLOT RAYS`	**Rays> Graphics> Plot Rays 2D**	Plot the ray locations with direction vectors attached in the **Plot Viewer**
`PLOT RAYS`	**Rays> Graphics> View Rays 3D**	View the ray locations with direction vectors attached in the **3D Viewer**
`LIST RAYS`	**Rays> Ray Information> List Ray Data**	List ray positions, directions, flux, and ray number in the **Command Output** window.
`GET`	**Rays> Ray Information> Show Ray Details**	List various details about a single ray (or averages) in the **Command Output** window.

Revisiting the four basic steps you must learn to complete any ASAP project, two have now been addressed:

- √ Building the system model
- √ Creating sources (rays)
- Tracing the rays
- Performing the analysis

We now know how to make several important surface-based entities, and to create and verify at least one kind of ASAP source. There is, of course, much more to be said about creating both objects and sources, but you now have enough experience to proceed with your first look at ray tracing and analysis.

Exercise 3: A source for the Cooke triplet

1. Recover your Builder file that defined a Cooke triplet lens system developed in Chapters 3 and 4 (see Figure 4.15a, “Builder file for element 1 of the Cooke triplet.” on page 80, and “Exercise 1: Completing the Cooke Triplet” on page 83).
2. Add a circular absorbing stop inside the lens system with the following specifications:

Z position	18.89 mm
Diameter	16.9 mm
Hole diameter	10.4 mm

   The result should appear as shown in profile in Figure 9.12 at the end of this exercise.
3. Define a collimated, on-axis, elliptical grid of rays (`GRID ELLIPTIC, SOURCE DIRECTION`) for the Cooke triplet. Locate the source at a position z = -10, with a diameter the same as that of Lens 1. Define the grid so that it has a total of 11 rays along both the **x** and **y** axes. The definition can be done at the bottom of the same Builder file that defines the geometry.
4. Set the flux of the source to be 10 Watts.
5. Plot the positions of the rays in the **x-y** plane using **Rays> Graphics> Plot Positions 2D**.
6. Use the cursor to estimate the coordinates of the ray lying along the $+x$ axis that has the largest **x** value (the last ray in that row).
7. Print the coordinates of all rays using **Rays> Ray Information> List Ray Data**. Locate the ray measured in step 6, and compare the results. Can you improve the precision of the cursor measurement by zooming in on the ray in the **Plot Viewer**?
8. Combine `PLOT RAYS` with `PLOT FACETS` to produce a two-dimensional and a three-dimensional view of the geometry, along with the initial positions and directions of the ray grid.

Hints

- The aperture stop can be made with the `PLANE` command, using the **Obs Ratio** to set the size of the hole.
- If your plot of positions does not look correct, check your choice of window, which should be the **Y X** window.
- Depending on the size of your monitor and choice of display setting resolution, you may need to scroll to the right in the **Command Output** window to see the ray numbers on the far right of the table, which was printed from **Rays> Ray Information> List Ray Data**.
- Remember to select the **Overlay** option in the **Plot Facets** dialog box to see both plots in the same **Plot Viewer**.

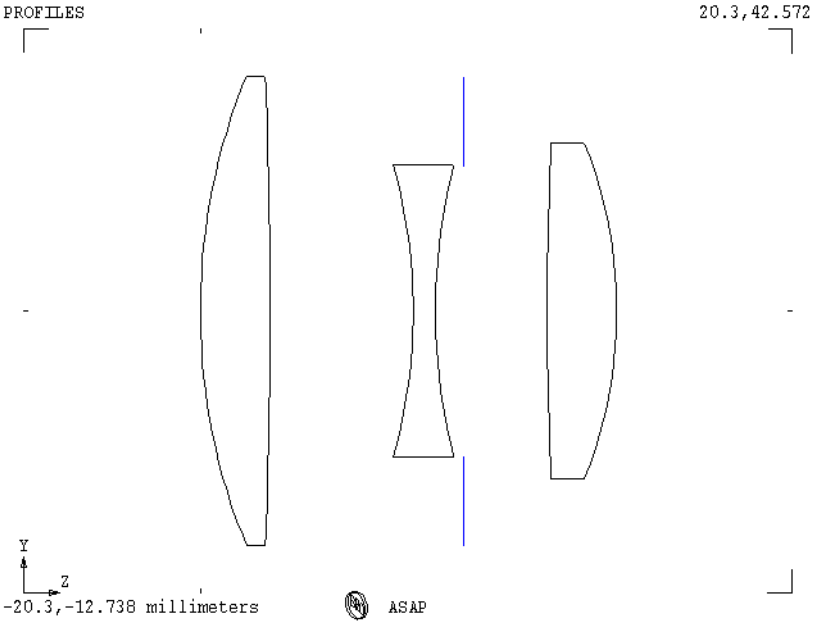


Figure 9.12 Profile plot results for Exercise 3

APPENDIX

9A

SCRIPTS FOR CHAPTER 9

The following ASAP scripts are referenced in Chapter 9, “Verifying Sources”.

Script 9-1

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY THE BUILDER FOR A POLAR GRID.
UNITS MILLIMETERS
GRID POLAR Z -10 0 1 0 360 7 7 1
  SOURCE DIR 0 0 1
```

Script 9-2

```
!!THIS IS THE COMMAND SCRIPT PRODUCED FROM THE PULL-DOWN MENU.
!!FOR A 2D PLOT OF THE POLAR GRID OF RAYS.
WINDOW Y Z
SPOTS POSITION ATTRIBUTE 4
```

Script 9-3

```
!!THIS IS THE COMMAND SCRIPT PRODUCED FROM THE PULL-DOWN MENU.
!!FOR A 3D PLOT OF THE POSITIONS OF THE POLAR GRID OF RAYS.
$IO VECTOR REWIND   !!REMOVING THE INFO ALREADY IN THE VECTOR FILE
$IO PLOT CANCEL
$PLOT OFF           !!TURN OFF PLOTTING OF THE SPOTS POSITION IN THE 2D WINDOW
SPOTS POSITION
$VIEW               !!VIEW THE RESULT IN THE 3D VIEWER WINDOW
$PLOT NORM          !!TURN PLOTTING BACK ON
$IO PLOT
```

Script 9-4

```
!!THIS IS THE COMMAND SCRIPT PRODUCED FROM THE PULL-DOWN MENU.
!!FOR A 3D PLOT OF THE DIRECTIONS OF THE POLAR GRID OF RAYS.
$IO VECTOR REWIND   !!REMOVING THE INFO ALREADY IN THE VECTOR FILE
$IO PLOT CANCEL
$PLOT OFF           !!TURN OFF PLOTTING OF THE SPOTS POSITION IN THE 2D WINDOW
PLOT RAYS  1
$VIEW               !!VIEW THE RESULT IN THE 3D VIEWER WINDOW
$PLOT NORM          !!TURN PLOTTING BACK ON
$IO PLOT
```

Script 9-5

```
!!THIS IS THE COMMAND SCRIPT PRODUCED FROM THE PULL-DOWN MENUS
!!FOR COMBINING THE RAY AND GEOMETRY GRAPHICS.
$IO VECTOR REWIND
WINDOW Y Z
OBLIQUE OFF
PLOT FACETS 5 5 OVERLAY 'Combining Ray and Geometry Graphics'

WINDOW Y -1.001 1.001 Z -0.021 0.421
PLOT RAYS .5
$VIEW
```

Script 9-6

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY THE BUILDER
!!AND FROM THE PULL-DOWN MENUS.
GRID RECT Z -10 -1 1 -1 1 3 3
  SOURCE DIR 0 0 1

LIST POSITION     !!CHOOSING POSITIONS FROM LIST RAY DATA
LIST DIRECTION    !!CHOOSING DIRECTIONS FROM LIST RAY DATA
LIST RAYS         !!CHOOSING POSITIONS AND DIRECTIONS FROM LIST RAY DATA
```

Script 9-7

```
!!THIS IS THE COMMAND SCRIPT PRODUCED FROM THE PULL-DOWN MENUS.
$STO ASAPTEMP   !!STORE THE INFORMATION IN ASAPTEMP
GET 5           !!GET THE INFORMATION FOR RAY NUMBER 5
$REG A0 B0 C0 D0 F0 J0 K0 M0 U0 V0 W0 X0 Y0 Z0      !!THE REGISTER VALUES
$RCL ASAPTEMP   !!RECALL THE REGISTER INFORMATION FROM ASAPTEMP
```

CHAPTER

10

TRACING RAYS

In this chapter, we will trace rays for the first time. Ray tracing is the easiest of all the steps because ASAP does this for us. The ASAP ray-tracing kernel is designed to trace rays quickly and efficiently, and needs little or no help from us. Using both the Cooke Triplet and the Cassegrain telescope models developed in the previous chapters, we will concentrate on some of the graphics that can be produced during the ray-trace process, and also on some ray-trace concepts that sometimes confuse new ASAP users.

Basic Concepts (Review)

Before performing the first trace, we will review some topics introduced in Chapter 7, “Basic Ray Tracing Concepts”. In particular:

- ASAP rays are best thought of as points in space with associated directions and flux. It will always make sense to ask, “where is the ray (the point) now?”
- Rays belong to objects. When rays are created, they are assigned to a fictitious “Object 0”. As they trace, rays are transferred from object to object. When the trace is finished, rays belong to the last object they touched.
- Rays intersect objects in the physically correct order, regardless of the order in which the objects were defined.
- ASAP rays can be traced only once from their initial source positions. To see the trace again, you must discard the old rays, define new ones, and trace.

In the following section, we will trace rays through the Cooke triplet. We will use the same rays that were defined in “Exercise 3: A source for the Cooke triplet” on page 172 of Chapter 9. If you have not already generated it for yourself, you can begin with that version.

Try to develop the habit of always knowing (or determining) the current state of ASAP. After you run the Builder file, look at the **Objects** tab in the ASAP Workspace window, which indicates that a total of 11 objects now exist in the system database. Each object is assigned the optical properties appropriate to its role in the optical system. The “points in space” representing the initial ray set are at z = -10. They are all pointed parallel to the z axis, and their x and y coordinates distribute the points in space in a rectangular grid pattern. All the rays have the same flux, and these values sum to roughly unit flux per unit area. The rays belong to “Object 0”, the fictitious object used to describe the source.

This description is of the state of ASAP *after* running the Builder file and *before* tracing the rays.

Trace Dialog Box

The **Trace** dialog box is found on the main menu bar: **Trace> Trace Rays**. It appears as shown in Figure 10.1.

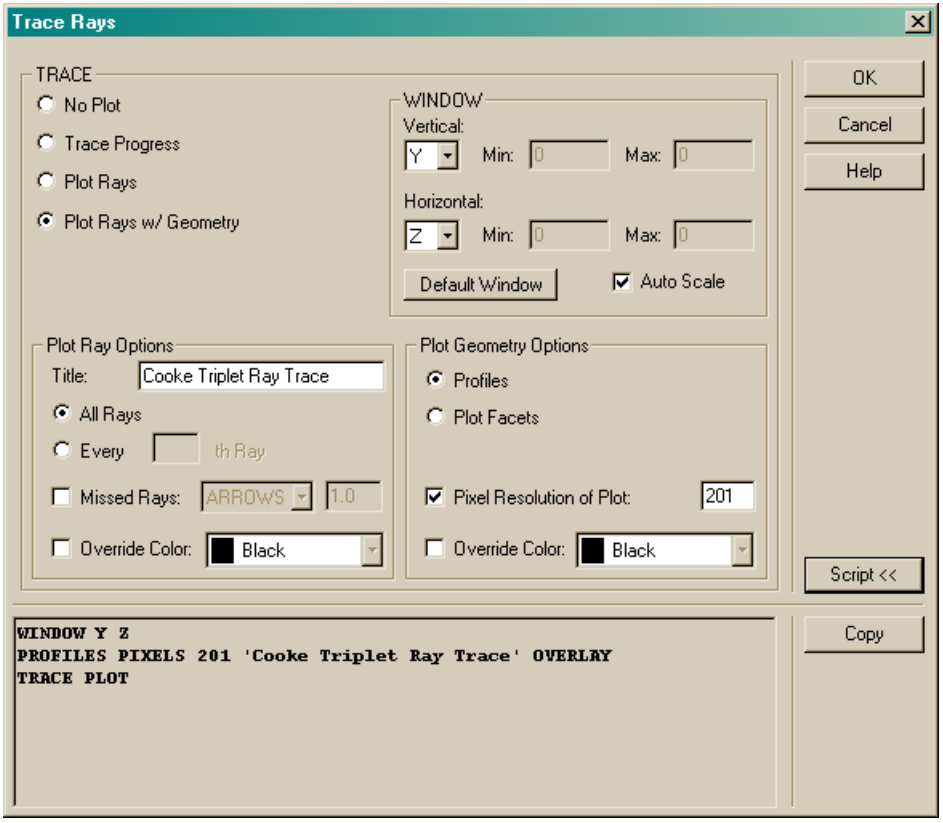


Figure 10.1 The **Trace Rays** dialog box not only is for tracing rays, but gives us the option to draw the ray paths on top of either a profile of the system or a faceted view of the geometry. By clicking the **Script** button, you can see the commands that will be sent to the kernel when you click **OK**. This area changes dynamically as you make new selections in the dialog box. This view lets you see and understand all that a complex dialog box like this is doing for you.

The **Trace Rays** dialog box not only initiates the ray trace, but also gives us several additional options. At the top left, select **Plot Rays w/Geometry** to see the ray paths as the rays are traced through the system. When this option is selected, a variety of other options become available.

You can now choose a window. In our case, Y-Z is appropriate, and **Auto Scale** is a good choice. In the **Ray Plot Options** area, choose **All Rays**. In the **Plot Geometry Options** area, select **Profiles**.

When you click **OK**, you should see a Plot Viewer window like the one shown in Figure 10.2.

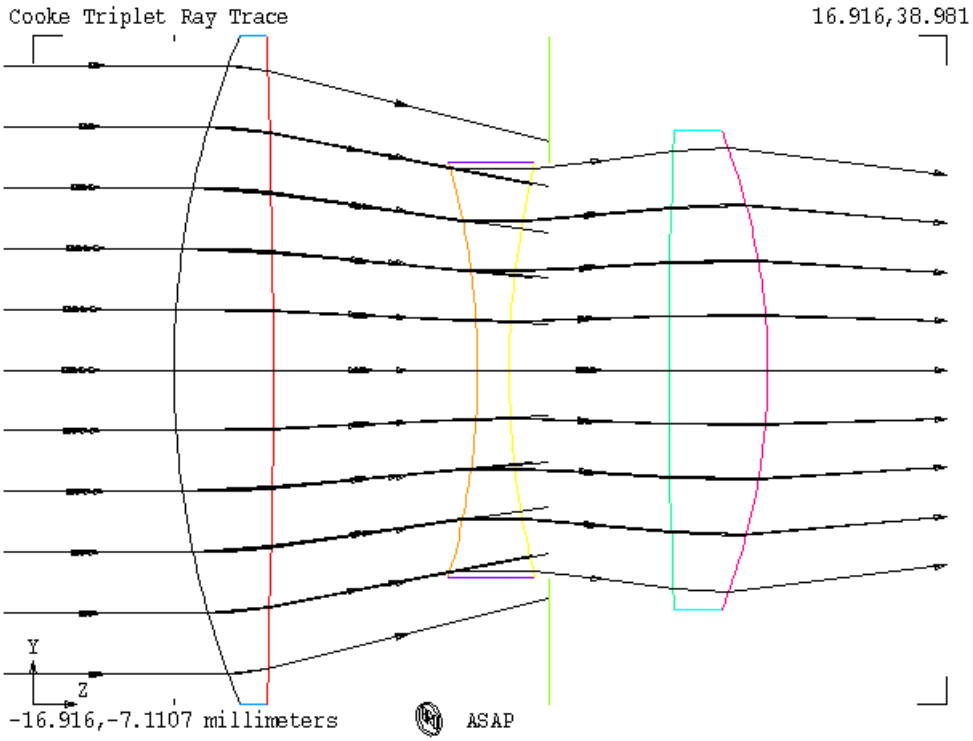


Figure 10.2 When the ray paths are plotted over a profile of the optical system, the results can be misleading or confusing. Some rays appear to enter the second element of the Cooke Triplet without refracting, and then they stop for no apparent reason. These rays have passed either in front of or behind the lens, and are stopping on the aperture stop. For a truly two-dimensional graph (for purposes of illustration only), try setting the number of **Major Axis Rays** to **1**, and tracing again over a profile.

This view is a little confusing, however. Most of the rays refract as expected, but some, it seems, do not. These also appear to stop in mid path after passing through the second lens.

See Chapter 10 Appendix, "Script 10-1" on page 197.

Often, a look in three dimensions clarifies what is happening. We will try performing the trace again, this time selecting **Plot Facets** instead of **Profiles**. Recall that **Plot Facets** creates a graphic that is well suited to display in the 3D Viewer. The extra information displayed there may show us why some of the rays appear to be behaving strangely.

ASAP rays can only be traced once from their initial source positions. If you want to see the trace again, you must discard the old rays, define new ones, and trace again.

Now, however, we run up against one of the basic ASAP ray tracing concepts: ASAP rays can be traced only once from their initial source positions. If you want to see the trace again, you must discard the old rays, define new ones, and trace again. The easiest way to do this with our simple system is to start over. Click the **End** button, and run the Builder file again.

> ***Note:*** There is really no need to run the geometry again. We can trace new rays through the same geometry over and over. When your geometry becomes complex enough that it takes a significant amount of time to run the definitions, it might be worth using a separate Builder file just for defining the source. For now, though, the time spent rerunning everything is insignificant.

Once you have run the Builder again, try tracing the rays with the **Plot Facets** option selected. Even in the Plot Viewer window (Figure 10.3a on page 184), it is more obvious what is happening: the rays in question are stopping on the aperture. Click the **3D View** button to see the rest of the explanation. If you rotate the view a small amount, it will be apparent that those rays missed the second lens in the system because the first element was overfilled by our source (Figure 10.3b). The unusual rays in our profile view were just foreground rays that, from our point of view, had gone in front of Lens 2.

The ***3D Viewer*** *is an essential tool for understanding ray paths through a three-dimensional system model.*

One of the functions of the aperture stop is to prevent these rays from exiting the system. An important lesson to learn here is that the 3D Viewer is an essential tool for understanding ray paths through a three-dimensional system model.

See Chapter 10 Appendix, "Script 10-2" on page 197.

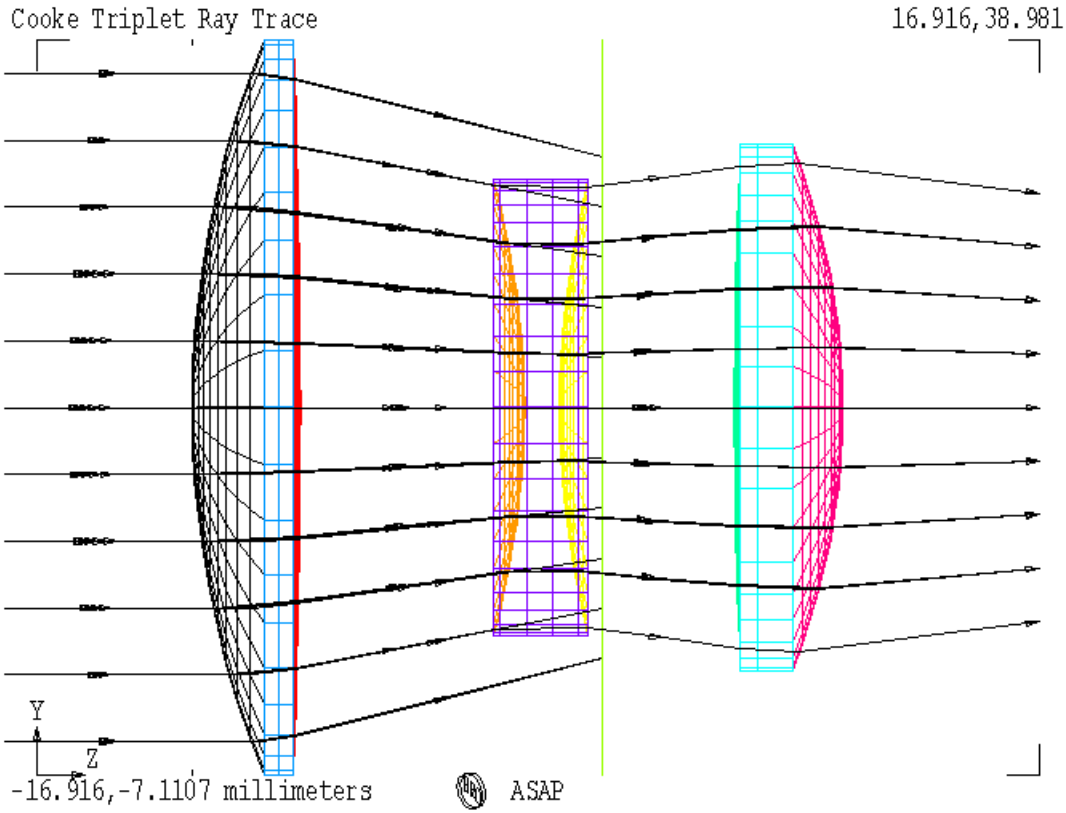


Figure 10.3a The graphic, produced by **PLOT FACETS**, shows some rays stopping on the aperture stop.

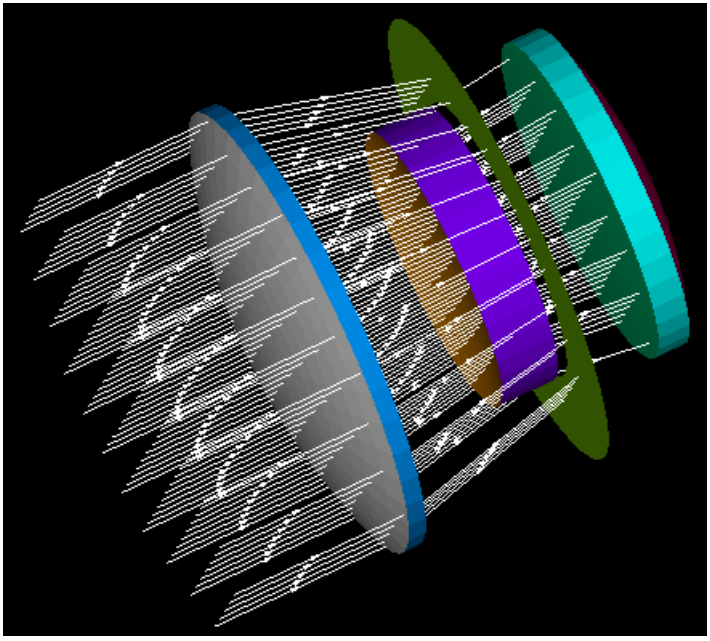


Figure 10.3b From the 3D view, it is clear that the rays that seem to pass through the second element are actually passing in front or behind it.

Rays that Miss

Rays belong to objects. When they are created, they are assigned to a fictitious "Object 0". As they trace, they are transferred from object to object. When the trace is finished, they belong to the last object they touched.

Figure 10.3a on page 184 shows rays leaving the back surface of Lens 3, with arrows indicating their direction. They appear to leave the system. Why don't these appear in Figure 10.3b on page 185, which is the same system shown in the 3D Viewer? This brings us back to another ASAP ray tracing fundamental: Rays belong to objects. When they are created, they are assigned to a fictitious "Object 0". As they trace, they are transferred from object to object. When the trace is finished, they belong to the last object they touched. So, where are the rays after the ray trace? They are on the back surface of Lens 3. They have refracted for the last time, they have a new direction vector, but they have nowhere else to go. The arrows in Figure 10.3a are drawn for our convenience to show the direction in which the rays would have proceeded if there were additional objects to intercept (like an image plane or detector). ASAP automatically draws the arrow to the edge of the Plot Viewer window, but the coordinates of the rays after the trace will correspond to the back surface of the lens.

So, where are the direction arrows in the 3D Viewer? The answer is that, by default, ASAP does not draw the arrows in the three-dimensional version of the graphic. There is a logical reason for this. While the Plot Viewer has a known size (defined by the `WINDOW` command), the 3D Viewer does not; it is boundless.

ASAP simply does not know how long to draw these “missed arrows”. You can still have the arrows drawn in this view, but you must tell ASAP how long you want them to be, measured in system units. Note that in the **Trace Rays** dialog box, shown in Figure 10.1 on page 181, there is a box labeled **Missed Rays** in the **Plot Ray Options** area. Try running the Builder and performing the trace one more time, checking this box, and choosing **Arrows of length 10**. Be sure to select **Plot Facets** (rather than the default **Profiles**) in the **Plot Geometry** options. This time, when you click the **3D View** button, the “missed arrows” is presented.

See Chapter 10 Appendix, “Script 10-3” on page 197.

> ***Note:*** You need to give the arrows a length of at least 90 units to see them go through a focus. Even then, they do not appear in the **Plot Viewer** window unless you override the auto scaling of the window. The **Plot Viewer** dimensions were fixed when you plotted the geometry. The missed arrows will, however, appear in the **3D Viewer**. Fortunately, there is a better way to find the best focus for this or any other imaging system. We will introduce that, among other analysis tools, in Chapter 11, “Basic Analysis”.

Front and Back of a Surface

ASAP does not distinguish between the front and the back of a surface.

Returning to the Cassegrain Telescope model you built in Chapter 6 and “Exercise 2: Cassegrain Telescope” on page 122, we will illustrate a few new points about ray tracing with ASAP. Figure 10.4 shows ray trace in profile. It also shows a problem: rays have reflected off both the front *and* the back of the secondary mirror. ASAP does not distinguish between the front and the back of a surface. If it did, we would be violating the principles of non-sequential ray tracing. The rays simply move through the system, encountering objects in whatever order they come to them.

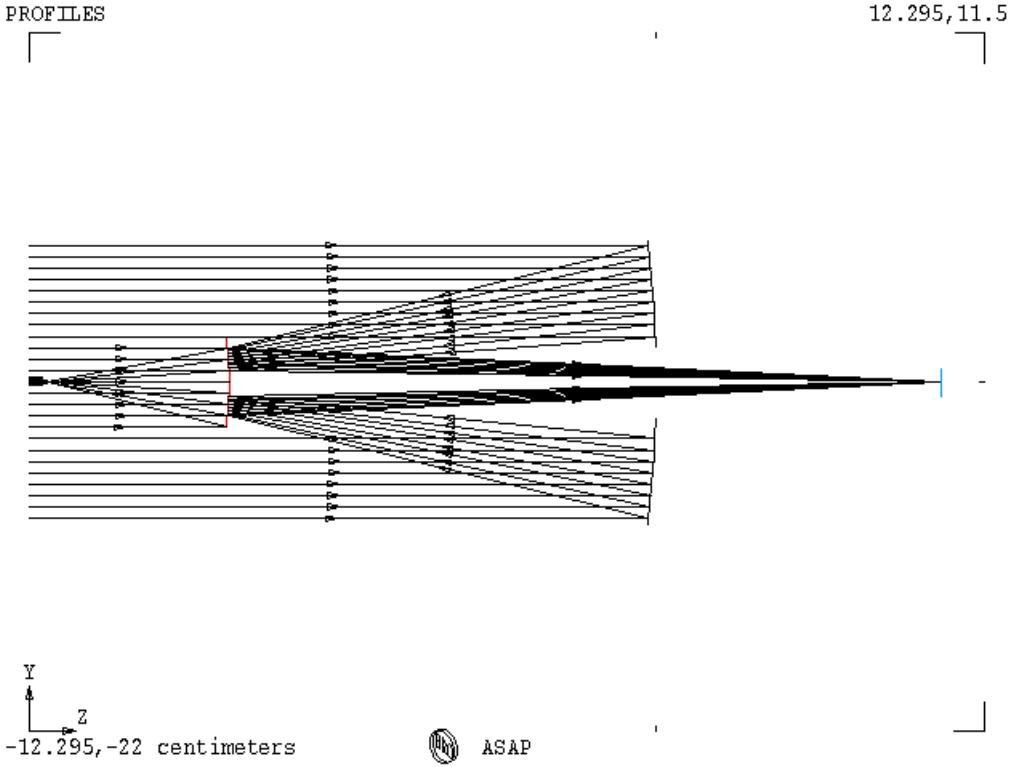


Figure 10.4 Real mirrors have a finite thickness with a front and a back. Our simple model of the Cassegrain telescope neglected this. While this omission is not a problem with the primary mirror, we find rays in the center of the grid reflecting off the wrong side of the secondary mirror.

Sometimes, problems like this can be fixed by relocating the source. What if we place the source just in front of the secondary? Figure 10.5 shows that this produces its own set of problems. Some rays from the center of the grid pass directly through the hole in the primary. This would not have been possible in the real system because the secondary mirror was in the way.

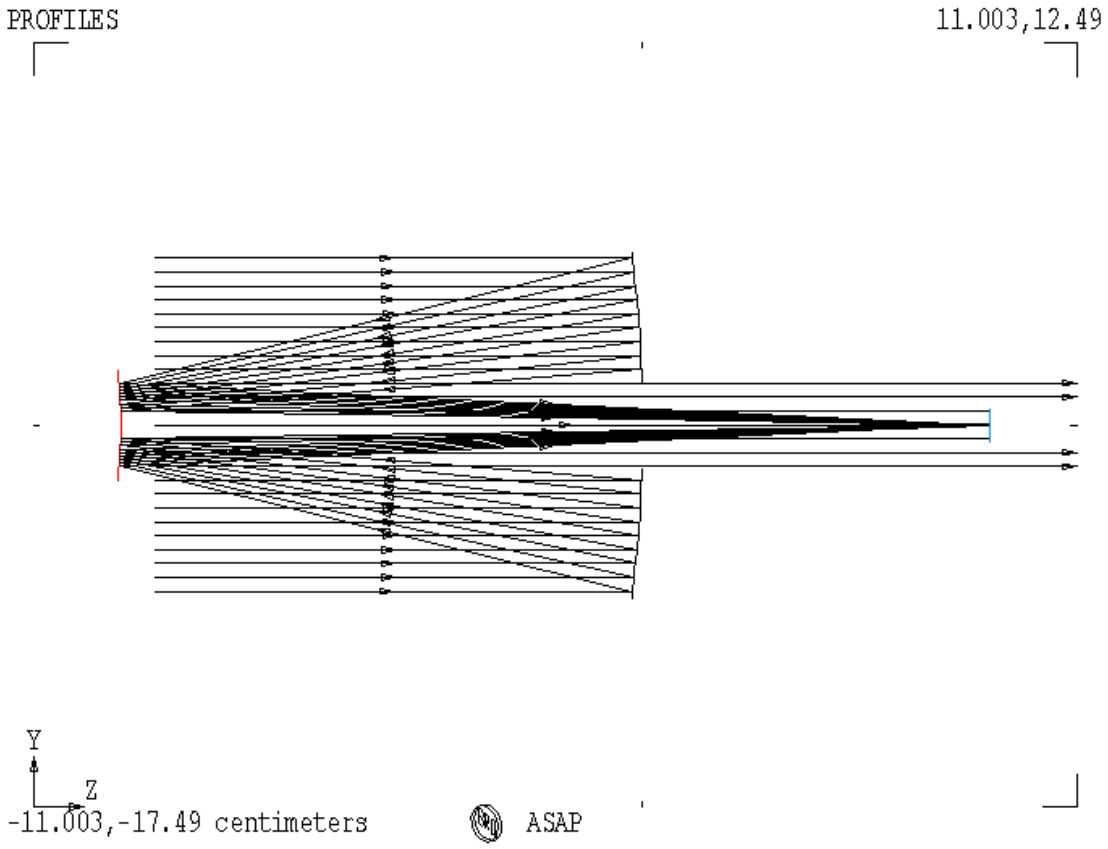


Figure 10.5 While clever positioning of the source can sometimes allows us to keep the optical system simple, we have, in this case, only introduced new problems. Without the "shadow" produced by the secondary mirror, some rays are able to pass directly through the hole in the primary mirror and land on the detector. This result is impossible in the real system, where they would be absorbed on the back of the secondary.

The real problem is that our model is not complete enough. In any realizable system there is no such thing as an infinitely thin surface. The real secondary also has a back, an edge, and probably a mechanical mount that prevents any rays from every reaching the front surface of the mirror from behind. While we can safely neglect the physical thickness of the primary and the detector, we cannot do so with the secondary. The correct solution is to add an absorbing back, as shown in Figure 10.6.

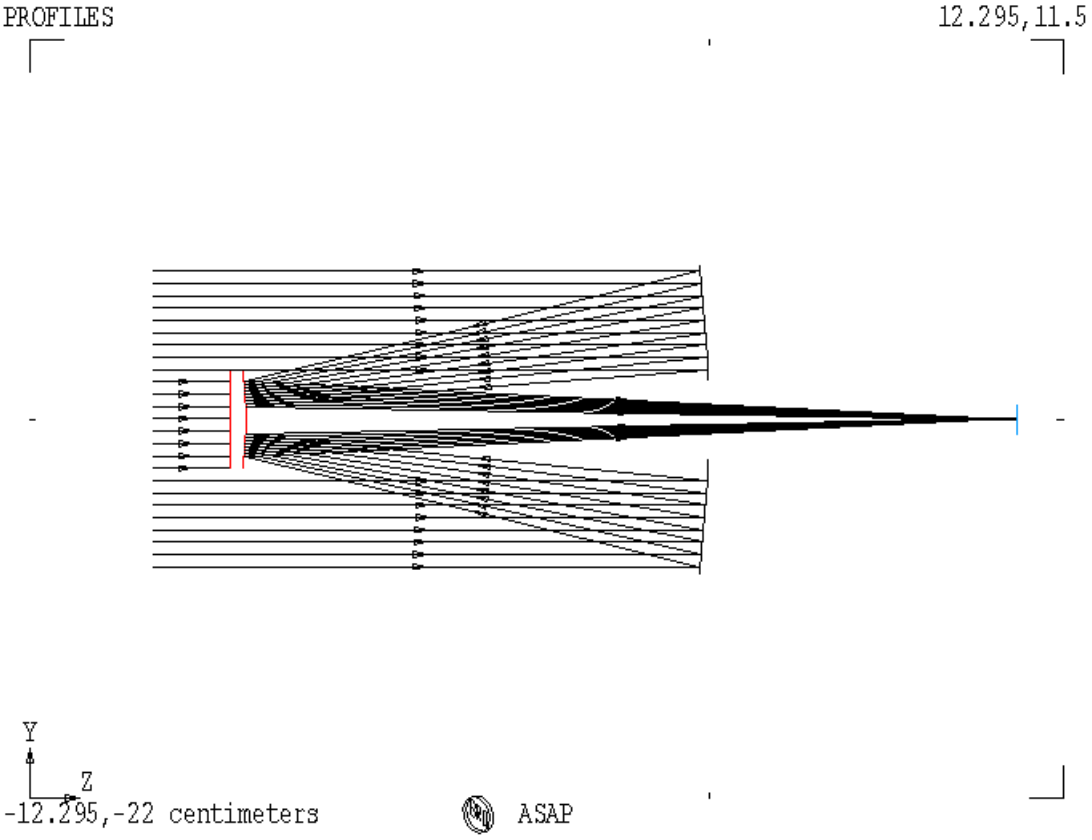


Figure 10.6 The correct solution for ray tracing in the Cassegrain telescope model is to add an absorbing back surface to the secondary mirror.

Note: In large systems, ray tracing is significantly faster if you define only the minimum number of surfaces. Many new users make the mistake of modeling every threaded screw hole and washer in the system, perhaps because that level of detail is already available in a CAD design. While safe and conservative, this approach can be costly in terms of ray-trace time. ASAP must do at least a quick check of every object at each step in the trace, to make sure that the rays do not intersect it next.

But what if you do not know what is important and what is not? Perhaps this is why you purchased a non-sequential ray-trace code. The answer often is to start as simply as you can, and add complexity as you discover problems, as we have done here.

Starting Rays on Surfaces

Always define the rays at least a small distance in front of objects you have defined.

Rays should never be defined at exactly the same position as an object in your system model. The rays, when traced, may interact with the surface, or they may not. Figure 10.7 shows what happens when rays were defined at the exact position of the secondary mirror's new back. ASAP is a computer program, limited by the precision of the calculations it is doing. If the ray positions exactly correspond to a surface location, the future of the ray depends on round-off error, and the details of the ray-intersection algorithms the program uses. Always define the rays at least a small distance in front of objects you have defined.

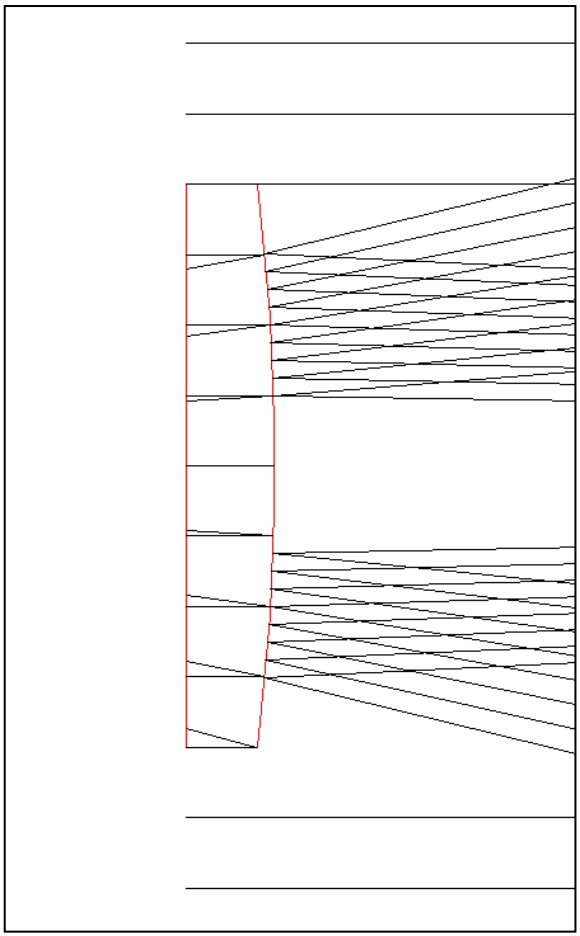


Figure 10.7 If you define rays at the exact position of a surface, the results are unpredictable. In this case, the rays were defined at the precise location of the back of the secondary mirror. All the rays missed the absorbing object, and found themselves inside the secondary mirror.

Duplicate Surfaces

Perhaps the strangest ray-trace result of all is shown in Figure 10.8. Some rays behaved correctly, while others passed directly through the reflective primary mirror, as if it were not there. What does this mean? Surely there has been an error. The answer in most cases can be found in the ASAP Workspace window, also shown in Figure 10.8. Every surface is listed twice.

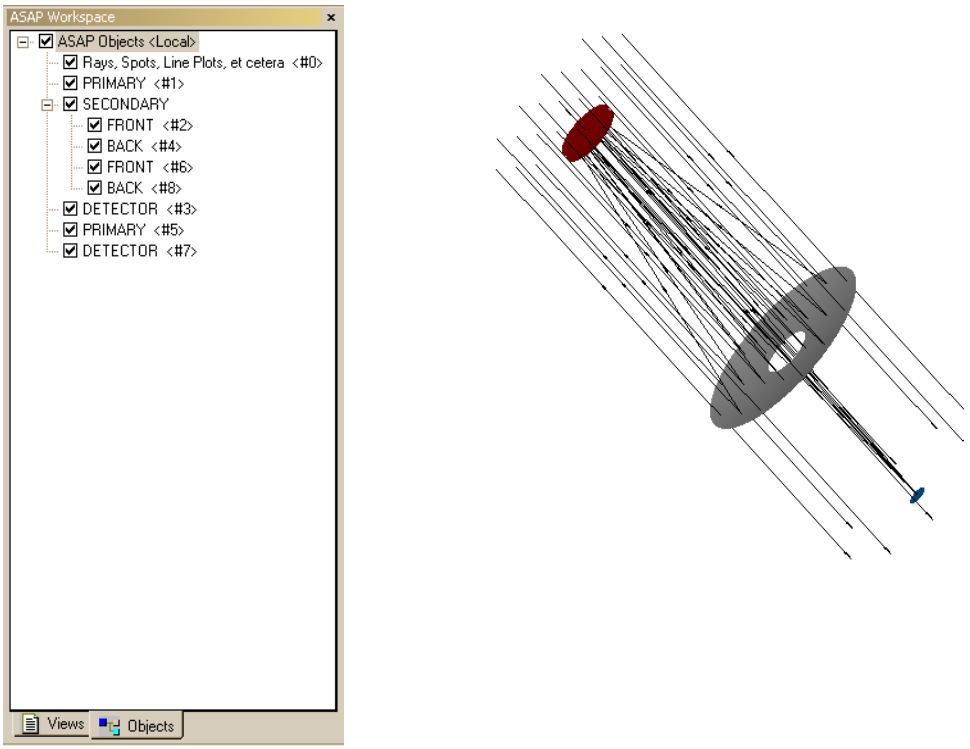


Figure 10.8 When some rays seem to pass directly through objects (right), the cause is usually due to duplicate surfaces. The **Objects** tab in the **ASAP Workspace** window (left) shows that two copies of each object were defined. This duplication occurs when we forget to click the **End** button before running the **Builder** a second time.

Once again, this duplication confuses the ray-trace algorithm. Sometimes, ASAP detects one of the duplicated surfaces, sometimes the other, and occasionally neither! The cause, once again, the limited precision of any computer calculation, round-off errors, and the details of the ray-trace algorithms used by ASAP.

Note Some explanation is needed for the choices of the colors of the rays and ray paths shown in this and previous chapters. Most of the examples in this Primer have no wavelength or only one wavelength specified. In this case, the rays and ray paths are plotted in colors given by the color order shown in on-line Help for the COLORS command. This color order corresponds to the order of source creation, where the first source is white or black, the second source is red, and so on.

When sources of different wavelengths are created, the colors range from blue to red, corresponding respectively to the wavelengths ranging from shorter to longer. This color variation is only a relative variation. The colors are not chosen to represent absolute spectral colors. That is, if wavelengths of 550 and 560 nanometers are chosen, the colors do not correspond to the spectral color meaning of these wavelengths. Instead, the 550 nm source is shown in blue and the 560 in red.

If all sources have the same wavelength, but you wish to show them in different colors, corresponding to a spectral relationship, add a small variance to the wavelength of each source. The colors are then shown as stated above.

Summary

In this chapter, we have discussed the following new commands:

ASAP Commands	Menu	Description
`TRACE`	**Trace> Trace Rays**	Trace rays through the system without graphics.
`TRACE PLOT`	**Trace> Trace Rays** (**Plot** Options	Trace rays through the system, plotting their paths in the **Plot Viewer**. The dialog box also allows us to include a profile or faceted view of the geometry in the same figure.
`MISSED ARROWS`	**Trace> Trace Rays** (**Missed Rays** Option	Show the final direction of any rays that stopped before reaching an absorbing surface, because there were no more objects along their path.

Several important lessons were presented in the examples in this chapter:

- Rays should never be created on or too close to a surface.
- Rays do not distinguish between the front and the back of a surface.
- Rays behave unpredictably when they encounter two surfaces occupying the same space.
- The first time you trace rays through a system, always plot them over the geometry to verify that everything is behaving as you expect.
- The 3D Viewer is an important tool for studying ray paths through a three-dimensional system.
- The 3D Viewer does not display “missed after” rays unless we ask it to, and specify the length of the arrows.
- To trace the rays again, you must redefine the source.

After finishing the exercise that follows, you will have completed three of the four basic steps:

1. √ Building the system model
2. √ Creating sources (rays)
3. √ Tracing the rays
4. Performing the analysis

Analysis is the subject of the next chapter.

Exercise 4: Tracing rays through the Cassegrain telescope

1. Recover the Builder file from your original Cassegrain telescope model, used in “Exercise 2: Cassegrain Telescope” on page 122.
2. Add an absorbing back to the secondary, giving this mirror a thickness of 0.75 cm.
3. Create a grid of rays parallel to the z axis that illuminates the primary. Assign a flux (luminous flux) of 8×10^{-10} lumen.
4. Create a graphic showing a profile of the system geometry, combined with the ray paths as they trace through the system.
5. Repeat the procedure, producing a three-dimensional view of the geometry and ray paths.

APPENDIX

10A

SCRIPTS FOR CHAPTER 10

The following ASAP scripts are referenced in Chapter 10.

Script 10-1

```
!!THIS IS THE COMMAND SCRIPT PRODUCED FROM THE PULL-DOWN MENUS.
WINDOW Y Z
PROFILES PIXELS 201 OVERLAY
TRACE PLOT
```

Script 10-2

```
!!THIS IS THE COMMAND SCRIPT PRODUCED FROM THE PULL-DOWN MENUS.
WINDOW Y Z
PLOT FACETS 5 5 OVERLAY
TRACE PLOT
```

Script 10-3

```
!!THIS IS THE COMMAND SCRIPT PRODUCED FROM THE PULL-DOWN MENUS.
WINDOW Y Z
PLOT FACETS 5 5 OVERLAY
MISSED ARROWS 10
TRACE PLOT
```

CHAPTER

11

BASIC ANALYSIS

In this chapter, we introduce some basic analysis tools and concepts. This is generally the most interesting step in the modeling process, because we can begin to answer questions about the optical behavior of the model. ASAP provides both numerical and graphical tools to accomplish this.

In this introduction to analysis, we will show methods for producing a variety of important results:

- Total power on objects
- Average ray positions
- Average ray directions
- Spot diagrams
- Best focus position

In Chapters 18 through 21, we return to the analysis topic again to discuss tools for performing radiometric and photometric calculations with ASAP. These additional tools will allow us to calculate power distributions—either as a function of position or direction—and other elements of formal radiometric analysis. The accessibility of the basic ray data maintained by ASAP, and the flexibility of the program allow us to organize the information in many different ways to suit the needs of a particular project. While some of the more advanced analysis techniques are best accomplished with command scripts and the macro language, your understanding of the underlying concepts begins here.

Basic Concepts of Analysis

In previous chapters, we learned not to think of ASAP rays as the ray-path lines that appear in the graphics during a ray trace. Instead, they are considered to be

points in space with direction vectors, flux values, and a variety of other parameters that carry important information about each ray in the system. As a ray moves through an optical model, most of these parameters change. At any given time, they describe the *current* state of the ray.

Unless instructed otherwise, ASAP analysis calculations are performed using all rays on all objects.

ASAP performs virtually all its analysis calculations by looking at the ray data. For example, if we want a plot of the positions of the rays (or some subset of them), ASAP produces the graphic simply by plotting the current coordinates of each selected ray in the requested window. If we want to know the sum of the fluxes of all rays located on a given object, ASAP consults the ray table, sorts the rays by object, and sums the flux values object by object.

We often use an object named "detector" for that place in our system model where we expect the rays to accumulate after a ray trace. There is nothing special about this detector, however, other than our assignment of an interface that neither reflects, nor transmits. This special interface causes the rays to stop there without further altering their flux. Your system model can include as many objects with this interface as you like. We can then do analysis on any of these surfaces.

Analysis calculations are performed with the positions, directions, fluxes, and so forth that the rays have at the time the analysis command is issued.

As we have seen, rays sometimes stop on objects with non-zero transmission or reflection coefficients specified by their interfaces. This happens any time that a ray misses all other objects in the system. They have stopped because they have nowhere else to go. We can also perform analysis on these rays, which often allows us to discover more about them, and perhaps make modifications to the system design so that this power is not lost.

The flexibility to do analysis on any number of objects is a powerful feature of ASAP, but its use requires care. As you will see in the examples that follow, we will make sure we have isolated the correct subset of rays before we issue the analysis commands. One of the most important ASAP analysis concepts can be stated in one sentence:

- Unless instructed otherwise, ASAP analysis calculations are performed using all rays on all objects.

ASAP does not determine which rays are important to us, and which are not. We must provide the program with this information. This step is called ray isolation.

A second, equally important analysis concept is summarized here:

- Analysis calculations are performed with the positions, directions, fluxes, and so forth that the rays have at the time the analysis command is issued.

As we have said, the analysis commands operate on the ray table. Since that table exists from the moment we define a set of rays, it follows that analysis commands can be used at any time after rays have been defined, whether they have been traced or not. We were actually using analysis commands in Chapter 9, "Verifying Sources" on page 159, when we learned methods for verifying

sources. As we progress through this introduction to analysis, you will see that there is some overlap between the commands in the **Rays** menus and those in the **Analysis** menus. If you click the **Script** button to see the actual ASAP commands that the dialog boxes generate and send to the kernel, you will find that many of the same commands are used in both places.

Creating a System and Sources

Figure 11.1 shows a simple optical system consisting of an aperture stop, a singlet lens, and a detector. We have created two sources, and traced them through the system.

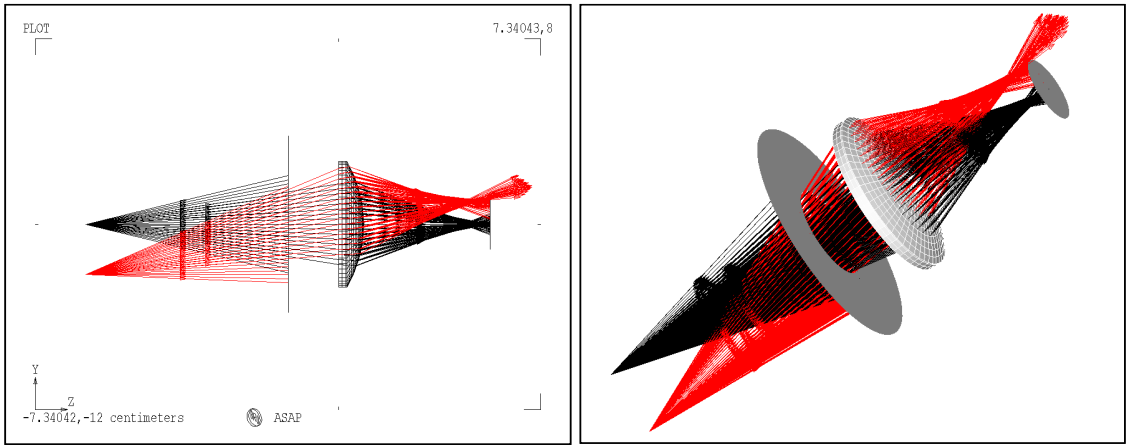


Figure 11.1 A profile view (left) and a three-dimensional view (right) of a simple optical system, consisting of an aperture stop, a lens, and a detector. We have traced two point sources, one on-axis (black paths) and one off-axis (red paths). The **Builder** file that created this geometry and the sources are reproduced in Figure 11.2. In the profile graphic, we reduced the number of rays in the grid to just one in the *x* dimension, giving us a simpler, two-dimensional ray fan for purposes of illustration. The view on the right shows the full 25x25 grid.

As Figure 11.1 shows, we have already traced rays through the system, and are ready to perform some analysis. Our goal is to analyze the image at the detector's assigned position, and determine whether this is the best place for the detector for best focus of the on-axis rays.

> ***Note:*** The example in Figure 11.1 illustrates many of the points in this chapter. If you would like to experiment with the new commands as you read through these descriptions, the Builder file used to create the system is shown in Figure 11.2.

*	Type	Option	Axis	Position	X Min	X Max	Y Min	Y Max	Number of X Rays	Number of Y Rays
SYS	System	New								
	Reset									
SYS	Units	CentiMeter								
CMD	Media	Glass	Media	2.4						
CMD	Coatings	Transmit	Properties		0	1				
CMD	Coatings	Absorb	Properties		0	0				
OBJ	Plane	L1.FRONT	Axis	Z	0.0	Ellipse	2.5			
MOD	Interface	Coating	Coating	'Transmit'	'Air'	'Glass'				
OBJ	Spherical	L1.Back	Z	1.0	-5	Ellipse	2.5			
MOD	Interface	Coating	Coating	'Transmit'	'Air'	'Glass'				
OBJ	Tube	L1.Edge	Axis	Z	0.0	2.5	2.5	1	2.5	2.5
MOD	Interface	Coating	Coating	'Absorb'	'Air'	'Glass'				
MOD	Bounds	Edge	-.2 +.3							
OBJ	Plane	Aperture	Axis	Z	-2	Ellipse	3.5	3.5	.4	
MOD	Interface	Coating	Coating	'Absorb'	'Air'	'Air'				
OBJ	Plane	Detector	Axis	Z	6	Ellipse	1.0			
MOD	Interface	Coating	Coating	'Absorb'	'Air'	'Air'				
EMT	Grid	Elliptic	Z	0	-2.5	2.5	-2.5	2.5	25	25
MOD	Source	Position	0	0	-10					
CMD	Move	To Point	0	0	-10					

Figure 11.2 The **Builder** file generates the system model and rays for most of the examples in this chapter.

Note: The **Move** command is used in the definition of both sources. We are using this new command to relocate the rays to the point specified on the **Source** line. Now, the rays not only appear to radiate from this point, but actually do start there, at the beginning of the ray trace. The MOVE command is discussed in Chapter 15, “Repositioning Geometry and Rays”.

Locating Rays

Once the rays have been traced, usually the first step in the analysis process is to find out where the rays are located.

Once the rays have been traced, usually the first step in the analysis process is to find out where the rays are located. While we might have some expectation of where they should have reached, and the graphics associated with the ray trace may have told us more, ASAP is now able to give us quantitative information. We can learn both how many rays are located on each object, and how much flux they represent.

The procedure for getting this information is to choose **Rays> Locate Rays**. The result appears in the Command Output window, as shown in Figure 11.3.

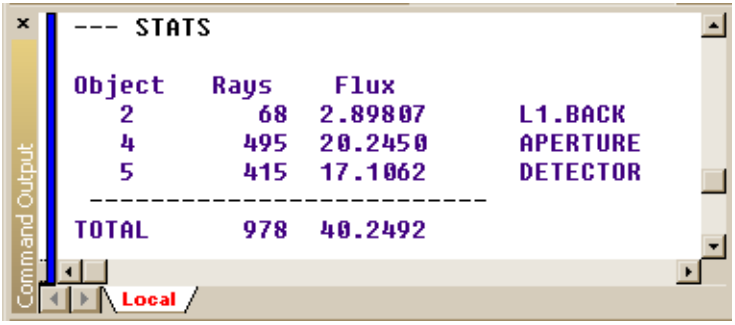


Figure 11.3 When we ask ASAP to locate the rays for us from **Rays> Locate Rays**, the result is printed in the **Command Output** window. In this case, most of the rays are on the aperture and the detector, but a few stopped on the back of the lens. The actual command that was executed by ASAP is the STATS command. It also shows the total flux on each of these surfaces.

ASAP has consulted the ray table, and tallied both the number of rays and the total flux on each object. We find flux located on three different objects: the aperture stop (named APERTURE in the model), the back of lens 1 (L1.BACK), and on the detector. The stop and the detector have an absorbing interface, so we might have expected rays to stop there. The rays on the back of lens 1 missed the detector. While the arrows in Figure 11.1 appear to show them leaving the system, we know from the previous chapter that they always stay on the last object they intersected.

See Chapter 11 Appendix, "Script 11-1" on page 219.

> ***Note:*** The ASAP command that performed the above work is STATS, which is short for "statistics". This is an example of a command that can be accessed either from the **Rays** or the **Analysis** menu. Since it sums fluxes as well as locates rays, STATS is also executed when we select **Analysis> Calculate Flux** and use the **Summary** option.

Choosing Rays for Analysis

One of the goals of this analysis is to find the best focus for the on-axis rays.

Recall that one of the goals of this analysis is to find the best focus for the on-axis rays. Before we can do this, we have to tell ASAP which rays are of interest. This is done in two steps:

1 Isolate only those rays that are on the object of interest.

In our case, we want to consider rays only on the detector. If necessary, we can proceed to step 2.

2 Further isolate the rays according to other criteria.

To find the best on-axis focus of the singlet lens, we also need to select only those rays that came from the on-axis source.

The ASAP menu provides a **Choose Rays** option, which you can access either from **Rays> Choose Rays**, or from **Analysis> Choose Rays**. In either case, the menu presents the choices: **Consider**, **Select Rays**, and **Discard Rays**. With the `CONSIDER` and `SELECT` commands, we can temporarily ignore some rays while we perform specific analysis tasks. **Discard Rays** permanently removes any rays not currently considered or selected. This option is rarely necessary, unless we are trying to create a new source out of a subset of old rays.

As mentioned above, the first step is to decide which objects should be considered during analysis. By this, we mean that ASAP ignores all other objects in all commands that follow. Rays that are on these objects are ignored as well, which is our goal in this case. This step is performed from **Analysis** (or **Rays**)**> Choose Rays> Consider.**

We are presented with the dialog box shown in Figure 11.4, on the left. Now we deselect everything but the detector.

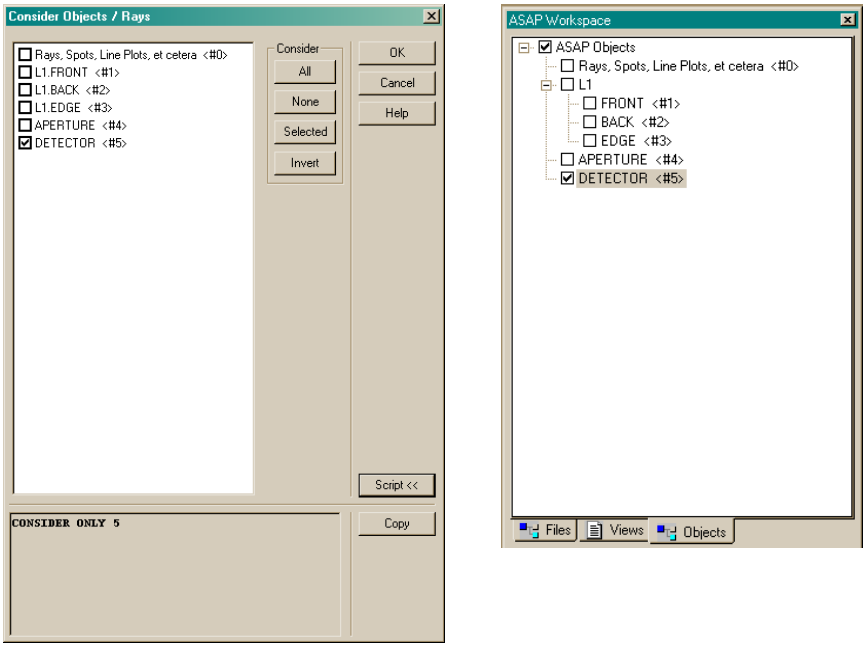


Figure 11.4 In the **Consider** dialog box (left)), we have deselected all objects, except the detector. After executing the `CONSIDER` command, the **Objects** tab in the **ASAP Workspace** window (right) reflects the changes. Alternatively, we could deselect the objects directly within **ASAP Workspace** (right), and achieve exactly the same result.

When you do this step, notice the change in the view for the **Object** tab on **ASAP Workspace** (on right in Figure 11.4). If you prefer, you can tell ASAP which object (or objects) to consider just by deselecting the objects in that window. With either approach, the view from the **Object** tab always reminds you which objects are currently being considered. You need only to reselect the boxes (by either method) to change the status of the objects again.

Note: The `CONSIDER` command goes beyond just ignoring the rays that happen to be on objects. Any objects that are not being considered will be ignored by all commands that follow. This includes plotting and viewing commands like `PROFILES` and `PLOT FACETS`, even during subsequent ray traces.

The second step, selecting rays by other criteria, is not always necessary. If we had created only one source, or created and traced the sources one at a time, we would be ready to proceed with the analysis. In our case, however, we have images of two point sources. With ASAP we have the ability to explore these independently. This step is done with **Analysis** (or **Rays**)**> Choose Rays> Select Rays**. The resulting dialog box is shown in Figure 11.5.

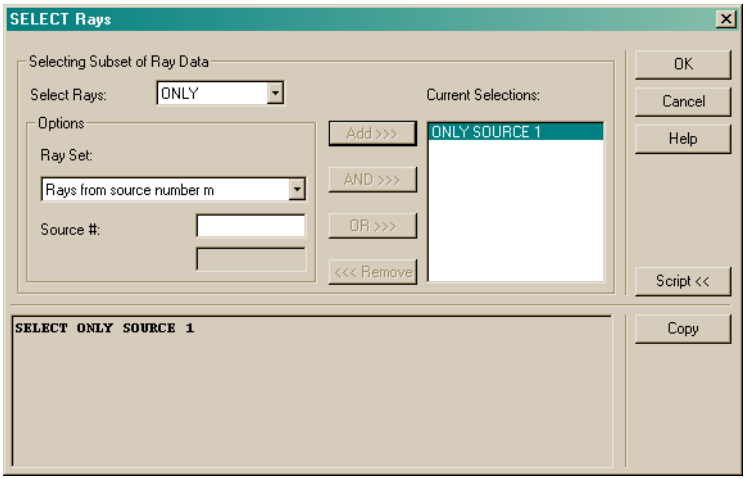


Figure 11.5 The **SELECT Rays** dialog box is used to further isolate rays for analysis, based on a variety of criteria. In this example, we are asking ASAP to ignore rays from the off-axis source.

At the **Select Rays** dropdown list, we are given three choices: **ALL**, **EXCEPT**, **ONLY**. Since we want to select rays from only one source, we choose **ONLY** this time.

Next, we move to the **Options** area, where the **Ray Set** dropdown list gives us many options:

All rays between *i* through *j* inclusive	(select a range of rays by number)
Rays from object *n*	(select rays that arrived here from object m)
Rays scattered from object *m*	(select depending on scattering object)
Rays from source number *m*	(select sources from the order of definition)
Rays scattered from source number *m*	(select depending on scattering object)
Rays belonging to path *l*	(select by path taken to arrive at current object)
Rays with axis radial coordinate > *r*	(select inner bound of a ring)
Rays with axis radial coordinate < *r*	(select outer bound of a ring)
Rays with X coordinate > *x*	(select lower x limit)
Rays with X coordinate < *x*	(select upper x limit)
Rays with Y coordinate > *y*	(select lower y limit)
Rays with Y coordinate < *y*	(select upper y limit)
Rays with Z coordinate > *z*	(select lower z limit)
Rays with Z coordinate < *z*	(select upper z limit)
Rays with X direction cosine > *a*	(select lower x direction cosine limit)
Rays with X direction cosine < *a*	(select upper x direction cosine limit)
Rays with Y direction cosine > *b*	(select lower y direction cosine limit)
Rays with Y direction cosine < *b*	(select upper y direction cosine limit)
Rays with Z direction cosine > *c*	(select lower z direction cosine limit)
Rays with Z direction cosine < *c*	(select upper z direction cosine limit)

As Figure 11.5 shows, we have selected the fourth choice (**Rays from source number m**), and entered the value **1** in the **Source #** box. The sources are numbered in the order in which they were defined.

In this example, the on-axis grid was defined first, so it is source number 1. After entering this information, we click **Add>>>** to place this condition for selection on the list.

When we click **OK**, the kernel executes the command, **SELECT ONLY SOURCE 1**. Rays that do not meet this condition are not used in any analysis commands that follow. When the operation is finished, the **Command Output** window shows the following message:

```
--- SELECT ONLY SOURCE 1
      489 ray flags changed
      489 rays now selected
```

See Chapter 11 Appendix, "Script 11-2" on page 219.

SELECT is one of the most powerful diagnostic analysis tools built into ASAP. ....As your models become more complex and realistic, you will find it easy enough to discover serious stray light or ghost image problems by looking at the basic analysis results.

Although we do not need to go further to solve our example problem, **SELECT** would also allow us to form logical combinations of the various selection criteria. **SELECT** is one of the most powerful diagnostic analysis tools built into ASAP. Even more criteria are available to you if you use the command language. As your models become more complex and realistic, you will find it easy enough to discover serious stray light or ghost image problems by looking at the basic analysis results. The **SELECT** command lets you go one step further, investigating the exact cause of the problem so that you can propose a solution.

> ***Note:*** When forming logical combinations of these selection criteria, be careful to understand the meaning of the logical **AND** and **OR** functions. While you might expect the combination,
>
> ```
> SELECT ONLY SOURCE 1 AND SOURCE 2
> ```
>
> to include all the rays in both sources, this logical combination results in no rays being selected. No rays are a member of *both* source 1 *and* source 2. You should use the logical **OR** operator to include rays from both sources.

Spot Diagrams

Having chosen the specific rays we want to analyze, we can explore the quality of the on-axis image on the detector plane. One common way to do this is with a spot diagram, which is a simple plot of the location of each ray in the image plane. The density of "spots" (ray markers) is a rough indication of how the power is distributed in the image. This density ignores differences that may exist in the flux of the individual rays, since one spot looks the same as any other.

In a more complex system, a significant difference may exist in the power represented by each spot in the diagram. We address this in Chapter 19, "Analyzing Total Flux and Positional Flux Distributions" when we perform quantitative irradiance calculations.

We have already learned a procedure that plots ray positions: **Rays> Graphics> Plot Positions 2D**. We used this method to verify grid sources in Chapter 9, "Verifying Sources". The executed ASAP command is **SPOTS POSITION**. Figure

11.6a shows the dialog box, and Figure 11.6b shows the resulting spot diagram for the on-axis rays on the detector of our singlet lens example.

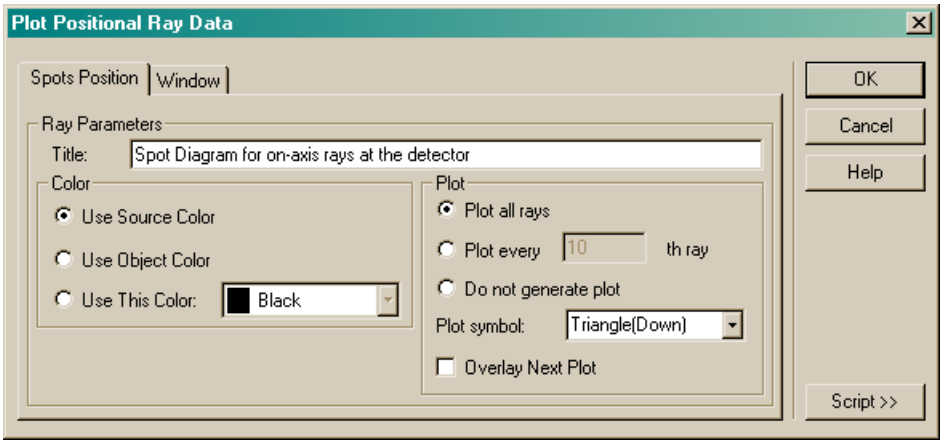


Figure 11.6a Options selected in the dialog box (**Rays> Graphics> Plot Positions 2D**) produced the spot diagram in Figure 11.6b.

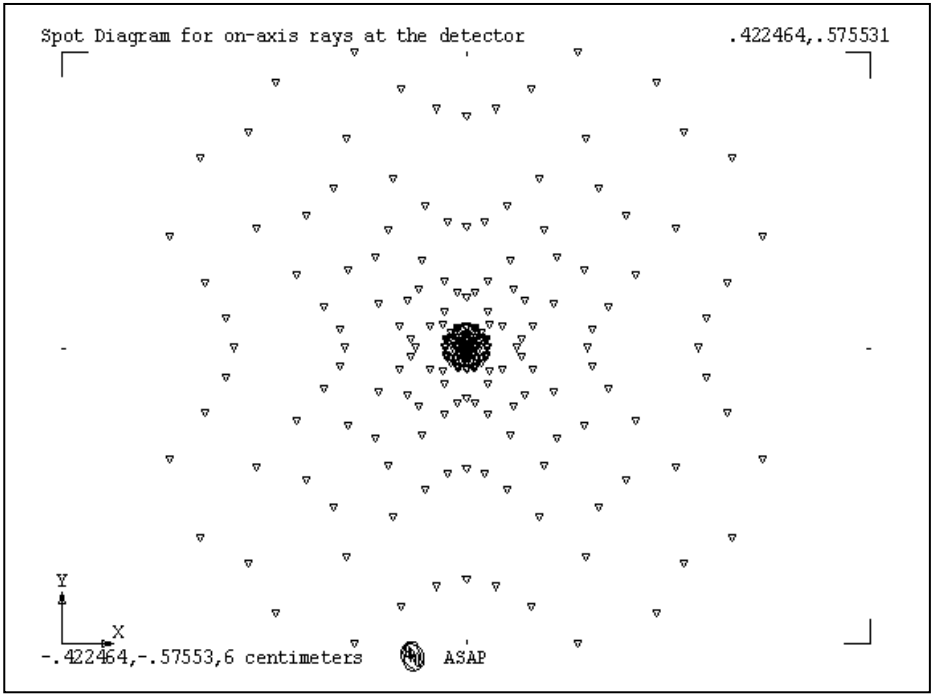


Figure 11.6b Spot diagram for the on-axis rays at the detector, projected into an x-y window.

Ray Statistics

If you want a more quantitative analysis of the image on the detector, you can calculate ray statistics with **Analysis> Calculate Flux**. Three options are available in that dialog box: **Summary, Position**, and **Direction**.

> ***Note:*** You can also reach these options from **Rays> Ray Information> Show Ray Statistics**.

The **Summary** option computes the total number of rays and total flux on each object currently considered. The ASAP command is **STATS**.

```
---STATS
Object    Rays     Flux
    5        241  10.3022       DETECTOR
  -------------------------
 TOTAL       241  10.3022
```

The **STATS** command is also used to locate all the rays before considering or selecting, which we accessed from **Rays> Locate Rays**. Now, however, only the currently considered and selected rays are included in the calculation.

The **Position** option gives us useful positional statistics for the currently considered and selected rays. The ASAP command is **STATS POSITION**.

```
--- STATS POSITION
Current Statistics for Object    5 - DETECTOR
  Total Flux =  10.3022     from      241 rays ( 24.64%)
                       X            Y            Z
        Centroid: -.989316E-08 -.578113E-11  6.00000
   RMS Deviation: 0.146840     0.146840     0.267437E-14
  Maximum Spread: -.422464     -.422464     -.710543E-14
               to 0.422464     0.422464     0.621725E-14
```

This version of **STATS POSITION** is useful for finding an image location (the centroid above), and quantitatively defining its size, expressed as both the root mean square (RMS) and maximum spread. The means and standard deviations are flux weighted to give the best possible estimate of the concentration of power.

The **Direction** option gives similar information for the direction cosine statistics. The ASAP command is **STATS DIRECTION**.

```
--- STATS DIRECTION
Current Statistics for Object    5 - DETECTOR
  Total Flux =  10.3022     from      241 rays ( 24.64%)
                      A            B            C
        Centroid: -.854625E-08 -.107988E-09  1.00000
   RMS Deviation: 0.186075     0.186075     0.424826E-01
  Maximum Spread: -.358986     -.358986     -.779242E-01
               to 0.358986     0.358986     0.111022E-15
```

The centroid printed here is the renormalized average direction vector, showing that these rays were traveling down the z axis. As with the **Position** option above, these values are based on a flux-weighted calculation.

Finding Best Focus

ASAP is able to calculate the best three-dimensional focal point, based on the current directions of the rays being analyzed.

We could experiment with different detector positions to find the best focus. For example, we could move the detector and repeat the ray trace until we found a minimum RMS deviation, using the **Analysis> Calculate Flux> Position** option. Fortunately, there is a much easier way.

Among the analysis tools provided by ASAP, you will find **Analysis> Focus Rays**. ASAP is able to calculate the best three-dimensional focal point, based on the current directions of the rays being analyzed.

The basic method is illustrated in Figure 11.7a, Figure 11.7b, and Figure 11.7c on page 213 and subsequent pages.

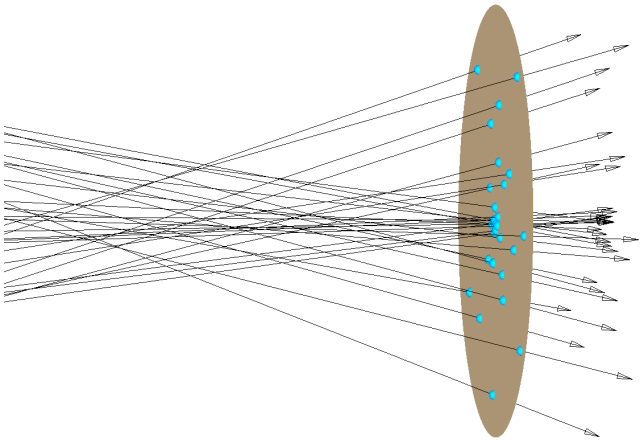


Figure 11.7a As a result of the ray trace, a subset of rays is isolated on the detector, ready for analysis. The current ray locations are shown as small, blue spheres. The rays' direction vectors are shown extending through the detector plane.

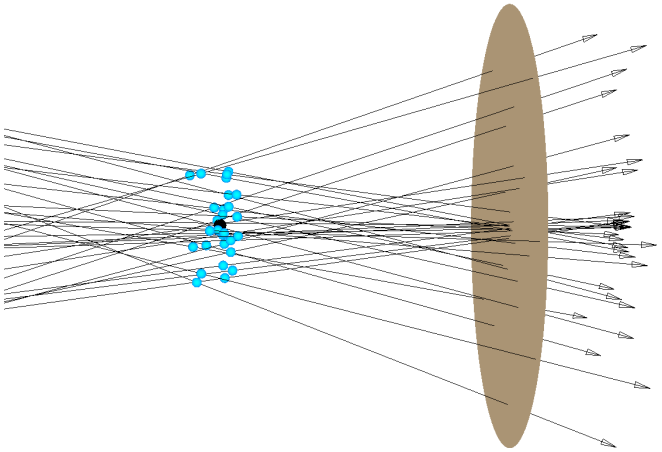


Figure 11.7b The `FOCUS` command allows ASAP to slide the rays forward or backward, along their direction vectors, like beads on a wire. ASAP calculates a centroid point so that the sum of the squares of the distance of each sliding ray, from this position, is also a minimum. The ray markers in this figure are as close as they can get to this centroid, constrained as they are to slide along their direction vectors. The rays are never actually moved to this place. ASAP reports only the coordinates of the centroid (the partially obscured black dot in this figure), leaving the rays on the detector.

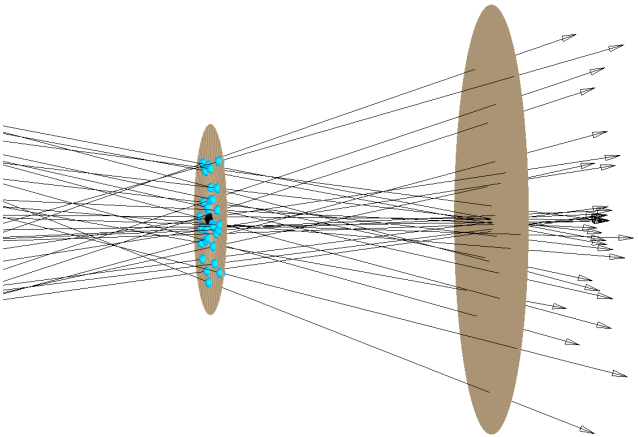


Figure 11.7c The `FOCUS MOVE` command not only calculates the centroid location, but also moves the rays (still constrained to slide along their direction vectors) to a plane passing through that centroid. The plane will be a z plane, if the average direction of all the rays is in the z direction.

In effect, the rays (those points in space located on the detector after the ray trace) are permitted to slide along their direction vectors, either forward or backward, until the sum of their distances squared from the flux-weighted centroid is as small as possible. Mathematically, we are trying to find a centroid point that minimizes the parameter *D* in the following expression:

$$D = \sqrt{\frac{\sum_j f_j d_j^2}{\sum_j f_j}}$$

where f_j is the flux of the j^{th} ray, and d_j is the distance from the j^{th} ray path to the centroid. This calculation is flux weighted, so the rays having the largest flux will have the greatest influence on the result.

The **Analysis: FOCUS** dialog box (**Analysis> Focus Rays**) gives us three choices for focus (see Figure 11.8).

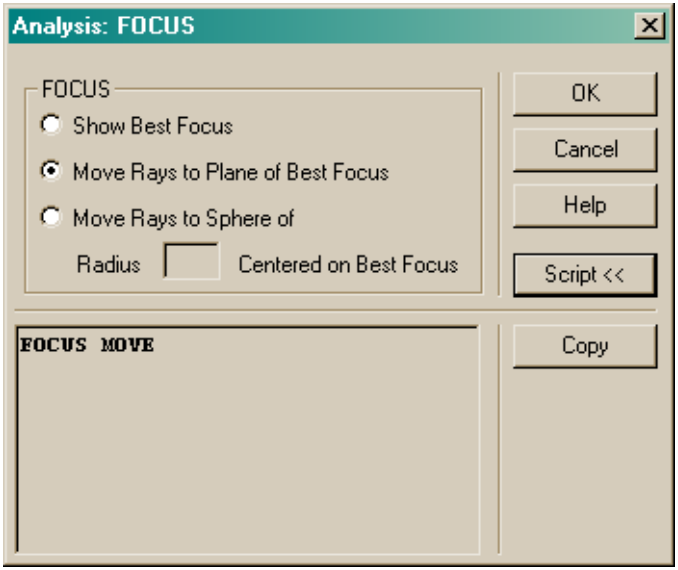


Figure 11.8 In the **Analysis: FOCUS** dialog box, **Show Best Focus** reports only the coordinates of the point in space that is at the centroid of the volume (Figure 11.7b). **Move Rays to Plane of Best Focus** performs this same calculation, and then moves the rays from the current position to a plane passing through the centroid (Figure 11.7c). We also have the option to move the rays to a reference sphere with the calculated centroid point at its center.

The **Show Best Focus** option prints the centroid location in the Command Output window, along with other useful information derived during or after the calculation. The ASAP command is **FOCUS**.

```
--- FOCUS
Least Squares Focus Calculation for      241 Rays:
                      X             Y             Z
  Centroid Point -.2229922E-07 -.1539269E-08  5.364451
  RMS Deviations 0.6682E-01    0.6682E-01    0.2740E-01
  Mean Direction  0.0000000     0.0000000     1.0000000


 Total Flux =  10.30     RMS Blur Diameter = 0.1889915
 Maximum Ray Angle from Mean =  22.7685 degrees, F/ 1.191    , 0.3870
N.A.
```

The **Move Rays to Plane of Best Focus** prints the same information to the Command Output window, and also moves the rays to a plane passing through the centroid. The ASAP command is **FOCUS MOVE**. If this option is selected, you can make a new spot diagram showing the effects of the focusing process (see Figure 11.9).

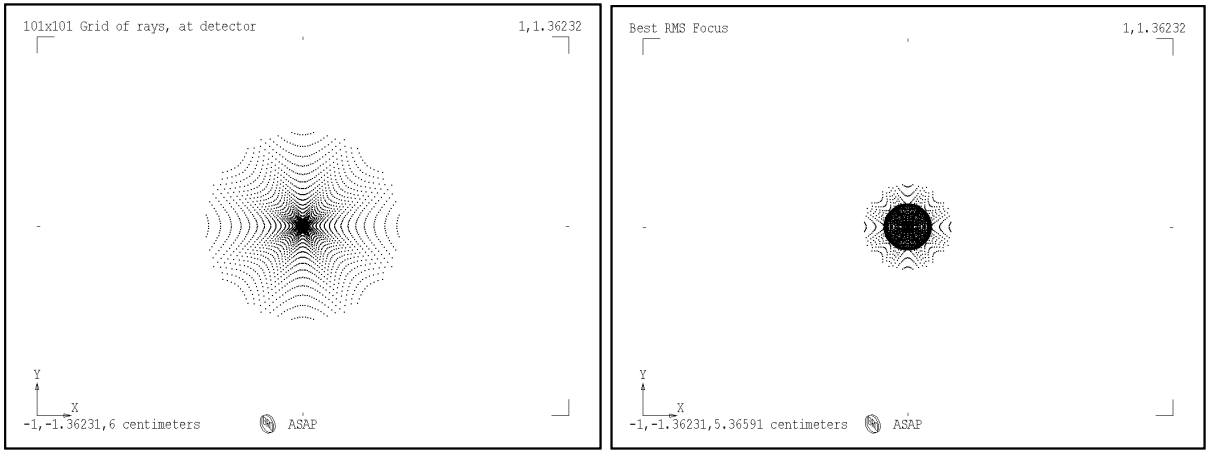


Figure 11.9 Show Best Focus only reports the coordinates of the point in space that is at the centroid of the volume of minimum approach (left). **Move Rays to Plane of Best Focus** performs this same calculation, and then moves the rays from their current position to a plane passing through the centroid (right).

Rays are always associated with the last object they touched during a ray trace, even though their coordinates have changed.

Note: We have stated before that ASAP rays are always associated with an object. In the case of **FOCUS MOVE**, however, we have allowed the rays to leave the detector. Which object are the rays associated with now? If you use **Rays> Locate Rays** after **FOCUS MOVE**, you will find that they still report being "on" the detector. Rays are always associated with the last object they touched during a ray trace, even though their coordinates have changed.

The final option in the **Analysis: FOCUS** dialog box is **Move Rays to Sphere**, where we must enter a sphere radius. This designation allows us to move the rays to a reference sphere, with a surface located at the entrance pupil of the system (roughly the position of the lens in this example). Now, in addition to the centroid data, ASAP also prints optical path deviations both as peak-to-valley measures and as RMS wavefront variance:

```
--- FOCUS MOVE 5.364451
...
  Flux weighted statistics:
  P-V Optical Path Difference =  2.4668179E-02
     Wavefront Variance (RMS) =  7.2155828E-03
```

If the rays were coming to a perfect geometrical focus (that is, focussing to a point), these values would be zero. The resulting finite variations are a direct measure of the magnitude of the aberrations present in this imaging system.

Summary

In this chapter, we have discussed the following new commands:

ASAP Commands	Menu	Description
STATS	**Rays> Locate Rays** or **Analysis> Calculate Flux** (**Summary** option)	Report total number of rays and total flux on each object.
CONSIDER	**Rays> Choose Rays> Consider** or **Analysis> Choose Rays> Consider**	Choose objects for consideration for future graphics or analysis.
SELECT	**Rays> Choose Rays> Select** or **Analysis> Choose Rays> Select**	Choose rays for analysis based on a variety of criteria.
STATS POSITION	**Analysis> Calculate Flux** or **Rays> Ray Information> Show Ray Statistics>** (**Position** option)	Calculate positional ray statistics.
STATS DIRECTION	**Analysis> Calculate Flux** or **Rays> Ray Information> Show Ray Statistics** (**Direction** option)	Calculate directional ray statistics.
FOCUS	**Analysis> Focus Rays>** (**Show Best Focus** option)	Find and print centroid of best RMS focus.
FOCUS MOVE	**Analysis> Focus Rays>** (**Move Rays to Plane of Best Focus** option)	Find and print centroid of best RMS focus, and move rays to a plane passing through this point.
FOCUS MOVE n	**Analysis> Focus Rays>** (**Move Rays to Sphere** option)	Find and print centroid of best RMS focus, and move rays to a reference sphere of radius n.

Perhaps the most important points to remember from this chapter are these:

1. Before analysis can begin, you must tell the program which rays to analyze.
2. You use the **CONSIDER** command (**Rays> Choose Rays> Consider**) to tell ASAP which objects to consider in any commands that follow. When you choose not to consider an object, this affects both the object and any rays that might be associated with that object.
3. You use the **SELECT** command (**Rays> Choose Rays> Select**) to further isolate a particular set of rays. This isolation may be necessary if you have more than one source.

The exercise at the end of this chapter gives you an opportunity to practice using the basic analysis tools introduced here. These tools include only the most basic analysis tools available in ASAP. Chapters 18 through 21 are also dedicated to the topic of analysis, allowing us to calculate and display quantitative flux distributions as a function of position or direction.

You have now seen all four elements of an ASAP project:

- √ Building the system model
- √ Creating sources (rays)
- √ Tracing the rays
- √ Performing the analysis

In the remaining chapters of this Primer, we revisit not only the analysis topic, but also the other three project elements several more times. You will learn more sophisticated tools within each area, and also practice and reenforce the basic skills introduced so far.

Exercise 5: Best Focus of a Spherical Mirror

1 Build a model of a spherical mirror with the following specifications:

Mirror vertex is at z = 10 centimeters

Diameter is 5 centimeters

Radius of curvature is -20 centimeters

Reflectivity is 98%, Transmission 0%

Note: ASAP does not require that you conserve energy. Assume that the balance was "absorbed", perhaps converted to heat.

2 Use **System> Profiles** and the cursor to verify the location and diameter of your mirror model.

3 Define a collimated 21x21 grid of rays. Give the grid a direction parallel to the z axis, and a diameter that matches the mirror dimensions.

4 Verify the source size and ray directions in the **3D Viewer** using **System> Plot Facets** and **Rays> Graphics> Plot rays 2D** (see "Combining Ray and Geometry Graphics" on page 164 of Chapter 9. Use a Y-Z window that goes from -40 to 40 in y, and -10 to 110 in z.

5 Trace the rays, graphing the geometry with **Plot Facets**. Use the same window as above.

6 Use **Rays> Locate Rays** to verify that all of the rays are now on the mirror.

7 Find best focus, and move the rays there using **Analysis> Focus Rays** with the **Move Rays to Plane of Best Focus** option.

8 Use **Analysis> Calculate Flux** (**Position** option) to calculate positional statistics.

9 Make a spot diagram at best focus (**Rays> Graphics> Plot Positions 2D**).

Some questions to test your understanding

- What object do the rays belong to after you move them to best focus?
- What are the coordinates of the centroid calculated by `FOCUS`?
- What is the RMS blur diameter at best focus?
- What are the RMS deviations, and maximum deviations of the rays (in x and y) at best focus?
- How do the RMS deviations relate to the blur diameter in the `FOCUS` output?
- Why are the positional statistics slightly different from the RMS deviations reported by the `FOCUS` command?
- Which object are the rays "on" after they are moved to best focus?

APPENDIX

11A

ASAP SCRIPTS FOR CHAPTER 11

The following ASAP scripts are referenced in Chapter 11, “Basic Analysis”.

Script 11-1

```
!!THIS IS THE COMMAND SCRIPT PRODUCED FROM THE PULL-DOWN MENUS.
!!FIRST TRACE THE RAYS USING THE SAME COMMANDS AS IN CHAPTER 10.
WINDOW Y Z
PLOT FACETS 5 5 OVERLAY
MISSED ARROWS 1.0
TRACE PLOT
!!THE NEXT COMMAND LOCATES THE RAYS.
STATS
```

Script 11-2

```
!!THIS IS THE COMMAND SCRIPT PRODUCED FROM THE PULL-DOWN MENUS.
!!THE NEXT COMMAND CHOOSES RAYS ONLY ON THE DETECTOR, OBJECT 5.
CONSIDER ONLY 5
!!THE NEXT COMMAND SELECTS RAYS ONLY FROM THE FIRST SOURCE.
SELECT ONLY SOURCE 1
```

CHAPTER

12

MORE ABOUT THE ASAP LANDSCAPE

You were first introduced to the ASAP landscape in Chapter 2, "A Short Tour" on page 33. At that time you were given only a quick tour of the user interface, which we call the landscape, with just enough information to get started using ASAP. We would like to return to this topic now to introduce several other features that will allow you to use ASAP more efficiently.

In this chapter, we will discuss three broad topics:

- **ASAP Workspace**. If you choose to define an ASAP project, you will be able to manage multiple files of many types more easily and efficiently. Frequently used files can be organized in a simple index that becomes part of the ASAP Workspace window. Any file can be opened for editing from this file list, or run without opening once you are finished developing the file. You have the option to link multiple files together, allowing them to run in a prescribed order.
- **Customizing the Landscape**. Since not every user agrees on the best and most efficient layout of a Windows application, ASAP gives you the ability to modify the landscape to suit your specific needs, computer hardware, and work style. The main ASAP menu bar and the status bar at the bottom of the landscape area are the only elements of the work environment that cannot be undocked, moved, and resized. Once you settle on a landscape that works well for you, ASAP will remember it, and make it the default in the future. Customize the toolbars for the main window and the Builder by right-clicking in the toolbar and selecting **Customize**.
- **Custom Toolbar buttons**. ASAP has predefined several buttons on the toolbar that are considered to be the most common ASAP functions. While these buttons perform a function that can also be accomplished from a menu or the scripting language, they represent a convenient, time-saving

shortcut to these tools. The particular work that you do, however, may require the frequent use of ASAP commands that were not included in the custom buttons. ASAP provides 20 additional custom buttons that can be programmed to suit your needs.

- **Quick Start toolbar**. You can easily access sources and apply media, scatter models, and roughness models to objects. Direct access to the Example script database is also available. Glass catalogs include Schott, Ohara, French, Hoya, Sumita, and Hikari. To view the dispersion chart for a glass, right-click the glass name. Media and models can be dragged onto objects in **ASAP Workspace** from the Quick Start toolbar after creating geometry with the Builder or a script file. From the User INR Scripts page, define your own INR scripts and store them for future use. A catalog of pre-defined lenses is available. You can dock the Quick Start toolbar within the ASAP window.

Using ASAP Workspace

Many experienced ASAP users organize their ASAP work into projects. You can place each design that you analyze with ASAP into a different working directory, keeping all related project files together. Designating a working directory is done just by using **File> Set Working Directory** in the main ASAP menus. You have likely done this when you started working through the *Primer*, selecting a directory name like **ASAP Primer** as a place to perform work related to the *Primer* examples and exercises.

Using ASAP Workspace takes this concept one step further. Suppose you have used a variety of Builder files representing various experiments, temporary simplifications, and apparent false starts that you are not quite ready to discard. This will eventually cause clutter of its own. You may have found this to be true already, while working through the examples and exercises in the first 11 chapters of this *Primer*. What if you could select a few of these Builder files that you are actively using or referencing, and put them on a special “short list” that is important to you right now? Perhaps you would put only the exercise solutions on the list, because you are frequently using these to remind yourself how a specific command is used. These kinds of activities are the primary function of ASAP Workspace. It is easy to add or subtract files from this list at any time. This does not affect the actual Builder files on your hard drive in any way.

While the use of ASAP Workspace is optional, we suggest that you try setting up your *Primer* work as an ASAP project to gain some experience with it. The next few sections lead you through the process of setting up the project, and describe some of the features available to you once you have done this.

Setting up an ASAP Project

You can define an ASAP project at any time—when you start a new ASAP task, or later if you decide to organize an existing working directory. In either case, the same procedure is used:

1. If ASAP is not already running, start the program as usual.

2. From the main ASAP Toolbar, click the down arrow to the right of the **New** button. Choose **Project** from the drop-down list.
3. If documents are opened from a current ASAP session, a dialog box appears to ask if you want to close them. Answer **Yes**. You are prompted to save your work, if necessary.
4. In the **New Project** dialog box (Figure 12.1), navigate to the location where you would like to perform the work. If you are organizing an existing project, you should browse to your current working directory (**Primer Project**). If this is a new ASAP task, this dialog box also allows you to create new folders as needed.
5. Enter a name for the project in the **File name** window. We have used the name **Primer Project**. A file named **Primer Project.apf** will be added to the working directory. When you want to open this project again, browse to this file using **File> Project> Load**, or select it from the list in **File> Recent Projects**.

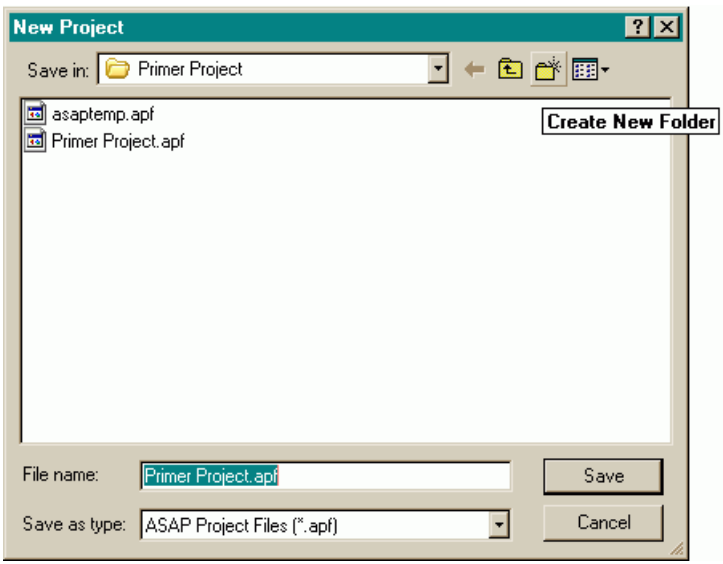


Figure 12.1 The **New Project** dialog box: We have browsed to the working directory, **Primer Project**, and named the new project, **Primer Project** in the **File name** field. The project location and name can be the same name or a different one.

Files Tab

After completing the procedure outlined above, there is only one significant change to your ASAP landscape, and that is a new tab in the ASAP Workspace window (Figure 12.2).

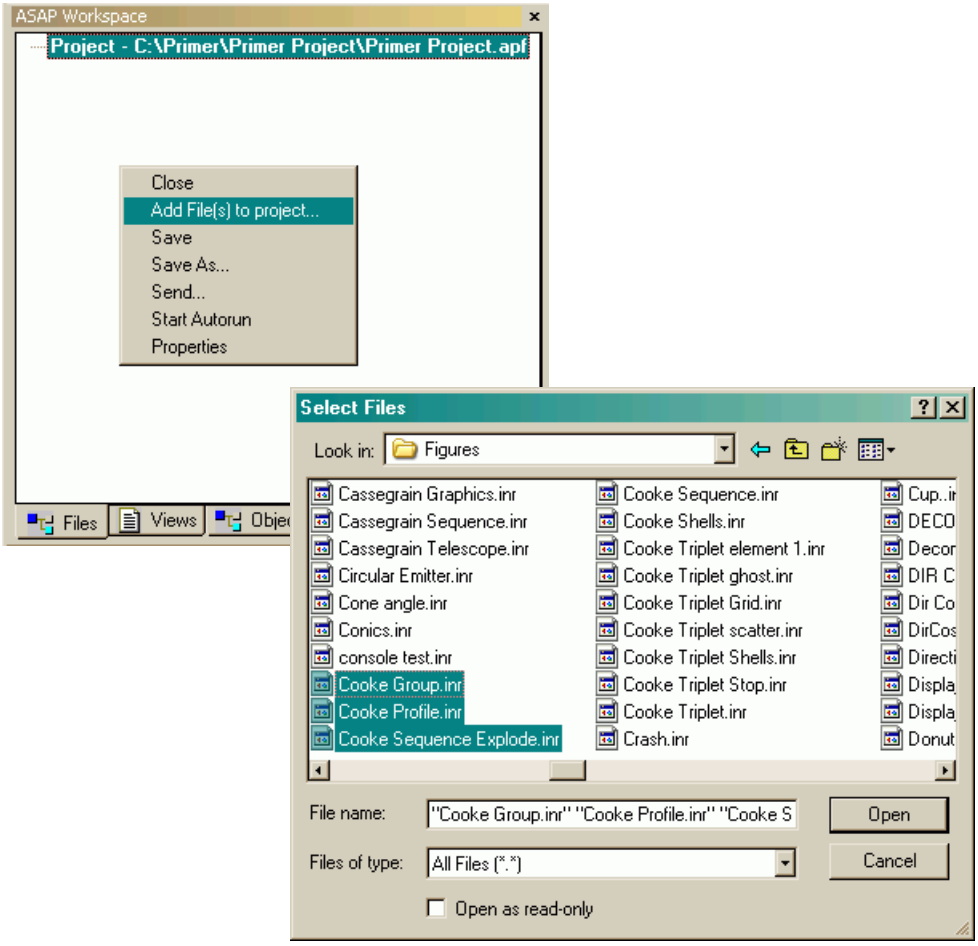


Figure 12.2 The **Files** tab in the **ASAP Workspace** window is on the left. New files are added by right-clicking in the background area to produce the **Select Files** dialog box (right). You can select multiple files by holding down the **Ctrl** key while clicking files in the list. After you click **Open**, ASAP adds these files to the **ASAP Workspace** window.

Select the **Files** tab to see active files in your project. The window is empty initially, except for the full path name of the project at the top. The next step is to add files. You can do this by right-clicking anywhere in the background of the window, and selecting **Add File(s) to Project**. Select one or multiple files in the resulting **Select Files** dialog box (also shown in Figure 12.2). You will

normally add the files in this or any other directory that you are actively using or referencing at this time. When you finish, click **Open** to complete the process. The files appear in the **Files** tab of the ASAP Workspace window after you click the plus sign to expand the list. You can change the order of the files by clicking and dragging a file to any position on the list.

Now that you have files on the list, you can quickly open one of them by just double-clicking on the file name. This step saves the time it would have taken to locate the file among a long list of others. It is particularly useful for files that are not in your working directory, since you will not need to browse to the file every time you want to look at it.

