3 OPTICAL DESIGN SOFTWARE

Douglas C. Sinclair
Sinclair Optics, Inc.
Fairport, New York

3.1 GLOSSARY

a	axial ray
b	chief ray
c	curvature
d, e, f, g	aspheric coefficients
efl	effective focal length
FN	focal ratio
$f( )$	function
h	ray height
m	linear, lateral magnification
n	refractive index
PIV	paraxial (Lagrange) invariant
r	entrance pupil radius
t	thickness
u	ray slope
y	coordinate
z	coordinate
α	tilt about x (Euler angles)
β	tilt about y (Euler angles)
γ	tilt about z (Euler angles)
ϵ	displacement of a ray from the chief ray
μ	Buchdahl coefficients
κ	conic constant
ρ	radial coordinate
σ_1	spherical aberration

σ_2 coma
σ_3 astigmatism
σ_4 Petzval blur
σ_5 distortion

3.2 INTRODUCTION

The primary function of optical design software is to produce a mathematical description, or *prescription,* describing the shapes, locations, materials, etc., of an optical system that satisfies a given set of specifications. A typical optical design program contains three principal sections: *data entry, evaluation,* and *optimization*. The optical design programs considered here are to be distinguished from *ray-trace* programs, which are mainly concerned with evaluation, and *CAD* programs, which are mainly concerned with drawings. The essence of an optical design program is its optimization section, which takes a starting design and produces a new design that minimizes an *error function* that characterizes the system performance.

The first practical computer software for optical design was developed in the 1950s and 1960s.[1–4] Several commercially available programs were introduced during the 1970s, and development of these programs has continued through the 1980s to the present time. Although decades have passed since the introduction of optical design software, developments continue in optimization algorithms, evaluation methods, and user interfaces.

This chapter attempts to describe a typical optical design program. It is intended for readers that have a general background in optics, but who are not familiar with the capabilities of optical design software. We present a brief description of some of the most important mathematical concepts, but make no attempt to give a detailed development. We hope that this approach will give readers enough understanding to know whether an optical design program will be a useful tool for their own work.

Of course, many different programs are available, each with its own advantages and disadvantages. Our purpose is not to review or explain specific programs, but to concentrate on the basic capabilities. Some programs work better than others, but we make no quality judgment. In fact, we avoid reference, either explicit or implicit, to any particular program. The features and benefits of particular optical design programs are more than adequately described by software vendors, who are listed in optical industry buyer's guides.[5]

Figure 1 is a flowchart of a typical optical design project. Usually, the designer not only must enter the starting design and initial optimization data, but also must continually monitor the progress of the computer, modifying either the lens data or the optimization data as the design progresses to achieve the best solution. Even when the performance requirements are tightly specified, it is often necessary to change the error function during the design process. This occurs when the error function does not correlate with the desired performance sufficiently well, either because it is ill-conceived, or because the designer has purposefully chosen a simple error function to achieve improved speed.

The fact that there are alternate choices of action to be taken when the design is not good enough has led to two schools of thought concerning the design of an optical design program. The first school tries to make the interface between the designer and the program as smooth as possible, emphasizing the interactive side of the process. The second school tries to make the error function comprehensive, and the iteration procedure powerful, minimizing the need for the designer to intervene.

3.3 LENS ENTRY

In early lens design programs, lens entry was a "phase" in which the lens data for a starting design was read into the computer from a deck of cards. At that time, the numerical aspects of optical design on a computer were so amazing that scant attention was paid to the lens entry process. However, as the use of optical design software became more widespread, it was found that a great deal of a designer's time was spent punching cards and submitting new jobs, often to correct past mistakes. Many times, it turned out that the hardest part of a design job was preparing a "correct" lens deck!