You can also run files on this list by right-clicking the file name, and selecting **Run** from the pop-up menu (Figure 12.3). The file does not need to be open to run it in this way, but it can be. You will be prompted to save the file if you have edited it since it was last saved to disk.

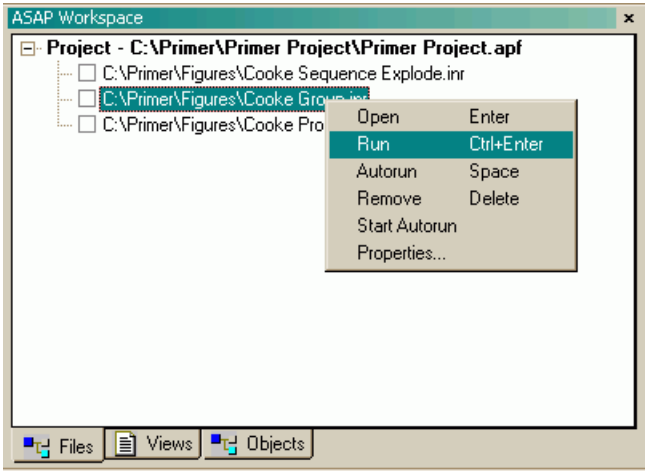


Figure 12.3 You can run a file from the **Files** tab just by right-clicking the file name and selecting **Run** from the menu.

As you can see from the pop-up menu in Figure 12.3, by right-clicking the file name, you can remove it from the list when you are no longer actively using it. Deleting the file name from this list does *not* remove the actual Builder file from your hard drive, however. The only change is the absence of the file name from the **Files** tab of the ASAP Workspace window.

Builder files (*.enx, enz file name extensions) are not supported in Projects. Howewver, you can add many other file types to the list. In general, they will open in the appropriate ASAP environment, and can be run as described above

when that is appropriate. Table 12.1 summarizes the various file types that you can add to the project list.

Table 12.1 File Types for Projects

File Type	Extension(s)	Opens in…	Run?
Command script	.inr	ASAP Text Editor	Yes
Macro library	.lib	ASAP Text Editor	Creates .mac files
Macro	.mac	ASAP Text Editor	Creates .mac files
Screen file (custom dialog)	.scr	ASAP Text Editor	Displays dialog
Text file	.txt	ASAP Text Editor	Yes, treated like .inr
Builder	.enx, enz	Builder	No
Scatter data	.hs .fit .atm	BSDF Fit utility	No
Plot/Chart file	.plr, plx	Plot Viewer	No
Distribution file	.dis	Display Viewer	No
Vector (3D) file	.vcr	3D Viewer	No
IGES (CAD export) file	.igs	IGES / ASAP Translator	Yes, same as Open
Oslo files	.len	Oslo to ASAP Translator	Yes, same as Open
Zemax files	.zmx	Zemax to ASAP Translator	Yes, same as Open
Code V files	.seq	Code V to ASAP Translator	Yes, same as Open
XML files	*.gtx	ASAP - importing XML files	Yes
All others	*.*	ASAP Text Editor	No

Autorun

While you may be content to just view or manually run the files on the project list, it is also possible to set up an **Autorun** sequence. Perhaps you have defined one Builder file that defines your geometry, another that defines an on-axis source, and a third that defines an off-axis source. You can turn this part of the list into an Autorun sequence by clicking and dragging the files into the desired top-down order, and clicking in the box to the left of each file, (Figure 12.4 on

page 227). A lightning bolt will appear in the box. Now, by right clicking in the background of the window, you can select **Autorun** to run these three files in order.

> ***Note:*** If you leave the lightning bolt in the **Autorun** box when you exit the project or close ASAP, the chain of files will automatically run next time the project is loaded. This can be useful if you always want to begin work with certain geometry or sources already defined, but it is not always desirable. To prevent this behavior, click the lightning bolts before you exit to remove **Autorun**.

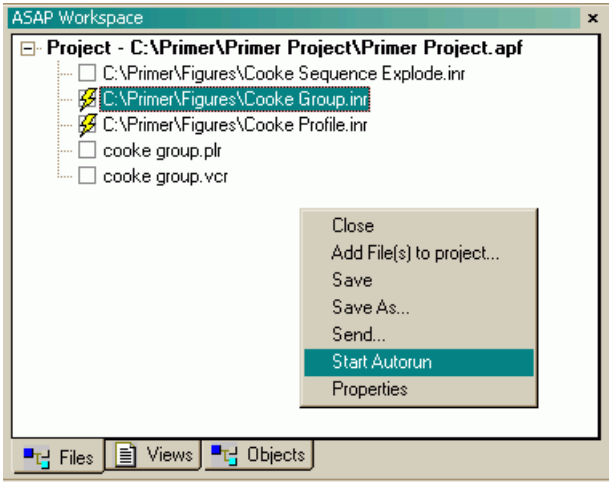


Figure 12.4 You can set up a file to run in sequence by selecting the box to the left of a file. When you right-click in the background of the window and select **Start Autorun**, ASAP runs the selected files in order, starting from the top. In this example, the file named **Cooke Sequence Explode.inr** is run first, followed by two files that define sources. Ray tracing and analysis can proceed. If you want to trace rays again, you only need to run one of the source files to define new rays, but you can run both. You can also quickly substitute different geometry by deselecting one box and selecting another one.

Project Preferences

ASAP allows us to set a variety of personal preferences for the user interface using the **File> Preferences** dialog box from the main menus. Three of these that are specific to projects can be found on the **General** tab (Figure 12.5). If you check the option labeled **Automatically load project on startup**, your current project starts automatically the next time you run ASAP. Without this setting, ASAP cannot start in project mode, and you will need to browse to your project with **File> Project> Load**. The option, **Auto Save Project**, automatically saves the current configuration when you exit the project or close ASAP. You can set a **Default Project Directory** to begin browsing from this location when you load a

project. Using a default project directory saves you time in finding the directories on your hard drive that contain your *.apf files.

> ***Note:*** Since we have not discussed the **Preferences** dialog box, this may be a good time for you to explore some of the other tabs. Some of the options on the **Builder**, **Plot Viewer**, and **3D Viewer** tabs may also be of interest.

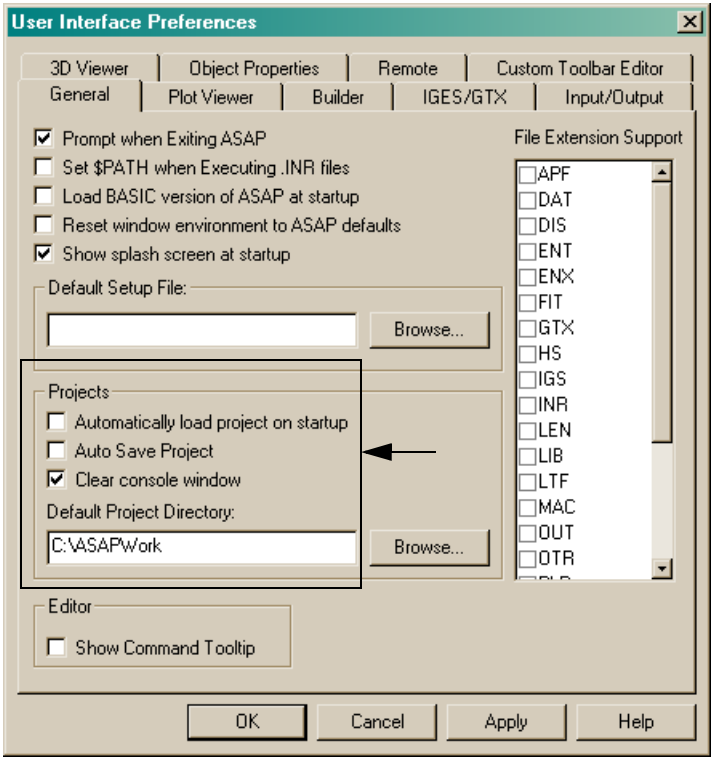


Figure 12.5 ASAP provides us with a variety of preference settings, which control the default behavior of many aspects of the user interface. Three options on the **General** tab affect us only when we use projects.

Customizing the Landscape

The **ASAP Workspace** *and* **Command Output** *windows are the primary means by which ASAP communicates with you about its current state.*

In Chapter 2, we provided a quick tour of the ASAP landscape in its default configuration (Figure 2.1). ASAP developers adopted this particular design to conform as much as possible to current Microsoft® Windows® user-interface standards. It has the advantage—particularly for new ASAP users—of keeping the ASAP Workspace and Command Output windows on top, rather than being covered by the Builder, Plot Viewer, 3D Viewer, or other document windows. These two windows, in particular, are the primary means by which ASAP communicates with you about its current state.

Depending on your circumstances, however, the default landscape may not be ideal for you. For example, if you are using a laptop computer or some other machine with a limited display size, the small area remaining in the **User Task Space** may not be sufficient. The first option for solving this, of course, is the **F11** key. Pressing this key temporarily maximizes any window to cover the screen space available to the ASAP application (see Chapter 3 under the heading, "Builder Introduction and Basics" on page 44*)*. However, a maximized window prevents us from looking at two or more windows at the same time, which is one of the major advantages of a multiple-document Windows interface.

Another possible way to increase the **User Task Space** is to shrink the size of the **ASAP Workspace** and **Command Output** windows. This, however, compromises their usefulness. If your need is extreme, two buttons are provided on the main ASAP Toolbar (**Toggle Workspace window**, and **Toggle Output window**) that allow you to temporarily close (or open) either or both of these large windows. This is certainly the least desirable option, however, since you may miss important error messages or information about which members of the **Object** list are currently considered.

These conflicting requirements often lead more experienced ASAP users to begin customizing the ASAP user interface, particularly by undocking and relocating key elements of the landscape.

Docking and Undocking

The term "docking" in the Windows context means that an element of the user interface (UI) can be attached or "docked" to the edge of the main application window, or to another docked element. The basic ASAP landscape consists of seven primary elements, most of which are dockable (see Table 12.2).

Table 12.2 Dockable elements on the ASAP landscape

Element	Dockable
Dynamic Menu Bar	No
ASAP Toolbar	Yes
User-Definable Toolbar	Yes
ASAP Workspace Window	Yes
Command Output Window	Yes
Command Input Window	Yes
Status Bar	No
Quick Start Toolbar	Yes
Builder Pop-up Menu	Yes

One advantage to undocking some of these elements is that they are no longer constrained to fill the full length or width of the ASAP landscape. Try clicking and dragging the title bar of the ASAP Workspace window to the right. Once you get it completely into the **User Task Space**, its outline changes, indicating that it is undocked. It can now be placed anywhere in the available **User Task Space** (which just became larger), and resized to suit your needs. The same method works for the Command Output window.

Even though the Command Output and ASAP Workspace windows may have been undocked, both still have the property that no other window can be placed on top of them. They always appear "in front". You can change this behavior for the Command Output window by right-clicking the background, and deselecting the **Docking View** option at the bottom. This large window can now be shrunk considerably, and has no special status. The **Docking View** option is not available with the ASAP Workspace window. It will always remain in front of any other window in the user task space. If it cannot be shrunk or moved sufficiently far from your other work, you will have to temporarily turn if off with the **Toggle Workspace window** button in the main Toolbar.

You can undock the other elements by clicking and dragging the "gripper bars" (Figure 12.6). Once free from the sides of the ASAP window, they can be placed anywhere, or re-docked to some other window edge. Figure 12.6 also shows a sample of an alternative landscape configuration, along with the steps necessary to get there.

> ***Note:*** You can return to the default settings at any time using the **File> Preferences** dialog box. On the General tab, check the box marked **Reset window environment to ASAP defaults**. Exit and restart ASAP to return to the default configuration.

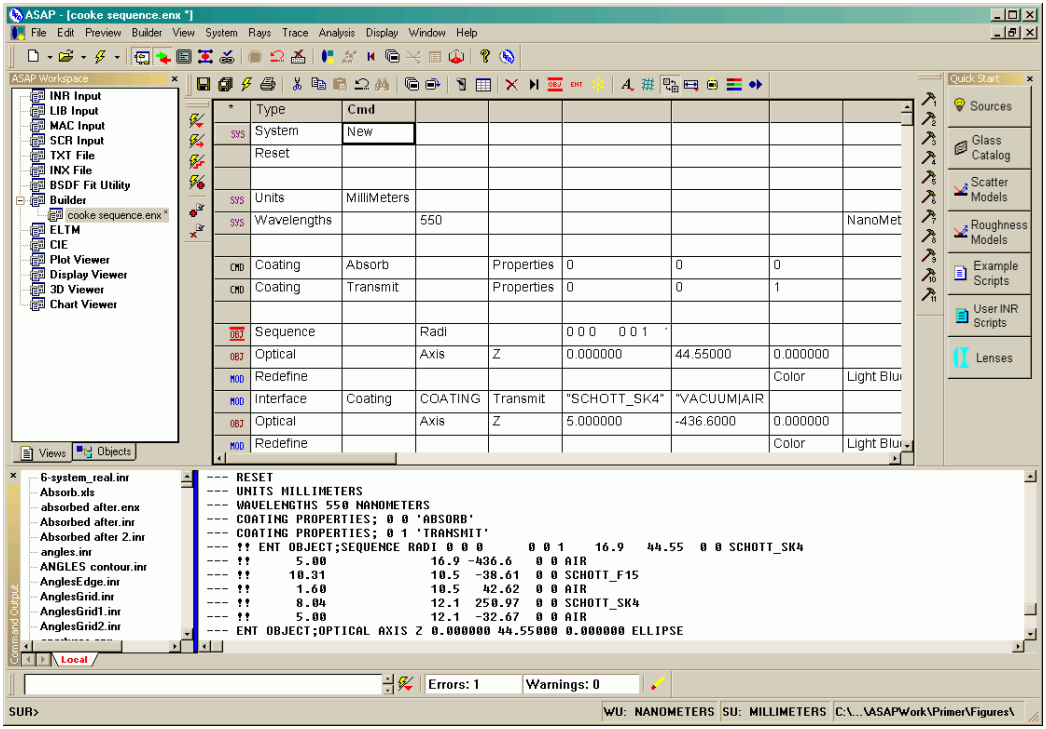


Figure 12.6 An example of an ASAP landscape after it was customized by the user.

Remembering Other Window Positions

We have not yet mentioned document windows, like the Plot Viewer, Chart Viewer, 3D Viewer, and the Builder windows. Their default positions are within the **User Task Space**, leaving space for additional windows of the same type to cascade down and to the right. The windows are generally given an aspect ratio suited to the shape of your **User Task Space**. You can change the default size and aspect ratio of these windows (and many others). First, position and size the window. Right-click in the title bar of the window, and select **Set as default position**. All future windows of this type will be positioned in this place. While future windows will not cascade, you can select the desired window using the **Views** tab in the ASAP Workspace window.

User-Definable Toolbar

Another way to customize ASAP is by using the Custom Toolbar. In the last chapter, you learned that the first step when beginning analysis is to locate the rays. We did this using **Rays> Locate Rays** on the menu. While many fundamental ASAP tasks have a button on the **ASAP Toolbar**, this important

function does not. You can, however, assign this task to one of the 20 available user-defined buttons. You need to know only the ASAP command (or commands) that performs this task. Although this feature may be of more interest as you learn more about the command language, it is worth a quick look now.

From the **File> Preferences** dialog box, select the **Custom Toolbar Editor** tab. The command executed by ASAP when you use **Rays> Locate Rays** is `STATS`. Type this command directly in one of the areas labeled **Tooltip Text**.

> ***Note:*** If you cannot remember this, recall that the ASAP kernel echoes all commands into the Command Output window as it executes them. Menu items with options have the **Script** button to show you the commands, even before they are executed.

Summary

In this chapter, we have discussed the following ASAP landscape areas:

- Organizing your work into projects from the ASAP Workspace window.
- Setting preferences for performing your work with the **User Interface Preferences** dialog box.
- Customizing the landscape by docking and undocking toolbars and repositioning document windows.
- Creating or editing buttons on the **Custom Toolbar Editor**.

CHAPTER

13

EDGES

In this chapter, you will learn the basics of EDGE-based geometry in ASAP. We will show how to use high-level edge definitions to make basic curves, and then how to turn these simple curves into ASAP objects. We will also look at the flexible POINTS command, which is the primitive definition behind all EDGE-based geometry used by ASAP.

You learned from Chapter 4 that ASAP works with surfaces or "shells" rather than solids when modeling geometry. A solid object like a lens was built up from three independent objects: a spherical surface at the front and back, and a tube representing the outer edge. Up to now, all the models we have made with ASAP were constructed with SURFACE-based geometry, grouped together in the Builder under **System> Geometry> Surfaces**.

> ***Note:*** We must be careful with our terminology because we are frequently tempted to use the word "surface" to describe one of these shells. This term, however, has a different meaning from SURFACE, which is a specific ASAP command. A SURFACE-based object is just one of three ways to make surfaces in ASAP. The program also provides us with EDGE- and LENS-based object definitions. EDGE-based objects are the topic of this chapter, and LENS-based objects are covered in Chapter 14, "Lens Entities". To avoid confusion, we will use the term "object" rather than "surface" to describe these generic shells.

SURFACE definitions like the TUBE, PLANE, and SPHERICAL are expressed within ASAP as implicit polynomials. This mathematical representation is ideal for efficient ray tracing. In most cases, there is an analytical solution for the intersection point of a ray and such a mathematical surface. Unfortunately, it is difficult or impossible to specify all geometry as an explicit polynomial. In general, many objects that have sharp corners or complex forms are better

defined in terms of parametric equations of the type used in computer-aided design (CAD) and other graphics-based applications.

Edges (also called “curves” in ASAP) were originally added to the program to enable importing geometry from CAD programs. ASAP also makes these same tools available to create **EDGE**-based objects directly, using the Builder or the scripting language. No CAD program is necessary in this case, and this class of objects is available whether you do or do not own the ASAP/CAD module. By supplementing **SURFACE**-based geometry with these tools, we can model virtually any object in ASAP. Edges are especially useful for modeling the mechanical structures in your system. **EDGE**-based objects are also ideal for exporting your designs from ASAP to CAD environments.

What is an Edge?

ASAP edges are a set of connected points.

ASAP edges are a set of connected points. The connections may be straight lines (Figure 13.1a), or smooth curves (Figure 13.1b on page 235). They are based on Bezier polynomials, which are parameterized functions.

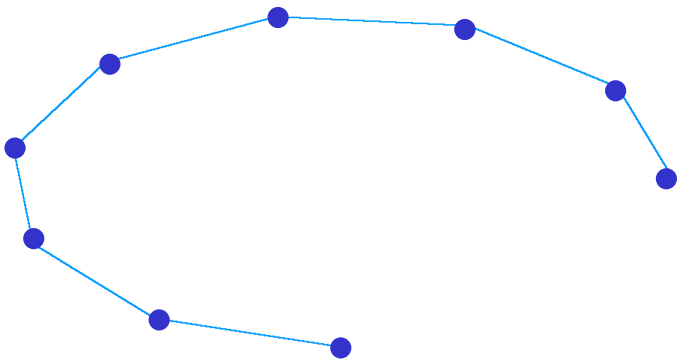


Figure 13.1a ASAP **EDGES** are connected points. In this example, the points are connected with straight lines. All edges are expressed within ASAP as Bezier polynomials.

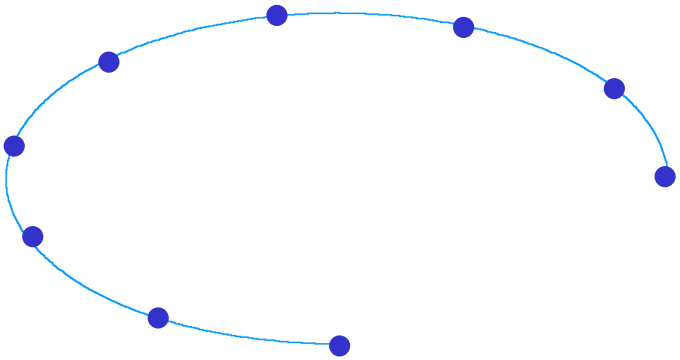


Figure 13.1b In this example, the points are connected with smooth curves.

Fortunately, you do not need to understand the details of Bezier mathematics to build ASAP edges. ASAP takes care of this for us. As a result, edges can be easily defined using simple, descriptive parameters much like the **SURFACE**-based objects we have been using until now.

> ***Note:*** If you are interested in a deeper understanding of ASAP edges and the mathematics of Bezier polynomials, this is the subject of the ASAP Technical Guide, *Edges*.

In most cases, we turn to edges only when there is no **SURFACE***-based option for building the structure we need.*

You may find that **EDGE**-based objects are somewhat slower to ray trace than **SURFACE**-based geometry when the same object can be constructed using either method. This result is due to their parametric mathematics. Since it is no longer possible to find a ray intersection analytically, ASAP must resort to algorithms that converge on a solution iteratively. Edges were implemented in ASAP in the most efficient way possible, so this penalty is often minor. Besides, we often have no choice. In most cases, we turn to edges only when there is no **SURFACE**-based option for building the structure we need.

This iterative ray-tracing method also has another undesirable consequence: sometimes rays "leak" from an **EDGE**-based object. When a ray is directed exactly toward a seam in the object (at one of the points defining the edge, for example), there is a small probability that the ray will fail to see the edge and trace on to the next object, which sometimes occurs in very large ray traces. These rays can usually be ignored.

Making Objects from Edges

An edge is just an arbitrary curve drawn in two or three dimensions.

An edge, as you have seen, is just an arbitrary curve drawn in two or three dimensions. As such, it cannot interact with rays. We need to use one of three methods to build these curves into real objects:

a. Bounding a plane with a closed curve,

b. Sweeping an edge in a direction, toward a point, or around an axis, or

c. Extruding two edges together.

The meaning of each of these options is illustrated in Figure 13.2.

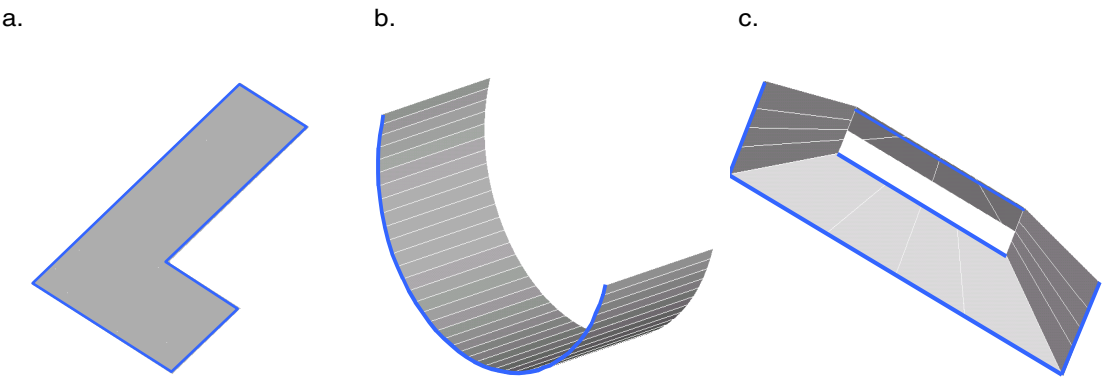


Figure 13.2 The blue lines and curves shown above are edges. By themselves, they do not interact with rays. These edges are used only as building blocks for making ASAP objects, and illustrate three methods for changing an edge into an object. In (a), an ell-shaped plane is defined by a planar edge that closes on itself. In (b), an arc is swept to form a trough. In (c), two rectangular edges are connected to form an extruded object.

Bounding a plane by a closed edge is perhaps the simplest of these. The ASAP Builder always attempts to use this method by default, automatically trying to convert any edge you define into such an object.

> ***Note:*** When this bounding succeeds, ASAP actually converts the new object into a `SURFACE`-based `PLANE`. The edge is then used only to define the `BOUNDS` of the plane. ASAP automatically performs all these steps for us. There is no ray-trace penalty whatsoever for defining a bounded `PLANE` in this way.

You can also sweep edges to define objects. The ASAP Builder allows us to sweep the curve along a direction, towards a point, or around an axis. With scripts, it is even possible to sweep one curve along another, a method known as

"lofting". Though the topic of lofting is beyond the scope of the *Primer*, it is discussed in the ASAP Technical Guide, *Edges*.

The last option we will discuss for turning edges into objects is extruding, where one edge is connected, point by point, to another. The object becomes the mesh connecting the two edges. While this method may sound complicated, it is easy and powerful. The process becomes clearer after a few examples.

Ellipse Example

As a first example of an **EDGE**-base object, we will build an **ELLIPSE** edge using the Builder. You can access this, or any of the edge definitions available in the Builder, by double-clicking a cell in the **Type** column and selecting **Geometry> Edges> Ellipse** from the menu. Try this, modifying the default ellipse to create a circular edge with an **X** and **Y Semiwidth** of **1** located at a position along the axis at **Z = 1**, as shown in Figure 13-3.

*	Type	Name	Axis	Location	Semiwidth	Semiwidth	Points	StartAngle	StopAngle
OBJ	Ellipse	ELLIPSE	Z	1	1	1	16	0	360

Figure 13.3 Example of an **ELLIPSE** edge in the **Builder**

Recall that all edges are connected points in space. The number of points that make up this ellipse is set to **16**, the default. This value is fine for our purposes. We also have the ability to define only part of the ellipse by specifying a **Start Angle** and **Stop Angle**. We will also leave these at their default values, giving us a complete circle. When you preview the object defined above, you should see the result shown in Figure 13.4 (left).

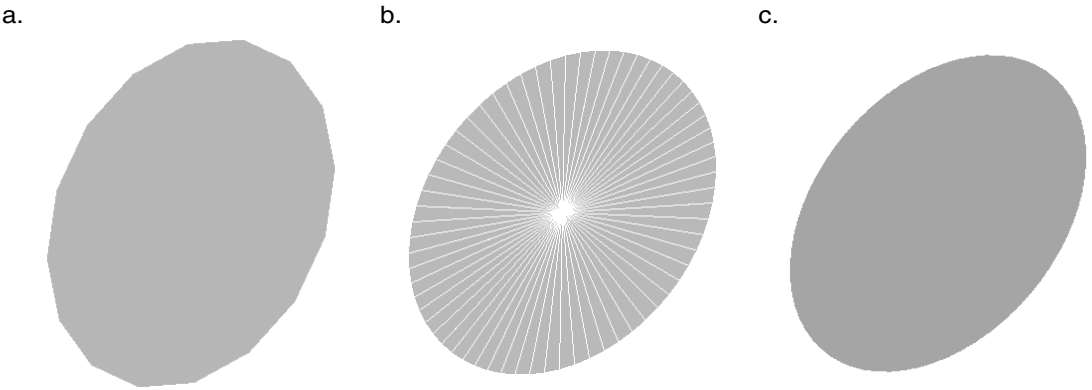


Figure 13.4 An **ELLIPSE** defined with 16 points (a) is actually a 16-sided polygon, not a true ellipse. The result can be improved by increasing the number of points in the edge. The center view (b) shows considerable improvement with 64 points. View (c) shows a mathematically perfect circle, obtained by using the edge modifier, **SMOOTH**.

In Figure 13.4, highlighted edges were use to produce the center graphic, showing facets radiating to the 64 points defining the ellipse. While this result is a much closer approximation to a circle, the object is still a polygon. The best result is obtained by smoothing the curve with the **SMOOTH** modifier (c), and producing (in this case) a mathematically perfect circle.

You may need to rotate the view if the object is initially drawn edge-on. As noted in the previous section, ASAP has automatically assumed we were trying to produce a planar object bounded by this edge. Note, however, that the resulting object is actually a 16-sided polygon, not a true circle. By default the connecting points are straight lines, as the figure shows.

Next, try increasing the number of points in the **ELLIPSE** definition to 64, and preview it again. The result of this change is shown in the center panel of Figure 13.4. While much improved, the resulting object is still fundamentally a polygon, not an ellipse.

In some cases, you may find it acceptable to approximate a continuous curve as a series of straight-line segments. It is possible to do much better, however. In the Builder line, directly under the **Ellipse** definition, try adding the command **Geometry> Edge Modifiers> Smooth**.

Using the default order value of two, this special edge modifier performs a quadratic smoothing of the edge in parameter space. In the case of a circle (or an ellipse), the resulting edge is now a perfect circle.

See Chapter 13 Appendix, "Script 13-1" on page 263.

The real power of edges comes from sweeping or extruding simple edges like this ellipse, or from bounding planes with more complex edge shapes.

We have really gained nothing by defining a circular plane in this way. It would have been easier to use the **Geometry> Surfaces> Plane** command that we introduced in Chapter 6, "Cassegrain Telescope Model", choosing the **Ellipse** option for the aperture shape. The real power of edges comes from sweeping or extruding simple edges like this ellipse, or from bounding planes with more complex edge shapes.

Extruding Edges

Next, we will extrude two edges together to form a single object. We begin by introducing another useful edge type, the **RECTANGLE**. Try making a rectangular edge in the same Builder file used for the **ELLIPSE** above. The command can be found in the Builder menu at **Geometry> Edges> Rectangle**. Define a **Z** rectangle at the origin (**Z = 0**), and set both dimensions (**X** and **Y Semiwidth)** to 2 units. Leave the number of points set to 16. It would take only four points to specify a perfect rectangle, of course, but our goal is to extrude two edges together. The requirements for extruding two edges together are that both edges must be similar and have exactly the same number of points. The **ELLIPSE** and the **RECTANGLE** edges meet these requirements, as long as we take care to set the

number of points to the same value in both cases. The resulting lines in the Builder should appear as shown Figure 13.5.

*	Type	Name	Axis	Location	Semiwidth	Semiwidth	Points	StartAngle	StopAngle
OBJ	Ellipse	ELLIPSE	Z	1	1	1	16	0	360
OBJ	Rectangle	RECTANGLE	Z	0	2	2	16	0	360

Figure 13.5 Addition of `RECTANGLE` edge in the **Builder**

When you preview all objects now, you should see the result shown in Figure 13.6a. Once again, this is not the best way to make a simple, rectangular plane. Our ultimate goal now, however, is to extrude these two edges together to make a very different object.

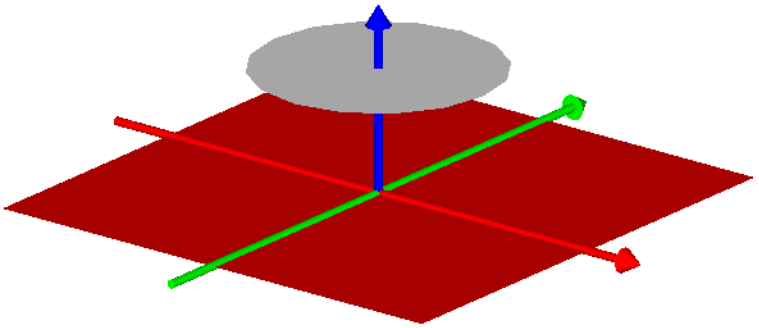


Figure 13.6a In this preview, planes bounded by an ellipse and a rectangle were created at Z=1 and Z=0, respectively. In both cases, ASAP automatically presumed that we wanted the planar edges to define the boundaries of a plane.

See Chapter 13 Appendix, "Script 13-2" on page 263.

When we extrude edges, we generally need to override the default behavior of the Builder, and prevent the edges themselves from automatically becoming objects. We do this by changing the status of the Builder lines from Object (**OBJ**) to Entity (**ENT**) status. We can change the status of any object definition in the Builder, whether it is `SURFACE`-based, `EDGE`-based, or `LENS`-based. We are doing this to define these pieces of geometry as intermediate building blocks, but there are a variety of other reasons why we might want to reduce the status of a geometrical definition from object to entity status. These are discussed in the sidebar, "Objects versus Entities" on page 261.

ENT

First, select the **Ellipse** definition by clicking any cell in that Builder line. Next, either right-click and select **Entity Status** from the pop-up menu, or click the **Entity Status** button on the Builder toolbar.

Do the same thing for the **Rectangle**, and preview the system again. The result should be the same as that shown in Figure 13.6b. Only the edges or "outlines" of the two entities exist now.

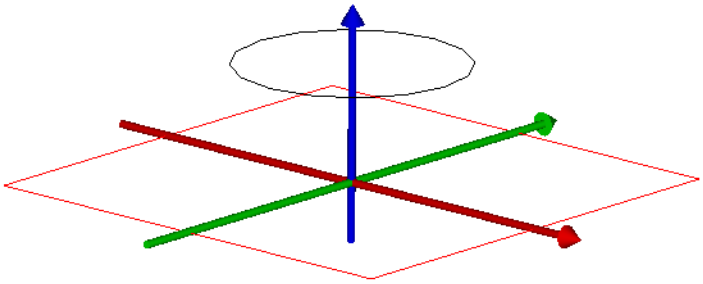


Figure 13.6b After changing the **ELLIPSE** and the **RECTANGLE** definitions to Entity status, only the edges remain. The edges do not interact with rays.

The last step in the extrusion process involves the **OBJECT** command. This is how we tell ASAP which entities to use to construct the object. This command is found in the Builder pop-up menu. Select **Object Control > Object**. In the cell labeled **Entity Number**, enter **.1 .2** (a space must be between the numbers).

*	Type	Name	Entity						
OBJ	Ellipse	ELLIPSE	Z	1	1	1	16	0	360
OBJ	Rectangle	RECTANGLE	Z	0	2	2	16	0	360
CMD	Object	EXTRUDED_OBJECT	.1 .2						

Figure 13.6c Last step in the extrusion process, involving the **OBJECT** command

We are again using the "dot" notation introduced in Chapter 4, which allows us to count backward (from the current line) to select the desired entities. In this example, we have chosen the most recently defined entity (**.1**) and the second most recent (**.2**).

Note: ASAP is counting *all* geometry definitions. It does not matter whether the definition has object or entity status. This is true for all references to geometry in ASAP, whether you are creating objects, bounding objects, or performing any of the other operations that require a reference to a previously defined geometrical definition. ASAP maintains two lists as part of its geometry database. The larger is a complete list of all geometry definitions or "entities". They reside there in the order in which they were defined. The second is the object list. It points to geometry in the entity list, using either relative or absolute referencing. Hence, all objects are entities also, and must be counted, no matter how they are used.

See Chapter 13 Appendix, "Script 13-3" on page 264.

Now, when you preview the Builder file, you should see the new object as it appears in Figure 13.6d on page 241.

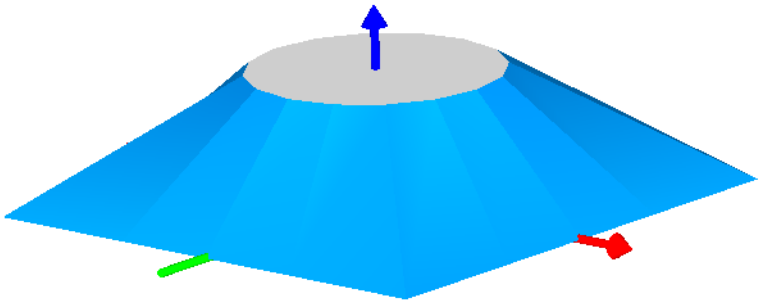


Figure 13.6d When the two edges are extruded together, a single object results. The object was formed by connecting the points of one edge to the other. This method works only for similar edges with exactly the same number of points.

As you can see, ASAP has literally connected the points to form flat-polygon sides. You can cause this to become a smooth blend between the rectangle and the ellipse by smoothing the entities (as described above) but be sure to smooth both entities. This guarantees that they remain “similar”, which is a requirement when extruding two entities in this way. The results are shown in Figure 13.7a and in the Builder file in Figure 13.7b.

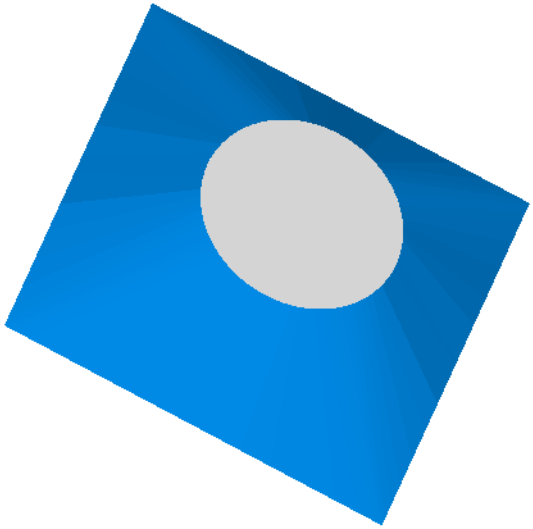


Figure 13.7a This is the same extruded object as seen in the previous figure, but now the entities have been smoothed before extruding. The resulting object is a good blend from the rectangular base, to the circular hole.

*	Type	Inter	Intra						
OBJ	Ellipse	ELLIPSE	Z	1	1	1	16	0	360
MOD	Smooth								
OBJ	Rectangle	RECTANGLE	Z	0	2	2	16	0	360
MOD	Smooth								
CMD	Object	EXTRUDED_OBJECT	.1 .2						
MOD	Interface	Coating	COATING	Reflect	Air	Air			
MOD	Facets	11	1						

Figure 13.7b As the **Builder** file shows, we have increased the number of facets applied to this one object by using **FACETS** as an Object Modifier. This change improves its appearance in the **3D Viewer**, more closely approximating the perfect mathematical object that the rays will see.

See Chapter 13 Appendix, "Script 13-4" on page 264.

As one last example of extruding, try placing both the ellipse and the rectangle at the same Z position (Z = 0, for example). This produces a circular hole in a square plane, as shown in Figure 13.8.

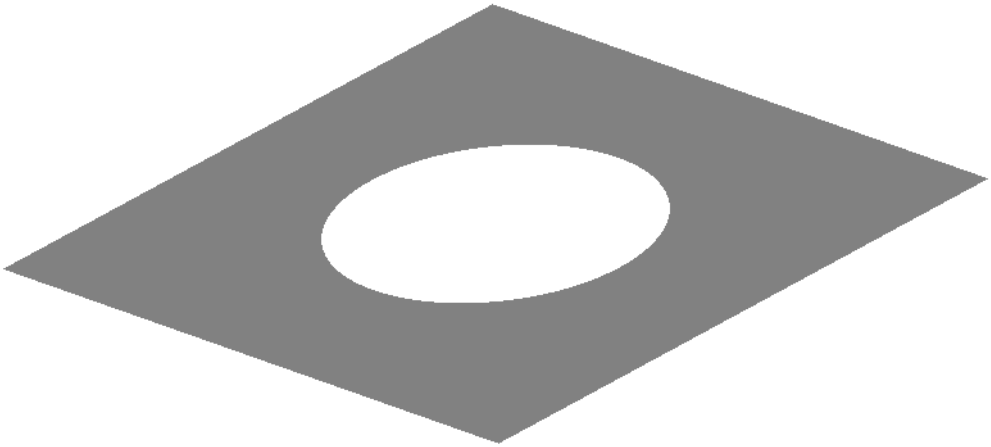


Figure 13.8 The extrusion process can also be applied to two edges lying in the same plane. This method works well for making aperture stops. Smoothing was still used on both entities to give a perfect circular aperture in the center.

The **SURFACE**-based **PLANE** command discussed in Chapter 6 allows us to have a square hole in a square plane or a circular hole in a circular plane, if we place a value in the **Obscuration Ratio** cell. There is, however, no option for creating a circular hole in a square plane, or vice versa. By extruding edges in the way described above, we can create a hole of virtually any shape without resorting to bounding.

See Chapter 13 Appendix, "Script 13-5" on page 264.

Sweeping a Line

Sweeping an edge is the third method of making curves into objects. This time we begin with the simplest of all edge entities, a straight line defined by two points, and sweep this around the Z axis to make a cup-shaped reflector (Figure 13.9).

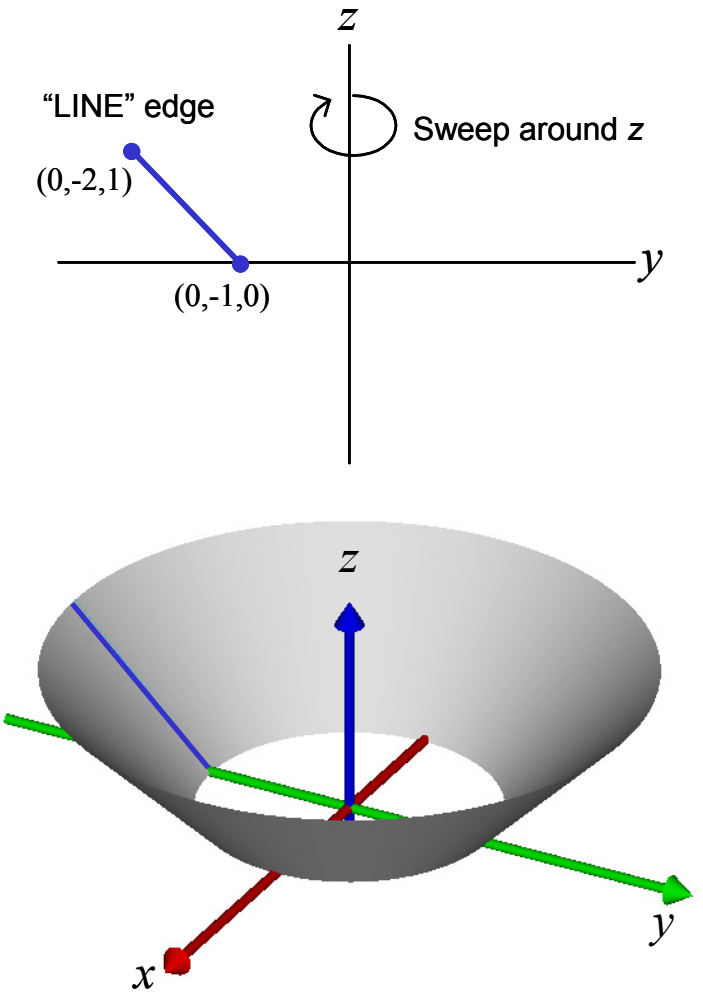


Figure 13.9 A line can be swept around an axis to form a continuous surface. On the top, the **EDGE**-based line entity is defined in terms of the Cartesian coordinates of the end points. On the bottom, the resulting swept edge forms the object.

We have specified the line using the global coordinates of the starting and ending points. We have also changed the definition from **OBJ** to an **ENT** status,

as before. Next, we will add a line directly under this definition using **Geometry> Edge Modifiers> Sweep** and select **Axis** from the drop-down list in the **Option** column. Like the `SMOOTH` command, this is a modifier that works only on edges. Because of this, it appears as part of the **Edge Modifiers** submenu. We specify the angle through which we want to sweep the line, and the axis of the sweep. We can complete the definition with the `OBJECT` command, this time specifying only one entity as shown here:

*	Type	Option	Option	Angle	Option	Axis	Point X	Point Y	Point Z
OBJ	Line		-2	1	0	-1	0	0	
MOD	Sweep	Dir	AXIS	360	Axis	X	0	0	
CMD	Object	CUP	1						

Figure 13.10 Sweeping an edge with a straight line

See Chapter 13 Appendix, "Script 13-6" on page 265.

The actual command being executed is `SWEEP AXIS`. ASAP also allows us to sweep towards a point `SWEEP POS`, or along a direction `SWEEP DIR`.

Facets

If you attempted to create the "cup" object described above, your preview might not look as smooth as the rendering shown on the bottom of Figure 13.9 on page 243. Your result probably looked like the view shown in Figure 13.11a.

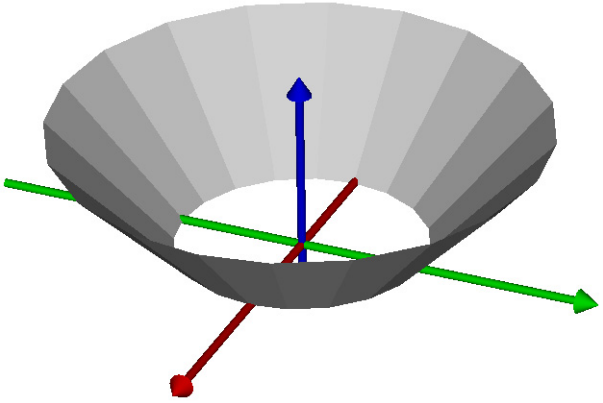


Figure 13.11a Using the default facet setting for the Preview function in the Builder, the swept-line version of the cup appears as shown here. The flat facets are an artifact of the rendering algorithm used by the 3D Viewer.

Why isn't it smooth and continuous? We are not connecting the points in two polygon edges; sweeping a line should be mathematically perfect. It is. The view shown in Figure 13.11a is an artifact of the rendering algorithm in the 3D Viewer. Recall from our first discussion of the 3D Viewer in Chapter 4,

"Building and Previewing Geometry" that the Viewer approximates all surfaces with a series of flat polygons. This approximation allows the program to rapidly manipulate the view—rotating, shifting, and zooming—with minimum computational delay.

In Chapter 5, "Running and Verifying Geometry", when you used **System> Plot Facets** on the ASAP menu bar to make your own 2D and 3D faceted graphics, you were able to control the number of facets to improve the view. We also explained that the `PLOT FACETS` command is fundamentally equivalent to what ASAP does with the **Preview** function of the Builder. Is there a way to improve the preview as well? Yes, there are actually two different ways. First, whenever the Builder window has focus, you will see the Builder menu on the ASAP menu bar. Try selecting **Builder> Facets**. ASAP presents the **Facets** dialog box for changing the default faceting level for previewing, which affects the faceting for all objects in your Builder file.

As an alternative to changing the default facet level for previewing, you can change the number of facets on an object-by-object basis. This flexibility is often useful when one object needs extra faceting to look good, while others do well at the default values (**5** and **5**). Remember that increasing the number of facets slows down the **3D Viewer**. The number of facets can be entered directly into a line in the Builder file, serving as an object modifier. It is on the pop-up menu as a general modifier (**Object Controls> Object Modifiers> Facets**), and can be applied to any object definition. The actual command being run is `FACETS`. We have used this option to produce the view shown in Figure 13.11b, setting the **Inter** facet setting to **1**, and **Intra** to **21**. Note that it is placed under the object definition.

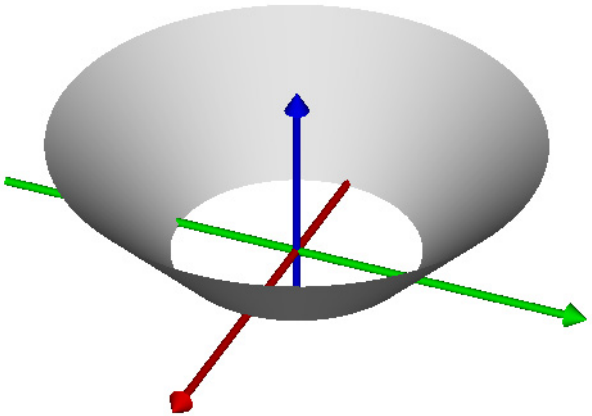


Figure 13.11b Using at least 11 facets (rather than the default of 5) in the **Intra** direction shows the smooth, continuous nature of the ASAP object.

New ASAP users often become confused over this issue of faceting, smoothing, and changing the number of points in an edge definition. They raise questions

such as, “When does smoothing help?”, “What really happens when the number of points in an elliptical edge is increased?”, “Does faceting change what the rays see?”, and “How can we see what the object looks like to rays?”

Here is a quick summary to address these questions:

- The number of points in an edge like the **ELLIPSE** command fundamentally changes the object. When no smoothing is performed, the ellipse is made up of points connected by straight lines.
- Smoothing also changes the mathematical nature of the object. The rays see a different structure if the edges that define the object are smoothed.
- The number of facets used to render the object *has no impact whatsoever* on the actual object in the system database, and the way rays interact with it. The number of facets affects only the graphics.

Here is a simple strategy to keep in mind:

If you are concerned about what you see in the **3D Viewer**, increase the number of facets used for any questionable objects. If the view improves, and continues to improve with even more facets, the problem is just the graphical approximations used by the Viewer. If no amount of faceting changes the view at all, what you are seeing is probably what the object really looks like (Figure 13.11c).

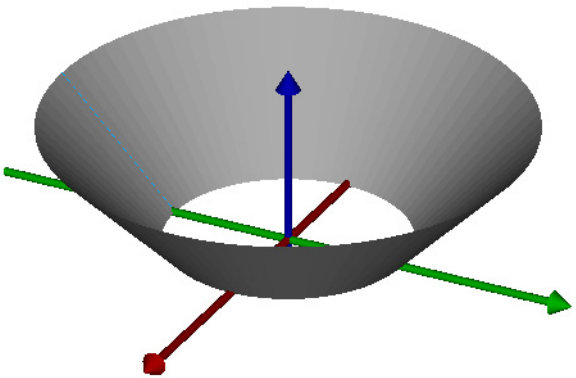
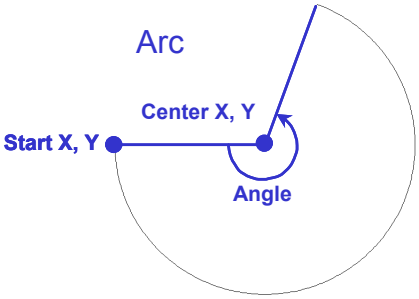


Figure 13.11c If the cup had been defined by extruding two elliptical edges (with no smoothing), no amount of faceting would change the appearance of the object in the **3D Viewer**. In this case, the segments between points really are flat, and this is what the rays are seeing.

Note: As for the question of what the object really looks like to rays, ASAP includes two commands that use ray-trace methods to produce graphics. One method is **PROFILES**, which we already discussed in Chapter 5. ASAP locates the surface by computing the equivalent of ray intersections over a sampling grid. The pixel value controls the number of samples. Connecting these samples produces the resulting profile. Another useful command is **RENDER**, which we have not discussed before. It is available on the **System** menu. It draws a shaded, three-dimensional (though fixed) representation of the objects in your system, again using ray-trace methods to locate and shade the objects. See the online help for details.

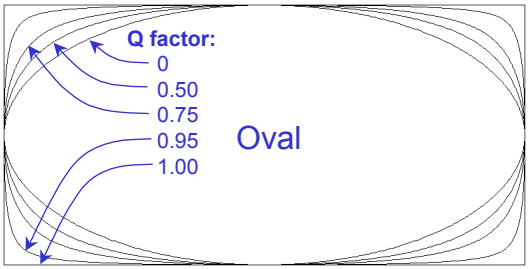
Other Edges

In addition to the **RECTANGLE**, **ELLIPSE**, and **LINE** commands that you have already seen, the Builder allows you to define five other high-level edges. Five of these are the **ARC**, **OVAL**, **RACETRACK**, **ROUNDED** and **SAWTOOTH** edges. An example of each, along with a brief description, appears in Figure 13.11d to Figure 13.11h. Five more edges are also available: **CHARACTER**, **CONIC**, **HELIX**, **BEZIER**, and **SPLINE**.



*	Type	Name	Option	Axis	Location	Radius	List
ENT	Arc		Centered	Z	0	0	

Figure 13.11d The **ARC** edge creates a circular arc in a plane. Specify the plane's axis, coordinates of the starting point, center point, and the angle subtended by the arc.



*	Type	Name	Axis	Location	Semiwidth X	Semiwidth Y	QFactor	Points	StartAngle	StopAngle
OBJ	Oval		Z	0	1	2	.0	16	0	360
MOD	Smooth	2								

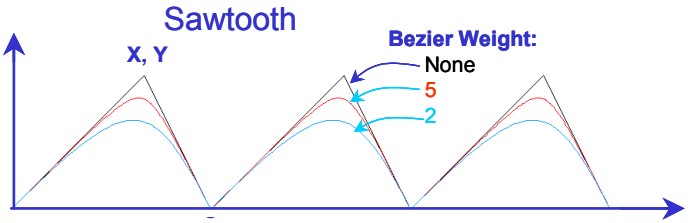
Figure 13.11e The **OVAL** command creates a polygonal edge that can continuously vary between an ellipse (Q=0) and a rectangle (Q=1). This example uses only 16 points, but is smoothed.

*	Type	Name	Axis	Location	Semiwidth X	Semiwidth Y	Corner Rad X	Corner Rad Y	Option
OBJ	Racetrack		Z	0	2	1	0.5	0.5	

Figure 13.11f The **RACETRACK** command creates a rectangle with smooth, rounded corners. Specify the dimensions of the rectangle and the distance from the corner (in both directions) where you want the curve to begin. No smoothing is required because the corners are already continuous curves.

*	Type	Name	Axis	Location	Semiwidth X	Semiwidth Y	Radius	Points	StartAngle	StopAngle
OBJ	Rounded		Z	0	2	1	0.5	64	0	360
MOD	Smooth									

Figure 13.11g The **ROUNDED** command is similar to **RACETRACK**, but the former consists of only line segments, which is similar to other edges, like **ELLIPSE** and **RECTANGLE**.



*	Type	Name	Axis	Location	X	Y	X'	Y'	Teeth	Bezier Weight
ENT	Sawtooth		Z	0.0	1.0	1.0	1.5	0.0	3	2

Figure 13.11h The **SAWTOOTH** command allows you to specify one "tooth" of a sawtooth that starts at the origin. Specify the other two points and the number of teeth desired. The teeth come to a sharp point by default, or can be "rounded" using the **Bezier Weight** parameter.

Points Connected by Straight Lines

All edges are nothing more than a series of connected points.

The **POINTS** command is the fundamental or "primitive" tool behind all edge entities in ASAP. All edges are nothing more than a series of connected points. The **POINTS** command allows you to specify the individual points, and control the way in which each point is connected to the next. While some uses of this command are beyond the scope of this *Primer*, **POINTS** has many relatively simple uses that are worth a look.

Consider the simple lens cell shown in profile in Figure 13.11i to Figure 13.11k. You have learned enough from the early chapters of this *Primer* to model the structure using **SURFACE**-based entities.

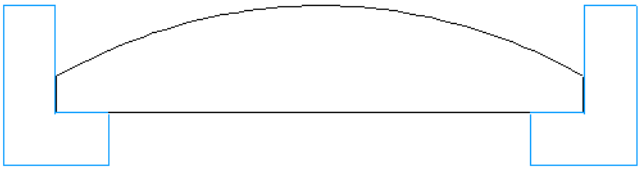


Figure 13.11i A model of a lens cell designed to mount a plano-convex lens (**Plot viewer**)

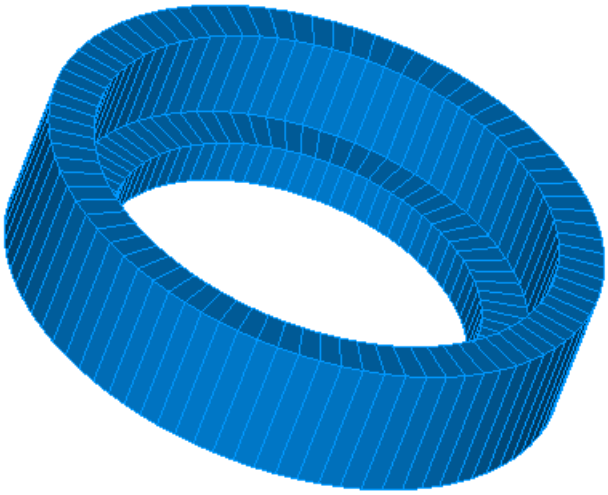


Figure 13.11jA model of a lens cell designed to mount a plano-convex lens (**3D Viewer**)

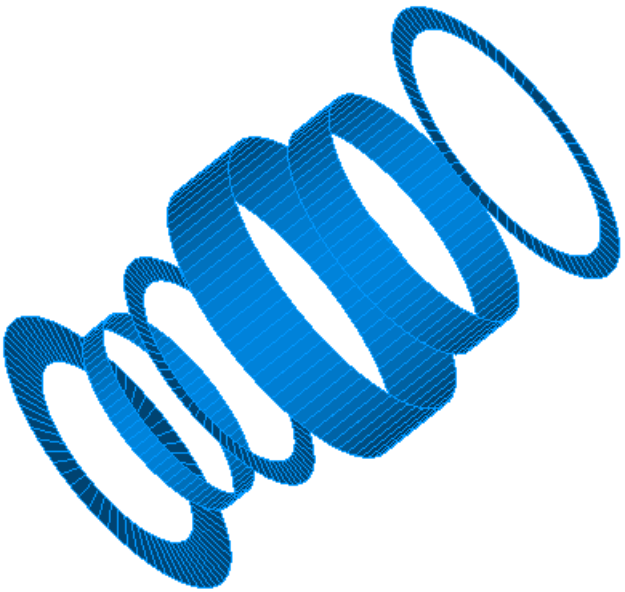


Figure 13.11k While it would take six **SURFACE**-based objects to model this simple lens cell, the same geometry can be produced by sweeping one edge (defined using **POINTS**) around an axis.

With the methods we have taught, however, the model would require six ASAP objects: three **TUBE** surfaces, and three **PLANE** surfaces with central "obscurations" (holes), also shown in Figure 13.11k. Now we will model this structure using just one swept **EDGE**-based entity defined with the **POINTS** command.

Figure 13.12 shows a cross section of the lens cell. Our plan is to calculate the global coordinates of the points at the corners of the ell-shaped cross section, use the **POINTS** command to define this edge, and sweep it about the *z* axis to form the object.

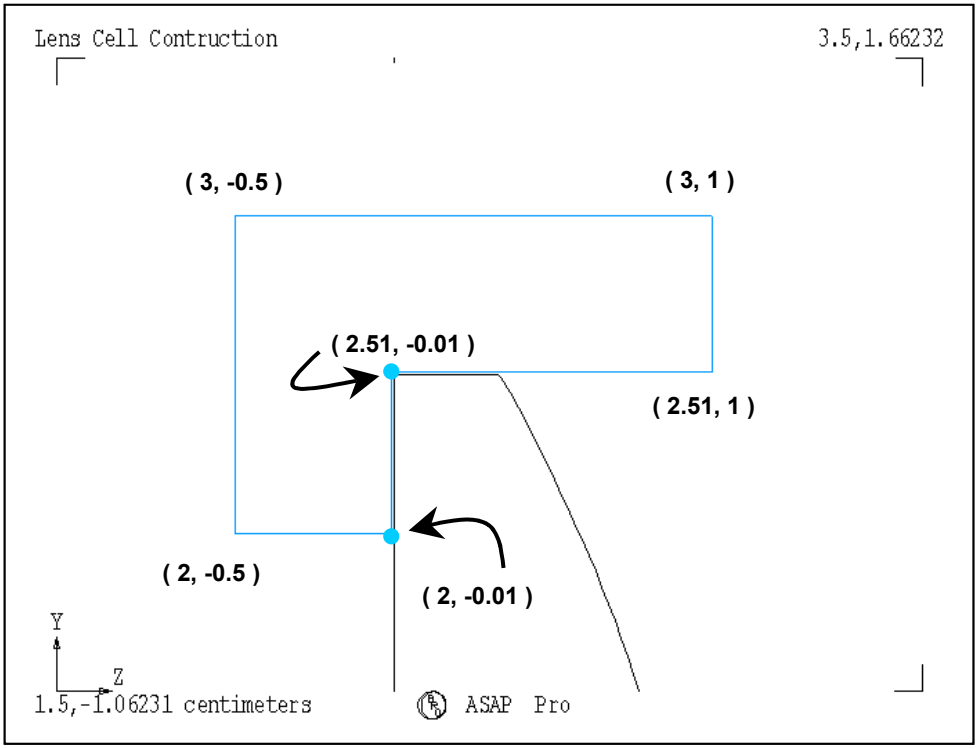


Figure 13.12 The coordinates of the six points are needed to define the ell-shaped edge that will be swept to form the lens cell. Note that a "clearance" of 0.01 cm was left around the 5 cm-diameter lens, so that two objects never occupy exactly the same position. Without this, the model would be both non-physical and problematic for the ray-tracing algorithm.

ASAP has two versions of the `POINTS` command, one for a curve defined in one of the global coordinate planes (**Geometry> Edges> Points> 2D**), and another more general form in three dimensions (...**Edges> Points> 3D)**. The two-dimensional form works well for our case.

We specify the **Axis** of the plane in which the points will be defined, and the **Location** along this axis. In this case, we have selected the *x* axis since the `POINTS` are defined in a *y*-*z* plane. The current parameters appear whenever the cursor hovers over this cell, as shown above. To enter or edit the parameters, you must double-click this cell to launch a special Command Editor window.

The first two values on each line are the coordinates of the point (*y* and *z* coordinates), since we are defining this edge in a plane perpendicular to the *x* axis. The third value is a connection factor. We use this to describe how we want to connect this point to the next. The connection factor on the last point on the list allows us to connect the last point back to the first, if that is desired. In our case, we have used a connection factor of **1** in all cases, which specifies a straight line.

Note: Some ASAP examples show commas between points, or after a set of three points in this command. The commas are not a requirement, but only a matter of style, and they are ignored. Multiple points can also be specified on the same line as long as the total length of the command line that ASAP generates does not exceed 344 characters.

We have immediately changed this Builder line to entity status, since this edge is only the building block for our actual object. If it had remained an object, we would have defined only an ell-shaped plane.

The rest of the commands necessary to the definition appear in Figure 13.13.

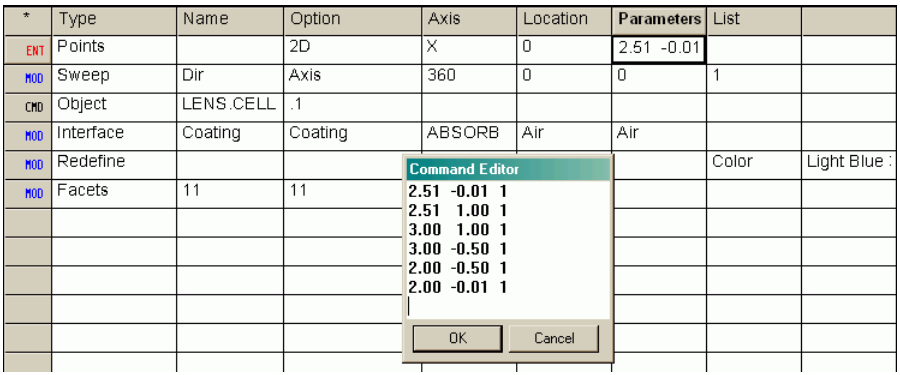


Figure 13.13 The `POINTS` command in the Builder allows us to define the plane in which the points will lie. By double-clicking the **Parameters** cell, we open a special window, the **Command Editor**, which is used to enter the actual points. In this case, we have defined an *x* plane, so the points are entered in the order of *y*, *z*, followed by the parameter. A parameter of **1** means that the points are connected by straight lines. The parameter of **1** on the last point connects the last point to the first with a straight line. The resulting edge is then swept about the Z axis to form the lens cell.

We used the `SWEEP` edge modifier in exactly the same way it was previously used, when we swept a line into the cup-shaped structure. We then used the `OBJECT` command to complete the definition, followed by the appropriate object modifiers, `INTERFACE`, `REDEFINE COLOR`, and `FACETS`.

See Chapter 13 Appendix, "Script 13-7" on page 265.

Connection Factors other than a Straight Line

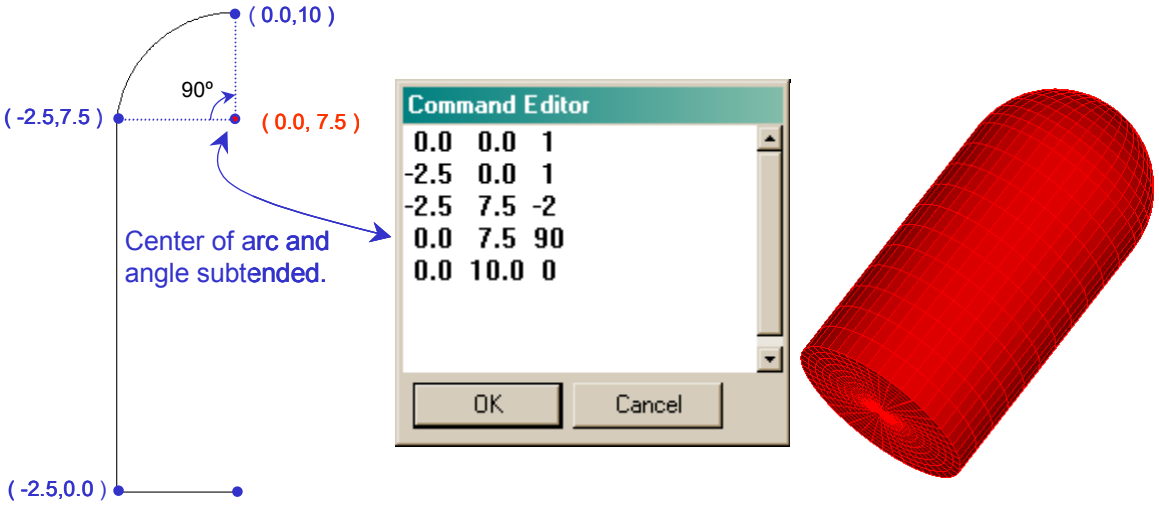
A straight line is not the only connection factor allowed by the `POINTS` command. Table 13.1, from on-line Help, summarizes other possible connection or *q* parameters.

Table 13.1 Connection Parameters

q Parameter	Edge Point Connection
q=0	Not connected to the next point (open)
q=1	Connected by a straight line to the next point
q=2	Connected by a quadratic rational Bezier (conic) curve to the point after next. The next point is the intermediate control vertex. The next q is the intermediate weighting factor and is always positive: 0=straight line <1 ellipse 1 parabola >1 hyperbola
q=-2	Connected by an elliptical arc to the point after next. The next point is the center of the parent ellipse. The next q is the angular subtense (in degrees) of the arc and must be less than 180 degrees.
q=n	Connected by an nth- (up to 10th) degree rational Bezier curve. The intermediate n-1 points are control vertices with given (positive) weight factors.

The **POINTS** command was built into ASAP primarily to facilitate the translating of IGES files into ASAP commands. When you import information from a CAD program, ASAP creates the **POINTS** commands for you, along with the necessary control points and weight factor described in the table above. However, many ASAP users continue to write their own **POINTS** commands, using the Builder or command scripts, because of its power and flexibility. For relatively simple situations like connecting two points with a circular arc or a simple conic, the procedures are not much more difficult than the examples you have already seen.

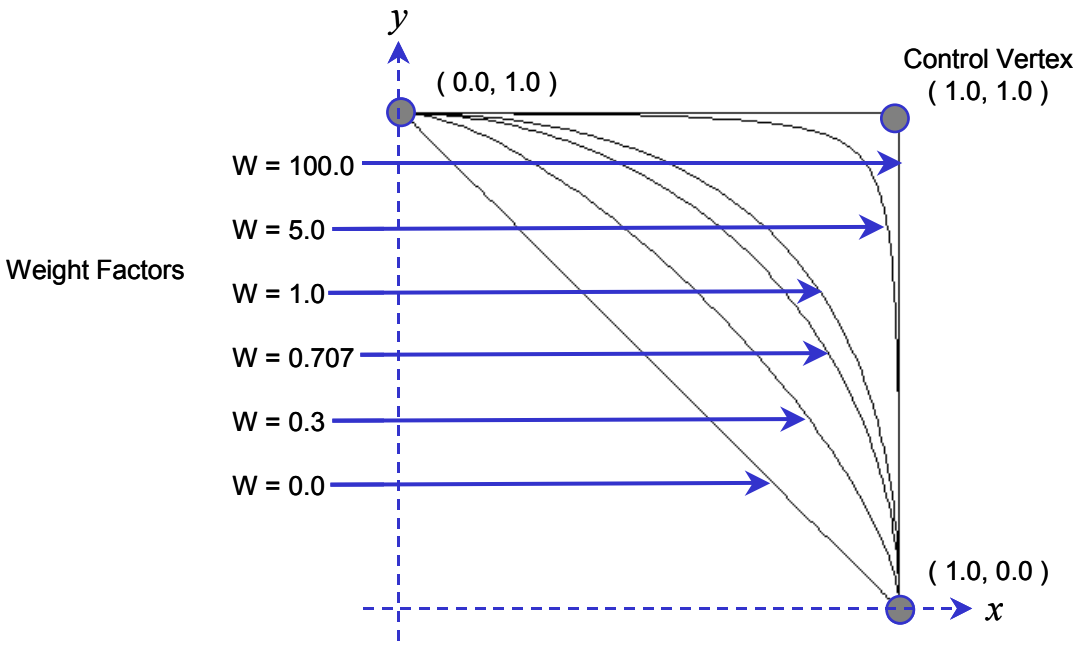
Figure 13.14 illustrates the use of the **q = -2** option to connect two points with an elliptical arc. In this example, we are able to create the body of a light-emitting diode with just one swept edge.



*	Type	Name	Option	Axis	Location	Parameter	List	
SYS	System	New						
	Reset							
CMD	Coating	TRANSMIT		Properties		0	1	
ENT	Points		2D	X	0	0.0 0.0		
MOD	Sweep	Dir	Axis	360	0	0	1	
CMD	Object	LED.BODY	.1					
MOD	Interface	Coating	Coating	TRANSMIT	Air	Air		
MOD	Facets	11	11					
MOD	Redefine						Color	Red 2

Figure 13.14 This example of the **POINTS** command uses a connection factor (**q** value) of **-2** to connect the third point to the fourth. The parameters required appear in the **Command Editor** window. In this case, the fourth point listed in the **Command Editor** window is the center of the circular (or elliptical) arc, and the third value on that line is the angle (in degrees) subtended by that arc. The result is a single ASAP object representing the body of a light- emitting diode. Note that this method should be used only when all the elements of the object (bottom, sides, and domed top) have exactly the same optical properties, since only one interface can be specified.

Figure 13.15 shows examples of control points and weights for the **q = 2** option. The weight factor is used as a free parameter to make a family of conic connections.



*	Type	Name	Option	Axis	Location	Parameters	List
OBJ	Points		2D	Z	0	0, 0 1.0 2 1.0 1.0 0707 1.0 0.0 0	

Figure 13.15 Connecting two points with a quadratic Bezier using the **POINTS** command. The two points being connected are listed first and third in the parameters list. We specify a connection factor (**q** value) of **2**. ASAP then expects the second point to be a control vertex. The third parameter on that line is a weight factor, which can be thought of roughly as how hard the curve is pulled toward the control vertex. The resulting curves are all conics. The ASAP Technical Guide, *Edges*, gives equations for calculating the position of the control vertex and weight factor for specific conic curves.

Importing and Exporting Edge-based Objects

We owe the existence of edge-based objects in ASAP to CAD programs. By far the most common use of edges in ASAP is still importing geometry from CAD programs into ASAP by way of the CAD translator. If you own ASAP CAD+, importing geometry is as simple as reading the IGES, GTX, CATPART, or CATPRODUCT file from within ASAP. The translator automatically generates all the commands necessary to model the objects, and places them in a single script file. You can run this file without detailed knowledge of the CAD file. You also have the option to add optical properties to the objects as part of the translation process. The procedure for doing this with IGES files is covered in the Technical Guide, *IGES Import*.

Another advantage to using edges in ASAP is that any edge can be easily exported from ASAP back to the CAD environment using **Export to CAD** on the **System** menu. The geometry information is written to a file in either IGES or DXF format. The commands being executed are `CAD IGES` or `CAD DXF`, respectively.

> ***Note:*** Many `SURFACE`-based objects that you construct within ASAP can also be exported in the same way as described above. There are limitations, however. Some objects, such as `OPTICAL` surfaces with aspheric coefficients, have to be approximated by cubic splines. There is no exact CAD standard for their general algebraic specifications.

Summary

In this chapter, we have discussed the following new commands:

ASAP Commands	Menu	Description
`EDGE; ELLIPSE`	Builder: **Geometry> Edges> Ellipse**	Creates an approximately elliptical edge by connecting points.
`EDGE; RECTANGLE`	Builder: **Edges> Rectangle**	Creates a rectangular edge.
`EDGE; ARC`	Builder: **Edges> Arc**	Creates a smooth arc.
`EDGE; OVAL`	Builder: **Edges> Oval**	Creates a general curve that varies continuously from an ellipse to a rectangle.
`EDGE; RACETRACK`	Builder: **Edges> Racetrack**	Creates a rectangular edge with rounded corners.
`EDGE; ROUNDED`	Builder: **Edges> Rounded**	Same as Racetrack, but it can be extruded to an Ellipse or a Rectangle.
`EDGE; SAWTOOTH`	Builder: **Edges> Sawtooth**	Creates a repeating triangular edge.

ASAP Command	Menu	Description
EDGE; POINTS	Builder: **Edges> Points**	Creates very general curves using points, connection factors, control points, and weight factors.
OBJECT	Builder: **Object Control> Object**	Turns one or more entities into an object.
SMOOTH	Builder: **Edge Modifiers> Smooth**	An edge-entity modifier; it quadratically smooths a curve.
SWEEP AXIS SWEEP DIR SWEEP POS	Builder: **Edge Modifiers> Sweep** Select from drop-down menu in Options column: Axis, Dir, Dir To, or Pos	An edge-entity modifier; it sweeps the curve/edge to form an object.
FACETS	Builder: **Object Modifiers> Facets**	An object modifier; it changes the faceting for the object it modifies.
RENDER	ASAP Menu: **Render**	Uses methods similar to ray tracing to create a shaded rendering of objects.
CAD IGES/DXF	ASAP Menu: **System> Export to CAD**	Translates ASAP objects to IGES or DXF format.

In this chapter, we introduced the EDGE, a new kind of ASAP entity that is distinct from the SURFACE-based entities used prior to this. An edge is generally a set of connected points. The connections may be smooth curves, or straight lines. If the edge closes on itself, ASAP can make it into a planar object with the edge as its boundary, which is the default behavior in the Builder.

Edges can also be turned into objects by sweeping them around an axis, along a direction, or toward a point. To achieve this, we must turn the edge definition into an entity in the Builder prior to sweeping it. Then the OBJECT command must be used to turn the swept entity into an object that can interact with rays.

Two edges can also be turned into an object by extruding. By this, we mean connecting the points that form the two edges to define the new object. Once again, the edges that form the basic building block for this object must be given entity status in the Builder. Otherwise, ASAP also creates the bounded planes above and below the extruded object.

Edges have several advantages that make them valuable for system modeling:

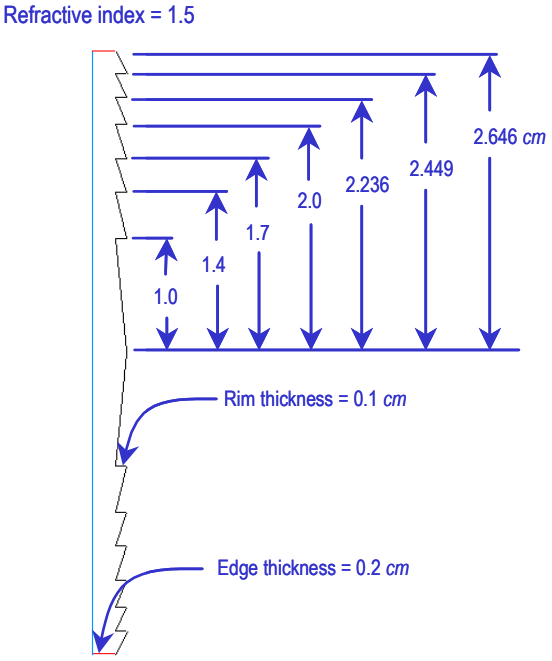
- Edges allow us to import CAD-based geometry into ASAP. For complex mechanical structures, a CAD program is often an easier and more efficient environment in which to work.
- Any object that we can model using ASAP edges will have an exact representation in the CAD environment, if you choose to export them. Some **SURFACE**-based objects must be approximated.
- Sweeping and extruding edges allow us to make many objects that would be difficult or impossible to make by using the implicit-polynomial mathematics of **SURFACE**-based objects.

There are also two disadvantages to edges that you should consider whenever you have a choice of an ASAP entity type:

- **EDGE**-based geometry generally slows down the ray trace compared with equivalent **SURFACE**-based models.
- **EDGE**-based geometry occasionally “leak”, allowing rays to miss a surface that was directly along their paths. These leaks are rare, however, and can usually be ignored.

Exercise 6: Creating a Fresnel Lens using EDGES

1 Define the Fresnel surface (back) of the lens with **POINTS**, both inner and outer, using the information in the figure below. Be sure to change the definition to entity status, and preview the edge to verify.



2 Sweep the edge around the *z* axis, and turn it into an object. Add a transmitting interface, assigning a medium with an index of refraction of 1.5. Set the number of facets used for this object at 11 11. Preview the resulting object to verify.

3 Define the outer edge of the Fresnel lens by extruding two arcs. Add an absorbing interface. Set the faceting for this object to 11 11. Preview this object to verify.

4 Define the front of the lens using a smoothed **ELLIPSE**. Give it the same transmitting interface as the lens back.

5 Define a collimated polar grid of rays directed at the lens front. Set the total flux of this source so that it will have 5 Watts/cm^2.

6 Trace the rays through the lens, and find best focus. Make a spot diagram of the rays at best focus.

Objects versus Entities

What is the distinction that ASAP makes between Object and Entity status?

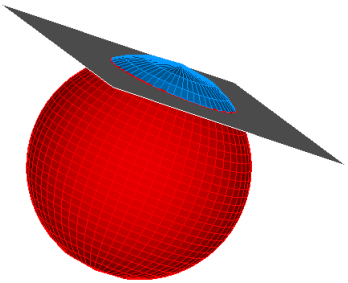
- An Object in ASAP is a geometrical surface with boundaries, a position, an orientation, and optical properties. Objects interact with rays.
- An Entity in ASAP is a geometrical definition that does not interact with rays. It has no interface command associated with it.

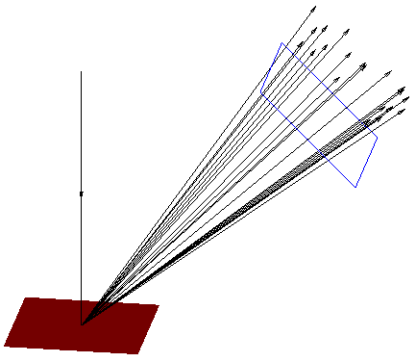
Objects and entities are defined using exactly the same ASAP commands. In fact, every object that you define in ASAP is considered to be an entity as well. Not all entities, however, will have object status.

But why would we want to go to the trouble to define an entity in our ASAP simulation if the rays will ignore it? There are at least four reasons:

***Object Building Blocks**. In this chapter we have used two edge entities to define a single object by extruding.*

***Bounding (trimming) entities.** In the figure on the right, a PLANE is used to bound an ELLIPSOID. We might need to do this to trim a cap off of an integrating sphere, keeping the lower (red) section, and discarding the top (blue) section. The bounding plane is used only to define the boundary of the spherical object; the rays will not see it.*





***Field clipping boundaries**. When using ASAP for wave optics, edge-entities are frequently used to define the limits of a field. The field outside or inside an edge is "clipped away" to simulate an obscuration or aperture before propagating the field forward.*

***Important-area boundaries**. In scattered-light applications ASAP allows us to use edge entities to define an important area or direction in space. This allows us to generate and trace only those rays that will scatter in an interesting direction.*

APPENDIX

13A

SCRIPTS FOR CHAPTER 13

The following ASAP scripts are referenced in Chapter 13, “Edges”.

Script 13-1

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY RUNNING THE ELLIPSE EXAMPLE.
!!THE SCRIPT HAS BEEN EDITED TO THE PREFERRED SCRIPT FORM.
!!EDGE STATUS IS ENTITY IN THE NEXT EXAMPLE.
EDGE
  ELLIPSE Z 1 1.0 1.0 16 0.0 360.0
    SMOOTH
!!EDGE STATUS IS OBJECT IN THE NEXT EXAMPLE.
EDGE
  RECTANGLE Z 1 1.0 1.0 16 0.0 360.0
    SMOOTH
OBJECT
```

Script 13-2

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY RUNNING THE BUILDER EXAMPLE.
!!EDGE STATUS IS ENTITY IN THIS EXAMPLE.
EDGE
  ELLIPSE Z 1 1.0 1.0 16 0.0 360.0
  RECTANGLE Z 0.0 2.0 2.0 16 0.0 360.0
```

Script 13-3

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY RUNNING THE BUILDER EXAMPLE.
EDGE
  ELLIPSE Z 1 1.0 1.0 16 0.0 360.0
  RECTANGLE Z 0.0 2.0 2.0 16 0.0 360.0
OBJECT
  .1  .2  'EXTRUDED OBJECT'
```

Script 13-4

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY RUNNING THE BUILDER EXAMPLE.
!!EACH OF THE EDGES NOW HAS SMOOTHING APPLIED.
EDGE
  ELLIPSE Z 1 1.0 1.0 16 0.0 360.0
  SMOOTH
  RECTANGLE Z 0.0 2.0 2.0 16 0.0 360.0
  SMOOTH
OBJECT
  .1  .2  'EXTRUDED OBJECT'
```

Script 13-5

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY RUNNING THE BUILDER EXAMPLE .
!!BUT NOW THE 2 EDGES ARE IN THE SAME PLANE.
!!EACH OF THE EDGES HAS SMOOTHING APPLIED.
!!THE RESULT IS AN SQUARE APERTURE WITH A CIRCULAR OPENING.
EDGE
  ELLIPSE Z 0.0 1.0 1.0 16 0.0 360.0
  SMOOTH
  RECTANGLE Z 0.0 2.0 2.0 16 0.0 360.0
  SMOOTH
OBJECT
  .1  .2  'EXTRUDED OBJECT'
```

Script 13-6

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY RUNNING THE BUILDER EXAMPLE .
!!THE COMMAND SCRIPT NOW INCLUDES THE COMMANDS TO SEE THE RESULT IN THE 3D VIEWER.
!!THE HIGHER FACETS VALUE MAKES THE OBJECT SMOOTHER IN THE 3D VIEWER.
EDGE
  LINE 0 -2 1 0 -1 0
  SWEEP AXIS 360 0 0 1
OBJECT
  .1  'CUP'
  FACETS 1 21
RETURN
PLOT FACETS
$VIEW
```

Script 13-7

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY RUNNING THE BUILDER EXAMPLE.
!!THE COMMAND SCRIPT INCLUDES THE COMMANDS TO SEE THE RESULT IN THE 3D VIEWER.
!!THE INITIAL COMMANDS TO RESET THE SYSTEM ARE INCLUDED AT THE BEGINNING.
SYSTEM NEW
RESET
COATING PROPERTIES; 0 0  'ABSORB '
EDGE
  POINTS X 0
  2.51  -0.01  1
  2.51  1.00   1
  3.00  1.00   1
  3.00  -0.50  1
  2.00  -0.50  1
  2.00 -0.01  1
  SWEEP AXIS 360 0 0 1
OBJECT  'LENS.CELL'
  INTERFACE COATING ABSORB AIR AIR
  REDEFINE COLOR light blue      3
  FACETS 11 11
PLOT FACETS
$VIEW
```

CHAPTER

14

LENS ENTITIES

ASAP supports three entity types. We have already introduced **SURFACE**-based and **EDGE**-based entities. We will now discuss the third entity type: lenses. Although we have modeled lenses before this, we always used the **OPTICAL** command (or in the Builder from **Geometry> Surfaces> Spherical**). Would lens entities have been a better choice? No, not necessarily. Lens entities have some special properties, and also some special limitations. The name is also somewhat misleading. This class of entity has been given the name “lens” because the nature of the entities and their parameterization is similar to that used by lens-design programs. ASAP lens entities include models of lenses, mirrors, windows, prisms, and even telescopes. In the sections that follow, we will explain what these components have in common, and how they are treated differently when modeled as lens entities, rather than surfaces or edges.

What is a Lens Entity?

A **LENS**-based entity in ASAP is a sequence of refractive or reflective conicoid surfaces (Figure 14.1). The entire sequence is treated as a single object. Rays must enter through the first or last conicoid in the sequence, and must exit through the last or first. The rays propagate sequentially through the intervening conicoids. If the ray fails to intersect the next conicoid in the sequence, it will stop.

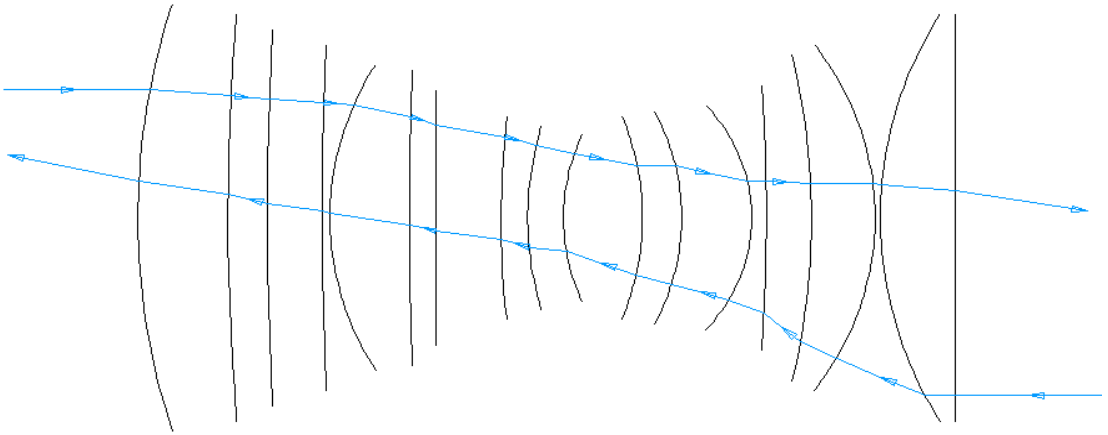


Figure 14.1 An ASAP lens entity is a sequence of conicoid surfaces. The sequence is treated as a single object. As the figure shows, rays may proceed from left to right, or right to left, but they must encounter each conicoid in sequence.

This sequential behavior is clearly contrary to the usual philosophy of ASAP. Up to this point, all our objects have been treated as independent, and rays proceeded through systems in a completely unconstrained way. Why would we want to define entities that break these fundamental rules of physically accurate ray tracing? There are several reasons:

- As we noted in Chapter 1, sequential ray tracing is much faster. The nine lenses (18 conicoids) shown in Figure 14.1 will trace rays five times faster when modeled as a lens sequence than as 18 **OPTICAL** and nine **TUBE** surfaces.
- Lens entities are easy to define in terms of classical optical parameters, like focal length, thickness, bending factor, magnification, and field of view.
- ASAP includes tools for locally or globally optimizing centered imaging systems that are defined in terms of lens entities. (These features are available only when using command scripts. See **MINIMIZE** and **ABERRATIONS** in the ASAP on-line Help.)
- Lens entities have an exact IGES representation, and so, like edges, export easily to CAD programs.

The ease of use and ray tracing speed come at a cost, however. Some significant disadvantages exist for using lens entities to construct ASAP objects. It is not possible:

- to assign different optical properties to the individual surfaces, since the entire lens sequence is treated as a single object.

- for a ray to scatter and exit the sequence.
- for a ray to enter the sequence of surfaces other than through the first or last surface.

Note: The one positive exception to this is: when a ray splits due to Fresnel reflections, the split ray can proceed through the sequence, even though it did not enter through the first or last surface in the sequence. This condition allows us to do ghost-image analysis with **LENS**-based objects.

One way exists to keep some of the best features of lens entities—their ease of use and logical parameterization—while having all the advantages of non-sequential ray tracing. ASAP includes a built-in feature to convert a lens sequence to an equivalent set of surface-based objects. We will demonstrate this after you have seen a few lens definitions.

Singlet Lenses

A singlet lens is a simple lens element made of a single, homogeneous glass type. Each of the three elements of the Cooke triplet was a singlet. A singlet is fundamentally a "sequence" of two surfaces. When you modeled these in Chapter 4, you were given the radii of curvature for the front and back surfaces, and used the **SURFACE**-based **OPTICAL** command to create the two surfaces. We could have defined both of these surfaces as just one **LENS** entity with the **SINGLET** command. This command is available in three forms on the Builder menu: **Geometry> Lenses> Singlet**. From the drop-down menu in the second Type column, select **RD** for radius of curvature, **CV** for curvature (1/radius), or **FL** for focal length. The first element of the Cooke triplet could have been defined with the **RD** option in this way:

*	Type	Name	Axis	Location	Thickness	Semidiameter	Media1	Type	Radius1	Radius2
OBJ	Singlet	L1	Z	0	5	16.9	SCHOTT_SK4	RD	44.55	-436.6
MOD	Interface	Coating	COATING	Transmit	Air	Air				

Figure 14.2 Builder line for the first element of the Cooke triplet

The medium is defined as **schott_SK4**. This is one of the materials known to ASAP through the built-in glass catalogs, so you do not need a **MEDIA** command. We will have more to say about the glass catalog in the following section.

See Chapter 14 Appendix,"Script 14-1" on page 291.

If you **Preview** this definition in the Builder, the result should appear as shown in Figure 14.3 on page 270.

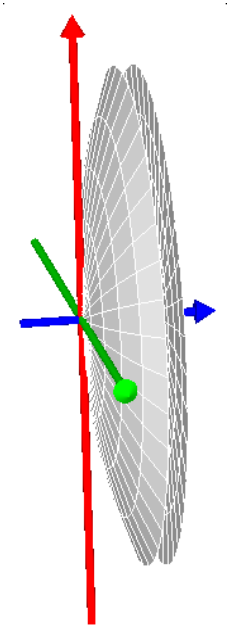


Figure 14.3 When you preview a single lens (or any other lens entity with multiple surfaces), note that all surfaces are the same color. This result occurs because ASAP treats the lens entity as a single object. ASAP does not model the edge, since no ray is allowed to enter or leave the sequence through this gap.

Two important features to notice are:

1. Both surfaces in the preview are the same color (white, by default). As we have said, ASAP treats the singlet as one object.
2. The surface has no edge. The tube that seals the front and the back surface is missing. It is, in fact, completely irrelevant to the model. Rays are not permitted to enter or escape through this gap. Any ray with such a trajectory will not be traced further.

Notice the **INTERFACE** command that was applied to the **SINGLET** lens. Normally, we would specify two media, one on each side of each surface. In this case, since we have specified the medium *between* the two surfaces within the **SINGLET** command, the **INTERFACE** command lists **Air** and **Air** as the medium at both ends of the lens sequence. The coating specified on the **Interface** line applies to all surfaces within the lens sequence. In this case, we have used a **TRANSMIT** coating, presumably defined previously with the **COATING** command. We have no option to assign different properties to the individual surfaces, unless we choose to convert the **LENS**-based object to independent surfaces (see “Converting Lenses to Surface-based Objects” on page 275).

You can also use the **SINGLET** command to create a lens of a specified focal length. Of course, an infinite number of lenses exist with a given thickness,

material, and focal length, if we are permitted to vary the radii of both surfaces. You could use the lens-maker's formula to discover an approximate prescription that would yield the correct focal length, but the **SINGLET** command with the **FL** option can design the lens for you. All you need to do is specify a bending parameter. The bending parameter is a single value that lets you choose from among the possible surface curvatures. It is defined as follows:

$$b = \frac{r_2 + r_1}{r_2 - r_1}$$

where r_1 and r_2 are the front and back radii of curvature respectively. Remember that these radii can be either positive or negative, depending on whether the center of curvature is to the right, or to the left of the surface. Figure 14.4 shows some examples of singlet lenses over a range of bending parameters. A plano-convex lens has a bending parameter of either +1 or −1, depending on whether the convex surface is in the front or the back. A symmetrical bi-convex lens has a bending parameter of 0.

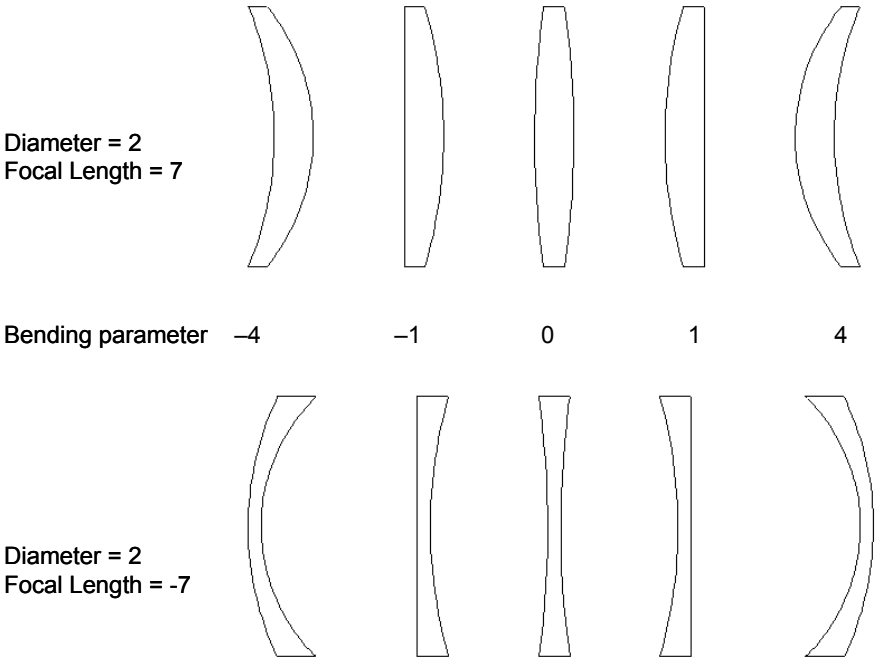


Figure 14.4 All the positive lenses on the top row have the same focal length. The same is true of all the negative lenses in the bottom row. The bending parameter was used to select among all the possible lenses with the same focal length. While these lenses have the same focal length, spherical aberration and coma do change as a function of the bending parameter. For example, a positive single lens made of BK7 has minimum spherical aberration for an object at infinity when the bending factor is +0.7 (highest curvature toward the object).

Try making a plano-convex lens using **Lenses> Singlet** with a focal length (**FL**) in the Builder. Give the lens a 25-mm diameter, a 4 mm thickness, and a focal length of 100 mm. Make the convex surface the first side: the bending parameter will be +1. When you finish, the Builder line should look like Figure 14.5.

*	Type	Name	Axis	Location	Thickness	Semidiameter	Media1	Type	Focal Length	Bending Factor
OBJ	Singlet	LENS1	Z	0	4	12.5	SCHOTT_BK7	FL	100	1.0

Figure 14.5 Builder line for a Plano-convex lens

See Chapter 14 Appendix,"Script 14-2" on page 291.

Using the Glass Catalogs

ASAP includes nine built-in glass catalogs: French, Hikara, Hoya, Hoyanew, Ohara, Schott, Schottnew, Sumita, and Unusual. These catalogs include most of the commercially available optical glasses. The Unusual catalog includes additional materials—such as calcite, sapphire, silicon, and zinc selenide—commonly used in optical design.

You can use the glass catalogs to enter pre-defined glass types by name in the **INTERFACE** command, or any of the **LENS** commands that require you to specify a medium. In the **Builder**, click once in the **Media 1** cell to select it, and then press **Ctrl-Enter**. You can then browse the catalogs to the desired material (Figure 14.6).

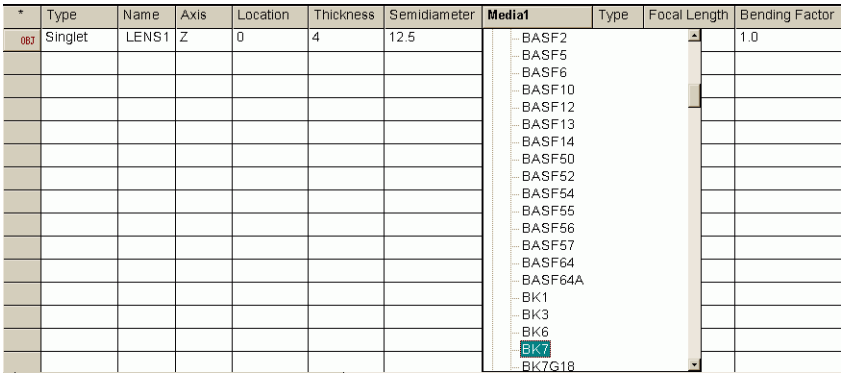


Figure 14.6 You can access the glass catalogs from any **Builder** line that requires you to specify a medium. Click once in the **Media** cell and press **Ctrl+Enter**. The same glasses can be used in command scripts by specifying the catalog and glass type separated by an underscore; for example, **SCHOTT_BK7**. This string can be used in place of a medium you had defined using the **MEDIA** command.

Another method for accessing the glass catalogs is through the **Quick Start** toolbar (see Figure 14.7). From the **Glass Catalog** tab, click the plus sign next to a catalog name (for example, **french**), and double-click a specific glass name. The **Glass Editor** dialog displays, where you can see the dispersion function (index versus wavelength) for the specific glass you selected. You can also add a new glass name to the tree list, or assign wavelength/refractive index pairs.

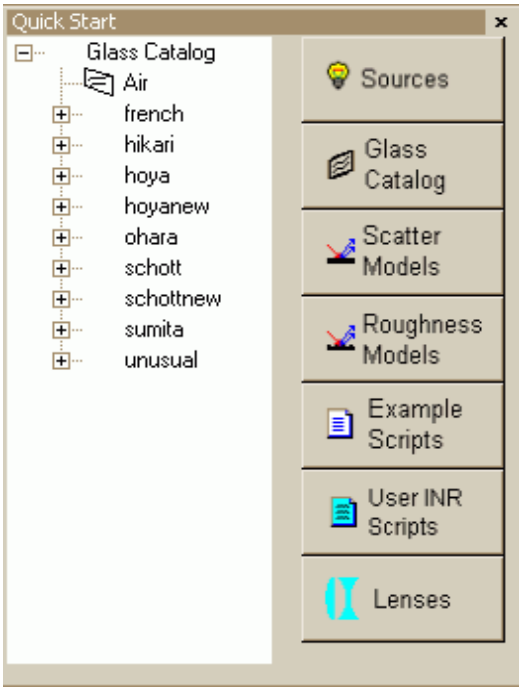


Figure 14.7 Optical Properties toolbar, with the Glass Catalog tab in view

One important advantage to using the glass catalogs is the inclusion of dispersion equations with the media definitions. All the entries in the catalogs include coefficients for calculating the index of refraction of the material as a function of wavelength. Unlike the linear interpolation performed for materials defined using the **MEDIA** command, the calculations for the materials in the glass calculations are based on the evaluation of a Schott or Selmeier polynomial.

Doublet Lenses

The **DOUBLET** command is similar to **SINGLET** with the **FL** option. It allows you to "design" a cemented doublet lens. Enough free parameters exist in such a lens to allow ASAP to create an element that minimizes axial chromatic aberration. In other words, the lens will have nearly the same focal length over a range of wavelengths. You need to specify the total thickness, semidiameter, the two

media used, the effective focal length, and a bending factor (as described in the section, “Singlet Lenses” on page 269). See Figure 14.8.

See Chapter 14 Appendix,“Script 14-3” on page 292.

*	Type	Name	Axis	Location	Thickness	Semidiameter	Media1	Media2	Focal Length	Bending Factor	Abbe F
SYS	Wavelength		450	550	650			Nanometers			
OBJ	Doublet	L1	Z	0	9.5	17	SCHOTT_BAK2	SCHOTTNEW_F2	100	0.30653	-5.0

Figure 14.8 Builder line for a cemented doublet lens

In practice, lens designers can also minimize other aberrations in the doublet, particularly spherical and coma, by using glass types and a bending factor as additional degrees of freedom. ASAP does not attempt to do this in the **DOUBLET** command. Our doublet only minimizes axial chromatic aberration.

We have used a **WAVELENGTHS** command to specify three wavelengths. ASAP is now able to calculate the dispersions of the two glasses using the glass catalog. This calculation is critical to the design of the achromatic doublet.

> ***Note:*** The calculation is done in terms of a generalized Abbe number, defined as follows:
>
> $$V = \frac{n_2 - 1}{n_1 - n_3}$$
>
> When we use catalog glasses, ASAP calculates the refractive indices n_1, n_2, and n_3 corresponding to the three wavelengths specified in the **WAVELENGTHS** command. It calculates V for both glass types (V_1 and V_2). The ratio of Abbe numbers dictates the ratio of powers of the two doublet components to yield the achromatic condition. The program determines this ratio using the dispersion functions provided by the glass catalog. If you do not use media from the glass catalogs, you must include **MEDIA** commands that specify three indices of refraction for each glass.

Alternatively, you have the option of specifying the ratio of Abbe numbers yourself, using the last parameter of the command: `Abbe Ratio`. This is defined as $r = V_2/V_1$. Abbe numbers are generally available from the glass manufacturers. In our example above, Schott lists the Abbe number for BAK2 as 59.7, and F2 is 36.4. The ratio in this case is $V_2/V_1 = 0.61$.

When using the glass catalogs, if you do not provide either a **WAVELENGTHS** command or an Abbe ratio, ASAP calculates the ratio for you, estimating the dispersion of the catalog glasses at 587.6 nm.

Converting Lenses to Surface-based Objects

The **DOUBLET** command (see “Doublet Lenses” on page 273) is a good example of the advantages of lens entities. With a single command, we have designed an achromatic objective. ASAP has determined the optimum curvature of three surfaces for us.

Now that we have the lens model, what happens when we want to apply a coating to the entrance surface, and some scattering properties to the rear? What happens when stray light in the system interacts with the (nonexistent) edges of this doublet lens? We can still accomplish our objective for the doublet, or any other sequences of surfaces defined, by using lens entities. ASAP has the ability to “explode” a lens prescription into independent **SURFACE**-based objects.

To illustrate the ASAP **EXPLODE** command, we will begin with the doublet defined above.

1. Select the Builder line defining the lens entity by clicking anywhere on that line.
2. Click **Explode** on the Builder toolbar.

Alternatively, you can run the **EXPLODE** command by right-clicking after selecting the lens entity definition and selecting **Explode Lens** in the pop-up menu.

The result of the explode operation appears in Figure 14.9 on page 276.

*	Type	Name	Axis	Location	Thickness	Semidiameter	Media1	Media2	Focal Length	Bending Fa	Abbe
SYS	Wavelengths		468.1					Nanometers			
OBJ	Doublet		Z	0	0.4	1	SCHOTT_BK7	SCHOTT_BK6	8	0	
OBJ	Optical		Axis	Z	0.000000	7.987073	0.000000				
MOD	Redefine	COLOR									
MOD	Interface	Coating	COATING	BARE	"schott_BK7"	"VACUUM|AIR					
OBJ	Optical		Axis	Z	0.3600000	-2.758996	0.000000				
MOD	Redefine	COLOR									
MOD	Interface	Coating	COATING	BARE	"schott_BK6"	"schott_BK7"					
OBJ	Tube		Axis	Z	0.1723964	1.000000	1.000000	0.6284843E-01	1.000000	1.000000	0
MOD	Redefine	COLOR									
OBJ	Optical		Axis	Z	0.4000000	-7.987073	0.000000				
MOD	Redefine	COLOR									
MOD	Interface	Coating	COATING	BARE	"VACUUM|AIR"	"schott_BK6"					
OBJ	Tube		Axis	Z	0.3371516	1.000000	1.000000	0.1723964	1.000000	1.000000	0
MOD	Redefine	COLOR									
MOD	Return										

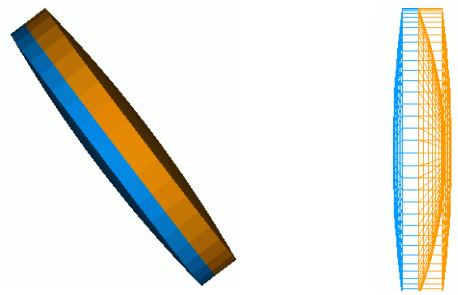


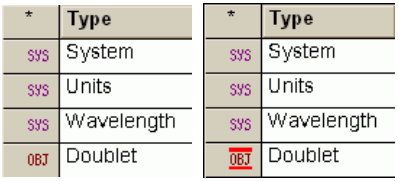
Figure 14.9 The **Builder** can “explode” a lens-based doublet into independent surface-based objects. The doublet line turns into several Builder lines, including five surface definitions, interface commands, and color changes for the graphics. You may want to modify some of these defaults. The “bare” coatings specified on the interface commands, for example, cause ray splitting due to Fresnel reflections, which may not be desirable, depending on the goals of your analysis.

ASAP has converted the sequence of surfaces into five independent **SURFACE**-based objects: three **OPTICAL** surfaces and two **TUBE** surfaces. All the restrictions that apply to lens entities are now removed. ASAP has added **INTERFACE** commands to the two optical surfaces, and given them the default **BARE** coating. The tube has no **INTERFACE** command. ASAP treats the tube as an absorbing surface (the default when there is no interface specified).

Note that there are only three optical surfaces in this exploded model. A lens maker would produce such a doublet by fabricating two lenses and cementing them together. There would actually be four optical surfaces in all. Many new ASAP users are tempted to model such a situation in the same way, resulting in two central surfaces. Recall, however, that in ASAP two surfaces should never

occupy the same space. To model this situation in the most accurate way possible, we might model all four surfaces separated by a cement layer of finite thickness. Since index-matching cement may be used to reduce Fresnel reflections, this might be necessary for an accurate estimate of ghost-image contributions from this interface. This detail is often neglected, however.

ASAP has also made a change to the **DOUBLET** command in the Builder. Note that the symbol to the left of the line now has red bars above and below it.



*	Type
SYS	System
SYS	Units
SYS	Wavelength
OBJ	Doublet

*	Type
SYS	System
SYS	Units
SYS	Wavelength
OBJ	Doublet

Figure 14.10 Builder status column (*****) before (left) and after (right) exploding the Doublet

ASAP has changed the status of this line so that it will be ignored when the Builder file is run or previewed. The line was left in the file only to serve as a comment, since it includes useful information about the lens prescription in a logical and convenient format. This line should not remain active, however. If it is executed, we will have two valid versions of the doublet at the same location.

> ***Note:*** You can toggle the status of any line yourself by right-clicking the line and selecting **Line Status** from the pop-up menu. You can also do this with the **Line Status** button in the Builder's tool bar.
>
> This is a convenient Builder feature, allowing you to temporarily ignore specific Builder lines without permanently deleting them.

See Chapter 14 Appendix,"Script 14-4" on page 293.

Other Lens Entities

Figure 14.11 on page 278 shows five additional lens entities available, and Figure 14.12 on page 279 shows a Builder file with each of the five lens entities. A description of each lens entity follows the caption for Figure 14.11.

Most of these entities are not, in fact, lenses at all. Remember, these objects are called "lens entities" only because the nature of the entities and their parameterization is similar to that used by lens-design programs.

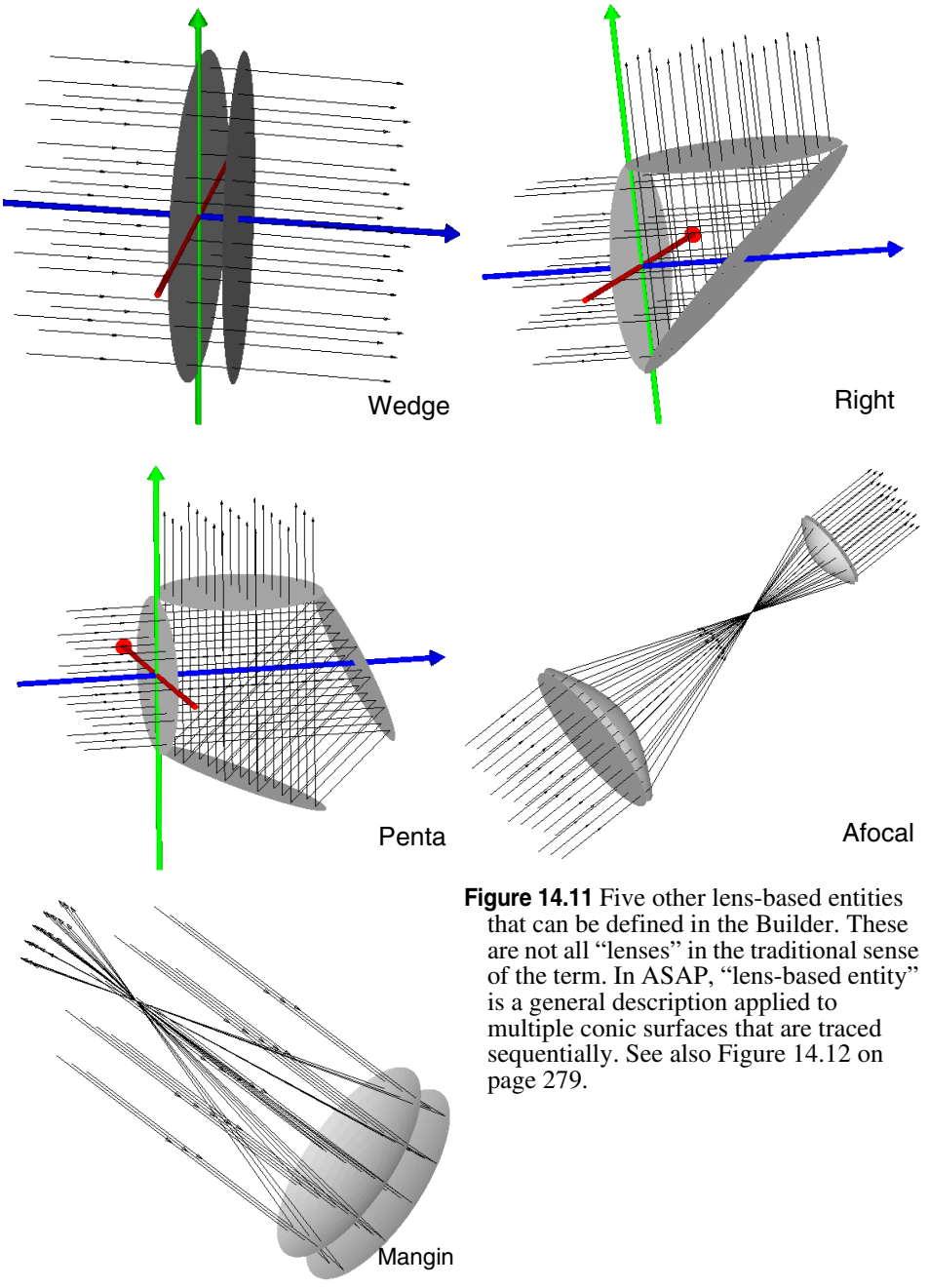


Figure 14.11 Five other lens-based entities that can be defined in the Builder. These are not all "lenses" in the traditional sense of the term. In ASAP, "lens-based entity" is a general description applied to multiple conic surfaces that are traced sequentially. See also Figure 14.12 on page 279.

*	Type	Name	Axis	Location	Thickness	Semidiameter	Media1	Type	Focal Length	Bending Factor	Conjugate Fac
OBJ	Wedge	WINDOW1	Z	0	1	SCHOTT_BK7	5	0.25			
OBJ	Right	PRISM1	Z	0	1	SCHOTT_BK7	Y,Z				
OBJ	Penta	PRISM2	Z	0	3	SCHOTT_BK7	Y Z				
OBJ	Afocal	TELESCOPE	Z	0	0.3	1	0.5	Media	SCHOTT_BK7	Media	SCHOTT_BK6
OBJ	Mangin	MIRROR1	Z	0	0.3	1	SCHOTT_BK7	FL	3.0	4.725	

Figure 14.12 Builder file for other lens-based entities.

Geometry> Lenses> Wedge—a wedge that creates a glass window. A wedge angle can be added, which is sometimes done to reduce interference effects between the two planar surfaces.

Geometry> Lenses> Right—a simple prism that bends rays at a right angle.

Geometry> Lenses> Penta—a penta prism that bends rays at a right angle, but does so without introducing a left-to-right or top-to-bottom inversion of images. In other words, an image formed through such a prism is congruent.

Geometry> Lenses> Afocal—an instrument with no focal length (that is, one that does not bring an image to a focus). A telescope is an example of such a system, since an object located at an infinite distance appears to be at infinity when viewed through the instrument. Only the magnification of the object is changed. The **AFOCAL** command can be used to define simple two-element refracting or reflecting telescopes. ASAP adjusts the conic constant of the elements to correct spherical aberration.

Geometry> Lenses> Mangin (select **CV, FL, RD** from the second **Type** column)**—**a Mangin mirror, which is a single optical component with both a reflecting and refracting surface. The parameters for the Mangin mirror are identical to those of the singlet lens. ASAP does not attempt to optimize the Mangin mirror in any way, though a lens designer can use the thickness, glass type, and surface curvatures to minimize spherical aberration, and partially correct coma.

Defining a Lens Sequence

The **SEQUENCE** command is the most fundamental of all lens-entity definitions. It allows us to enter an arbitrary sequence of conic surfaces, again in a format compatible with lens-design tools. Figure 14.13 on page 280 shows a sequence specification for the Cooke triplet that we modeled in “Exercise 1: Completing the Cooke Triplet” on page 83 of Chapter 4.

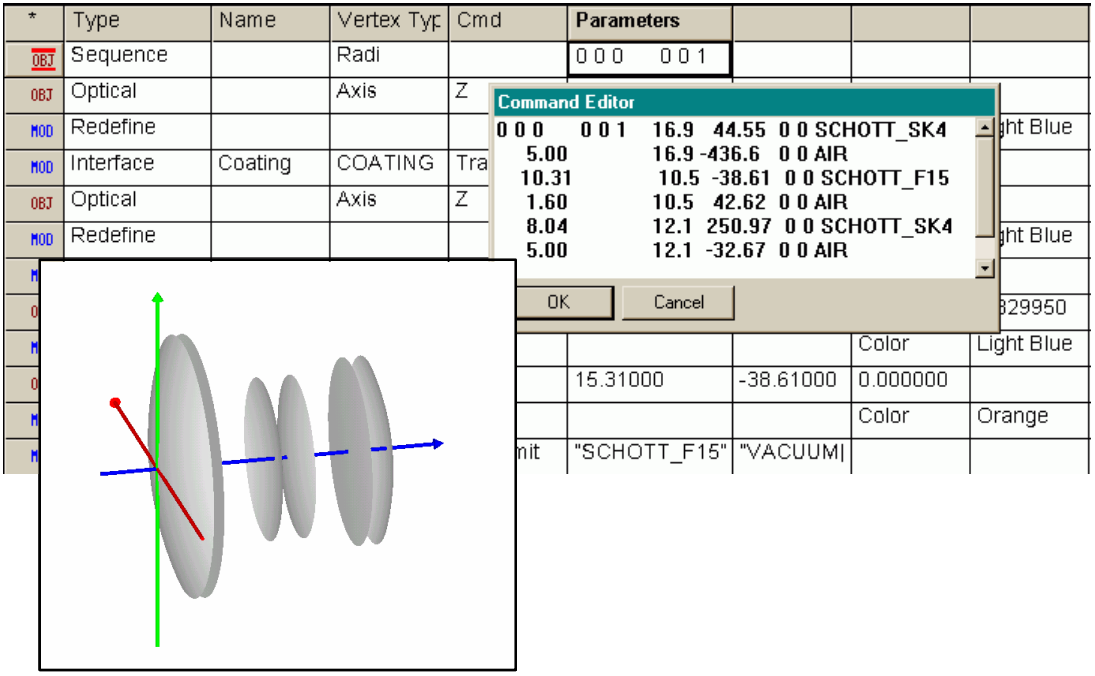


Figure 14.13 The Cooke triplet cast as a lens sequence. The parameters for the sequence are displayed in the **Command Editor** window, which is opened by double-clicking the **Parameters** cell on the **Sequence** line of the **Builder**.

You can define this sequence in the Builder using **Geometry> Lenses> Sequence**. As with all objects entered in the Builder, you can name the sequence in the **Name** cell. Like other **LENS**-based objects, the entire sequence is treated as a single object, and the name applies to the whole.

The next cell is labeled **Vertex Type**. You can specify each conic either in terms of its radius of curvature (**Radi**) or its inverse, the curvature (**Curv**). Double-click this cell to make the selection from the drop-down list box. Our original specification for the Cooke triplet was given in terms of radii of curvature, so we used it for this example.

We enter the specific parameters for the sequence in the Command Editor window. This approach is similar to the way we input parameters for the **POINTS** command in Chapter 13. The editor is launched by double-clicking in the **Parameters** cell (or by pressing **Ctrl-Enter**). Each conic in the sequence is specified on one line. The first line must be specified in the "long format" consisting of 11 values:

x y z	*a b c*	*h*	*r*	*k*	*o*	*m*
0 0 0	0 0 1	16.9	44.55	0	0	SCHOTT_SK4

The first three parameters are the *x, y*, and *z* global coordinates of the vertex of conic number 1. In this case, we want to locate this first conic at the origin of coordinates (0, 0, 0).

The next three parameters specify the orientation of the conic. This is done in terms of the surface normal at the position of the vertex of the conic, expressed as a direction-cosine vector (a, b, c). We are defining the Cooke triplet along the *z* axis, so the direction vector is (0, 0, 1).

The seventh parameter is *h*, the aperture height (semidiameter), which is 16.9 mm for the first element of the Cooke triplet.

Next comes the *r* parameter. Since we have selected the **Radi** option, this number is a radius of curvature, 44.55 mm.

The ninth parameter is the conic constant. All the surfaces in the Cooke triplet are spherical, so the conic constants are all 0. Parameter 10 is *o*, the obscuration (or hole) ratio. There is no hole in this element, so the ratio is entered as a zero.

The final parameter is the number or name of the medium into which a ray will pass. We do not need to specify two media as we did when using the **INTERFACE** command. We presume that the sequence is specified as though a ray were passing from the outside of the object, through the first conic in the sequence.

> ***Note:*** If this conic were a reflecting surface, you would use the keyword **REFL** in place of the media name. For example, the long format specification for the primary mirror of the Cassegrain telescope modeled in Chapter 6, “Exercise 2: Cassegrain Telescope” on page 122, would look like this:

x y z	a b c	h	r	k	o	m
0 0 0	0 0 1	5.0	-40.0	-1.00	0.25	REFL

The other five conics in the Cooke triplet can be entered in the “short format”, where the first six parameters are replaced by a single number: the distance from the previous conic. You may use this shorter format anytime subsequent conics have the same surface normal specified in the last long-format entry. The other entries look like this:

d	h	r	k	o	m
5.00	16.9	-436.60	0	0	AIR
10.31	10.5	-38.61	0	0	SCHOTT_F15
1.60	10.5	42.62	0	0	AIR
8.04	12.1	250.97	0	0	SCHOTT_SK4
5.00	12.1	-32.67	0	0	AIR

Once this sequence is created, you have the option to explode it as described above. Note, however, that ASAP attempts to connect *all* the resulting independent **SURFACE**-based objects with tubes, even when this is not appropriate. This behavior is necessary so that **EXPLODE** can work under the most general circumstances. You will need to delete the inappropriate tubes manually (see “Exercise 7: The Cooke Triplet as a Lens Sequence” on page 286 at the end of this chapter).

Ideal Lenses

The ideal lens is a useful ASAP object if you want to build a system containing refracting elements that behave “perfectly”. The *ASAP Reference Guide* describes the ideal lens as perfect, but non-physical. It is perfect in the sense that there are no ray aberrations over the full range of object and image distances. It is non-physical in that such a perfect optical element is impossible to make in practice. To behave perfectly, the ideal lens must violate, or at least stress, a few laws of physics along the way.

Why would you want to include a “non-physical” element in your ASAP model? ASAP engineers often use ideal lenses in the initial layout of an optical system just to verify the basic “first order” design. Further, by substituting ideal lenses for real refractive elements, you can place an upper bound on system performance. The ideal lens is listed among the other lens entities in the Builder’s command menu: **Geometry> Lenses> Ideal**. An example is shown below.

*	Type	Name	Axis	Location	In Semiap	Distance	Out Semiap	Random	FCN	ABCD Matrix	Obj Distance	Jones Matrix
OBJ	Ideal		Z	0	1	0.4	1		;	1 0 -0.125 1 0	0	

Figure 14.14 Builder line for the Ideal lens

See Chapter 14 Appendix,“Script 14-5” on page 294.

This particular example of the **IDEAL** command can be used to model an *f*/2 lens to image objects located at infinity. Its behavior is shown in Figure 14.15.

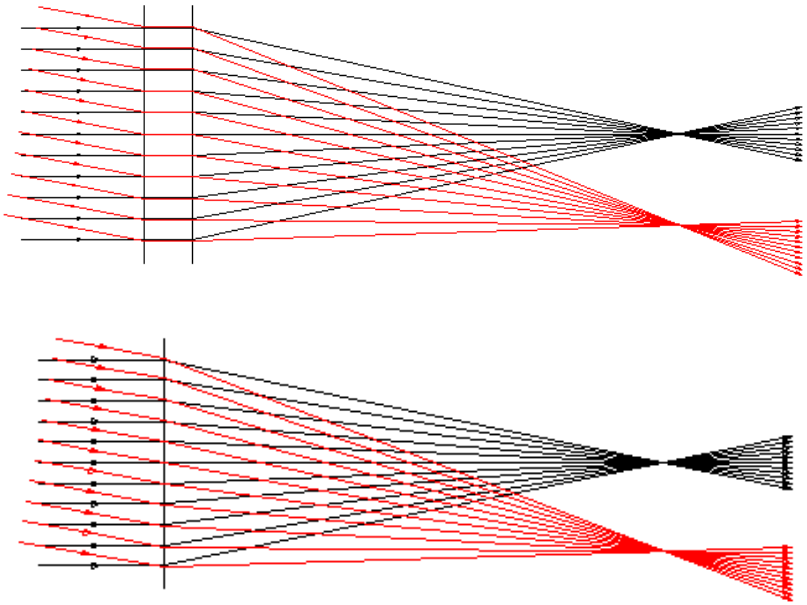


Figure 14.15 These ideal lenses simulate the behavior of perfect focusing elements. In the example on the top, an ideal thick lens is represented graphically by an input and output plane. These are the "principal planes" of the thick lens. The ABCD matrix dictates how the rays should behave during the ray trace. In this example, the matrix causes them to focus. On the bottom, a perfect thin lens was obtained by setting the **Distance** parameter to zero. The specification is otherwise identical.

The ideal lens has an input plane and an output plane. We can specify the size of these planes in the **In Semiap** and **Out Semiap** cells, and their separation in the **Distance** cell. These are the principal planes of the lens. As the figure shows, the rays are passed from the input to the output principle plane. They then bend appropriately to come to a perfect focus. Their behavior is dictated by the ABCD matrix elements for the device we wish to model. You do not need to understand the details of this process, though a brief discussion is provided in the sidebar, "Principal Planes and Matrix Optics" on page 289. It is sufficient to know that the ABCD matrix for a lens with focal length *f* (measured from the second principal plane to the focal point) is given by

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -1/f & 1 \end{pmatrix}$$

We enter this equation into ASAP as (*A, B, C, D*) = 1, 0, -1/*f*, 1. In the example above, the focal length is 4, so *C* = -1/4.

The last parameter in the **IDEAL** command is **Obj Distance**, the distance from the object to the first principal plane. Although the ideal lens has no ray aberrations

for any object distance, the wave fronts from this lens can be perfectly spherical only for a single object distance. If this is important to your analysis, you should enter the appropriate distance in this cell. Otherwise ASAP assumes that the object is at infinity, which is entered as an object distance of zero.

> ***Note:*** When the `IDEAL` command is used with the scripting language, you also have the option to enter a second matrix. This is a Jones matrix, which allows you to change the state of polarization of the input and output rays. This feature is discussed in detail in the Technical Guide, *Polarization*.

Summary

In this chapter, we have discussed the following new commands:

ASAP Commands	Builder Menu	Description
`LENS; SINGLET`	**Geometry> Lenses> Singlet**	Creates a singlet lens.
`LENS; DOUBLET`	**...Lenses> Doublet**	Creates a cemented doublet lens with minimum chromatic aberration.
`LENS; WEDGE`	**...Lenses> Wedge**	Creates a window, with an optional wedge.
`LENS; RIGHT`	**...Lenses> Right**	Creates a right-angle prism.
`LENS; PENTA`	**...Lenses> Penta**	Creates a penta prism.
`LENS; AFOCAL`	**...Lenses> Afocal**	Creates a two-element, afocal telescope.
`LENS; MANGIN`	**...Lenses> Mangin**	Creates a Mangin mirror.
`LENS; SEQUENCE`	**...Lenses> Sequence**	Creates a sequence of conic surfaces.

ASAP Commands	Builder Menu	Description
LENS; SINGLET	**Geometry> Lenses> Singlet**	Creates a singlet lens.
LENS; IDEAL	**...Lenses> Ideal**	Creates an idealized optical element.
EXPLODE	Right-click or Builder button	Converts lens definition into equivalent surfaces.

We introduced the LENS, the third and last kind of ASAP entity. We have already discussed SURFACE-based objects in Chapter 4 and Chapter 6, and EDGE-based objects in Chapter 13. All LENS–based objects are defined as a sequence of conic reflecting or refracting surfaces. As such, they are not limited to "lenses" in the traditional sense of the term. Prisms, telescopes, reflecting and catadioptric systems can also be defined using LENS–based objects. ASAP traces rays through these objects "sequentially" In other words, rays must interact with the conics in the order (or reverse order) in which they were defined.

Because they are conics, any object defined as a LENS can also be modeled exactly with surfaces or edges. You might choose to model with the lens version over the equivalent surfaces or edges for these reasons:

- In general, rays trace faster through LENS–based objects when compared with equivalent SURFACE-based or EDGE-based objects.
- The parameterization of LENS–based objects is simple, logical, and convenient for the optical components they describe.
- In many cases ASAP can actually design a simple optical component, sometimes even minimizing aberrations for you.

Modeling with LENS–based objects also has disadvantages, however. Their sequential nature, and the treatment of the entire sequence of conics as a single object, make it impossible to perform most types of scattered light analysis, for example.

You can use the EXPLODE command to turn LENS–based objects into equivalent SURFACE-based objects. While the exploded version traces more slowly, you have still retained the other advantages, and eliminated all the disadvantages noted above.

Exercise 7: The Cooke Triplet as a Lens Sequence

1. Define the Cooke triplet using **Geometry> Lenses> Sequence** as partially shown in Figure 14.13 on page 280, and described in, “Defining a Lens Sequence” on page 279. Name the sequence **L1**.
2. Add a pupil stop, defined as a circular **PLANE** with a hole in the center. The specification is as follows:

   Location: $z = 18.8$

   Outer Semidiameter: 16.9 mm

   Hole Semidiameter: 10.4

   Name: Stop

   Interface: Absorb
3. Add a detector, defined as a **PLANE** with the following specification:

   Location: $z = 118.82$

   Semidiameter: 10 mm

   Name: Detector

   Interface: Absorb
4. Verify the geometry with a profile.

Figure 14.16 A profile of the geometry for the Cooke Triplet

5. Define an elliptical grid of collimated, on-axis rays (**GRID ELLIPTIC** with **SOURCE DIRECTION**) with a size that fills the first Cooke element. The rays should be located at $z = -1$. Create 11 rays along each axis.
6. Run the Builder to create the Cooke triplet and the rays.
7. Initiate a ray trace by selecting **Trace Rays** from the main menu. When the dialog box appears, select **Plot Rays w/Geometry**, check **Missed Rays** with **Arrows** of length **5**, and select **Plot Facets** in the **Plot Geometry Options** section.
8. View the result in the 3D Viewer. Note that some of the rays that should have gone to the stop are drawn as “missed rays”. These are the rays leaving element 1 near the outside. Why didn’t they trace to the stop?

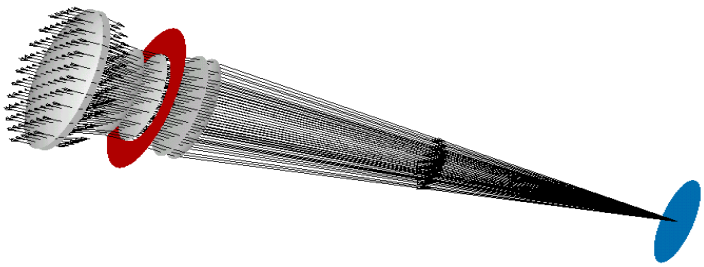


Figure 14.17 View of rays in the 3D Viewer

9 Increase the number of rays to 101 along each axis, run the Builder again, and repeat the trace. This time, choose **No Plot** in the **Trace Options** area. Make a note of the time ASAP spent performing the trace. This information is printed in the **Command Output** window after the `TRACE` command is executed:

```
--- TRACE
Total of 440.19 millisec (439.94 millisec CPU) to trace 8,021
```

10 Explode the **Sequence** definition in the Builder. Verify the exploded geometry with a profile. Note the “extra” tubes where ASAP has attempted to connect the Cooke elements. If this region were filled with glass to form a solid, cemented sequence, these tubes would be appropriate. They are not useful here since this is an air space. ASAP named these tubes **L1.Tube.4** and **L1.Tube.8**. They are the only objects with a red color. Delete these two tubes and associated `REDEFINE COLOR` commands.

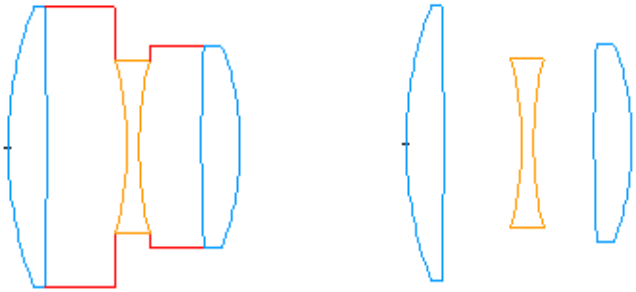


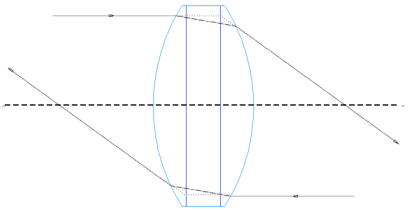
Figure 14.18 Exploded sequence with all tubes (left), and exploded sequence after deleting two tubes (right)

11 Define a **TRANSMIT** coating, and assign this coating to all six **Optical** interfaces in place of the default **BARE** coating.

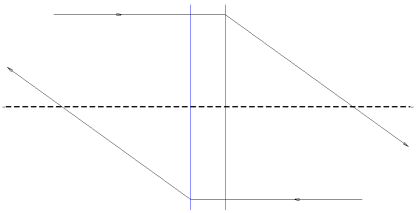
12 Trace 101 rays again with no graphics, and compare ray-trace speeds.

Principal Planes and Matrix Optics

Consider a thick, singlet lens like the one shown below. If we trace a ray from left to right near the top edge of the lens, a ray that is parallel to the axis, it will refract as shown at the top of the figure. Imagine, however, that we cannot see what has happened inside the glass. We might conclude that the ray was bent only once, as shown by the dotted ray path. We can find the location of this alternative refraction point by extending the entering ray along its original direction, and doing the same for the exiting ray. The intersection of these two dotted lines defines the location of one principal plane. The other is found by tracing another marginal ray from right to left, as shown at the bottom of the figure.



Since the actual ray path and the alternative path involving the principal planes are indistinguishable once the ray exits the lens, we can replace the actual lens by the principal planes, as shown in the second figure. This is what the **IDEAL** *lens represents when modeling a perfect lens.*



The input ray vectors are linearly related to the output ray vectors by the 2 x 2 ABCD matrix. Many optics texts describe how to derive the ABCD matrix for general optical components and systems. (for example, see Optics, by Eugene Hecht.) You have already seen the matrix for an ideal lens, (A, B, C, D) = (1, 0, -1/f, 1). The matrix for an afocal system would be (A, B, C, D) = (1/m, 0, 0, m), where m is the magnification of the system.

From a computational perspective, this is often a convenient approach when tracing rays. For example, ASAP can rapidly evaluate the effect of ideal "lenses". This is computationally much more efficient than the usual procedure of finding ray intersections, calculating surface normals, and applying Snell's law surface by surface. Further, the combined effect of a chain of optical components is just the matrix product of the individual matrices, combined in the appropriate order.

The results we obtain using the **IDEAL** *command will be "ideal" rather than realistic, however. The non-physical nature of this procedure is also apparent in this example. For a thick lens of the sort shown here, the so-called "principal planes" are, in reality, curved surfaces. This fact introduces optical aberrations in real lenses, which are completely ignored in this treatment.*

APPENDIX

14A

SCRIPTS FOR CHAPTER 14

The following ASAP scripts are referenced in Chapter 14A, “Lens Entities”.

Script 14-1

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY RUNNING THE BUILDER EXAMPLE ABOVE.
!!THE COMMAND SCRIPT INCLUDES THE COMMANDS TO SEE THE RESULT IN THE 3D VIEWER.
!!THE INITIAL COMMANDS TO RESET THE SYSTEM ARE INCLUDED AT THE BEGINNING.
SYSTEM NEW
RESET
COATING PROPERTIES
  0 .95  'TRANSMIT'
LENS
  SINGLET Z 0.0 5 16.9 SCHOTT_SK4 RD 44.55 -436.6
OBJECT 'L1'
  INTERFACE COATING TRANSMIT AIR AIR
PLOT FACETS 5 5
$VIEW
```

Script 14-2

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY RUNNING THE BUILDER EXAMPLE ABOVE.
!!THE COMMAND SCRIPT INCLUDES THE COMMANDS TO SEE THE RESULT IN THE 3D VIEWER.
!!THE INITIAL COMMANDS TO RESET THE SYSTEM ARE INCLUDED AT THE BEGINNING.
SYSTEM NEW
RESET
COATING PROPERTIES
```

```
  0 .95  'TRANSMIT'
LENS
  SINGLET Z 0.0 4 12.5 SCHOTT_BK7 FL 100 1
OBJECT 'LENS1'
  INTERFACE COATING TRANSMIT AIR AIR
PLOT FACETS 5 5
$VIEW
```

Script 14-3

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY RUNNING THE BUILDER EXAMPLE ABOVE.
!!THE COMMAND SCRIPT INCLUDES THE COMMANDS TO SEE THE RESULT IN THE 3D VIEWER.
!!THE INITIAL COMMANDS TO RESET THE SYSTEM ARE INCLUDED AT THE BEGINNING.
SYSTEM NEW
RESET
COATING PROPERTIES
  0 .95  'TRANSMIT'
UNITS CENTIMETERS
WAVELENGTHS 486.1 587.6 656.3 NANOMETERS
LENS
  DOUBLET Z 0.0 0.4 1.0 SCHOTT_BAK2 SCHOTT_SF2 8.0 0.3
OBJECT 'DL1'
  INTERFACE COATING TRANSMIT AIR AIR
PLOT FACETS 5 5
$VIEW
```

Script 14-4

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY RUNNING THE BUILDER EXAMPLE ABOVE.
!!THE COMMAND SCRIPT INCLUDES THE COMMANDS TO SEE THE RESULT IN THE 3D VIEWER.
!!THE INITIAL COMMANDS TO RESET THE SYSTEM ARE INCLUDED AT THE BEGINNING.
SYSTEM NEW
RESET
UNITS MILLIMETERS
WAVELENGTHS 468.1 NANOMETERS

!! ENT OBJECT;DOUBLET Z 0 0.4 1 SCHOTT_BK7 SCHOTT_BK6 8 0

ENT OBJECT;OPTICAL Z 0.000000 7.987073 0.000000 ELLIPSE 1.000000 1.000000 0.000000
  REDEFINE COLOR light blue 3
  INTERFACE COATING BARE "SCHOTT_BK7" "VACUUM|AIR"
ENT OBJECT;OPTICAL Z 0.3600000 -2.758996 0.000000 ELLIPSE 1.000000 1.000000 0.000000
  REDEFINE COLOR orange 4
  INTERFACE COATING BARE "SCHOTT_BK6" "SCHOTT_BK7"
ENT OBJECT;TUBE Z 0.1723964 1.000000 1.000000 0.6284843E-01 1.000000 1.000000 0 0
  REDEFINE COLOR light blue 3
ENT OBJECT;OPTICAL Z 0.4000000 -7.987073 0.000000 ELLIPSE 1.000000 1.000000 0.000000
  REDEFINE COLOR orange 4
  INTERFACE COATING BARE "VACUUM|AIR" "SCHOTT_BK6"
ENT OBJECT;TUBE Z 0.3371516 1.000000 1.000000 0.1723964 1.000000 1.000000 0 0
  REDEFINE COLOR orange 4
PLOT FACETS 5 9
$VIEW
```

Script 14-5

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY RUNNING THE BUILDER EXAMPLE ABOVE.
!!THE COMMAND SCRIPT INCLUDES THE COMMANDS TO SEE THE RESULT IN THE 3D VIEWER.
!!THE INITIAL COMMANDS TO RESET THE SYSTEM ARE INCLUDED AT THE BEGINNING.
SYSTEM NEW
RESET
LENS
  IDEAL Z 0.0 1.0 0.4 1.0 ; 1.0 0.0 -1/4 1.0 0.0
OBJECT 'L1'
PLOT FACETS 7 7
$VIEW
```

CHAPTER

15

REPOSITIONING GEOMETRY AND RAYS

Until now, we have used the geometry we have created without adjusting either its position or orientation. The three lenses in the Cooke triplet example were all aligned perpendicular to one of the global coordinate axes, and centered on it as well. The same was true for components of the Cassegrain telescope. The commands for creating the objects had enough flexibility to create all these objects exactly where we needed them. But what should we do when we need to introduce something as simple as a folding flat oriented at 45 degrees to our chosen coordinate system? ASAP obviously needs to provide tools to rotate and shift geometry to any arbitrary location and orientation.

We have emphasized from the beginning of the Primer that rays always have a set of spatial coordinates associated with them. They can be translated and rotated in much the same way as geometry.

All the sources we have introduced so far were also defined in a plane perpendicular to one of the global coordinate axes. Sources, like geometry, may also need to be shifted and rotated into position. We have emphasized from the beginning of the *Primer* that rays always have a set of spatial coordinates associated with them, so it should not be surprising that they can be translated and rotated in much the same way as geometry.

In a modern, graphical computing environment, it is tempting to hope for a simple point-and-click or drag-and-drop method for positioning geometry. Unfortunately, optical models normally require greater precision than this would allow. In the end, we will enter numbers, whether angles or distances, expressed to the desired level of accuracy. Fortunately, the commands are simple and intuitive, offering the flexibility to translate objects in either relative or absolute terms.

At the end of this chapter, we will also demonstrate how to use variables and expressions in the ASAP Builder. This feature will allow you to parameterize all aspects of your model, and let ASAP do the necessary mathematical calculations as you adjust these parameters.

Placing Geometry—Absolute Translations

As a first example, let us assume that we have used the **PLANE** command to create a plane mirror at the origin of the global coordinate system and perpendicular to the *z* axis. In some cases you may know the *absolute* global coordinates at which this object should be placed. The best command to use in this situation is **PLACE**, which can be found on the **Builder** menu at **Geometry> Edge Modifiers> Place** and select **Global** from the Option column. If we want to place the mirror at the point (1.0, 2.0, 1.0), the Builder file would look like Figure 15.1.

*	Type	Name	Option	Axis	Thickness	Aperture	Semiwidth	Semiwidth	Obs Ratio	Offset X
OBJ	Plane	MIRROR_1	Axis	Z	0	Ellipse	0.5	0.5		
MOD	Local	Long Form	-1.5	1.5	-1.5	1.5	-0.5	3.5	Z	0
MOD	Interface	Coating	COATING	REFLECT	Air	Air				
MOD	Place	Global	Axis	Z	0					

Figure 15.1 Using the **PLANE** command in the **Builder** to create a plane mirror, and the **PLACE** command to place the mirror at a point.

The **PLACE** command is an object modifier, just like **INTERFACE**, and **BOUNDS**. It can also be applied to entities (building blocks that make up objects, or that are used to bound objects). You should place it under the object (or entity) definition in the Builder with the other modifiers that apply to it. The order does not matter in most cases.

See Chapter 15 Appendix, "Script 15-1" on page 321.

But there is a question we must ask: what point on the plane is, in fact, placed at the point (1.0, 2.0, 1.0)? As Figure 15.2 on page 297 shows, it is the center of the circular plane in this example. This location is usually the case for simple, surface-based objects, and the **PLACE** command is most useful under these circumstances. The situation can become more complicated, however. (See the sidebar, "Determining Reference Points" on page 318.) When the reference point is not so easily predicted or convenient, you may find relative translations easier to use.

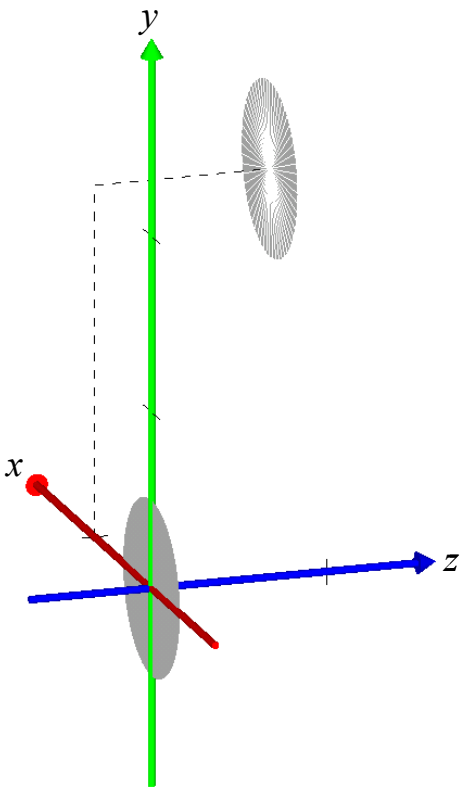


Figure 15.2 The **PLACE** command is an object or entity modifier. It translates the reference point of the object or entity to the specified absolute global coordinates. In this case, the circular plane, for which the original reference point was the origin, was placed at the point (1, 2, 1).

Shifting Geometry—Relative Translations

Although ASAP always works in a single, global coordinate system, it is often more convenient for the user to specify relative translations. You may, for example, create a simple rectangular mirror five units along the z axis, and then shift it relative to this location. ASAP includes the **SHIFT** command to accomplish this. Like **PLACE**, **INTERFACE**, and **BOUNDS**, **SHIFT** is an object modifier. Like **PLACE**, it can also be used to modify entities that do not have object status. ASAP offers three versions of this command in the Builder. Figure 15.3a shows exactly the same result in the 3D Viewer for the three examples— Figure 15.3b, Figure 15.3c, and Figure 15.3d on page 299.

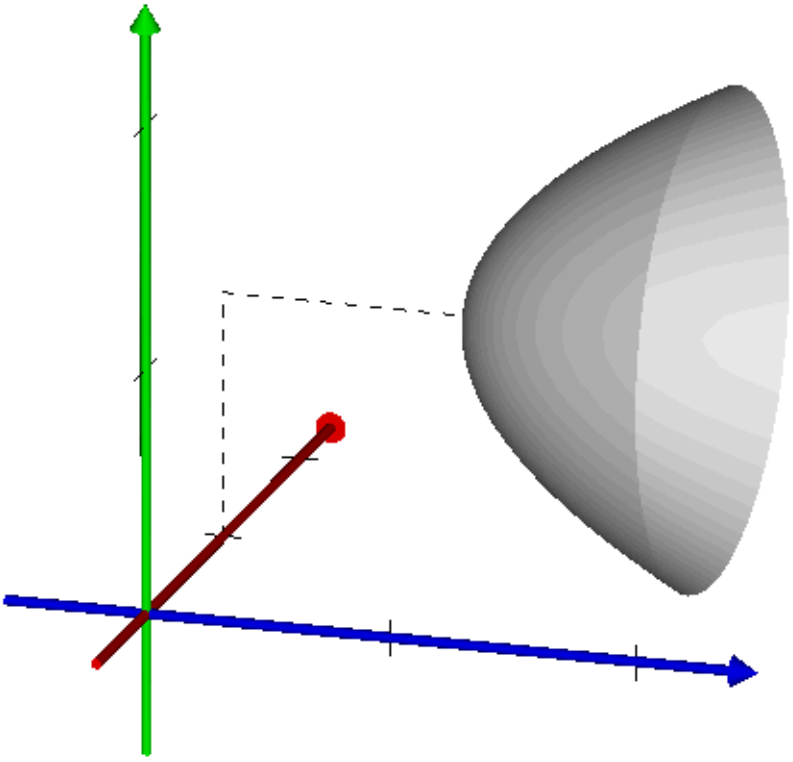


Figure 15.3a Results of the three **Builder** examples of the **SHIFT** command

Object Control> Object Modifiers> Shift, and keep the default of **One Axis—** Shift the object along just one of the three global axes, *x*, *y*, or *z*. The shift, in this case, is relative to the initial location of the object or entity, and is specified in system units. See Figure 15.3b.

*	Type	Option	Axis	Distance	List			
OBJ	Parabolic	Mirror1	Z	0	0.25	Ellipse	1	1
MOD	Interface	Coating	COATING	Reflect	Air	Air		
MOD	Shift	One Axis	X	1				
MOD	Shift	One Axis	Y	1				
MOD	Shift	One Axis	Z	1				

Figure 15.3b Example of **Modifiers> Shift** with **One Axis**

See Chapter 15 Appendix, "Script 15-2" on page 322.

Object Control> Object Modifiers> Shift and select **All Axes—**Shift by a specified amount in system units along each of the global coordinates *x*, *y*,

and *z*, again relative to the current location of the object or entity. See Figure 15.3c.

*	Type	Option	X	Y	Z	List		
OBJ	Parabolic	Mirror2	Z	0	0.25	Ellipse	1	1
MOD	Interface	Coating	COATING	REFLECT	Air	Air		
MOD	Shift	All Axes	1	1	1			

Figure 15.3c Example of **System> Object Control> Shift** with **All Axes**

See Chapter 15 Appendix, "Script 15-3" on page 322.

Object Control> Object Modifiers> Shift and select **Vector—**Shift by a specified amount in system units along an arbitrary direction vector. You specify the direction vector using either direction cosines or spherical coordinates. See Figure 15.3d.

*	Type	Option	Distance	Cmd	Vector A	Vector B	Vector C	List
OBJ	Parabolic	Mirror3	Z	0	0.25	Ellipse	1	1
MOD	Interface	Coating	COATING	REFLECT	Air	Air		
MOD	Shift	Vector	0	ALONG	0.577	0.577	0.577	

Figure 15.3d Example of **System> Object Control> Shift** with **Vector**

See Chapter 15 Appendix, "Script 15-4" on page 323.

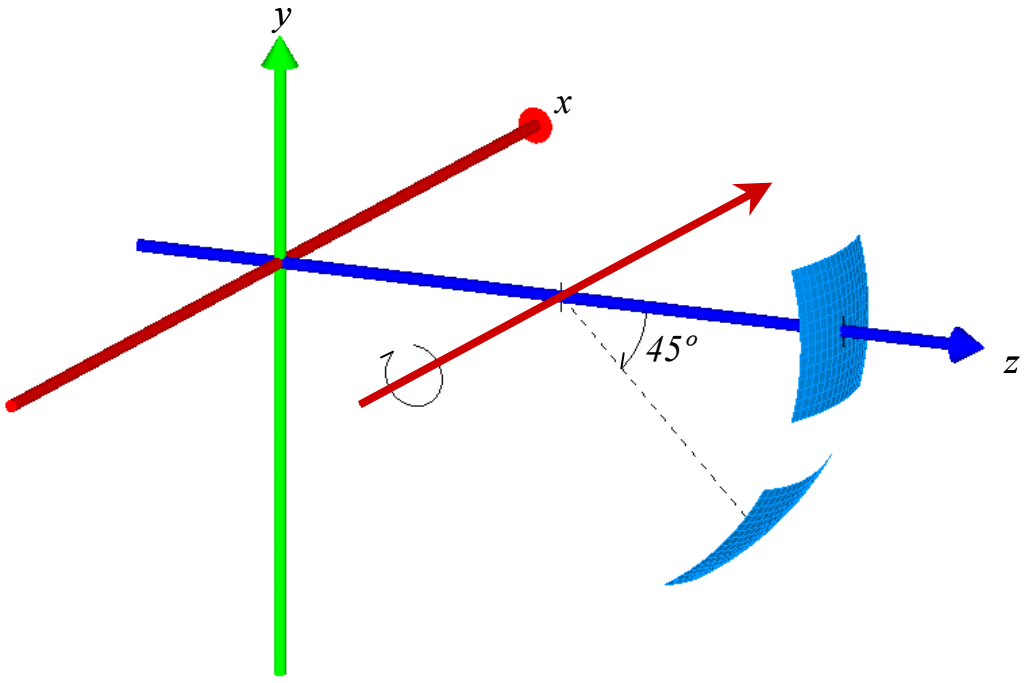
Note: You may have more than one `SHIFT` command modifying a single object or entity. The **All Axes** or **Vector** syntax could have been accomplished using three separate shifts along one coordinate with exactly the same results. There is no ray-trace speed penalty for specifying multiple shifts or multiple rotations. When the system database entry is made for the object, a single matrix is formed, which specifies the combination of all translation and rotation modifiers that apply to that object. (See the `MATRIX` command in the on-line Help for details.) Always choose the method that is most convenient and logical for you.

Rotating Geometry

Now that you know how to translate objects or entities to any arbitrary place in the system, you need to learn only how to rotate them to have the flexibility to create geometry in any orientation. This rotation is accomplished with the `ROTATE` command, which comes in two forms. Select the one that is most convenient and logical in your situation.

Object Control> Object Modifiers> Rotate and select **Axis—**Rotate the object around one of the global coordinate axes, or around a line that is parallel to that axis but displaced. If you specify the *x* axis, for example, ASAP rotates the object around a line parallel to the *x* axis, but passing through a point *y, z* that you specify. The rotation angle is given in degrees.

The sign convention for the angle uses the right-hand rule. This convention is also illustrated in Figure 15.4 below and continued on page 300. If no offset point is given, the displacement point defaults to the object or entity's reference point.

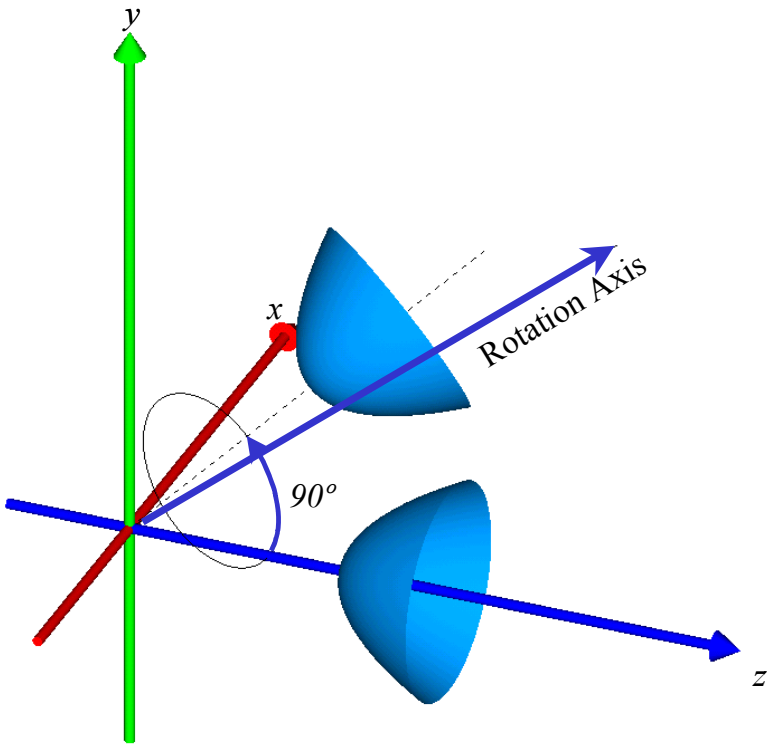


*	Type	Cmd	Axis	Degrees	Y	Z	List	
OBJ	Spherical	MIRROR_1	Z	2	-1	Rectangle	3	3
MOD	Interface	Coating	COATING	REFLECT	Air	Air		
MOD	Rotate	Axis	X	45	0.0	1.0		

Figure 15.4 The **Axis** version of the **ROTATE** command allows you to rotate an object about a line that is parallel to one of the global axes. In this example, the mirror is rotated about a line parallel to the *x* axis, which passes through the point y = 0, z = 1. The sign convention for rotation follows the right-hand rule: if you point the thumb of your right hand down the rotation axis, your curled fingers point in the direction of positive rotation.

See Chapter 15 Appendix, "Script 15-5" on page 323.

Object Modifiers> Rotate and select **Vector—**Rotate around any arbitrary direction vector passing through any arbitrary point. This version of the ROTATE command is shown in Figure 15.5 on page 301. You must specify a direction vector representing the rotation axis, and a point through which this vector passes. You might choose this version if you prefer to perform compound rotations in the coordinate system of the object. In this case, you need to know or be able to calculate the current direction of the local axis before each subsequent rotation.



*	Type	Cmd	Degrees	Cmd	A	B	C	X	Y	Z
OBJ	Parabolic	MIRROR_1	Z	2	0.25	Ellipse	1	1		
MOD	Interface	Coating	COATING	REFLECT	Air	Air				
MOD	Rotate	Vector	90	About	0	0.65279	0.76604	0	0	0

Figure 15.5 The **Vector** version of the ROTATE command is more general than **Axis**. It allows you to rotate around any arbitrary axis, specified by a direction vector and a point through which it passes. In this example, we show a fairly complicated rotation: the rotation axis is in the *y*-*z* plane, at 40 degrees to the *z* axis, and passing through the global origin. We have rotated a parabolic reflector through 90 degrees about this axis.

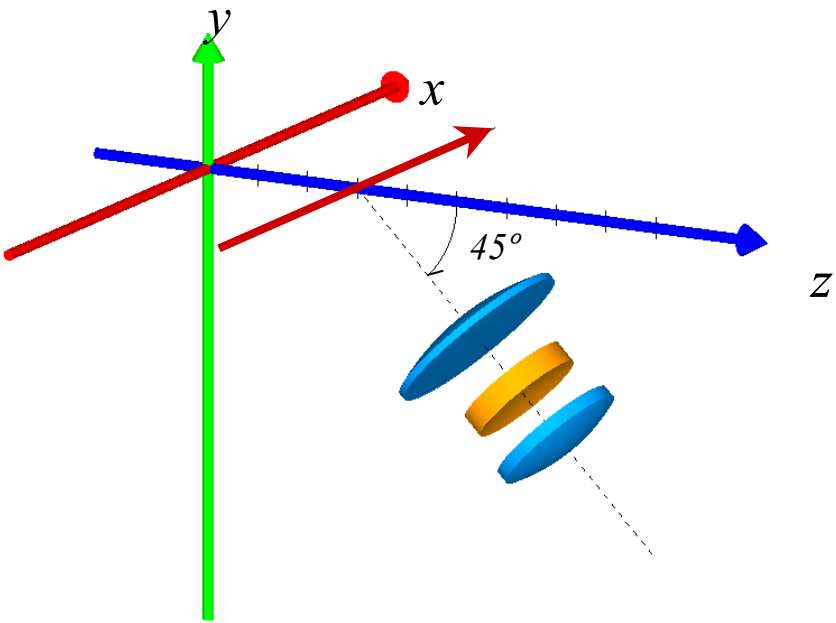
See Chapter 15 Appendix, "Script 15-6" on page 324.

Note: As with shifts, you can also apply multiple rotations to a single object or entity. Be sure to remember, however, that ASAP is performing all rotations in the global coordinate system. There is no set of local axes that rotate with the object. As noted above, there is no speed penalty when multiple translations or rotations are applied to a single object. In the end, all translations and rotations are reduced to a single matrix in the system database.

Grouping Geometry

What if we want to place, shift, or rotate an optical element that consists of several independent surfaces? Our Cooke triplet, for example, consists of nine surfaces. Do we need to insert nine separate `SHIFT` and `ROTATE` commands to accomplish this? The answer is "no". ASAP provides the `GROUP` command, which allows us to group several objects, and then move them together. You will find it on the **Builder** menu, under **Object Control> Object Modifiers> Group**.

The use of the `GROUP` command is illustrated in Figure 15.6, below and on page 303.



*	Type	Objects	Cmd								
MOD	Interface	Coating	COATING	Trans	"SCHOTT_SK4"	"VACUUM|AIR"					
OBJ	Optical	COOKE.Optical.9	Axis	Z	29.95000	-32.67000	0.000000				
MOD	Redefine						Color	Light Blue			
MOD	Interface	Coating	COATING	Trans	"VACUUM|AIR"	"SCHOTT_SK4"					
OBJ	Tube	COOKE.Tube.10	Axis	Z	27.62665	12.100000	12.100000	25.24186	12.100000	12.100000	0
MOD	Redefine						Color	Light Blue			
MOD	Group	COOKE.?									
MOD	Shift	One Axis	Z	60							
MOD	Rotate	Axis	X	45	0	30					

Figure 15.6 The GROUP command allows us to temporarily group a set of objects so that they can be shifted and rotated together. The Builder file shows the last few lines of the Cooke triplet definitions after exploding the lens sequence (see the exercise at the end of Chapter 14). We have used the **"?"** wildcard to group all objects with names beginning the "Cooke". The group was shifted 60 mm along the z axis, and then rotated about a line parallel to the x axis, passing through y = 0, z = 30.

See Chapter 15 Appendix, "Script 15-7" on page 324.

All elements of the Cooke triplet have been grouped, and then shifted and rotated together. You can tell ASAP which objects are members of the group by listing them in the **Objects** cell on the **Group** line. You have several options for doing this:

- List by name. You can include the name of each object in the group, separated by spaces. For example, **"Front Surface" "Back Surface" "Tube" "Detector"** will make a group of the four named objects. Note that quotation marks were used in names that include spaces.
- List by name using wild cards. In the example shown in page 303, we have simply written **Cooke.?** The **"?"** symbol is a "wildcard". Any object having a name beginning with "Cooke"—and hence part of the Cooke "assembly"—is included in the group.
- List by relative reference. If you type **.1 .3 .4**, for example, ASAP groups the first, third, and fourth objects defined, counting upward from the position of the GROUP command in the Builder.
- List by absolute reference. If you type **1 2 3**, ASAP groups the first, second, and third objects, counting from the top of the Builder file.

Any shifting and rotating that you perform after the GROUP command is applied to all objects in the group. The reference point used by the PLACE or ROTATE command defaults is the reference point of the first object in the group.

Note: While it is tempting to try to use GROUP to assign other object modifiers (like INTERFACE) to a group of objects, the command is

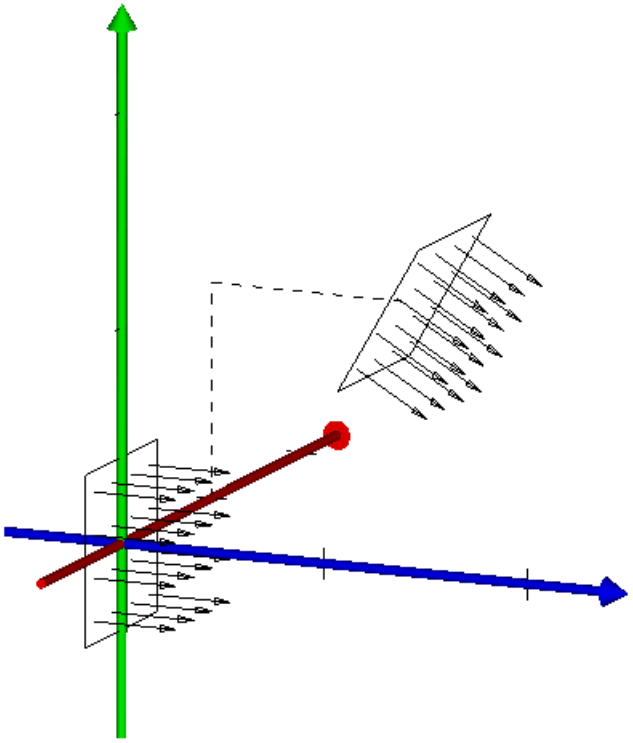
intended only to *temporarily* group objects so that they can be moved as a unit.

Shifting and Rotating Rays

Rays are shifted and rotated in exactly the same way as geometry, using the same commands.

Rays are shifted and rotated in exactly the same way as geometry, using the same commands. As you have seen in Chapter 7, rays are represented within ASAP as points in space with associated direction vectors. It is a simple matter for ASAP to apply shifts and rotations to these rays as if they were geometric entities resembling arrows. Figure 15.7 shows a grid of parallel rays initially located at the origin, which was shifted and rotated.

There is one subtle difference in using the **SHIFT** command with rays compared to using it with geometry. After rays are created, the **SHIFT** command can be applied to the rays only one time. When it is used, the **SHIFT** applies to all currently selected rays. If a second **SHIFT** is required, the **SELECT** command must be used to re-select the rays to be shifted.



*	Type	Cmd	Axis	Degrees	Y	Z	List			
ENT	Grid	Rect	Z	0	-0.4	0.4	-0.4	0.4	4	4
MOD	Source	Direction		0	0	1				
MOD	Shift	All Axes	1	1	1					
MOD	Rotate	Axis	X	30	1	1				

Figure 15.7 Rays are rotated and shifted in exactly the same way as geometry. In this figure, a rectangular edge is shown around the periphery of the ray grid to help show that the plane in which the rays were initially defined was shifted and rotated only to modify the location and direction of the rays In this case, the rays were shifted one unit along each axis, and then rotated around a line parallel to the x axis passing through a point at the center of the grid (y = 1, z = 1).

See Chapter 15 Appendix, "Script 15-8" on page 327.

Note: When using the PLACE command, or using the default reference point in either version of the ROTATE command, you need to pay careful attention to the reference point of the source. ASAP generally uses the centroid (average position) of all rays as the default reference, rather than any point related to the initial location. This is different from the behavior of PLACE and ROTATE when they are applied to geometry. To avoid confusion, we strongly recommend that you use SHIFT in preference to PLACE when translating sources, and always specify a reference point when rotating.

Moving Rays

When rays are translated using the MOVE command, system geometry is ignored.

ASAP makes a distinction between placing or shifting rays, and *moving* rays. ASAP includes the MOVE command, which allows us to slide rays along their direction vectors like beads on a string. We encountered this in Chapter 11, when we discussed finding the best focus of a set of rays. In that situation, we moved the rays forward or backward along their final trajectories to find the place in space where they were as tightly grouped as possible. The MOVE command performs a similar function, allowing us to reposition rays at various places along their trajectories. But how is moving rays different from tracing rays? The answer is simple: when rays are translated using the MOVE command, system geometry is ignored. The MOVE command causes rays to pass directly through all intervening geometry without reflecting, refracting, or having their fluxes altered in any way.

When will we need to move rays along their direction vectors? We have already seen one obvious example when we defined a grid of rays using SOURCE POSITION. You may recall from Chapter 8 that this command is used to create a set of rays in a plane that has directions *as though* the rays had come from a specified point in space. The MOVE command allows us to have them *actually* come from that point. Note, however, that the directions of the rays never change, only their positions.

You can move rays at any time. You may wish to reposition rays in this way when they are initially defined, part way through a ray trace, or as part of your analysis. For this reason the MOVE command is available to you in a variety of places in the ASAP graphical user interface. It can be found on the **Rays** menu (**Rays> Move Rays**), the **Analysis** menu (**Analysis> Move Rays**), and also on the **Builder** menu (**Ray Control> Ray Modifiers> Move**). In all cases, the basic ASAP command is MOVE. Figure 15.8 on page 306 shows the choices available by way of either the ASAP menu-generated dialog boxes or on the Builder submenus.

*	Type	Cmd	Axis	Amount						
EMT	Grid	Elliptic	Z	-10	-1	1	-1	1	11	11
MOD	Source	Direction		0	0	1				
CMD	Move	By	Z							
		By								
		To								
		To Foci								
		To Point								
		To Plane								
		To Sphere								
		To Surface								
		Parabasals								

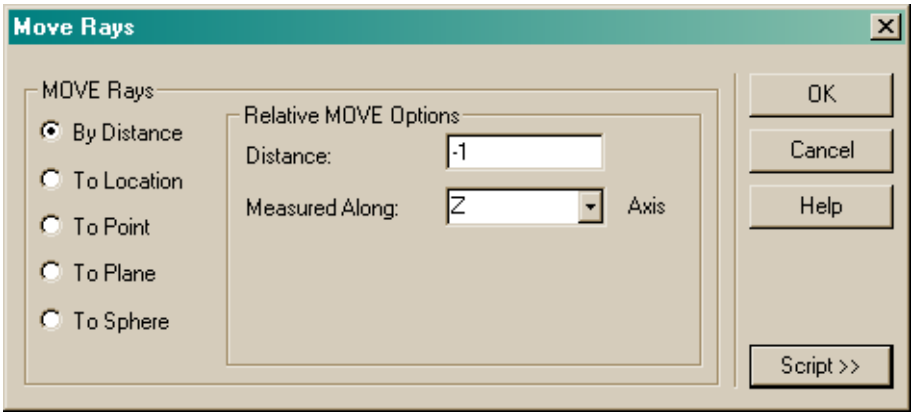
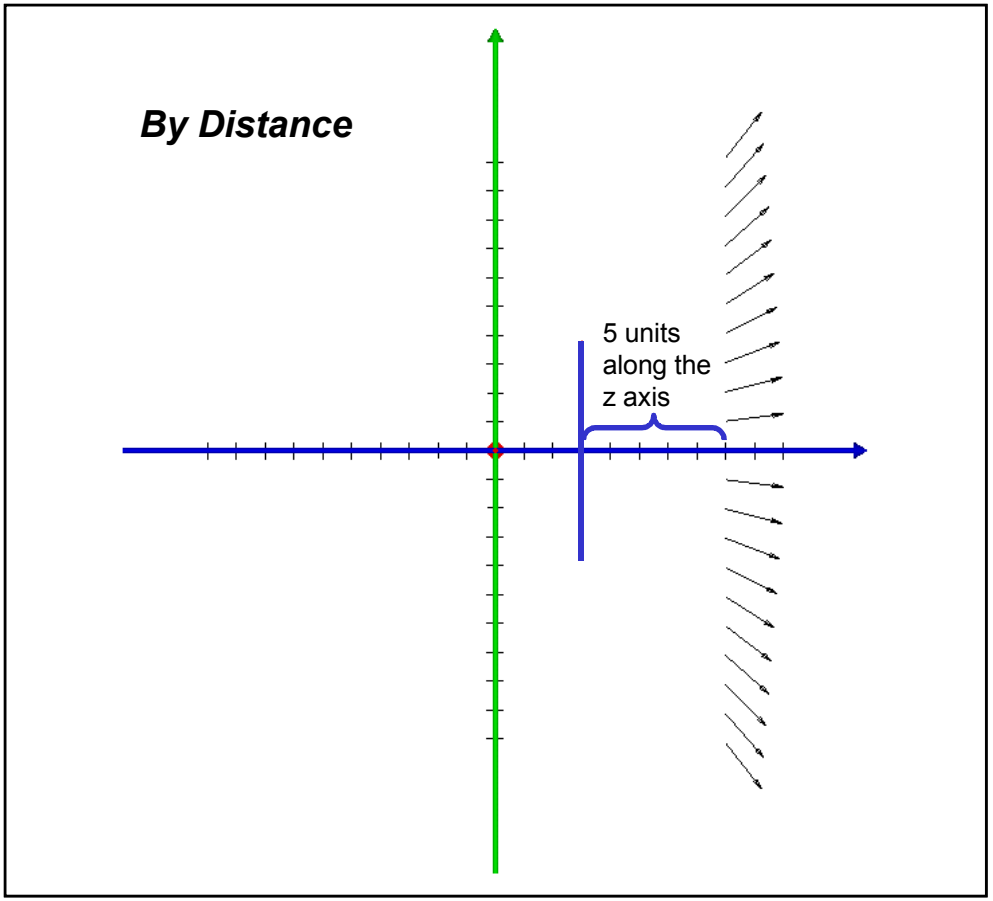


Figure 15.8 Rays can be moved along their direction vectors either by entering the MOVE command into the Builder (top) or using the dialog box (bottom). The **Move Rays** dialog box is available on both the **Rays** and **Analysis** menus.

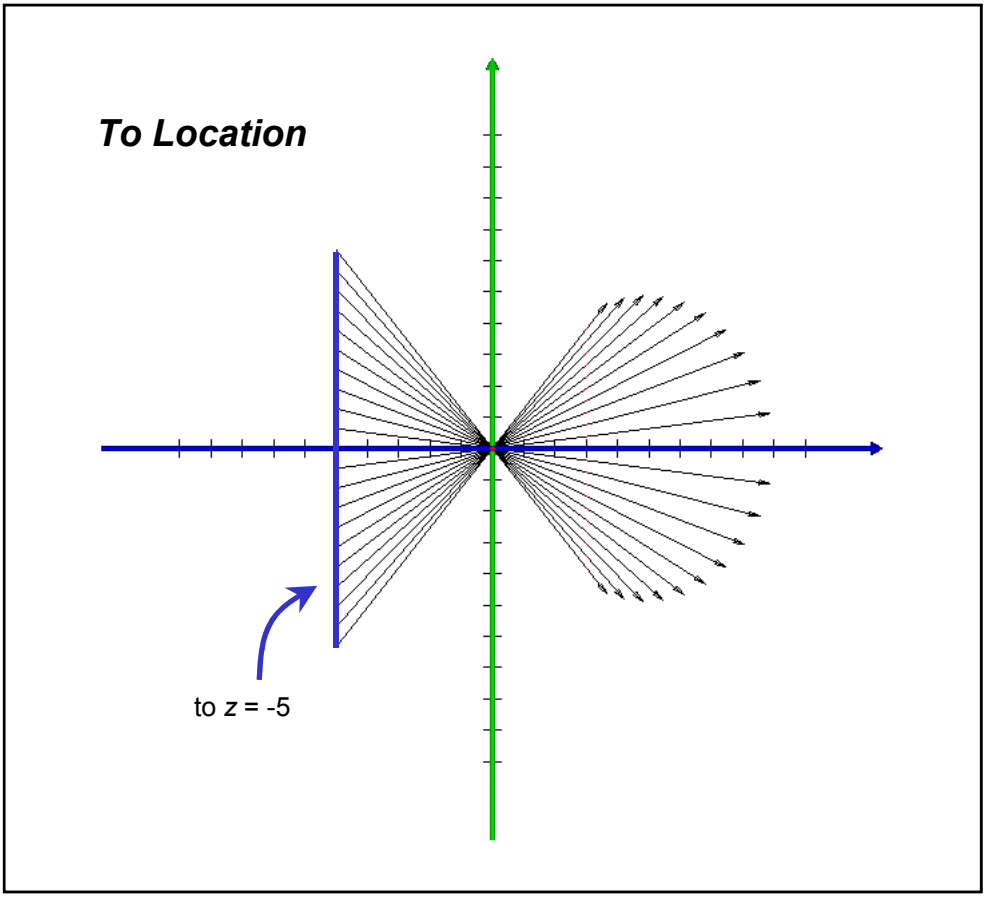
Figure 15.9a to Figure 15.9e show an example of each option.



*	Type	Cmd	Axis	Amount						
EMT	Grid	Elliptic	Z	-10	-1	1	-1	1	11	11
MOD	Source	Direction		0	0	1				
CMD	Move	By	Z	-1						

Figure 15.9a Move by Distance. Each ray is moved from its initial position by a specified distance along one of the global coordinate axes. A version of `MOVE BY` moves a specified distance along the direction vector.

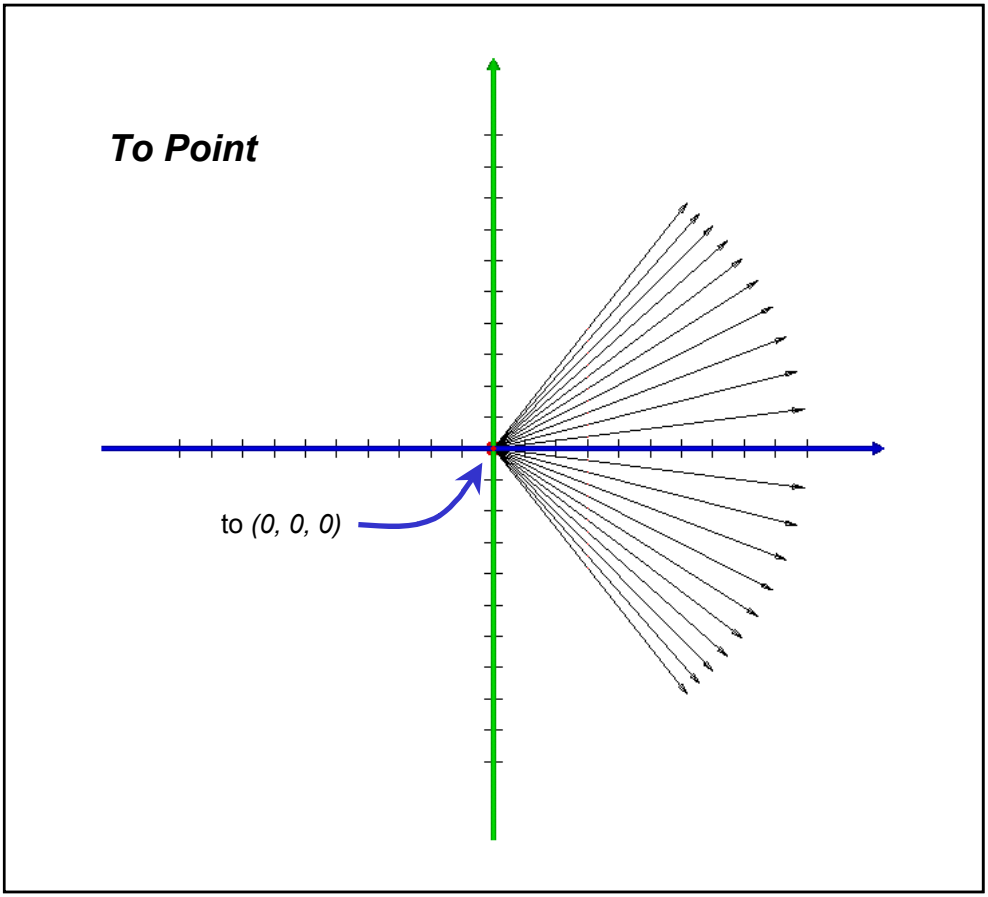
See Chapter 15 Appendix, "Script 15-9" on page 327.



*	Type	Cmd	Axis	Amount							
EMT	Grid	Rect	Z	3	-4	4	-4	4	1	12	10
MOD	Source	Position	0	0	0						
CMD	Move	To	Z	-5							

Figure 15.9b Move to Location. Each ray is moved from its initial position in a plane located at z = 3 until one of its coordinates reaches a specified value (until z = -5 in this example).

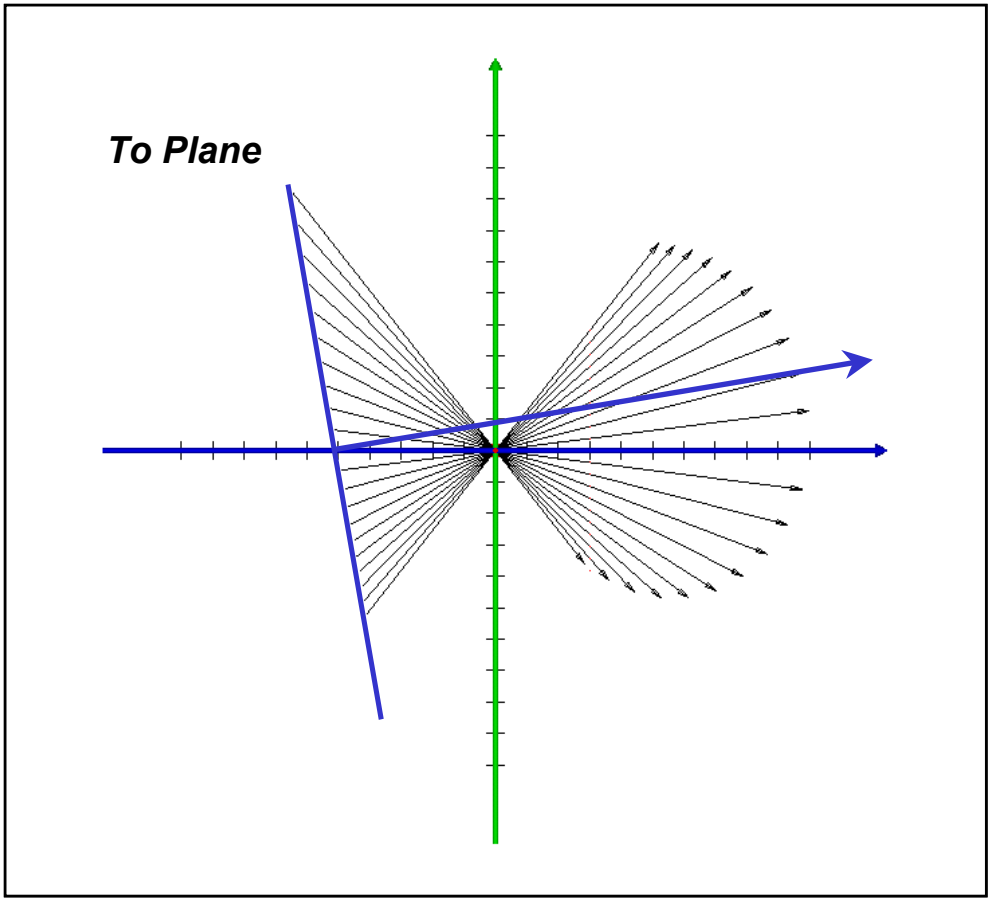
See Chapter 15 Appendix, "Script 15-10" on page 328.



*	Type	Cmd	Point X	Point Y	Point Z						
EMT	Grid	Rect	Z	3	-4	4	-4	4	1	12	10
MOD	Source	Position	0	0	0						
CMD	Move	To Point	0	0	0						

Figure 15.9c Move to Point. Each ray is moved from its initial position to a specified point, or as close to that point as it is able to come. In this example, the rays were moved to (0, 0, 0), which is also the point specified by the `SOURCE POSITION` command. If the point specified does not lie along a given ray's direction vector, ASAP moves the ray to the point of closest approach.

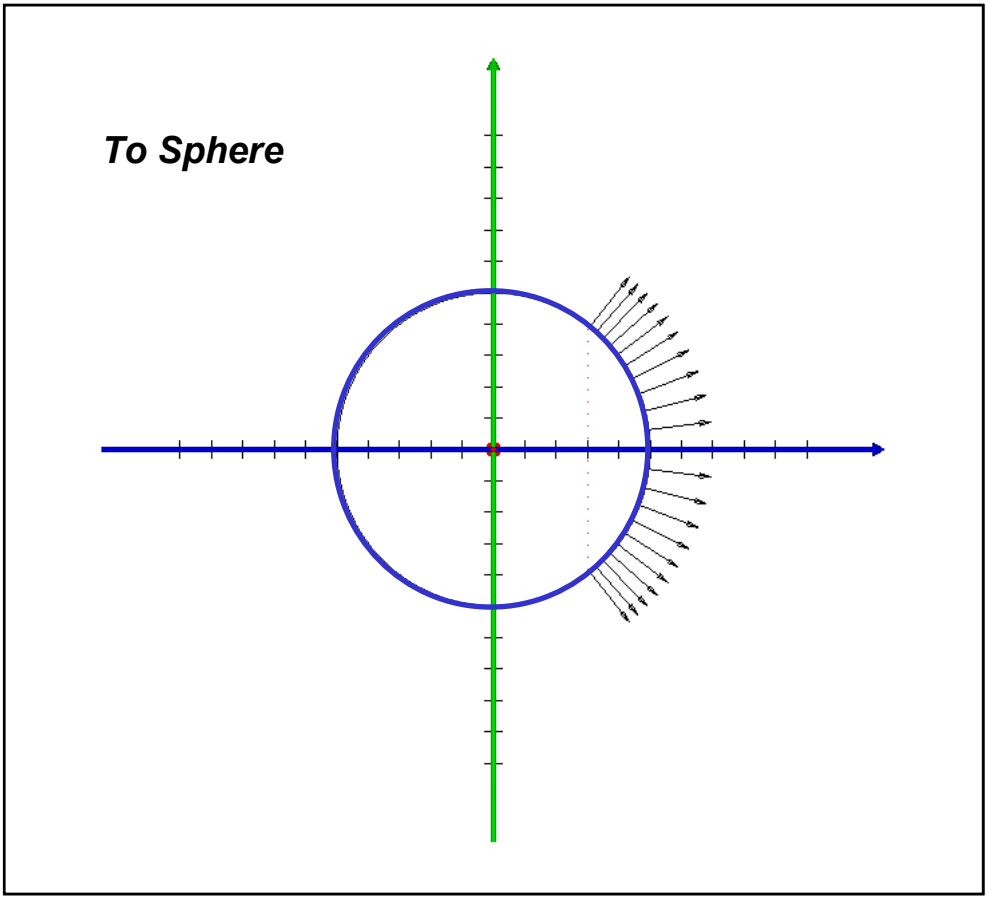
See Chapter 15 Appendix, "Script 15-11" on page 328.



*	Type	Cmd	Point X	Point Y	Point Z	Vector A	Vector B	Vector C			
EMT	Grid	Rect	Z	3	-4	4	-4	4	1	12	10
MOD	Source	Position	0	0	0						
CMD	Move	To Plane	0	0	-5	0	0.17365	0.98481			

Figure 15.9d Move to Plane. Each ray is moved from its initial position to an arbitrary plane. The plane is specified by a single point on its surface, and a vector normal to the plane. In this example, the plane passes through the point (0, 0, -5), and is tipped at 10 degrees relative to the z axis.

See Chapter 15 Appendix, "Script 15-12" on page 328.



*	Type	Cmd	Radius	Point X	Point Y	Point Z					
EMT	Grid	Rect	Z	3	-4	4	-4	4	1	12	10
MOD	Source	Position	0	0	0						
CMD	Move	To Sphere	5	0	0	0					

Figure 15.9e Move to Sphere. Each ray is moved from its initial position to the surface of a sphere. The sphere is specified by its radius and the coordinates of its center.

See Chapter 15 Appendix, "Script 15-13" on page 329.

Using Variables and Expressions in the Builder

The ASAP Builder lets you define and use variables. Your file can also include mathematical expressions that do basic calculations for you. Any cell in the Builder that expects a number also accepts the name of a variable or a mathematical expression. While this topic is not specific to repositioning objects and rays, we will take this opportunity to introduce it. As your models

become increasingly complex, you may find it useful to begin introducing programming elements to your systems. Variables and expressions are one example of this.

By changing the value of one or more variables, you can efficiently modify all elements of a system that depend on those parameters.

The use of variables and expressions allows you to parameterize your models. By changing the value of one or more variables, you can efficiently modify all elements of a system that depend on those parameters. An example Builder file that uses variables and expressions is shown in Figure 15.10. We have explicitly calculated the sagittal height (sag) of the spherical surfaces of the lens, allowing us to begin and end a lens edge at the appropriate places. This value depends only on the size (diameter) of the optical surface, and its radius of curvature. It would probably have been easier to use the **BOUNDS** command for this; we are using this method only to illustrate the use of variables and expressions.

*	Type	Name	Value									
SYS	System	New										
	Reset											
SYS	Units	CentiMeter										
CMD	Media	Glass	Media		1.5							
CMD	Coating	Transmit		Properties		0	1					
CMD	Coating	Absorb		Properties		0	0					
SYS	Variable	A	2.5									
SYS	Variable	R	10									
SYS	Variable	SAG	R-SQRT(R^2-A^2)									
SYS	Variable	ANGLE	5									
OBJ	Spherical	Lens.Front	Z	0.0	(R)	Ellipse	(A)	(A)				
MOD	Interface	Coating	Coating	Transmit	Air	Glass						
OBJ	Spherical	Lens.Back	Z	1.0	-(R)	Ellipse	(A)	(A)				
MOD	Interface	Coating	Coating	Transmit	Air	Glass						
OBJ	Tube	Lens.Edge	Axis	Z	(SAG)	(A)	(A)	1-SAG	(A)	(A)	0.0	0.0
MOD	Interface	Coating	Coating	Absorb	Air	Glass						
EMT	Grid	Elliptic	Z	-1	-(A)	(A)	-(A)	(A)	11	11		
MOD	Source	Direction		0	SIN[ANGLE]	COS[ANGLE]						

Figure 15.10 Example of how the Builder can use variables and expressions in place of numerical arguments. We have defined a symmetric singlet lens of half-diameter **A** and radius of curvature **R**. We have defined another variable named **SAG** that represents the sagital height for a spherical surface with these parameters. We can now change the lens by changing **A** and **R**. The variable **ANGLE** is used to control the direction of the rays in a grid.

See Chapter 15 Appendix, "Script 15-14" on page 330.

In the Builder file shown in Figure 15.10, we have defined four variables using **Setup> Variable** on the **Builder** menu. We have assigned explicit numerical values to the variables named **A** and **R**. The initial value of the variable named **SAG** is calculated using an expression. In this case, the expression is the sag

equation (see the sidebar, “The Mathematics of Conics” on page 118 in Chapter 6). We have also defined a variable named **ANGLE**, which we can use to vary the direction of the rays in a grid. ASAP calculates the sine and cosine of this angle to form the direction vector in the `SOURCE DIRECTION` command.

Basic Rules for Variables and Expressions

We will discuss variables and expressions in more detail in Chapter 23, “Running Command Scripts”, after we have introduced the ASAP command language. Until then, here are a few basic rules to get you started using these features in the Builder.

- Variables can have names up to 32 characters long, but the name must begin with a letter.
- Variable names by themselves should be enclosed in parentheses when used in place of a number in a Builder cell. The parentheses signal to ASAP to substitute the value of the variable in place of the number that the program is expecting. The parentheses are not necessary if the variable is part of a larger expression.
- An expression is a collection of mathematical operators, functions, variables, and numbers that can be evaluated to a single value.
- ASAP expressions cannot contain spaces.
- For a complete list of ASAP mathematical operators, search for *mathematical operators* in the on-line Help, or see Chapter 23 of this *Primer*.
- For trigonometric functions, use square brackets if the argument is expressed in degrees, or parentheses if the argument is in radians.

Summary

In this chapter, we have discussed the following new commands.

ASAP Commands	Menu	Description
`PLACE...`	Builder: **Object Control> Object Modifiers> Place**	Translate an object, entity, or source to a specified place (absolute)
`SHIFT...`	Builder: **...Object Modifiers> Shift** select **One Axis**	Translate an object, entity, or source in one global coordinate (relative)
`SHIFT...`	Builder: **.... Object Modifiers> Shift> All Axes**	Translate an object, entity, or source in all global coordinates (relative)
`SHIFT...`	Builder: **...Object Modifiers> Shift,** select **Direction**	Translate an object, entity, or source by a specified amount in an arbitrary direction (relative)
`ROTATE... ABOUT`	Builder: **...Object Modifiers> Rotate> Axis**	Rotate an object, entity or source through a specified angle around a line parallel to one of the coordinate axes
`ROTATE...`	Builder: **...Object Modifiers> Rotate,** select **Vector**	Rotate an object, entity or source through a specified angle around an arbitrary axis
`GROUP`	Builder: **...Object Modifiers> Group**	Temporarily group a set of objects for translating or rotating them as a unit

ASAP Commands	Menu	Description
MOVE MOVE TO MOVE BY	Main Menu: **Rays> Move Rays** **Analysis> Move Rays** Builder: **Object Control> Object Modifiers,** select **Move**	Move rays along their direction vectors
<Name>=Value	Builder: **Setup> Variable**	Define and initialize a variable

Most ASAP geometry definitions are too constraining to create an entity, object or source in any arbitrary place and in any arbitrary orientation. ASAP provides us with the PLACE, SHIFT, and ROTATE commands to overcome this limitation. Here are a few important points worth emphasizing in summary:

- The same commands and syntax are used for shifting geometry or rays, because ASAP rays are points in space with associated directions. They can be shifted and rotated just like physical “arrows”.
- The PLACE and SHIFT commands perform the same function (translating objects, entities, and rays). PLACE uses an absolute positioning of a reference point. To use it correctly, you must pay careful attention to the reference point of the entity, object, or ray set. SHIFT performs a relative offset.
- All ASAP rotations, even compound rotations, are performed relative to the global coordinate system. There are no local coordinates or axes.
- The GROUP command can be used to group a set of objects prior to translating and rotating them. The grouping is only temporary, and applies only to subsequent SHIFT, PLACE, or ROTATE commands that immediately follow GROUP.
- ASAP allows us to substitute a variable or expression into any Builder cell where a number is expected.

Exercise 8: Best Focus of a Singlet Lens

This exercise combines many of the concepts introduced so far in this *Primer*. We have included chapter references to help you, if some review is necessary.

1. Use lens entities to define a singlet lens with a focal length of 25 centimeters made of Schott SK-4 glass. The lens should have a 5 cm diameter, and a thickness of 1 centimeter. (Review “Singlet Lenses” on page 269 in Chapter 14.) Let the bending factor be a variable named **BEND**, set initially to 0.
2. Explode the lens to produce the two `OPTICAL` surfaces and the `TUBE` representing the edge (Review “Converting Lenses to Surface-based Objects” on page 275 in Chapter 14). Change the coating from the default **BARE** to a 100% transmitting coating. The tube should be absorbing. (See “Interface Command” on page 72 in Chapter 4).
3. Create an aperture stop a short distance in front of the lens, with a rectangular outer boundary of 8 cm on each side, and a circular inner boundary of 4 cm diameter using edges. (Review “Extruding Edges” on page 238 in Chapter 13.)
4. Use a surface-based plane (Review “Plane Surface” on page 115 in Chapter 6) to create a circular detector 35 cm from the front of the lens, having a diameter of 10 cm.
5. Verify your geometry using both the **Preview** function in the Builder, and by running the Builder and making a profile (Review “Profiles Command” on page 89 in Chapter 5). Place the cursor in the **Plot** window to verify the distances (Review “Plot Viewer Window” on page 91 in Chapter 5).
6. Set up two collimated rectangular grid sources, 6 cm on each side. They should be created a few centimeters in front of the aperture stop. Use 11×11 rays for both sources initially. The first source should have all its rays parallel to the *z* axis (Review “Rectangular Grid of Parallel Rays” on page 137 in Chapter 8). Rotate the second source 5 degrees around a line parallel to the *x* axis, and passing through the center of the grids.
7. Verify the sources using **Rays> Graphics> Plot rays 2D**, showing a facetted plot of the geometry in the same view. (Review “Combining Ray and Geometry Graphics” on page 164 in Chapter 9.)

Figure 15.11 Exercise 8: Faceted plot of the geometry

8. Trace the rays on top of a profile of the geometry. (Review “Trace Dialog Box” on page 181 in Chapter 10.)
9. Isolate only those rays that are on the detector after the ray trace. (Review “Choosing Rays for Analysis” on page 204, Step 1 in Chapter 11.) Then

make a spot diagram of the rays on the detector. (Review “Spot Diagrams” on page 208).

10 Isolate only those rays that are on the detector, and are members of the first (on axis) source. (Review “Choosing Rays for Analysis” on page 204, Step 2 in Chapter 11.) Find the best focus for these rays, and move them to this position. (Review “Finding Best Focus” on page 211 in Chapter 11.)

11 Select the rays only from the second source that are on the detector. Use the `MOVE` command to slide the rays from the off-axis source to the same *z* position that is the best focus for the on-axis rays. Make a spot diagram of only these rays after they have been moved.

12 Increase the number of rays to 101 × 101 and repeat the ray trace without showing any graphics.

13 Repeat the analysis steps above to make the two spot diagrams again.

14 Experiment with the bending factor variable to see how the shape of the lens affects the quality of the two images.

Determining Reference Points

Every ASAP geometrical entity has a reference point. It is this point that is moved to the absolute coordinate specified by the **PLACE** *command. For surface-based objects, the reference point is usually at the intersection of the surface and the coordinate axis. Even in the case of our Cassegrain primary mirror, which had a hole where it should have intersected the z axis, the nonexistent vertex of the mirror is still the reference point for that object. This is still convenient when you are performing absolute translations.*

If you had created an object with an offset, as is allowed by many of the surface definitions, the axis intersection is still the reference point, rather than the center of the object. If you plan to translate your geometry using **PLACE**, *it is probably better to avoid using the offset option that is built into the command.*

The reference point for the **TUBE** *command can also be determined in several ways. If you specify the axis as +X, +Y, or +Z, the reference point is at the positive end of the tube, along its axis of symmetry. If you specify -X, -Y, or -Z, the reference point is at the negative end of the tube, still along the axis. If no sign appears on the Axis specification, the reference point is in the middle of the tube. Note once more that the reference point is under no circumstances actually a point on the tube! Always consult the on-line Help for a given surface-based entity when using the* **PLACE** *command.*

It can be even more difficult to guess the reference point of an edge-based object or entity. We often construct edge-based objects by extruding or sweeping a curve. What reference point would ASAP select under such circumstances? When we create an object by sweeping a line, for example, the reference point is actually the first point specified in the **LINE** *definition. For a curve specified using the* **POINTS** *command, the default reference point is actually the origin of the global coordinate system, even if the entity is defined far from this point.*

Under circumstances where you cannot be sure of the reference point, it is best to create the un-shifted version of the object by running the Builder and consulting the object's properties directly in the system database. (See the sidebar: "System Database" on page 108 in Chapter 5.) After creating the object, select the **Objects** *tab in the* **ASAP Workspace** *window, expand the drawing tree, and find the object in question. When you right-click the object's name and select* **Properties** *from the pop-up menu, ASAP prints the database entry for that object in the* **Command Output** *window. The reference point is the first point listed in the entity definition. See Figure 15.12 on page 319.*

```
CURVEDG    2: Pts=   16 (RECTANGL) X  0.00000     Y  0.00000     Z  1.50000
   16 segments of degree  1
   16 total with     0 breaks
 Sweep distance & direction   0.000    0.00000        0.00000        1.00000
                                       0.00000        0.00000        1.50000
 This entity used by objects    1
 Points and Connection Factors:
    x          y          q          x          y          q
 2.000000  0.000000   1.00000  2.000000  0.828427   1.00000
 2.000000  2.000000   1.00000  0.828427  2.000000   1.00000
 0.000000  2.000000   1.00000 -0.828427  2.000000   1.00000
-2.000000  2.000000   1.00000 -2.000000  0.828427   1.00000
-2.000000  0.000000   1.00000 -2.000000 -0.828427   1.00000
-2.000000 -1.999999   1.00000 -0.828428 -2.000000   1.00000
 0.000000 -2.000000   1.00000  0.828427 -2.000000   1.00000
 1.999999 -2.000000   1.00000  2.000000 -0.828428   1.00000

OBJECT    1: APERTURE
 Physical Surface     1 (RECTANGL) to     2 (RECTANGL)
    1 segments of degree  1
    1 total with     1 breaks
       Box X -4.080     4.080    Y -4.080     4.080    Z  1.420     1.580
          width=  8.1600        width=  8.1600        width= 0.16000
       Bounding sphere  5.77055        0.00000        0.00000        1.50000
```

Figure 15.12 Print out in the **Command Output** window of database entry for a specified object

APPENDIX

15A

SCRIPTS FOR CHAPTER 15

The following ASAP script is referenced in Chapter 15, “Repositioning Geometry and Rays”.