OPTICAL DESIGN SOFTWARE

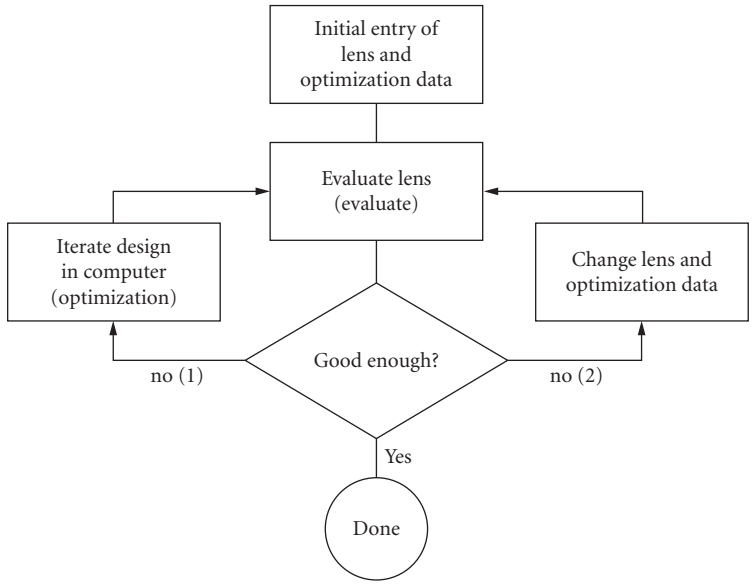

FIGURE 1 Flowchart for the lens design process. The action taken when a design is not satisfactory depends on how bad it is. The designer (or a design program) may change the lens data, or redefine the targets to ones that can be achieved.

Over the years, optical design programs have been expanded to improve the lens entry process, changing the function of this part of the program from simple lens entry to what might be called lens database management. A typical contemporary program provides on-line access to a library of hundreds of lenses, interactive editing, automatic lens drawings, and many features designed to simplify this aspect of optical design.

The lens database contains all items needed to describe the optical system under study, including not only the physical data needed to construct the system (curvatures, thicknesses, etc.), but also data that describe the conditions of use (object and image location, field of view, etc.). Some programs also incorporate optimization data in the lens database, while others provide separate routines for handling such data. In any case, the lens database is often the largest part of an optical design program.

The management of lens data in an optical design program is complicated by two factors. One is that there is a tremendous range of complexity in the types of systems that can be accommodated, so there are many different data items. The other is that the data are often described indirectly. A surface curvature may be specified, for example, by the required slope of a ray that emerges from the surface, rather than the actual curvature itself. Such a specification is called a *solve*, and is based on the fact that paraxial ray tracing is incorporated in the lens entry portion of most optical design programs.

It might seem curious that paraxial ray tracing is still used in contemporary optical design programs that can trace *exact* rays in microseconds. (The term *exact* ray is used in this chapter to mean a real skew ray. Meridional rays are treated as a special case of skew rays in contemporary software; there is not sufficient computational advantage to warrant a separate ray trace for them.) In fact, paraxial rays have some important properties that account for their incorporation in the lens database.

First, paraxial rays provide a linear system model that leads to analysis of optical systems in terms of simple bilinear transforms. Second, paraxial ray tracing does not fail. Exact rays can miss surfaces or undergo total internal reflection. Finally, paraxial rays determine the ideal performance of a lens. In a well-corrected lens, the aberrations are balanced so that the exact rays come to the image points defined by the paraxial rays, not the other way around.

Two basic types of data are used to describe optical systems. The first are the *general* data that are used to describe the system as a whole, and the other are the *surface* data that describe the individual surfaces and their locations. Usually, an optical system is described as an ordered set of surfaces,

beginning with an *object* surface and ending with an *image* surface (where there may or may not be an actual image). It is assumed that the designer knows the order in which rays strike the various surfaces. Systems for which this is not the case are said to contain *nonsequential* surfaces, which are discussed later.

General System Data

The general data used to describe a system include the aperture and field of view, the wavelengths at which the system is to be evaluated, and perhaps other data that specify evaluation modes, vignetting conditions, etc.

Aperture and Field of View The aperture and field of view of a system determine its conditions of use. The aperture is specified by the *axial ray*, which emerges from the vertex of the object surface and passes through the edge of the entrance pupil. The field of view is specified by the *chief ray*, which emerges from the edge of the object and passes through the center of the entrance pupil.

There are various common ways to provide data for the axial and chief rays. If the object is at an infinite distance, the entrance pupil radius and semifield angle form a convenient way to specify the axial and chief rays. For finite conjugates, the numerical aperture in object space and the object height are usually more convenient.

Some programs permit the specification of paraxial ray data by image-space quantities such as the *f*-number and the image height, but such a specification is less desirable from a computational point of view because it requires an iterative process to determine initial ray-aiming data.

Wavelengths It is necessary to specify the wavelengths to be used to evaluate polychromatic systems. Three wavelengths are needed to enable the calculation of primary and secondary chromatic aberrations. More than three wavelengths are required to provide an accurate evaluation of a typical system, and many programs provide additional wavelengths for this reason. There has been little standardization of wavelength specification. Some programs assume that the first wavelength is the central wavelength, while others assume that it is one of the extreme wavelengths; some require wavelengths in micrometers, while others in nanometers.