Script 15-1

```
!!THE COMMAND SCRIPT INCLUDES THE COMMANDS TO SEE THE RESULT IN THE 3D VIEWER.
!!THE INITIAL COMMANDS TO RESET THE SYSTEM ARE INCLUDED AT THE BEGINNING.
SYSTEM NEW
RESET
COATING PROPERTIES
  1 0  'REFLECT'
SURFACE
  PLANE Z 0.0 ELLIPSE 0.5 0.5
OBJECT 'MIRROR_1'
  INTERFACE COATING REFLECT AIR AIR
  PLACE 1.0 2.0 1.0
PLOT FACETS 5 5
$VIEW
```

Script 15-2

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY RUNNING THE BUILDER
!!TO DEMONSTRATE THE SHIFT 1 COORDINATE EXAMPLE.
!!3 SEPARATE SHIFTS SHOW THE SHIFT IN EACH COORDINATE DIRECTION.
!!THE INITIAL COMMANDS TO RESET THE SYSTEM ARE INCLUDED.
!!THE SCRIPT INCLUDES COMMANDS TO SEE THE RESULT IN THE 3D VIEWER.
SYSTEM NEW
RESET
COATING PROPERTIES
  1 0  'REFLECT'
SURFACE
  OPTICAL Z 0.0  2*0.25  -1 ELLIPSE 1.0
OBJECT 'MIRROR1'
  INTERFACE COATING REFLECT AIR AIR
  SHIFT X 1
  SHIFT Y 1
  SHIFT Z 1
PLOT FACETS 5 5
$VIEW
```

Script 15-3

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY RUNNING THE BUILDER
!!TO DEMONSTRATE THE SHIFT ALL COORDINATES EXAMPLE.
!!THE INITIAL COMMANDS TO RESET THE SYSTEM ARE INCLUDED.
!!THE SCRIPT INCLUDES COMMANDS TO SEE THE RESULT IN THE 3D VIEWER.
SYSTEM NEW
RESET
COATING PROPERTIES
  1 0  'REFLECT'
SURFACE
  OPTICAL Z 0.0  2*0.25  -1 ELLIPSE 1.0
OBJECT 'MIRROR2'
  INTERFACE COATING REFLECT AIR AIR
  SHIFT 1 1 1
PLOT FACETS 5 5
$VIEW
```

Script 15-4

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY RUNNING THE BUILDER
!!TO DEMONSTRATE THE SHIFT ALONG DIRECTION EXAMPLE.
!!THE INITIAL COMMANDS TO RESET THE SYSTEM ARE INCLUDED.
!!THE SCRIPT INCLUDES COMMANDS TO SEE THE RESULT IN THE 3D VIEWER.
SYSTEM NEW
RESET
COATING PROPERTIES
  1 0  'REFLECT'
SURFACE
  OPTICAL Z 0.0  2*0.25  -1 ELLIPSE 1.0
OBJECT 'MIRROR3'
  INTERFACE COATING REFLECT AIR AIR
  SHIFT 1.732 ALONG .577 .577 .577
PLOT FACETS 5 5
$VIEW
```

Script 15-5

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY RUNNING THE BUILDER
!!TO DEMONSTRATE THE ROTATE ABOUT AXIS EXAMPLE.
!!THE INITIAL COMMANDS TO RESET THE SYSTEM ARE INCLUDED.
!!THE SCRIPT INCLUDES COMMANDS TO SEE THE RESULT IN THE
!!3D VIEWER FROM BEFORE AND AFTER THE ROTATION.
SYSTEM NEW
RESET
COATING PROPERTIES
  1 0  'REFLECT'
SURFACE
  OPTICAL Z 2 -1  0 RECTANGLE 0.25
OBJECT 'MIRROR_1'
  INTERFACE COATING REFLECT AIR AIR
  REDEFINE COLOR 3
PLOT FACETS 5 5   !!PLOT BEFORE ROTATION
OBJECT .1
  ROTATE X 45 0 1
PLOT FACETS 5 5   !!PLOT AFTER ROTATION
$VIEW
```

Script 15-6

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY RUNNING THE BUILDER
!!TO DEMONSTRATE THE ROTATE ABOUT POINT EXAMPLE.
!!THE INITIAL COMMANDS TO RESET THE SYSTEM ARE INCLUDED.
!!THE SCRIPT INCLUDES COMMANDS TO SEE THE RESULT IN THE
!!3D VIEWER FROM BEFORE AND AFTER THE ROTATION.
SYSTEM NEW
RESET
COATING PROPERTIES
  1 0  'REFLECT'
SURFACE
  OPTICAL Z 2  2*0.25  -1 ELLIPSE 1.0
OBJECT 'MIRROR_1'
  INTERFACE COATING REFLECT AIR AIR
  REDEFINE COLOR 3
PLOT FACETS 5 5   !!PLOT BEFORE ROTATION
OBJECT .1
  ROTATE 90 ABOUT 0 .64279 .76604 0 0 0
PLOT FACETS 5 5   !!PLOT AFTER ROTATION
$VIEW
```

Script 15-7

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY RUNNING THE BUILDER
!!TO DEMONSTRATE THE USE OF THE GROUP COMMAND
!!TO SHIFT AND ROTATE A GROUP OF OBJECTS.
!!THE INITIAL COMMANDS TO RESET THE SYSTEM ARE INCLUDED.
!!THE SCRIPT INCLUDES COMMANDS TO SEE THE RESULTS IN THE
!!3D VIEWER FROM BEFORE AND AFTER THE SHIFT AND ROTATE.
SYSTEM NEW
RESET
UNITS MILLIMETERS
WAVELENGTHS 550 NANOMETERS
COATING PROPERTIES
  0 0  'ABSORB'
  0 1  'TRANS'
!! ENT OBJECT;SEQUENCE RADII 'COOKE' 'COOKE'
!! 0 0 0      0 0 1    16.9   44.55  0 0 SCHOTT_SK4
```

```
!!      5.00              16.9 -436.6   0 0 AIR
!!     10.31              10.5  -38.61  0 0 SCHOTT_F15
!!      1.60              10.5   42.62  0 0 AIR
!!      8.04              12.1  250.97  0 0 SCHOTT_SK4
!!      5.00              12.1  -32.67  0 0 AIR
!!
SURFACE
  OPTICAL Z 0.0 44.55 0.0 ELLIPSE 16.9 16.9 0.0
OBJECT 'COOKE.OPTICAL.0'
  REDEFINE COLOR light blue      3
  INTERFACE COATING TRANS SCHOTT_SK4 VACUUM|AIR

SURFACE
  OPTICAL Z 5.0 -436.6 0.0 ELLIPSE 16.9 16.9 0.0
OBJECT 'COOKE.OPTICAL.1'
  REDEFINE COLOR light blue      3
  INTERFACE COATING TRANS VACUUM|AIR SCHOTT_SK4

SURFACE
  TUBE Z 4.672793 16.9 16.9 3.329950 16.9 16.9 0 0
OBJECT 'COOKE.TUBE.2'
  REDEFINE COLOR light blue      3

SURFACE
  OPTICAL Z 15.31 -38.61 0.0 ELLIPSE 10.5 10.5 0.0
OBJECT 'COOKE.OPTICAL.3'
  REDEFINE COLOR orange          4
  INTERFACE COATING TRANS SCHOTT_F15 VACUUM|AIR

SURFACE
  OPTICAL Z 16.91 42.62 0.0 ELLIPSE 10.5 10.5 0.0
OBJECT 'COOKE.OPTICAL.5'
  REDEFINE COLOR orange          4
  INTERFACE COATING TRANS VACUUM|AIR SCHOTT_F15

SURFACE
  TUBE Z 18.22365 10.5 10.5 13.85484 10.5 10.5 0 0
OBJECT 'COOKE.TUBE.6'
  REDEFINE COLOR orange          4
```

```
SURFACE
  OPTICAL Z 24.95 250.97 0.0 ELLIPSE 12.1 12.1 0.000
OBJECT 'COOKE.OPTICAL.7'
  REDEFINE COLOR light blue      3
  INTERFACE COATING TRANS SCHOTT_SK4 VACUUM|AIR

SURFACE
  OPTICAL Z 29.95 -32.67 0.0 ELLIPSE 12.1 12.1 0.000
OBJECT 'COOKE.OPTICAL.9'
  REDEFINE COLOR light blue      3
  INTERFACE COATING TRANS VACUUM|AIR SCHOTT_SK4

SURFACE
  TUBE Z 27.62665 12.1 12.1 25.24186 12.1 12.1 0 0
OBJECT 'COOKE.TUBE.10'
  REDEFINE COLOR light blue      3

PLOT FACETS 5 5    !!PLOT BEFORE SHIFT AND ROTATION
GROUP COOKE.?
  SHIFT Z 60
  ROTATE X 45 0 30
GROUP -10
SCALE FROM MILLIMETERS
 PLOT FACETS 5 5   !!PLOT AFTER SHIFT AND ROTATION
 $VIEW
```

Script 15-8

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY RUNNING THE BUILDER
!!TO SHIFT AND ROTATE A SET OF RAYS.
!!THE INITIAL COMMANDS TO RESET THE SYSTEM ARE INCLUDED.
!!THE SCRIPT INCLUDES COMMANDS TO SEE THE RESULTS IN THE
!!3D VIEWER FROM BEFORE AND AFTER THE SHIFT AND ROTATE.
SYSTEM NEW
RESET
ARROWS 10
GRID RECT Z 0 -0.4 0.4 -0.4 0.4 4 4
  SOURCE DIR 0 0 1
PLOT RAYS 1      !!PLOT BEFORE SHIFT AND ROTATION
  SHIFT 1 1 1
  ROTATE X 30 1 1
PLOT RAYS 1      !!PLOT AFTER SHIFT AND ROTATION
$VIEW
```

Script 15-9

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY RUNNING THE BUILDER
!!TO MOVE A SET OF RAYS WITH MOVE BY DISTANCE.
!!THE INITIAL COMMANDS TO RESET THE SYSTEM ARE INCLUDED.
SYSTEM NEW
RESET
GRID RECT Z 3 -4 4 -4 4 1 12
  MOVE BY Z 5
RETURN
```

Script 15-10

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY RUNNING THE BUILDER
!!TO MOVE A SET OF RAYS WITH MOVE TO LOCATION.
!!THE INITIAL COMMANDS TO RESET THE SYSTEM ARE INCLUDED.
SYSTEM NEW
RESET
GRID RECT Z 3 -4 4 -4 4 1 12
  SOURCE POSITION 0 0 0
  MOVE TO Z -5
RETURN
```

Script 15-11

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY RUNNING THE BUILDER
!!TO MOVE A SET OF RAYS WITH MOVE TO POINT.
!!THE INITIAL COMMANDS TO RESET THE SYSTEM ARE INCLUDED.
SYSTEM NEW
RESET
GRID RECT Z 3 -4 4 -4 4 1 12
  SOURCE POSITION 0 0 0
  MOVE TO POINT 0 0 0
RETURN
```

Script 15-12

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY RUNNING THE BUILDER
!!TO MOVE A SET OF RAYS WITH MOVE TO PLANE.
!!THE INITIAL COMMANDS TO RESET THE SYSTEM ARE INCLUDED.
SYSTEM NEW
RESET
GRID RECT Z 3 -4 4 -4 4 1 12
  SOURCE POSITION 0 0 0
  MOVE TO PLANE 0 0 -5 0 0.17365 0.98481
RETURN
```

Script 15-13

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY RUNNING THE BUILDER
!!TO MOVE A SET OF RAYS WITH MOVE TO SPHERE.
!!THE INITIAL COMMANDS TO RESET THE SYSTEM ARE INCLUDED.
SYSTEM NEW
RESET
GRID RECT Z 3 -4 4 -4 4 1 12
  SOURCE POSITION 0 0 0
  MOVE TO SPHERE 5 0 0 0
RETURN
```

Script 15-14

```
!!THIS IS THE COMMAND SCRIPT PRODUCED BY RUNNING THE BUILDER
!!TO DEMONSTRATE THE USE OF VARIABLES AND EXPRESSIONS.
!!THE INITIAL COMMANDS TO RESET THE SYSTEM ARE INCLUDED.
SYSTEM NEW
RESET
UNITS CENTIMETERS
MEDIA
  1.5  'GLASS'
COATING PROPERTIES
  0  1  'TRANSMIT'
  0  0  'ABSORB'
A=2.5
R=10
SAG=R-SQRT(R^2-A^2)
ANGLE=5
SURFACE
  OPTICAL Z 0.0 (R)  0 ELLIPSE (A) (A)
OBJECT 'LENS.FRONT'
  INTERFACE COATING TRANSMIT AIR GLASS
SURFACE
  OPTICAL Z 1.0 -(R)  0 ELLIPSE (A) (A)
OBJECT 'LENS.BACK'
  INTERFACE COATING TRANSMIT AIR GLASS
SURFACE
  TUBE Z (SAG) (A) (A) 1-SAG (A) (A) 0.0 0.0
OBJECT 'LENS.EDGE'
  INTERFACE COATING ABSORB AIR GLASS
GRID ELLIPTIC Z -1 -(A) (A) -(A) (A) 11 11
  SOURCE DIR 0 SIN[ANGLE] COS[ANGLE]
RETURN
```

CHAPTER

16

EXTENDED SOURCES

The grid sources we have used up until now are suitable only for modeling point sources, either at infinity (`SOURCE DIRECTION`) or at a finite distance (`SOURCE POSITION` and `SOURCE FOCUS`). These sources are useful for simulating imaging systems, but less so for illumination problems. A typical illumination source is extended. This extension can be thought of as a set of closely spaced point sources that extend over a finite area or volume. An example of this is an incandescent filament of the type used in flashlights or automotive lamps. High-intensity discharge sources (arc sources) are also extended, emitting out of a plasma discharge volume.

All the extended source models in ASAP are characterized by randomness. ASAP automatically generates a random set of rays either on a surface, or within a volume. The set is randomly placed within the constraints of the surface or volume specified. The ray directions are also random, although constraints may be placed on the statistical distribution of those directions.

One of the strengths of ASAP lies in its ability to make realistic extended-source models, along with its power to trace a statistically significant number of rays in a reasonable amount of time.

One of the strengths of ASAP lies in its ability to make realistic extended-source models, along with its power to trace a statistically significant number of rays in a reasonable amount of time. As you will see in this chapter, ASAP allows us to turn any object or entity into a surface emitter. This flexibility, along with an intensity distribution appropriate to the physics of incandescent surfaces, allows us to make excellent models of a large class of extended sources. We can also modify the flux of random ray sets according to their positions and directions. In this way, we can alter basic sources to make them match laboratory measurements or behavior reported by the light-source manufacturer. The details of this process, known as ray apodization, is beyond the scope of this *Primer*, but is briefly described along with other advanced source modeling techniques, at the end of this chapter. These methods are discussed in more detail in the ASAP Technical Guide, *Sources*.

Emitting Disk and Rectangle

All the extended source models in ASAP are characterized by randomness.

You will find all the sources discussed in the chapter on the **Builder** menu, under **Rays> Emitting**. The **Disk** and **Rect** options both produce a random set of rays, distributed over the surface of an imaginary plane (see Figure 16.1). This randomness is a key element in these and all other ASAP emitters (see the sidebar, "Random Number Generator" on page 350). The `EMITTING DISK` and `EMITTING RECTANGLE` commands differ only in the shape of the outline of the plane. These commands might be appropriate sources for simulating a flat, hot, glowing metal like a ribbon filament, or the surface of the die in a light-emitting diode. The actual ASAP commands generated by the Builder for the `EMITTING DISK` and `EMITTING RECTANGLE` are shown in the script referenced below.

See Chapter 16 Appendix, "Script 16-1" on page 351.

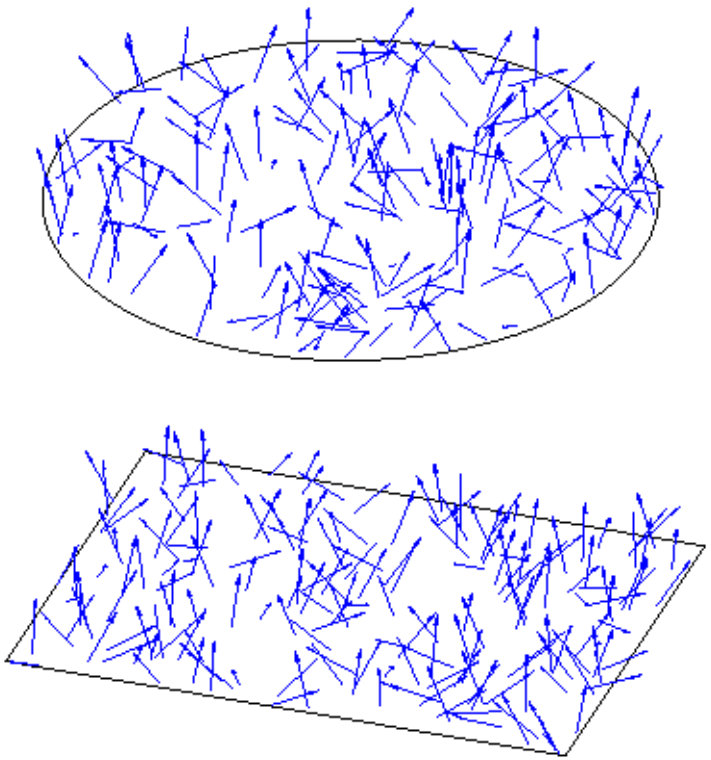
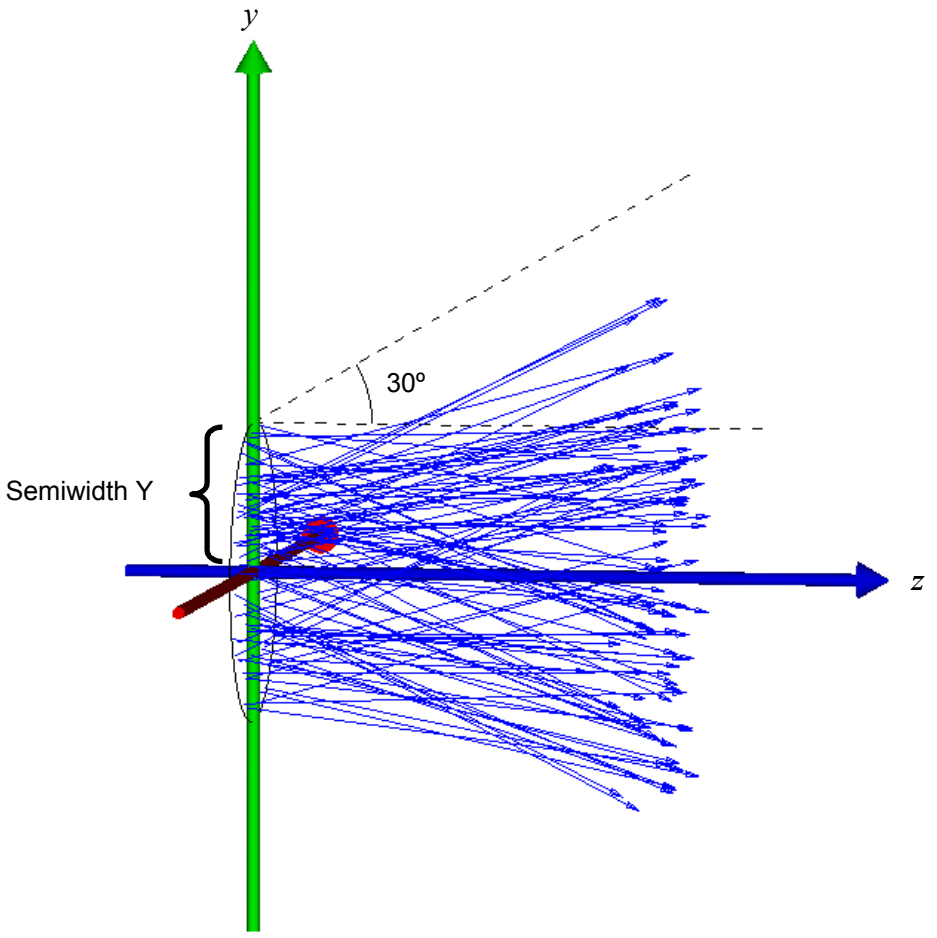


Figure 16.1 The ASAP `EMITTING DISK` and `EMITTING RECTANGLE` commands create a set of random rays located on a plane with elliptical (top) or rectangular (bottom) boundaries. Both the location and direction of the rays are random within certain constraints. Elliptical and rectangular edges were added to these illustrations to help clarify the limits of the two sources. No actual ASAP geometry is created by the commands.

An example of an emitting disk is shown in Figure 16.2.



*	Type	Option	Axis	Location	Semiwidth X	Semiwidth Y	Ray Count	Half Angle	Half Angle'	RECT Flag	ISO Flag
EMT	Emitting	Disk	Z	0	1	1	100	30	30		
			X								
			Y								
			Z								
			-X								
			-Y								
			-Z								

Figure 16.2 The ASAP `EMITTING DISK` command creates a set of random rays in an elliptical plane. This example sets the cone angle in both axes to 30 degrees so that no rays are created with directions beyond this limit.

The Builder line shown in Figure 16.2 was created using **Rays> Emitting**, and select **Disk** from the drop-down box in the Option column. The first parameter is

the **Axis** of the emitter. Your choices are limited to those listed in the drop-down box that appears when you double-click this cell. The axis defines the plane in which the rays are initially created. It must be perpendicular to one of the global coordinate axes, though you can always rotate the source, as described in Chapter 15, to obtain other orientations. Note also that you can pick either the positive or negative direction for this axis. Since we are simulating a surface emitter, the ray directions fill one hemisphere, at most. The sign determines whether this hemisphere is in the positive or negative direction along the chosen axis. The rays in our example were created in the $+z$ direction.

Three cells, **Location**, **Semiwidth X**, and **Semiwidth Y** determine the location and dimensions of the disk. These have the same meaning that they would if you were defining a surface-based plane, though no actual surface is being created. We are creating only a virtual surface over which the rays will be distributed. Later, we will discuss how to make ASAP objects into emitting surfaces (see "Emitting Objects" on page 344).

The next required parameter is the **Ray Count**, which is the total number of random rays that this emitter creates.

The two cells, **Half Angle** and **Half Angle′**, allow you to limit the cone angle into which the rays are directed. The numbers represent the cone half-angle in degrees measured relative to the normal to the plane. In our example, we set both angles to 30 degrees. We may choose to limit the cone angle in this way if the source is coupling into a circular or elliptical lens or mirror some distance away (see the top of Figure 16.3 on page 335). There is no point in creating and tracing rays that have no chance of entering your optical system. The default cone angles are 90 degrees, producing emission into the entire forward hemisphere.

The next cell, **RECT Flag**, is optional. If you select **RECT** in this cell, however, the result is a rectangular "cone" (like a pyramid). This cone is more appropriate when the entrance aperture of your optical system is square or rectangular, as shown at the bottom of Figure 16.3 on page 335.

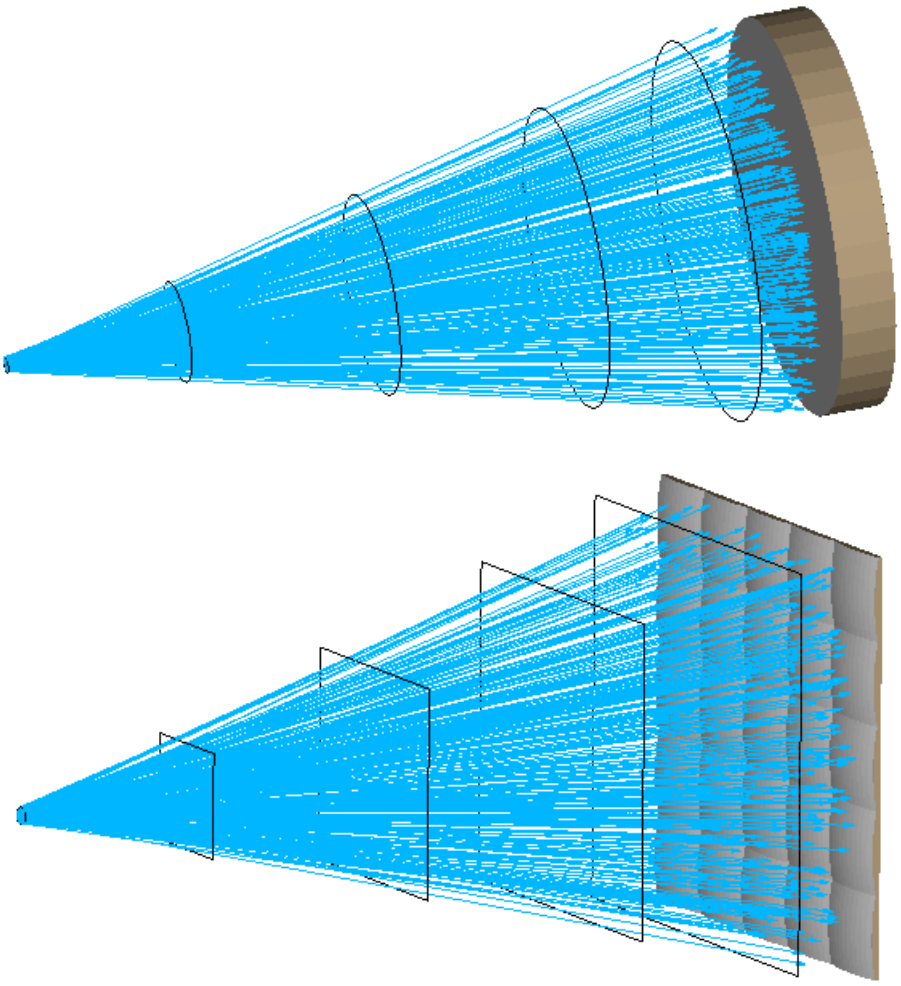


Figure 16.3 This example creates random rays in a Lambertian distribution. By default, ASAP **EMITTING DISK** or **RECTANGLE** commands create a set of rays that fill a complete hemisphere. If a cone angle is specified, it is possible to create only those rays that fill a circular aperture (top figure) or rectangular aperture (bottom figure). This specification is appropriate if you know in advance that the other rays will not couple into your system, and hence cannot possibly affect the behavior of your model. An elliptical aperture is the default. To obtain the rectangular cone, you must select **RECT** in the **RECT Flag** column.

The last optional cell in this command is in the **ISO Flag** column. By default, all ASAP surface emitters have Lambertian intensity distributions. The alternative is an isotropic distribution, obtained by double-clicking this cell, and selecting **ISO** from the menu. The difference between these two distributions is the topic of the next section.

Note: Unlike grid sources where two Builder lines were needed to complete the source definition, one Builder line is all that is needed to define this class of sources. Grids require one line to specify where the rays will be located, and another to define their directions. In this case, the type of emitter you select and the optional parameters you choose place constraints on both position and direction of the rays. The random-number generator does the rest of the work for us. As a result, all the information is contained in this one line.

Lambertian and Isotropic Intensity Distributions

A Lambertian emitter follows Lambert's cosine law: the power leaving the surface will fall off as the cosine of the angle from the surface normal at that point. While no surface is truly Lambertian, it is often an excellent approximation of an incandescent metal or other optically dense materials that appear to emit only from the surface rather than from a volume.

ASAP can produce rays that all carry exactly the same flux, but create fewer rays as the angle to the normal increases.

There are many ways to generate a set of rays that simulate a Lambertian surface. ASAP can produce a random set of rays with an equal probability of pointing in any direction within the hemisphere. The program can then modify the flux of each ray according to Lambert's cosine law. Alternatively, ASAP can produce rays that all carry exactly the same flux, but create fewer rays as the angle to the normal increases. These two situations are shown in Figure 16.4a and Figure 16.4b. While either method gives roughly the same result when a large enough number of rays were created, ASAP uses the latter method. Since it takes the same amount of time to trace a ray with a large flux value as it does one with almost no flux at all, the second method is often more efficient when many rays are traced.

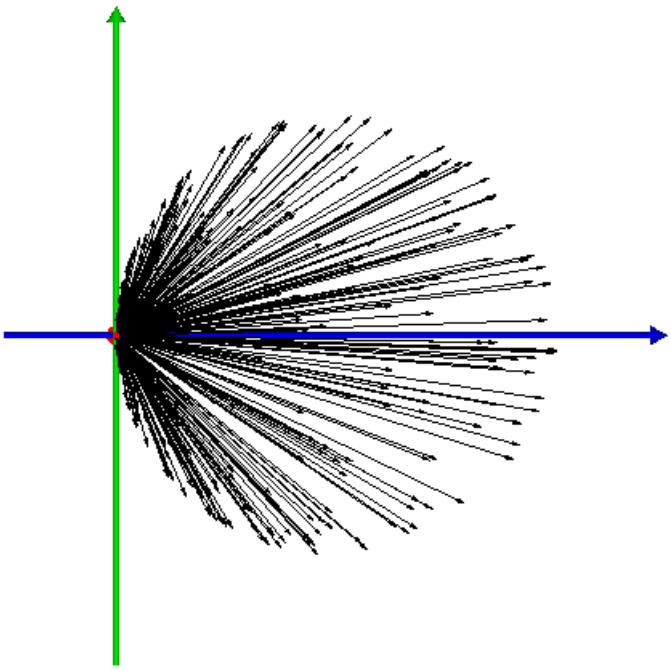


Figure 16.4a A Lambertian power distribution can be simulated two different ways using rays. Rays were created with an equal probability of pointing in any direction, then their fluxes were adjusted to obtain the desired *cos*(θ) dependence. This figure was made using **PLOT RAYS**, so ray lengths are proportional to flux, which is called "flux weighting".

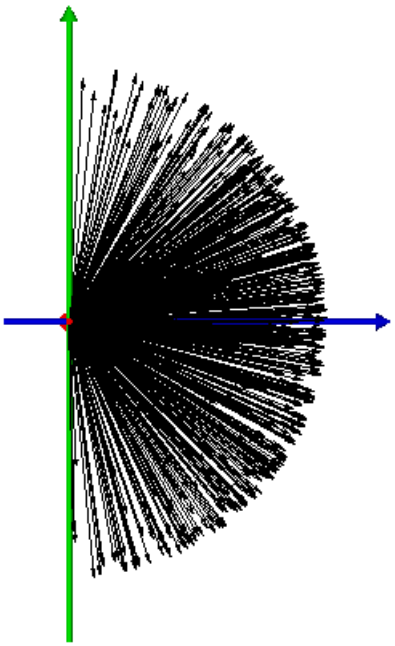


Figure 16.4b All the rays have exactly the same flux, but the ray density was adjusted in a way that depends on their angle to the surface normal. ASAP uses this method with its surface emitters.

An isotropic source has a power distribution that is independent of angle. The simplest way to model such a source with rays is to produce a ray set with completely random positions and directions, giving each ray equal flux. While this is exactly how ASAP models volume emitters (see "Emitting Spheroid"), this is not what the **ISO** option does on surface emitters like the emitting disk and rectangle. ASAP creates the rays with a Lambertian ray density as described above, and then adjusts the flux of the individual rays *upward* in inverse proportion to the cosine of their angle with the surface normal.

Emitting Spheroid

Another class of extended sources is volume emitters. Instead of distributing rays randomly over a surface, they are distributed randomly within a volume. Their directions are also completely random—they have an equal probability of pointing in any direction. All ray fluxes are identical, leading to an isotropic intensity distribution. The simplest of these ray fluxes is the emitting spheroid, found on the Builder menu under **Rays> Emitting** and select **Spheroid** from the drop-down box in the Option column. The resulting command looks like Figure 16.5.

*	Type	Option	X	Y	Z	U'	V'	W'	Ray Count	Axis
EMT	Emitting	Spheroid	0	0	0	1	1	1	1000	

Figure 16.5 Builder line for an emitting spheroid

The first three parameters, **X**, **Y**, and **Z**, are the global coordinates of the center of the spheroid. The next three parameters, **U'**, **V'** and **W'**, are the semi-major heights along each axis. As you can see, this command actually allows us to make *ellipsoidal* volumes in spite of the command's name. Note also that ASAP lets us set all three of these dimensions to exactly zero, allowing the simulation of an isotropic point source. The **Ray Count** cell is the total number of rays created.

The **Axis** parameter is optional. By double-clicking this cell, you can choose from **X**, **Y**, or **Z**. When one of these is selected, ASAP modifies the ray fluxes to yield a toroidal intensity distribution, like that shown in Figure 16.6.

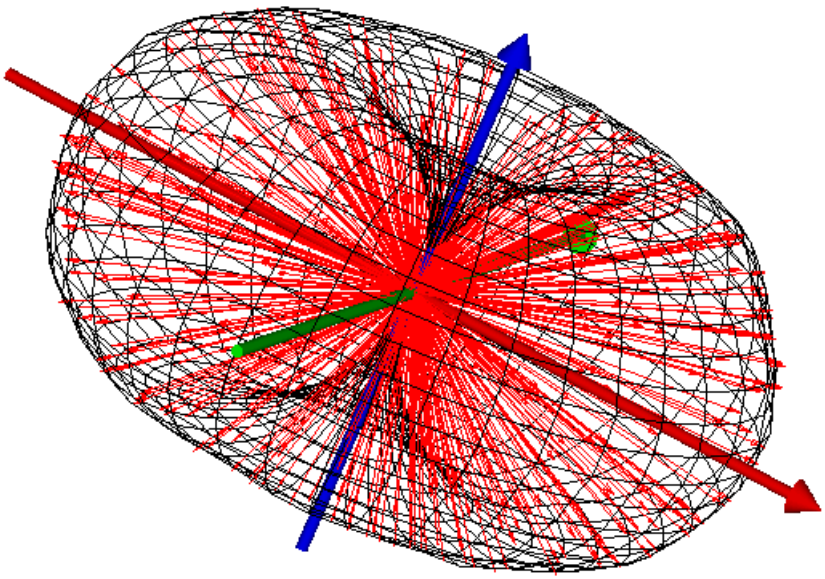


Figure 16.6 The isotropic power distribution of the emitting spheroid can be modified to reduce the flux in a way that varies with the sine of the angle to the named axis (*z*, in this case), roughly simulating the behavior of a small, straight lamp filament. The envelope drawn around the rays was made using the MESH command, whish is one of several methods ASAP provides for visualizing intensity distributions. It will be introduced in Chapter 20, "Display Tools".

Other Volume Emitters

The ASAP Builder provides three other isotropic volume emitters: the emitting box, cone, and pyramid. Examples of these and the emitting spheroid appear in

Figure 16.7a. The **EMITTING BOX** command works much like **EMITTING SPHEROID**, generating a set of rays with random positions, directions, and uniform fluxes. The boundaries of the emitter are, however, a box of rectangular cross section, rather than an ellipsoid. The cone and pyramid are also isotropic in their intensity distributions, but are not uniform in positional ray density. Figure 16.7a clearly shows that the density of rays increases toward the tip of the cone or pyramid.

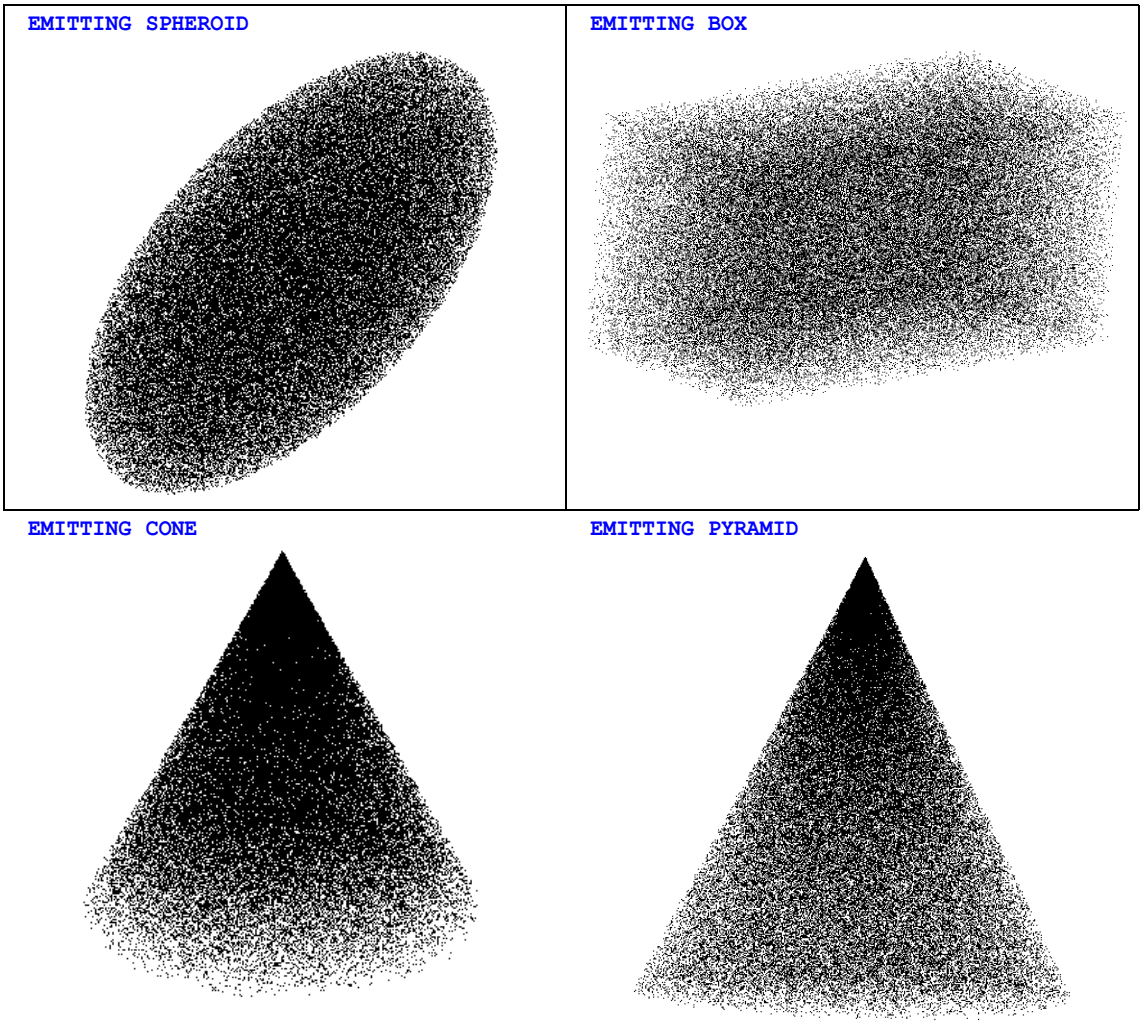


Figure 16.7a Four isotropic emitters, with each point in the illustration representing the location of a ray. There is an equal probability of the ray pointing in any direction. Every ray has exactly the same flux. These illustrations were produced using the Builder's **Preview** function, which (in the case of emitting sources) shows a three-dimensional version of the spot diagram in the **3D Viewer** window.

*	Type	Option	Axis	Z	X	Y	Z'	X'	Y'	Ray Count	ISO Flag
EMT	Emitting	Spheroid	0	0	0	1	1	1	100000		
EMT	Emitting	Box	0	0	0	1	1	1	100000		
EMT	Emitting	Cone	Z	0	0	0	2	1	1	100000	
EMT	Emitting	Pyramid	Z	0	0	0	2	1	1	100000	

Figure 16.7b Builder lines for the four isotropic emitters shown in Figure 16.7a

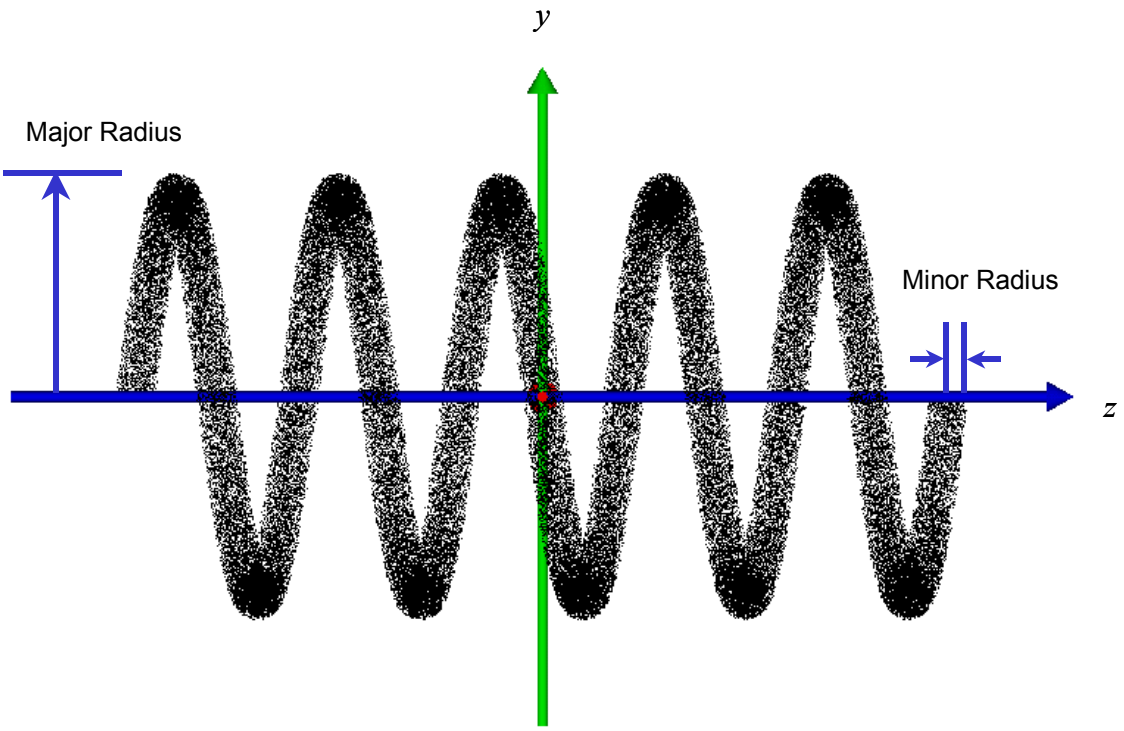
When the rays are created, the number of rays per unit length along the chosen axis is held constant, causing a concentration of flux toward the tip. This concentration approximates the behavior of the plasma discharge near an electrode in a compact arc lamp. The actual ASAP commands generated by the Builder for the four volume sources are shown in the script referenced below.

See Chapter 16 Appendix, "Script 16-2" on page 351.

> ***Note:*** At one time, the best way to model plasma discharges in ASAP involved carefully arranging multiple volume emitters—primarily cones and ellipsoids—to obtain the observed behavior. While this modeling is still useful in some cases, ASAP now includes the inverse Abel transform to help us build better arc models. See "Advanced Source Modeling" on page 347 for a brief description of this method.

Emitting Helix

The **EMITTING HELIX** command combines some of the features of surface and volume emitters, allowing us to approximate the behavior of a helical filament. It fills a helical-shaped volume with rays (Figure 16.8), but each segment or slice emits like a Lambertian cylinder (Figure 16.9).



*	Type	Option	Axis	Start	End	Turns	Major Radius	Minor Radius	Number of	Emit Type
EMT	Emitting	Helix	Z	-1	1	5	.5	.05	100000	

Figure 16.8 The **EMITTING HELIX** command produces a volume emitter in the shape of a helical wire. It is not a true surface emitter, however, since the rays are distributed within a helical-shaped volume.

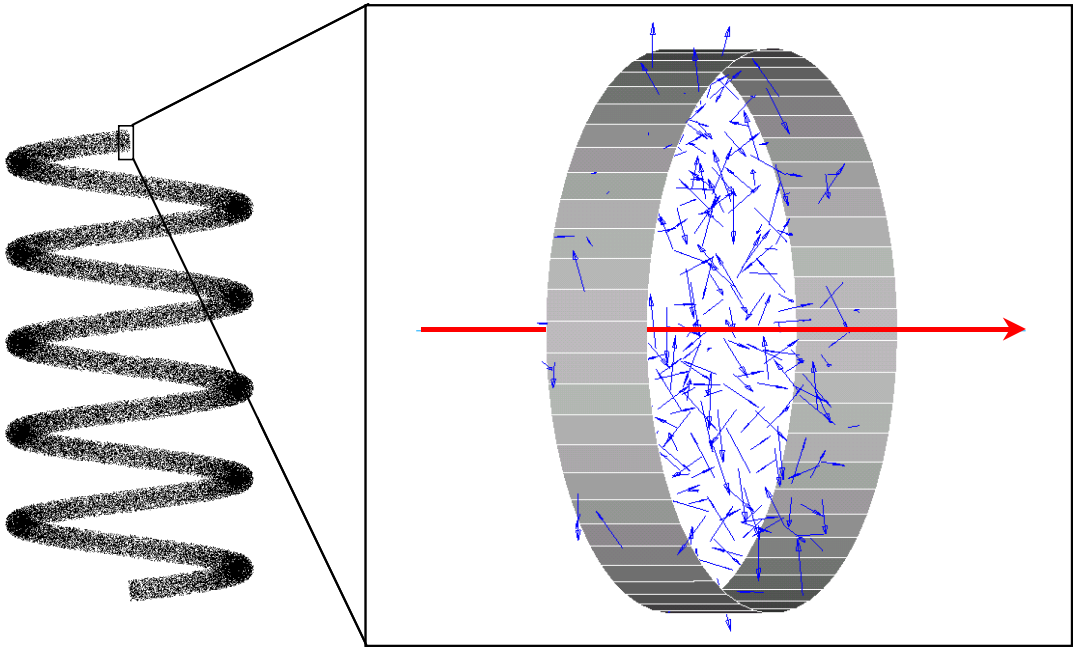


Figure 16.9 Ray locations and directions within a small slice of an emitting helix. Each such slice, if it is thin enough, has roughly the shape of a short cylinder. ASAP fills this volume with random rays, then adjusts the ray fluxes to simulate a Lambertian surface emitter. This is done by reducing each ray's flux according to the projection of its direction vector along the axis of this local cylinder (the red arrow in the figure). For example, a ray that is perpendicular to the red axis retains its full flux, while a ray exactly parallel to it is reduced to zero flux. The short section of a cylinder, shown in this figure, was added to illustrate the shape of the small volume. No such surface is actually part of this source.

The **EMITTING HELIX** command is on the Builder menu under **Rays> Emitters> Helix**. You specify the **Axis** of the filament, the **Start** and **End** coordinates of the helix measured along that axis, and the number of **Turns**. The **Major Radius** parameter controls the size of the coil loops, and the **Minor Radius** controls the wire size. The actual ASAP command generated by the Builder for **EMITTING HELIX** is shown in the script referenced below.

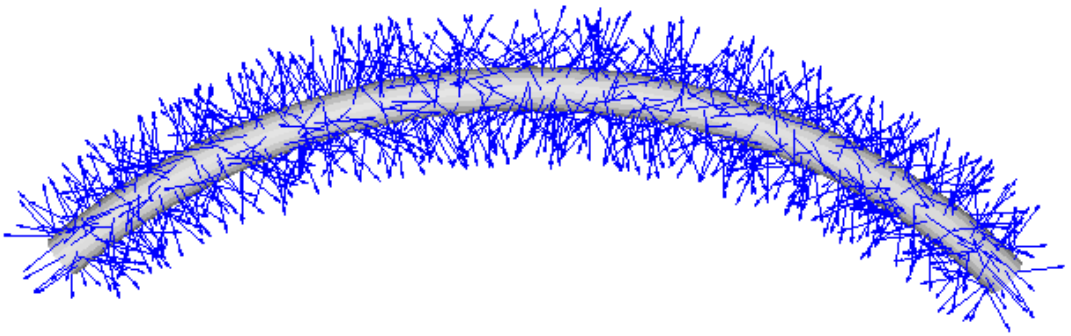
See Chapter 16 Appendix, "Script 16-3" on page 352.

This source has the advantages that it is easy to make and use, the rays come from more or less the right place, and they have the appropriate Lambertian character. It is not perfect, however. A real incandescent helical filament is a surface, not a volume emitter. Further, the emitting helix lacks the geometry corresponding to the wire itself. This source, if present, will produce self-obscuration or "shadows", particularly when viewed along its axis. In summary, this simple model works well when viewed from a position perpendicular to its axis, but poorly along the axis.

Note: We will explore the emitting helix source further in the radiometry exercises at the end of Chapter 19 and Chapter 21. You will see quantitative results that illustrate the limitations of this simple model.

Emitting Objects

Perhaps the most powerful option for creating surface emitters in ASAP is the `EMITTING OBJECT` command. It allows us to turn any object into a Lambertian surface emitter. An example of this appears in Figure 16.10.



*	Type	Option	Object Name/#	Rays	Position	Ray Direction				
SYS	System	New								
	Reset									
CMD	Coating	Absorb		Properties		0	0			
ENT	Arc			Z	0.0	0.1	0.0	0.0	0.0	360
MOD	Shift	One Axis	Y	3						
MOD	Sweep	Dir	AXIS	90	Axis	X	1	0	0	
CMD	Object	Filament	.1							
MOD	Interface	Coating	Coating	Absorb	Air	Air				
MOD	Facets	5	51							
MOD	Rotate	Axis	X	-45	0	0				
EMT	Emitting	Object	.1	1000						

Figure 16.10 A wire filament, modeled by sweeping an `EDGE`-based `ARC`. The `EMITTING OBJECT` command then uniformly distributes 1000 random, Lambertian rays over the surface of this object. Because the object has absorbing optical properties, any rays that intersect it during the ray trace produce the appropriate "shadows" in the analysis.

We have defined a circle using the `ARC` command, shifted it along the *y* axis, and then swept it around the *x* axis to form an arching wire. Note that this is an actual ASAP object, with optical properties specified by its `INTERFACE`

command. This object, named **Filament** here, is treated like any other object in the system during a ray trace. The actual ASAP commands generated by the Builder for the **EMITTING OBJECT** are shown in the script referenced below.

See Chapter 16 Appendix, "Script 16-4" on page 352.

Perhaps the most powerful option for creating surface emitters in ASAP is the **EMITTING OBJECT** *command.*

The **EMITTING OBJECT** command, shown at the bottom of the Builder file in Figure 16.10 on page 344, does the actual work to create rays, turning **Filament** into an emitter. Only two arguments are required.

The first argument specifies the object in the system database that is to be turned into an emitter. We have accepted the default value, **.1**, thereby using relative referencing to specify that the most recently defined object will become an emitter.

Although the cell is named **Object #**, ASAP can also accept the name of the object in this cell.

In either case, you can place a minus sign in front of the name or number to change the direction of the rays relative to the surface normal. This change of direction can be used, for example, to direct rays inward from the surface of a spherical object, rather than outward (the default for a sphere).

The rays are not actually placed "on" the surface. Instead, they are placed slightly above a faceted version of the object.

Before we go on to the next required parameter, we need to discuss how ASAP distributes the rays on the object. The rays are not actually placed "on" the surface. Instead, they are placed slightly above a faceted version of the object. You may recall that when ASAP uses **PLOT FACETS** to create the graphics shown in the 3D Viewer, the object is approximated for that purpose by a series of flat polygons. This approximation was done so that the object could be quickly and easily manipulated (rotated, panned, and zoomed) by the 3D Viewer. We emphasized at the time that this approximation was relevant only to the artwork, having no effect whatsoever on what the rays will "see" during a ray trace. ASAP also uses this meshed version of the object when defining emitting objects, and it does this for basically the same reasons. The process of generating perhaps millions of random rays with a Lambertian intensity distribution would be relatively slow if ASAP had to do the calculations relative to an arbitrary, perhaps complex surface. Such precision is not necessary. It is much faster to develop a Lambertian distribution relative to a series of flat planes.

Since ASAP is creating the rays relative to the wire-frame mesh, be sure to apply the **FACETS** command as an object modifier when you define the object. We have shown this step in the Builder fragment shown in Figure 16.10 on page 344, setting the faceting to 5 and 51 in this example. You can experiment using **System> Plot Facets** until you discover a faceting level that creates an acceptable approximation to the actual surface. Figure 16.11 on page 346 gives a close-up view of our result.

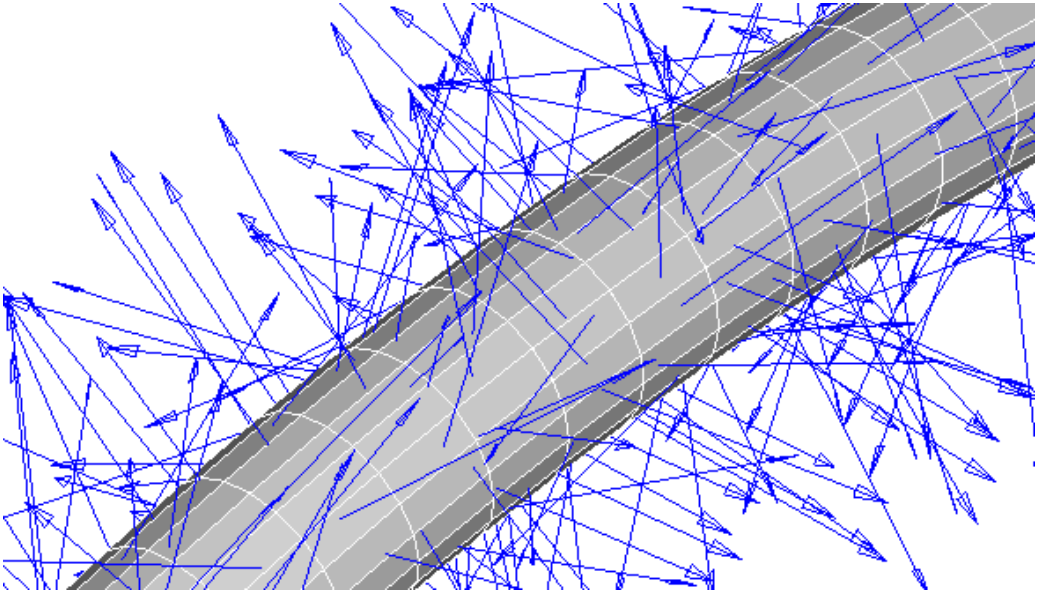


Figure 16.11 The rays created by the **EMITTING OBJECT** command are actually generated in position and direction relative to the facets associated with the object, not the mathematics of the object itself. It is generally important to use the **FACETS** command as an object modifier when the object is defined. Otherwise, the default facet value is used by the **EMITTING OBJECT** command, which is often too low to give a good approximation of the real mathematical surface.

We are now ready for the second required parameter, which is placed in the cell labeled **Rays**. In older versions of ASAP, we actually specified how many rays should be placed on each facet. This method had two problems. First, it was difficult to predict the total number of facets into which an object would be divided, so we were never quite sure how many rays we were asking for in total. Second, and more serious, not all facets have the same area. This difference caused subtle sampling errors, since facets with smaller areas had higher ray densities. While the old syntax is retained to allow backward compatibility, we are now able to specify the *total* number of rays on the object. You signal this to ASAP by placing a minus sign (or no sign) in front of the number. In our example, we specified **-1000**. ASAP then creates a total of about 1000 rays, distributed over the facets in such a way that the ray density remains roughly constant. Placing a plus sign in front of the number tells ASAP to create this number of rays on each facet of the object. You usually get a few more or less than you specified, since ASAP needs the flexibility to adjust this number for a uniform distribution.

The last two parameters are optional:

1. **Position:** allows you to adjust the initial distance (in system units) of the rays above the surface. By default, ASAP creates the rays slightly above the object (or below when a negative sign is placed before the object name

or number in the **EMITTING OBJECT** command). Recall from the ray-tracing discussion in Chapter 10, "Tracing Rays" that rays should never be created at the exact location of an ASAP object. You need only to specify this parameter if you would like to control this initial position more carefully for some reason.

2 **Emit Type:** just as with the emitting disk/rectangle, allows you to override the default Lambertian behavior of this source by double-clicking this cell, and selecting **ISO.** You will obtain an isotropic intensity distribution from each facet of the object.

Advanced Source Modeling

When modeling sources in ASAP, you can create virtually any distribution of rays that can be defined mathematically by using the ASAP command language.

You are not limited to the constraints and limitations of the commands provided in the Builder when modeling sources in ASAP. You can create virtually any distribution of rays that can be defined mathematically by using the ASAP command language. For example, the illustration on the left side of Figure 16.4 is an ASAP source that began as an emitting spheroid, created while the **CLIP** command was in force to eliminate the rays going in the $-z$ direction. A $\cos(\theta)$ "apodization" was then applied to modify the ray fluxes according to their z direction cosine.

Many additional source-modeling tools like these are provided in ASAP. While most are not difficult to use, they are beyond the scope of this *Primer*. Some of these are listed here, along with additional references.

- You can model arc or high-intensity discharge sources based on a bitmap (*.bmp) image of the discharge. The inverse Abel transform is applied to the image to create a set of rays that will simulate the observed behavior. Three examples are included among the sample projects located in **asapxxxx\Projects\samples**. See **SimpleArc**, **CompleteArc**, and **BananaArc**.
- You can import complete source models for commercially available lighting products from the BRO Light Source Library. These can be created in ASAP, using the main menu: **Rays> Create Sources> Use BRO Light Source Library Wizard**.
- You can modify ray fluxes as a function of position, direction, or both. ASAP calls this process "apodization". See the on-line Help for **APODIZE** and **USERAPOD**, or consult the ASAP Technical Guide, *Sources*.
- You can generate ray sets externally and read them into ASAP by a variety of methods. See the on-line Help entries for **EMITTING RAYS**, **RAYS**, and **$READ**. These topics are also covered in the Technical Guide, *Sources*.
- While each source in ASAP can have only one wavelength, broadband sources are easily modeled using an appropriately flux-weighted set of monochromatic sources. ASAP provides tools to modify fluxes as a

function of wavelength, according to the response of the human eye (both photopic and scotopic), a thermal black body of a given temperature, or any other form of spectral apodization that can be described by a mathematical function. See the on-line Help for the SPECTRUM command, or refer to the Technical Guide, *Sources*.

Summary

In this chapter, we have discussed the following new commands.

ASAP Commands	Builder Menu	Description
EMITTING DISK	**Rays> Emitting** select **Disk**	Simulates a surface emitter with elliptical boundaries.
EMITTING RECTANGLE	**Rays> Emitting** select **Rectangle**	Simulates a surface emitter with rectangular boundaries.
EMITTING SPHEROID	**Rays> Emitting** select **Spheroid**	Simulates a volume emitter with ellipsoidal boundaries
EMITTING BOX	**Rays> Emitting** select **Box**	Simulates a volume emitter in the shape of a box with rectangular cross sections.
EMITTING CONE	**Rays> Emitting** select **Cone**	Simulates a volume emitter in the shape of an elliptical cone, with flux concentrated toward the tip.
EMITTING PYRAMID	**Rays> Emitting** select **Pyramid**	Simulates a volume emitter in the shape of a pyramid, with flux concentrated toward the top.
EMITTING HELIX	**Rays> Emitting** select **Helix**	Simulates a helical filament.
EMITTING OBJECT	**Rays> Emitting** select **Object**	Creates a surface emitter in the shape of any ASAP object.

ASAP includes two families of source commands. We introduced grid sources in Chapter 8, and we have been using these since then to simulate point sources. The emitters discussed in this chapter are extended sources, and are characteristically random. ASAP uses a random-number generator to distribute rays on surfaces or within volumes. The program also gives the rays random directions, although the directional distributions are modified to simulate Lambertian emitters in some cases.

Unlike grid sources that require a GRID and a SOURCE command to completely specify the source, the emitter definitions are all contained within one command.

Most of the emitters produce a “cloud” of rays in space in two or three dimensions. There is no geometrical object or entity associated with them. The one important exception to this is the **EMITTING OBJECT**. In this case, rays are defined slightly above an object that was defined previously.

You can use any of the ray graphics tools you learned in Chapter 9, “Verifying Sources” to verify these emitting sources. You can even use the **Preview** feature of the Builder to see their positions relative to existing geometry. The rays appear as small points in the 3D Viewer in this case.

Random Number Generator

Randomness plays a fundamental role in the ASAP emitters. When you define a set of rays using any one of the **EMITTING** *sources, ASAP performs the task of randomly distributing the rays both spatially and in direction. This distribution is done with a random number generator. Like most computer programs, ASAP uses a "pseudo random" algorithm, which means that a very predictable, repeatable, deterministic calculation is performed. As a result, every time you run a Builder file containing the* **EMITTING RECTANGLE** *command, you get exactly the same set of "random" rays. This result is true no matter what computer you use, or what day you run the file.*

What if your file defines two emitters? Will they each produce identical rays? No, they will not. Every time the random-number generator is used, the seed value is automatically changed. If you repeat exactly the same emitter definition in the same file, a different set of random rays will result.

The accuracy of any ASAP simulation involving random emitters ultimately depends on how well a finite number of random rays represents the actual situation you are trying to model. This is the essence of Monte Carlo ray tracing. ASAP may report an analysis result with plenty of precision (many numbers to the right of the decimal place), but this says little about the accuracy of the result.

One way to investigate the accuracy of an ASAP simulation is to take control of the random number generator. Every time you run a simulation with a different set of random rays, it is a little like performing an imperfect experiment that includes sources of error. If the errors are "random" (caused by the specific set of random rays generated) rather than systematic (caused by an imperfect model of the system or source), you can gain some insight into the accuracy by repeating the experiment several times. To do this with an ASAP simulation involving random emitters, you will need to force the program to use a different seed value for each run. You can do this by issuing a **SEED** *command above the emitter. This command is available in the* **Builder** *menu (Rays> Ray Control> Seed). The seed value should be a large, odd number. The default value is 2,000,000,001.*

APPENDIX

16A

SCRIPTS FOR CHAPTER 16

The following ASAP scripts are referenced in Chapter 16, “Extended Sources”.

Script 16-1

```
!!THE ACTUAL ASAP COMMANDS GENERATED BY THE BUILDER ARE THE FOLLOWING:
EMITTING DISK      Z 0.0 1.0 1.0 10000 90.0 90.0
EMITTING RECTANGLE Z 0.0 1.0 1.0 10000 90.0 90.0
!!10000 RAYS ARE GENERATED BY EACH EMITTER.
!!THE RAYS ARE EMITTED INTO A FULL HEMISPHERE.
```

Script 16-2

```
!!THE ACTUAL ASAP COMMANDS GENERATED BY THE BUILDER
!!FOR THE 4 VOLUME SOURCES ARE THE FOLLOWING:
EMITTING SPHEROID   0.0 0.0 0.0 1.0 1.0 2.0 100000
EMITTING BOX        0.0 0.0 0.0 1.0 1.0 2.0 100000
EMITTING CONE     Z 0.0 0.0 0.0 2.0 1.0 1.0 100000
EMITTING PYRAMID  Z 0.0 0.0 0.0 2.0 1.0 1.0 100000
```

Script 16-3

```
!!THE ASAP COMMAND GENERATED BY THE BUILDER FOR THE EMITTING HELIX FOLLOWS:
EMITTING HELIX    Z -1 1 5 .5 .05 100000
```

Script 16-4

```
!!THE ASAP COMMANDS GENERATED BY THE BUILDER FOR THE EMITTING OBJECT FOLLOWS:
COATING PROPERTIES
  0 0  'ABSORB'
EDGE
  ARC Z 0.0 0.1 0.0 0.0 0.0 360
  SHIFT Y 3
  SWEEP AXIS 90 1 0 0
OBJECT  'FILAMENT'
  INTERFACE COATING ABSORB AIR AIR
  FACETS 5 51
  ROTATE X -45 0 0
EMITTING OBJECT .1 1000
```

CHAPTER

17

RAY CESSATION—STOPPING RAYS PREMATURELY

In this chapter, we will briefly return to the subject of the ray trace process itself. Specifically, we will discuss the reasons why rays stop tracing before they reach the end of an optical system. Ideally rays will leave the sources, interact with the appropriate objects along their paths, and finally come to rest on one or more surfaces with the special "absorbing" interfaces—objects with a zero reflection and transmission coefficient (see "Coatings" on page 50 in Chapter 3). There are a variety of reasons why it may not be possible, or even desirable, for rays to continue tracing this far, however. They may stop because of a defect in our model, like having the absorbing surfaces in the wrong place. There may be logic errors in an **INTERFACE** command. In other cases, they may stop because they are no longer worth tracing due to an undesirable direction of travel, or low flux value. You may choose to stop them in the interest of speed and efficiency.

Take note of ray-cessation warnings, and determine why they occurred.

Any time a ray stops on a surface that has non-zero reflection and transmission coefficients, ASAP issues a *ray cessation warning*. You need to check these warnings and make sure you determine why they occurred. New users often neglect this step, and may be missing critical information that will affect their analysis results. While intelligent defaults have been built into ASAP, you have control over this "ray cessation" process. Monitoring and controlling ray cessation is the subject of this chapter.

> ***Note:*** Some ray-cessation warnings are generated by mechanisms that we do not discuss in depth in this *Primer*. However, they are mentioned briefly to make this chapter a complete reference on the topic. When this is the case, we will give references to other ASAP documents to supplement the discussions here.

Ray-Cessation Table

At the end of every ray trace, ASAP prints a ray cessation table in the Command Output window, if any rays have stopped on a surface with a finite reflection or transmission coefficient. An example appears in Figure 17.1. The table is entitled `PARENT warnings`.

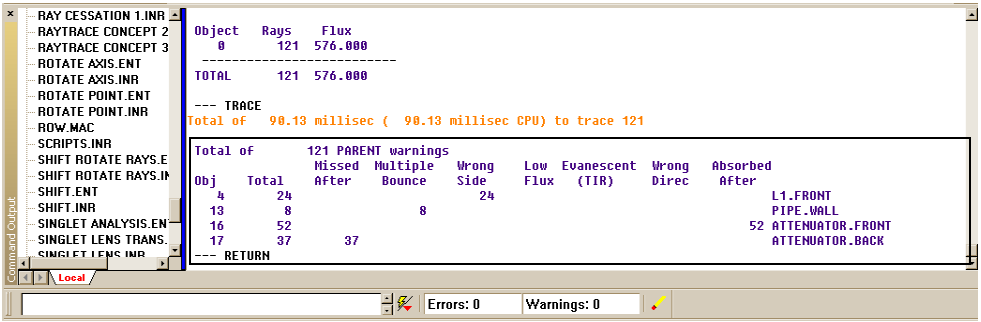


Figure 17.1 The ray-cessation table appears in the **Command Output** window after the ray trace is complete. If you have issued other commands after the ray trace, you can still find the table by scrolling up the window until you find, ---`TRACE` and any ray-trace progress output that follows it. The table (if there is one) is next. ASAP does not print a table when all the rays stopped on absorbing surfaces.

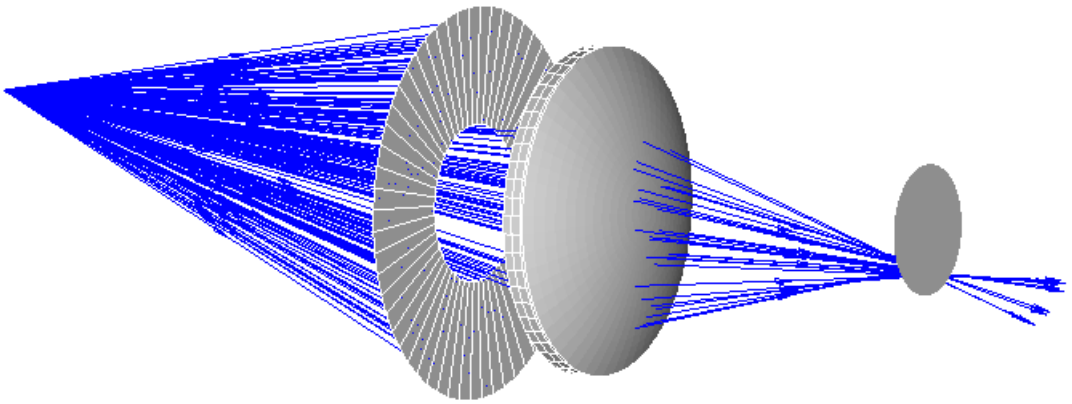
In some cases, another table may be in place of or in addition to this, which is named `CHILD warnings`. Child rays result from processes like Fresnel reflections, scattering, birefringence, and diffraction gratings when the energy is taken form the original (parent) rays, and divided among new (child) rays. We will not encounter any of these topics in this *Primer*.

> ***Note:*** As Figure 17.1 shows, these ray-cessation "warnings" do not appear in the warning count maintained on the status bar at the bottom of the ASAP window. The warning counter shows other warnings, such as duplicate object names, or uninitialized variables. Ray warnings accumulated during a ray trace are considered to be a special class, outside of this general count.

The first column of the table is labeled **Obj**, and includes the object number of all non-absorbing objects in your system that have rays on them. The last column gives the corresponding names of these objects. The column labeled **Total** gives the number of rays stopped on each of these for any reason. ASAP shows the specific reason in the remaining columns, placing them in one of seven classifications. We discuss each of these below.

Missed-After Warnings

Rays are listed as `Missed After` if they have traced to an object with a non-zero reflection or transmission coefficient, and find no other objects along their paths. See Figure 17.2.



```
Total of  190.43 millisec ( 160.23 millisec CPU) to trace 200

 Total of        10 PARENT warnings
                 Missed  Multiple    Wrong     Low  Evanescent  Wrong   Absorbed
 Obj    Total    After    Bounce     Side      Flux    (TIR)     Direc    After
    2       10       10                                                    L1.BACK
```

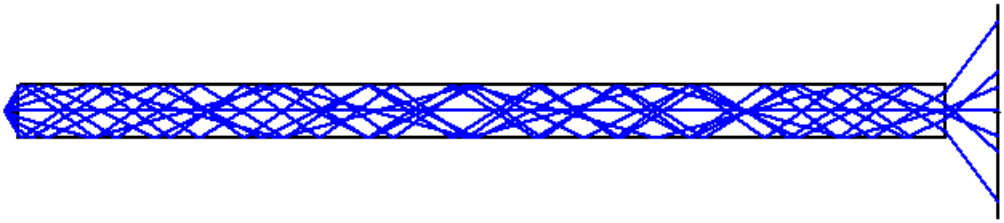
Figure 17.2 Many of the rays from this off-axis point source miss the absorbing detector. While the graphics may show them apparently exiting the system, they actually remain on the back of the lens (**L1.BACK**), and are listed as `Missed After` in the ray cessation table. All other rays stopped on the absorbing surfaces.

While ASAP graphics may show the rays apparently exiting the system, the rays are, in fact, still located on the last object that they touched. Remember that these rays have been traced to *and through* the lens. They have refracted at the exit surface, and will have the appropriate directions and fluxes. This result allows us to perform analysis on the rays, which would be impossible if they had actually escaped the system.

Multiple-Bounce Warnings

A ray will stop due to `Multiple Bounce` when it interacts with the same object 12 times consecutively, or when it interacts with any number of objects more than 1000 times. ASAP places these default limits on rays to prevent them from becoming trapped inside an object (like a reflecting sphere), or a system of multiple mirrors. Under these circumstances, a ray trace could continue indefinitely, or until you become concerned and push the **Abort** button.

Many situations exist, however, where we expect and need more than 12 consecutive ray intersections. Figure 17.3 shows a light pipe where rays enter from the left and proceed through the pipe by total internal reflection (TIR).



```
Total of   20.03 millisec (  20.03 millisec CPU) to trace 9

 Total of         2 PARENT warnings
                 Missed  Multiple   Wrong    Low  Evanescent  Wrong   Absorbed
 Obj    Total    After    Bounce    Side     Flux   (TIR)     Direc    After
    2        2                2                                            PIPE
```

Figure 17.3 Nine rays enter a light pipe, all reflect by TIR with no flux loss, but only seven rays leave the pipe. This reduction occurs because ASAP limits the number of interactions with the same object to 12 by default. After 12 ray intersections, two rays stopped due to Multiple Bounce. To prevent this result, you must use the **HALT** command to change the multiple-bounce limit.

Some of the rays are not able to make it to the end of the pipe in 12 or fewer reflections. If ASAP is issuing a Multiple Bounce ray-cessation warning in this situation, it is clearly a problem that needs to be corrected.

You can override the default consecutive-hits, multiple-bounce limit with the **HALT** command. This command is found among the source modifiers in the Builder on **Trace> Trace Control> Halt**. See Figure 17.4.

*	Type	Max Interactions	Direction	Flux Total
CMD	Halt	12		1E-6

Figure 17.4 **Builder** line using the **HALT** command.

The first parameter is **Max Intersections**, which defaults to just 12 consecutive interactions with the same object. For light-guide problems, you may need to increase this number to 1000 or even higher. The **Direction** and **Flux Total** parameters will be discussed in “Low-Flux Warnings” on page 359 and “Wrong Direction Warnings” on page 360. They can remain blank to accept the default values. You can place the **HALT** command anywhere in your Builder file before the ray trace. These parameters are used only during the ray trace.

See Chapter 17 Appendix, “Script 17-1” on page 369.

You may also need to override the limit on the *total* number of times a single ray intersects any of the objects in the system. This step may be necessary if you

are modeling a large system, or a device that utilizes many reflections. You can modify this parameter with the CUTOFF command, found in the Builder on **Trace> Trace Control> Cutoff**. See Figure 17.5.

*	Type	**Threshold**	Interaction
CMD	Cutoff	1.00E-18	1000

Figure 17.5 Builder line using the CUTOFF command.

We will discuss the **Threshold** parameter in "Low-Flux Warnings" on page 359. The **Intersections** cell controls the total number of object-intersections for a single ray, and represents the other possible solution to `Multiple Bounce` ray-cessation warnings. The CUTOFF command can also be issued any time before you perform the ray trace.

See Chapter 17 Appendix, "Script 17-2" on page 369.

Wrong-Side Warnings

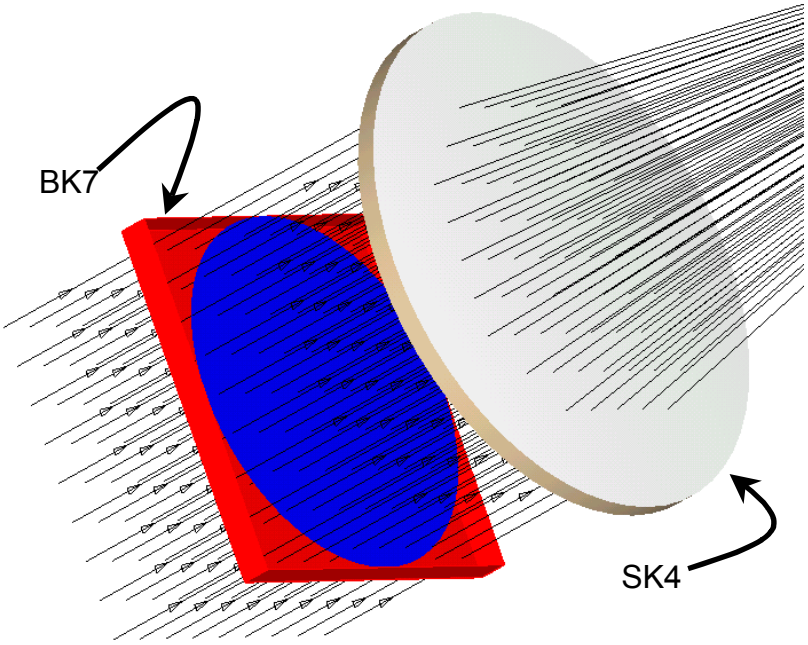
Wrong-side warnings occur when a ray reaches an interface, but cannot reconcile the media choices with the current medium of the ray.

Wrong-side warnings occur when a ray reaches an interface, but cannot reconcile the media choices with the current medium of the ray. These warnings are usually caused by one of two modeling errors. You may have:

1. assigned incorrect media to an interface command for one or more system components.
2. failed to "seal up" a volume, allowing rays to "leak" out of a component.

An example of a leaking component is shown in Figure 17.6 on page 358. The first optical element encountered by the rays is a glass filter made of BK7. The rays enter through a square front surface. The exit face (shown as blue in the figure) was inadvertently made round instead of square. As a result, many rays leak out of the component without interacting with the exit surface.

As mentioned in Chapter 4, "Building and Previewing Geometry", rays always know the medium in which they are currently traveling (see the sidebar, "How does ASAP know?" on page 74). When they reach the SK4 lens, these rays behave as though they are still in BK7. The interface command for the entrance face of the lens lists the choices **AIR** and **SK4**. Since neither of these lenses matches the current medium of the ray, the ray stops on the entrance face of the lens, and ASAP issues a wrong-side warning.




```
Total of  199.95 millisec ( 180.26 millisec CPU) to trace 81

 Total of        12 PARENT warnings
                 Missed  Multiple    Wrong     Low  Evanescent  Wrong   Absorbed
 Obj    Total    After    Bounce     Side      Flux    (TIR)     Direc    After
    4       12                          12                                    L1.FRONT
```

Figure 17.6 Because of an error in the construction of one system element, the BK7 filter, rays were able to leak from the component without interacting with the exit face (shown in blue). The result is a wrong-side error when the rays reach the SK4 lens. These warnings always need to be studied and understood before proceeding with the analysis step.

If you receive wrong-side warnings after a ray trace, it is critical that you understand their cause before proceeding. If you have leaks or incorrect interfaces assigned to some of your surfaces, even rays that traverse the system without warnings may have refracted inappropriately. This warning is usually the sign of a modeling error that should be corrected.

Some tips for diagnosing the exact cause of wrong-side warnings are given in the sidebar, “Tracking Down Wrong-Side Problems” on page 366.

> ***Note:*** As we saw in Chapter 13, edge-based objects sometimes leak when rays intersect them right at their boundaries, or at segment boundaries. While these are rare, you may sometimes see them in very large ray traces involving edge-based objects,

perhaps imported from a CAD program. Generally, the `Wrong Side` warning is your first and best indication that leakage is occurring. Once you have confirmed that edge leaks are the cause of the warnings, these few rays can usually be safely ignored.

Low-Flux Warnings

You will see `Low Flux` ray-cessation warnings when individual ray fluxes drop below either of two user-controlled flux thresholds. Rays lose flux, or are replaced by rays having less flux, for a variety of reasons, as they move through an ASAP model. Some examples include Fresnel reflections, imperfect coatings, absorption, and scattering. ASAP ceases to trace a ray if its flux has become too low. This behavior is often desirable, since the rays may not play a significant role in the analysis. Remember that it takes just as long to trace a weak ray as it does a strong one—you may be wasting valuable computing time.

Have the fluxes really dropped so low that they cannot possibly affect your results at these default values? That is really up to you to decide, based on how weak these stopped rays have become, relative to the ones that are still proceeding, and how many there are. In some cases, you may want to lower, or even raise the thresholds from their default values. The **HALT** and **CUTTOFF** commands are both used to modify these thresholds. As we saw above, both commands are on the Builder menu, **Trace> Trace Control**.

The **HALT** command controls the *relative* flux threshold. The default value is 1.0×10^{-6}. At this setting, any ray that drops to less than 1 part in 1,000,000 of its *original* flux (the flux it had when it was created, or when the flux was modified with the **FLUX TOTAL** command) stops, and the low-flux warning is issued. You can override this warning by entering a different value in the **Flux Total** cell.

The **CUTOFF** command controls the *absolute* threshold. The default value is 1.0×10^{-18}. Any ray whose flux drops below this value stops, and the `Low Flux` warning is issued. You can change this behavior by entering a different value in the **Threshold** cell.

Evanescent (TIR) Warnings

If you are doing wave optics with ASAP using Gaussian beam decomposition, you will get *Evanescent (TIR)* ray-cessation warnings any time evanescent beams are created. These beams represent energy that travels a very short distance along or into a non-transmitting medium. We often see the beams when the **DECOMPOSE DIRECTION** command is used to simulate a small field with a set of tipped Gaussian beams. See the Technical Guide, *Wave Optics in ASAP*. Since this energy does not propagate any significant distance, ASAP does not trace these beams.

In geometrical ray tracing, ASAP issues an *Evanescent (TIR)* warning if you are modeling an object that functions by total internal reflection, but have permitted no ray splitting. Although we have discussed only ideal coatings in this *Primer*, ASAP is able to model angle-of-incidence dependent behavior using either Fresnel's equations or **COATING MODELS**. See the sidebar, "Bare Surface and Fresnel's Equations" on page 367.

Wrong Direction Warnings

You may see *Wrong Direction* ray-cessation warnings if you have used the **HALT** command to limit the directions in which rays can propagate. The halt parameter that is labeled **Direction** in the Builder entry allows you to specify an "undesirable" propagation direction:

*	Type	Max Interaction	Direction	Flux Total
CMD	Halt	12	-Z	1E-6

Figure 17.7 Specify an "undesirable" propagation direction with the **Halt** parameter, **Direction**.

This direction must be along one of the global axes (**X**, **Y**, **Z**), and can be positive or negative. If, after interacting with an object, a ray has a *component* of its direction vector lying along the specified axis, the ray will stop, and a *Wrong Direction* warning will be issued. In the example above, a ray will cease if it has a *z* direction cosine that is negative. ASAP allows rays to move in any direction if this cell is left blank, which is also the default.

See Chapter 17 Appendix, "Script 17-3" on page 369.

Absorbed-After Warnings

You will see an `Absorbed After` ray cessation warning when the flux of a ray falls below either the **HALT** or **CUTOFF** threshold while traversing an absorbing medium. So far, all the indices of refraction for our media have been pure real numbers. In general, the index of refraction can be a complex number,

$$\eta = n + ik.$$

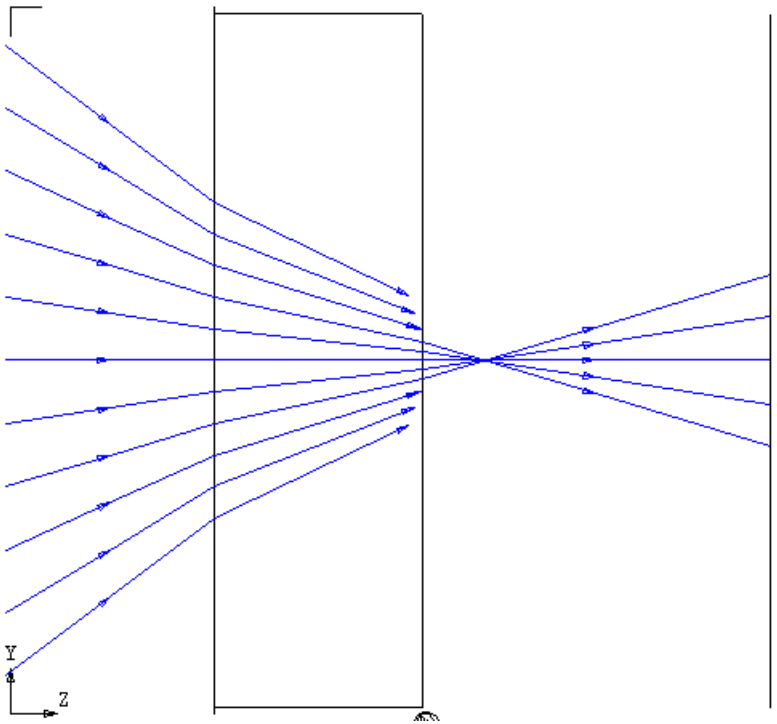
The imaginary index, *k*, controls the amount of absorption in the medium. For a ray with initial flux F_0, wavelength λ, traveling a distance L in an absorbing medium with imaginary index of refraction k, the new flux is

$$F = F_0 e^{-\frac{4\pi k}{\lambda} L}$$

ASAP allows us to model complex indices of refraction in the Builder. See Figure 17.8a below and Figure 17.8b on page 362.

*	Type	Name	Coating Number	Cmd	Angle o	Reflect 1	Transmit 1	Reflect 2
SYS	System	New						
	Reset							
CMD	Coating	Transmit		Properties		0	1	
SYS	Units	MilliMeters						
SYS	Wavelengths		550					Nanometer
CMD	Media	NDGLASS	Media	1.4`0.000195				
OBJ	Plane	ND.FRONT	Axis	Z	0.0	Rectangle	5	5
MOD	Interface	Coating	Coating	'TRANSMIT'	'Air'	'NDGLASS'		
OBJ	Plane	ND.BACK	Axis	Z	3.0	Rectangle	5	5
MOD	Interface	Coating	Coating	'TRANSMIT'	'Air'	'NDGLASS'		

Figure 17.8a Builder file for optical element with an absorbing medium.



```
Total of   20.03 millisec (  20.03 millisec CPU) to trace 11

 Total of 6 PARENT warnings
                Missed  Multiple    Wrong     Low  Evanescent  Wrong    Absorbed
 Obj    Total    After    Bounce     Side     Flux    (TIR)     Direc     After
    1        6                                                            6 NDFILTER.FRONT
```

Figure 17.8b If an optical element is made of an absorbing medium, the flux of a ray may drop below one of the thresholds before it exits the material. In this example, the rays with the most direct path through the neutral-density filter were able to exit, but the rays traveling at higher angles did not exit. Of the 11 rays that entered the filter, only five reached the other side. The other six stopped part way through, and ASAP issued an `Absorbed After` ray cessation warning.

The example models a neutral-density filter with a thickness of 3 mm. We have entered the index of refraction as **1.4`0.000195**. ASAP uses the *grave* accent symbol (`) to separate the real and imaginary parts of a complex number. In this case, n = 1.4 , and k=0.000195. The entire complex number must be entered into the cell with no spaces. Applying the flux equation above shows that any ray that traverses more than 3.1 mm of this material will drop below the relative flux threshold of 1.0×10^{-6}. See "Low-Flux Warnings" on page 359.

ASAP traces rays through non-homogeneous materials in short steps. After each step is taken, the flux is checked against both the relative flux threshold controlled by **HALT**, and the absolute threshold controlled by **CUTOFF**. If the flux drops below either of these, the ray stops after that step, and the ray cessation warning is issued. As Figure 17.8b shows, some of the converging rays took a path that was longer than 3.1 mm, and stopped before reaching the exit face. These rays caused the *Absorbed After* ray cessation warning shown in the figure. While their positions in the ray table after the trace are somewhere between the two surfaces, ASAP lists them as belonging to **ND.FRONT**, the entrance surface of the filter.

The *Absorbed After* ray-cessation warning can also occur when using the **SPLIT MONTECARLO** command. A detailed explanation of this command is beyond the scope of this *Primer*, but a brief comment is needed here to complete the discussion of this ray-cessation warning.

When the **SPLIT MONTECARLO** command is used at an interface, no additional specular rays are actually created. For each ray in, there is one ray out with the same energy as the incoming ray. The direction of the new parent ray is probabilistically selected from the possible child rays based on the reflection and transmission coefficients at the interface. When these coefficients sum to less than 1, there is the finite probability

$$p = (1 - R - T)$$

that the ray stops at the interface. To account for all the incoming energy, this fraction, p, of the incoming rays is left on the interface, and a ray-cessation warning of *Absorbed After* is issued for these rays.

Ray Cessation in Wave Optics

ASAP can decompose any complex field (phase and amplitude) into an ensemble of Gaussian beams. The motivation for doing this is that Gaussian beams can be propagated by ray trace methods.

ASAP also uses ray-trace methods to propagate coherent wave fronts through optical systems. ASAP accomplishes this using a method called Gaussian beam decomposition. This is the topic of the Technical Guide, *Wave Optics in ASAP*. In short, ASAP can decompose any complex field (phase and amplitude) into an ensemble of Gaussian beams. The motivation for doing this is that Gaussian beams can be propagated by ray trace methods. ASAP represents each Gaussian beam in the ensemble with a base ray and parabasal rays, which are traceable geometrical rays. For the usual choice of **`PARABASAL 4`**, there are four parabasal rays. (However, if **`PARABASAL 8`** is chosen, each beam is represented with a base ray and eight parabasal rays.) All these rays proceed through the system using the rules of geometrical ray tracing, including Snell's law, and the law of reflection. The parabasal rays carry the information that allows ASAP to reconstruct the Gaussian beam at any point along the way. The coherent superposition of all the Gaussians in the ensemble now represents the field at that place.

These parabasal rays may also cease to trace in the course of a ray trace. Only two possible reasons exist in this case: `Wrong Side` and `TIR`. The situation for Wrong Side warnings in the case of parabasal rays is a little different, however. Because the parabasal rays are following independent paths through the system, it is possible for them to literally hit the wrong side of an object, relative to the base ray. ASAP saves the entity normal of the base ray intersection point and compares that to those of the parabasal rays. If they have not hit the same side, ASAP issues the `Wrong Side` warning.

The base ray of the Gaussian beam is much like the geometric rays we have been tracing. It represents the normal to the wave front at that point in the energy field. It stops for any of the reasons described in the previous sections, and is controlled by the same parameters. If either the base ray or one of the parabasal rays stops prematurely, all five rays stop at that object.

Summary

In this chapter, we have discussed the following new commands.

ASAP Commands	Builder Menu	Description
HALT	**Trace> Trace Control> Halt**	Controls ray cessation parameters, including consecutive intersections with the same object, undesirable direction, and relative flux threshold.
CUTOFF	**Trace> Trace Control> Cutoff**	Controls ray cessation parameters, including total intersections with all objects, and absolute flux threshold.
FRESNEL AVE	**Trace> Trace Control> Fresnel**	Instructs ASAP to use Fresnel's equations to calculate R and T coefficients with **Bare** coatings.
SPLIT	**Trace> Trace Control> Split**	Controls the number of generations permitted when splitting rays.
IMMERSE	**Rays> Ray Control> Immerse**	Causes new sources to begin in the named medium, rather than **Air**.

Unlike real optical power, ASAP rays may stop before that energy completely vanishes. The normal way to stop rays for analysis is to define surfaces that have both reflection and transmission coefficients set to zero in their interface commands. This is a flag to ASAP that we want rays to stop at these places. As we saw in Chapter 4, the ray fluxes are not actually set to zero, but retain the values they had when they arrived.

Rays may stop before reaching an "absorbing" object for any of the following reasons:

- Rays may miss all absorbing objects in the system. If this happens, they stop on the last object they touched, and ASAP issues a `Missed After` warning.

- Rays stop when they interact with the same object too many times consecutively, or suffer too many total object intersections. These limits prevent ray traces from continuing indefinitely.
- Rays stop when ASAP cannot reconcile the media choices at an interface with the current medium of the ray. In this case, ASAP issues a `Wrong Side` warning.
- A ray whose flux drops below user-controlled thresholds ceases, and ASAP issues a `Low Flux` warning. You can specify both a relative and an absolute flux threshold.
- Rays stop when ASAP is not correctly configured to create TIR rays, or when evanescent beams are created while working in wave optics mode. In this case, ASAP issues an `Evanescent (TIR)` warning.
- Rays may stop if you have specified an undesirable direction with the **`HALT`** command. ASAP issues a `Wrong Direction` warning when a ray has any component of its direction vector along the direction specified.
- Rays stop when their flux drops below either the relative or absolute flux threshold, while they step through an optical component with an imaginary (absorbing) component to its index of refraction. ASAP issues an `Absorbed After` warning when this happens.

Rays may be stopping for good reasons that make your overall ray trace more efficient. They may also stop for bad reasons that indicate a problem with your model, or inappropriate ray-trace parameters. You should always look for ray-cessation warnings in the Command Output window. Make sure you understand why the rays stopped, and that you are satisfied that they have stopped for acceptable reasons.

Tracking Down Wrong-Side Problems

What should you do when ASAP has issued a wrong-side warning after a ray trace? Sometimes the problem is obvious. Did you forget to define a surface? Did you cut and paste some interface commands into your definitions without modifying the media cells appropriately? For new users, these are frequently the source of the problem.

In many cases, however, the solution may not be apparent. ASAP provides many useful analysis tools that you can use in this situation, most of which you have already seen in previous chapters.The procedure below usually leads you quickly to the source of the problem even in the most complex systems.

1. Isolate the problem rays for analysis. This is the first step in any ASAP analysis task, and this is no exception. The ray cessation warning tells us where the problem rays are now. In Figure 17.6 on page 358, they stopped on object 4, which is **L1.FRONT**. From the ASAP menu bar, select **Analysis> Choose Rays> Consider** to isolate only those rays on this object.
2. Find a ray number corresponding to one of these problem rays. You can do this with **Rays> Ray Information> List Ray Data**. Select the **Positions** and **Directions** option in the dialog box. ASAP lists the problem rays in the Command Output window. The last column shows the ray numbers. Pick any one of these ray numbers for detailed analysis.
3. Find out the medium in which the ray assumes it is located, and the object encountered previously. This is also done from the **Rays** menu by selecting **Rays> Ray Information> Show Ray Details**. Enter the ray number in the dialog box that you discovered in Step 2, and select the **Basic Data** and the **Paths** options. ASAP also prints these data in the Command Output window. Note the number associated with the variables **M0**, and **I0**. These are the medium number and the previous object number for this ray. You can find the medium name corresponding to this number by selecting **System> List Media**. The object's names and numbers are shown together at the **Object** tab of the ASAP Workspace window.
4. Carefully inspect the previous optical component discovered in Step 3. Use the geometry verification tools that were discussed in Chapter 5 to look for modeling errors. Then look for errors in the interface commands associated with all of that object's surfaces. This usually leads you to the problem, since the previous object is often where the problem occurred.
5. Leaks can often be found graphically when we follow a problem ray through the system using the ASAP 3D Viewer. You can do this by selecting **Rays> Choose Rays> Select** after defining the source, but before performing the ray trace. (You may need to place your source definition in a separate file to accomplish this.) Once the source is defined, pick one of the problem rays discovered in Step 2 using SELECT. When you trace, be sure to plot rays with Plot Facets graphics of the geometry. Then use the 3D Viewer to carefully follow the ray through the system, looking for gaps in the geometry.
6. If all else fails, isolate a single ray before the trace, as you did in Step 5, but this time use **Trace> Advanced Trace Rays**. You now have complete control of the process, including the ability to step through the system, one surface at a time. Use the option to list positions, so you know where the ray is as it moves along. Check the medium number as described in Step 3 after each surface intersection. Compare the results carefully with your expectations, given your knowledge of the system. With care, this will certainly lead you to the error.

Although we do not try to introduce ray splitting in the limited space available in the Primer, using Fresnel's equations to compute reflection and transmission coefficients from a bare surface is not difficult. Three steps are necessary to turn on the Fresnel calculation:

1. Issue the command **FRESNEL AVE** near the top of your Builder file before defining any geometry. The command is found on the Builder cell's drop-down menu, **Trace> Trace Control> Fresnel**.
2. Issue the **SPLIT** command to control the number of generations of child rays that ASAP will produce. Your original rays (called "parent rays") can continue to split an unlimited number of times, but if **SPLIT** is set to only 1, the child rays cannot split again. The command is found in the Builder under **Trace> Trace Control> Split**.
3. Set the coating to **Bare** in the **INTERFACE** command. This coating is pre-defined, so no **COATING** command is needed.

The combination of these three steps in a file causes ASAP to use Fresnel's equations to calculate the reflection and transmission coefficients as a function of angle of incidence, and index of refraction for every ray that reaches this interface. Here is an example:

*	Type	Option	Cmd	Coating	Media 1	Media 2	List			
MOD	Fresnel	Ave								
MOD	Split	1	Off							
CMD	Media	Glass	Media		1.5					
OBJ	Plane	ENDFACE	Axis	Z	0	Rectangle	1.5	1.5		
MOD	Local	Long Form	-1.5	1.5	-1.5	1.5	-0.5	3.5	Z	0
MOD	Interface	Coating	COATING	Bare	Glass	Air				
SYS	Immerse	Glass								
EMT	Grid	Rect	Z	-0.2	-0.5	0.5	-1	0	1	11
MOD	Source	Position	0	-1	-0.5					
CMD	Move	To Point	0	-1	-0.5					

Note that we have introduced another new command just before the source is defined. The **IMMERSE** *command is located in the Builder on* **Rays> Ray Control> Immerse**. *When this command is entered before defining sources, all rays created after that command begin in the named medium, rather than in the default* **Air** *medium. As a result, we are able to use this example to show that ASAP is correctly reflecting rays by Total Internal Reflection (TIR) when the angle of incidence exceeds the critical angle for a glass of index 1.5. See Figure 17.9 on page 368.*

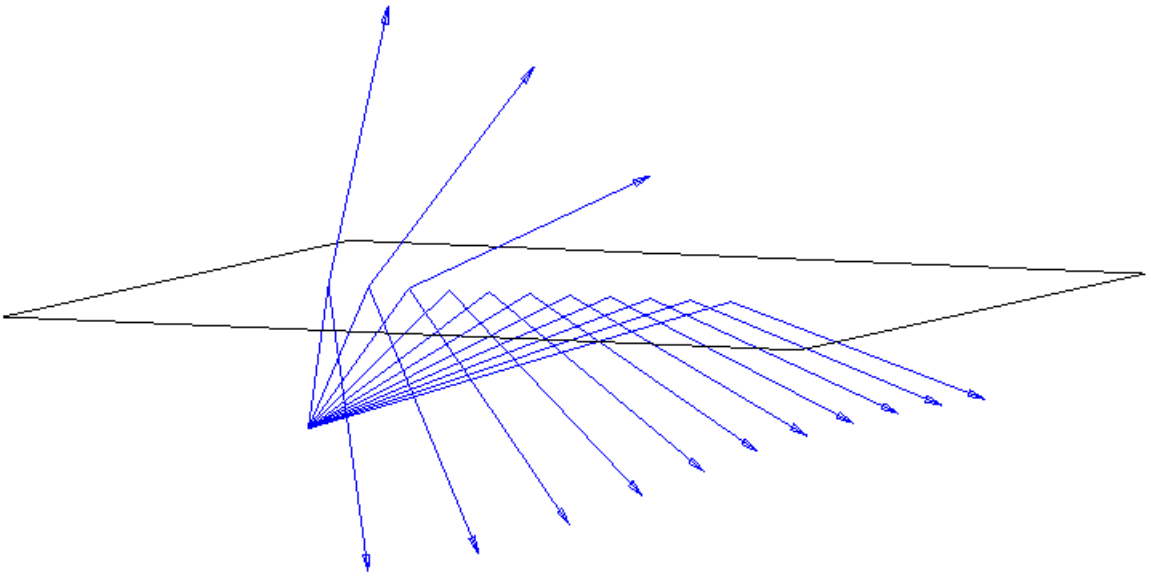


Figure 17.9 ASAP has correctly reflected rays by Total Internal Reflection (TIR)

APPENDIX

17A

SCRIPTS FOR CHAPTER 17

The following ASAP scripts are referenced in Chapter 17, “Ray Cessation—Stopping Rays Prematurely”.

Script 17-1

```
!!THE ASAP COMMAND GENERATED BY THE BUILDER FOR THE HALT COMMAND FOLLOWS:
HALT 12
```

Script 17-2

```
!!THE ASAP COMMAND GENERATED BY THE BUILDER FOR THE CUTOFF COMMAND FOLLOWS:
CUTOFF 1.E-18 1000
```

Script 17-3

```
!!!THE ASAP COMMAND GENERATED BY THE BUILDER FOR THE HALT COMMAND FOLLOWS:
HALT 12 -Z 1E-6
```

CHAPTER

18

BASIC RADIOMETRIC AND PHOTOMETRIC CONCEPTS

Our goal here is to define the terms you will need for subsequent chapters, and give examples of when specific radiometric analysis methods are appropriate.

This chapter offers a short introduction to, or review of, basic radiometry and photometry concepts. If you are new to optics and quantitative measurements of system performance, you may find this description useful. The treatment here is largely descriptive in nature. It is not necessary to have a deep mathematical understanding of all the integrals to use ASAP effectively. ASAP, after all, is just sorting rays and adding fluxes when it does these calculations for us. It is, however, important that you understand the terminology, and what questions we are trying to answer when we apply a particular command to a set of ASAP rays. The goal is to define the terms you will need for subsequent chapters, and give examples of when specific radiometric analysis methods are appropriate.

If you are already familiar with the definitions used in radiometry and photometry, you can probably skip this chapter.

A more rigorous mathematical treatment of radiometry is offered in the ASAP Technical Guide, *Radiometric Analysis.* Several excellent texts on the subject also exist, including *Introduction to Radiometry and Photometry* by Ross McCluney, *Radiometry and the Detection of Optical Radiation* by Robert W. Boyd, and *Introduction to Radiometry* by William L. Wolfe.

This chapter contains nothing specific about the way in which ASAP does radiometric and photometric calculations, except in the most general terms. No new ASAP commands are introduced. If you are already familiar with the definitions used in radiometry and photometry, you can probably skip this chapter. A discussion of how to do radiometric and photometric analysis with ASAP is left to Chapter 19 through Chapter 21.

Basic Definitions

Every analysis tool you apply in ASAP is ultimately trying to make a radiometric or photometric assessment of your virtual system within the context of Monte Carlo ray-trace simulation.

Radiometry is about quantifying and measuring electromagnetic radiation at any wavelength. Photometry is exactly the same thing, but is constrained by the response of the human eye. This sub-field of physics concerns itself with explicit mathematical definitions, terminology, and measurement techniques. Every analysis tool you apply in ASAP is ultimately trying to make a radiometric or photometric assessment of your virtual system within the context of Monte Carlo ray-trace simulation.

While the concepts involved in both radiometry and photometry are extremely simple, many students of science and engineering struggle with this topic. The trouble usually begins with terminology. Only four fundamental quantities are under consideration, but each has a name that offers few hints to its meaning:

Table 18.1 Four fundamental qualities of radiometry concepts

Radiant flux	Power
Irradiance	Power per unit area
Radiant intensity	Power per unit solid angle
Radiance	Power per unit area per unit solid angle

The number of terms immediately doubles when we consider photometry. Different terms are used to distinguish between the photometric and radiometric version of each definition, even though they are based on the same measurements. If the spectral response of the human eye is taken into account, the four quantities above become *luminous flux*, *illuminance*, *luminous intensity*, and *luminance*.

Table 18.2 Four fundamental qualities of photometry concepts

Luminous flux	Power
Illuminance	Power per unit area
Luminous intensity	Power per unit solid angle
Luminance	Power per unit area per unit solid angle

These new words also give little hint as to their meaning, and usually just have to be memorized.

A lack of consistency also exists in the use of the terminology. An example is the term *intensity*. This commonly used word in the English language was commandeered by scientists to mean something specific and technical. The only problem is a lack of consistent usage even by them. Some scientists use *intensity* as a short form of *radiant intensity*, or power as a function of direction. Others use it to mean irradiance, (power per unit area), sometimes adding the modifier *field* or *optical* when they do this, but sometimes not. Still others add the modifier *specific intensity* to mean *radiance*. While any school child can tell

you when something is “bright”, an optical engineer has to stop and think whether a source is “brighter” just because it has a higher “intensity”.

An even more serious source of confusion centers on the units used to quantify radiometric and photometric measurements. Sometimes, it seems as though every discipline has adopted a favorite set of units for publishing results. Common photometric and radiometric units include the *watts*, *lumen*, *candela*, *lux* and *nit*. (We will define each of these in the sections below.) But you may also be confronted with other units, like *candle*, *foot candle*, *Lambert*, *foot Lambert*, *phot*, *nox*, *stilb*, *skot*, *glim*, and two different kinds of *apostilb* (International and German Hefner), one of which is also know as a *blondel* (see McCluney, *Introduction to Radiometry and Photometry* for conversion factors).

Finally, confusion sometimes arises because radiometric and photometric terms are applied indiscriminately at both the source and the detector in a system. Some of the authors who develop these topics do so with emphasis on the source. In astrophysics, for example, all the radiometric quantities are essential to characterizing the surface of a star and inferring its physical properties. Observational astronomers use the same definitions, without modification, to characterize detector systems with sufficient sensitivity when observing that star from the earth. The definitions themselves are general. We can apply them at the source or destination of the radiated energy, or anywhere along the way.

If you are new to radiometry and photometry, try approaching this topic with two important principles in mind:

1. Only the terms and units are confusing. Use the fundamental descriptions of the quantities rather than the terms associated with them. For example, use “power per unit area” rather than “irradiance” or “illuminance”, since this basic description tells you much more about the measurement you are performing. Also, remember that when you see “power per unit area”, the implication is that the power must be changing over some area to make this quantity interesting and useful. We will follow this convention in this and the two chapters that follow, once we have introduced and defined the other terms.
2. The instruments used to measure radiometric and photometric quantities give insights into the nature of the quantity being measured. Try to visualize the actual experimental apparatus used to make the measurement. The parts that move, and the units that describe their position tell you what parameters are allowed to vary. While, in most cases, ASAP does not require that you model the measuring device to perform analysis, the methods employed by ASAP are likely to be closely related to, or at least analogous to the instrument. We give general descriptions of such devices and common applications in the sections that follow.

Power (Flux)

	Radiometry	Photometry
Quantity	Power or Radiant flux	Luminous flux
Units	Watt (W)	Lumen (lm)

A ray is not a photon; each ASAP ray carries flux, not energy.

In optics, the terms *flux* and *power* are synonymous and mean *energy per unit time*. Remember, a ray is not a photon; each ASAP ray carries flux, not energy. A ray represents the surface normal of a wave front that is delivering a steady flow of energy from a source, through intervening volumes and ultimately onto a surface. ASAP does not attempt to do "time domain" problems. All our results are presumed to represent a steady-state solution.

It is appropriate to ask about total flux from a source, when it makes no difference to us exactly where on the source that power came from, or what direction it was going when it left. An example is when we want an upper bound, just on the power that could possibly be coupled even into an ideal optical system from a given source. How would we measure this? Ideally, we would need a large spherical detector that fits around the source, with an inside surface sensitive to all wavelengths. In practice, we might use an integrating sphere, or an instrument with a single, moving detector that samples the power at all angles, and sums the results. In any case, the final answer is just one number: the total power in watts.

Similarly, we would be interested only in total power at a detector if the device were equally sensitive to radiation coming from all directions, with no variation in positional sensitivity over its surface. A simple, monolithic PIN photodiode (a device the converts light energy into an electrical signal), without imaging or concentrating optics, behaves something like this, at least over optical wavelengths. The voltage generated by such a device will be roughly proportional to the optical power falling on it.

Remember, however, that any practical detector does not respond to power at all wavelengths. A silicon PIN photodiode is generally sensitive only between 400 and 1100 nm. Furthermore, it is not equally sensitive to all wavelengths even within that band. An integrating sphere is not equally efficient at all wavelengths either. In practice, a device designed to measure total power needs to be carefully calibrated against a source of known spectral distribution to correct for this, and obtain a true "radiometric" result. If we are interested in the photometric quantity (luminous flux, instead of radiant flux), we have to make an additional (or different) correction to the result to make the instrument mimic the spectral response of the human eye. When this is done, the unit becomes *lumens* rather than *watts*. The details of the conversion are discussed below in the section, "Power per Unit Solid Angle".

We will discuss ASAP methods for measuring total flux in Chapter 19, *Analyzing Total Flux and Flux per Unit Area*.

Power per Unit Area

	Radiometry	Photometry
Quantity	Irradiance	Illuminance
Units	W/m^2	lm/m^2 = Lux (lx)

The irradiance and illuminance quantities are concerned with how flux is distributed spatially. For example, consider photographic film, or a CCD array. Like the PIN photodiode, both of these sensors do not care much about the direction of the light falling on them. Both, however, have localized sites, the film's grains or the CCD's pixels, that can respond independently to the amount of flux at that specific place. This response is what allows them to record the image behind a lens. Now we are exploring variations in flux per unit area, or stated another way, flux as a function of position.

When an illumination engineer is given the task of lighting a parking lot, the requirements generally consist of minimum and, perhaps, maximum illuminance levels on the ground, below the lighting system. These requirements can be confirmed, in practice, by using one or more detectors to sample a grid of points on the ground, to make a map of flux as a function of position (Figure 18.1 on page 376). A measure of irradiance is expressed in watts per meter squared. If the response of the eye is considered, the unit becomes *lux*. The process for converting from watts per meter squared to lux is described under "Power per Unit Solid Angle" on page 377.

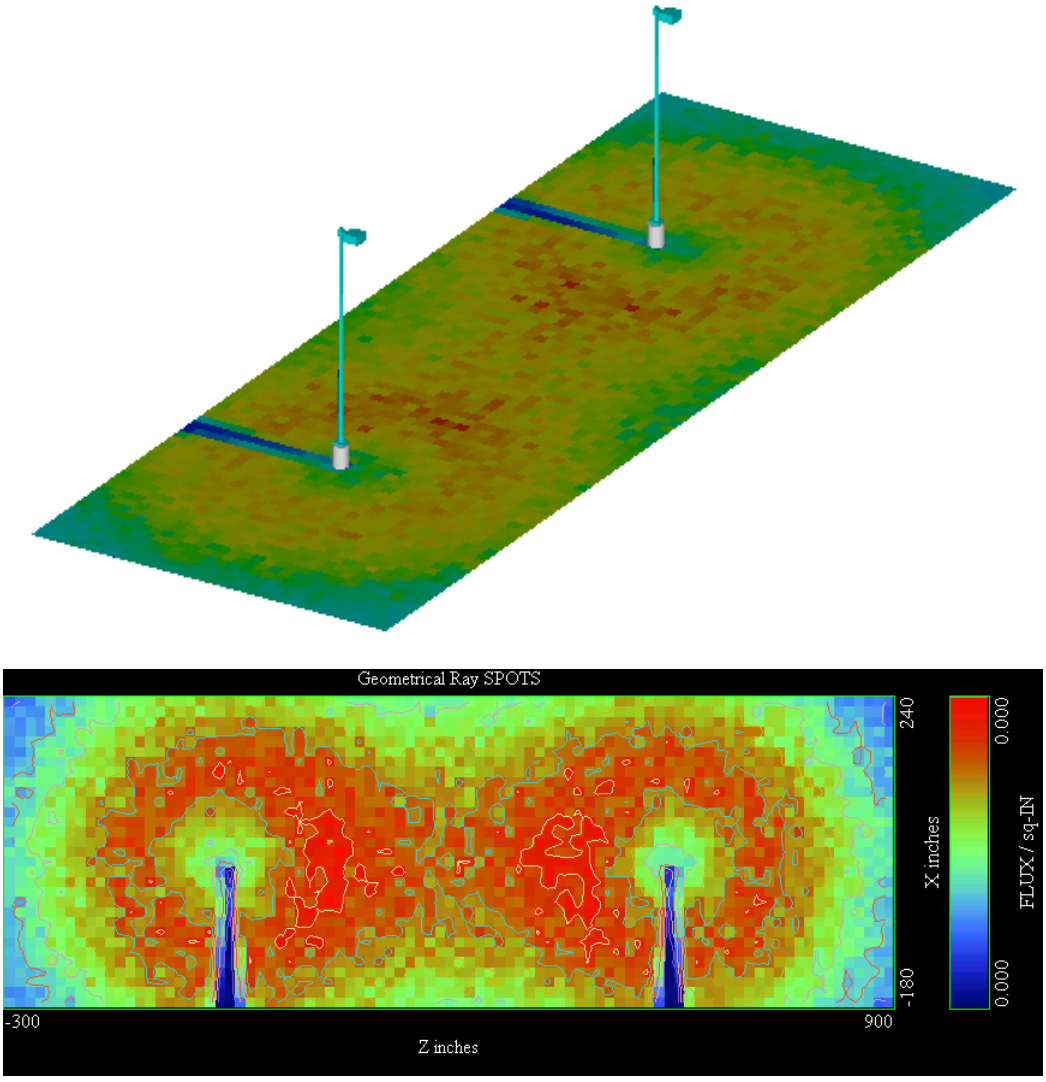


Figure 18.1 Flux per unit area (irradiance or illuminance) can be used to investigate the effectiveness of an outdoor lighting system. (Top) Two lighting fixtures, with the illumination pattern on the ground below. The surface of the ground is divided into a grid of squares, and the flux within each square is indicated by its color. (Bottom) Overhead view of the same information, with iso-flux contours drawn over the pseudo-color map.

We will discuss ASAP methods for measuring and displaying flux as a function of position in Chapter 19, “Analyzing Total Flux and Positional Flux Distributions”.

Power per Unit Solid Angle

	Radiometry	Photometry
Quantity	Radiant Intensity	Luminous Intensity
Units	W/sr	Candela (cd) = Lumens/sr

Radiant and luminous intensity are concerned with flux as a function of direction. In this case, we do not care where on an extended source the flux originated, or how the power is distributed on the surface of the detector. We care only about direction.

The purpose of an automobile headlamp system is to concentrate light from the lamp into a forward beam. Usually, regulations exist about where that light can and cannot be directed. For example, the specifications restrict the amount of light that has the potential to be directed into the eyes of an oncoming motorist. The luminaire designer must carefully limit the amount of flux that is directed above the horizon and in the forward direction. Now we are interested in flux as a function of direction. We need to produce a map of flux as a function of direction (Figure 18.2).

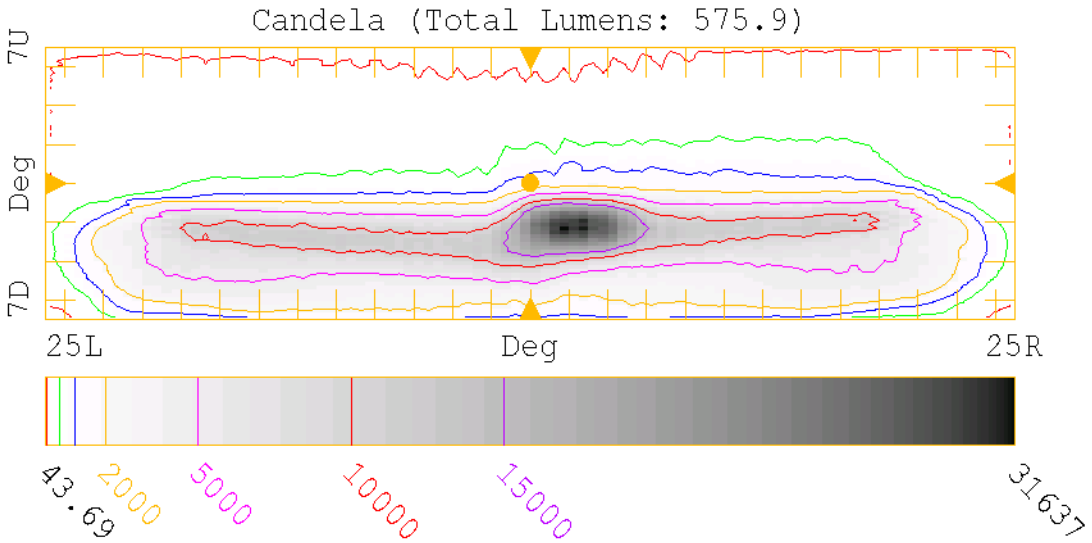


Figure 18.2 Flux per unit solid angle (radiant or luminous intensity) is a map of flux versus direction. This figure shows a luminous intensity distribution for an automobile headlamp design, covering angle space between ± 25 degrees left and right, and ±7 degrees up and down. This analysis contains no information about where on the headlight reflector or source the radiation originated. Figure 18.2 was produced by ReflectorCAD™, a Breault software program, used for designing segmented reflectors. ASAP produces similar graphical displays with the **Picture Viewer**.

Investigators make measurements of flux as a function of angle. They use a device called a goniometer or goniophotometer, shown schematically in Figure 18.3.

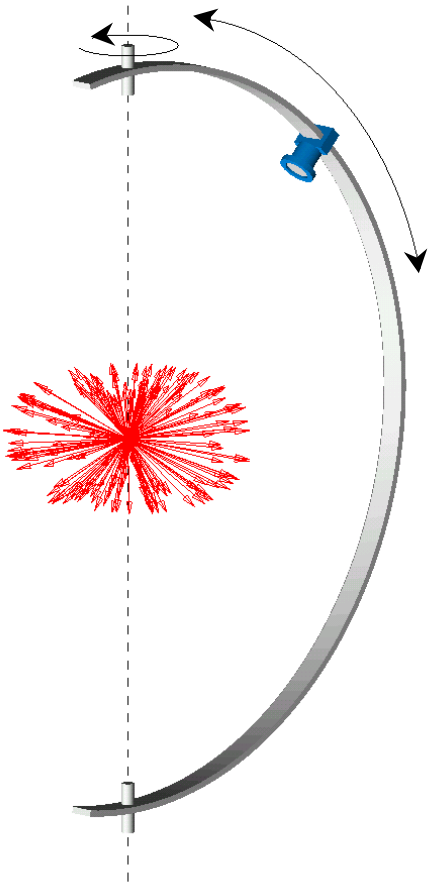


Figure 18.3 Basic concepts behind a goniometer or goniophotometer. A detector is mounted in such a way that it can be moved in both altitude and azimuth to sample the power of a source as a function of direction over the full direction sphere. Other possible mechanical arrangements exist. In many cases, it is more convenient to rotate and tip the source while keeping the detector fixed, but the principle is the same.

Although there are a variety of mechanical arrangements for these devices, the ultimate goal is to sample power as a function of direction. The distance to the detector generally needs to be large, compared to the size of the light source. Ideally, the detector is located at an infinite distance, where all radiation appears to come from a point. Measurements from a great distance give rise to the

concept of a far-field distribution and the far-field sphere. In a practical device, we generally settle for a distance that is at least ten times the dimensions of the source. You will see in Chapter 21, "Analyzing Directional Distributions" that it is not necessary to trace rays to the surface of a distant sphere to explore the angular distribution of rays. The rays already have a direction as soon as they have refracted or reflected the last time.

The conversion from radiant intensity to luminous intensity is the fundamental point of departure from absolute radiometry to photometry. All other relationships between radiometric and photometric quantities are derived from this fact.

A candela is the luminous intensity of a monochromatic source radiating at 556 nm with a radiant intensity of 1/683 watts per steradian in a given direction.

A candela is the luminous intensity of a monochromatic source radiating at 556 nm with a radiant intensity of 1/683 watts per steradian in a given direction. While this modern definition seems arbitrary, its roots go deep into the history of radiometry and photometry. It was originally based on an actual candle, later refined to reference a specific laboratory blackbody source, and finally defined, as stated above, to retain some consistency with the older conventions. Once we specify this point of departure, all other conversions are done based on the response of the eye relative to this reference wavelength. A lumen, for example, is the luminous flux emitted into a solid angle of one steradian by an isotropic point source with one candela of luminous intensity.

We will discuss ASAP methods for measuring and displaying flux as a function of direction in Chapter 21.

Power per Unit Area per Unit Solid Angle

	Radiometry	Photometry
Quantity	Radiance	Luminance or Brightness
Units	W/m^2-sr	lm/m^2-sr = cd/m^2 = nit

Radiance and luminance (also knows as *brightness*) are concerned with flux as a function of both direction and position. Now we are concerned with where the radiation comes from on a source or arrives on the detector, and also the angle at which it exits or arrives.

Radiance and luminance are important because they describe what an object looks like to the eye or to an imaging system. If you look at the display on a laptop computer, the details of the image that forms on the retina of your eye depend on both where the light originated on the screen and the angle at which you view it. Stated another way, any attempt to quantify the performance of such a device has to specify the spatial coordinates on the screen and your viewing angle.

A camera and the human eye are probably the most common instruments used to measure radiance and luminance. Both use an imaging lens to map the surface of the object onto the detector. This lens also defines the size and location of the solid angle. While the film, CCD, or the eye's retina is actually measuring flux as a function of position, you will get a different picture depending on where you stand. An imaging goniophotometer might be used to obtain thousands of images from different angles to characterize the radiance or luminance of an object.

Radiance and luminance are the most challenging quantities to predict by Monte-Carlo ray-trace methods, because a large number of rays must be traced to obtain useful results. To obtain a single quantitative value, we must limit ourselves to the rays that are located in a particular place, and then further restrict ourselves to rays directed into a finite solid angle centered on a specific direction.

Figure 18.4 shows one such result. A model of an automobile tail light reflector is shown as it might appear from a specific viewing angle.

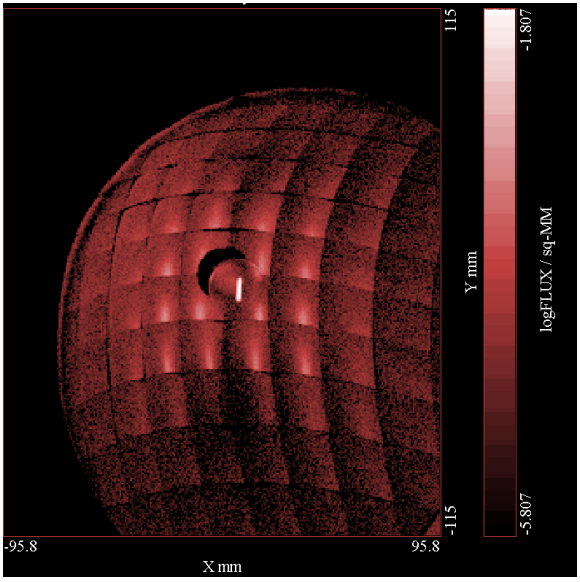


Figure 18.4 A luminance result is shown, depicting the distribution of power on a segmented reflector at one particular viewing angle. In ASAP, this analysis is termed, "View-Driven Radiometric Analysis" (VDRA). It is not strictly "lit appearance" because the solid angle admitted is a cone with five-degree half angle, significantly larger than that admitted by the human eye at a reasonable distance. This example comes from the tail-lamp sample project (**asapxxxx\projects\samples\TailLamp**).

For best results, you must have an excellent model of the extended source illuminating the reflector, and knowledge of the scattering properties of all visible surfaces on the source and luminaire.

We will discuss ASAP methods for measuring and displaying flux as a function of both position and direction in Chapter 21, "Analyzing Directional Distributions".

Summary

The following table summarizes the radiometric and photometric quantities discussed in this chapter:

Quantity	Radiometry	Photometry
Power or Flux	Power or Radiant Flux Units: watt (W)	Luminous Flux Units: lumen (lm)
Power per unit area	Irradiance Units: Watts/meter^2 (W/m^2)	Illuminance Units: lux = lumens/meter^2 (lx = lm/m^2)
Power per unit solid angle	Radiant Intensity Units: Watts/steradian (W/sr)	Luminous Intensity Units: candela = lumens/ steradian (cd = lm/sr)
Power per unit area per unit solid angle	Radiance Units: Watts/ meter^2-steradian (W/m^2-sr)	Luminance or Brightness Units: Nit = lumens/meter^2-steradian (nit = lm/m^2-sr = cd/m^2)

CHAPTER

19

ANALYZING TOTAL FLUX AND POSITIONAL FLUX DISTRIBUTIONS

In this chapter, we will introduce new ASAP analysis commands that allow you to investigate how energy is distributed as a function of position within your system model. Using the terminology introduced in the previous chapter, we will use ASAP to compute the radiant or luminous power on various objects in our system, and the irradiance or illuminance (power as a function of position). You will see that each of these operations involves just a single pass through the ray table, during which time ASAP sorts the rays according to their positions. For the irradiance or illuminance calculations, ASAP creates a distribution file, which you can then analyze graphically or numerically.

Reviewing Basic Analysis

In Chapter 11, we introduced basic analysis concepts. These are worth reviewing now, since they apply to every task we will perform both here and in the next chapter. Always keep these two concepts in mind:

1. Unless instructed otherwise, ASAP analysis calculations are performed using all rays on all objects.

   You will frequently need to isolate a subset of rays before proceeding with your analysis. Isolation was performed with the **Choose Rays** dialog box, available on either the **Rays** or **Analysis** menus. You can use the **Consider** option to isolate rays on certain objects, and **Select Rays** to further isolate the rays according to the source in which they were created, the object with which they just interacted, and many other criteria.

2. Analysis calculations are performed with the positions, directions, fluxes, and so forth that the rays have at the time the analysis command is issued.

As this implies, we can perform an analysis at any time. You do not have to trace rays completely through your system—or at all—before you perform an analysis on the rays. As we have seen in Chapter 9, doing some analysis as soon as the sources are created helps you determine whether you have created the intended rays.

Total Flux—Sorting by Object

Every radiometric or photometric analysis always starts with a pass through ASAP ray data, sorting the rays according to one or more parameters.

Recall that ASAP keeps all the information about rays in a binary file called **virtual.pgs** (see the sidebar, "ASAP Ray Details" on page 133 in Chapter 7). This file contains information like the object on which each ray is currently located, its global position coordinates (*x, y, z*), direction cosines (*a, b, c*), flux, and other information useful during tracing and analysis. Every radiometric or photometric analysis always starts with a pass through this ray data, sorting the rays according to one or more of these parameters. In the case of radiant or luminous power, the sort is performed by object. We want to know the total flux of all rays on a given object. If you had to do this yourself, you could begin by creating a table of all the rays using **Rays> Ray Information> List Ray Data** from the ASAP menu bar, and selecting **Positions** in the dialog box. The result appears in the Command Output window.

```
--- LIST POSITION

   Ray        X           Y           Z          Flux      Object     OPL

     1 0.228751      2.76662     -15.0884     0.200E-03       282  0.32792872
     2 0.251000      3.84960     -10.8653     0.141E-03       278   7.0502343
     3 0.251000     -3.39779     -5.83922     0.168E-03       278   18.201050
     4 0.251000     0.982077     -8.79764     0.180E-03       278   10.434788
     5 0.251000      2.28862     -7.05577     0.182E-03       278   13.010703
     6 0.251000     -1.41388     -9.75946     0.167E-03       278   10.553659
     7 0.251000      3.11390     -8.91536     0.149E-03       278   12.820347
     8 0.764849E-01  2.13273     -15.1016     0.191E-03       282  0.59069556
     9 0.171840      2.36452     -15.2510     0.200E-03       281   1.1291233
    10 0.251000     -.676680     -1.05815     0.184E-03       278   25.363649
```

Figure 19.1 An ASAP table in the **Command Output** window showing the flux for all the rays.

This table shows only ten rays. In most cases, there would be many more rays, and we would have to keep a running tally of the flux on each object. Fortunately, ASAP can do this sorting and summing for us. The command ASAP uses is **STATS**, which is short for "statistics". We used this command in Chapter 11 when we began our post-trace analysis with **Rays> Locate Rays**. The same command is also available on **Analysis> Calculate Flux** and selecting **Summary**.

Figure 19.2 shows a device for which you might need to calculate the total power on each object in your system after a ray trace.

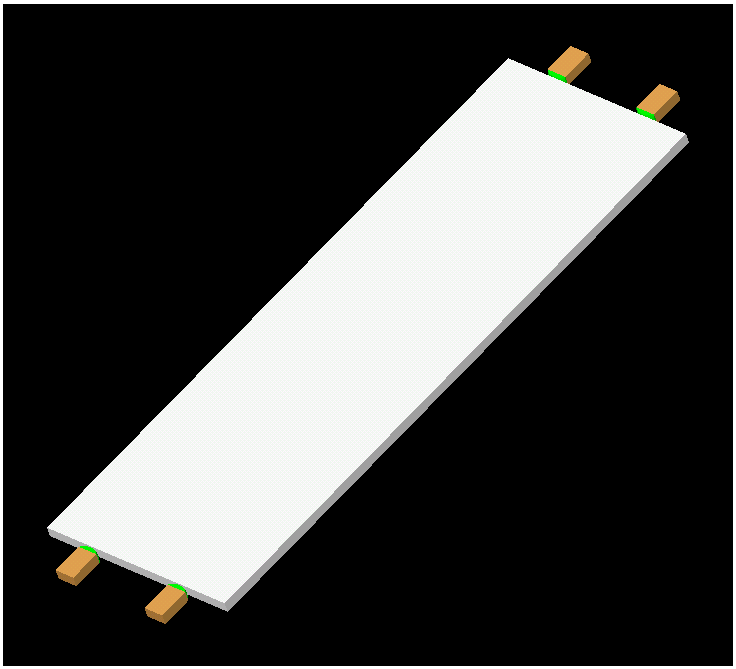


Figure 19.2 A backlight panel, used to provide uniform illumination behind a liquid crystal display. Such devices are used in cell phones and other electronic displays that must be read in the dark.

This model of a backlight shows a thin, plastic panel illuminated from the sides by four light-emitting diodes. Light proceeds through the device by total internal reflection (TIR), but prisms divert the light out and perpendicular to the front face in a controlled way. One of the design goals of such a device is to couple the maximum amount of light into the panel from the LEDs.

Once the ray trace is complete, we can let ASAP sort the rays by object to find out how much power stopped on (or escaped through and "missed after") each object in the system. The results, generated from **Analysis> Calculate Flux** with the **Summary** option, appear below.

```
--- STATS
Object    Rays     Flux
   1     353034   4.06701       DISPLAY.BOTTOM
   2      34072   1.06418       DISPLAY.FRONT
   3      33926   1.07769       DISPLAY.BACK
 276      18442   1.03600       DISPLAY.LEFT.EDGE
 277      18262   1.01388       DISPLAY.RIGHT.EDGE
 278     573689   35.1097       SAMPLE_SURF
 279       1415 0.938571E-01    LED.1.SIDES
 281      59278   4.53944       LED.1.BACK
 282      46054   4.30742       LED.1.DOME
 283      59133   3.48255       LED.1.DOME.SIDES
 284       1463 0.100568        LED.2.SIDES
 286      59728   4.56923       LED.2.BACK
 287      45672   4.27652       LED.2.DOME
 288      59166   3.49311       LED.2.DOME.SIDES
 289       1420 0.942260E-01    LED.3.SIDES
 291      59110   4.54830       LED.3.BACK
 292      45531   4.25911       LED.3.DOME
 293      59211   3.47768       LED.3.DOME.SIDES
 294       1441 0.989486E-01    LED.4.SIDES
 296      59032   4.53675       LED.4.BACK
 297      45916   4.29422       LED.4.DOME
 298      59017   3.46881       LED.4.DOME.SIDES
 -------------------------
TOTAL   1694012   93.0092
```

Figure 19.3 Results of ray sorting in ASAP to determine how much power stopped or escaped, by object.

We can learn that, while the four LEDs produce a total of 100 mW of power, only about 35 mW reached the exit surface (named **SAMPLE_SURF**). A small fraction of the power leaked out of the display through other surfaces, but rays that never coupled into the panel account for most of the losses in this simple design.

Flux versus Position—Sorting by Position Coordinates

What if, in addition to knowing how much power arrives at a surface, we also need to know how it is distributed there? We got close to this result in both Chapter 9 and Chapter 11 when we used **Rays> Graphics> Plot Positions 2D** to plot ray locations within a designated window. The resulting "spot diagram" showed graphically how the rays were distributed on a detector surface. While this plot was useful for investigating image quality, it can be misleading when you are trying to perform quantitative radiometry or photometry. It contains no information about the *flux* of each ray. One "spot" looks much like another in the graphic, though they could differ by many orders of magnitude in power. The methods used in this chapter are closely related to the spot diagram, but also take the flux of each ray into account.

As an example of flux distribution calculations, consider again the backlight example in “Total Flux—Sorting by Object” on page 384. Another design goal of such a device would be to make the distribution of light on the sample surface as uniform as possible. Now we are interested in power as a function of position, termed irradiance (for radiometric calculations) or illuminance (for photometry). While this calculation is not quite so simple for ASAP as sorting flux by object, it is still just a matter of passing through the ray table, looking at the spatial coordinates of each ray, and sorting the associated flux according to these coordinates. In this way, ASAP can make a map of power per unit area on a surface.

What do we need to tell ASAP so that the program can sort rays according to their spatial location? ASAP needs to know:

- which rays to sort, since you are not usually interested in all the rays on all the surfaces;
- the area or “window” over which to perform the sorting to further limit the area of interest.
- the spatial resolution you want to achieve. Ultimately, we will accumulate the ray flux into discrete bins, or “buckets”, distributed over the surface. We must specify the size of these bins.

You already know how to isolate rays for analysis. The tools for doing this were first introduced in Chapter 11. From the main menu, select **Analysis> Choose Rays> Consider** to isolate only those rays on one or more objects. In the LED backlight example, we would consider only **SAMPLE_SURF**, since that is where we want to calculate the power distribution. If we wanted to see the power from only one of the four LED sources, we could use **Analysis> Choose Rays> Select**, to further limit the rays involved in the sorting.

Everything else is done using **Analysis> Calculate Flux Distribution**. When you select this command from the menu, the dialog box shown in Figure 19.4 appears. Once you have filled in the appropriate information and click **OK**, ASAP performs the actual sort.

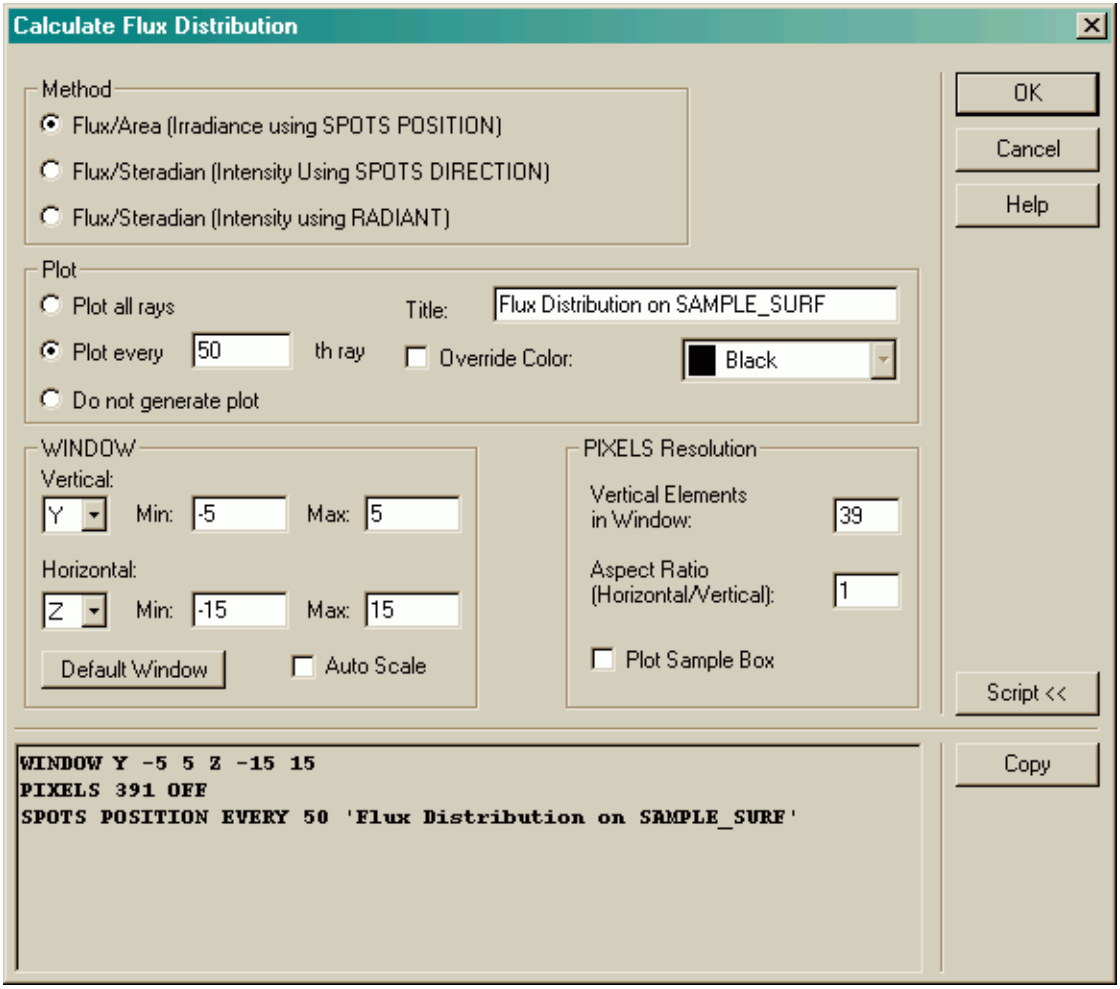


Figure 19.4 The dialog box, **Calculate Flux Distribution** is one of the most elaborate in ASAP. It collects the necessary information for three different ASAP commands: `WINDOW`, `PIXELS`, and `SPOTS`. The `WINDOW` and `PIXELS` commands establish parameters that ASAP needs to perform the binning of the data with `SPOTS`.

The **Method** (top left area in the dialog box) is **Flux/Area (Irradiance using SPOTS POSITION)**. As the name of the method implies, and as we already noted above, this activity is closely related to making a spot diagram.

You can view the ASAP script generated from selections in a dialog box by clicking its **Script** *button.*

Figure 19.4 also shows the actual commands that the ASAP kernel runs. You can display the script from this and any ASAP dialog box by clicking its **Script** button.

Although we did not mention it in our previous encounters with spot diagrams, the single ASAP command, `SPOTS POSITION` can perform the sorting and binning of the ray data, as well as producing the spot-diagram graphic. The dialog box includes the option for making the plot of the spot diagram or suppressing it.

> ***Note:*** If you have a large number of rays, it may take a long time to produce a spot diagram. The graphic is, in fact, much more computer intensive than the actual sorting of rays into bins. Further, the spot density might be so high that little information is conveyed by the graphic. Consider using the **Plot every [n]th ray** or **Do not generate plot** options when you perform this analysis on a large number of rays.

Window coordinates allow us to map the distribution over the entire surface—whether it contains rays or not, and also give us complete control over the resolution.

Before ASAP can perform the actual ray binning, we still need to specify the window and resolution. In our example, **SAMPLE_SURF** is a Y-Z plane, so this is the correct choice to enter in the **WINDOW** area of the dialog. If we checked the **Auto Scale** box, we would get a window just large enough to include all the rays located on **SAMPLE_SURF**. Under many circumstances, you may prefer to explicitly specify the exact window coordinates, as we have done. These coordinates allow us to map the distribution over the entire surface—whether it contains rays or not, and also give us complete control over the resolution.

The ASAP command that sets the resolution is called `PIXELS`. You enter the necessary information in the **PIXELS Resolution** area of the dialog. All that is required is an integer value (default 39) that sets the number of bins (or pixels) into which the window will be divided in the *vertical* direction.

Note once again that ASAP specifies the vertical direction first in the window, which is somewhat non-standard. We have specified the number of pixels in the Y direction, as seen in Figure 19.5. ASAP automatically calculates the number of horizontal pixels needed to fill the specified window with square pixels.

Since ASAP must use an integral number of pixels in both directions, the column of pixels on the extreme left and right may somewhat overlap the window. You can force rectangular pixel dimensions by changing the **Aspect Ratio** value, which is the ratio of horizontal-to-vertical-pixel dimension (default **1**).

Figure 19.5 shows the resulting pixels drawn on top of the spot diagram. ASAP now has the task of summing the flux of each ray that falls within a given pixel.

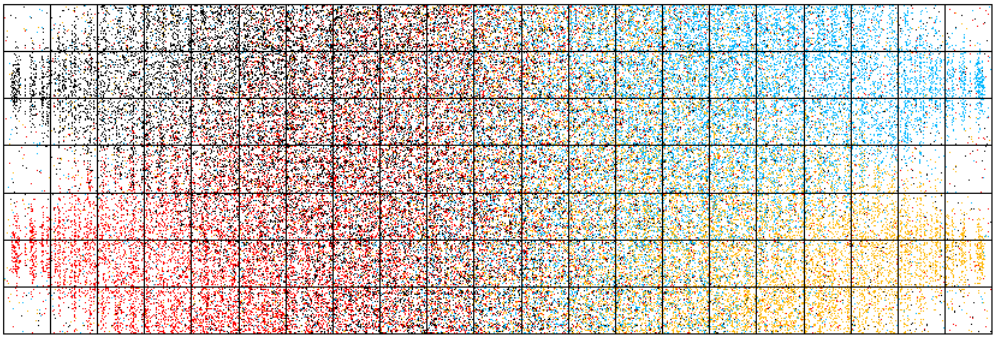


Figure 19.5 This 10 x 10 **Plot Viewer** window was divided into seven vertical pixels. ASAP then automatically generated 21 horizontal pixels to fill the entire window. If an integral number of pixels can not be fit in the horizontal direction, ASAP allows the left and right pixel columns to overlap the window, even though no rays lying outside the window are considered. Note that ASAP does not provide a tool for drawing these grid lines. The plot was produced by creating a series of edge-based lines, and overlaying them on top of the spot diagram using **PLOT EDGES**.

How many pixels should you choose in a particular situation? We have implied that this choice is just a matter of picking the desired resolution. Good resolution is always desirable, so why not pick the largest number allowed in your version of ASAP? An unavoidable consequence of Monte Carlo ray-trace simulations is accuracy limitations due to practical restrictions on the number of rays we can trace. These limitations cause statistical variation in the results, which only worsen when you sort too few rays into too many bins. We will return to this topic again when we discuss the **AVERAGE** command, and also in the sidebar, "How many rays, and how many pixels?" on page 449 in Chapter 20.

> ***Note:*** The total number of pixels available to you varies depending on your version of ASAP. The number also tends to increase over time as new versions of ASAP are introduced in response to more powerful computers. You can see the limits of your version by running **DIMENSIONS** from the **Command Input** window. Be sure to use only upper-case letters. Check the value labeled **Sampling Pixels on a Side**.

The choices offered within this dialog box appear to limit us to performing analysis on planar surfaces, perpendicular to one of the three global coordinate axes. This limitation is, in fact, the case when the dialog box is used. In this situation, ASAP sorts the rays according to only two of their three Cartesian coordinates (the two named in the **WINDOW** command). The third coordinate is ignored. As a result, we always perform this type of analysis by projecting all currently considered and selected rays into the specified window, regardless of their "depth" coordinate, and that window is always one of the Cartesian planes. It is still possible to perform an analysis on tipped or skewed planes, however.

The procedure is described in the sidebar, “Distribution on a Tipped Detector” on page 398. You can also perform the analysis on a planar surface associated with the normal direction at the reference point of a chosen object. Use the **AXIS LOCAL** command, with a window override on the analysis command, which is explained in online Help.

Distribution Files

If you do not specify a plot of the spot diagram in the **Calculate Flux Distribution** dialog box, ASAP may work for several minutes (if you have a large number of rays), and finish with no apparent result. Much work has been accomplished, however; ASAP has produced a *distribution file*. The only indication of this activity is a summary that appears in the Command Output window:

```
--- SPOTS POSITION ATTRIBUTE 0
Distribution of data within:
    Window Vertical: Y = -5.00000     to  5.00000      ( 10.0000    )
          Horizontal: Z = -15.0000     to  15.0000      ( 30.0000    )
Opening NEW     distribution file   9: C:\ASAP Work\ASAP Primer\bro009.dat
    Rays    Flux
  573686  35.1097
 Maximum =  0.3052285     minimum =  0.0000000E+00
```

Figure 19.6 Summary of a distribution file in the **Command Output** window.

The binary distribution file consists of header information and a two-dimensional array of floating-point numbers, representing the power within each pixel.

Among other things, this output shows that ASAP has created a file in your working directory named **bro009.dat**. This is the distribution file. It is a binary file consisting of header information and a two-dimensional array of floating-point numbers, representing the power within each pixel.

Viewing the Results

ASAP provides us with an extensive set of tools for working with distribution files. They allow us to display and further process the distribution data. Although in this chapter we are interested in the tools in the context of irradiance and illuminance (power as a function of position), distribution files and the display tools are general concepts that permeate the ASAP program. We will go through most of these systematically in the next chapter. For now, however, we can briefly dip into these tools to have a quick look at the distribution file created above. The graphic, produced from **Display> Graphics> Picture**, appears in Figure 19.7.

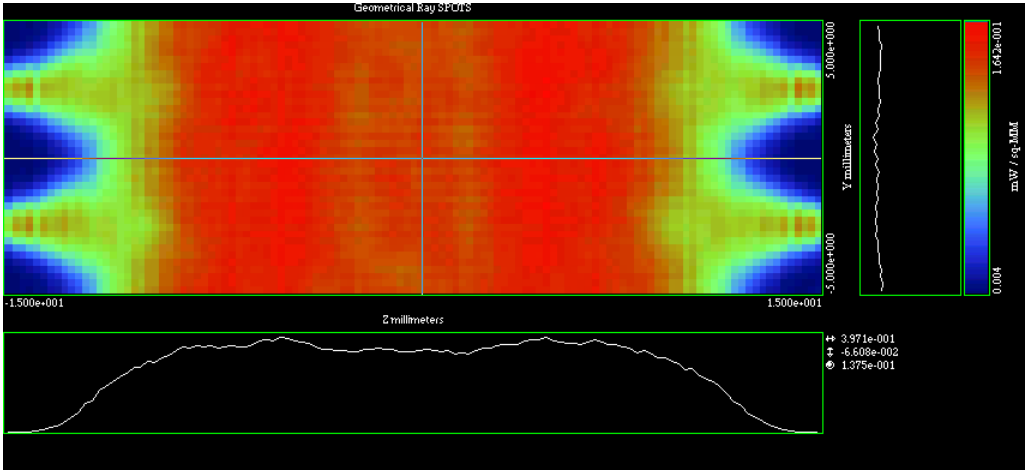


Figure 19.7 The **PICTURE** command creates a pseudo-color map of the distribution file. This distribution was created using 39 vertical pixels (the default value). ASAP provided 117 horizontal pixels that, in this case, exactly filled the window.

The view appears in the Display Viewer, a window that we have not yet encountered in this *Primer*. The default view features a pseudo-color map of the data distribution generated by the **SPOTS POSITION** command and cross-sectional graphs through one vertical and one horizontal section of the data. We will have much more to say about the Display Viewer and the other display tools in Chapter 20.

Summary

In this chapter, we have discussed the following new commands.

Command	Menu	Description
STATS	**Analysis> Calculate Flux**	Sort rays by object, and sum the fluxes on each.
CONSIDER SELECT	**Analysis> Choose Rays**	Specify objects (and the rays on them) or rays (according to other criteria) for analysis.
WINDOW PIXELS SPOTS POSITION	**Analysis> Calculate Flux Distribution**	Similar to **Rays> Graphics> Plot Positions 2D**, except a distribution file is created, and the graphics are optional.
DIMENSIONS		Displays a table of maximum array dimensions for the most important ASAP arrays, depending on the version you own.
DISPLAY PICTURE	**Display> Graphics> Picture**	Displays a distribution file in the Display Viewer window.

The topic of this chapter is using ASAP to analyze two fundamental radiometric and photometric analysis quantities:

1. Total radiant or luminous power on each object in a system model.
2. Power as a function of position (irradiance or illuminance).

Computing either of these quantities involves a single pass through the ray table (**virtual.pgs**). The total power calculation is simply a sort by object number, adding up the flux of each ray on a given surface. This task is performed via the menus using **Analysis> Calculate Flux**. Calculating power as a function of position is a three-step process:

- Isolate the rays you wish to analyze from **Analysis> Choose Rays**.
- Sort the rays according to their position, specifying the window in which to perform the analysis and the desired spatial resolution. You perform this step from **Analysis> Calculate Flux Distribution**. The result is a distribution file, stored in your ASAP working directory as **bro009.dat**.
- View the result. Many options are available to you within the **Display** tools. We introduced only **Display> Picture**, which produces a pseudo-color map and graphical cross sections through the distribution file. Many other graphical and processing tools designed exclusively for working with distribution files are introduced in Chapter 20.

Exercise 9: Simple Flashlight Model

Build a simple model of a flashlight using a parabolic reflector and an emitting helix. Direct the flashlight in the z direction. Look at the power distribution as a function of position on a wall located a short distance away from the flashlight.

1 Using the Builder, begin defining a new system. You may choose your own units, but we will be giving all spatial dimensions in centimeters with flux units in Watts. Define a 100% reflective coating for use with the parabolic reflector defined below, and an absorbing coating for the wall.

2 Model a parabolic reflector from **Surfaces> Optical** with the following parameters:

Vertex location: z = -10 cm

Vertex radius of curvature*:* 5 cm

Conic constant: -1

Diameter: 40 cm

3 Create a rectangular wall from **Surfaces> Plane> Axis** with the following parameters:

Location: z = 200

Dimensions: 400 × 400 cm

4 Create an emitting helix from **Sources> Emitters> Helix** with the following parameters:

Orientation: Along the y axis

Total length: 2 cm centered at the global origin

Helix diameter: 0.4 cm

Wire diameter: 0.04 cm

Number of turns: 10

Number of rays: 100,000

5 Use **Rays> Ray Modifiers> Flux Total** in the Builder to set the total power of the source at 2 Watts.

6 Run the Builder file to create the geometry and source. Watch for the ASAP prompt to return on the status bar before continuing.

7 Verify the source from **Rays> Graphics> Plot positions 2D**, selecting the **Y-Z** window. Check the length and diameter of the helix by holding down the **Shift** key and moving the mouse to read the position within the Plot Viewer.

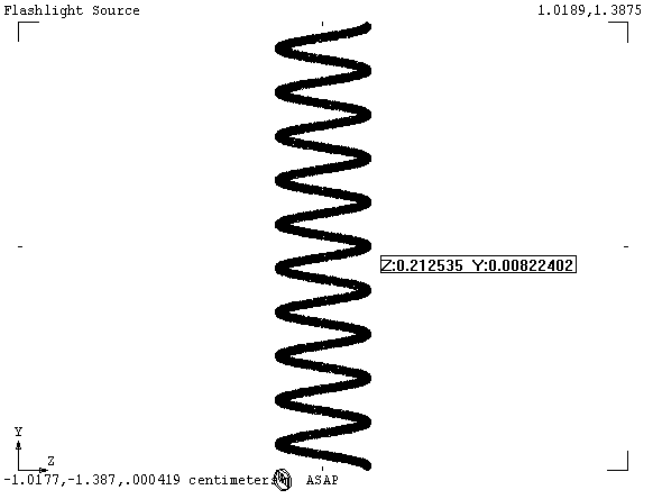


Figure 19.8 Verifying the location and dimensions of the source

8 Verify the relative positions of the source and the reflector. From **System> Profiles** (Y-Z window), create a profile view of the reflector and wall. Set the **Pixel Resolution of Plot** to 501 to improve the quality of the graphic, and select **Overlay Next Plot** to keep the plot window open for more data. Then use **Rays> Graphics> Plot positions 2D** again to plot the ray positions in the same window. Plot only every tenth ray. Check the positions of the emitting helix relative to the reflector and the wall, using the cursor in the Plot Viewer. Zoom as required.

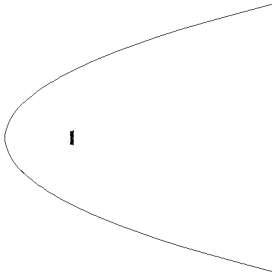


Figure 19.9 Verifying the location of the source with respect to the reflector

9 Rewind the vector file to remove old 3D information from previous graphics.

10 Trace the rays, plotting every 100th ray overlaying **Plot Facets**. View the results in the 3D Viewer.

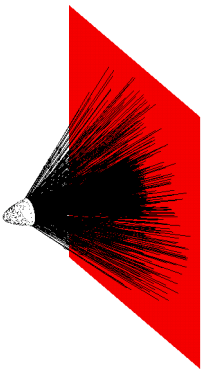


Figure 19.10 Overlaying the ray trace on the geometry

11 From **Analysis> Calculate Flux,** select the **Summary** option to verify that all the power reached the wall.

12 From **Analysis> Calculate Flux Distribution**, create a distribution of the flux as a function of position (irradiance). Select a Y-X window the same size as the wall. Create a spot diagram at the same time, plotting only every 10th ray.

13 From **Display> Graphics> Picture** to display the result.

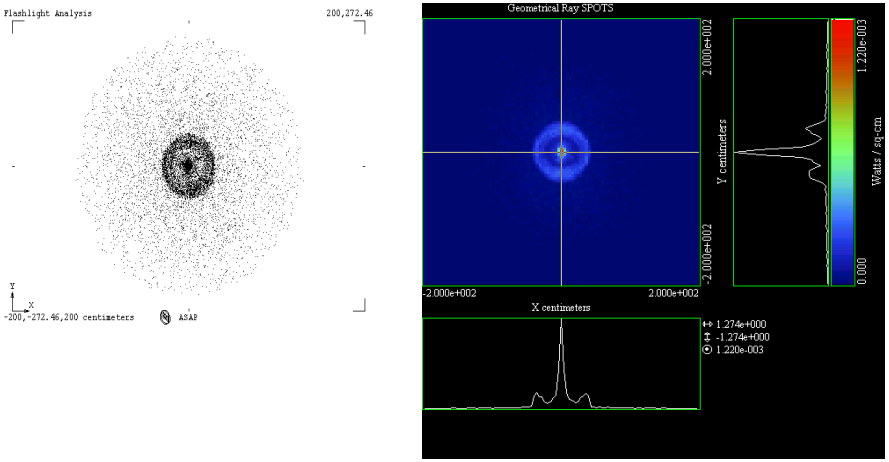


Figure 19.11 Graphical results of calculating the flux distribution. The figure on the left shows the spot diagram, which shows only the distribution of ray positions without any indication of the flux of each ray. The figure on the right shows the flux-weighted picture of the ray distribution.

Hints

- See Chapter 3 to review setting system units and coating properties, if needed.
- Note that we have not specified a wavelength, nor defined any media. The only optical element in this system is reflective, so we expect neither refraction nor wavelength-dependent behavior. As a result, no wavelengths or media need to be specified in this case.
- We have specified all dimensions in terms of diameters and full widths. Remember that ASAP expects dimensions expressed as *half* diameters and *half* widths!
- In this case, it is not necessary to use **Analysis> Choose Rays** to isolate rays for analysis, because all the rays reached a single object in the system.

Distribution on a Tipped Detector

The 3D view reproduced in Figure 19.12 shows a simple situation in which rays have been collected on a tipped surface. The detector has been rotated so that its surface normal is 45 degrees to the z axis. When we are forced to select a window that is normal to one of the coordinate axes, the resulting spot diagram in Figure 19.13 and the distribution file do not show the result in the reference frame of the detector.

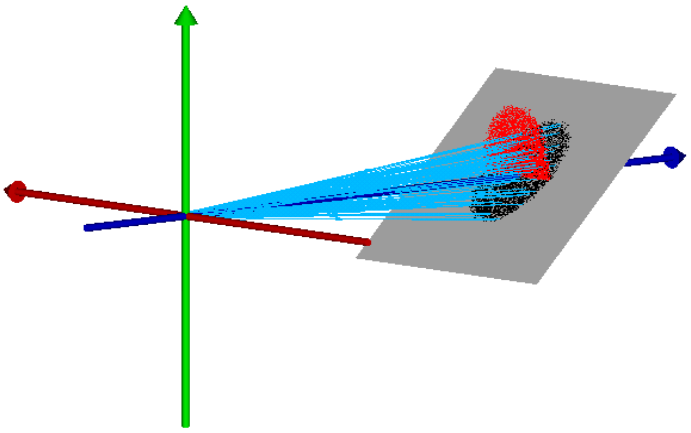


Figure 19.12 3D view of the ray trace and spot diagram overlaid on the geometry. The black dots show the original spot diagram, while the red dots show the spot diagram after rotating the rays to see the spot diagram in the reference frame of the detector.

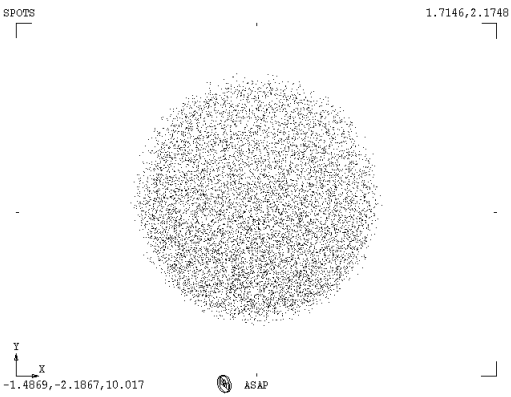


Figure 19.13 The original spot diagram

This situation is easy to remedy. While we cannot rotate the analysis window, we can temporarily rotate the rays. Recall that the rays have coordinates and directions.

The positions and direction vectors can be rotated through any angle, about any point, as though they were elements of geometry.

We need to rotate them in the same way that the detector plane was rotated, changing only the sign on the angle. This particular plane was defined in the Builder as shown in Figure 19.14.

*	Type	Cmd	Axis	Degrees	Y	Z	List			
OBJ	Plane	Detector	Axis	Z	10	Rectangle	3	3		
MOD	Local	Long Form	-1.5	1.5	-1.5	1.5	-0.5	3.5	Z	0
MOD	Interface	Coating	COATING	Absorb	Air	Air				
MOD	Rotate	Axis	X	45	0	10				

Figure 19.14 The Builder file used to define the rotated plane

The analysis is correctly performed by rotating the rays through -45 degrees about the X axis, centered on the point y = 0, z = 10. You can achieve this rotation from **Rays> Rotate Rays**. *The commands used to produce this rotation are:*

```
RAYSET
ROTATE X -45 0 10
```

The resulting spot diagram is shown in Figure 19.15.

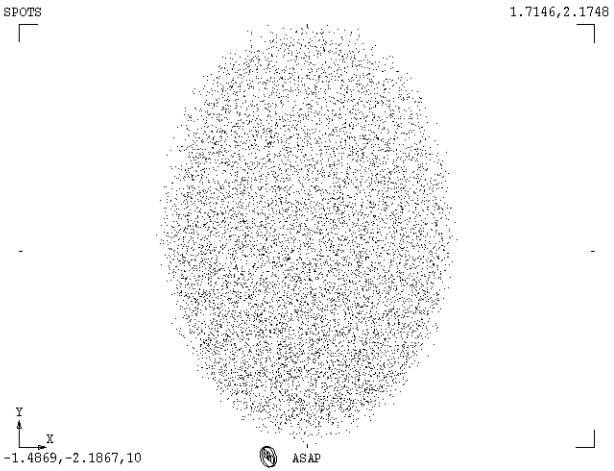


Figure 19.15 The spot diagram after rotating the rays to see the spot diagram in the reference frame of the detector

CHAPTER

20

DISPLAY TOOLS

This chapter contains a level of detail that is not typical for this Primer. You may find it useful to just browse many of the entries here, and use this chapter as a reference later on.

ASAP provides us with a comprehensive set of analysis tools for visualizing and processing distribution data files. In this chapter, we will describe and demonstrate these commands. Most of them apply to all types of radiometric distributions, not just the irradiance or illuminance results discussed in the previous chapter. For now, however, we will continue using the backlight irradiance example developed in the previous chapter to illustrate these tools whenever it is appropriate. Only two display tools are specific to the analysis of angular distributions. These tools are discussed in Chapter 21, "Analyzing Directional Distributions".

This chapter contains a level of detail that is not typical for this *Primer*. Many aspects of the display tools are not well documented elsewhere in the ASAP documentation, so they have been described here in some detail in the interest of completeness. You may find it useful to just browse many of the entries here, and use this chapter as a reference later on, as the need arises. A quick review of the summary table at the end of the chapter will likely prove to be a useful guide for the most interesting commands described here. The early discussions in the first two sections, "Distribution File Details" and "Display Tools—Overview", are fundamental, however, and should be studied with more care.

Distribution File Details

In Chapter 19, we introduced the concept of a distribution file. The work of sorting rays and accumulating flux as a function of position was accomplished using **Analysis> Calculate Flux Distribution**. The end result of this process is a file in your working directory named **bro009.dat**, which contains the distribution data.

> ***Note:*** While **bro009.dat** is a binary file, there are several ways described below to write its contents as ASCII text. If you are

interested in learning more about the format of the binary file, details are described in the topic in online Help, “Distribution File Structure”.

SPOTS POSITION is not the only command that generates a distribution file. Many ASAP commands use this flexible file format to store data arrays. The commands relevant to photometry and radiometry include **SPOTS** and **RADIANT** for analysis of geometrical rays. These are all discussed in this *Primer*. **FIELD** and **SPREAD** produce similar power distributions when ASAP is used in the coherent beams mode. See the Technical Guide, *Wave Optics in ASAP* for more about these commands. Other ASAP commands that write to **bro009.dat** include **SAMPLED**, **FMAP**, **RENDER**, **MAP**, **VOXELS**, **APODIZE**, **USERAPOD**, **COLLECTION**, and **OPDMAP**. Most of these make a single two-dimensional array, while others (notably **VOXELS**) create a three-dimensional distribution suitable for viewing in the 3D Viewer (see Figure 20.1).

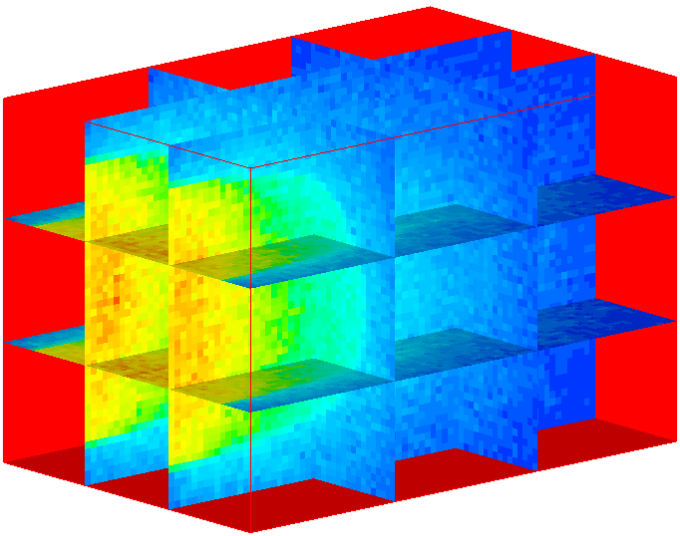


Figure 20.1 Some ASAP commands, like **VOXELS**, create three-dimensional data distributions. This display shows the result of a beam of rays entering a scattering volume, depicting the fluence (flow) of power within the material. The **3D Viewer** window displays six moving planes, so you can graphically explore the distribution within the volume. The graphic was produced by the example script, **Volume_Scatter01.inr**, located in **asapxxxx\projects\examples.**

Note: While the default output product of all these commands is a distribution file named **bro009.dat**, in many cases, you have the option to name the file yourself. For largely historical reasons, files you name will have a .**dis** rather that a .**dat** file name extension. The file format is the same, however, and all the follow-on analysis tools and methods provided by ASAP apply to either.

Several other ASAP commands and macros also create distribution files, though they do not use **bro009.dat** as their default file name. We list them here because many of the graphical and processing tools described below can also be applied to these files:

Command or Macro	Use
`$ITER` (first syntax)	Explore the behavior of a user-defined figure of merit as a function of one or more parameters; place the result in **iter.dis**.
`$EVAL`	Evaluate a user-defined function; place the result in **eval.dis**.
`MODEL ... PLOT`	Plot the BSDF of a scatter mode; place the results in a file named <**model name**>.**dis** (for example, **harvey.dis**

Display Tools—Overview

The only function in ASAP of the display tools is to display and process data in distribution files.

The display tools in ASAP are based on a flexible set of graphical and processing commands provided by the ASAP kernel. Their only function in ASAP is to display and process data in distribution files. Although we will look at them in the context of power as a function of position (irradiance and illuminance), the tools are general. They can be used to work with distribution files produced by any of the commands listed above.

Most tools are available through the **Display** menu. The submenus of **Display** show that these tools can be sorted into three broad categories:

1. file operations,
2. graphics, and
3. processing.

The **bro009.dat** *file remains intact until it is overwritten the next time you ask ASAP to calculate a new flux distribution.*

All the **DISPLAY** commands work with a copy of the distribution file that is maintained in the random access memory (RAM) of your computer. The distribution file itself, **bro009.dat**, remains untouched on your hard drive. If you make a mistake during processing, or would like to try a different approach, the original data are always available in **bro009.dat**. This file persists even after you end your current ASAP session. It remains intact until it is overwritten the next time you ask ASAP to calculate a new flux distribution, or run any of the other commands that use **bro009.dat**, as listed in "Distribution File Details" on page 401.

> ***Note:*** Occasionally, new ASAP users fail to create a distribution file after a ray trace, and proceed straight to the Display commands. If an old **bro009.dat** file exists in your working directory, ASAP will graph or process these old data without issuing an error or warning message. While this is sometimes useful behavior when done intentionally, it is more often a source of confusion. Always remember, when doing any sort of analysis that involves sorting rays by position or direction coordinates, you must make a distribution file before you can proceed with graphing or processing. If you are getting unexpected results, or if the results never seem to change no matter what you do to your geometry or sources, you may be looking at an old distribution file. Check the Command Output window to make sure you have successfully created the new distribution file after the **SPOTS** command.

Always remember, when doing any sort of analysis that involves sorting rays by position or direction coordinates, you must make a distribution file before you can proceed with graphing or processing.

Several methods exist for loading the distribution file into memory for analysis. All the graphics and processing tools discussed below perform this step automatically the first time you select an operation from the **Display** menu. When you use commands scripts, you use the **DISPLAY** command.

Once **bro009.dat** is loaded, the prompt at the bottom of your ASAP window changes from **ASAP>** to **DIS>** on the status bar. This prompt tells you that the array is now loaded in memory, and that all subsequent **DISPLAY** commands will operate on that copy, even if you have modified it from its original form. In the descriptions that follow, we refer to this working copy as the "distribution data" or "distribution data array". ASAP automatically leaves this special part of the program any time you issue a command that is not one of the display tools, and the **ASAP>** prompt (or some other) returns to the status bar. If you then issue a display tool command, you may notice a small warning in the Command Output window that lets you know this has happened:

```
*** Unrecognized DISPLAY subcommand!
```

This warning does not prevent the new command from being run, however.

> ***Note:*** For more information about the various prompts that appear on the status bar and their meaning, see "ASAP Prompts" in the on-line Help.

When you initially read **bro009.dat** into the distribution data array, ASAP always prints the file's header information in the Command Output window. All the relevant information about the data and how it was created is part of this header. Data created by **SPOTS POSITION**, for example, looks something like this:

```
--- DISPLAY
Opening OLD     distribution file   9: C:\ASAP Work\ASAP Primer\bro009.dat
 File header:
  Geometrical Ray SPOTS              1
  X          0.2510000      mW / sq-MM
  Z mm              -15.00000       15.00000             117
  Y mm              -5.000000       5.000000              39
Statistics on    39 by     117 data set:
        mW / sq-MM              Location       Y mm             Z mm
 Minimum    0.000000             38   116         4.615385         14.61539
 Maximum   0.3052285             30   116         2.564103         14.61539
 Average   0.1170321             20    59        0.2831221E-06    0.8493662E-06
 TOTAL mW  =  35.10962
```

File Operations

Four file operations are supported within the display tools. These operations are used primarily for reading distribution data from or writing to files on your disk drive. You access them from the **Display> File** submenu.

Display> File> Open / Read

The **Open / Read** file operation is used to read a distribution file into memory for graphing or processing. If you want to work with only the most recently created distribution file, you do not need to use this operation. All **GRAPHICS** and **PROCESSING** commands automatically read **bro009.dat** into memory for you, if you have not already done so, as long as you are accessing them from the **Display** menu. This command is useful, however, if you have performed some processing on the distribution data array and are not satisfied with the result, or interested in starting over for any other reason. Select **Display> File> Open / Read**, choose **Files of type: All Files**, and browse to **bro009.dat** in your working directory.

> ***Note:*** The ASAP command you are running is DISPLAY. When you learn to write command scripts in Chapter 22, "Writing Command Scripts", you will find that this operation is *not* automatically performed for you, and the **DISPLAY** command must be included in your script to load the distribution data into memory. ASAP will not

recognize other **DISPLAY** commands until you have completed this step.

Display> File> Save / Write

ASAP includes the **Save / Write** command so that you can save the current distribution data array to disk. You may want to do this to keep results from being overwritten next time you ask ASAP to calculate a flux distribution, or before you use any of the other commands that produce **bro009.dat** files (see "Distribution File Details" on page 401). You may also choose to save your results to a permanent file after you have processed data. The resulting dialog box (**Display: WRITE**) gives you the option of writing in binary (**.dis**) format, text (**.din**) format, or both. The binary files can be read back into ASAP for future analysis and processing using **Display> File> Open / Read**, as described above. While the text version of the **WRITE** command is useful for seeing what the distribution file (including header information) looks like, **TEXTFILE** is the preferred method of exporting data.

Note: The text version of the file (*.din) can be treated as a command script. You could copy the text lines from this *.din file into a script file, or use **$READ <Filename.din>** to read the *.din file from another script file. While there is no advantage to reading distribution files in this way (it is actually slower), it does show you how you might create your own distribution data file if the need arises. We will discuss this process later in this chapter in the sidebar, "Creating Your Own Distribution Data" on page 451.

Display> File> TEXTFILE

The **TEXTFILE** command writes the current distribution data to a text file. This method exports distribution data out of ASAP for use by other programs. The only option available to you in the graphical user interface is the name of the file and its location on your hard disk. When you click **OK**, ASAP generates a text file containing a two-dimensional table of the pixel values:

```
0.4526071E-01  0.3371493E-01  0.3546931E-01  0.6527250E-01  0.6236592E-01
0.8467023E-01  0.1678598E-01  0.7625486E-01  0.2014765E-01  0.9125160E-01
0.7576492E-01  0.3053934E-01  0.1381782E-01  0.2591616E-01  0.4370465E-01
0.8373917E-01  0.4448859E-01  0.4358093E-01  0.2940370E-01  0.9486641E-01
0.6137512E-01  0.4137176E-01  0.5233926E-01  0.4504054E-01  0.6426470E-01
0.8616499E-01  0.6474341E-01  0.6617995E-01  0.7447043E-01  0.9523474E-01
0.4750829E-01  0.2147532E-01  0.3024282E-01  0.3587401E-01  0.3979226E-01
0.2788155E-01  0.2090962E-01  0.7447678E-01  0.5757725E-01  0.8735236E-01
0.9495741E-01  0.8703909E-01  0.2226160E-01  0.8425464E-01  0.8596223E-01
0.9788040E-01  0.5620706E-01  0.3064831E-01  0.1821572E-01  0.1772603E-01
```

Note: One source of confusion with **TEXTFILE** is that the data are printed sideways relative to the way they appear in the Display Viewer by **PICTURE**. The pixel value at the top left of the table appears at the bottom left of the Display Viewer. As you read left-to-right across a row in the table, the values correspond to pixels, starting at the bottom of the Display Viewer and progressing upward. To visualize the table values as they would appear in the Display Viewer, imagine rotating the table counter clockwise through 90 degrees.

The **TEXTFILE** command offers additional options and flexibility for exporting data. It allows you to specify the format of the data in detail, and has options to include any of the header information. You can, for example, print the data in a three-column format consisting of the two spatial coordinates at the center of each pixel beside each pixel value. You can also control the format of the values printed:

```
-2.000E+00  -4.500E+00   4.526E-02
-1.000E+00  -4.500E+00   3.371E-02
 0.000E+00  -4.500E+00   3.547E-02
 1.000E+00  -4.500E+00   6.527E-02
 2.000E+00  -4.500E+00   6.237E-02
-2.000E+00  -3.500E+00   8.467E-02
-1.000E+00  -3.500E+00   1.679E-02
 0.000E+00  -3.500E+00   7.625E-02
 1.000E+00  -3.500E+00   2.015E-02
 2.000E+00  -3.500E+00   9.125E-02
   ...
```

See the on-line Help for details on the **TEXTFILE** command.

Display> File> IESFILE

The **IESFILE** command converts certain types of distribution files to a standard IESNA[1] photometric file. This feature is relevant for only radiant or luminous intensity data. You can download a file from the following Web page that contains the IESNA file format specification:

http://www.helios32.com/resources.htm#Formats

Scroll down to “Thinking Photometrically II—Lightfair International 2001 Workshop Notes” to find the link to the Adobe® Acrobat® PDF file.

1.Illumination Engineering Society of North America

Graphics Tools

ASAP provides us with 10 graphical options, or tools, for viewing raw or processed distribution data. The graphics are displayed in various output tools, including the **Plot Viewer**, **Chart Viewer**, **Display Viewer**, and **3D Viewer**.

Line graphics have been traditionally displayed in the same **Plot Viewer** that provides the output of the `PLOT FACETS`, `PROFILES`, and `TRACE PLOT` commands, seen throughout the previous chapters. Another option, the **Chart Viewer**, was added to provide output and user controls. The controls take advantage of common user interface characteristics of current windowing environments.

The **Plot Viewer** and **Chart Viewer** provide eight of the graphical options described here. This chapter shows a few instances of the output of all the viewers.

> ***Note*** To use the **Chart Viewer** for those options, open the **User Interface Preferences** dialog box (**File> Preferences**), select the **Plot Viewer tab**, and select **Use Chart viewer for displaying graph**. When this option is not selected, the **Plot Viewer** is the default viewer. Alternatively, you can run `$GUI CHARTS_ON` from the **Command Output** window or a script. To exit this mode, run `$GUI CHARTS_OFF`.

All viewers are shown in default states, but this is just a starting point. The possibilities are worthy of exploration.

The graphical tools operate on the distribution data array, reflecting its current state. The first time you access one of these tools—or one of the processing tools—ASAP automatically loads the raw data from **bro009.dat** into memory. If you subsequently use any of the processing commands, the original data in memory are replaced in the distribution data array by the processed values, and subsequent use of the graphics tools reflects those changes. If you want to see the original data again, you must reload it from **bro009.dat** as described in “Display> File> Open / Read” on page 405.

The first four display tools we will describe are used to visualize the information in all distribution data pixels at once. All four panels of Figure 20.2 show the same distribution data for the backlight display from Chapter 19, but displayed in different ways.

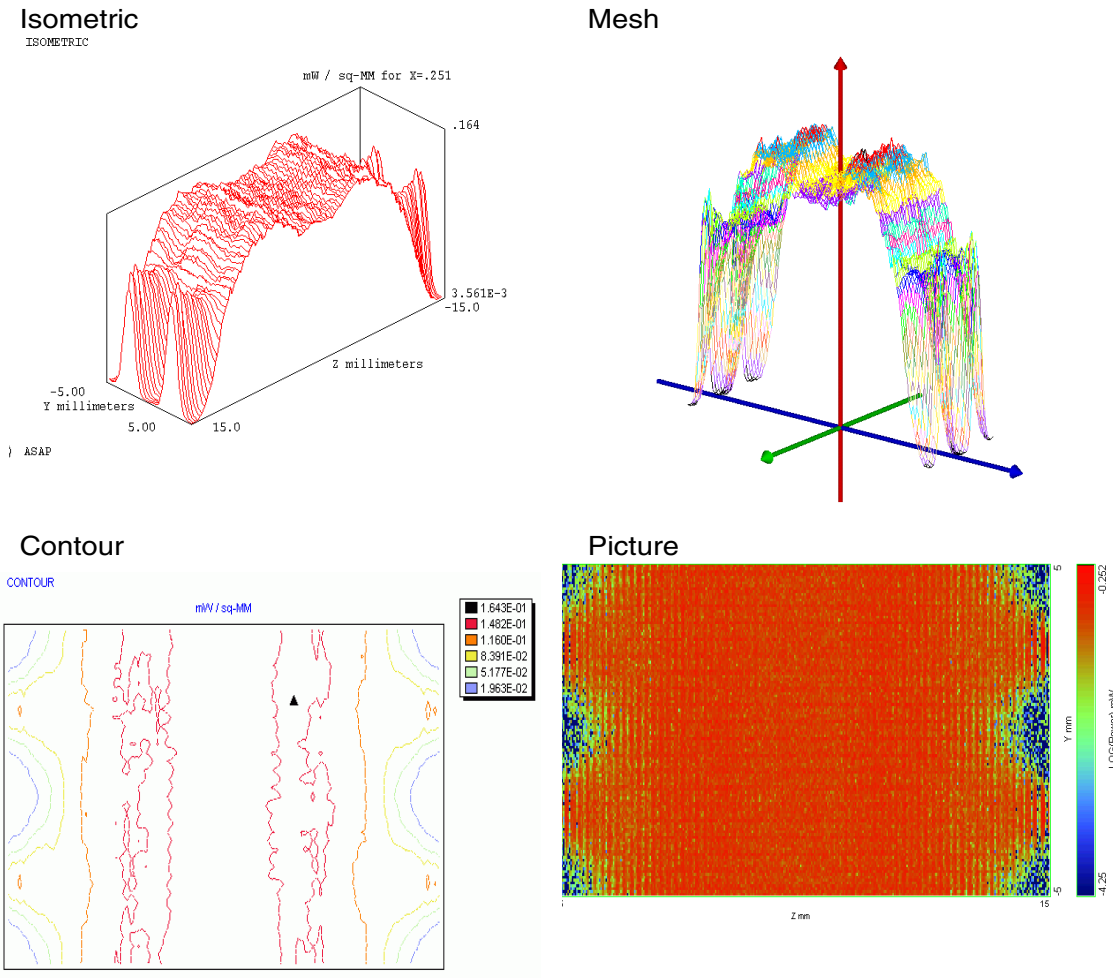


Figure 20.2 Four of the graphical display tools offer a means of visualizing the entire working copy of the distribution file. **CONTOUR** and **ISOMETRIC** are displayed in the **Plot Viewer** window, **PICTURE** appears in the **Display Viewer** window, and **MESH** produces a vector version of the distribution for the **3D Viewer**.

Display> Graphics> Isometric

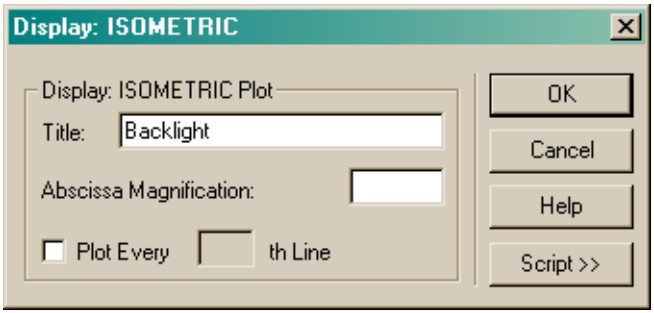


Figure 20.3 Dialog box for **Display: ISOMETRIC**

The ISOMETRIC command provides an easy way for us to visualize the entire distribution file. An example appears at the top left of Figure 20.2 on page 409. The command has only a few options. It displays an isometric projection of cuts through the distribution array. By default, you get the same number of cuts through the data as the number of pixels you specified. You also have the option to **Plot Every nth Line**. The **Abscissa Magnification** parameter lets you magnify or reduce the height of the data in the resulting graphic by a scale factor. This value should be entered as a floating-point (decimal) value.

See Chapter 20 Appendix, "Script 20-1" on page 453.

Display> Graphics> Contour

The CONTOUR command produces a contour plot, as shown in the bottom left panel of Figure 20.2 on page 409. The parameters that produced it are shown in Figure 20.4 on page 411.

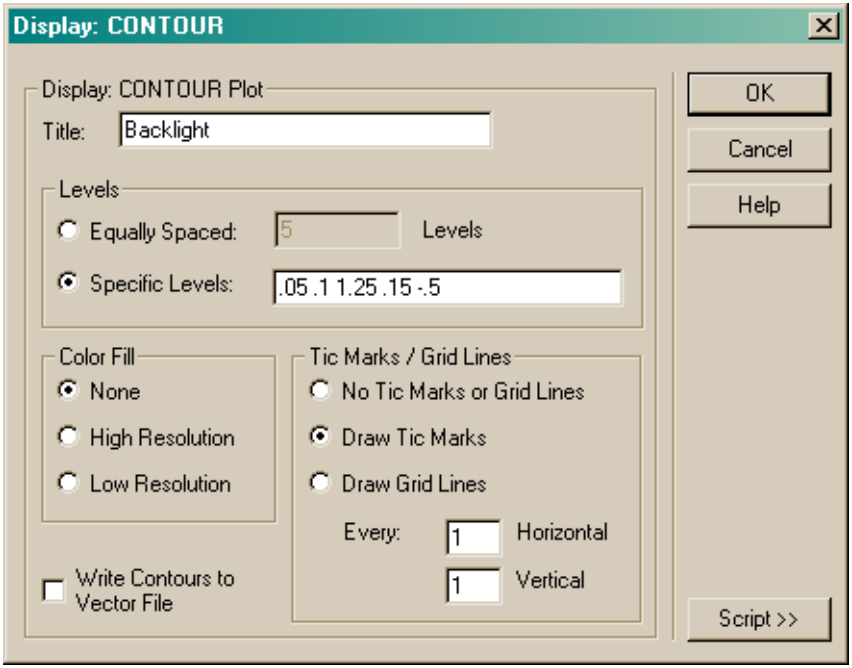


Figure 20.4 Dialog box for **Display: CONTOUR**

By default, ASAP generates five **Equally Spaced Levels**, but you can change that number or enter **Specific Levels** yourself. If you select the **Specific Levels** option, ASAP normally expects to see fractional values between 0 and 1. For example, if you want to see contours at 25%, 50%, and 75% of the maximum value, enter 0.25, 0.50, and 0.75. If you want absolute values, signal this to ASAP by including at least one value that is either greater than one or less than zero. This "signal value" can be outside of the data range if necessary, as we were forced to do in this example.

> ***Note:*** While it is possible to control the location of the contours, there is not currently a way to specify the color to be displayed. ASAP rotates through the same palette of 27 colors that are used throughout the program to color code geometrical elements and multiple sources.

The **Color Fill** option fills in the space between the contours with the color of the lower-valued contour. There is no difference between the **High Resolution** and the **Low Resolution** option for most modern printers and graphical display devices. This option survives to maintain backward compatibility with older versions of ASAP.

You also have the option to **Draw Tick Marks** (as shown in the figure) or **Draw Grid Lines**. The tic or grid spacing is entered in system units. The figure shows tic marks every millimeter to provide a positional scale within the plot.

If you select **Write Contours to Vector File**, the contour graphic also appear in the **3D Viewer** the next time you open it.

It is also possible to overlay interpolated values within this plot. See the discussion below under **Display> Processing> Table**.

See Chapter 20 Appendix, "Script 20-2" on page 453.

Display> Graphics> Mesh

You can use the `MESH` command to produce a vector version of the distribution data suitable for displaying in the 3D Viewer. The result for the backlight display appears in the top right panel of Figure 20.2 on page 409. There are several advantages to using the 3D Viewer to display distribution data. First, while the display is similar to that of the **Isometric** option, the 3D Viewer provides us with an easy way to rotate the data, changing our point of view. Second, the 3D Viewer allows us to simultaneously display geometry, ray data, and analysis results all in one view. Figure 20.5 shows an example of this.

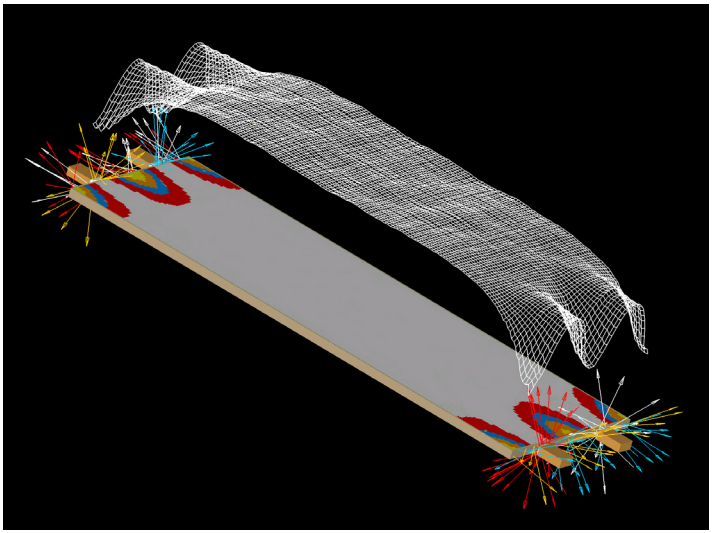


Figure 20.5 The 3D Viewer combined four different elements in this example. The basic geometry was placed into the vector file with `PLOT FACETS`. The "missed after" rays were placed into the vector file during the ray trace using `TRACE PLOT` after setting `MISSED ARROWS`. `SPOTS POSITION` created a distribution file after the ray trace. **Contour** produced the filled contours by specifying **Color Fill** and **Write Contours to Vector File**. Finally, **Mesh** produced the three-dimensional version of the distribution results hovering over the other elements.

Note: ASAP automatically sets the vertical scale of the mesh so that the maximum is similar to the largest spatial dimension of the window. You can control this yourself with the `RANGE` command. See below under **Display> Processing> Range** in the **Processing** section.

See Chapter 20 Appendix, "Script 20-3" on page 453.

Display> Graphics> Picture

The `PICTURE` command shows a fourth version of the data, this time displayed as a pixel-by-pixel pseudo-color map of the data. The result is displayed in the Display Viewer, a powerful and versatile window that we introduced briefly in the previous chapter. No options are available when you select **Picture** from the menus. The Display Viewer menu, which appears in the main ASAP menu bar when this viewer window has focus, is also deceptively sparse. There are actually many options available within this tool, accessed by right-clicking the plot display, palette, and graph areas, or the background area of the window (see Figure 20.6).

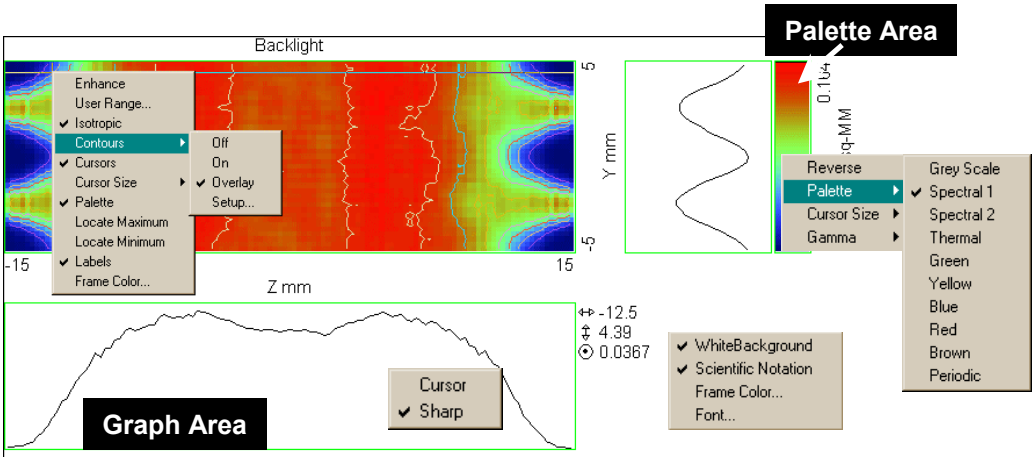


Figure 20.6 The **Display Viewer** window has a few options available from the menu bar, but right-clicking in a specific region of the window opens a pop-up menu with more options. The plot display area (top left) controls the plot display itself, and also toggles the other two elements of the window on and off. Deselect **Cursors** to eliminate the graph windows. Deselect **Palette** to eliminate the palette. Right-click in either graph display area to extend the cursors into the graph area, and select graph smoothing. Right-click in the palette area to access the palette options. Right-click anywhere in the background to select general-appearance options.

In addition to the pseudo-color map, the Display Viewer includes the capabilities of several other **Display> Graphics** options, including **Contour** and a cursor-driven alternative to the `GRAPH` command. Experiment with this command to discover the various options, using Figure 20.6 as a guide. You can also refer

to the on-line Help for this window: select **Help> Display Help** from the ASAP menu bar when the Display Viewer has focus. When you select your favorite display options, ASAP remembers the configuration next time you use the tool.

See Chapter 20 Appendix, "Script 20-4" on page 453.

The next four Graphics tools offer other ways of displaying all or part of the information available in the current distribution.

Display> Graphics> Graph

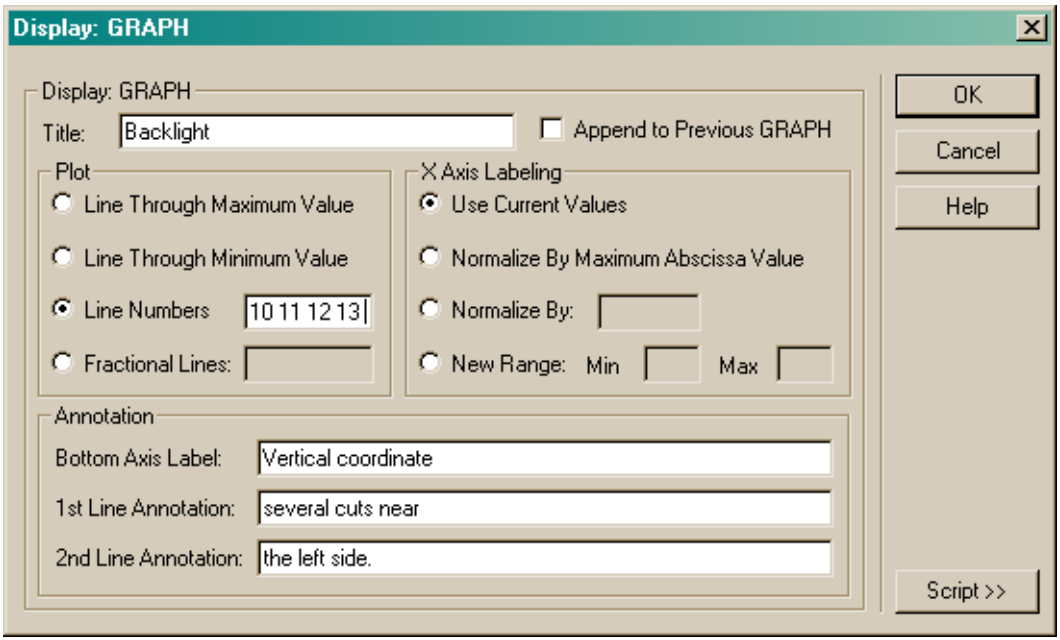


Figure 20.7 Dialog box for **Display: GRAPH**

The GRAPH command plots cross sections of distribution files in the **Chart Viewer** window. Examples appear in Figure 20.8a and Figure 20.8b on page 416. Both panels show alternative graphic methods for analyzing distribution data, using the backlight example.

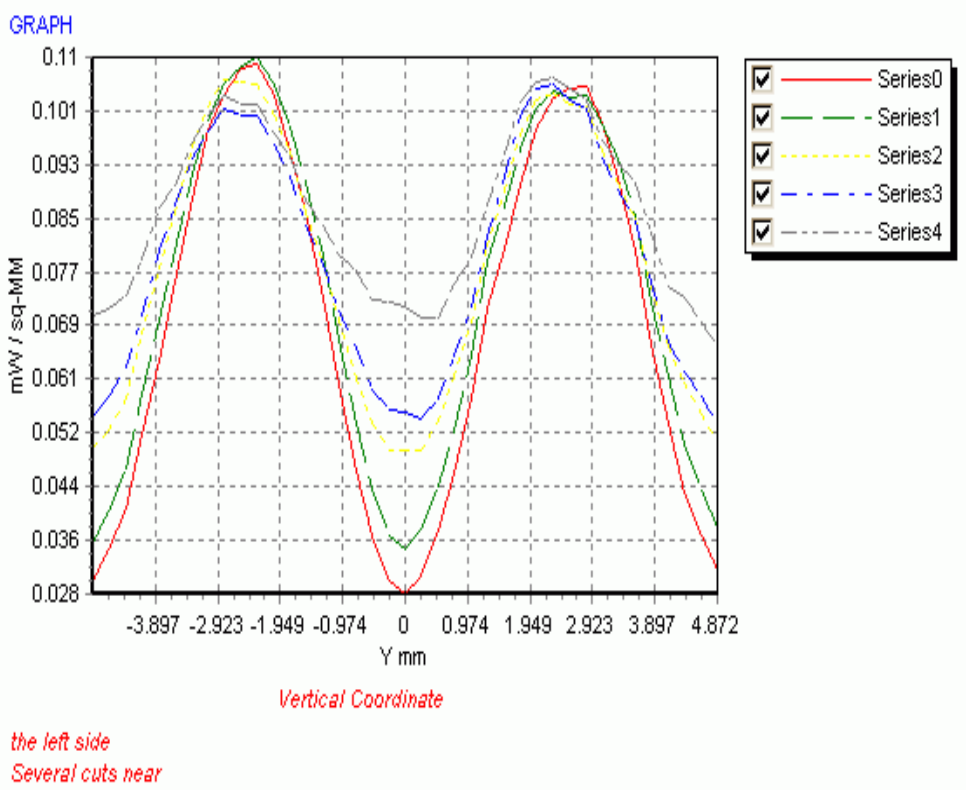


Figure 20.8a The **GRAPH** command generates one or more vertical cross sections through the distribution file. In this case, as displayed in the **Chart Viewer**, five vertical cuts were specified near the left edge of the backlight panel.