Other General Data Several other items of general data are needed to furnish a complete lens description, but there is little consistency between programs on how these items are treated, or even what they are. The only one that warrants mention here is the aperture stop. The *aperture stop* is usually defined to be the surface whose aperture limits the angle of the axial ray. Once the aperture stop surface is given, the positions of the paraxial pupils are determined by the imaging properties of the system. Since the aperture and field of view are determined formally by the paraxial pupils, the apertures are not associated with the exact ray behavior.

The "vignetting factor" is used to account for the differences between paraxial and exact off-axis ray heights at apertures. In particular, the vignetting factor provides, in terms of fractional (paraxial) coordinates, the data for an exact ray that grazes the apertures of a system. Typically, there is an upper, lower, and skew vignetting factor. The details of how such factors are defined and handled are program dependent.

Surface Data

Surface Location There are two basic ways to specify the location of surfaces that make up a lens. One is to specify the position of a surface relative to the immediately preceding surface. The other is to specify its position relative to some fixed surface (for example, the first surface). The two ways lead to what are called *local* and *global* coordinates, respectively. For ordinary lenses consisting of a series of rotationally symmetric surfaces centered on an optical axis, local coordinates are more convenient, but for systems that include reflectors, tilted, and/or decentered surfaces, etc., global

coordinates are simpler. Internally, optical design programs convert global surface data to local coordinates for speed in ray tracing.

Most optical design programs use a standard coordinate system and standard sign conventions, although there are exceptions.[6] Each surface defines a local right-handed coordinate system in which the z axis is the symmetry axis and the yz plane is the meridional plane. The local coordinate system is used to describe the surface under consideration and also the origin of the next coordinate system. Tilted elements are described by an Euler-angle system in which α is a tilt around the x axis, β is a tilt around the y axis, and γ is a tilt around the z axis. Since tilting and decentering operations are not commutative; some data item must be provided to indicate which comes first.

Surface Profile Of the various surfaces used in optical systems, the most common by far is the rotationally symmetric surface, which can be written as[7]

$$z = \frac{cr^2}{1+\sqrt{1-c^2(\kappa+1)r^2}} + dr^4 + er^6 + fr^8 + gr^{10}$$

$$r = \sqrt{x^2 + y^2}$$

c is the curvature of the surface; κ is the conic constant; and d, e, f, and g are aspheric constants. The use of the above equation is almost universal in optical design programs. The description of conic surfaces in terms of a conic constant κ instead of the eccentricity e used in the standard mathematical literature allows spherical surfaces to be specified as those with no conic constant. (The conic constant is minus the square of the eccentricity.)

Although aspheric surfaces include all surfaces that are not spherical, from a design standpoint there is a demarcation between "conic" aspheres and "polynomial" aspheres described using the coefficients d, e, f, and g. Rays can be traced analytically through the former, while the latter require numerical iterative methods.

Many optical design programs can handle surface profiles that are more complicated than the above, including cylinders, torics, splines, and even general aspheres of the form $z = f(x, y)$, where $f(x, y)$ is an arbitrary polynomial. The general operation of an optical design program, however, can be understood by considering only the rotationally symmetric surfaces described here.

As mentioned above, the importance of paraxial rays in optical system design has led to the indirect specification of lens data, using *solves,* as they are called, which permit a designer to incorporate the basic paraxial data describing a lens with the lens itself, rather than having to compute and optimize the paraxial performance of a lens as a separate task. Considering the jth surface of an optical system, let

y_j = ray height on surface
u_j = ray slope on image side
c_j = curvature of surface
n_j = refractive index on image side
t_j = thickness on image side

The paraxial ray trace equations can then be written as[8]

$$y_j = y_{j-1} + t_{j-1}u_{j-1}$$

$$n_j u_j = n_{j-1}u_{j-1} - y_j c_j (n_j - n_{j-1})$$

These equations can be inverted to give the curvatures and thicknesses in terms of the ray data. We have

$$c_j = \frac{n_{j-1}u_{j-1} - n_j u_j}{y_j(n_j - n_{j-1})}$$

$$t_j = \frac{y_{j+1} - y_j}{u_j}$$

The specification of curvatures and thicknesses by solves is considered to be on an equal basis with the direct specification of these items. The terminology used to specify solves is that the solves used to determine thickness are called *height solves,* and the solves used to determine curvature are called *angle solves.* Often, an axial ray height solve on the last surface is used to automatically locate the paraxial image plane, a chief ray height solve on the same surface to locate the exit pupil, and an axial ray angle solve is used to maintain a given focal length (if the entrance pupil radius is fixed). In some programs, additional types of solves are allowed, such as center of curvature solves, or aperture solves.