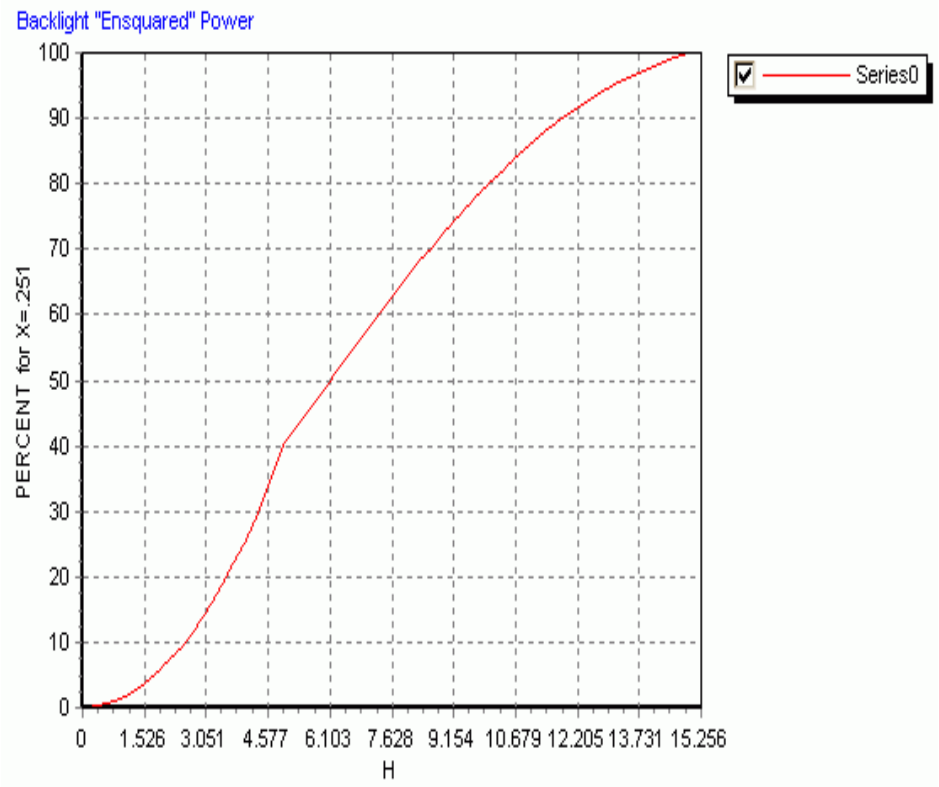


Figure 20.8b The **ENCLOSED** command shows the result of some processing of the data. It graphs the percentage of power enclosed as a function of square half-width. The squares expand from the center of the distribution. Note the inflection point at 5. This example was produced by the **Chart Viewer**.

By default, **GRAPH** graphs vertical cuts through the data. Horizontal cuts can be obtained by first transposing the data (see “Display> Processing> Transpose” on page 442). The **Display: GRAPH** dialog box offers a variety of options. For the plot itself, you can graph a **Line Through Maximum Value**, a **Line Through Minimum Value**, selected **Line Numbers**, or a cut at some fraction of the total vertical width. This **Fractional Number of Lines** is the most common option. You will frequently specify **0.5** here to get a slice through the middle of the distribution.

You also have the option to add a **Bottom Axis Label**, and two lines of annotation to the bottom of the graph. The vertical axis is always the flux of the data at a given pixel, although this scale can be altered using the **X Axis Labeling** options.

You can display more than one **GRAPH** command in the same plot window by selecting **Append to Previous Graph**. Note that, unlike the overlay option available in many other ASAP plotting contexts, you express your intention to

append data into a previously opened plot window *after* the first plot is made. Additional data is plotted on the same vertical scale.

If you need to control the vertical scale of the graph, you can do this with the RANGE command and the THRESHOLD command. These are discussed below among the processing tools.

See Chapter 20 Appendix, "Script 20-5" on page 454.

Display> Graphics> Enclosed

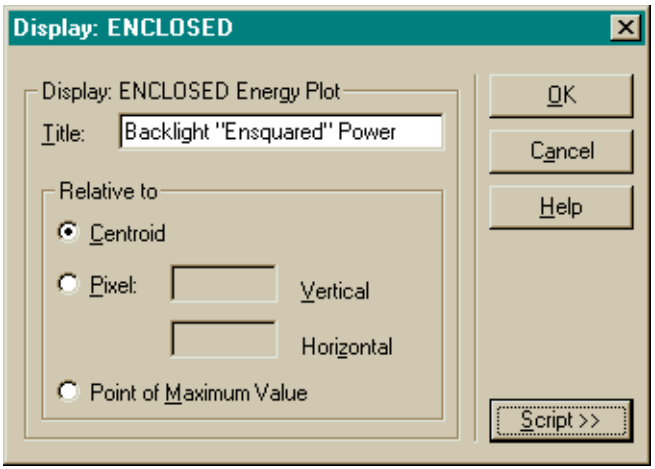


Figure 20.9 Dialog box for **Display: ENCLOSED**

The ENCLOSED command produces a graph of the type shown in Figure 20.8b on page 416. The horizontal axis is the half dimension of the square enclosing a subset of the pixels in your original window. The vertical axis is the percentage of the total power within that subset. This is sometimes referred to as "ensquared" (as opposed to encircled) power. The square grows until 100% of the power is enclosed. You can specify whether you want the square centered on the center pixel (**Centroid**), the maximum value (**Point of Maximum Value**), or a specified pixel. In the figure, the enclosed power increases roughly quadratically (as expected for a uniform distribution), until the square reaches the vertical edge of the data window ($y = 5$). It then increases in a roughly linear way through the uniformly illuminated regions, then drops near the poorly illuminated edges.

See Chapter 20 Appendix, "Script 20-6" on page 454.

> ***Note:*** While some processing was done to obtain the **Enclosed** plot, it is not considered a processing command, because the values in the current distribution array are not altered in any way. The graph is the only product of this command.

Display> Graphics> Plot3D

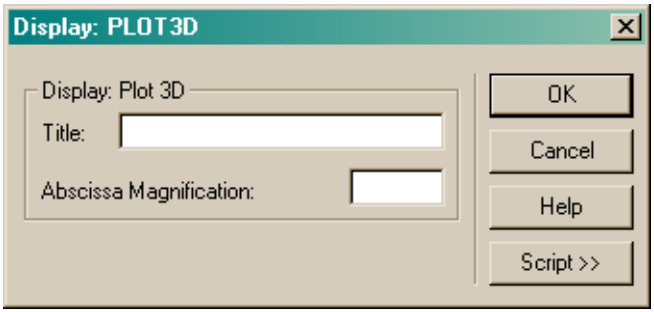


Figure 20.10 Dialog box for **PLOT3D**

The PLOT3D command combines the isometric display with graphs of several cross sections through the data (see Figure 20.11). This is one of the oldest DISPLAY commands in ASAP, and remains in the program largely for backward compatibility. It can be produced by only the **Plot Viewer**, regardless of the Preferences setting for using the **Chart Viewer**, which was described at the beginning of this chapter.

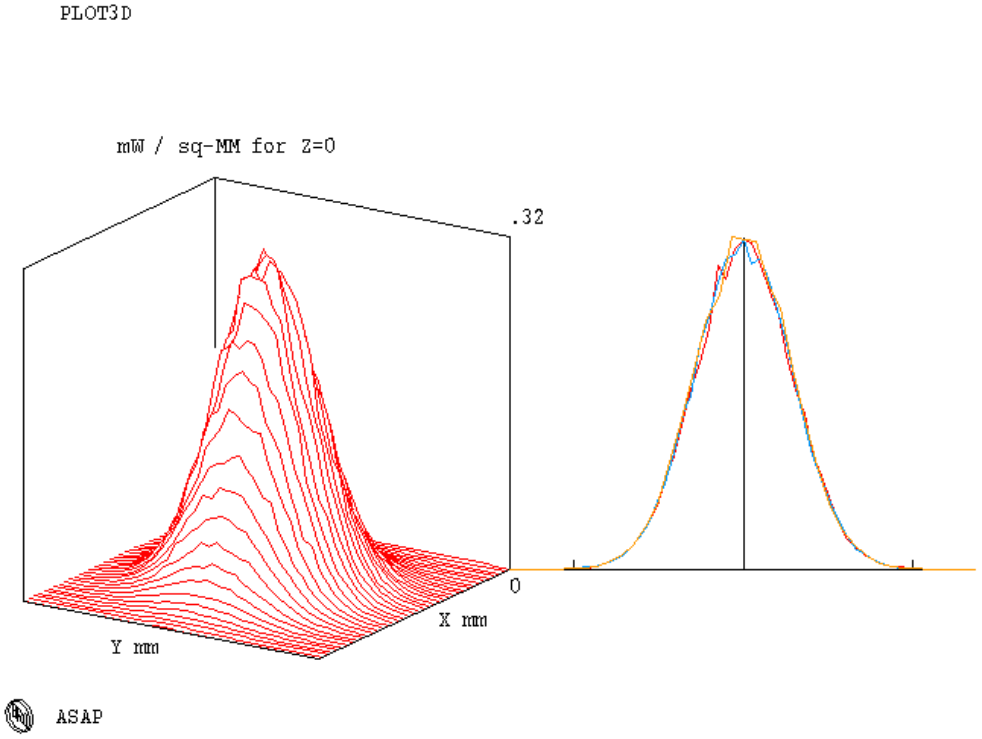


Figure 20.11 The PLOT3D graphic, produced by the **Plot Viewer**, combines an isometric plot with three cross sections: vertical (red), horizontal (blue), and diagonal (orange).

The cross sections are drawn through the peak value of the distribution, not the centroid, and there is no option for changing this behavior. You will always see a vertical and a horizontal slice through the data in the cross-section view. You may also see a third, diagonal slice, if the pixel containing the maximum value has odd indices.

All the functionality of **PLOT3D** also exists in other **DISPLAY** commands, with considerably more flexibility. See the discussion of **GRAPH**, **TRANSPOSE**, and **ISOMETRIC**.

Display> Graphics> Radial

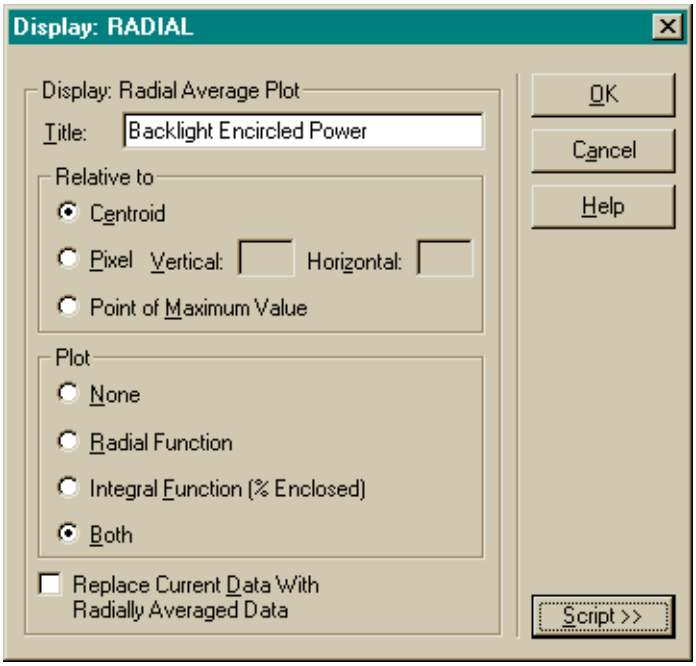


Figure 20.12 Dialog box for **Display: RADIAL**

The **RADIAL** command performs radial averages of distribution data around a point. In other words, it averages the data in each pixel with other information at the same radial distance from a specified center. It can then produce either a graph of the radially averaged values (**Radial Function**), the encircled power, (**Integral Function**) or both.

Based on a distribution shown in Figure 20.13a on page 421, the example shown in Figure 20.13b shows both. In most cases, radial averaging is appropriate only when there is radial symmetry present in the distribution data.

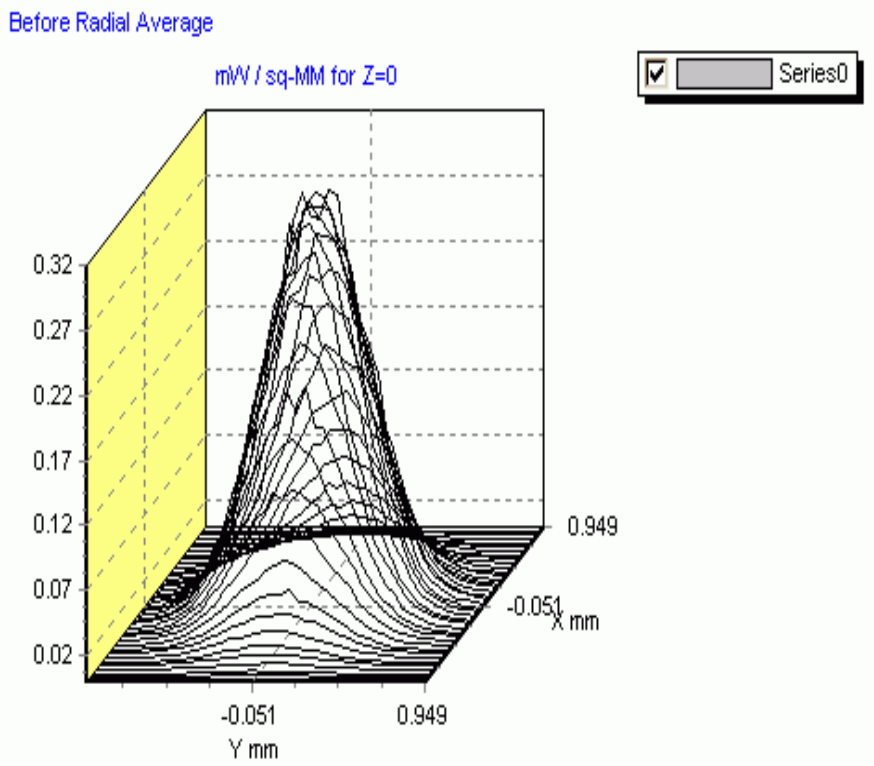


Figure 20.13a An isometric plot, shown in the **Chart Viewer**, of the original distribution file created by `SPOTS POSITION` for a radially symmetric distribution.

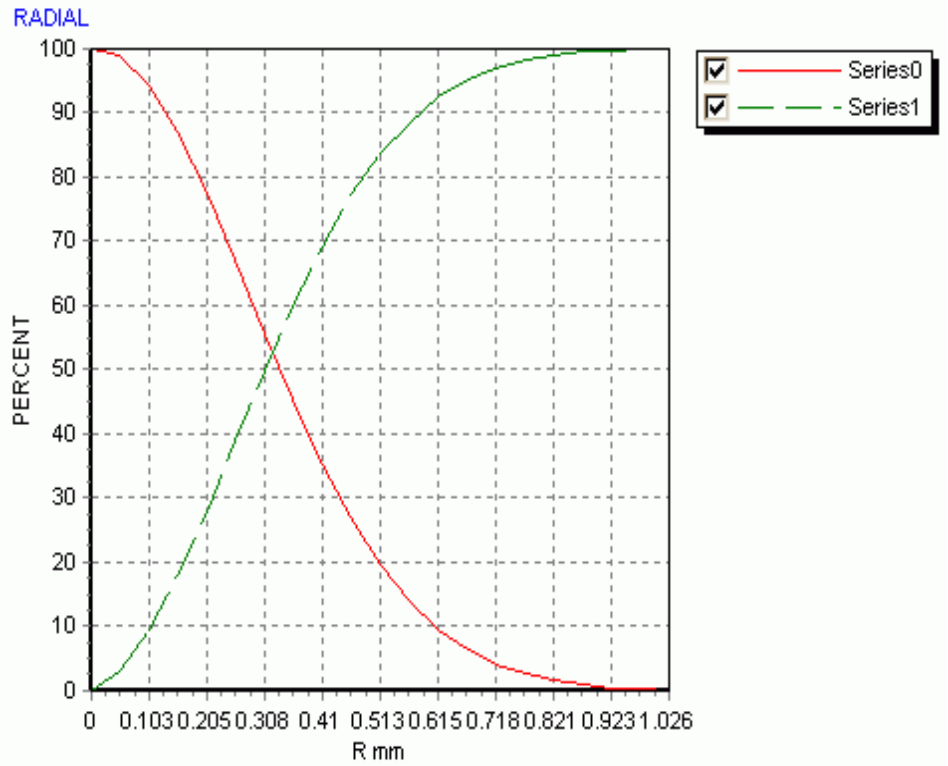


Figure 20.13b A cross section, shown in the **Chart Viewer**, through the center of this distribution, after being radially averaged (red curve), and the integral of the data as a function of radial position (blue curve).

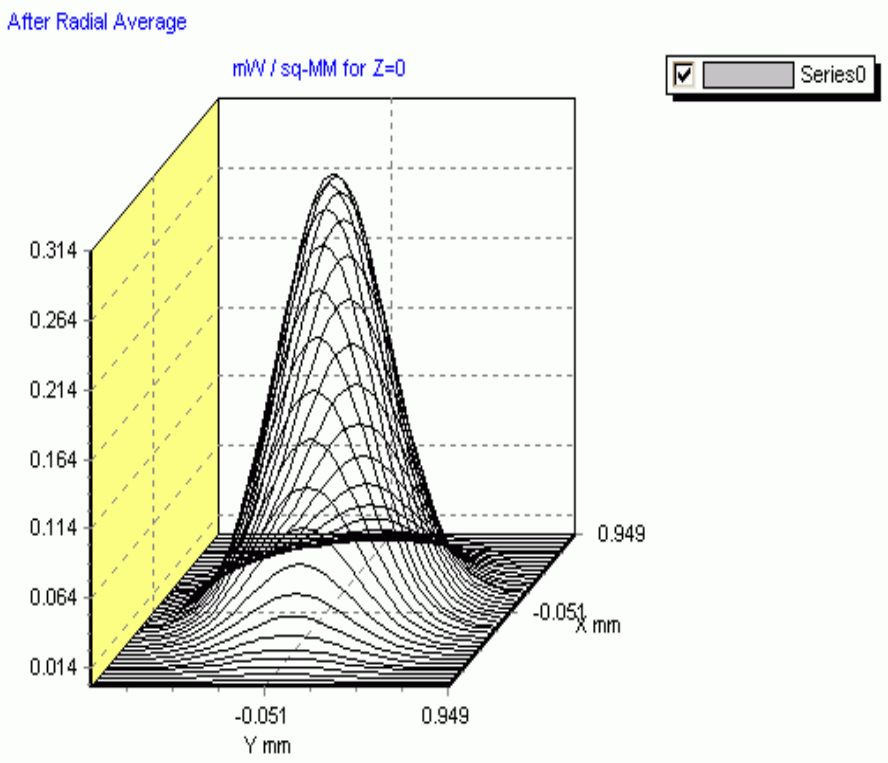


Figure 20.13c The new distribution data, shown in the **Chart Viewer**, after using the **Replace** option to overwrite the working copy of the distribution with the averaged data.

As with **ENCLOSED** (described above), you can perform the radial average around the center of the distribution (**Centroid**), a specific **Pixel**, or the **Point of Maximum Value**.

> ***Note:*** Since ASAP allows only a square or rectangular analysis window, there will consequently be problems with data that extend all the way into the corners of the window. Be sure to select a window that is large enough to inscribe the circular distribution, keeping the corners empty. Otherwise, any averaging performed on flux outside of the inscribed circle will likely produce artifacts in the resulting plots.

While this display tool is in many ways similar to **Enclosed**, performing much the same function for data with radial symmetry, there is one crucial difference: **Radial** gives you the option to modify current distribution data, replacing each value by the radial average of all values at that distance from the center. This modification is accomplished by selecting **Replace Current Data With Radially Averaged Data**. When used in this way the **RADIAL** command is a **Processing** as

well as a **Graphical** tool. The result of this is sown in an isometric chart in Figure 20.13c on page 423. If you look closely at the resulting distribution data using PICTURE or GRAPH, you will see that it now displays perfect symmetry about the center point you specified, and that even the value of the center point has been subjected to some "smoothing" based on the behavior of nearby points in the new distribution.

See Chapter 20 Appendix, "Script 20-7" on page 454.

Display> Graphics> Histogram

The HISTOGRAM command displays a plot of the number of data values within *n* intervals equally spaced between the minimum and maximum values. The default for *n* is 21. The plot can be optionally normalized to a peak value of one. The plot can also scale data values so that the area under the curve is unity, consistent with the definition of a probability density function. An example is displayed in "Dialog box for Display: HISTOGRAM".

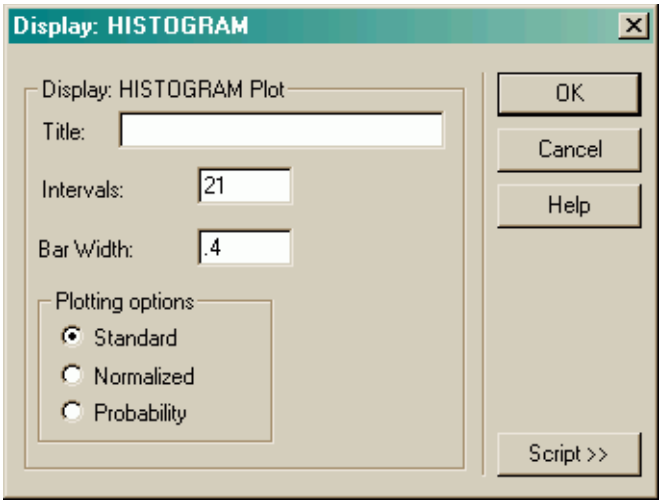


Figure 20.14 Dialog box for Display: HISTOGRAM

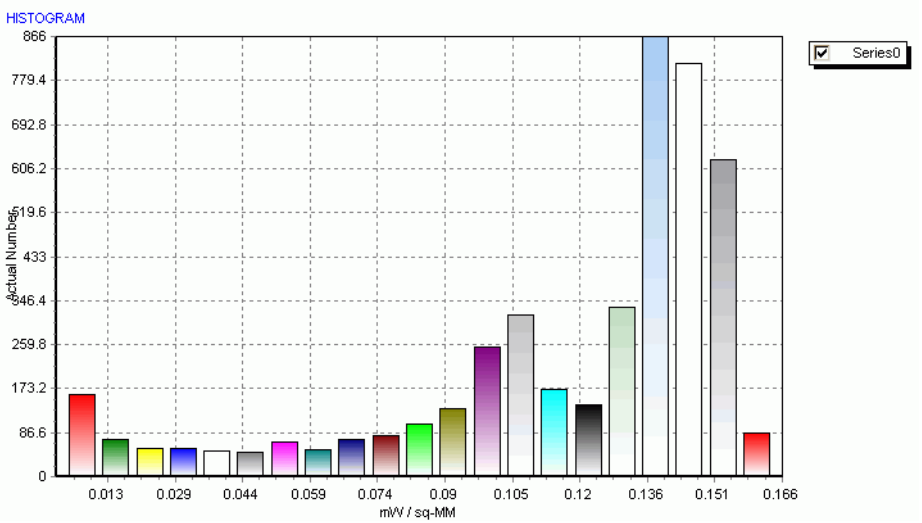


Figure 20.15 Histogram output as shown in Chart Viewer

Display> Graphics> Directional

The **DIRECTIONAL** command is appropriate for use only with directional data distributions (**SPOTS DIRECTION**, rather than **SPOTS POSITION**). It will be discussed in Chapter 21, “Analyzing Directional Distributions”.n

Data Processing Tools

In addition to displaying the distribution data as described above, you can also modify or numerically display the data using various display data processing tools. All these tools are available on the **Display> Processing** menu**.** The commands you access there allow you to smooth, transform, or otherwise manipulate distribution files. Remember that while many of these commands modify the data array, you are working with only a *copy* of the distribution file. The original sorting and accumulating accomplished with **SPOTS POSITION** is still intact on your disk drive in **bro009.dat**. You can recover the original data, or save the modified values using the **Display> File** operations described in “File Operations” on page 405. Also, the processing commands that alter the current distribution data do not automatically display the modified data or update open Display Viewer, 3D Viewer, or Plot Viewer windows. You will need to repeat the **Display> Graphics** commands to see the new results. A total of fifteen processing options are available through the menus, and 24 from within the Builder. We will summarize the use of most of these processing options below.

Display> Processing> Angles

The **ANGLES** command converts angular distributions stored as direction cosines into angles measured in degrees. This processing tool is discussed in detail in Chapter 21.

Display> Processing> Average

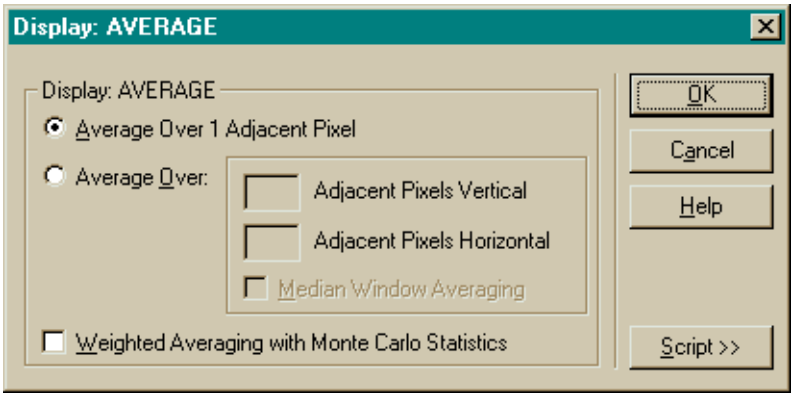


Figure 20.16 Dialog box for **Display: AVERAGE**

The **AVERAGE** command smooths the current distribution data by averaging adjacent pixels. This command is frequently applied to “noisy” data resulting from too few rays spread across too many pixels. The only numerical parameters required by the command are the number of vertical and horizontal pixels over which to average. The details of the averaging process are shown in Figure 20.6 on page 413.

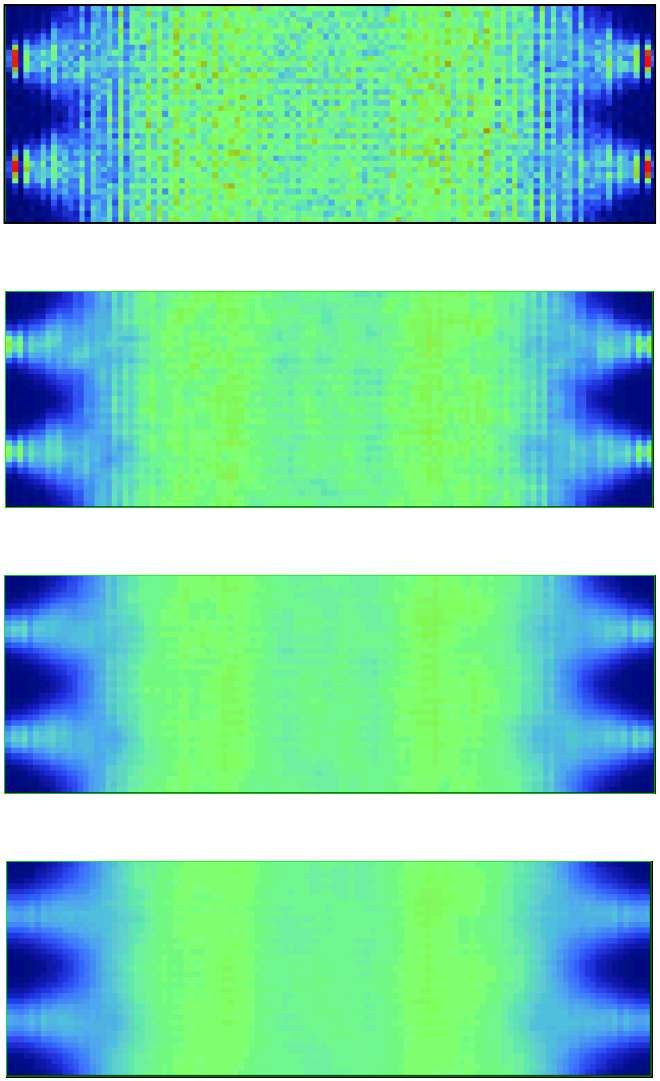


Figure 20.17 This progression of four **Display Viewer** panels demonstrates the effects of the **AVERAGE** command on the backlight flux distribution. The top frame is the original distribution created by the **Calculate Flux Distribution** process. The three that follow use varying amounts of averaging, as indicated next to each panel. Note that the detail visible near the sources on the left and right edges is lost as the size of the averaging square is increased.

Note that if you select **Average Over 1 Adjacent Pixel**, you are actually averaging over a 3 × 3 area, or *nine* pixels. If you average over two pixels vertically and

horizontally, you are averaging a 5 × 5 area, and so forth. As you average over more pixels, the results look smoother, but you are steadily losing spatial information (see Figure 20.18 on page 428). But which is the most “accurate”?

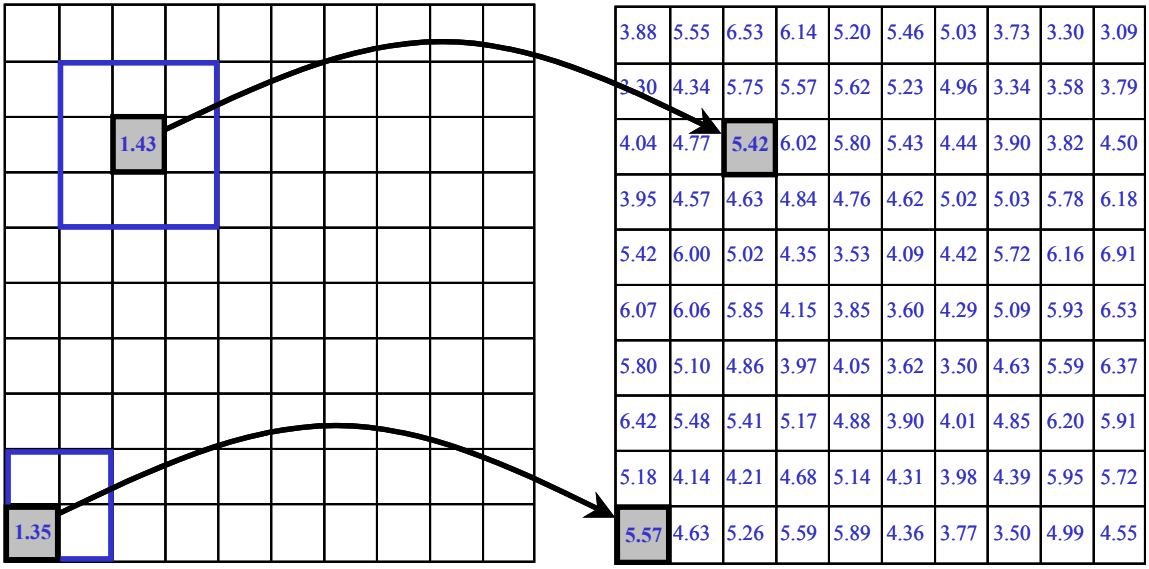


Figure 20.18 The **AVERAGE** command in ASAP is a “boxcar” filter. If you ask to average over one adjoining pixel, ASAP is actually drawing a 3 x 3 box around each pixel, and replacing it by the average of those nine values. The box then slides one pixel to the right and calculates an average for that pixel, using the original values (not the averages) for pixels in the box that have already been treated. In the corners, only four pixels are averaged, and along the sides, only six. If you select the **Weighted Averaging** option, a weighted average of the nine pixels is computed, using the square root of the value to determine the weight.

As the example in the figure shows, you may be masking real variations in flux as a function of position that you are trying to investigate. Averaging can never add information. It can only clarify a feature that is already present.

Note that you could have obtained effects that are similar (but not identical) to averaging by using larger (fewer) pixels when you originally had ASAP calculate the flux distribution. Doing this after the fact, however, requires backing up and performing the **SPOTS** calculation again with **Analysis> Calculate Flux Distribution**. Repeating this step becomes increasingly costly as the number of rays involved in the analysis increases. Using the **AVERAGE** command with a relatively large number of pixels also makes the display look less “pixelated” than it would by increasing the size of the pixels. The information content is still basically the same, however.

Figure 20.19 on page 429 shows the same ray data analyzed in two different ways.

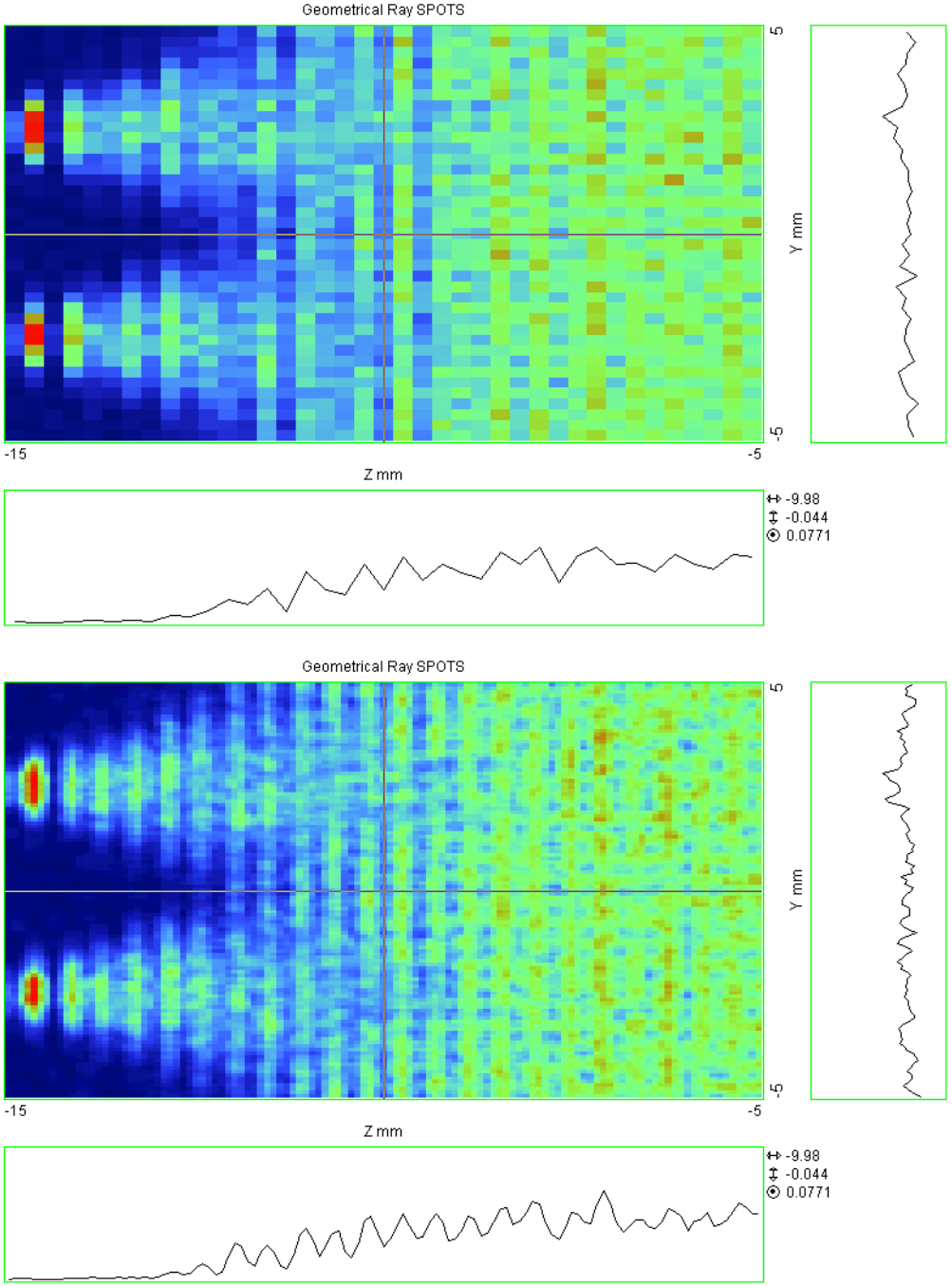


Figure 20.19 For the first view, 39 pixels were used. For the second view, three times as many pixels were used, and then a 3 × 3 boxcar average was applied.

Both **Display Viewer** displays clearly show vertical banding that is due to underlying prisms in the geometry, which divert the light out of the light guide and onto the sample surface. The second view is more smoothly pixelated, perhaps more pleasing to look at, but contains fundamentally the same information as the first view.

The other option available in the **Display: AVERAGE** dialog box is **Weighted Averaging with Monte Carlo Statistics**. If selected, ASAP assumes that the error in each pixel is proportional to the square root of its value, and weights the pixel accordingly in the average. This option is useful for preserving sharp peaks or strong positional gradients in the raw data.

Statistics clearly play an important role in interpreting data resulting from Monte Carlo ray-trace simulations. If you are unfamiliar with these concepts, see the sidebar, “How many rays, and how many pixels?” on page 449.

See Chapter 20 Appendix, “Script 20-8” on page 454.

Display> Processing> Combine

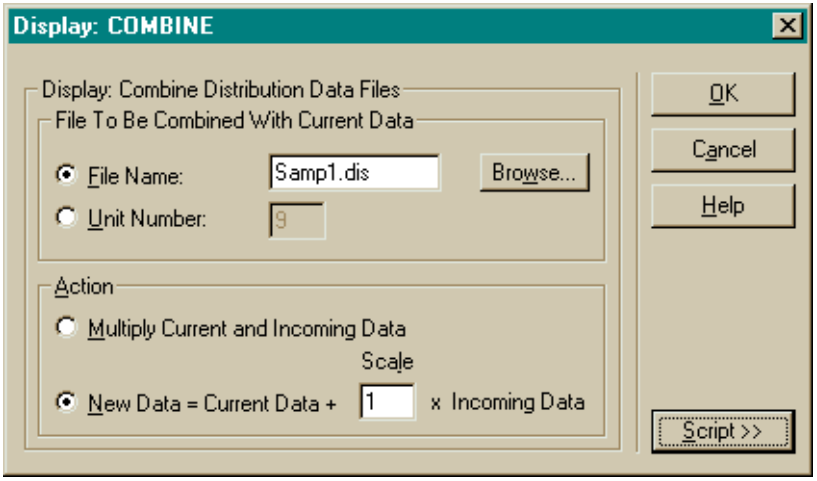


Figure 20.20 Dialog box for **Display: COMBINE**

The `COMBINE` command merges two distribution files. The most common use of this tool is to add distribution files from independent ray traces to form a single result. Some analysts, for example, perform large ray traces by running ASAP on several computers in parallel. Each computer traces rays through identical system geometry, but the seed value is changed in each instance to generate a different set of random rays. The analysts use **Display> File> Save / Write** to archive each individual result, and merge them later with `COMBINE`.

In the above example Display dialog box, we have presumed that one result was saved in a file named **Samp1.dis**. As soon as you use any `DISPLAY` command

(like **Display> Graphics> Picture** to confirm the new result), the most recently created distribution file (**bro009.dat**) is in memory, becoming the **Current Data**.

See Chapter 20 Appendix, "Script 20-9" on page 454.

> ***Note:*** Alternatively, you may have previously used **Display> File> Open / Read** to load additional archived data for combining with **Samp1.dis**. This data is then the **Current Data**.

Select the **File Name** option and browse to an archived result, which becomes the **Incoming Data**. The **Action**, in this case, is to set **New Data = Current Data + 1** × **Incoming Data**, thereby adding the two results. You have the option to include a multiplicative scale factor, if needed. You can use a scale factor of **-1,** if you are performing a background subtraction.

You can also use the **DISPLAY: Combine** tool to multiply two distributions, although this is less common. If you use the command language, you can also combine distributions in any way that can be expressed with ASAP functions and expressions. You must resort to this if you need the ratio of two files, for example.

> ***Note:*** When combining files, always confirm that the data in both distributions were formed using exactly the same window and pixel settings. To accomplish this, you must explicitly specify the window boundaries, rather than allowing ASAP to auto scale based on ray location. The only error checking performed by `COMBINE` is a simple test to make sure both distributions have the same number of data points. The command then just performs a pixel-by-pixel combination without registration or interpolation.

Display> Processing> FFT

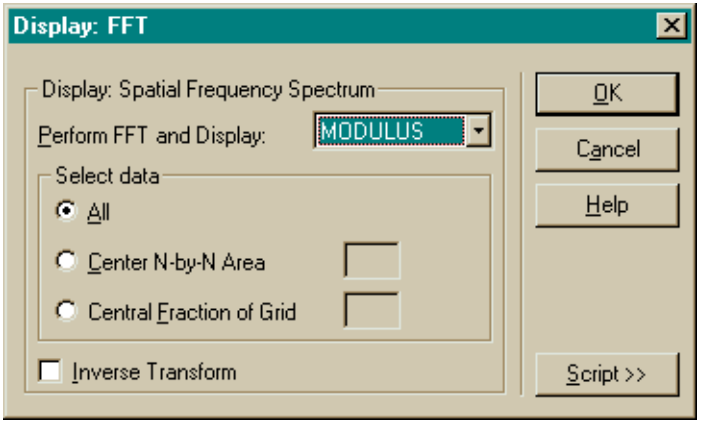


Figure 20.21 Dialog box for **Display: FFT**

The **FFT** command computes the Fourier transform of the distribution data to produce a spatial frequency spectrum or its inverse. It can also be used, for example, to compute the modulation transfer function (MTF) for an image of a point source. Several command-script examples of this procedure are in the example database, located in the ASAP install area on your disk in **<install directory>\projects\examples**. See **mtf_obscured.inr**, **psf_poly_doublet03.inr**, and **mtf_ideal_lens.inr**.

See Chapter 20 Appendix, "Script 20-10" on page 454.

Display> Processing> Fold

The **FOLD** command averages the current distribution data around a vertical or horizontal line, or both. It is ideally suited to a problem like the backlight display example, since it has lateral symmetry in both directions. Since the symmetries of the sources and geometry dictate that the differences between the four quadrants of the display will be only statistical, we can replace each pixel by the average of four pixels (see Figure 20.17). This has the same effect as tracing four times as many rays.

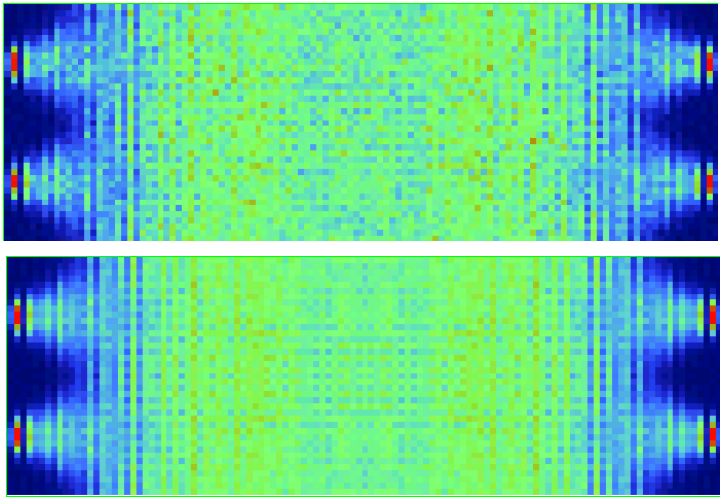


Figure 20.22 These two figures show the effects of applying **Display: FOLD**. The top panel shows the raw distribution data, and the lower panel shows the result after folding the data around both the vertical and horizontal centers. Now we can take advantage of symmetries in the design. Note that, while the new distribution data still has the same number of pixels (39 x 117, in this case), the same data values are only replicated in all four quadrants.

Display> Processing> Form

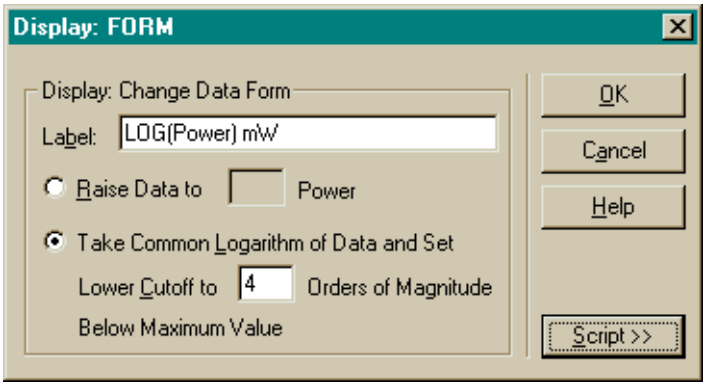


Figure 20.23 Dialog box for **Display: FORM**

The **FORM** command changes the form of the data, either by raising all values to a specified power, or by taking the common (base 10) logarithm.

You have the option of writing a new 16-character **Label** to the header block of the distribution file by entering it in the **Display: FORM** dialog box. This label appears in the appropriate places on subsequent graphical windows, as shown in Figure 20.24.

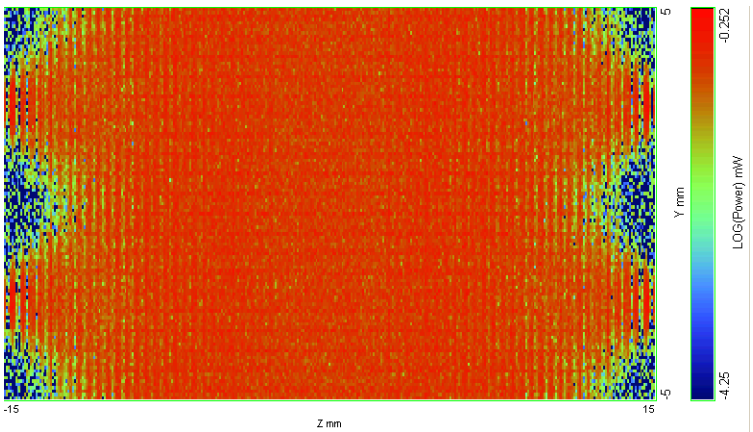


Figure 20.24 The result of **Display: FORM** followed by **Display: PICTURE**. Four decades are displayed. This processing method is also useful for showing results in a manner that mimics the way the human eye would interpret it. The eye is more “forgiving” of power variations; a factor of two is perceived as a fairly subtle change. This view of data used 117 vertical pixels with no averaging. The logarithmic option is particularly useful when your distribution data span a large range of values. It is also useful for approximating the visual appearance of the distribution data, since the human eye is capable of seeing over a large dynamic range. Our response to the stimulus in perceived in a more logarithmic than linear way.

If you choose the logarithm option, you need to specify the number of **Orders of Magnitude** (decades) to display, relative to the maximum pixel value. This number establishes a “floor” to the distribution file and all subsequent graphics, so that the dynamic range of future graphics is not “used up” by displaying a few extremely small values. It also prevents `FORM` from encountering zeros in the data, whose logarithms would be infinite.

See Chapter 20 Appendix, “Script 20-11” on page 455.

Display> Processing> Modify

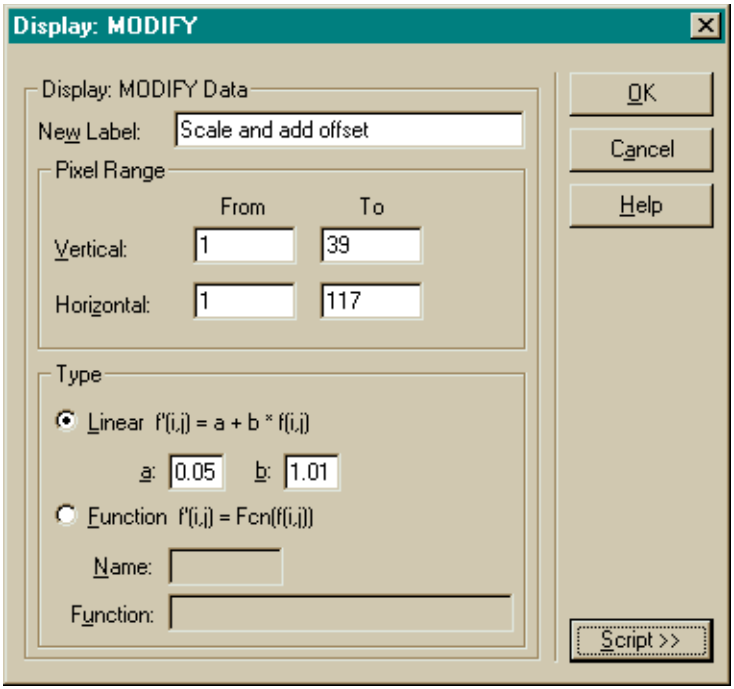


Figure 20.25 Dialog box for **Display: MODIFY**

The `MODIFY` command alters all data or a range of pixels in the current distribution array. The example here shows the use of the **Linear** option. All the values in the distribution file will be altered by multiplying by the constant **b**, and adding the constant **a**. You might choose to do this to make a careful comparison of ASAP results with laboratory measurements that include a small bias and gain factor. If you set **b** to **0**, one or more pixels can be set to the value of **a**.

The second option alters the data according to some other functional relationship. This relationship can be one of the internally defined functions included in the ASAP command language, or a user-defined function. See the

on-line Help topic for **Mathematical Functions** for a list of internal functions. The user-defined function is entered in the form of a **$FCN** macro. See the on-line Help for **$FCN**, and **MODIFY** for details. See also **MODIFY.inr** in the example database.

See Chapter 20 Appendix, "Script 20-12" on page 455.

> ***Note:*** **$FCN** appears throughout ASAP, and behaves differently in each context. In some cases, the user-defined function can have only one independent variable, while multiple arguments are allowed in other cases. When used with the **MODIFY** command, only a single value is expected, and this is the pixel value. It is entered as the underscore (_) character.

If you want to take only the logarithm of the data or raise the values to a power, see "Display> Processing> Form" on page 433. If the modification you need to make to the data is a function of position, you may need to use the **COMBINE** command in conjunction with a user-defined distribution file. See the sidebar, "Creating Your Own Distribution Data" on page 451.

Display> Processing> Normalize

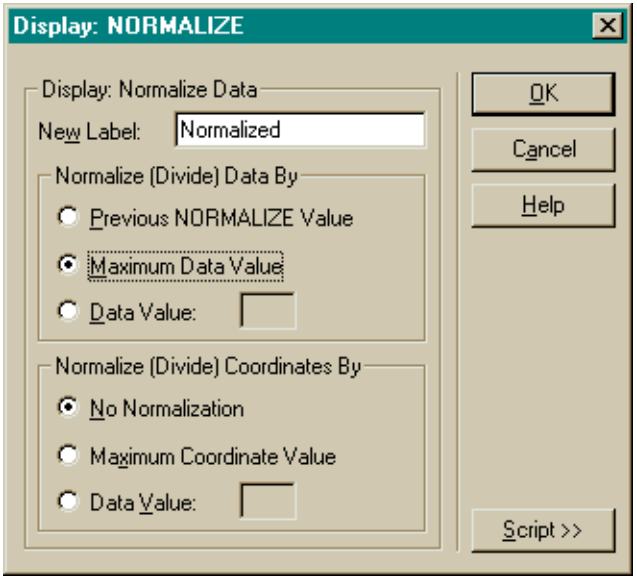


Figure 20.26 Dialog box for **Display: NORMALIZE**

The **NORMALIZE** command divides all values in the current distribution by a constant factor. The most common use is to normalize (divide) by the **Maximum Data Value**. The "peak" then has the value **1**, and all other values are scaled

down accordingly. ASAP also remembers the **Previous NORMALIZE Value,** so that other distribution files can be read and normalized to the same level for direct comparison. You also have the option to normalize to an arbitrary **Data Value**. This number is then divided into all the data.

The **Display: NORMALIZE** dialog box also offers the option of rescaling the coordinate axes, which are the positional (or directional) coordinates specified by the WINDOW command (*y* and *z* in our backlight example). You can normalize both coordinates by only the same factor, but the command language permits changing them independently.

See Chapter 20 Appendix, "Script 20-13" on page 455.

Display> Processing> Offset

The OFFSET command shifts the origin of the spatial coordinate system of the current distribution data. You specify a **Vertical** and **Horizontal** shift. This coordinate shift then shows in any subsequent figure that you produce.

> ***Note:*** The OFFSET command is not an appropriate way to register distribution files that have a systematic coordinate shift. Only the header information that specifies the window setting is changed. The pixel values are not altered in any way.

Display> Processing> Range

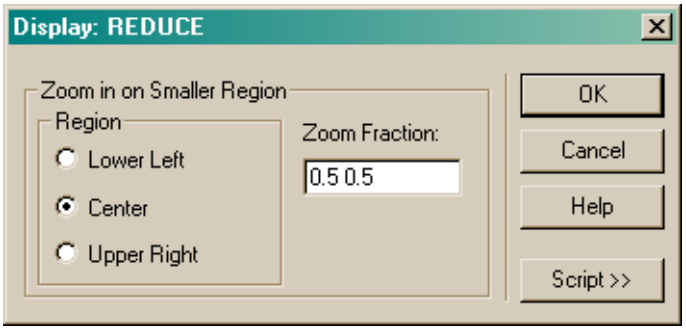
The RANGE command can be used to *increase* the range of the vertical plotting scale in the graphics commands GRAPH, PLOT3D, ISOMETRIC, MESH and CONTOUR. You could use this tool to force more than one distribution data set to be displayed on the same scale. It can be used only to make the plotting range larger, however.

> ***Note:*** If you need to *decrease* the vertical plotting range, use the THRESHOLD command to place a floor or ceiling on the data. THRESHOLD is not currently on the **Display** menus, but can be typed directly into the **Command Input** window. See the discussion of "THRESHOLD Command" on page 444.

Display> Processing> Reduce> Crop

The REDUCE command appears in two forms in the **Display** menu. The **Crop** version reduces the size of the distribution file by eliminating any rows and columns around the outside of the distribution whose elements contain only the array's minimum value (usually zero).

Display> Processing> Reduce> Zoom

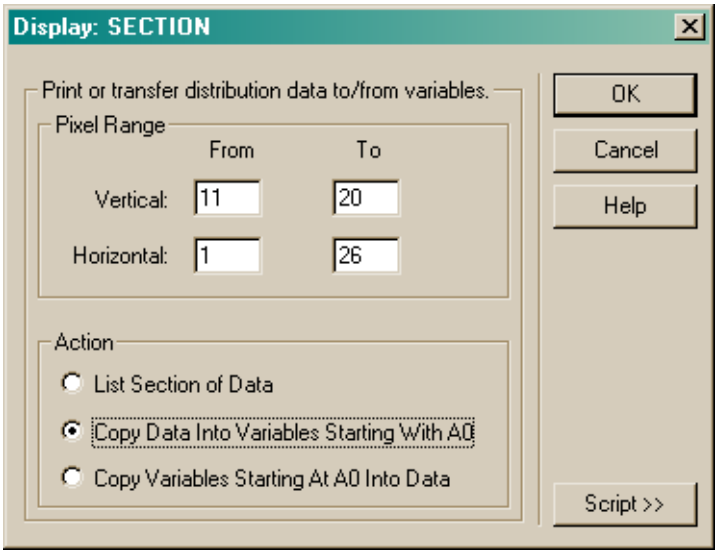


The **Zoom** version of the command selects a subset of the pixels in the distribution data. All other pixels are discarded, resulting in a smaller distribution array. The zoom region can be located at the **Center** of the distribution, **Lower Left**, or **Upper Right**. The size of the subset is determined from the **Zoom Fraction**, which is entered as a fraction of the number of pixels in each dimension of the window. If the distribution is square (has an equal number of pixels in the vertical and horizontal direction), you only need to enter one parameter. Two fractions should be entered for all rectangular distributions, entering vertical first, and horizontal second.

See Chapter 20 Appendix, "Script 20-14" on page 455.

Once again, there is more flexibility to be found in the command script version of `REDUCE`. If you write a command script, or type the command directly from the Command Input window, you have the option of specifying an absolute pixel range, allowing you to pick any square or rectangular subset of the pixels in any region of the distribution file. See the on-line Help entry for the `REDUCE` command for details.

Display> Processing> Section



The **SECTION** command specifies a subsection of the distribution that you would like to list, copy to variables, or replace with new values from variables you have previously set. You specify the **Vertical** and **Horizontal** pixel ranges of interest to you. The vertical range is limited to 10 pixels, and the horizontal is limited to 26. If you work with larger ranges, you need to run **SECTION** more than once to access all the pixels. Remember that pixel (1, 1) is in the lower left corner of the view created by **Display> Graphics> Picture**. In that view, vertical is up, and horizontal increases left to right.

See Chapter 20 Appendix, "Script 20-15" on page 455.

The first option, **List Section of Data**, lists a table of display data values in the Command Output window with row and column headings:

```
--- SECTION 11 20 1 26
 11          12          13          14          15          16          17          18
1  0.54706E-01 0.30388E-01 0.13097E-01 0.31929E-02 0.22988E-02 0.47225E-02 0.11774E-01.
2  0.28685     0.20335     0.94495E-01 0.47342E-01 0.20079E-01 0.72554E-02 0.18298E-03.
3  0.39825E-01 0.15145E-01 0.16906E-01 0.41085E-02 0.97273E-02 0.40027E-02 0.75574E-02.
4  0.17298     0.15320     0.13502     0.71015E-01 0.22104E-01 0.12295E-01 0.22123E-02.
5  0.12164     0.84314E-01 0.63384E-01 0.42751E-01 0.36156E-01 0.20400E-01 0.87769E-02.
  ...
```

As with the output of the **TEXTFILE** command, these values must be rotated 90 degrees counter clockwise to correspond to the Display Viewer presentation.

Note that for larger sections of data, it may be more convenient to use **TEXTFILE** (**Display> File> Textfile**), since there are no limits on the pixel ranges in that case.

The second option, **Copy Data Into Variables**, places these values into a prescribed set of register variables. After running this option of section, you can confirm the result by typing **&REG** in the Command Input window, and reviewing the variable names and values in the Command Output window.

```
--- &REG
V11_H1=5.470554158091545E-2
V12_H1=3.038817271590233E-2
V13_H1=1.309719495475292E-2
V14_H1=3.192948177456856E-3
V15_H1=2.298831241205335E-3
V16_H1=4.72249137237668E-3
V17_H1=1.177440024912357E-2
V18_H1=6.266158074140549E-3
V19_H1=5.307181272655725E-3
```

These values are then available as parameters within the Builder or the scripting language. They persist (even through **SYSTEM NEW** and **RESET**) until the **END** command is issued, or until ASAP is restarted.

> ***Note:*** It may be apparent to you from the above partial list that the pixel index is included within each variable's name. For example, V11_H1 is pixel (11, 1). Remember to enclose the variable name in parenthesis, if you use the value in isolation rather than as a part of a mathematical expression.

The last option, **Copy Variables** replaces a range of pixels with values previously stored in ASAP register variables. The first vertical column must be stored in the registers A0 through A10, the second in B0 through B10, and so forth up to Z0 through Z10. If you are working with the graphical user interface exclusively you most likely will not use this option, because of the difficulty of loading values into the variables. This variation of the **SECTION** command is best suited for replacing a subset of values with numbers computed within a command script. The more direct approach to changing a few specific pixel values using only the menus is by the methods described above under "Display> Processing> Modify" on page 434.

Display> Processing> Table

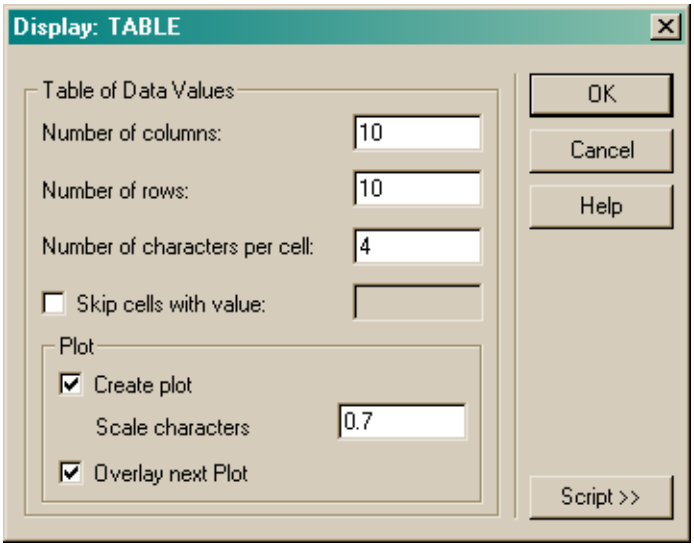


Figure 20.27 Dialog box for **Display: TABLE**

The primary function of the **TABLE** command is overlaying a table of interpolated values onto contour plots. As shown in Figure 20.27, the dialog box produces 10 rows and 10 columns of values, both in the Command Output window, and in a Plot Viewer window (see Figure 20.28 on page 441).

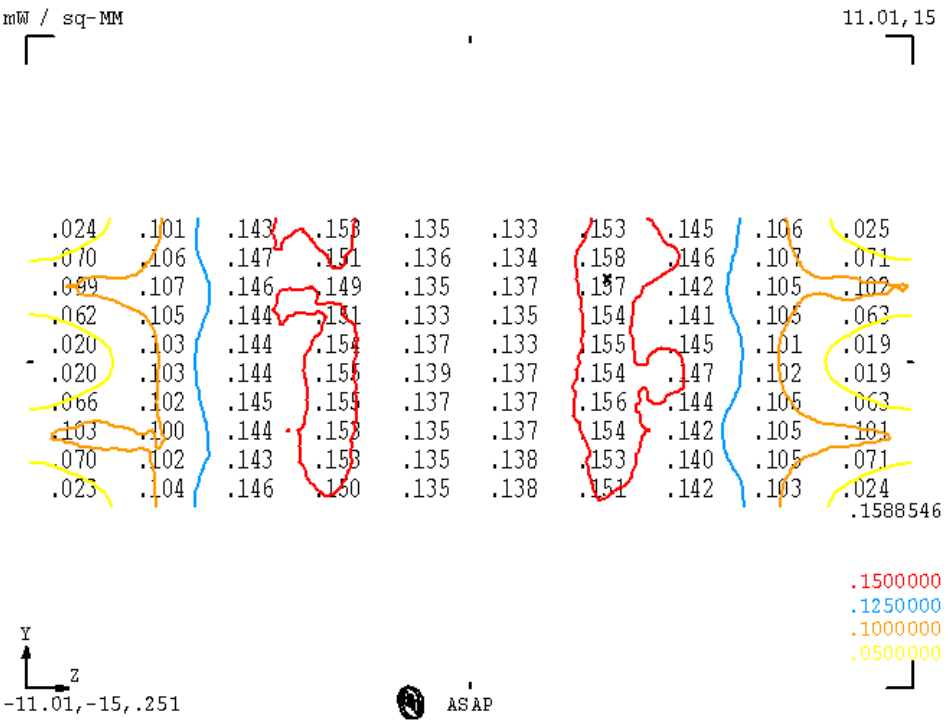


Figure 20.28 Once again, the sense of the terms, rows and columns are reversed in the two instances. The conventional meaning (columns going vertically and rows, horizontally) applies only to the table in the **Command Output** window, while the **Plot Viewer** shows the reverse of this.

Next, you can specify the **Number of characters per cell** to control the amount of precision that is reported. Precision—particularly in the case of the Plot Viewer—must be balanced against the total space available. You can also change the size of the numbers in the Plot Viewer by filling in the **Scale characters** field (default, **1.0**) in the **Plot** section of the dialog box. Often, some experimenting is required to get everything adjusted to your needs.

See Chapter 20 Appendix, "Script 20-16" on page 455.

While you have the option not to create the Plot Viewer output, `TABLE` is rarely used this way. Normally, you will check both the **Create plot** and **Overlay next Plot** options. When you click **OK**, the numbers will be printed in both windows, and you can proceed to use **Display> Graphics> Contour** to complete the figure. You should avoid using the **Draw Tic Marks** or **Draw Grid Lines** options in the **Display: CONTOUR** dialog box to preserve the relative scaling of the two elements of the resulting graphic.

Note: The TABLE command always interpolates. It treats the *center* of the pixels making up the current border of the distribution as the actual boundary, and then interpolates pixel values inward from there even when reporting edge and corner values. This is done so that ASAP never has to extrapolate outside of the pixel centers. Use SECTION or TEXTFILE to interrogate exact pixel values.

Display> Processing> Transpose

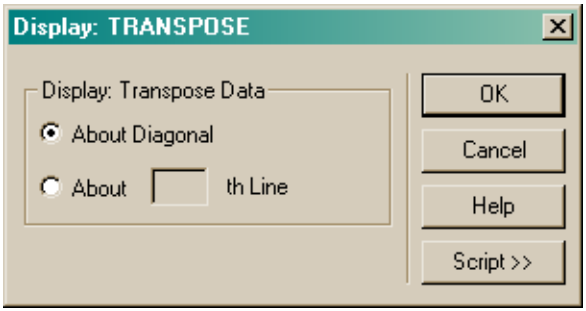


Figure 20.29 Dialog box for **Display: TRANSPOSE**

The TRANSPOSE command allows you to reverse the sense of the two spatial (or directional) coordinates in the distribution: vertical becomes horizontal, and horizontal becomes vertical. The data array is effectively flipped about the diagonal. This operation is often used in conjunction with the GRAPH command to create graphs of the horizontal values (the second coordinate in the WINDOW command).

See Chapter 20 Appendix, "Script 20-17" on page 456.

While using the **About Diagonal** option is the most common use of transpose, you can opt to transpose **About n th Line**, which would be a horizontal line as displayed by the PICTURE command.

Display> Processing> Values

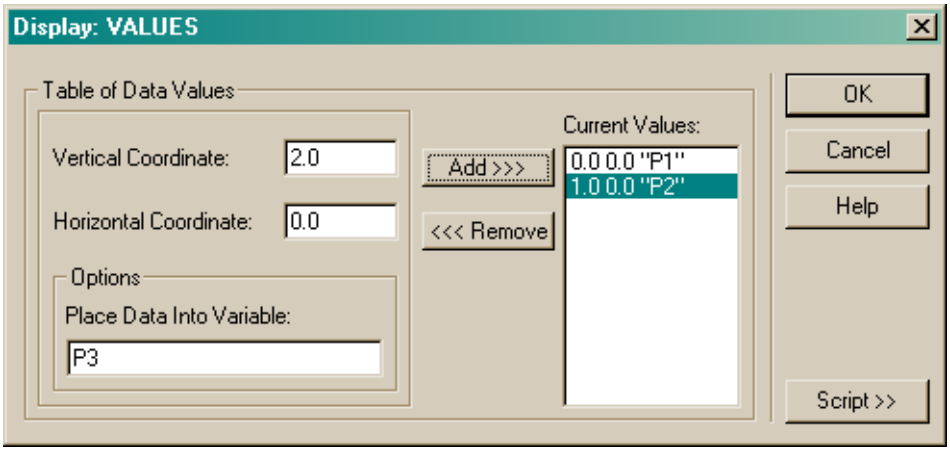


Figure 20.30 Dialog box for **Display: VALUES**

The **VALUES** command prints one or more distribution data values to the Command Output window. You list the **Vertical Coordinate** and **Horizontal Coordinate** of each point that you would like to see, expressed in system units. ASAP will linearly interpolate between the actual pixel values as required. You also have the option to save each value to an ASAP register variable. The **Place Data Into Variable** field allows you to select a name for each designated variable. Just leave this field blank if you do not wish to save the results, then click **Add.**

```
--- VALUES 0.0 0.0 "P1" 1.0 0.0 "P2" 2.0 0.0 "P3"
 Y mm              Z mm              mW / sq-MM
    0.000000          0.000000          0.1392046     P1
    1.000000          0.000000          0.1380654     P2
    2.000000          0.000000          0.1369262     P3
```

See Chapter 20 Appendix, "Script 20-18" on page 456.

Since we saved these values into the register variables **P1**, **P2**, and **P3**, they are now available for future use. For more about ASAP register variables, see "Using Variables and Expressions in the Builder" on page 311 in Chapter 15, and "Using Variables" on page 545 in Chapter 24.

Two additional **DISPLAY** commands are not available from the menus, but can be run by typing them, and their arguments, directly into the Command Input window.

ABEL Command

The **ABEL** command performs an Abel or inverse Abel transform on each line of the current distribution data about a specified horizontal line. It is an important

tool for modeling arc sources. The process begins when **ABEL INVERSE** can take a distribution file derived from a CCD image of the arc discharge (using the **bmp2dis.exe** utility), and turn it into a specification for a three-dimensional volume emitter. The source is presumed to be axially symmetric about a specified horizontal line of the original distribution file. You would normally save the resulting transform (using **Display> File> Save / Write**) for use with **EMITTING DATA**, which produces the actual ray set.

THRESHOLD Command

The **THRESHOLD** command establishes a floor or ceiling on values within the current distribution array. This is useful for limiting the range of data values displayed using graphical commands, like **GRAPH**, **CONTOUR**, **PICTURE**, **ISOMETRIC**, and **MESH**. For example, to display only the data that has values within the range 0.5 to 0.6, type `THRESHOLD 0.5 0.5 0.6 0.6` in the Command Input window. This "clamps" any value less than 0.5 to 0.5, and any value greater than 0.6 to 0.6.

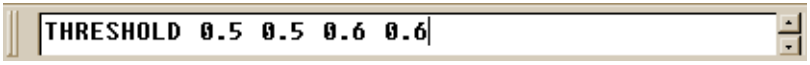


Figure 20.31 Displaying data values, within a range, in the **Command Input** window.

Unlike **RANGE**, the **THRESHOLD** command does alter the contents of your distribution array. You may wish to use **Display> File> Save / Write** to preserve the full range of values before using **THRESHOLD**.

APPEND Command

The **APPEND** command "stacks" distribution files into a three-dimensional "data cube". Stacking is useful for forming a series of views that can be stepped or even animated in the Display Viewer. Let us presume that you have previously used **Display> File> Save / Write** to save two distribution files named, for example, **VIEW1** and **VIEW2**, and that you would like to combine them (and perhaps other views) into a three-dimensional stack named **VIEW_ALL**. Use **Display> File> Open / Read** to recall **VIEW1**, and then **Display> File> Save / Write** to copy the first frame to **VIEW_ALL**. Then use **Open / Read** again to recall **VIEW2**. Form the two-frame stack by typing **APPEND VIEW_ALL** in the Command Input window. You can add more frames by recalling more files and appending them to **VIEW_ALL** in this same way.

Summary

In this chapter, we discussed the following new commands.

ASAP Commands	ASAP Menu	Description
	Display> File:	
DISPLAY	**Display> File> Open / Read**	Read a distribution file into a data array.
WRITE	**Display> File> Save / Write**	Save the current distribution data array to disk.
TEXTFILE	**Display> File> TEXTFILE**	Write the current distribution data array to a text file.
IESFILE	**Display> File> IESFILE**	Write the current directional distribution data to a file in standard IESNA photometric format.
	Display> Graphics:	
ISOMETRIC	**Display> Graphics> Isometric**	Display an isometric projection of cuts through the distribution array in the Plot Viewer.
CONTOUR	**Display> Graphics> Contour**	Produce a contour plot of the current distribution data array in the Plot Viewer or 3D Viewer.
DIRECTIONAL	**Display> Graphics> Directional**	Graph directional distribution data as angles in a polar or unwrapped format.
MESH	**Display> Graphics> Mesh**	Produce a vector version of the distribution data, suitable for displaying in the 3D Viewer.
HISTOGRAM	**Display> Graphics> Histogram**	Display a plot of the number of data values.
PICTURE	**Display> Graphics> Picture**	Display a pseudo-color map of current distribution data. The result is displayed in the Display Viewer.
GRAPH	**Display> Graphics> Graph**	Plot cross sections of the current distribution data array in the Plot Viewer.
ENCLOSED	**Display> Graphics> Enclosed**	Graph the percentage of the total power as a function of the half-dimension of a square.

ASAP Commands	ASAP Menu	Description
PLOT3D	**Display> Graphics> Plot 3D**	Combine an isometric plot with graphical cross sections.
RADIAL	**Display> Graphics> Radial**	Graph the percentage of the total power as a function of circle radius, with an option to replace data.
	Display> Processing:	
ANGLES	**Display> Processing> Angles**	Convert directional data from direction cosines to vertical and horizontal angles.
AVERAGE	**Display> Processing> Average**	Smooth the current distribution data by averaging adjacent pixels.
COMBINE	**Display> Processing> Combine**	Combine the current distribution data with previously calculated results.
FFT	**Display> Processing> FFT**	Compute the Fourier transform of the current distribution data to produce a spatial frequency spectrum or its inverse.
FOLD	**Display> Processing> Fold**	Average the current distribution data around a vertical or horizontal line, or both.
FORM	**Display> Processing> Form**	Replace each value in the current distribution data by the value raised to a specified power or by its natural logarithm.
MODIFY	**Display> Processing> Modify**	Modify each value (or a subset of values) in the current distribution data by an arbitrary function.
NORMALIZE	**Display> Processing> Normalize**	Divide all values in the current distribution array by a constant factor.
OFFSET	**Display> Processing> Offset**	Shift the origin of the spatial coordinate system for the current distribution data.

ASAP Commands	ASAP Menu	Description
RANGE	**Display> Processing> Range**	Increase the range of the vertical plotting scale in the graphics commands, GRAPH, PLOT3D, ISOMETRIC, MESH, and CONTOUR.
REDUCE	**Display> Processing> Reduce> Crop**	Crop rows and columns on the outer boundaries of the distribution if they contain the distribution's minimum value.
REDUCE	**Display> Processing> Zoom**	Select a rectangular subset of pixels in the distribution data array.
SECTION	**Display> Processing> Section**	Specify a subsection of the distribution data that you want to list, copy, or replace.
TABLE	**Display> Processing> Table**	Place a grid of interpolated values into contour plots
TRANSPOSE	**Display> Processing> Transpose**	Reverse the sense of the two spatial (or directional) coordinates in a distribution: vertical becomes horizontal, and horizontal becomes vertical.
VALUES	**Display> Processing> Values**	Print one or more distribution data values to the Command Output window.
ABEL		Compute the Abel or Inverse Abel transform of the distribution data.
THRESHOLD		Establishes a floor or ceiling on values within the current distribution data array.

The DISPLAY commands are a set of tools for displaying and manipulating distribution files. Before these commands can be used, you must create the distribution file using **Analysis> Calculate Flux Distribution** from the **Analysis** menu. The result is a file named **bro009.dat**. Once created, you can load the contents of this file into memory to continue your analysis of how power is distributed spatially or directionally. This loading takes place automatically when you use the **Display** menu to access any of these commands.

The commands can be divided into three broad categories:

Display> File—Use these commands to read data from disk files for viewing or processing, or to write results to disk files.

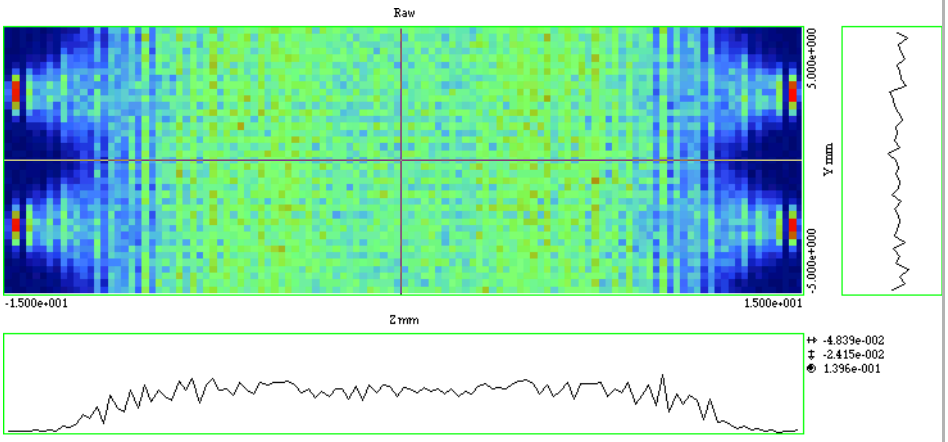
Display> Graphics—Use these commands to display distribution data graphically. Most of the commands do not change the data in any way, though **RADIAL** does offer the option of replacing the original data with radially averaged values. They always display the state of the display data array when the command was issued, and never update an open graphical window. This allows side-by-side comparison of data before and after processing.

Display> Processing —Use these commands to perform various processing operations on the display data array. In most cases, the display data array is permanently altered in memory by these commands. You must reread **bro009.dat** from disk to recover the original, unaltered data.

How many rays, and how many pixels?

If you are new to statistical methods like Monte Carlo simulations, you may find some of the terminology and program options baffling and a little disturbing. There are so many choices, and so many ways to look at the same data. Why can't ASAP tell us which answer is right? In the last analysis, you must have some understanding of statistical methods to use a tool like ASAP. You are ultimately responsible for interpreting and "certifying" the results. Further, the experienced eye of the analyst is often better at making critical decisions about data than any specific algorithm based on formal errors. There are, however, a few simple guidelines that will help you get the experience you need to make these decisions with some confidence.

We can begin by looking again at the three aspects of Figure 20.17 on page 427. Recall that our task is to evaluate the uniformity of illumination on this sample surface. All four panels look different. Which one is correct? Strictly speaking, none of them is. All suffer from having a limited number of rays in the original data. It is likely, however, that we have enough information from this ray trace to quantitatively evaluate the design—if we interpret the results carefully. Looking critically at the first panel of Figure 20.17 in full window view, as shown below,

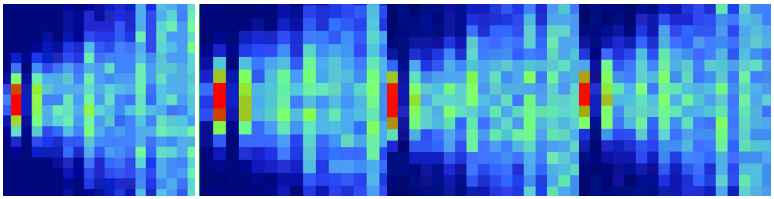


a good deal of pixel-to-pixel variation shows up, both in the **Display Viewer** *and in a graph, through one row or column of data. Is this real? Not likely! There are only 570,000 rays on this surface (which we learned using* **Analysis> Calculate Flux** *with the* **Summary** *option). They are, more or less, spread uniformly over 39 × 117 = 4,568 pixels. (Remember, the* **Command Output** *window shows us the total number of pixels used when the distribution file is loaded into memory.) Hence, there are roughly 120 rays in each pixel. If we further assume that each ray has about the same flux (which is justified in this case), we can expect an uncertainty something like* $120 \pm \sqrt{120}$ *, or about 10% based on statistics. Since that is roughly the RMS pixel-to-pixel variation we see, we can conclude that the pixel-to-pixel variations are "just noise". We must be cautious*

in drawing conclusions about features at or below the 10% level. If you need a 1% result, you need to reduce the noise by a factor of 10, which implies 100 times as many rays!

But what if the rays have wildly different flux levels, and what if they are not uniformly distributed? You can create another set of unique rays in the sources by changing the random number seed, and then repeat the ray trace and analysis. Variations that persist are, most likely, real. Those that do not persist are, most likely, statistical artifacts. A careful comparison of the two or more "independent experiments" tells you something about the uncertainty in all the results.

Next, infer what you can from symmetry. The backlight display is lit by four LEDs, which are symmetrically placed at the edges. This placement implies that there should be symmetry in the result. of our example, look at the area around all four sources. (Two of the sources were flipped horizontally to allow direct comparison.



See the discussion of "Display> Processing> Reduce> Crop" on page 436. All show a "hot spot" near the edge, and there is some evidence of vertical bands. Each LED source had an independent set of random rays, yet still these features persist. You can conclude that they are not statistical artifacts, and must somehow be related to the geometry. Now you can safely do some additional processing to try to "bring the features out". In this case, the `FOLD` *command would be a good choice.*

Creating Your Own Distribution Data

As your experience with ASAP increases, you may eventually have a need to create your own distribution data file. One reason for doing this might be simply to display externally generated data in the ASAP **Display Viewer**, **Plot Viewer**, *or* **3D Viewer**. *Another motivation might be to divide an existing ASAP distribution by a "flat field" frame to correct for effects that vary with position. Distribution files can also be used to create sources with a prescribed irradiance distribution (using* **EMITTING DATA**), *or even to specify a sag table for a complex optical element (using a* **SAMPLED** *surface).*

The example we have developed here assumes that we are making the file for use as a "flat field" correction. We would like to read this information into ASAP so that we can use the **COMBINE** *command to divide these values into other ASAP results. The basic format should look like this:*

```
DISPLAY 0 'Geometrical Ray SPOTS'
"Z"  0.000000 "FLUX / UNIT-AREA",
"X" -2.000000  2.000000  10,
"Y" -1.000000  1.000000   5
$FAST     5       10
  0.984639  0.988578  0.992532  0.996502  1.000488
  0.986608  0.990555  0.996502  1.000488  1.004490
  0.988581  0.992536  1.000488  1.004490  1.008508
  0.990559  0.994521  1.004490  1.008508  1.012542
  0.992540  0.996510  1.008508  1.012542  1.016592
  0.994525  0.998503  1.012542  1.016592  1.020658
  0.996514  1.000500  1.016592  1.020658  1.024741
  0.998507  1.002501  1.020658  1.024741  1.028840
  1.000504  1.004506  1.024741  1.028840  1.032955
  1.002505  1.006515  1.028840  1.032955  1.037087
```

The format is the same as if you had used **Display> File> Save / Write** *to write a distribution file in text (*.**din**) *format. It is often useful to save distribution data that is similar to your needs (same size and window), so you can to use it as a template for entering your own data.*

The first line starts with **DISPLAY 0**. *Had it just contained the* **DISPLAY** *command without the trailing zero, ASAP would simply have loaded* **bro009.dat** *into the distribution data array. The zero is a signal to ASAP that you will provide the data values. The string, delimited by single quote marks (***' '***), is the default title that appears in the* **Display Viewer** *and elsewhere if you do not specify a title. This string can be up to 24 characters long.*

The next line specifies the axis that is perpendicular to the data plane, and the location of the data along that axis. The character inside the double quote marks (" ") is typically

either X, or Y, or Z, although in many cases it is treated as only a label on the axes of graphs and other displays. In those cases, you may specify a string up to 16 characters long. The number that follows the first axis label is the location of the data along that axis. Once again, this may be used only in labels, but in other cases (like **EMITTING DATA***) this becomes the location of the rays when they are later created from this information.*

The third and fourth lines include the information that is usually supplied by the **WINDOW** *and* **PIXEL** *commands when ASAP makes a distribution file. The lines shown would be appropriate for*

```
WINDOW Y -1 1 X -2 2
PIXELS 5
```

$FAST *is a predefined ASAP macro that is used for quickly reading large data arrays into ASAP. It is fast because no values are echoed to the* **Command Output** *window, and only minimal parsing and error checking is performed. The macro tells ASAP to expect 5 columns and 10 rows of data values on the lines that follow. Note that these values are the values at the center of the pixel, not at the corners. The first value in the table is at Y = -0.8 and X = -1.8, not Y = -1 and X = -2.*

If you place these lines into an ASAP script and run the file, ASAP leaves you at the **Display** *prompt. These values were loaded into the display data array, ready to be plotted, processed, and saved in binary format for future use.*

APPENDIX

20A

SCRIPTS FOR CHAPTER 20

The following ASAP scripts are referenced in Chapter 20, “Display Tools”.

Script 20-1

```
!!THE ASAP COMMANDS FROM THE DISPLAY...GRAPHICS...ISOMETRIC PULL-DOWN MENU FOLLOW:
DISPLAY
ISOMETRIC 'Backlight'
```

Script 20-2

```
!!THE ASAP COMMANDS FROM THE DISPLAY...GRAPHICS...CONTOUR PULL-DOWN MENU FOLLOW:
DISPLAY
CONTOUR .05  .1  .125  .125  -.5 TICS 1 1 'Backlight'
```

Script 20-3

```
!!THE ASAP COMMANDS FROM THE DISPLAY...GRAPHICS...MESH PULL-DOWN MENU FOLLOW:
DISPLAY
MESH
$VIEW
```

Script 20-4

```
!!THE ASAP COMMANDS FROM THE DISPLAY...GRAPHICS...PICTURE PULL-DOWN MENU FOLLOW:
DISPLAY
PICTURE
```

Script 20-5

```
!!THE ASAP COMMANDS FROM THE DISPLAY...GRAPHICS...GRAPH PULL-DOWN MENU FOLLOW:
DISPLAY
GRAPH 314 COMMENT 3
'Vertical Coordinate
'Several cuts near
'the left side
```

Script 20-6

```
!!THE ASAP COMMANDS FROM THE DISPLAY...GRAPHICS...ENCLOSED PULL-DOWN MENU FOLLOW:
DISPLAY
ENCLOSED 'Backlight "Ensquared" Power'
```

Script 20-7

```
!!THE ASAP COMMANDS FROM THE DISPLAY...GRAPHICS...RADIAL PULL-DOWN MENU FOLLOW:
DISPLAY
RADIAL BOTH 'Backlight Encircled Power'
```

Script 20-8

```
!!THE ASAP COMMANDS FROM THE DISPLAY...PROCESSING...AVERAGE PULL-DOWN MENU FOLLOW:
DISPLAY
AVERAGE
```

Script 20-9

```
!!THE ASAP COMMANDS FROM THE DISPLAY...PROCESSING...COMBINE PULL-DOWN MENU FOLLOW:
DISPLAY
COMBINE "Samp1.dis" 1
```

Script 20-10

```
!!THE ASAP COMMANDS FROM THE DISPLAY...PROCESSING...FFT PULL-DOWN MENU FOLLOW:
DISPLAY
FFT MODULUS
```

Script 20-11

```
!!THE ASAP COMMANDS FROM THE DISPLAY...PROCESSING...FORM PULL-DOWN MENU FOLLOW:
!!DISPLAY
FORM -4 'LOG(Power) mW'
```

Script 20-12

```
!!THE ASAP COMMANDS FROM THE DISPLAY...PROCESSING...MODIFY PULL-DOWN MENU FOLLOW:
DISPLAY
MODIFY 1 39 1 117 0.05 1.01 'Scale and add offset'
```

Script 20-13

```
!!THE ASAP COMMANDS FROM THE DISPLAY...PROCESSING...NORMALIZE PULL-DOWN MENU FOLLOW:
DISPLAY
NORMALIZE MAX 'Normalized'
```

Script 20-14

```
!!THE ASAP COMMANDS FROM THE DISPLAY...PROCESSING...REDUCE...ZOOM PULL-DOWN MENU
FOLLOW:
DISPLAY
REDUCE 0.5  0.5
```

Script 20-15

```
!!THE ASAP COMMANDS FROM THE DISPLAY...PROCESSING...SECTION PULL-DOWN MENU FOLLOW:
DISPLAY
SECTION 11 20 1 26 GET
```

Script 20-16

```
!!THE ASAP COMMANDS FROM THE DISPLAY...PROCESSING...TABLE PULL-DOWN MENU FOLLOW:
DISPLAY
TABLE 10 10 4 PLOT 0.7 OVERLAY
```

Script 20-17

```
!!THE ASAP COMMANDS FROM THE DISPLAY...PROCESSING...TRANSPOSE PULL-DOWN MENU FOLLOW:
DISPLAY
TRANSPOSE
```

Script 20-18

```
!!THE ASAP COMMANDS FROM THE DISPLAY...PROCESSING...VALUES PULL-DOWN MENU FOLLOW:
DISPLAY
VALUES 0.0 0.0 "P1" 1.0 0.0 "P2" 2.0 0.0 "P3"
```

CHAPTER

21

ANALYZING DIRECTIONAL DISTRIBUTIONS

In this chapter, we will first introduce more new ASAP analysis commands that allow you to investigate how energy is distributed as a function of direction rather than position. The radiometric term used to describe this sort of distribution is *radiant intensity* (power per steradian), while in photometry it is called *luminous intensity* (lumens per steradian or candela). As with positional distributions, this information can be derived from a single pass through the ray table. This time, however, ASAP sorts the rays and sums their fluxes according to their directions rather than their positions.

At the end of the chapter, we will give a brief introduction to ASAP methods that sort rays by both position and direction. This information is necessary if you want to study the visual appearance of objects as discussed in Chapter 18.

Mapping Problem

Perhaps the simplest and most intuitive way to visualize directional distributions is the radiant, or far-field sphere.

Perhaps the simplest and most intuitive way to visualize directional distributions is the radiant, or far-field sphere. Imagine shrinking your optical system until it is so small that the direction vectors of the rays under analysis seem to be radiating outward from a single point. This perspective is appropriate, since we are ignoring the positions of the rays. We are interested only in directions.

Next, consider a hollow sphere centered on this point. If we extend each ray's direction vector until it touches the sphere, the point of intersection specifies the direction of the ray. Every possible direction in three-dimensional space is a point on this sphere. If the sphere has a radius of one unit, the Cartesian coordinates of this point are just the direction cosines of the vector (see the sidebar, "Direction Cosines" on page 156 of Chapter 8).

Since ASAP stores ray directions as direction cosines, placing the ray on the unit sphere is easy. Figure 21.1 shows a directional distribution of rays for an ASAP model of a 9006 automobile headlamp.

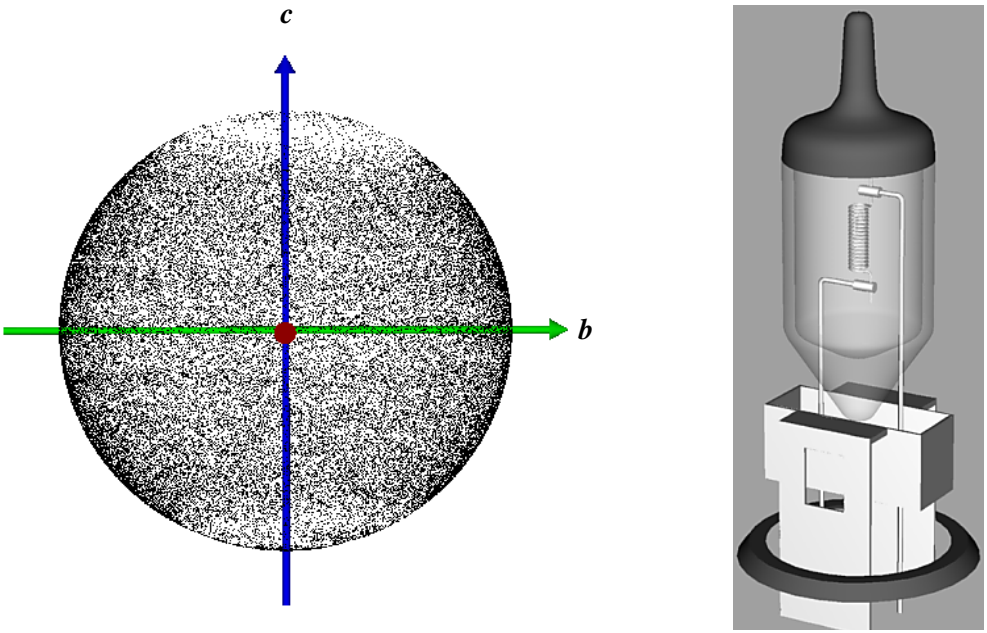


Figure 21.1 Directional spot diagram (left) showing distribution of ray density in direction space for the 9006 headlamp (right). Each point on the surface of the unit sphere represents the direction of one ray. The absence of rays in the +*z* and -*z* directions results from the black-painted top and the mechanical base, which absorb rays headed in those directions. However, the lack of flux information about each point makes it impossible to draw any quantitative conclusion from this information.

We generated the 3D Viewer graphic in Figure 21.1 by selecting **Rays> Graphics> View Directions 3D**. The rays were traced from the filament, through the rest of the bulb geometry until they stopped on the last transmitting object they touched: the outer surface of the glass envelope.

> ***Note:*** Tracing rays to a very large spherical surface is never necessary for far-field analysis. As soon as the rays reach the last object in your system and respond to its `INTERFACE` command to obtain their final direction, their direction cosines dictate exactly where they will hit the radiant sphere.

The directional spot diagram in Figure 21.1 has the same shortcomings as the spatial spot diagram discussed in Chapter 19, which is no information about the flux of the individual rays. The ray density depicted in this way is useful only when all the rays have exactly the same flux, which is generally not true.

The solution, however, is somewhat more challenging than it was in positional analysis. How should we bin data that is located on the surface of a sphere? This is a classic mapping problem. Any attempt to display this information on a flat surface forces us to distort the view. Further, what shape should the bins have? It is not possible to make a useful approximation of a spherical surface using uniform, non-overlapping square pixels.

ASAP offers three methods for performing directional analysis, each solving this mapping problem in a different way.

1. **3D radiant sphere**—uses triangular bins that occupy roughly equal solid angle.
2. **Direction-cosine**—uses square pixels on a unit circle, which is a projection of one hemisphere of the directional distribution down onto the equatorial plane.
3. **Spherical-coordinate**—uses constant increments of polar angle *(θ)* and azimuth angle*(ϕ)* to define the data bins on the surface of the sphere.

We discuss each of these methods in the sections that follow.

Radiant Sphere Method

As always, we begin analyzing rays in direction space by isolating a subset of rays with the `CONSIDER` and `SELECT` commands. For the headlamp example discussed above, we would consider only the outer surface of the glass envelope. This step is the only aspect of directional analysis that pays any attention to the location of the rays. All subsequent steps are concerned only with ray directions.

You can produce the graphic itself from **Rays> Graphics> Radiant Sphere**. The result for the 9006 lamp is shown in Figure 21.2.

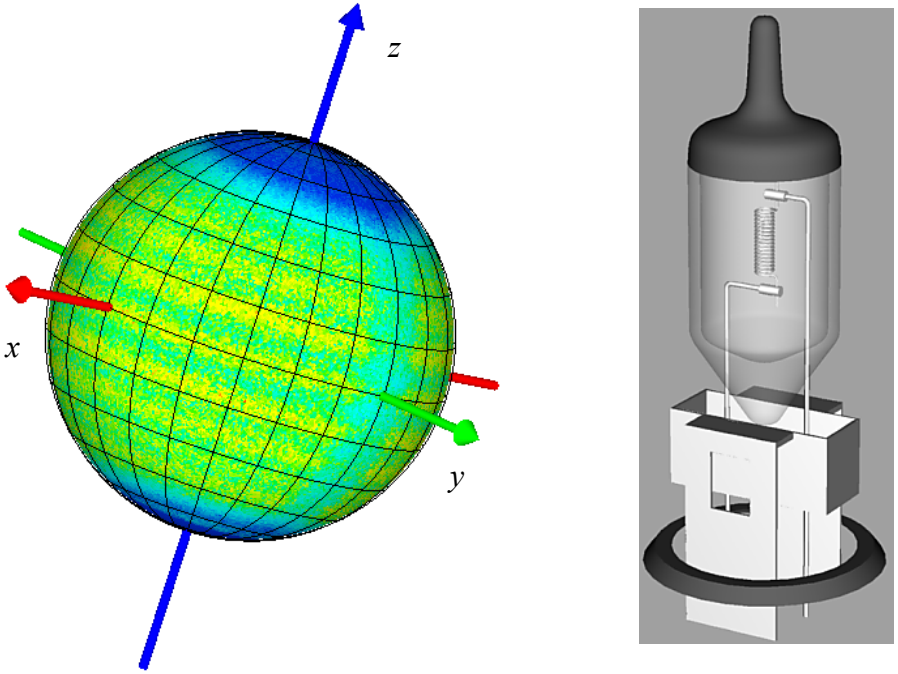


Figure 21.2 Results of the radiant-sphere method (**Rays> Graphics> Radiant Sphere**) to bin and display directional information for the 9006 headlamp. Each dot on the surface of the unit sphere represents the direction of one ray. Now that flux within each solid-angle element has been considered, we can see the shadow due to the vertical filament support structure. It runs from pole to pole through the +*y* axis. We can also see horizontal banding caused by the structure of the filament.

We can interrogate any position on the sphere by clicking in the 3D Viewer window, and reading the radiant or luminous intensity values in the pop-up window.

The graphic contains much more information than the directional spot diagram shown in Figure 21.1.

In addition to mapping the direction vector of each ray onto the radiant sphere, ASAP has also summed the fluxes of all rays falling within triangular bins (Figure 21.3).

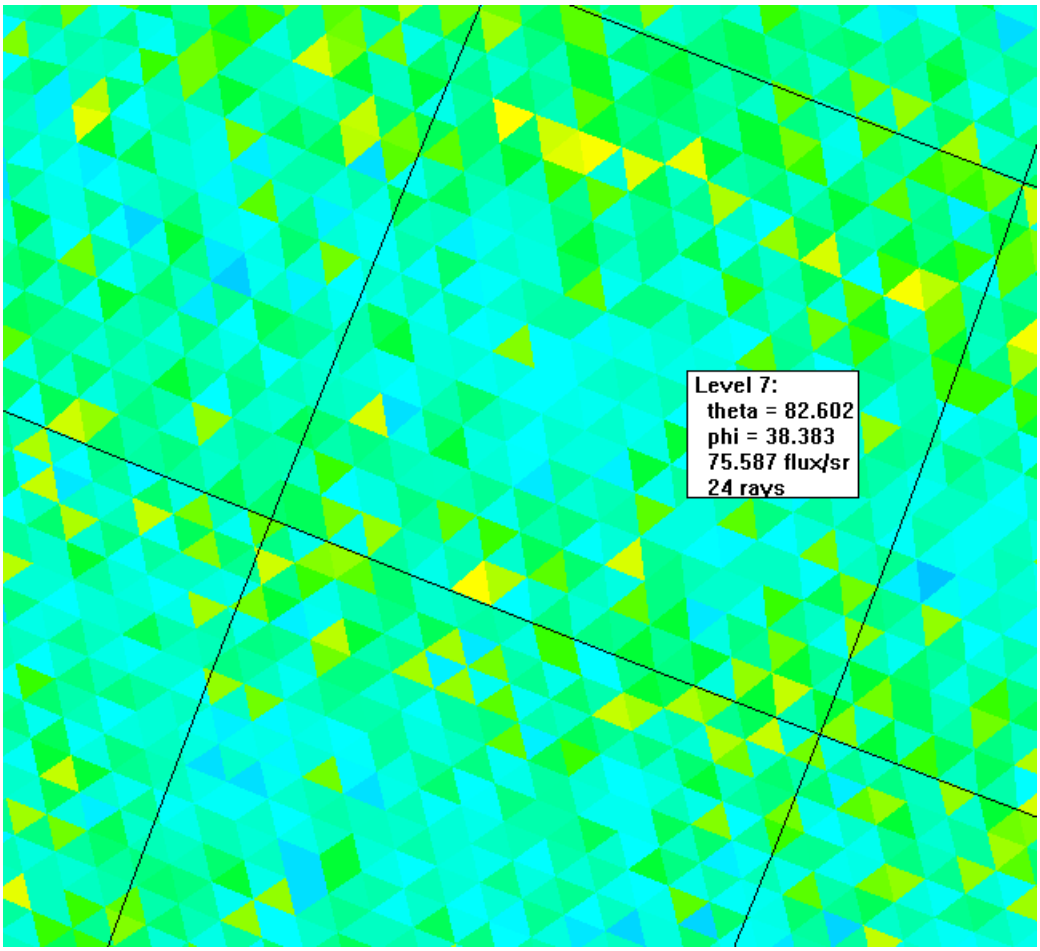


Figure 21.3 By clicking one of the triangular bins that make up the radiant sphere, the spherical coordinates and flux per steradian can be read from the pop-up window. The spherical coordinate system is the same as that defined in Figure 21.14.

We can interrogate any position on the sphere by clicking in the 3D Viewer window, and reading the radiant or luminous intensity values in the pop-up window. The level of tessellation is the only controlling parameter necessary for viewing direction data by the radiant sphere method. This level ultimately controls the number of triangles into which the surface of the sphere is divided.

You can set the maximum level any time the 3D Viewer has focus by selecting **File> 3D Viewer Preferences**, and selecting the **RaySet** tab. Note, however, that

the change does not take effect until the next time you use **Rays> Graphics> Radiant Sphere** to read and sort rays. The default level is five, but level six or seven is not unreasonable if you have in excess of one million rays.

The figures shown here for the 9006 headlamp included about four million rays, and were sorted at level seven. This level does, however, slow down the sorting. It may take several minutes to read and sort the rays, depending on the speed of your computer. Once sorted, however, you may view the maximum level and all lower levels as shown in Figure 21.4.

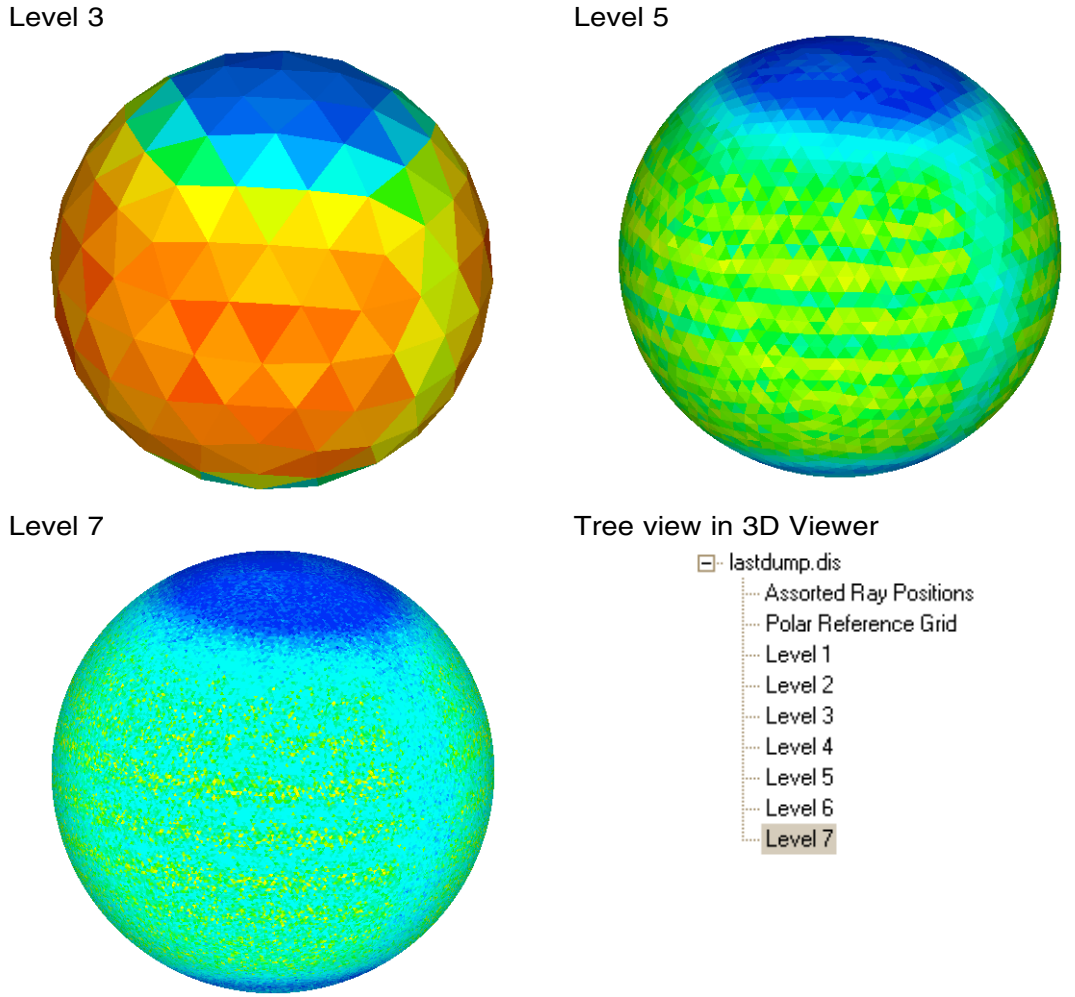


Figure 21.4 All three views were created from the same ray set for the 9006 headlamp. A "level" of tesselation can be selected within the **3D Viewer** tree view (left pane), as if it were a geometric object. You can hide the current level (type the hotkey, **H** after clicking the level in the tree), and making another visible (type the hotkey, **V**). The 3D Viewer always opens with the highest level selected.

Of the three directional analysis methods discussed in this chapter, the radiant-sphere method is perhaps the easiest to use and interpret. It does, however, have disadvantages. The only way to obtain quantitative results is by the point-and-click procedure described above. The individual triangular bins do not organize themselves into convenient rows and columns, but instead are seen to meander around on the surface of the sphere (Figure 21.3 on page 461). They are following the boundaries of arbitrary pentagons and hexagons that form the basis of the tessellation being applied to the sphere. This action makes it difficult to create flat contour plots, tables, and other graphical representations of the data traditionally used to display directional information. However, the next two methods correct this situation, allowing us to use the display tools introduced in Chapter 20.

Direction Cosine Method

As we have already noted, ASAP stores all ray directions internally in the form of three direction cosines. It would be computationally efficient if we could develop a data-binning method that allowed us to work in this native system. What if we simply treated the three direction-cosine coordinates as if they were Cartesian coordinates?

When we were analyzing positional data in Chapter 19, we specified a window that selected two of the three spatial coordinates and plotted the result. This window was the positional spot diagram. In the positional case, however, it made sense to ignore one of the three coordinates. Our rays were generally traced to a flat plane perpendicular to one of the three global coordinates. As a result, one of the coordinates was a constant value for all rays on the analysis plane, and it could be ignored.

In the case of directional analysis, all the rays are located on the surface of the unit direction sphere. What meaning would it have to ignore one of the three direction cosine coordinates? To aid in answering this, consider a simple light-

emitting diode of the type shown in Figure 21.5 along with its far-field power distribution using the radiant-sphere method.

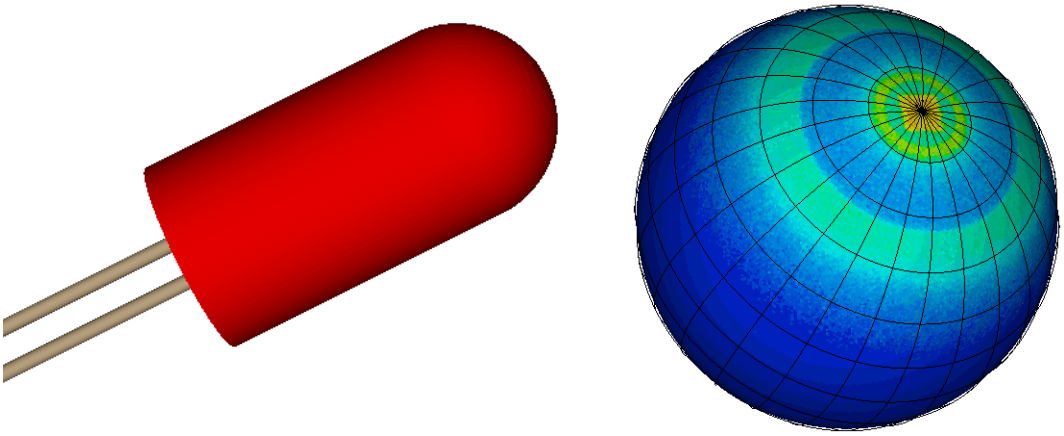


Figure 21.5 The light emitting diode (left) produces a directional distribution like that shown on the right. Such a source has its peak intensity distribution directly along its axis.

The important thing to notice in this particular distribution is its limitation to one hemisphere. In cases such as this, all the directional information is contained in just two of the direction cosines.

Note: Looking at the problem mathematically, recall that the direction vector has unit length:

$$\sqrt{\cos^2\theta_x + \cos^2\theta_y + \cos^2\theta_z} = \sqrt{A^2 + B^2 + C^2} = 1$$

As a result, one of the direction cosines can be written in terms of the other two, although the sign is ambiguous:

$$C = \pm\sqrt{1 - \left(A^2 + B^2\right)}$$

The sign on this direction component determines the hemisphere in which the ray is located; the rest of the information contained in it is redundant. If you know in advance that you are working with a directional distribution that is limited to one hemisphere of the direction sphere, you can ignore one of the direction cosines without loss of information.

By plotting just two of the direction cosines on a flat plane as if they were Cartesian coordinates, we are, in effect, looking at the projection of the unit sphere into the equatorial plane of the sphere. It is now a unit circle (Figure 21.6b).

```
 --- LIST RAYS
!     X            Y            Z            A            B            C            F            S            D        ;$FAST 9 _
 -1.12247     -.518645      9.67280     -.574314     -.329417     0.749432  0.322078E-03 0.409003E-01 0.112059E-01 !       89
 -1.25323     -.379218E-01  9.66286     -.233291     0.600159     0.765104  0.259506E-03 0.879454E-02 0.119472E-01 !       90
  2.35660     0.834538      3.60616     0.605744     0.224312     0.763386  0.316679E-03 0.298142E-01 0.175347E-01 !       91
  2.48151     -.303468      3.49423     0.546569     -.328429     0.770322  0.313277E-03 0.311820E-01 0.170994E-01 !       92
  2.49855     0.850909E-01  3.33027     0.700940     0.220204     0.678375  0.318919E-03 0.288952E-01 0.168965E-01 !       93
```

Figure 21.6a The values in the rectangular box above correspond to the circled areas in views a, b, and c in Figure 21.6b.

.

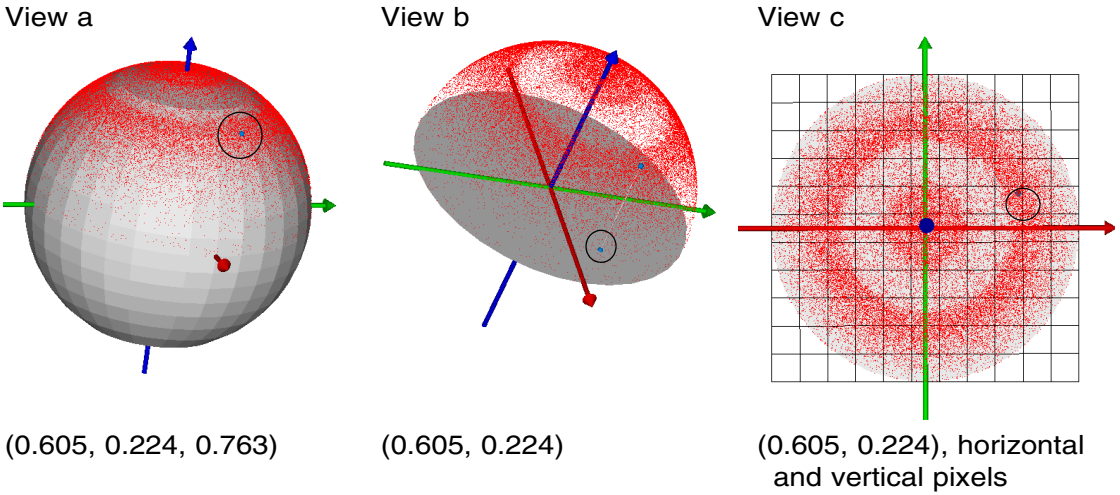


Figure 21.6b a) The A, B, and C values for a given ray are that ray's Cartesian coordinates on the unit sphere. They represent the tip of a direction vector whose tail is at the origin of coordinates. b) When we deal with intensity in the "forward" direction only (emission into a hemisphere), all the direction information is contained in just two of the direction cosine values. We can learn everything we need to know when we look at only the projection of the point down onto the equatorial plane. c) Looking from "above" (down the *z* axis, in this case), we see the projections of all the ray directions. "Sorting by Direction Cosine" is just a matter of binning the data within pixels in the unit circle. A spot in the center of the unit circle represents a ray traveling parallel to the *z* axis in our LED example. Spots near the edge of the unit circle are rays moving at right angles to the *z* axis.

If we divide this unit circle into square pixels, we now have a method similar to the way in which we sorted rays by position in Chapter 19, as well as a way of interpreting its meaning. See the ***Note*** below about the use of the word "intensity" in direction cosine space. We can ask the same questions we asked then, with only minor modifications. What do we need to tell ASAP so that the program can sort rays according to their positions on the unit direction circle?

ASAP needs to know

- which rays to sort, since you are not usually interested in all the rays on all the surfaces in your optical system.
- which "window" to use. If all the ray directions were in the $+z$ hemisphere as is the case for the LED example, the correct window choice would be Y X.
- what directional resolution you want to achieve. In this case, the number of pixels controls the size of the projected solid angle.

Step one, of course, is performed with **Analysis> Choose Rays> Consider** or **Select Rays**. As for the second two steps, the procedure is nearly identical to that used for sorting rays by position. We will use **Analysis> Calculate Flux Distribution** as before, but this time we will choose the second method presented at the top of that dialog box: **Flux per Projected Steradian (Intensity Using SPOTS DIRECTION)** (Figure 21.7).

> ***Note:*** The word intensity is generally used to mean flux/steradian. However, in this direction cosine space, we are using a slightly different meaning of intensity: the flux per pixel where the pixels are in a direction cosine space. The quantity is still power/(unitless quantity). Therefore, to distinguish between the usual meaning of intensity in a coordinate space and "intensity" in direction cosine space, we refer to the latter as a "pseudo-intensity".

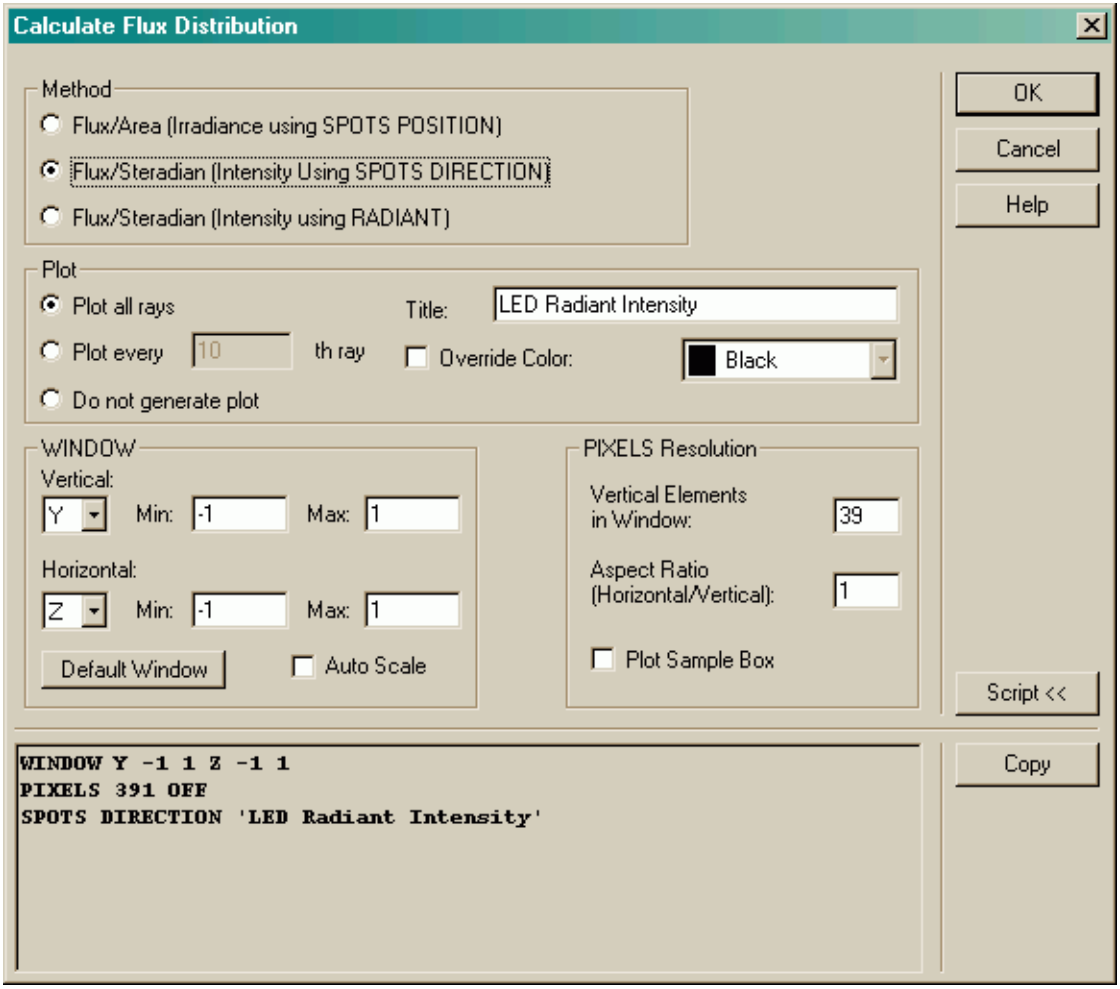


Figure 21.7 Dialog box for **Calculate Flux Distribution** (**Analysis** menu), using the **Flux/Projected Steradian** method, instructs ASAP to sort rays and bin them according to two of their direction cosines.

When we click **OK**, ASAP runs three commands: `WINDOW`, `PIXELS`, and `SPOTS DIRECTION`. ASAP proceeds to sort the direction cosines into the specified number of vertical pixels on the unit circle. The results are stored in **bro009.dat**, and are available for further processing and graphical analysis using the display tools.

Viewing and Interpreting Direction Cosine Results

Since we have projected the surface of the unit direction sphere onto a flat plane, we should expect to see some distortion when we view the results. Consider, for example, rays that are distributed uniformly (isotropically) in

direction. Figure 21.8a shows the directional spot diagram and a graphical cut through the center of the resulting distribution file. The distribution does not appear uniform when viewed in projection.

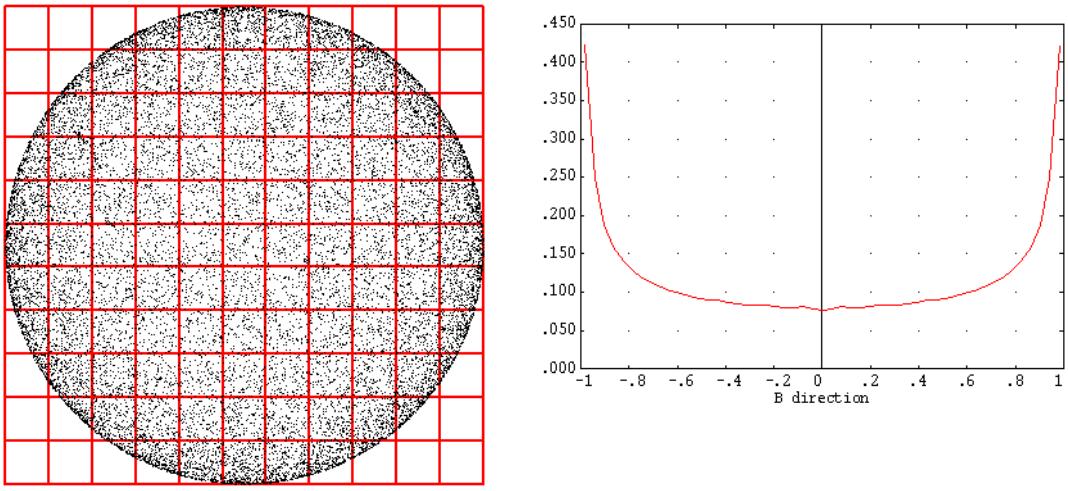


Figure 21.8a The directional spot diagram (left) shows the "pile up" of rays near the outside of the unit circle, even though rays are distributed uniformly on the surface of the three-dimensional unit sphere. A graph (right) of the power distribution through the center of an isotropic distribution in direction cosine space shows the effects of working with pixels of varying solid angle.

A Lambertian distribution, however, does look uniform, even though the flux should decrease as the cosine of the angle from the preferred axis (Figure 21.8b). All these effects are consequences of viewing the results in projection on the unit circle rather than the unit sphere. While the pixels are square on the

unit circle, their projections back up onto the sphere do not subtend a constant solid angle, but increase as we move away from the preferred axis.

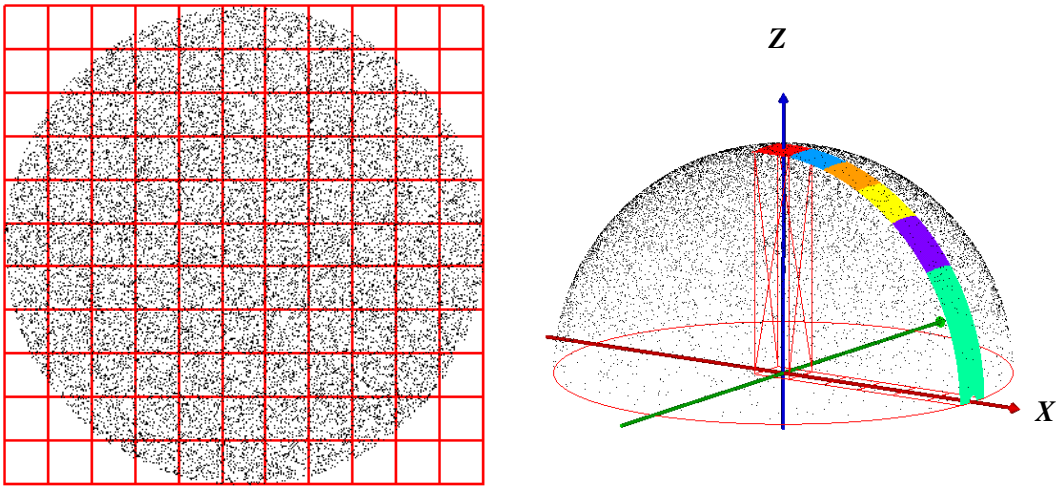


Figure 21.8b The directional spot diagram (left) for a Lambertian flux distribution looks uniform, even though ray density is decreasing as the cosine of the angle measured from the *z* axis. The solid angle subtended by each square pixel on the unit circle (right) is growing as the reciprocal of the cosine of the angle from the *z* axis, balancing the fall-off in ray density for the Lambertian distribution.

Once we understand the consequences of working in a projected coordinate system, we can proceed to view the resulting distributions using the same graphical display tools introduced in the previous chapter. Figure 21.9 shows several views of the LED distribution.

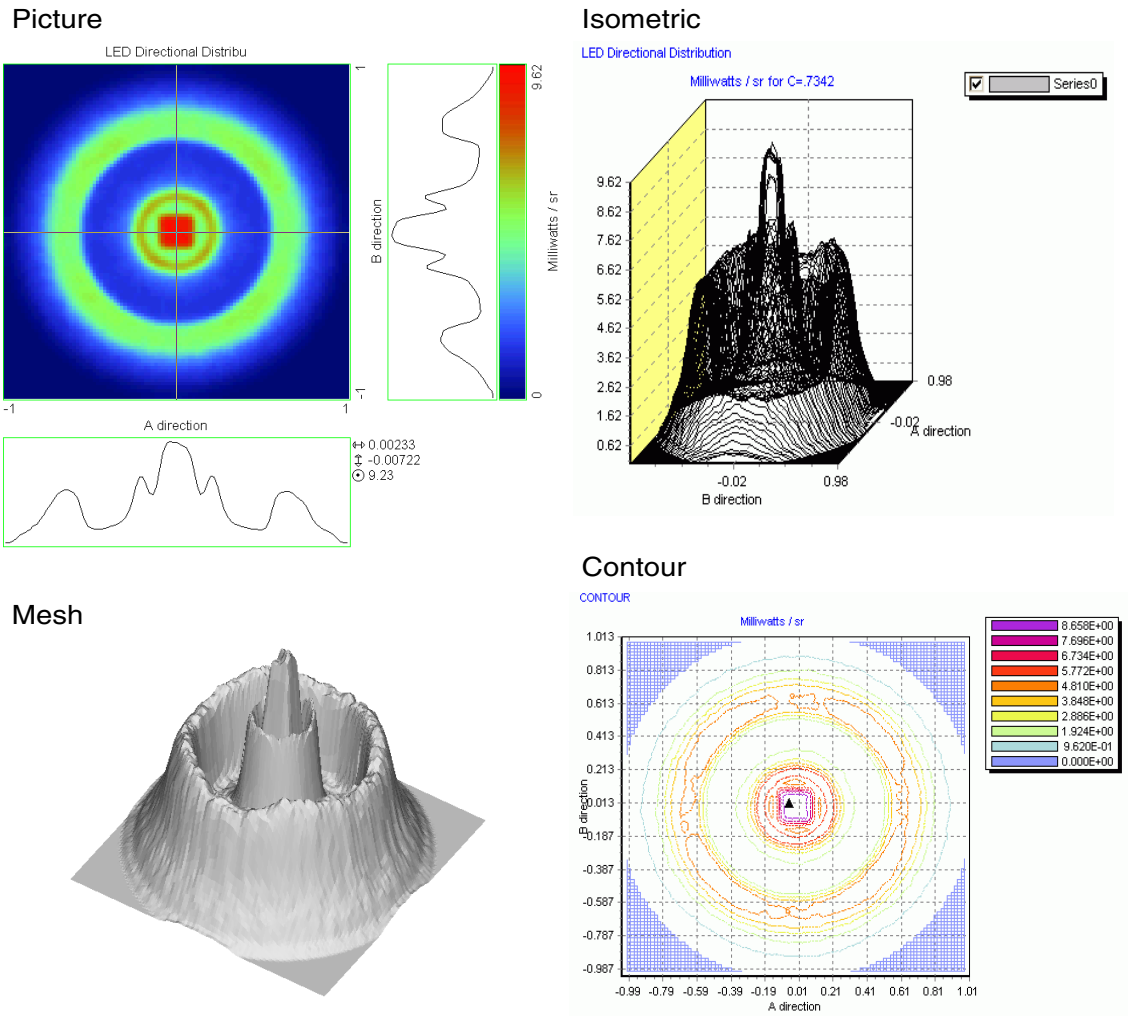


Figure 21.9 All the display tools introduced in Chapter 20 are available for graphing and processing directional distributions. Four examples are shown here.

Two additional tools are included among the display commands, designed specifically for displaying and processing directional data produced by **SPOTS DIRECTION**.

Display> Graphics> Directional

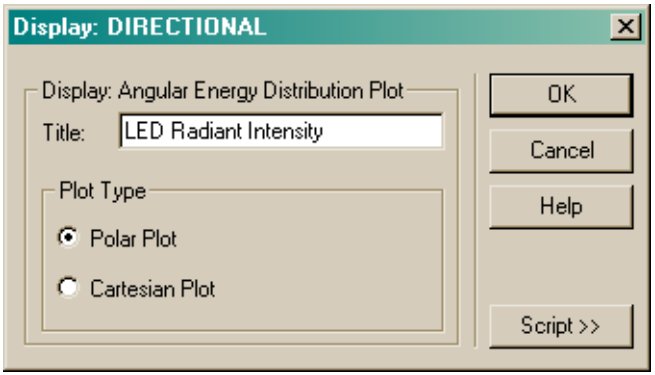


Figure 21.10 Dialog box for **Display> DIRECTIONAL**

The **DIRECTIONAL** command displays plots of directional data produced by the **SPOTS DIRECTION** method, but converts the direction cosine values to angles measured in degrees. This tool displays the data either as a **Polar Plot** or unwrapped to a **Cartesian Plot** (Figure 21.11).

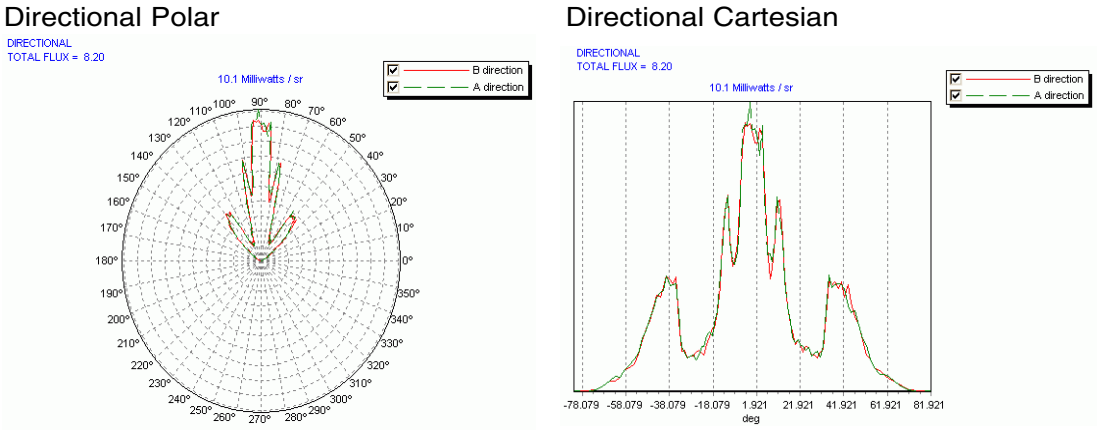


Figure 21.11 The LED pseudo-intensity distribution can be remapped from direction cosines to angles from **Display> Directional**. (Left) Polar plot of two cuts through the remapped data, one along the B direction (shown in red) and one along A (shown in blue). Because of the symmetry of this distribution, there is little difference between the two cuts. (Right) Results of "unwrapping" the same data to display it in a Cartesian system.

In both cases, the graphic shows two curves or "slices" through the distribution, one vertical and one horizontal. The basic distribution data array is not altered in any way; the graphic showing those two slices is the only result of the operation. To convert the entire array to angles so that you can display any slice, see the following section, **"Display> Processing> Angles"**.

Display> Processing> Angles

The **ANGLES** command converts distribution files made using **SPOTS DIRECTION** from direction cosine space into spherical coordinates. No options are associated with the processing command, so no dialog box is displayed. While **DIRECTIONAL** shows a graph of only two slices through the angular distribution, this method converts the entire distribution file, overwriting the RAM copy of the distribution file of the direction-cosine data. The polar axis of the spherical angle coordinate system is assumed to be horizontal (consistent with IES type B photometry), as shown in Figure 21.12 (left). The data are re-binned into this new coordinate system, and then "stretched flat" as shown in Figure 21.12 (right).

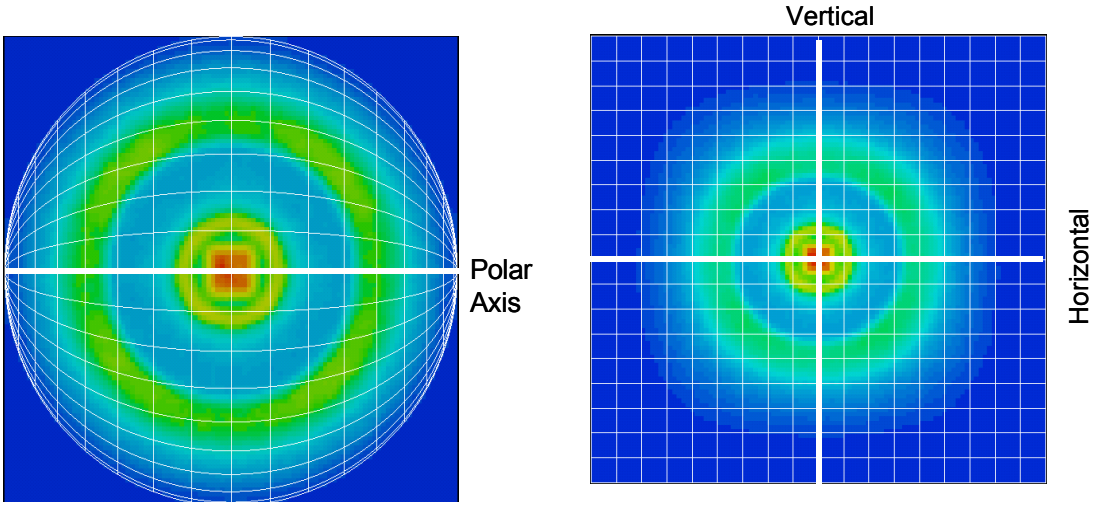


Figure 21.12 The **ANGLES** command re-maps a direction-cosine distribution onto a spherical coordinate system. (left) LED pseudo-intensity distribution previously shown, projected down onto the unit circle in the *x*-*y* plane. (right) Spherical coordinate grid projected on top of the unit circle. The **ANGLES** command re-maps the distribution of these two new bins, and displays them by distorting the curved grid lines into the flat, square pixels shown on the right.

After the **ANGLES** command is applied, you can use other appropriate display tools (like **Graph**, **Isometric**, or **Picture**) to graph or process the remapped results. Figure 21.13 shows another look at the LED contours, this time in terms of horizontal and vertical angels, rather than direction cosines.

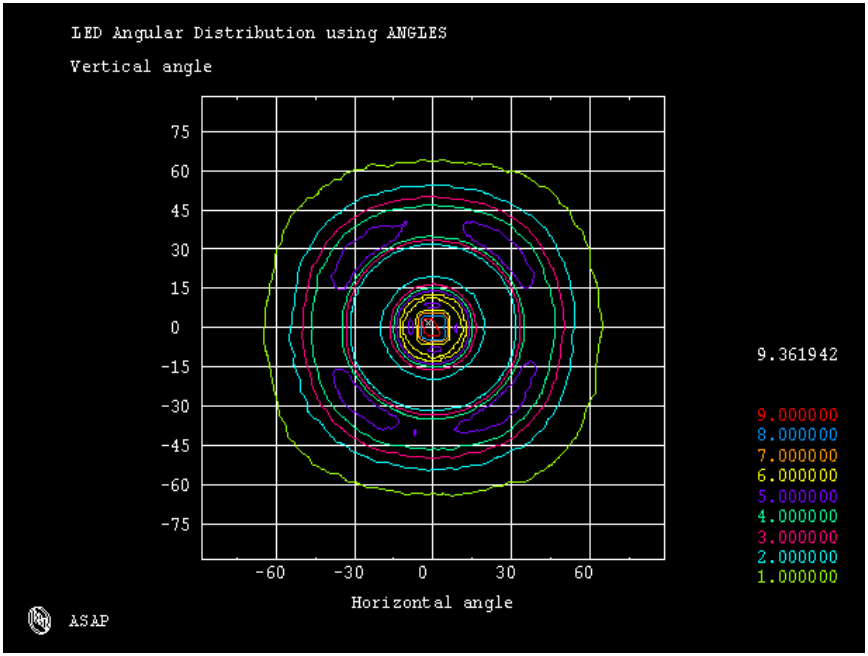


Figure 21.13 After applying the **ANGLES** command to sorted data, using the direction cosine method, the coordinate labels are now "Horizontal angle" and "Vertical angle".

ANGLES (unlike DIRECTIONAL) alters the working copy of the original distribution file. If you need to return to the original distribution, reload **bro009.dat**.

Remember, however, that **ANGLES** (unlike **DIRECTIONAL**) has altered the working copy of the original distribution file. (If you need to return to the original distribution, you can do so by reloading **bro009.dat** from **Display> File> Open / Read**, selecting **Files of Type: All Files**, and browsing to **bro009.dat**.)

> ***Note:*** A given display tool is not necessarily suitable for use with any distribution file type. The **Directional** and **Angles** tools, for example, are intended for use only with distribution files created with **SPOTS DIRECTION**. It is neither necessary nor appropriate to use these commands with distributions made by other methods (such as with the **RADIANT** discussed below, or **SPOTS POSITION** commands), although ASAP does not issue a warning or an error message if you do so. Further, **Directional** should not be used *after* **Angles**, since data have been remapped to a different coordinate system, and the **DIRECTIONAL** command is presuming a direction-cosine distribution. A table of these and other limitations and restrictions is supplied in the "Summary" on page 484.

Spherical-Coordinate Method

Working directly with the direction-cosine ray data with **SPOTS DIRECTION** is easy for ASAP, since it is a simple extension of the positional methods. The

method may have disadvantages from your perspective, however. Visualizing the results in the presence of odd projection effects is difficult enough, although you can overcome some of these using commands like **DIRECTIONAL** and **ANGLES**. But what if you are working with data that are not confined to one hemisphere? The 9006 automotive headlamp, discussed earlier in this chapter, radiates power in every direction, except for a narrow range of angles toward the top and bottom of the bulb. The direction-cosine method cannot be used under these circumstances without using **SELECT** to perform separate "up" and "down" analyses.

ASAP offers an alternative method of performing directional analysis that bins rays into a spherical coordinate system covering the entire direction sphere. The method is based on the **RADIANT** command. In this system, directions in space are specified in terms of the two spherical angles: one measured down from a specified polar axis (the polar or zenith angle), and one around that axis (the azimuth angle). This coordinate system is illustrated in Figure 21.14.

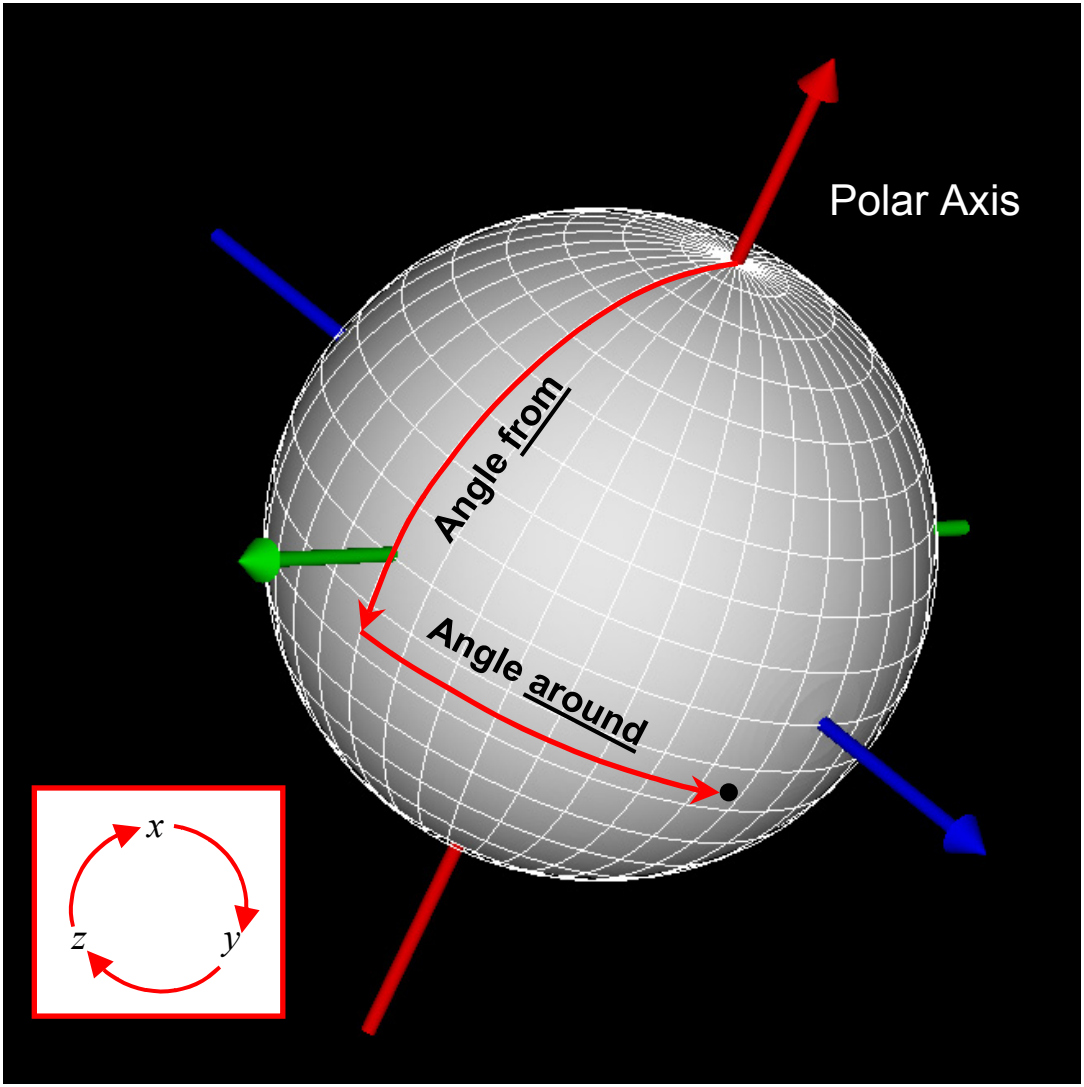


Figure 21.14 Any direction in space can be defined in terms of the angle from a designated polar axis and the angle around that axis. ASAP uses a right-handed coordinate system. This implies that if +*x* is the polar axis, the "angle around" (also called "azimuth") is measured from +*y* toward +*z*. For other polar axes, this convention rotates as implied by the insert. If +*z* is the polar axis, azimuth is measured from +*x* toward +*y*, and so forth.

While the spherical-coordinate method is similar to the 3D radiant sphere method, it differs in two important ways:

1. The spherical-coordinate method divides the surface of the direction sphere into fixed ranges of zenith angle and azimuth angle, while the 3D radiant sphere uses seemingly arbitrary triangles.
2. As a result of this binning method, it is again possible to create a distribution file (**bro009.dat**), and view or process the results with the display tools.

With the **RADIANT** command, the method of binning data results in "pixels" that do not all have the same size (that is, solid angle) or shape. Bins closest to the poles are shaped like pie slices, while those near the equator are roughly rectangular. Still, these bins are all stretched and distorted, as required, to map into a flat rectilinear grid. ASAP converts direction cosines to spherical coordinates, sorts all the rays into the appropriate bins, and reports *flux per unit solid angle* in each bin (thus compensating for the varying bin sizes).

For ASAP to proceed with the sorting of directional ray data by this new method, we must provide the program with four pieces of information:

1. Which rays should be sorted?
2. Which of the three global coordinate axes used by ASAP should be defined as the polar axis of the spherical-coordinate system?
3. What range of angles should be considered?
4. How many bins should be used in the sorting?

As always, you designate the rays from **Analysis> Choose Rays> Consider** or **Select Rays** (question 1).

You are now able to enter the rest of the information (questions 2 to 4) from the **Analysis> Calculate Flux Distribution** dialog box, in the area labeled **RADIANT Options**. This time, however, be sure to select the method named, **Flux per Steradian (Intensity Using RADIANT),** as shown in Figure 21.15.

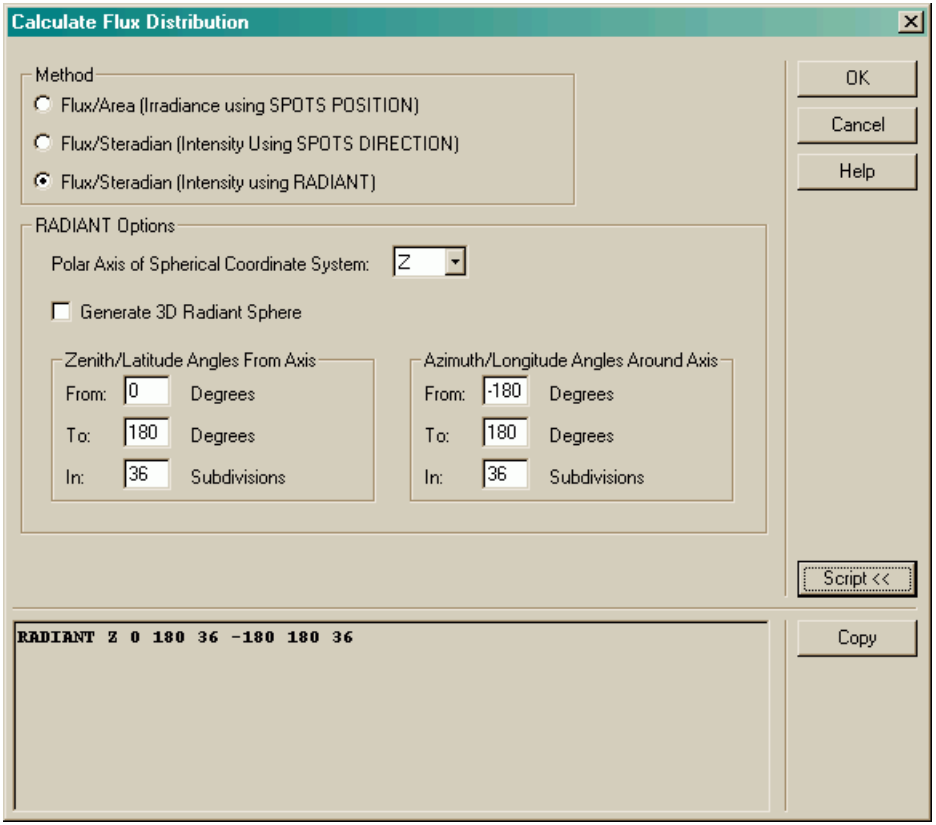


Figure 21.15 Dialog box for **Calculate Flux Distribution** (**Analysis** menu) using the **Flux/Steradian (Intensity using RADIANT)** method. It sets up the parameters necessary to sort the rays into an angular distribution in spherical coordinates.

Choose the polar axis (question #2 on page 476) of your system carefully. As you can see from the previous discussion of mapping a spherical-coordinate system into a rectilinear grid, considerable distortion occurs as you get close to the poles. For this reason, in general, choose your polar axis to be located in a direction where there is little or no important information. For example, the z axis would be an ideal choice for analyzing the 9006 bulb, since little or no flux is expected in either the $+z$ or $-z$ directions. Any polar axis will do, but a clever choice might make it easier to interpret the resulting distribution file visually.

Enter the range of angles to be considered in the **From** and **To** parameters within the **Calculate Flux Distribution** dialog box (see Figure 21.15). While the spherical-coordinate method can perform analysis on the entire direction sphere, you may prefer this method even when only a limited range of directions needs to be considered. Remember that the polar angle is measured from the specified pole downward. Hence, this coordinate is allowed to assume values only between 0° and 180°. Negative angles or angles greater than 180° make no

sense. The azimuth range can, however, extend between 0° and 360°, or -180° to +180° if you prefer. The default values presented in the dialog box utilize the entire direction sphere.

Last, you must enter the number of subdivisions (bins) into which ASAP should divide the flux over the specified angular ranges. Remember, if you specify too many subdivisions, you will have too few rays in each bin. The resulting distribution data will be noisy. Figure 21.16 shows some typical results obtained using the spherical-coordinate method. All coordinate axes are now labeled as angles, relative to the chosen polar axis, measured in degrees.

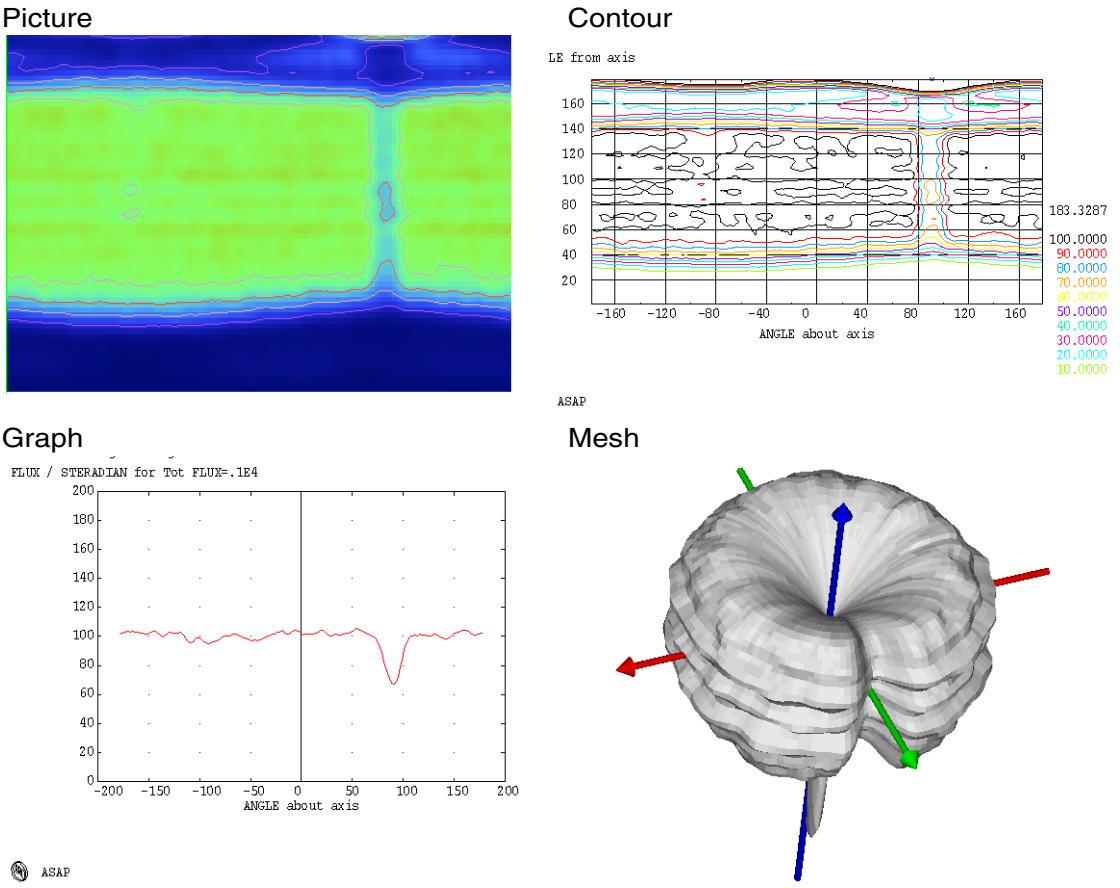


Figure 21.16 After sorting the rays in direction space using the radiant method, ASAP display tools can process and display the results. These graphics of the 9006 automotive headlamp directional distribution clearly show the “notch” caused by the shadow of the vertical filament support structure. Compare these to Figure 21.2 where the radiant-sphere method was used to view the same ray data on the direction sphere.

The top two graphics (made with **Picture** and **Contour**) show the entire distribution file. The "equator" of the coordinate system maps to a horizontal line through the center of these two graphics. The "poles"— *points*, on the direction sphere—have been distorted into *lines* parallel to the equator, the top and bottom boarders of the graphics. This distortion is similar to that which makes the small continent of Antarctica appear to stretch across the bottom of a flat map of the world.

The graphic at the bottom right of Figure 21.16 deserves special comment. It was made using the **MESH** command. We have seen this command used to make three-dimensional representations of distribution files suitable for viewing in the 3D Viewer. See Chapter 20, Figure 20.2 and Figure 20.5, and in this chapter, Figure 21.9. The behavior of the **MESH** command must necessarily be somewhat different when used on distribution files made with the **RADIANT** command, however. Now, distributions extending into both hemispheres must be accommodated.

As the mesh example at the bottom right of Figure 21.16 shows, ASAP is producing a three-dimensional *polar* plot of the angular distribution. While similar to the single-cut polar plot shown on the left of Figure 21.11, we are now able to look at complicated, bi-directional distributions in the 3D Viewer, where is it possible to rotate your viewpoint to see the details. Note that this view does not suffer any of the distortion inherent in the other methods of visualizing distributions made with using the spherical-coordinate method. To make this clearer, the LED pseudo-intensity distribution discussed previously is displayed

in Figure 21.17 using both versions of MESH so that you can study the differences between the two graphics side by side.

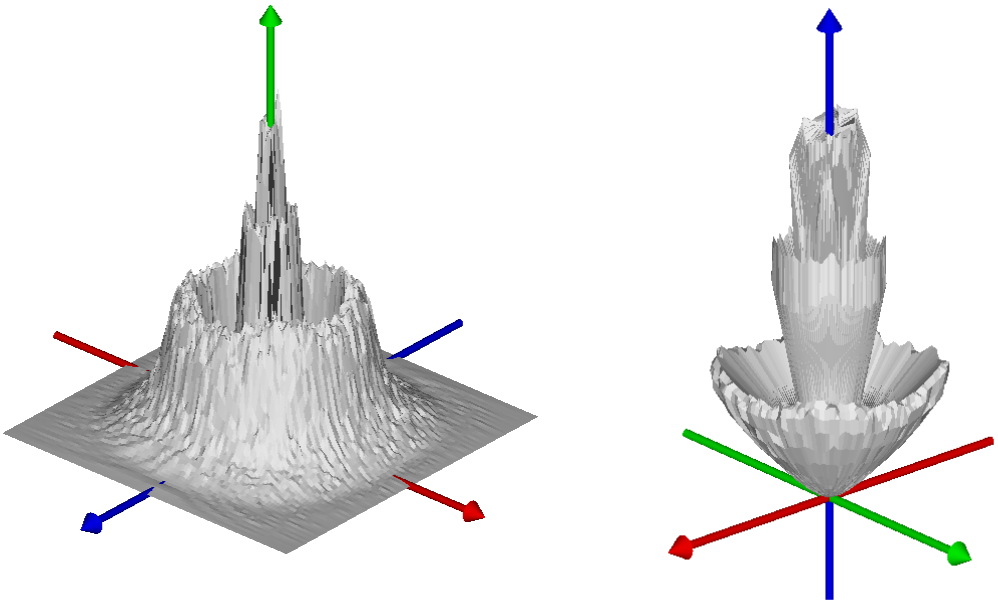


Figure 21.17 The same ray data (from the LED example) was used to make both of these **Mesh** graphics. The left view used the direction cosine method to bin the data, and then applied the **Angles** transformation. The result is a Cartesian display of the data. The right view used the spherical coordinate method. The resulting **Mesh** graphic is a three-dimensional polar plot, in this case, allowing bi-directional distributions to be displayed.

Radiance, Luminance, and View-Driven Radiometric Analysis

Among the most time-consuming radiometric and photometric calculations performed with Monte Carlo ray tracing methods are simulations involving radiance or luminance. Still, this simulation is one of the most important in many fields, because this calculation represents the “lit appearance” of an object (the way it looks to the human eye). It takes a vast number of rays to obtain good results in these situations, because we must bin the data both according to where it originates on the surface of an object, and the angle into which it is redirected after interacting with the surface. The human eye is small—just a few millimeters in diameter—so the angle that it subtends at any reasonable distance is also very small. It might be necessary to trace many millions of rays to get just a handful directed in a specific direction from a specific place on the object.

For this reason, much of the lit appearance work is still done using radiosity calculations, rather than Monte Carlo ray tracing.

As computers become faster, users of programs like ASAP, which were developed for optimum ray-trace speed, are beginning to use them for applications in radiance and luminance calculations. While radiosity calculations are often more efficient, it is difficult to include the kind of realistic source modeling possible with ASAP. Often, detailed source modeling is necessary for accurate, quantitative results.

Most of the ASAP methods for calculating radiance and luminance are beyond the scope of the *Primer*. The difficulty is not only the number of rays that must be traced, but also the amount of data generated by the relevant analysis commands. Previously, only limited tools were available to graphically analyze the results, and no straightforward way existed to interrogate the numerical values stored in the resulting multi-dimensional distribution files. Graphical improvements to the ASAP interface have made most of the analysis tools available from the graphical user interface. We will summarize the tools and techniques available, and refer you to the database of examples for more information on this topic.

The most fundamental tools for performing radiance and luminance calculations with ASAP are options of the **RADIANT** *command. These are* **RADIANT... MAP** *and* **RADIANT... AREA**.

The most fundamental tools for performing radiance and luminance calculations with ASAP are options of the **RADIANT** command, introduced in the previous section. These are **RADIANT... MAP** and **RADIANT... AREA**. Both options use the basic feature of **RADIANT**, which is to bin the data according to direction into a spherical coordinate system. They are both used in conjunction with the current settings provided by **WINDOW** and **PIXELS**, which combine to supply information about the desired spatial resolution. The result is a large, four-dimensional data set. To access a value, you must specify two angular parameters (polar angle and azimuth) and two spatial dimensions (defined by the **WINDOW** command). **RADIANT... MAP** generates results with units of flux per solid angle for each pixel, while **RADIANT... AREA** outputs radiance (flux per solid angle per projected area).

The challenge to using these commands at present is that only one display tool exists for viewing or processing the results (see **asapxxxx\projects\examples\RADIANT_MAP01.INR**).Consequently, you would typically specify angular binning into just one angular element, over a spatial range covering the object of interest. For example,

```
RADIANT Z 85 95 1 40 50 1 MAP
```

specifies a single “point of view” centered on the horizon (polar angle of 90), at an azimuth angle 45 degrees. These methods produce quantitative data in each spatial pixel with the appropriate units. However, they do not produce a rendering of an illuminated object showing us what it will look like.

Perhaps one of the best current approaches to simulating the appearance of an object using ASAP is illustrated in the sample project named, TailLamp

(**asapxxxx\project\samples\TailLamp**). Like all the sample projects, this one performs the modeling tasks using the ASAP scripting language. Still, the basic strategy employed can be understood:

1. **Construct an accurate model** of the all-reflecting tail light assembly that includes realistic scattering properties on the reflecting surfaces (see **stopreflect.inr** in the **TailLamp** project folder). The reflector geometry was developed within a CAD program, and imported into ASAP for analysis using the ASAP smartIGES™ translator.
2. **Define a realistic, detailed source model**. (The 1156 tail lamp from the BRO source library was used in this example.)
3. **Trace the rays**. Because there is no detector in the system, the rays will eventually reach the reflector segments, reflect to obtain their final directions, but remain on those surfaces due to "missed after" ray cessation.
4. **Rotate the rays**. Because the `SELECT` command applied in the next step can isolate rays only in a cone, relative to one of the global coordinate axes, it will, in general, be necessary to rotate the rays so that the desired direction now lies along one of these axes. This step is performed in the file, **vdra_results.inr**.
5. Select only those rays emitted into a narrow cone around a global axis. The *Z* axis was used in this example, and the cone has a half angle of five degrees. See **vdra_results.inr**.
6. **Make a spatial distribution file** with `SPOTS POSITION`. Use the `FORM` command among the display tools to convert the data to a log scale, thus simulating the response of the human eye.
7. **Produce a rendering**, with the `PICTURE` command. The result appears in Figure 21.18.

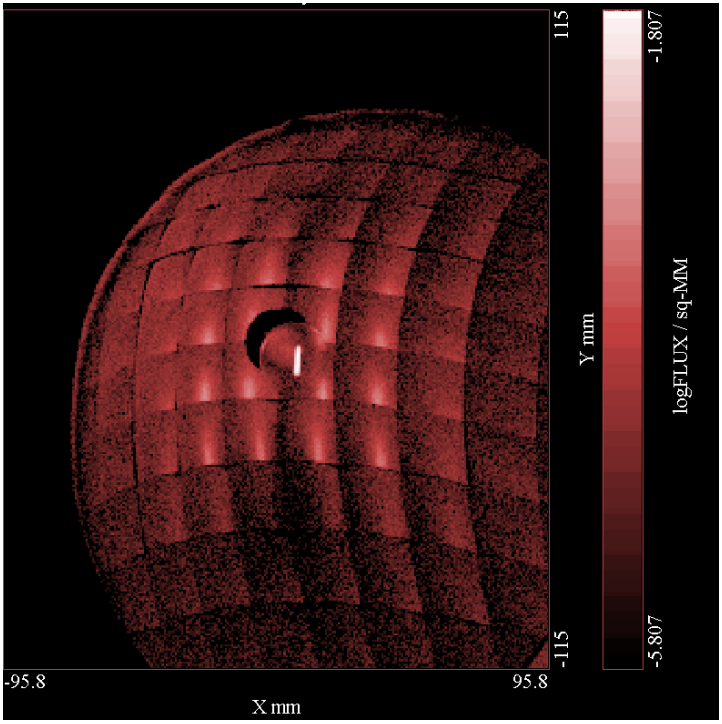


Figure 21.18 Generated with the **PICTURE** command after tracing rays and selecting only the subset that is directed into a specific five-degree cone angle. It approximates the appearance of a segmented automobile tail light, viewed from 15 degrees above and 15 degrees to the left of the assembly. Although noisy, results such as this are often more photometrically accurate than a better looking result obtained by radiosity calculations.

A careful look at the files used to produce this result shows that it took 40 million rays to produce these admittedly "grainy" renderings. Further, the cone angle selected was five degrees, which is significantly larger than the angle subtended by the pupil of the human eye viewing the tail light from a realistic distance. For this reason, we call this analysis technique "View-Driven Radiometric Analysis (VDRA) rather than "lit appearance". Even 40 million rays are insufficient to accurately simulate lit appearance by Monte-Carlo ray tracing methods. But with faster computers and the parallel processing made possible by the **Remote** feature in ASAP/Pro, analysis like this is now possible, if not easy.

Summary

In this chapter, we have discussed the following new commands:

ASAP Commands	ASAP Menu	Description
DUMP $VIEW LASTDUMP.DIS	**Rays> Graphics> Radiant Sphere**	Show directional power distributions on the surface of the unit sphere in 3D.
WINDOW... PIXELS... SPOTS DIRECTION...	**Analyze> Calculate Flux Distribution**	Create a directional distribution file using the direction cosine method. Similar to **Rays> Graphics> Plot Directions 2D**, except a distribution file is created, and the graphics are optional.
DIRECTIONAL or DIRECTIONAL UNWRAP	**Display> Graphics> Directional**	Create a polar or Cartesian graph of two cuts through the intensity distribution.
ANGLES	**Display> Processing> Angles**	Transform the entire distribution file from direction cosines to spherical angles.
RADIANT	**Analysis> Calculate Flux Distribution**	Create a directional distribution file using spherical coordinates.
RADIANT... MAP	(Not available)	Create radiance and luminance distributions.
RADIANT...AREA	(Not available)	Create radiance and luminance distributions.

We introduced three different methods for analyzing directional ray information (power per steradian or projected steradian). Each one has advantages and disadvantages. The important differences are summarized in the following table.

Radiant Sphere	Direction Cosine: SPOTS DIRECTION	Spherical Coordinate: RADIANT
The 3D Viewer performs the sorting from a dump file of ray data.	The ASAP kernel performs the sorting, using the information in the ray file (virtual.pts).	The ASAP kernel performs the sorting, using the information in the ray file (virtual.pgs).
Sorting is performed into triangular areas on the surface of the direction sphere.	Sorting is performed into bins of constant direction cosine increments.	Sorting is performed into bins of constant increments of polar and azimuth angle.
Angular resolution is defined in terms of predefined levels, which are part of the 3D Viewer setup.	Needs WINDOW and PIXEL settings to define angular range and resolution.	WINDOW and PIXEL settings are not used. All information is supplied as part of the RADIANT command.
The method works over the full direction sphere.	The method is limited to analysis within one hemisphere.	The method works over the full direction sphere.
No distribution file is made; display tools cannot be used.	ASAP produces a bro009.dat file for analysis with the **Display** tools.	ASAP produces a bro009.dat file for analysis with the **Display** tools.
Results can be read out in terms of spherical angles by clicking the 3D Viewer window.	Use the display commands DIRECTIONAL or ANGLES to convert to spherical angles.	Data are already in spherical coordinates. Never use DIRECTIONAL or ANGLES commands with RADIANT data.
No graphs are possible.	Use DIRECTIONAL to get a two-dimensional polar plot.	Use MESH to get a three-dimensional polar plot.
No mesh plots are possible.	Mesh plots are always Cartesian.	Mesh plots are always polar.

Not all display tools in ASAP are compatible with every distribution file type. Some of the **Display** menu commands are designed to work on a particular type of data, and will produce misleading, inaccurate, or meaningless results when applied to other distribution files. *ASAP does not issue error or warning messages under these circumstances*, however. The following table summarizes

most of these situations. A note is provided to explain where the command is not appropriate.

Command Method	SPOTS POSITION	SPOTS DIRECTION	SPOTS DIRECTION WITH ANGLES	RADIANT
ANGLES	Not appropriate[1]	√	—	Not appropriate[2]
DIRECTIONAL	Not appropriate[3]	√	Not appropriate[4]	Not appropriate[5]
RADIAL	√	√	Not appropriate[6]	Not appropriate[7]
ENCLOSED	√	Interpret carefully[8]	Not appropriate[9]	Not appropriate[10]
MESH	Produces Cartesian plots	Produces Cartesian plots	No mesh plot produced	Produces polar plots

Notes:

1 and 2: The ANGLES command is designed to convert direction cosines to spherical coordinates. Only the SPOTS DIRECTION analysis command (the direction cosine method) produces distribution files in this form.

3, 4, and 5: The DIRECTIONAL command, like ANGLES, presumes that the distribution was binned in terms of direction cosines. It will not produce meaningful results under any other circumstances.

6 and 7: The RADIAL command is performing radial averaging in a way that presumes rectilinear data of the type provided only by SPOTS POSITION and SPOTS DIRECTION. Still, it is frequently desirable to produce enclosed energy plots in terms of degrees rather than direction cosines. Two methods for obtaining such plots are offered in the examples database. The first uses the ASAP macro language to manually bin the data within specified angular ranges using SELECT, and the second (more efficient) example uses the RADIANT command to bin the data, and then manually integrates the result using the VALUES command. See **radial_angles_enclosed01.inr** and **radial_angles_enclosed02.inr**.

8: While mathematically appropriate to calculate “ensquared” energy in a direction-cosine distribution, it is difficult to imagine many situations where this would be useful on the unit circle.

9 and 10: As with RADIAL, the ENCLOSED command presumes a rectilinear coordinate system.

Exercise 10: Intensity of the Emitting Helix

Analyze the intensity distribution (power as a function of direction) for the emitting helix used in the flashlight model (“Exercise 9: Simple Flashlight Model” on page 394). The first steps below are identical to those used to create the rays.

1. Create an emitting helix using **Rays> Emitters> Helix** with the following parameters:
   - *Orientation:* Along the *y* axis
   - *Total length:* 2 cm centered at the global origin
   - *Helix diameter:* 0.4 cm
   - *Wire diameter:* 0.04 cm
   - *Number of turns:* 10
   - *Number of rays:* 1,000,000
2. Use **Rays> Ray Control> Ray Modifiers> Flux Total** in the Builder to set the total power of the source at 2 Watts.
3. Run the Builder file to create the source. Watch for the return of the ASAP prompt on the status bar before continuing.
4. Verify the source. Use **Rays> Graphics> Plot positions 2D** on the main menu, selecting the **Y-Z** window. Check the length and diameter of the helix by holding down the **Shift** key while moving the mouse to read the cursor position within the Plot Viewer.

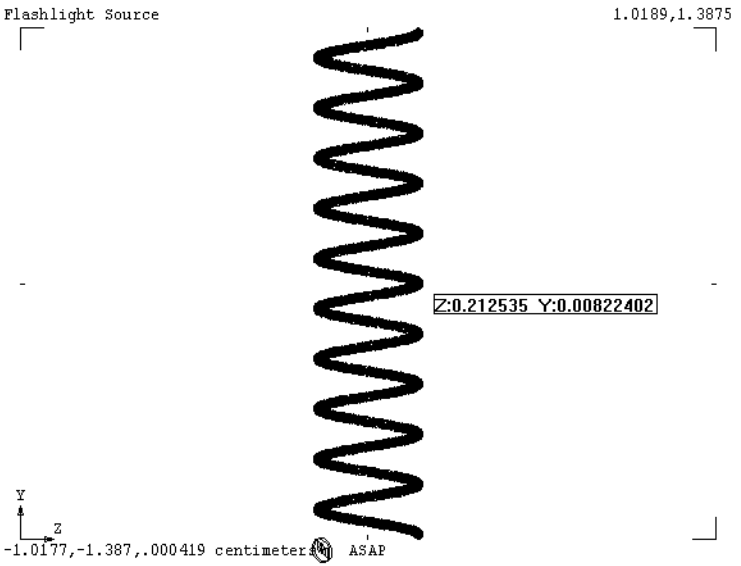


Figure 21.19 Verifying the location and dimensions of the source

5 Compute the directional flux distribution using the spherical-coordinate method. Chose **Y** as the polar axis. Include the entire directional sphere, with 36 subdivisions “from axis” and 72 subdivisions “around axis”.

6 Average over one adjacent pixel using **Display> Processing> Average**.

7 View the result using **Display> Graphics> Picture**.

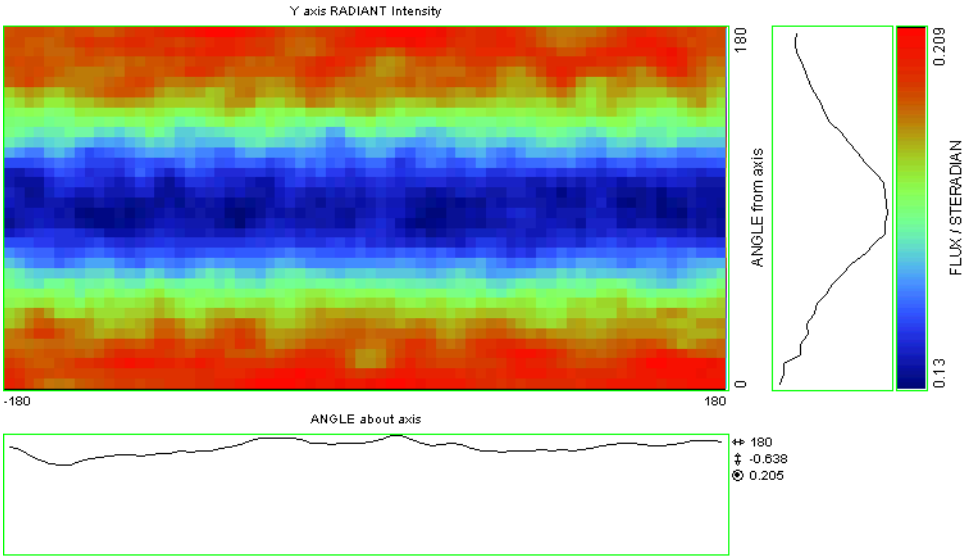


Figure 21.20 Calculating the directional flux distribution with the spherical-coordinate method. This **Display Viewer** window was generated after averaging each pixel value over 1 adjacent pixel.

8 View the result using **Display> Graphics> Mesh**.

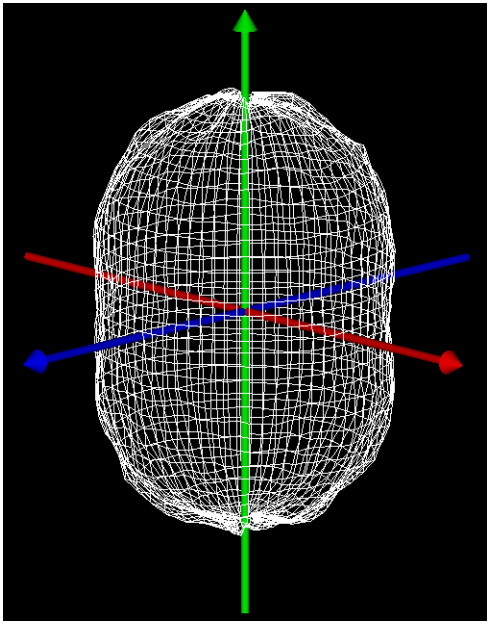


Figure 21.21 The directional flux distribution displayed as a mesh

9 View the result using **Rays> Graphics> Radiant Sphere**.

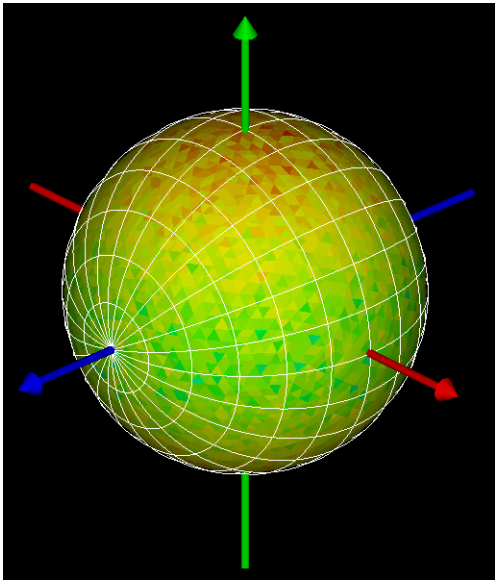


Figure 21.22 The directional flux distribution created with the radiant sphere method, and viewed in the **3D Viewer**

Some questions to test your understanding

- You were not instructed to trace rays in this exercise. Why was this step unnecessary?
- Where is the pole in the result shown in step 7? Where is the equator?
- What is the direction of maximum intensity (along which axis)?
- The figures below show the directional distribution for a *solid* helical filament. The model was built by creating a helical wire in RhinoCAD™ with the same dimensions used above. It was then imported into ASAP via the ASAP smartIGES™ translator, and turned into an emitting object. In this case, the rays were traced so that they could interact with the helix object, modeled as absorbing. What causes the dips in intensity along the **Y** axis, and why is this feature not present in the results obtained above? (Hint: you may want to review the discussion of the characteristics of an emitting helix in Chapter 16.)

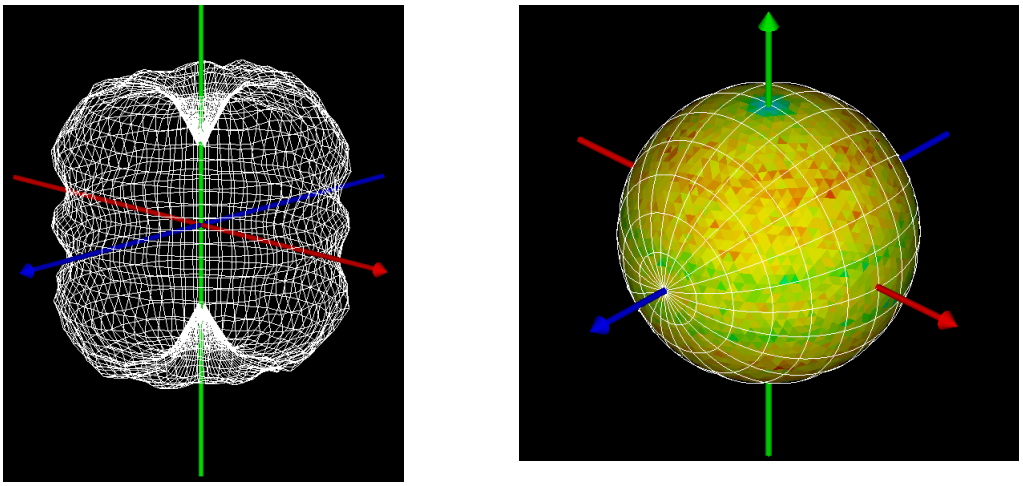


Figure 21.23 The directional flux distribution with a more realistic helical filament. The 3D view on the left shows the mesh graphic, while the view on the right shows the 3D view of the radiant sphere.

CHAPTER

22

WRITING COMMAND SCRIPTS

In this chapter, we will demonstrate how to use the command language to do everything.

Up to this point, you have probably performed all your ASAP work using the Builder and the pull-down menus. This method is an excellent way to get started with ASAP, and, for many problems, it is all you will ever need. There is less to learn this way since the various menus remind us of the names of the ASAP commands, and dialog boxes help us assemble the required information to run the commands.

Before long, however, many new users begin to want to write a simple "script" that would allow them to run a series of ASAP commands in a planned order. Then, when they want to repeat the steps, making only small changes to the basic procedure, it would be easy to edit a few lines of the script and run again. Such a script would run more quickly, since manual communication through mouse and keyboard is far slower than interpretation of a script of commands in the language of the ASAP kernel. The ASAP command language provides this scripting capability.

Using ASAP scripts is by no means an all-or-nothing proposition. Once you acquire a basic understanding of the ASAP command language, you just have more options. Many users develop a style that mixes the best features of the graphical user interface with the power of the command language. Your own style will change and adapt as your experience increases.

In this chapter, we will first discuss basic ASAP script files. Although you may choose to continue to perform parts of your work using the Builder and menus, we will demonstrate how to use the command language to do everything. You may pick and choose how much of this method you would like to adopt at this time.

Recall that every ASAP project has four elements:

1. Build the system model
2. Create sources (rays)
3. Trace the rays
4. Perform the analysis

We will describe how to accomplish each of these tasks in a single ASAP command script that defines a singlet lens and a grid of rays, and traces them for analysis on a detector.

Making the Transition

ASAP offers several built-in aids to making the transition from the Builder and graphical user interface to command scripts:

Script templates. Many common tasks that involve more than one ASAP command have been made available as "script templates". You can insert various templates into your editor window, and "fill in the blanks" with the specific commands and parameters you need for your specific application.

Script commands. You have likely noticed that most dialog boxes associated with the ASAP menus include a **Script** button. The button expands the window to show the equivalent script commands that will be sent to the kernel when you finish entering parameters. If you do not know or remember the commands needed to perform an operation that can also be done from the menus, just cut and paste the commands from the dialog box into your script file.

Mini Builder. This optional feature in the Editor offers many advantages of the Builder as an aid to entering script lines in the Editor window. It gives you menu-based access to most commands and labeled headings for the arguments, while you assemble the command line. ASAP translates the line into the command language syntax, and transfers the result into the Editor window when you drag and drop the line from the Mini Builder to the Editor window.

Command Tips. A windows-style Command Tips aid you in getting the syntax right while you type commands directly into the Editor window when you drag the line from the Mini Builder to the Editior window.

Exporting from the Builder. If you prefer to continue developing some or all of your work entirely in the Builder, you still have the option to export the final result to a script file.

Everyone's style is a little different, and everyone learns in different ways. You can experiment with these methods to find out which works best for you. For the rest of this chapter, we will emphasize the most fundamental method: typing

commands into the editor window. We will, however, use the script templates where they can save us time and trouble.

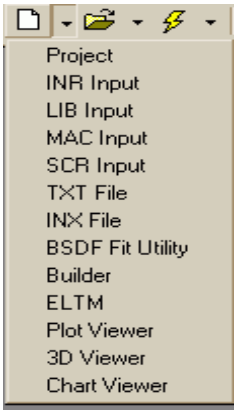
Getting Started in the Editor

The first step in producing a command script is opening an editor window. In ASAP, Editor windows initially open with the name, **INR Input**. There are a variety of ways to open this window. Choose whichever works best for you.

Table 22.1 Methods for opening a new ASAP Editor window

ASAP Toolbar

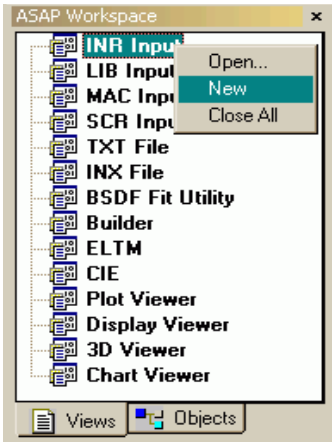
Click the paper button on the left part of the **New** button.



ASAP Toolbar

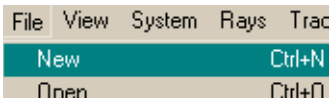
Click the down arrow on the right of the **New** button to open the drop-down menu.

Select **INR Input**.



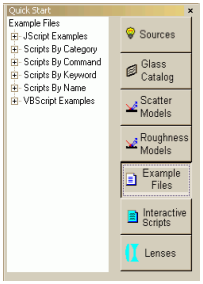
ASAP Workspace window

Right-click **INR Input** from the tree and select **New**.



ASAP **File** menu

File> New



Quick Start toolbar

Select a script on the **Example Scripts** tab.

Note: Historically, ASAP command scripts have used the file extension, *.inr, which is an abbreviation for **in**put **r**abet. RABET, in turn, is an abbreviation for the module in ASAP named, **R**ay **A**nd **BE**am **T**race. RABET was the program's name prior to 1992, and continues to be the name of the batch version of the code. The executables, rabet.exe and rabetp.exe (**P**ro) still ship with ASAP, allowing you to run the program from the command prompt, or call it from Mathcad™ or similar codes that can execute external batch programs. These files are in **asapxxxx\bin**. There is an explanation of their use in the on-line Help topic, "Batch mode" (which you can find on the **Index** tab).

At this point, it is always good practice to save the file with a new name. This can be done from the toolbar in the Editor window. The **Save As** and the other buttons are identified in Table 22.2.

Table 22.2 ASAP Editor Toolbar

	Save		Undo
	Save As		Redo
	Run		Find
	Save and Run		Repeat Find
	Run from Cursor		Replace
	Run to Cursor		Import Template
	Step		Command tips
	Cut		Mini Builder
	Copy		Scripting languages
	Paste		Tolerance Editor
	Erase		Perturb system
	Print		Update Tolerance Editor
	Undo		
	Redo		

The ASAP Editor, like some of the other ASAP windows, contains many optional features. Some are highly functional, while others control only the appearance of the text in the window. Like the Display Viewer, 3D Viewer, and Plot Viewer, you can start simply, and learn the rest by trial and experiment. You will find that most of the functional aspects of the Editor closely adhere to Windows® standards, so you can navigate with some familiarity.

The last section of this chapter gives a full introduction to the ASAP Editor, but for now, it is sufficient to know that behind it all, it is fundamentally a plain text editor. In fact, it is possible to write ASAP script files with any other plain text editor, like Microsoft® Notepad. However, the ASAP Editor does contain many useful features that make it particularly well suited for developing and running ASAP scripts.

- **Color syntax highlighting** of a type common to many modern software-development environments. This will make your file easy to read and navigate, and even show you when you have misspelled a command name. We will have more to say about this feature shortly.
- **Drag-and-drop editing** within the window or even between documents, and a feature that allows selecting and editing text in columns. The latter feature is useful for commands that work with tables of numbers.
- **Step through scripts** with the ASAP Editor, as well as **run selected parts** of the code based on cursor position and using elements of the Editor toolbar.
- **Access to the script templates**, the **Command Tips**, and the **Mini Builder** mentioned in the previous section.
- **Tolerancing** in the Editor with a two-step process: First, you create a tolerance data file, and then perform the Monte Carlo analysis.

Because of these advantages, we recommend that you give the ASAP Editor a try before you run to the comfort and security of your favorite text editor. The balance of this chapter and the next presumes that all script development and debugging is being done using the ASAP Editor.

Basic Script Template

We begin our introduction to the scripting language with the aid of a template that sets up the basic elements of the script file for us. Click the **Import Template** button on the right of the ASAP Editor toolbar. The dialog box shown in Figure 22.1 lists the available templates.

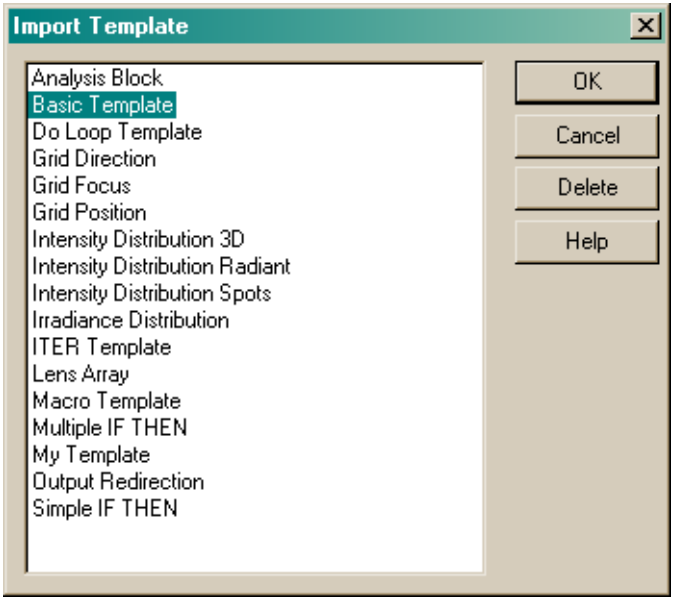


Figure 22.1 Dialog box for **Import Template**

Select **Basic Template,** and click **OK**. The result should appear as shown in Figure 22.2.

```
SYSTEM NEW
RESET

!! Define system units and wavelengths

UNITS MILLIMETERS 'Watts'
WAVELENGTH 550 NM

!! Define coatings

COATING PROPERTIES
  0 1 'TRANSMIT'
  1 0 'REFLECT'
  0 0 'ABSORB'

!! Define Media

MEDIA
  1.5 'GLASS'

!! GEOMETRY DEFINITIONS BEGIN HERE


!! SOURCE DEFINITIONS BEGIN HERE


!! TRACE RAYS WITH OPTIONAL PLOT

!!WINDOW Y Z
!!PLOT FACETS 5 5 OVERLAY
TRACE !!PLOT

!! ANALYSIS BLOCK BEGINS HERE
```

Figure 22.2 Using the Basic Template, you can quickly create an ASAP script file with a few common commands already in the appropriate places (shown in blue), and comments (shown in green, beginning with !!). The comment marks (!!) are the starting points for adding commands of your own. Note that some valid commands and command options (like **WINDOW** and **PLOT**) have been commented out.

Here are a few key features to notice from the basic template.

- Comments begin with "!!" and are colored green. Any characters after the "!!" symbols are ignored by ASAP. Comments can be placed at the beginning of a line or after commands. Use comments liberally to help you and others understand your thinking at the time you wrote the script.
- Keywords (ASAP commands and command options) appear in blue. If you type a command and it does not turn blue immediately, you have probably misspelled it or remembered the command incorrectly.

Note: The Editor uses a simple method for determining color syntax highlighting: it looks up all character strings in a table of keywords as you type. If the word is there, it becomes blue.

Because there is no further context checking, the Editor sometimes incorrectly selects something from the table. For example, one of the coatings defined in Figure 22.2 is named **ABSORB**. The Editor recognized the name because **ABSORB** is also an allowed option on the **MEDIA ABSORB** command, used as an alternative method for defining an absorbing medium. That is not its function in this file; **ABSORB** is just a name we chose to assign to a coating definition. This mistaken identification does not cause any problems when the command is passed to the ASAP kernel, because the kernel is sensitive to context.

- Most other strings and numerical arguments are black.
- All commands and options are in UPPER CASE. Although the case can be changed (see **$CASE** in on-line Help), the default behavior of the ASAP kernel is to ignore all alphabetic characters that are not typed in upper case. Lower case characters are, in fact, treated the same way as white space.
- ASAP ignores extra white space (spaces, tabs, and blank lines). You can use this white space to make the script easier to read and navigate. The choice is a matter of style.

Many of the blue keywords in the script should look familiar. The architects of the ASAP Builder and graphical user interface have tried to conform to the same naming conventions, terminology and syntax used by the ASAP command language wherever possible. This consistency should help to make your transition to the command language easier.

Note: Your ASAP installation provides you with working examples of most ASAP commands, written in the form of command scripts.

The Example Scripts button on the **Quick Start toolbar** opens the **Example Scripts** page--a tree list of all the available scripts of examples for ASAP commands. The list is divided into Categories, Commands, JScript, Keywords, and VBScript. The examples in the Commands group typically involve one command; whereas, the examples in the Categories group involve several commands. JScript and VBscript examples are also included.

System Preliminaries

The first two command lines in the basic template are **SYSTEM NEW** and **RESET**. Their presence at the top of your files is not a requirement, but their habitual use can prevent trouble over time. Together, these two commands perform most of the important functions of the **END** button, and thereby save you if you forget to click **END** before running the file.

Reinitialize the system databases

The first command, **SYSTEM NEW**, reinitializes the system databases, effectively throwing away any existing definitions of coatings, media, scatter models, objects, and entities that were part of the ASAP knowledge base when the kernel executed the command. This step prevents you from accidentally creating multiple copies of the same object.

Dispose of all rays in existence

The second command, **RESET**, disposes of all rays in existence, and resets several other program parameters to their startup state. By placing these commands at the start of your script, you will avoid inadvertently defining the same items multiple times.

> ***Note:*** Some ASAP engineers like to use multiple script files. They might use one to define their geometry, one for their sources, and another to trace rays and perform analysis. If you adopt this style, be sure that you place **SYSTEM NEW** only at the top of the geometry file, and use **RESET** only before defining rays.

Declare system units, wavelength, and properties

We must begin by declaring system units, wavelength (or wavelengths) and some properties of the optical materials that we will use. This is the same procedure that we used in the Builder in Chapter 3, performed in the same order (without the drop-down menu).

Units are set with the **UNITS** command. You can choose from the same set of units available in the Builder by typing either the short or long form of your choice directly into the definition.

Table 22.3 Unit settings

IN	INCHES
MM	MILLIMETERS
YD	YARDS
FT	FEET
MI	MILES
M	METERS
KM	KILOMETERS
MIL	MILS
UM	MICRONS
CM	CENTIMETERS
UIN	MICROINCHES

Units cannot be mixed. You must stay with a unit convention for your entire project.

Units cannot be mixed. You must select a unit convention, and stay with it for your entire project.

If you want to define flux units too, you can use the same command:

```
UNITS MILLIMETERS 'Watts'
```

ASAP is now set to use the string “Watts” in place of the default word “flux” in all of your analysis results.

To set the working wavelength, use the **WAVELENGTH** command. Again, this works as it did in the Builder. You have the usual choices, including a short and long form of each.

Table 22.4 Wavelength settings

A	ANGSTROMS
NM	NANOMETERS
UM	MICRONS
MM	MILLIMETERS
CM	CENTIMETERS
M	METERS

Define a medium

Next, the basic template defines a medium:

```
MEDIA
   1.5 'GLASS'
```

This definition is just a placeholder for any actual media you use in your model, but it does show the basic format of the command. It will work fine for our singlet example, however. Note that unlike in the Builder, **MEDIA** has become a two-line command. We have named the medium **'GLASS'**, which must be in single quotes. You can reference the medium later either by this name or by number. There is already a **MEDIA 0**, predefined by ASAP as **AIR/VACUUM** (with an index of exactly 1), so **'GLASS'** becomes medium number 1.

When you add media, insert the commands on subsequent lines, up to 100 media (depending on your version of ASAP):

```
MEDIA
   1.5  'GLASS'
   1.49 'ACRYLIC'
```

ASAP includes built-in glass catalogs for common commercial glasses available from Schott, Hoya, Ohara, and French Corning.

Remember that ASAP includes built-in glass catalogs for common commercial glasses available from French, Hikara, Hoya, Ohara, Schott, and Sumita. You do not have to define these materials in a **MEDIA** command, since ASAP already knows about them. We will demonstrate the use of catalog media in "Object Modifiers" on page 508.

Specify the coating

The last step before defining geometry in the basic ASAP script is to specify the coating, using the **COATING PROPERTIES** command. The template specifies the three most common coatings used to define systems in which Fresnel reflections can be neglected:

```
COATING PROPERTIES
    1 0 'REFLECT'
    0 1 'TRANSMIT'
    0 0 'ABSORB'
```

Again, the conventions are the same as those used in the Builder: the first number specified is the reflection coefficient, and the second is transmission. As with the **MEDIA** command, additional coatings appear on subsequent lines under the single **COATING PROPERTIES** command.

Defining Objects

Once we have established our working units and defined some media and coatings, we can begin to define our geometry. The optical system in our example is only a singlet lens and a detector plane. The definitions to accomplish it appear in Figure 22.3.

```
!! GEOMETRY DEFINITIONS BEGIN HERE

!! Front of lens

SURFACE
  OPTICAL Z 0 5 0 ELLIPSE 2.5
OBJECT 'LENS.FRONT'
  INTERFACE COATING TRANSMIT AIR GLASS

!! Back of Lens

SURFACE
  PLANE Z 1 ELLIPSE 2.5
OBJECT 'LENS.BACK'
  INTERFACE 0 1 AIR GLASS

!! Edge of Lens

SURFACE
  TUBE Z 0 2.5 2.5 1.0 2.5 2.5
OBJECT 'LENS.EDGE'
  INTERFACE 0 0 AIR GLASS
  BOUNDS +.3

!! Detector

SURFACE
  PLANE Z 9.8 ELLIPSE 1
OBJECT 'DETECTOR'
  INTERFACE 0 0 AIR AIR
```

Figure 22.3 Sample script for Step 1, Define the System. System definitions also include the preliminaries, like units, wavelengths, media and coatings. For this example, most of these definitions were provided by the basic template (Figure 22.2).

Just as in the Builder, three entity types are available to you for defining geometry: surfaces, edges, and lenses. Now, however, these are not just subclasses in the Builder menu, but actual ASAP commands. The front surface of the singlet lens shows the typical syntax for defining a surface-based object:

```
SURFACE
  OPTICAL Z 0 5 0 ELLIPSE 2.5
OBJECT 'LENS.FRONT'
  INTERFACE COATING TRANSMIT AIR GLASS
```

The **OPTICAL** command looks similar to the way it appeared in the Builder, but many of the arguments are missing. The ASAP scripting language has many useful default settings that the Builder cannot always use to advantage. While the Builder forces you to skip over many cells representing optional aspheric terms, they are simply left out of the command script version. The on-line Help now becomes a critical reference, much more so than with the Builder, which guides you through the details of the command syntax. The next section provides a quick tour of the format of on-line Help.

This example also brings out another important difference between creating geometry in the Builder and command scripts. In the Builder, all geometry created becomes an “object” by default. Recall that “objects” interact with rays, and “entities” do not. In the command language, the convention is reversed: all definitions are entities by default, and an additional step is required to turn them into objects. This step is performed with the **OBJECT** command, where we also give it a name:

```
OBJECT 'FRONT'
```

What if we need to extrude two edges together, as we did in Chapter 13 when we needed a square surface with a round hole cut into it? You may recall that we encountered the **OBJECT** command there. Here is the equivalent definition of that aperture stop, this time defined using the command language:

```
EDGE
  RECTANGLE Z 0 1 1 64
EDGE
  ELLIPSE Z 0 .5 .5 64
OBJECT
  .1 .2 'APERTURE'
```

First, the two entities are defined. Then the **OBJECT** command does the extruding. Under these circumstances, **OBJECT** becomes a two-line command. The entity numbers and object name move to the second line. We used the relative referencing method introduced in Chapter 4, as indicated by the “dot” in front of the entity numbers.