Of course, specifying lens data in terms of paraxial ray data means that whenever any lens data is changed, two paraxial rays must be traced through the system to resolve any following data that are determined by a solve. In an optical design program, this function is performed by a *lens setup* routine, which must be efficiently coded, since it is executed thousands of times in even a small design project.

Other functions of the lens setup routine are to precalculate values that are needed for repetitive calculations, such as refractive indices, rotation and translation matrices, etc. Many programs have the capability of specifying certain data items to be equal to (±) the value of the corresponding item on a previous surface. These are called *pickups,* and are needed for optimization of systems containing mirrors, as well as maintaining special geometrical relationships. Programs that lack pickups usually have an alternate means for maintaining the required linking between data items. Like solves, pickups are resolved by the lens setup routine, although they do not use paraxial data.

Other Surface Data *A* variety of other data is required to specify surfaces. Most important are apertures, discussed below, and refractive indices. Refractive indices are usually given by specifying the name of a catalog glass. In the lens setup routine, the actual refractive indices are calculated using an index interpolation formula and coefficient supplied by the glass manufacturer, together with the design wavelengths stored with the lens data. Other surface-related items include phase data for diffractive surfaces, gradient-index data, holographic construction data, and coatings.

Apertures have a somewhat obscure status in many optical design programs. Although apertures have a major role to play in determining the performance of a typical system, they do not usually appear directly in optimization functions. Instead, apertures are usually controlled in optimization by targets on the heights of rays that define their edges. If an aperture is specified directly, it will block rays that pass outside of it and cause typical optimization procedures to become unstable. Accordingly, some programs ignore apertures during optimization. Other programs allow the apertures to be determined by a set of exact "reference rays" that graze their extremities.

Nonsequential Surfaces In some optical systems, it is not possible to specify the order in which a ray will intersect the surfaces as it progresses through the system. The most common examples of such systems are prisms such as the corner-cube reflector, where the ordering of surfaces depends on the entering ray coordinates. Other examples of nonsequential surfaces include light pipes and a variety of nonimaging concentrators. Nonsequential surfaces can be accommodated by many optical design programs, but for the most part they are not "designed" using the program, but rather are included as a subsystem used in conjunction with another part of the system that is the actual

system being designed. Data specification of nonsequential surfaces is more complicated than ordinary systems, and ray tracing is much slower, since several surfaces must be investigated to see which surface is the one actually traversed by a given ray.

Lens Setup

Whenever the lens entry process is completed, the lens must be "set up." Pickup constraints must be resolved. If the system contains an internal aperture stop, the position of the entrance pupil must be determined. Then paraxial axial and chief rays must be traced through the system so that surface data specified by solves can be computed. Depending on the program, a variety of other data may be precomputed for later use, including aperture radii, refractive indices, and various paraxial constants.

The lens setup routine must be very efficient, since it is the most heavily used code in an optical design program. In addition to running whenever explicit data entry is complete, the code is also executed whenever the lens is modified internally by the program, such as when derivatives are computed during optimization, or when configurations are changed in a multiconfiguration system. Typically, lens setup takes milliseconds (at most), so it is not noticed by the user, other than through its effects.

Programming Considerations

In writing an optical design program, the programmer must make a number of compromises between speed, size, accuracy, and ease of use. These compromises affect the usefulness of a particular program for a particular application. For example, a simple, fast, small program may be well suited to a casual user with a simple problem to solve, but this same program may not be suited for an experienced designer who routinely tackles state-of-the-art problems.

The lens entry portion of an optical design program shows, more than any other part, the difference in programming models that occurred during the 1980s. Before the 1980s, most application programs were of a type called *procedural* programs. When such a program needs data, it requests it, perhaps from a file or by issuing a prompt for keyboard input. The type of data needed is known, and the program is only prepared to accept that kind of data at any given point. Although the program may branch through different paths in response to the data it receives, the program is responsible for its own execution.