Using On-line Help

As indicated above, you are likely to use on-line Help more with command scripts than you will with the Builder and menus. There are many ways to enter the help pages. Perhaps the most efficient in the current context is by double-clicking to select a command keyword that you have typed in your editor window, and pressing the **F1** key. The top section of the help page for the **OPTICAL** command appears in Figure 22.4. You may find it useful to keep the Help page visible while you type in the Editor. You can do this by right-clicking the Help window, and selecting **Keep Help on Top> On Top**.

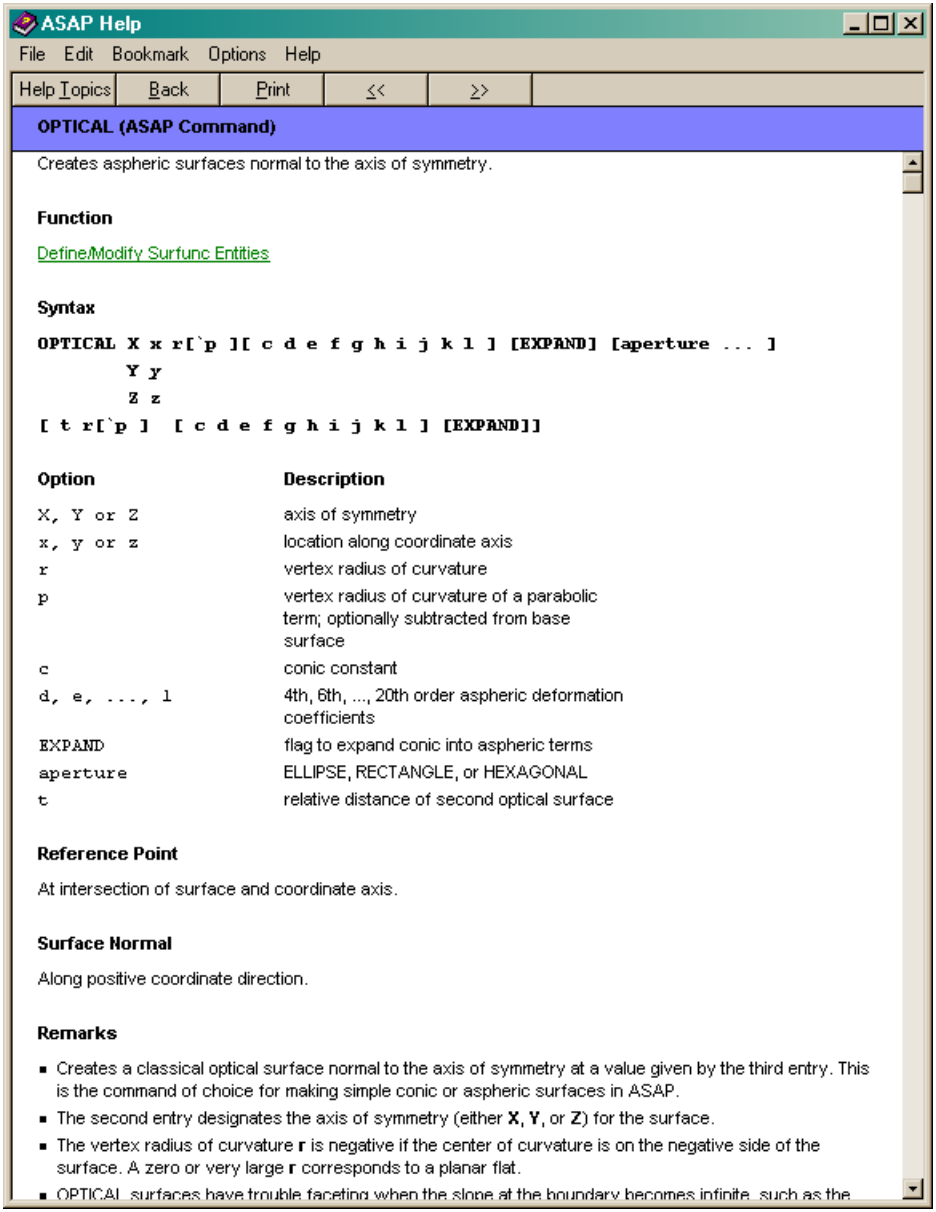


Figure 22.4 Top section of the on-line Help topic for the **OPTICAL** command. Key elements of all command entries are: **Syntax**, **Option** descriptions, and **Remarks**.

The **OPTICAL** command gives us a good opportunity to describe the conventions used in the ASAP on-line Help. The basic syntax for **OPTICAL** looks like this:

```
OPTICAL X x r[‘p][ c d e f g h i j k l ] [EXPAND] [aperture ... ]
        Y y
        Z z
[ t r[‘p ]  [ c d e f g h i j k l ] [EXPAND]]
```

This syntax is, admittedly, a little cryptic at first glance, but there is a simple underlying system. The same conventions are applied consistently throughout on-line Help and the Command Tips, so this is worth a little study. Here are some rules:

1. The first line always shows one complete version of the command, though it may not be the only version. Elements of the actual command appear in an uppercase typeface. Parameters generally are in a lowercase typeface. Here, for example, `OPTICAL X` is part of the command, but the characters **x** and **r** are parameters that you need to replace with numerical values (or variable names, or expressions, as we will see in Chapter 24).
2. Additional characters on subsequent lines, and appearing directly under characters on the first line, represent alternatives to the elements directly above them. In the **OPTICAL** command that was used in our singlet lens, we used the third option: `OPTICAL Z 0...` rather than `OPTICAL X 0...`.
3. Parameters or keywords appearing in square brackets, like **[‘p]**, are optional parameters. If you leave them out, ASAP uses a default value that is explained on the help page.

Note: The **p** parameter is, unfortunately, a rather exceptional case. This parameter is a seldom-used parabolic radius of curvature that can be subtracted from the base surface. The lack of space between the **r** parameter and the **p** parameter is important here. If you ever use the parameter, you will write it as extension of **r** in the form **r‘p** (for example, `-0.5‘0.1`) with no spaces between characters. This new option was “squeezed into” the **OPTICAL** command in recent years, using this unusual syntax to maintain backward compatibility with previous versions.

4. When several parameters appear within square brackets, like `[ c d e ...]`, they are all optional, and can be neglected partially or totally from right to left. In other words, if you need to specify the **e** parameter, you must also specify **c** and **d**. Sometimes, when fewer optional parameters are involved, ASAP shows nested brackets to make this more clear (see the table below).
5. The fourth line of the **OPTICAL** syntax is used to specify adding an optional second surface. With the first line and the second optional line, we can specify the front and back surface of the singlet as a single object.

You are not expected to memorize the meaning of all the lower-case variables. They are always provided in brief form in the help topic for a command.

Table 22.5 Lower-case variables in ASAP

Option	Description
`X, Y or Z`	Axis of symmetry
`x, y or z`	Location along the coordinate axis
`r`	Vertex radius of curvature
`p`	Vertex radius of curvature of a parabolic term; optionally subtracted from the base surface
`c`	Conic constant
`d, e, ..., l`	4th, 6th, ..., 20th order aspheric deformation coefficients
`EXPAND`	Flag for expanding conic into aspheric terms
`aperture`	`ELLIPSE, RECTANGLE, or HEXAGONAL`
`t`	Relative distance of the second optical surface

These short explanations are often enough if you have used the command before. If not, you can read on to the next section of the help topic for a more thorough explanation and a discussion. A typical example is the explanation provided for the **`r`** parameter in **`OPTICAL`**:

> The vertex radius of curvature **`r`** is negative if the center of curvature is on the negative side of the surface. A zero or very large **`r`** corresponds to a planar flat.

One place where you will likely need the additional explanation shown below is the [`aperture ...`] option. This is a shorthand placeholder for an aperture specification common to most surface-based definitions. It has its own syntax, as noted further down in the help topic:

```
ELLIPSE   a [ a' [ o [ s [ s' ] ] ] ]
RECTANGLE
HEXAGONAL a [ o [ s [ s' ] ] ]
```

Now, the full meaning of our **`OPTICAL`** command in the singlet example can be understood. Recall the specification:

```
OPTICAL Z 0 5 0 ELLIPSE 2.5
```

This specification is a surface at z = 0, radius of curvature +5 (with no parabolic term), conic constant 0 (a sphere), with a circular aperture of half-diameter 2.5. There are no aspheric deformations, holes, nor offsets. As a result of the ability of ASAP to assume reasonable defaults, 15 optional parameters were omitted. Sometimes, your task actually gets simpler when using the scripting language.

Object Modifiers

In ASAP command scripts, the object modifiers appear immediately below the **OBJECT** command. The only modifier on the first two elements of the singlet lens is **INTERFACE**.

Recall that the **INTERFACE** command tells a ray what to do, if it intersects with that object. Every object should have one. If you forget to add this line, the object becomes absorbing by default. For **LENS.FRONT** the **TRANSMIT** coating was already defined above using **COATING PROPERTIES**. We then declared an **AIR-GLASS** media change at this interface. As with the Builder, the order of **AIR** and **GLASS** is not important (see “Interface Command” on page 72 of Chapter 4).

You can use glass from the media catalogs by prefacing the name of the material with the catalog name and an underscore “_” character. For example, to use BK7 glass from the Schott catalog, you would use the following definition:

```
INTERFACE COATING TRANSMIT AIR SCHOTT_BK7
```

Notice that the **INTERFACE** command used to define **LENS.BACK** in Figure 22.3 is a little different from that used for **LENS.FRONT**. The `COATING TRANSMIT` words are missing, and two numbers have been substituted in their place. This is an alternative syntax for **INTERFACE**. With **TRANSMIT** defined as we have, these two versions are equivalent:

```
INTERFACE COATING TRANSMIT AIR GLASS
INTERFACE 0 1 AIR GLASS
```

The first number is the reflection coefficient, and the second is transmission. Many ASAP users prefer this syntax because it allows them to understand the interface at a glace, without references to definitions at the top of the file. Choose whichever syntax you prefer.

The tube used to form the edge of our singlet lens also requires a **BOUNDS** modifier to trim away the parts of the tube that we do not need:

```
SURFACE
  TUBE Z 0 2.5 2.5 1.0 2.5 2.5
OBJECT 'LENS.EDGE'
  INTERFACE 0 0 AIR GLASS
  BOUNDS +.3
```

The conventions used here are exactly the same as those used in the Builder. The plus sign (which is actually optional) indicates that we want to keep that part of the tube that is on the positive side of the third previously defined entity. Once again, we have elected to use relative referencing: **.1** is the most recently defined entity (the **TUBE**), **.2** is the **PLANE**, and **.3** is the **OPTICAL**. Unlike the first element of the Cooke Triplet constructed in Chapter 4, it was not necessary to trim the tube with the back surface, since this lens has a planar back. We already

know exactly where the tube should end, so we specified this location in the **TUBE** definition.

> ***Note:*** The ASAP Editor has colored the plus sign red in the BOUNDS command. This is the color syntax highlighting used by the ASAP Editor for arithmetic operators. The sign is not actually used in the role of an algebraic operator here, but the Editor is not smart enough to recognize that.

The other object modifiers you have already learned have the names you would expect, and parameterization that you likely could have guessed:

```
SHIFT x y z
SHIFT X x
      Y y
      Z z
SHIFT d ALONG a b c
ROTATE X d [ y z ]
       Y d [ z x ]
       Z d [ x y ]
ROTATE d ABOUT a b c [ x y z ]
REDEFINE COLOR k
FACETS n [ n' ]
```

Creating Sources

With the system model defined, we can now proceed to step 2, creating some rays. We want to write ASAP definitions that produce a circular grid of rays just large enough to fill the 2.5-millimeter half diameter of the lens, pointing in the +z direction, with a total flux of 100 watts.

The three commands necessary to accomplish are shown in Figure 22.5 along with the equivalent Builder lines shown for comparison.

*	Type	Option	Axis	Position	X Min	X Max	Y Min	Y Max	Number of X Rays	Number of Y Rays
EMT	Grid	Elliptic	Z	-2	-2.5	2.5	-2.5	2.5	101	101
MOD	Source	Direction		0	0	1				
CMD	Flux	Total	TOTAL	100						

```
!! SOURCE DEFINITIONS BEGIN HERE

GRID ELLIPTIC Z -2 -2.5 2.5 -2.5 2.5 101 101
  SOURCE DIRECTION 0 0 1
FLUX TOTAL 100
```

Figure 22.5 Step 2: Define rays. A comparison of the definition of a grid source in both the Builder (top) and the command language (bottom) illustrates the similarities.

Note: There is also a script template to help you define grid sources. If you click the **Import Template** button in the Editor window, and select **Grid Direction**, the result will look like this:

```
GRID RECT Z -10   -1 1 -1 1    11 11
  SOURCE DIRECTION  0 SIN[0] COS[0]
```

In this case, ASAP provides us with working examples of the two commands necessary to produce a basic grid, although we would need to change several of the parameters to suit our immediate needs. The **FLUX TOTAL** command would also have to be added. Still, you may find even the simple templates like this one useful when you are new to ASAP and its scripts.

Tracing Rays

The third step in every project is tracing rays. The basic template has already provided us with the one command needed to trace rays through our system: TRACE. The complete default entry looks like this:

```
!! TRACE RAYS WITH OPTIONAL PLOT
!!WINDOW Y Z
!!PLOT FACETS 5 5 OVERLAY
TRACE !!PLOT
```

Commands to optionally include a plot of the geometry (suitable for viewing in the 3D Viewer) have also been included, but they have been commented out. For this example, we will just “uncomment” these commands and add another new one (**$VIEW**) to actually open the 3D Viewer and display the result. Our trace section of the script appears in Figure 22.6.

```
!! TRACE RAYS WITH OPTIONAL PLOT

WINDOW Y Z
PLOT FACETS 5 5 OVERLAY
TRACE PLOT
$VIEW
```

Figure 22.6 Step 3: Trace the rays

The first command in Figure 22.6 is `WINDOW`. Up to this point we have not needed to include this command explicitly because it was always part of a command stream built up by menu dialogs like **System> Plot Facets** and **Trace> Trace Rays**. The options for using `WINDOW` look like this:

```
WINDOW X [ a a']  Y [ d d']
       Y          Z
       Z          X
```

Remember that the first axis specified is vertical, and the second horizontal in the plots. If you do not enter `a a'` and `d d'` parameters, ASAP will auto-scale your plot. Include them if you want to take control of the window dimensions.

The next line, `PLOT FACETS 5 5 OVERLAY`, draws a plot of the geometry into the Plot Viewer, and also prepares a three-dimensional "vector" version of the same data that can be viewed later in the 3D Viewer. The `OVERLAY` keyword keeps the Plot Viewer window open for more graphical data, which will be forthcoming when ASAP runs `TRACE PLOT`.

This set of commands may seem more complicated than it used to be when you traced rays using the **Trace** menu, but as Figure 22.7 shows, these are exactly the same commands that the dialog box would have sent to the kernel when you clicked **OK**.

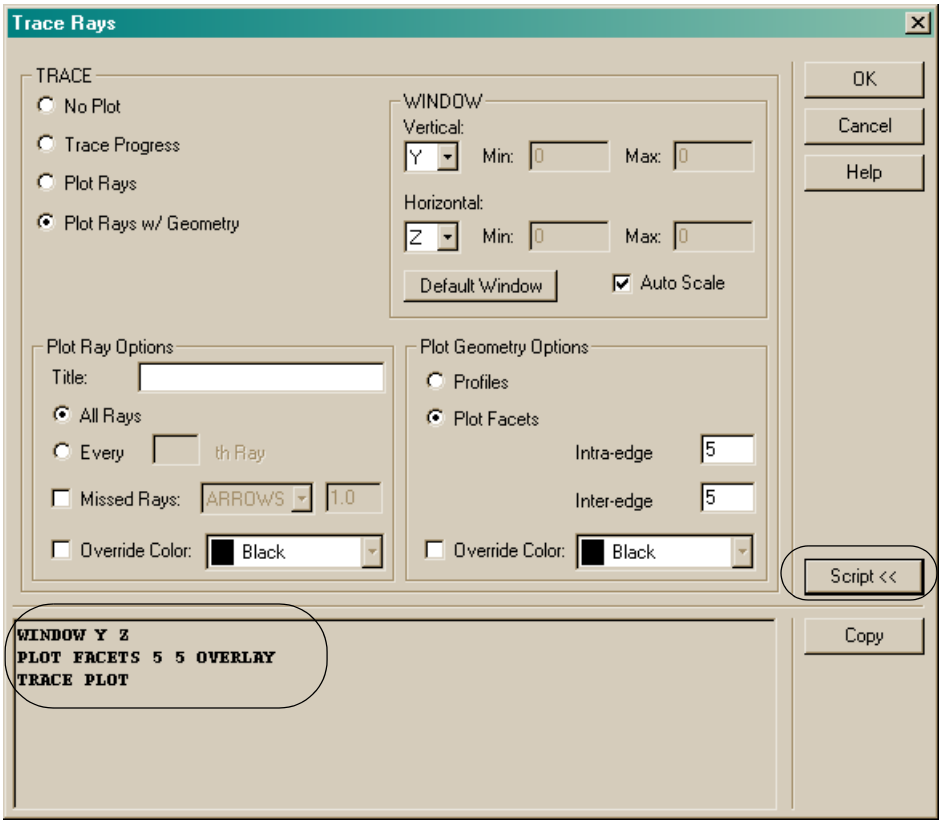


Figure 22.7 Dialog boxes in the ASAP user interface assist you when you cannot remember all the necessary steps to accomplish a common modeling process. The **Trace Rays** dialog box is an example. It generates multiple commands in the right order for the ASAP kernel to process.

Remember, you can always get help forming script entries by clicking the **Script** button, copying the commands from that window, and pasting them into your Editor window.

The last line in Figure 22.6 is a new command, **`$VIEW`**. This command, like any ASAP command you see that begins with **`$`** or **`&`**, is part of the ASAP macro language. Macros are not fundamentally different from other ASAP commands in the way they are used or in the way parameters are passed. The only significant difference is that they have more to do with programming (in the context of command scripts) and little to do with optics. This particular macro allows you to open the 3D Viewer to see the current contents of the vector file. Its function is identical to clicking the 3D Viewer button on the ASAP toolbar. In fact, if you click that button, you will see that---`$VIEW` is echoed in the Command Output window.

If you prefer to see your ray trace overlaid on a profile of your geometry, the following lines will do that without opening the 3D Viewer:

```
WINDOW Y Z
PROFILES OVERLAY
TRACE PLOT
```

Performing Analysis

The ASAP Editor also provides a template for setting up and performing analysis within a command script. If you place your cursor in the position where you want to insert commands, click the **Import Template** button, and select **Analysis Block** from the list, you will insert the following lines:

```
CONSIDER ONLY <<object name or number>>
WINDOW Y X
PIXELS 39
SPOTS POSITION
DISPLAY
  ISOMETRIC 'Distribution of flux'
RETURN
```

This template establishes the basic steps for performing analysis: isolate rays (`CONSIDER`), pick a window and make a distribution file (`WINDOW, SPOTS POSITION`) and use display tools to look at the resulting distribution. You are prompted to add specific information of your own by the <<...>> symbols. In our case, we will replace this with `DETECTOR`. We have made that and just a few other modifications to the template to build the analysis block shown in Figure 22.8.

```
!! ANALYSIS BLOCK BEGINS HERE

CONSIDER ONLY DETECTOR
FOCUS MOVE
WINDOW Y X
PIXELS 101
SPOTS POSITION 'Spots at best focus'

DISPLAY
  AVERAGE
  ISOMETRIC 'Best Focus'
  PICTURE
RETURN
```

Figure 22.8 Step 4: Perform the analysis

The `DISPLAY` command was introduced and discussed extensively in Chapter 20. Its function is to load the current distribution file (**bro009.dat**) made by the `SPOTS`

command into memory, so that you can use the display tools to graph and process it. The user interface has been sending this command to the kernel while you were using the menus to do your analysis.

The **RETURN** command is also new, and should always be present below your display commands. Its function is to tell the ASAP kernel that you are finished processing the display data, and ready to return to other work (or the idle state). The display tools almost have the status of a program within a program. You enter this state with the **DISPLAY** command, and leave it with **RETURN**. If you forget to do this, and issue additional commands, ASAP responds with the warning,

```
*** Unrecognized DISPLAY subcommand!
```

It will usually process the new command anyway, however.

> ***Note:*** If you study a few scripts in the examples database, you may also notice use of the **RETURN** command in other parts of the scripts. There are actually many sub-programs or "levels" within the ASAP kernel. One such level is entered when the kernel encounters the **COATINGS** command. It stays at that level while reading subsequent lines describing additional coatings, and leaves automatically when the change in syntax indicates that you are finished entering coatings. If you entered these definitions manually in the Command Input window, you would notice that the small prompt just below the window changed from **ASAP>** to **COA>** during this process. In most cases, ASAP can find its own way "home" to the **ASAP>** prompt without your help.

Figure 22.9 shows the script we created in the ASAP Editor. (The script is split over two pages.)

```
SYSTEM NEW
RESET

!! Define system units and wavelengths

UNITS MILLIMETERS 'Watts'
WAVELENGTH 550 NM

!! Define coatings

COATING PROPERTIES
  0 1 'TRANSMIT'
  1 0 'REFLECT'
  0 0 'ABSORB'

!! Define Media

MEDIA
  1.5 'GLASS'

!! GEOMETRY DEFINITIONS BEGIN HERE

!! Front of Lens

SURFACE
  OPTICAL Z 0 5 0 ELLIPSE 2.5
OBJECT 'LENS.FRONT'
  INTERFACE COATING TRANSMIT AIR GLASS

!! Back of Lens

SURFACE
  PLANE Z 1 ELLIPSE 2.5
OBJECT 'LENS.BACK'
  INTERFACE 0 1 AIR GLASS
```

(continued)

```
!! Edge of Lens

SURFACE
  TUBE Z 0 2.5 2.5 1.0 2.5 2.5
OBJECT 'LENS.EDGE'
  INTERFACE 0 0 AIR GLASS
  BOUNDS +.3

!! DETECTOR

SURFACE
  PLANE Z 9.8 ELLIPSE 1
OBJECT 'DETECTOR'
  INTERFACE 0 0 AIR AIR

!! SOURCE DEFINITIONS BEGIN HERE

GRID ELLIPTIC Z -2 -2.5 2.5 -2.5 2.5 101 101
  SOURCE DIRECTION 0 0 1
FLUX TOTAL 100

!! TRACE RAYS WITH OPTIONAL PLOT

WINDOW Y Z
PLOT FACETS 5 5 OVERLAY
TRACE PLOT
$VIEW

!! ANALYSIS BLOCK BEGINS HERE

CONSIDER ONLY DETECTOR
FOCUS MOVE
WINDOW Y X
PIXELS 101
SPOTS POSITION 'Spots at best focus'


DISPLAY
  AVERAGE
  ISOMETRIC 'Best Focus'
  PICTURE
RETURN
```

Figure 22.9 Complete Singlet lens script, showing all four elements of a project in a single script file.

Figure 22.10 (top) shows a Builder file that creates exactly the same geometry as the script file we have been writing. If we had elected to develop our system model in the Builder instead of in the ASAP Editor, we could still make a script from the Builder file. To do this, first make sure the Builder has focus (click on the title bar or in any other part of the Builder window), and then select **File> Export> INR**. Instead of sending commands to the ASAP kernel, the Builder redirects the commands to a text file, and gives it the **.inr** extension. The result is shown at the bottom of Figure 22.10. You can now proceed to add commands, comments, macros or other script elements to the file by opening it

in the ASAP Editor. Unfortunately, you cannot go back again; this translation works only in one direction.

*	Type									
SYS	System	New								
	Reset									
SYS	Units	Millimeters	Watts							
SYS	Wavelengths		550					Nanometer		
CMD	Coating	TRANSMIT		Properties		0	1			
CMD	Coating	REFLECT		Properties		1	0			
CMD	Coating	ABSORB		Properties		0	0			
CMD	Media	GLASS	Media	1.5						
OBJ	Optical	LENS.FRONT	Axis	Z	0.0	5.0	0			
MOD	Interface	Coating	Coating	'TRANSMIT'	'Air'	'GLASS'				
OBJ	Plane	LENS.BACK	Axis	Z	1.0	Ellipse	2.5	2.5		
MOD	Interface	Coating	Coating	'TRANSMIT'	'Air'	'GLASS'				
OBJ	Tube	LENS.EDGE	Axis	Z	0.0	2.5	2.5	1.0	2.5	2.5
MOD	Interface	Coating	Coating	'ABSORB'	'Air'	'GLASS'				
MOD	Bounds	Edge	+.3							
OBJ	Plane	DETECTOR	Axis	Z	9.8	Ellipse	1	1		
MOD	Interface	Coating	Coating	'ABSORB'	'Air'	'Air'				

```
SYSTEM NEW
RESET
UNITS MILLIMETERS 'Watts'
WAVELENGTHS 550 NANOMETERS
COATING PROPERTIES; 0 1   'TRANSMIT'
COATING PROPERTIES; 1 0   'REFLECT'
COATING PROPERTIES; 0 0   'ABSORB'
MEDIA; 1.5   'GLASS'
ENT OBJECT;OPTICAL Z 0.0 5.0 0 ELLIPSE 2.5 2.5   'LENS.FRONT'
  INTERFACE COATING TRANSMIT AIR GLASS
ENT OBJECT;PLANE  Z 1.0 ELLIPSE 2.5 2.5 'LENS.BACK'
  INTERFACE COATING TRANSMIT AIR GLASS
ENT OBJECT;TUBE  Z 0.0 2.5 2.5 1.0 2.5 25 0.0 0.0   'LENS.EDGE'
  INTERFACE COATING ABSORB AIR GLASS
  BOUNDS +.3
ENT OBJECT;PLANE Z  9.8 ELLIPSE 1 1   'DETECTOR'
  INTERFACE COATING ABSORB AIR AIR
RETURN
```

Figure 22.10 Geometry created by this **Builder** file (top) is equivalent to the one defined by the script in Figure 22.9. It can also be exported to produce the script file (bottom).

A careful comparison of Figure 22.9 and Figure 22.10 shows some differences in many of the commands. For example, look at the coating definitions:

Our Script

```
COATING PROPERTIES
    1 0 'REFLECT'
    0 1 'TRANSMIT'
    0 0 'ABSORB'
```

Builder Output

```
COATING PROPERTIES; 1 0 'REFLECT'
COATING PROPERTIES; 0 1 'TRANSMIT'
COATING PROPERTIES; 0 0 'ABSORB'
```

The most important difference between our script and Builder output is the way in which objects are defined.

These differences are caused by the Builder's strong preference for avoiding multi-line commands. The Builder has inserted a semicolon (;) to merge two lines (more about this in Chapter 24, "Other Command Script Features"), and has unnecessarily repeated the base command (**COATING PROPERTIES**) on each line. Still, the effect when processed by the ASAP kernel is exactly the same.

The most important difference between the two sets of definitions is the way in which objects are defined. As we have already noted, the Builder assumes that you always want to create objects, while our command scripts need an explicit **OBJECT** command to turn mathematical entities into objects that can interact with rays. For more about how this approach is done in the Builder, see the sidebar, "Alternative Styles for Creating Geometry" on page 521.

Summary

In this chapter, the following new commands were introduced.

ASAP Commands	Menu	Description
SYSTEM NEW	Builder: **Setup> System**	Initializes the system database, removing all previous definitions.
RESET	Builder: **Ray Control> Reset**	Deletes all existing rays.
EDGE	—	Indicates that the following command or commands are edge entities.
SURFACE	—	Indicates that the following command or commands are surface entities.
LENS	—	Indicates that the following command or commands are lens entities.
ENTITIES OBJECTS	(Automatic)	All geometrical definitions that follow this command automatically become objects and do not need an explicit OBJECT command.
ENTITIES	Builder: (Right-click a definition and select **Entity Status**)	All geometrical definitions that follow this command have entity status.
DISPLAY	(Automatic)	Loads **bro009.dat** into memory for file operations, graphics, or processing with the display tools.
RETURN	(Automatic)	Returns the ASAP kernel to the top command level, the **ASAP>** prompt.

Command scripts are written in the native language of the ASAP kernel. All commands and macros are available when you use ASAP in this way.

ASAP command scripts offer an alternative to working exclusively with the Builder and the menus that make up the graphical user interface. The Builder and the menus do not give you access to every command, nor every command option. Command scripts, however, are written in the native language of the ASAP kernel. All commands and macros are available when you use ASAP in this way.

A comparison of Figure 22.9 and Figure 22.10 show somewhat different definitions of the **OPTICAL** *surface. Our script looks like this:*

```
SURFACE
  OPTICAL Z 0 5 0 ELLIPSE 2.5
OBJECT 'LENS.FRONT'
  INTERFACE COATING TRANSMIT AIR GLASS
```

The Builder produces this:

```
ENT OBJECT;OPTICAL Z 0.0 5.0 0 ELLIPSE 2.5 2.5  'LENS.FRONT'
  INTERFACE COATING TRANSMIT AIR GLASS
```

Both produce exactly the same object when submitted to the kernel for processing.

The most important difference between the two specifications is the unfamiliar command **ENT OBJECT**. *This is actually an abbreviated version of the command* **ENTITIES OBJECTS**. *(See the discussion of command abbreviation in "Abbreviations, Shortcuts, and Special Characters" on page 539 in Chapter 24.) Once executed, all subsequent definitions automatically become objects that do not need the* **OBJECT** *command.*

The Builder places **ENT OBJECT** *in front of every line. While this is harmless, it is not necessary. With that in mind, we could use this alternative style to write compact object definitions.*

The following lines define all three elements of the singlet lens:

```
ENTITIES OBJECTS
  OPTICAL Z 0.0 5.0 0 ELLIPSE 2.5  'LENS.FRONT'
    INTERFACE 0 1 AIR GLASS
  PLANE Z 1 ELLIPSE 2.5  'LENS.BACK'
    INTERFACE 0 1 AIR GLASS
  TUBE Z 0.0 2.5 2.5 1.0 2.5 2.5  'LENS.EDGE'
    INTERFACE 0 0 AIR GLASS
    BOUNDS +.3
```

We never need to use the **OBJECT** *nor the* **SURFACE** *commands with this syntax.*

If, in the midst of a series of object definitions, you need to define some entities (perhaps to extrude to edges or build bounding surfaces), you can change the mode ASAP is working in by using the **ENTITIES** *command without the* **OBJECTS** *option. You would define the aperture stop discussed earlier in this chapter as follows:*

```
ENTITIES
  RECTANGLE Z 0 1.0 1.0 64
  ELLIPSE   Z 0 0.5 0.5 64
OBJECT
  .1 .2 'Aperture'
```

Many ASAP users prefer this compact syntax. We chose to use the older style in this Primer to remain consistent with the ASAP Examples database, which is a large body of command examples written before use of **ENTITIES OBJECTS** *became fashionable.*

CHAPTER

23

RUNNING COMMAND SCRIPTS

In this chapter, you will learn how to run ASAP script files. As you might expect, this usually involves only clicking a button, so it is hardly worthy of an entire dedicated chapter. But things do not always go exactly as you planned, so running scripts naturally leads to a discussion of some useful techniques for correcting or "debugging" your scripts. We will also take a brief look at some additional ways to customize ASAP to suit your individual needs while producing and running script files. You will learn ASAP Editor features that will save you time and effort, a procedure to add your own script templates to the Editor, and a way to program the **Custom Toolbar** buttons to run one or more ASAP commands just by clicking.

Running Script Files

While the ASAP Editor may be looking up keywords and coloring them blue, no formal syntax or error checking is performed until you actually "run" the file.

As was the case with the Builder, the ASAP kernel is not "reading over your shoulder" as you type commands into the ASAP Editor window. While the ASAP Editor may be looking up keywords in a list and coloring them blue if it finds them, no formal syntax or error checking is performed until you actually "run" the file. You do this either by clicking the **Run**, or **Save and Run** buttons in the Editor window.

If you are not using the ASAP Editor, you will need to click the **Run** button on the main ASAP toolbar.

Now you can browse to your script file, select it, and run it. If you are running a file that was run recently, you can pick it from the drop-down list by clicking the arrow just to the right of the **Run** button.

Also, recall organizing a project. In Chapter 12, you learned how to create a project, adding files that you were actively using to a short list on the **Files** tab of ASAP Workspace. If you have done this, you can run files directly from that list

just by selecting them and pressing **Ctrl+Enter**, or right-clicking and selecting **Run** from the pop-up menu. The file does not need to be open in the Editor, though it can be. If you use this method, however, be sure to save your work in the Editor window to make sure the most recent version of the script is written to your working directory.

If you have entered the command script from the previous chapter, you should be able to run it now by one of these methods. If you made no typing errors, ASAP will proceed through all four stages: defining the system, defining rays, tracing rays, and performing the analysis, and leave you at the **ASAP>** prompt when it has finished. The results from the `ISOMETRIC` and `PICTURE` commands appear in Figure 23.1.

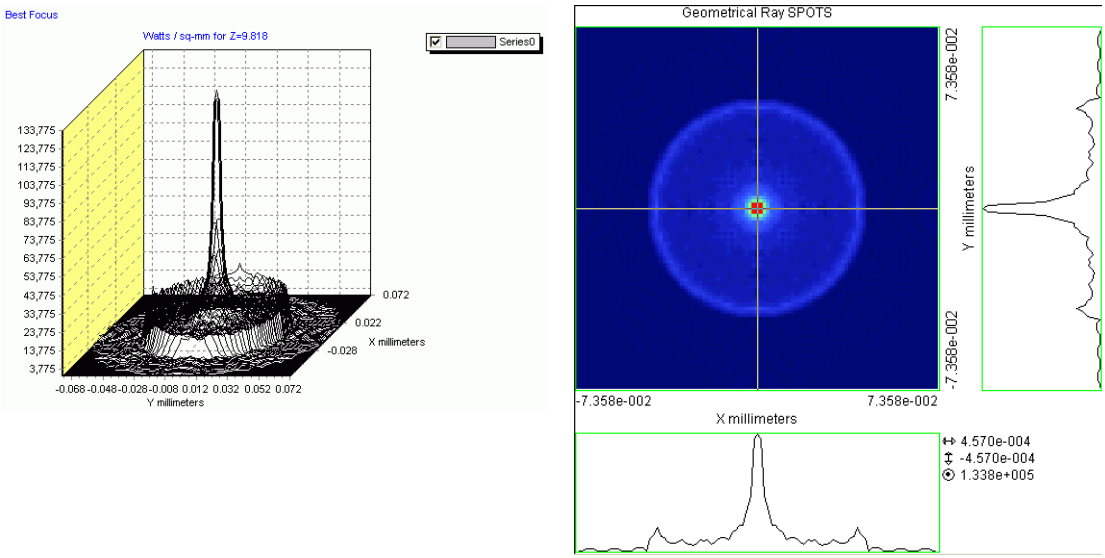


Figure 23.1 Results from the Analysis section of the **Singlet.inr** script. Left, isometric view in the Chart Viewer; right, the same distribution file viewed with the Display Viewer.

The beauty of using ASAP in the command-script mode is that once you get it right, the script itself completely documents every step you used in modeling, tracing, and analysis. It is not necessary to remember the exact sequence of mouse clicks that were necessary to produce the results.

Correcting Errors

But what if you did not get it right on the first try? If you typed carefully, you were perhaps fortunate enough to have entered all 77 lines of the singlet lens script without errors. It is more likely, however, that your first attempt to run the script was unsuccessful, at least in some detail. This result would be indicated

by an error message in the Command Output window of the type shown in Figure 23.2.

```
--- OBJECT 'LENS.FRONT'
---   INTERFACE COATING TRANSMIT AIR GLASS
--- !! Back of Lens
--- SURFACE
---   PLANE Z 1 ELLIPSE 2.5
--- OBJECT 'LENS.BACK'
---   INTERFACE 0 1 AIR GLASS
--- !! Edge of Lens
--- SURFACE
---   TUBE  0 2.5 2.5 1.0 2.5 2.5
***         ^ERROR
   Axis of TUBE must be a literal

--- TUBE X x y z x' y' z' [ q [  q' ] ]
         Y y z x y' z' x'       INNER
         Z z x y z' x' y'       OUTER
```

Figure 23.2 If an error occurs in your script, ASAP issues a red error message below the offending command. The parser tried to interpret the numerical value in that position as a literal letter (X, Y, or Z) and failed. When appropriate, ASAP types the correct syntax below to help you find your error.

As you can see, ASAP stops at the offending line and points to the problem. In situations like this one, you are also shown an example of the syntax that ASAP was expecting to help you with debugging.

A few words are in order about exactly what happens when you run an ASAP script. The ASAP command language is interpreted, not compiled. This means that ASAP never reads ahead to find out what might come later. Further, a "machine language" version of the script is never created, as it would be in most compiled languages. In ASAP scripts, the commands you typed are read one line (or "record") at a time, and error and syntax checking is performed at that time. If it passes, the record is sent to the kernel and run. When the kernel finishes executing a record, interpretation proceeds on the next line. If a record fails the syntax test, an error message is printed in the Command Output window, and no further lines are read. This brings us to an important point:

> When ASAP detects an error part way through a script, the computational kernel has already successfully run the lines above the error, and the state of the ASAP knowledge base has been changed accordingly.

In Figure 23.1, we inadvertently left out the axis designation (the **Z** character), so interpretation failed on the **TUBE** record. A quick look on the **Object** tab of the ASAP Workspace window confirms that all the objects defined above the tube now exist (Figure 23.3).

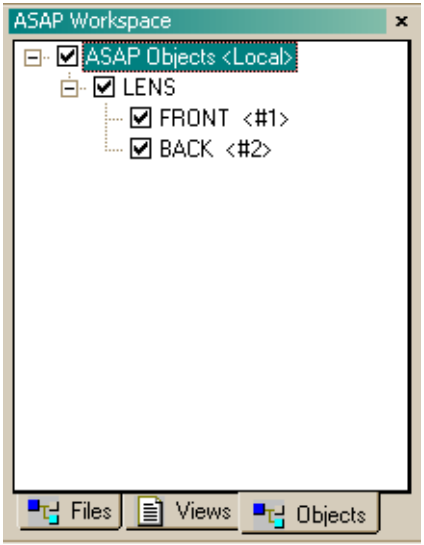


Figure 23.3 The **Objects** tab on **ASAP Workspace** confirms that the error on the line defining the lens edge (see Figure 23.2) did not alter the fact that the two objects were already successfully created.

There is no need to run those lines again. After making the correction in the Editor, leave the cursor anywhere on the line that previously contained the error, and click the **Run From Cursor** button.

ASAP proceeds to run the rest of your script to completion, or at least to the next error. Our file is simple enough that we would not have saved much time when compared with running the entire file again from the top. But if you have spent all night tracing rays, and then misspelled the `DISPLAY` command in the middle of your analysis block, you will appreciate this "resume" feature of interpreted languages.

Verifying As You Work

The preceding discussion implies that you should write an entire script file from beginning to end before checking to see if smaller pieces are working correctly. This approach is not a good idea. In the Builder, we can periodically click the **Preview** button to look at some or all of our system to make sure we had it right. Have we lost that ability when using command scripts? No, we simply need to inject a few extra verification commands at the appropriate places in the script to give us confidence that the work so far is correct.

Perhaps the closest thing to the **Preview** button on the Builder is the `VUFACETS` command. This command expects two integer arguments, and behaves somewhat like `PLOT FACETS`. Try placing `VUFACETS 5 5` at the end of the geometry

definitions in Singlet.inr. Then place the cursor on the next line *below* **VUFACETS** and click **Run To Cursor**.

ASAP will show all geometry created prior to the **VUFACETS** command in the 3D Viewer. Note that it does this without the usual intermediate step of making a two-dimensional **PLOT FACETS** version of the graphics in the Plot Viewer, much as the **Preview** button in the Builder does. There is one big difference, however: you have now run your file, and the state of ASAP has been altered. This was not the case with **Preview**.

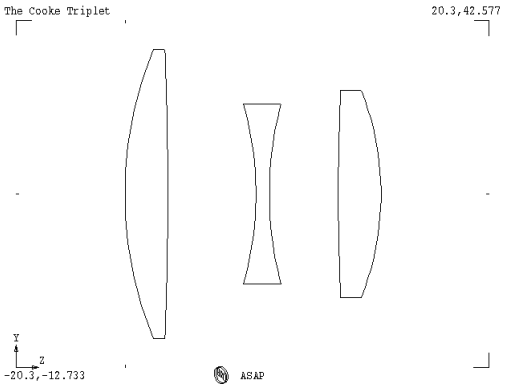
> ***Note:*** The **Preview** button on the Builder toolbar can give you a quick look at all or some of your geometry under development without altering the current state of ASAP. You can see how ASAP does this by studying the commands sent to the Command Output Window when the **Preview** button is clicked. ASAP saves a copy of the current state of ASAP, runs the selected Builder lines, makes a vector file (without displaying it in the Plot Viewer), launches the 3D Viewer to display the result, and then returns ASAP to its original state. Unfortunately, it is not possible to do this in quite so automatic a fashion using command scripts.
>
> However, with simple programming, you can capture the commands sent to the Command Output window when **Preview** is selected from the Builder. You can then place captured commands in a script file to create the same preview. Alternatively, you can assign the commands to one of the Custom Toolbar buttons, as described in “Programming the Custom Toolbar” on page 533.

You have learned other methods of geometry and source verification as well, using the menu options after running your Builder files. You can always click the **Script** button on the menu dialog boxes to see what commands are being used, and copy these into the appropriate places in your scripts. A few of the most common verification graphics are shown in Figure 23.4.

Once you have verified a piece of your geometry or your source, it is no longer necessary to include that verification step every time you run your script in the

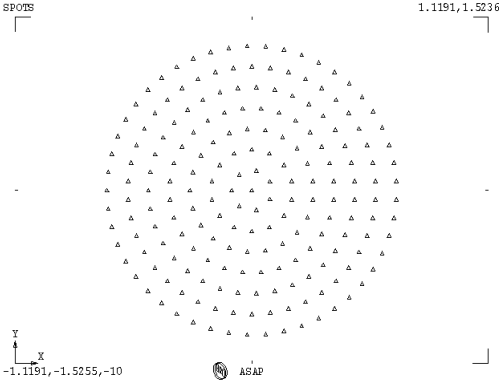
future. While you can just remove the verification commands, many ASAP users just comment them out after use, in case they need to use them again later.

System> Profiles



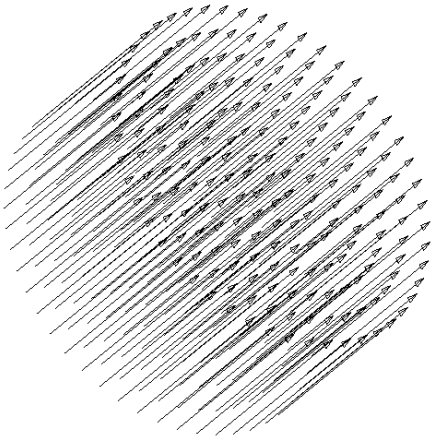

```
WINDOW Y Z
OBLIQUE OFF
PROFILES 0 0 -1 PIXELS 201
```

Rays> Graphics> Plot Positions 2D



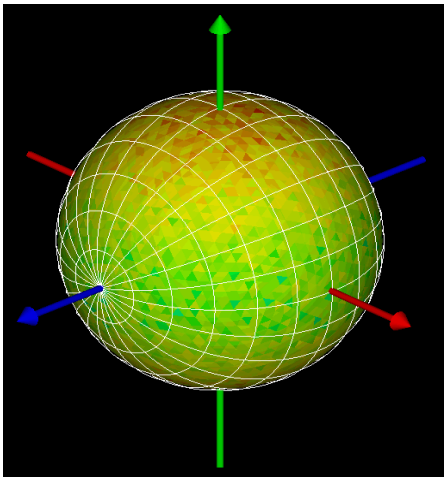

```
WINDOW Y X
SPOTS POSITION ATTRIBUTE 3
```

Rays> Graphics> Plot Rays 3D



```
$10 VECTOR REWIND
$IO PLOT CANCEL
$PLOT OFF
PLOT RAYS 1
$VIEW
$PLOT NORM
$I0 PLOT
```

Rays> Graphics> Radiant Sphere



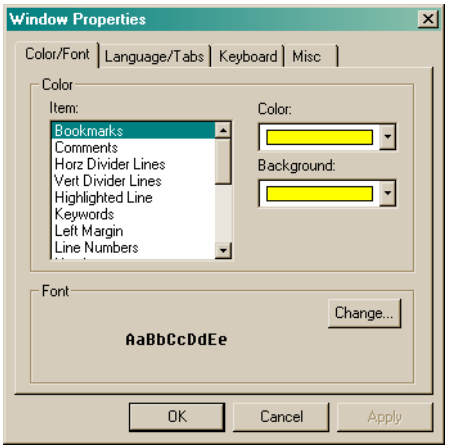
```
DUMP
$VIEW LASTDUMP.DIS
```

Figure 23.4 All the geometry and ray verification methods you have learned can be inserted into ASAP scripts. Most of the commands shown in this figure are familiar, except for some of those associated with **Plot Rays 3D**. The macros are being used to rewind (initialize) the vector file, suppress plotting to the **Plot Viewer** and, in the end, restore normal plotting.

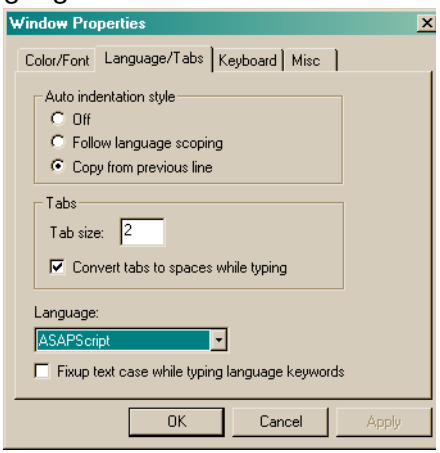
More Editor Features

You have already seen the basic functionality of the ASAP Editor and its toolbar. There are, however, many other features of this Editor that can be used to customize the view and the functionality to suit your style. You gain access to most of these options by right-clicking anywhere within the Editor window, and selecting **Properties** from the pop-up menu. This brings you to the **Window Properties** dialog box, which has four tabs. Each tab is shown in Figure 23.5.

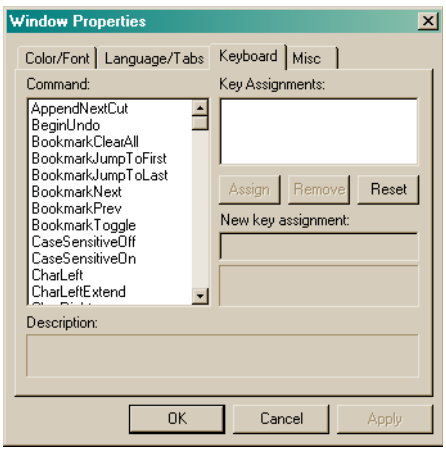
Color/Font tab



Language/Tabs tab



Keyboard tab



Misc tab

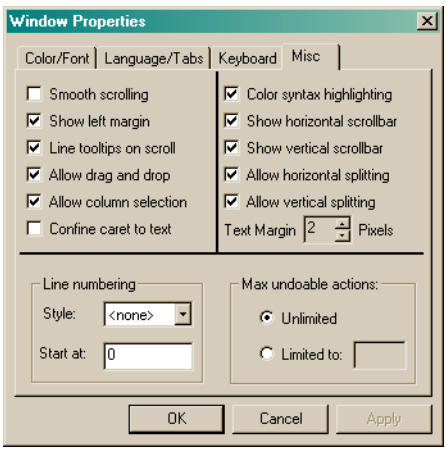


Figure 23.5 Four tabs in the **Properties** dialog box of the **Editor**, used to customize the behavior of the **Editor** window.

We will highlight a few of its features below.

Color/Font. Controls window colors, including **Color Syntax Highlighting**. You can also choose between several alternative fixed-width fonts if you do not like the default typeface. Note that not all syntax highlighting color options affect the coloring of your text.

> ***Note:*** The ASAP Editor is third-party software designed to support a wide variety of software applications. Not all its features have been implemented in ASAP, primarily in the area of color-syntax highlighting. Some of the highlighting options listed do not exist in the ASAP command language.

Language/Tabs. Controls indenting in your file, and even allows you to automatically convert all keywords to upper case when you select **Fixup text case**.

Keyboard. Lists the keyboard shortcuts that are assigned to common Editor functions, and allows you to edit or assign new keyboard shortcuts. You will benefit from exploring this tab on your own since it highlights many hidden features of the Editor. For example, you will learn how to place “bookmarks” in the colored margin on the left of your Editor window that allow you to mark key locations in your script, and jump quickly between them with a few keystrokes.

Misc. Enables and disables features within the Editor, including line numbering and color syntax highlighting.

The **Misc** tab also includes hints of features that are worth additional discussion. One of these features is **Column Selection**. This feature, perhaps one of the most useful “extras” provided by this Editor, is enabled by default. It allows you to select data along columns, which you can then cut, paste, drag and drop to get data correctly ordered.

To use **Column Selection**, hold down the **Ctrl** key *before* clicking to select text. Figure 23.6a and Figure 23.6b provide examples and some instruction for using it.

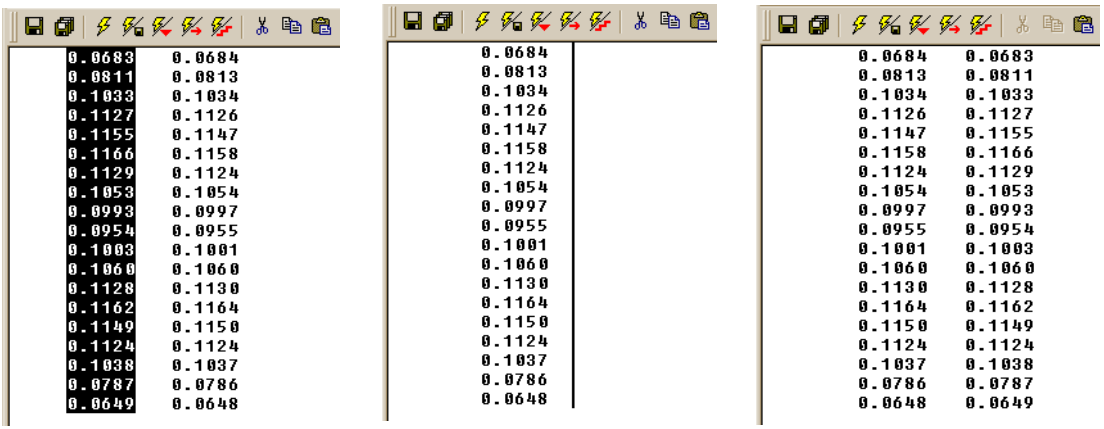


Figure 23.6a With the **Column Selection** feature in the ASAP **Editor**, you can manipulate columns of text. To switch two columns, hold down the **Ctrl** key and select a column (left). Once selected, this column can be **Cut** to remove it and store a copy on the Windows **Clipboard**. Hold down the **Ctrl** key, select and drag down the column to establish the area for the column you will insert (middle). **Paste** the column at the marked position (right).

Another column feature is selecting a series of lines to comment out.

```
!! Detector

SURFACE
  PLANE Z 9.8 ELLIPSE 1
OBJECT 'DETECTOR'
  INTERFACE 0 0 AIR AIR

!! SOURCE DEFINITIONS BEGIN HERE

GRID ELLIPTIC Z -2 -2.5 2.5 -2.5 2.5 101 101
  SOURCE DIRECTION 0 0 1
FLUX TOTAL 100
```

```
!! Detector

!!SURFACE
!!  PLANE Z 9.8 ELLIPSE 1
!!OBJECT 'DETECTOR'
!!  INTERFACE 0 0 AIR AIR
!!
!! SOURCE DEFINITIONS BEGIN HERE

GRID ELLIPTIC Z -2 -2.5 2.5 -2.5 2.5 101 101
  SOURCE DIRECTION 0 0 1
FLUX TOTAL 100
```

Figure 23.6b With the **Column Selection** feature, you can quickly comment out a series of lines in a file. Hold down the **Ctrl** key, and click and drag downward to select the lines for commenting (left). Type **!!** to simultaneously comment out all the selected lines. The comment characters can be removed just as quickly in a similar way.

Other useful features that are enabled by default include horizontal and vertical window splitting. These features can be used to divide the Editor window into two sections (either vertically or horizontally), which you can scroll

independently. Common uses of these features are shown in Figure 23.7a and Figure 23.7b.

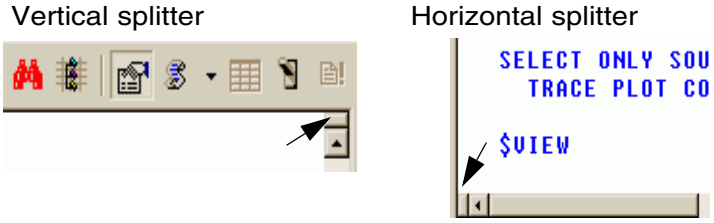


Figure 23.7a Use horizontal and vertical splitting to turn the **Editor** window into two independently scrollable windows. You can split a window up/down or sideways by selecting and dragging the vertical or horizontal splitter buttons in the corners of the window.

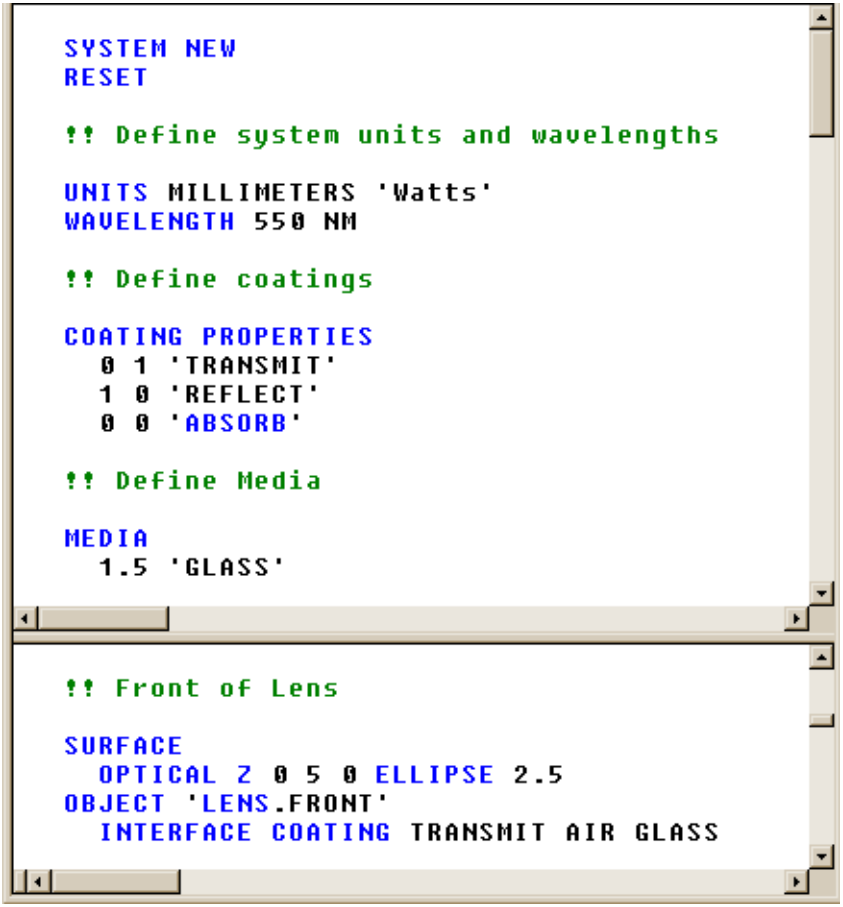
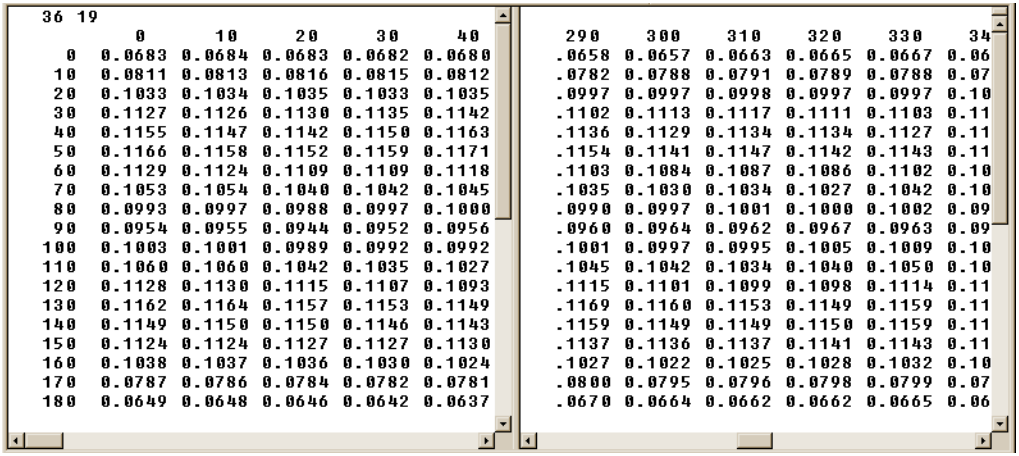


Figure 23.7b With vertical splitting of the **Editor** window, we can view system database definitions in a previous section of the script that was out of sight, while we continue entering commands further down.



36 19

	0	10	20	30	40	290	300	310	320	330	34
0	0.0683	0.0684	0.0683	0.0682	0.0680	.0658	0.0657	0.0663	0.0665	0.0667	0.06
10	0.0811	0.0813	0.0816	0.0815	0.0812	.0782	0.0788	0.0791	0.0789	0.0788	0.07
20	0.1033	0.1034	0.1035	0.1033	0.1035	.0997	0.0997	0.0998	0.0997	0.0997	0.10
30	0.1127	0.1126	0.1130	0.1135	0.1142	.1102	0.1113	0.1117	0.1111	0.1103	0.11
40	0.1155	0.1147	0.1142	0.1150	0.1163	.1136	0.1129	0.1134	0.1134	0.1127	0.11
50	0.1166	0.1158	0.1152	0.1159	0.1171	.1154	0.1141	0.1147	0.1142	0.1143	0.11
60	0.1129	0.1124	0.1109	0.1109	0.1118	.1103	0.1084	0.1087	0.1086	0.1102	0.10
70	0.1053	0.1054	0.1040	0.1042	0.1045	.1035	0.1030	0.1034	0.1027	0.1042	0.10
80	0.0993	0.0997	0.0988	0.0997	0.1000	.0990	0.0997	0.1001	0.1000	0.1002	0.09
90	0.0954	0.0955	0.0944	0.0952	0.0956	.0960	0.0964	0.0962	0.0967	0.0963	0.09
100	0.1003	0.1001	0.0989	0.0992	0.0992	.1001	0.0997	0.0995	0.1005	0.1009	0.10
110	0.1060	0.1060	0.1042	0.1035	0.1027	.1045	0.1042	0.1034	0.1040	0.1050	0.10
120	0.1128	0.1130	0.1115	0.1107	0.1093	.1115	0.1101	0.1099	0.1098	0.1114	0.11
130	0.1162	0.1164	0.1157	0.1153	0.1149	.1169	0.1160	0.1153	0.1149	0.1159	0.11
140	0.1149	0.1150	0.1150	0.1146	0.1143	.1159	0.1149	0.1149	0.1150	0.1159	0.11
150	0.1124	0.1124	0.1127	0.1127	0.1130	.1137	0.1136	0.1137	0.1141	0.1143	0.11
160	0.1038	0.1037	0.1036	0.1030	0.1024	.1027	0.1022	0.1025	0.1028	0.1032	0.10
170	0.0787	0.0786	0.0784	0.0782	0.0781	.0800	0.0795	0.0796	0.0798	0.0799	0.07
180	0.0649	0.0648	0.0646	0.0642	0.0637	.0670	0.0664	0.0662	0.0662	0.0665	0.06

Figure 23.7c Horizontal splitting shows the extreme right columns of a file containing wide lines, while keeping the row labels visible on the left. (This file is a typical source-apodization file.)

Adding Your Own Script Templates

If you find the script templates useful, you might also like to extend the list by writing a few of your own. You can do this by following these steps:

1. Open a new Editor window.
2. Click the **Import Template** button, and choose **My Template** from the list.
3. Delete the sample lines provided, and insert your own script lines.
4. From the main menu, select **File> Template> Export**.
5. Name and **Save** your new template.

Programming the Custom Toolbar

Now that you know some ASAP commands, you might like to begin using the ASAP **Custom Toolbar**. Recall from Chapter 2 that this is the series of 10 preprogrammed buttons labeled with hammer icons. The toolbar is located at the right of the ASAP landscape (if you have not moved it). You can edit each of these custom buttons or add up to 10 more programmed buttons to run one or more ASAP commands that you routinely perform. ASAP sends these commands to the ASAP computational kernel for immediate execution whenever you click that button.

The new `VUFACETS` command, introduced earlier in this chapter, is a good candidate for this treatment. You begin the programming process by selecting **File> Preferences** from the main menu, and selecting **Custom Toolbar Editor** from among the tabs.

To program one of the buttons, select it from the **Button Name** list, and fill in the four fields as shown in Figure 23.8.

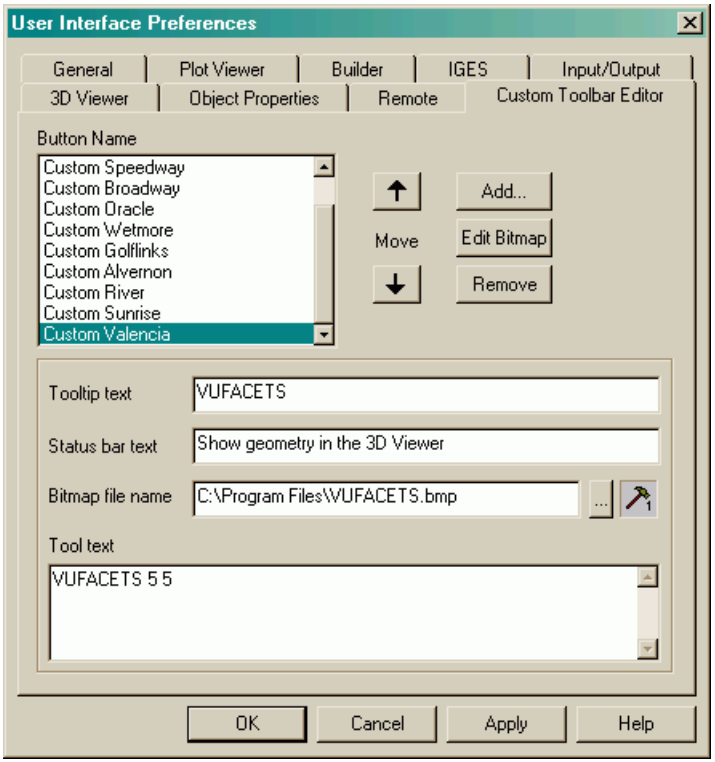


Figure 23.8 The **Custom Toolbar Editor** tab on the **User Interface Preferences** dialog box is for adding, editing, and removing buttons. The first 10 buttons are pre-assigned. You can add ten more buttons with command text to the toolbar. To edit an icon on a button, click **Edit Bitmap**.

ToolTip text. Text that appears when you hover over the button with the cursor without clicking.

Status bar text.Text that appears at the bottom of the ASAP landscape when you hover over the button with the mouse cursor without clicking. This is typically a longer, more detailed explanation of the button's function than that provided by the tool tip.

Tool text is the window where you list the commands that will be sent to the ASAP kernel, whenever this button is clicked.

Bitmap file name. The full path to an optional bitmap file that will replace the numbered hammer icon if you choose to do so. You can make your custom icon using the built-in Bitmap Editor (click the **Edit Bitmap** button on the dialog box) shown in Figure 23.9, or any program capable of creating and storing images as

bitmaps (bmp format), including the Paint program that comes with the Windows® operating system.

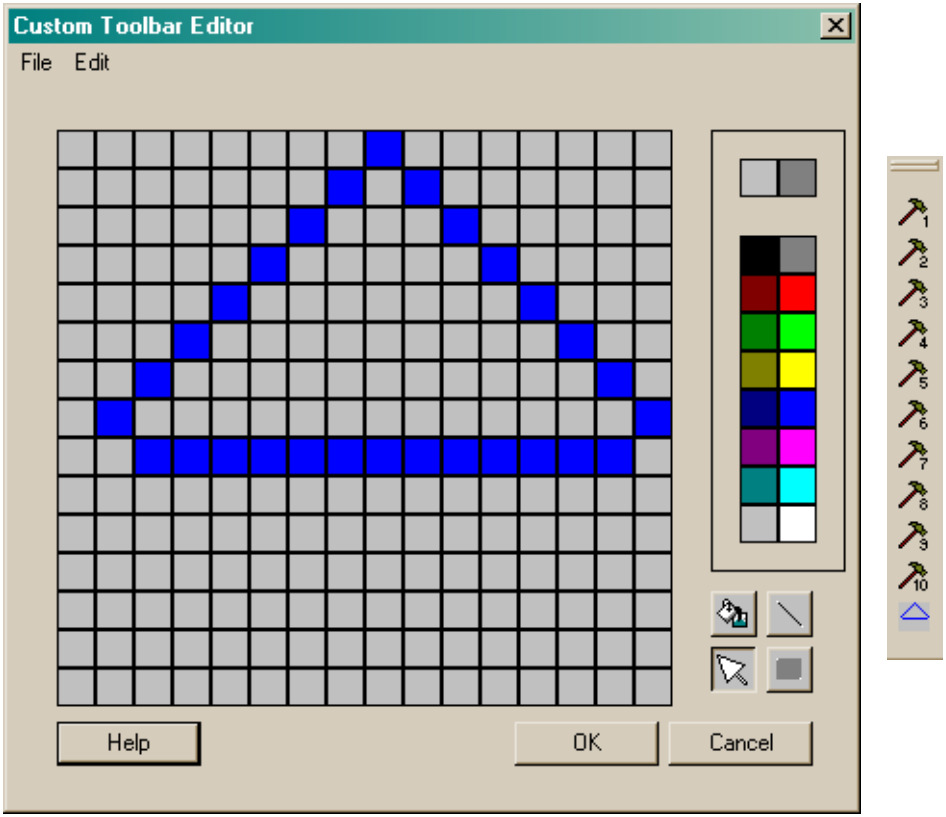


Figure 23.9 Use the **Custom Toolbar Editor**, or any drawing program, to design custom icons for your tool buttons. You do not need to reduce the size of your bitmap; ASAP scales the image to fit. The new button graphic is shown on the **Custom Toolbar** (right).

Summary

In this chapter, you were introduced to the following new commands.

ASAP Commands	Menu	Description
VUFACETS	—	Makes a vector file of the geometry, and displays it in the 3D Viewer.

When running and debugging ASAP scripts, we recommend developing and testing your script in small pieces. Editor buttons like **Run To Cursor**, and **Run From Cursor** make it easy to check out sections of your script. Many new and seasoned ASAP users also find it valuable to start simple, get the script running correctly, and add complexity in incremental steps.

As your skill and experience with the ASAP command language increases, you will likely be spending more and more of your time in this scripting environment. Some of the customization options will be of use to you, while others may not conform to your personal style. Learn the ones that look interesting, and skip the rest!

Exercise 11: Converting Flashlight Model to a Script

Write one ASAP script that duplicates the flashlight model of “Exercise 9: Simple Flashlight Model” on page 394, and recreates all of the graphical results.

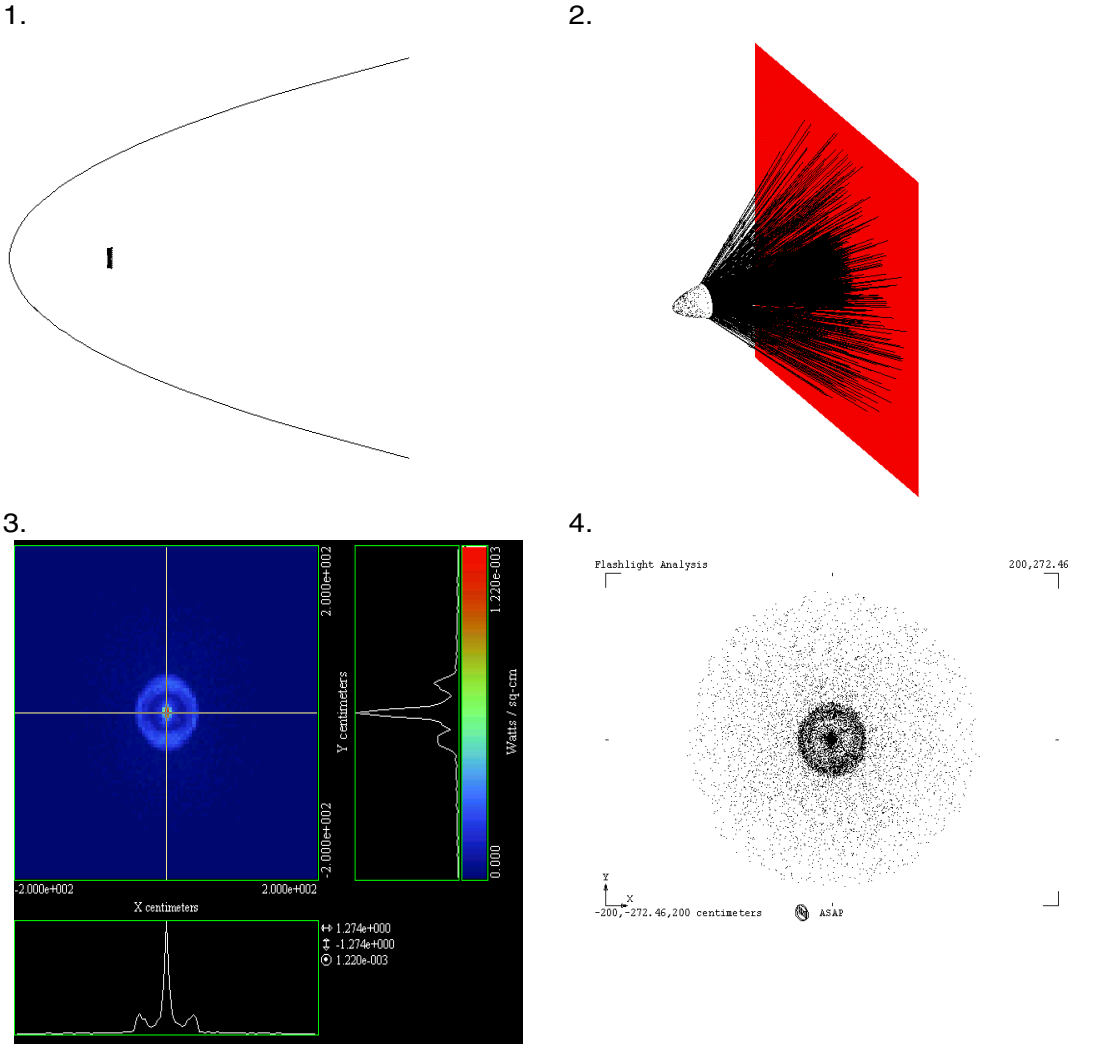


Figure 23.10 1. Verification of the dimensions of the source and its location with respect to the reflector, using the **PROFILES** and **SPOTS POSITION** commands. 2. Overlaying of the ray trace on the geometry. 3. Flux-weighted picture of the ray distribution. 4. Spot diagram, which shows only the distribution of ray positions without any indication of the flux of each ray.

Hints

- If you prefer, you can start by exporting your Builder file created as part of that exercise to script (inr) format, as described in Chapter 19. The Builder can speed you through the first few steps.
- Remember the **Script** button! You can use the dialog boxes associated with each tracing and graphical steps spelled out in “Exercise 9: Simple Flashlight Model” on page 394 to build the necessary command scripts to perform that work.

CHAPTER

24

OTHER COMMAND SCRIPT FEATURES

The previous two chapters concentrated on how to use ASAP command scripts to perform tasks previously accomplished with the Builder and the graphical user interface. In this chapter, we will expand on these topics to include some features that are specific to, or at least more commonly encountered in command scripts. We will also spend some time discussing the formal rules for expressing numerical values and writing mathematical expressions. We have encountered a few of these rules before, since many of them apply to Builder cells as well as command scripts. But now we will write them for you in one location to serve as a quick reference. In the last section, we will give a brief introduction to predefined macros that will allow you to add program flow control (conditional branching and do loops) to your scripts. This is a large topic, with the Technical Guide, *Predefined ASAP Macros*, dedicated to it. With the advent of script templates in the ASAP Editor, using these macros has become easy enough to deserve a brief introduction in this *Primer*.

Abbreviations, Shortcuts, and Special Characters

Strong arguments exist in favor of writing computer programs that use a minimum number of tricks and shortcuts. While saving you the trouble of typing a few extra keystrokes, such tricks sometimes make it difficult for others to read your code. The same rules apply to scripting languages like the ASAP command and macro languages. For this reason, we have avoided all use of shortcuts, abbreviations, and other special characters built into the ASAP command language that let you write shorter, more condensed ASAP scripts. The large database of example scripts included with your ASAP release was also written for clarity rather than economy of characters. Still, in the end it is a matter of style. You may know other ASAP users that have adopted these shortcuts, so we offer this section as a means of introducing you to the rules of the alternative philosophy.

Repeating numbers with @

ASAP offers a useful shortcut for entering numbers that are repeated in a command. When ASAP encounters 4@1.5 in a command, it expands this into four identical arguments:

```
1.5 1.5 1.5 1.5
```

If preceded by a minus sign, the signs of the numbers alternate: -4@1.5 becomes

```
-1.5 1.5 -1.5 1.5
```

One tempting opportunity to use this condensed syntax is the **GRID** command. The two lines below produce exactly the same grid of rays with somewhat less typing:

```
GRID ELLIPSE Z -2   -2.5 2.5   -2.5 2.5   9 9
GRID ELLIPSE Z -2   -4@2.5 on 2@9
```

Combining lines with ;

You can use semicolons if you want to put more than one command on the same line:

```
WINDOW Y Z; PROFILE OVERLAY; TRACE PLOT
```

This line is interpreted as:

```
WINDOW Y Z
PROFILE OVERLAY
TRACE PLOT
```

You have seen the semicolon before. It is frequently used by the Builder to concatenate two commands into one record (see Figure 22.10 on page 517 in Chapter 22). This notation has some advantages in a few situations, like user-defined macros in a large library. It puts more commands on fewer records, and allows ASAP to search the library more quickly. In general, however, it just makes commands harder to find in your file.

Command Continuation with ,

The comma (,) character can be used to continue a long command onto a second line. This technique is sometimes used just to make long commands easier to read. For example, consider the **OPTICAL** command with aspheric terms:

```
SURFACE
  OPTICAL Y -2 -30 -1,
          1E-3 -1E-6 1E-10 1E-14, !! Aspheric terms
          ELLIPSE 5
   OBJECT 'ASPHERE'
```

Abbreviations

ASAP keywords (commands and command options) may be shortened to as few as the minimum number of characters to remain unambiguous. For example,

```
SOURCE DIRECTION 0 0 1
```

can be shortened to

```
SOU DIR 0 0 1
```

In general, the ASAP Editor does not recognize the abbreviated forms of commands for the purposes of color syntax highlighting. Only some acceptable abbreviated forms have been included in the lookup file. Some notable examples are **SOU DIR**, **SPOTS POS** and **SPOTS DIR**. These forms are common enough to have gained entry into the table. If you consistently use certain abbreviations in your ASAP scripts, you may want to enter these into the table.

You can customize the names of ASAP commands, typically to shorten their names for easier scripting in the ASAP Editor. Keywords appear in colored syntax, and have associated command tips, which display command syntax. Keywords are created or deleted while you are in the Editor window by using the **Keyword Editor** dialog box, accessible from the **Editor** menu.

Entering Numerical Values

You may enter numbers onto a command line record (or into Builder cells) as integers, floating point, or in exponential form. ASAP converts your input to the type of number it is expecting.

Exponential numbers (like 1.602E-12) cannot contain more than 40 characters. The decimal point can be neglected; for example, 1.00E3 is the same as 1E3.

Some ASAP commands can accept complex numbers. For example, the **MEDIA** command can interpret an index of refraction with a real and imaginary part:

```
MEDIA
  0.20`3.44  'SILVER'
```

Note the use of the slanting single quote or grave accent. This symbol is found below the ~ key on most USA keyboards, and should not be confused with the single-quote (') punctuation symbol found on the same USA keyboard key as the double-quote punctuation symbol (") . At the risk of adding confusion to an already tricky problem, the single-quote symbol (or the acute accent on non-

USA keyboards) is also used to represent a complex number, but the two components now represent amplitude and phase (in degrees). To summarize,

`0.1`0.2` represents the complex number $0.1 + 0.2i$

`2.0'3.0` represents the complex number $2.0e^{3.0i}$

The **MEDIA ABSORB** command offers yet another way to indirectly specify a complex index of refraction. You can use this version of **MEDIA** to specify an absorption coefficient in inverse length units. See the online help for details and conversion formulae.

Expressions, Operators, and Functions

In the singlet example developed in Chapter 22, we used explicit numbers for the arguments in our various ASAP commands. We could have replaced any of these numbers by an "expression" to be evaluated. An expression is a collection of numbers, variables (see below), operators, and functions that ASAP can interpret and reduce to a single value. For example, if you have been given the diameters rather than the half apertures of your optical components, let ASAP do the math:

```
PLANE Z 0 ELLIPSE 2.5
```

is the same as

```
PLANE Z 0 ELLIPSE 5.0/2
```

You cannot use white space within an expression. ASAP, while interpreting your commands, is looking for spaces to mark the end of one expression and the beginning of another within the command line. Consequently, ASAP tries to interpret the delimited pieces as separate arguments.

The complete list of ASAP arithmetic operators is shown in Table 24.1.

Table 24.1 Arithmetic operators in ASAP

Operation	Description	Precedence
+	ADD the two entries	4
-	SUBTRACT the second from first	4
*	MULTIPLY the two entries	5
^	RAISE first TO second POWER	6
/	DIVIDE first by second	5
\	REMAINDER after dividing first by second	6
<	Take LESSER of the two	6
>	Take GREATER of the two	6
%	ARCTANGENT (angle in degrees) of the first divided by second	6
~	Uniformly distributed RANDOM number between first and second	7
`	Form complex number from REAL and IMAGINARY parts (grave accent)	0
'	Form complex number from MODULUS and PHASE angle in degrees (acute accent)	0
([	Store value of expression and operator to the left	1
)]	Recall previous value and operator, evaluate new expression	2

All these operators are colored red in the ASAP Editor, when the default color syntax highlighting is on. When any one of these symbols appears between two numbers (or other expressions), ASAP replaces the entry as described in the table. All these operators are assigned a default precedence as shown, but—as with most programming languages—you can use parentheses to take control over the order in which the operations are performed. You can create up to 10 levels of parenthetic nesting.

ASAP also has a set of mathematical functions. These are displayed in Table 24.2. Most of these functions probably look familiar, though it is unlikely that you have seen `FBI`, `LPW`, or `EYE`. These three functions are used to help you flux weight multiple-wavelength sources according to some prescription. They are discussed more completely in the Technical Guide, *Sources*.When ASAP encounters one of these functions, it replaces it and its argument with the value described in the table.

Table 24.2 Mathematical functions in ASAP

	( )	Either	[ ]
INT	Truncate to integer		Round to nearest integer
EXP	e^x		10^x
LOG	Natural logarithm $\log_e(X)$		Common logarithm $\log_{10}(X)$
ABS		Absolute Value	
SGN		Sign (returns -1, 0 or 1)	
SQRT		Square root	
CBRT		Cube root with same sign	
SIN	Sine of angle in RADIANS		Sine of angle in DEGREES
COS	Cosine of angle in RADIANS		Cosine of angle in DEGREES
TAN	Tangent in RADIANS		Tangent in DEGREES
ASIN	Arcsine in RADIANS		Arcsine in DEGREES
ACOS	Arcosine in RADIANS		Arcosine in DEGREES
ATAN	Arctangent in RADIANS		Arctangent in DEGREES
BJ#		#th-order Bessel J function (# from 0 to 9)	
BK#		Modified Bessel K function (# from 0 to 9)	
STEP		0 for $x<0$, 1 for $x>0$	
RECT		0 for $\|x\|> ½$ and 1 for $\|x\|< ½$	
GAUS	Gaussian $\exp(-x^2)$		Gaussian $\exp(-\pi x^2)$
SINC	$\sin(x)/x$		$\sin(\pi x)/\pi x$
SOMB	$2J_1(x)/x$		$J_1(\pi x)/\pi x$
FACT[1]		$x!$	
ERF	Error function		Complement: 1-ERF ( )
FBI[2]	Fractional black body energy		Fractional black body photons
LPW		Lumens per Watt for x degree Kelvin blackbody	
EYE[3]	Photopic (bright) response		Scotopic (dim) response
LIT[4]	Eight or less characters		16 or less characters
RAN	Unit RMS		Unit maximum

[1] FACT is actually gamma ($|x|+1$) with the sign of x.

[2] FBI is the Fractional Blackbody Integral from 0 to x, where x= λ T in microns and degrees Kelvin.

[3] EYE is the normalized Visual response at x microns; and [4] LIT converts a numerical value to the equivalent character string.

Note that many of these functions have two forms, depending on whether the argument is enclosed in parentheses ( ) or square brackets [ ]. For example, the cosine function accepts arguments as either radians (with parentheses) or degrees [with brackets]:

```
COS(3.14159) is the same as COS[180]
```

Functions with only one form are defined in the middle column of the table. Either parentheses or square brackets yield the same result.

Using Variables

ASAP allows you to store numerical values in named variables. Variable names can be up to 32 characters long (16 characters before version ASAP 7.1), and must begin with a letter. Variables do not have to be declared before use. When you store a value in a variable that ASAP has not seen before, it automatically places the variable name and value in one of 1768 internal registers. See the sidebar, “ASAP Registers” on page 551.

- Store a value in a variable using the “equals” sign:

  ```
  RADIUS=5.00

  DIAMETER=2*RADIUS

  TOTAL_FLUX=150*LPW(300)
  ```

  As with other expressions, no spaces are permitted in these command records.

- When you want to use the value stored in a variable, enclose the name of the variable in parentheses:

  ```
  SURFACE

    PLANE Z 0 ELLIPSE (RADIUS)
  ```

- The parentheses are not necessary when the variable name is used as part of a larger expression:

  ```
  SURFACE

    PLANE Z 0 ELLIPSE DIAMETER/2
  ```

Here are two important points about the initialization of variables:

1. You can use a variable without an initial value. It is set to zero, and ASAP proceeds with only minimal notification in the Command Output window. For example, if the **PLANE** command shown below was the first time ASAP encountered the variable `LOCATION`, ASAP would proceed with only the following statement in the Command Output window:

   ```
   ---   PLANE Z (LOCATION) ELLIPSE 2.5
   ```

```
LOCATION uninitialized - assuming a value of 0
```

2. ASAP registers (variables) retain their values after your script has finished running. Even **SYSTEM NEW** and **RESET** do not alter their state. All 1768 registers are re-initialized only when ASAP is restarted or when you click the **End** button.

Note: If you have an ASAP script that worked fine yesterday, but gives different results or errors today, the most common reason is failure to initialize variables. During a previous session, you may have had a reasonable value in an ASAP register from earlier runs, even though you neglected to initialize the variable in the final version of the script. Next time ASAP is started, a zero value is assumed for the uninitialized variable.

ASAP also supports up to 286 string variables for storing strings up to 32 characters long. There are restrictions on the names of these variables, however. They must be designated by the single letter A to Z by itself, or followed by a single number, 0 to 9 (for example, **A**, **R**, **N5**, **Z9**).

- String variables are also assigned a value with the equals sign, followed by the string within quotation marks:

```
S="ELLIPSE"
```

To use the string, use a quotation mark following the variable name:

```
 SURFACE

PLANE Z 0 S" 5.0
```

ASAP Macro Language

An ASAP macro is fundamentally no different from any other ASAP command, except that macros have more to do with programming than with optics.

You have already seen a few ASAP macro commands. These are the commands that begin with $ or &, like **$VIEW**, **$REG**, and **&REG**. An ASAP macro is fundamentally no different from any other ASAP command, except that macros have more to do with programming than with optics. We will demonstrate three of the most important capabilities that you can perform with ASAP, including do loops, conditional branches, and redirecting output to a text file. For more information, see the technical guides, *Predefined Macros* and *User-Defined Macros.*

$DO (Do loops)

Most programming languages include at least one way to repeat or “loop through” a group of commands. In ASAP, you do this with **$DO**. It allows us to repeat a block of commands, perhaps changing one or more parameters each time through the loop.

Begin by clicking the **Import Template** button on the ASAP Editor, and selecting **Do Loop** from the list. The result appears as shown at the top of Figure 24.1. As with all script templates, you will need to replace all the areas between (and including) the << >> symbols.

A simple example appears in the lower section of Figure 24.1. It will create three planes with unique names. The $DO macro works only in integer steps. This example loops three times, starting with a counter at -1, and ending when it reaches +1, in steps of one unit. An optional third argument allows you to increment by integers other than 1.

```
$DO <<start integer>> <<end integer>> <<increment integer>>
  {
    <<ASAP Commands or Macros>>
    <<...>>
  }
```

```
$DO -1 1
 {
 SURFACE
   PLANE Z 0 RECTANGLE 2@0.5
 OBJECT 'PLANE_?'
   INTERFACE 0 0 AIR AIR
   SHIFT X ? }
```

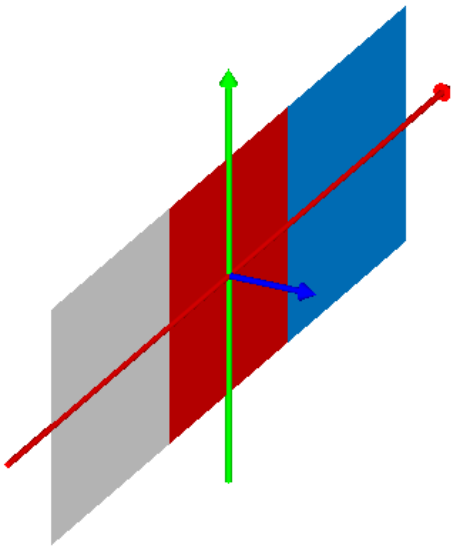


Figure 24.1 The $DO macro is used to loop through a series of ASAP commands. The script template is shown at the top, followed by an example script, and **3D Viewer** output. In this case, the loop was used to produce three planes, using the index counter to offset each.

Enclose the ASAP commands that you want to loop through in curly braces { } as shown in the figure. Everything else in the loop block looks like a normal ASAP command (or macro) except for the question mark (?) character. This place holder is for the counter. The first time through the loop, the characters -1 (a minus sign, and a 1) are substituted at the location of this place holder. In this way, the counter can be used either as part of a string as it is in the name on the OBJECT line (creating objects named PLANE_-1, PLANE_0 and PLANE_1), or as a numerical entry, as it was on the argument for the **SHIFT** command. You can form a more complex offset using an arithmetic expression that includes the loop counters.

$IF (Conditional Branches)

The **$IF** macro allows you to run one or more ASAP commands only when certain conditions are met. Once again, there is a template to help you with this. If you select **Simple IF THEN**, you see the template shown at the top of Figure 24.2. The example shown in the lower part of that figure presumes that the number of rays created has been saved in the variable NRAYS. We have tested the value of NRAYS to make sure it is not too large. If it is 100 or less, ASAP is asked to print the ray positions and directions using the **LIST RAYS** command. If there are more than 100 rays in the system, this list is skipped.

```
$IF <<Expression>> EQ <<Expression>> THEN
  <<ASAP Commands or Macros>>
  <<...>>
$ENDIF
```

```
$IF (NRAYS) LE 100 THEN
  CONSIDER ALL
  LIST RAYS
$ENDIF
```

```
--- $IF (NRAYS) LE 100 THEN
---   CONSIDER ALL
---   LIST RAYS
!      X             Y             Z             A             B             C             F             S             D          ;$FA
 0.518938      0.548041      0.00000       0.667278      0.209591      0.714711      0.100000      0.236925      0.349429      !
-.861983E-01 0.371442      0.00000       -.449266      -.364109      0.815834      0.100000      0.253131      0.327057      !
-.639310      -.529825      0.00000       -.465017      -.240083      0.852126      0.100000      0.258700      0.320016      !
0.973200      -.163058E-01  0.00000       -.170724      0.246333      0.954030      0.100000      0.273732      0.302442      !
-.835972      0.264218      0.00000       -.710153      0.429227      0.558074      0.100000      0.209359      0.395437      !
-.392065      0.458188      0.00000       0.559450      0.652826      0.510719      0.100000      0.200279      0.413364      !
0.135252E-01 0.174103      0.00000       0.715451      -.522840      0.463432      0.100000      0.190782      0.433941      !
0.863430      0.217749      0.00000       -.260898      0.596389E-01  0.963522      0.100000      0.275091      0.300949      !
0.144608      -.368358E-01  0.00000       -.209710      -.468036      0.858466      0.100000      0.259661      0.318832      !
0.405918      0.377538      0.00000       0.165320      0.460910      0.871913      0.100000      0.261687      0.316364      !
--- $ENDIF
```

Figure 24.2 The **$IF** macro is used to perform a series of ASAP commands when specified conditions are met. The **Simple IF THEN** template appears at the top, with a complete example below (along with some sample output). In this case, ASAP creates a numerical listing of ray data only when there are 100 or fewer rays.

A slightly more complex example appears in Figure 24.3, based on the **Multiple IF THEN** template. In this case, we have chosen whether or not to create a plot during the ray trace based on the number of rays, again stored in the variable NRAYS. We can test for several cases, and finally default to the lines following the $ELSE macro.

```
$IF <<Expression>> EQ <<Expression>> THEN
  <<ASAP Commands or Macros>>
  <<...>>

$ELSEIF <<Expression>> EQ <<Expression>> THEN
  <<ASAP Commands or Macros>>
  <<...>>

$ELSEIF <<Expression>> EQ <<Expression>> THEN
  <<ASAP Commands or Macros>>
  <<...>>

$ELSE
  <<ASAP Commands or Macros>>
  <<...>>

$ENDIF
```

```
$IF (NRAYS) LE 1000 THEN
  WINDOW Y Z
  PLOT FACETS 5 5 OVERLAY
  TRACE PLOT
$ELSE
  TRACE
$ENDIF
```

Figure 24.3 The $IF macro also has a multiple-case form, including a default ($ELSE), which allows more elaborate testing of conditions. This example, based on the **Multiple IF THEN** template, makes a plot of the geometry and rays when there are 1000 rays or less. Otherwise, it only traces.

$IO OUTPUT

The $IO macro has many functions, all of which affect ASAP input and output in some way. We will discuss only one of these: output redirection. This version of $IO allows you to take information that would normally appear only in the Command Output window, and redirect a copy to a named text file. The template for this is named **Output Redirection**, which appears at the top of Figure 24.4.

```
$ECHO NONE
$IO OUTPUT <<filename>> FILE(11)
  <<ASAP commands and macros>>
  <<...>>
$IO OUTPUT CLOSE
$ECHO
```

```
$ECHO NONE
$IO OUTPUT RAYS FILE(11)
  LIST RAYS
$IO OUTPUT CLOSE
$ECHO
```

Figure 24.4 Output redirection is used to send any information that would normally appear in the **Command Output** window to a text file. The above script was based on the **Output Redirection** template.

The example shown below the template redirects the same output shown in Figure 24.2, and places it into a file named **RAYS.OUT** in your working directory. Another macro, `$ECHO NONE`, is used to stop echoing commands into the Command Output window while you are redirecting the output. This macro prevents the `LIST RAYS`, and other commands used while redirection is on, from cluttering the output file. When we are finished, we revert to echoing with the `$ECHO` command with no arguments, which returns ASAP to the state prior to the last `$ECHO` macro.

Summary

In this chapter, we have discussed the following new commands.

ASAP Commands	Menu	Description
`$DO`	—	Loops through (repeats) a block of ASAP commands (input records).
`$IF`	—	Triggers a simple conditional branch.
`$IF...THEN`	—	Triggers a multiple case conditional branch.
`$ELSEIF`		
`$ELSE`		
`$ENDIF`		
`$IO OUTPUT...`	—	Controls input redirection.
`$ECHO`	—	Controls command echoing.

ASAP Registers

ASAP supports only a limited amount of space for user variables; a total of 1768 "registers" available for your use. Each of these registers holds two pieces of information:

- The name of the variable (up to 32 characters)
- A double-precision value

ASAP provides several useful predefined macros to help you manage your variables. The first, named **&REG**, *lets you list the current values of the register variables in use. You can type* **&REG** *into the* **Command Input** *window, or into a script like any ASAP command. The variable prints the current value of all registers in use to the* **Command Output** *window:*

```
--- RADIUS=21.3
--- RC=19.2224
--- PI=4*ATAN(1)
--- A=101.538
--- &REG
A=101.538
PI=3.14159265358979
RC=19.2224
RADIUS=21.3
```

Another version of this macro is named **$REG**. *In addition to printing the value of the registers in use, it also shows you the internal register names (in the left-most column) used to store the variables:*

```
--- $REG
A : A                                101.538
X~: PI                               3.14159265358979
Y~: RC                               19.2224
Z~: RADIUS                           21.3
```

While this extra information is seldom of much use to a user, it does give insight into the details of ASAP register use. Note, for example, that the internal register name of "A" is the same as the name assigned by the user (see below in the discussion of "direct registers").

Another useful macro is **$SHOW**. *It prints (among other things) the number of registers currently in use. You can use this macro to check to see how close you are to using up 1768 registers:*

```
REGISTER/VARIABLE STORAGE
    4 used  1764 left
```

Of the 1768 registers, 286 have a special status. They are known as "direct registers" and are available to us for the storage of strings. Strings can be up to 32 characters

long. There is no difference between the 286 direct registers and their 1482 indirect cousins, except that the 32 bytes used to store the variable name have been given over to the storage of your string. That is why you cannot name your string variables, but must use the internal name of the ASAP register (A, B, C, ... Z, A1, A2, ... Z9). A little experimenting will convince you that the string variables behave just like any other variable, if you treat the string value as a variable name:

```
--- A1="VARIABLE"
--- VARIABLE=42
--- $REG
A1: VARIABLE         42
```

When you use up the 1482 indirect variables, additional variable definitions begin using any available direct register space.

APPENDIX

1B

ASAP TECHNICAL PUBLICATIONS

The following ASAP technical publications are available on the CD. BRO recommends that you install the Primer on your machine from CD Autoplay. The documents are provided in Adobe Acrobat Reader format (PDF files). To view these documents, you must have Adobe Acrobat Reader 6.0 or later installed on your computer. You can also download the PDF files from the BRO Knowledge Base at http://www.breault.com/k-base.php

BRO #	Title	Description
0909	Radiometric Analysis	How to apply radiometric concepts in ASAP.
0911	Sources	How to use multiple sources and wavelengths.
0912	Edges	How to apply the terminology of edges and the basics of Bezier curves, to understand how parametric modeling works. Particular attention is given to the POINTS, PATCHES, and CONIC commands.
0913	Arrays and Bounds	How to use arrays and bounds in ASAP.
0915	Diffraction Gratings and DOEs	How to model diffractive optical elements in ASAP.
0916	Lens Entities	How to implement lens entities in ASAP and how these entities differ from surface and edge entities.

BRO #	Title	Description
0917	Predefined ASAP Macros	How to use available ASAP macros that allow flow control, input/output control, and subroutine capabilities within the ASAP scripting language.
0918	User-defined Macros	How to reuse blocks of custom ASAP script as user-defined macros.
0919	Wave Optics in ASAP	How to perform wave-optics calculations, including interferometry, diffraction, and partial coherence.
0920	IGES Import	How to use ASAP smartIGES™ with a CAD program to create geometry, import the geometry into ASAP for further development, add sources, and perform an analysis in ASAP.
0922	Scattering	How to use tools and methodologies for simulating light scattering in ASAP.
0923	Polarization	How to perform polarization analyses in ASAP.

Index

D

E

T

U

V

W

Z