With the popularization in the 1980s of computer systems that use a mouse for input the model for an application program changed from the procedural model described above to what is called an *event-driven* model. An event-driven program has a very simple top-level structure consisting of an initialization section followed by an infinite loop usually called something like the *main event loop.* The function of the main event loop is to react to user-initiated interrupts (such as pressing a key, or clicking a mouse button), dispatching such events to appropriate processing functions. In such a program, the user controls the execution, unlike a procedural program, where the execution controls the user.

An event-driven program usually provides a better user interface than a procedural program. Unfortunately, most optical design programs were originally written as procedural programs, and it is difficult to convert a procedural program into an event-driven program by "patching" it. Usually it is easier to start over. In addition, it is harder to write an event-driven program than a procedural program, because the event-driven program must be set up to handle a wide variety of unpredictable requests received at random times. Of course, it is this very fact that makes the user interface better. There is an aphorism sometimes called the "conservation of complexity," which states that the simpler a program is to use, the more complicated the program itself must be.

The data structures used to define lens data in an optical design program may have a major impact on its capabilities. For example, for various reasons it is usually desirable to represent a lens in a computer program as an array of surfaces. If the maximum size of the array is determined at compile time, then the maximum size lens that can be accommodated is built into the program.

As more data items are added to the surface data, the space required for storage can become unwieldy. For example, it takes about 10 items of real data to specify a holographic surface. If every surface were allowed to be a hologram, then 10 array elements would have to be reserved for each surface's holographic data. On the other hand, in most systems, the elements would never be used, so the data structure would be very inefficient. To avoid this, a more complicated data structure could be implemented in which only one element would be devoted to holograms, and this item would be used as an index into a separate array containing the actual holographic data. Such an array might have a sufficient number of elements to accommodate up to, say, five holograms, the maximum number expected in any one system.

The preceding is a simple example of how the data structure in an optical design program can grow in complexity. In fact, in a large optical design program the data structure may contain all sorts of indices, pointers, flags, etc., used to implement special data types and control their use. Managing this data while maintaining its integrity is a programming task of a magnitude often greater than the numerical processing that takes place during optical design.

Consider, for example, the task of deleting a surface from a lens. To do this, the surface data must of course be deleted, and all of the higher-numbered surfaces renumbered. But, in addition, the surface must be checked to see whether it is a hologram and, if so, the holographic data must also be deleted and that data structure "cleaned up." All other possible special data items must be tested and handled similarly. Then all the renumbered surfaces must be checked to see if any of the "pick up" data from a surface that has been renumbered, and the reference adjusted accordingly. Then other data structures such as the optimization files must be checked to see if they refer to any of the renumbered surfaces, and appropriate adjustments made. There may be several other checks and adjustments that must also be carried out.

Related to the lens entry process is the method used to store lens data on disc. Of course, lens data are originally provided to a program in the form of text (e.g., "TH 1.0"). The program *parses* this data to identify its type (a thickness) and value (1.0). The results of the parsing process (the binary values) are stored in appropriate memory locations (arrays). To store lens data on disc, early optical design programs used the binary data to avoid having to reparse it when it was recovered. However, the use of binary files has decreased markedly as computers have become fast enough that parsing lens input does not take long. The disadvantages of binary files are that they tend to be quite large, and usually have a structure that makes them obsolete when the internal data structure of the program is changed. The alternative is to store lens data as text files, similar in form to ordinary keyboard input files.

3.4 EVALUATION

Paraxial Analysis

Although the lens setup routine contains a paraxial ray trace, a separate paraxial ray trace routine is used to compute data for display to the user. At a minimum, the paraxial ray heights and slopes of the axial and chief ray are shown for each surface, in each color, and in each configuration.

The equations used for paraxial ray tracing were described in the previous section. Although such equations become exact only for "true" paraxial rays that are infinitesimally displaced from the optical axis, it is customary to consider paraxial ray data to describe "formal" paraxial rays that refract at the tangent planes to surfaces, as shown in Fig. 2. Here, the ray ABC is a paraxial ray that provides a first-order approximation to the exact ray ADE. Not only does the paraxial ray refract at the (imaginary) tangent plane BVP, but also it bends a different amount from the exact ray.

In addition to the computation of ray heights and slopes for the axial and chief ray, various paraxial constants that characterize the overall system are computed. The particular values computed depend on whether the system is *focal* (finite image distance) or *afocal* (image at infinity). For focal systems, the quantities of interest are (at a minimum) the focal length efl, the *f*-number FN, the paraxial (Lagrange) invariant PIV, and the transverse magnification *m*. It is desirable to compute such

OPTICAL DESIGN SOFTWARE

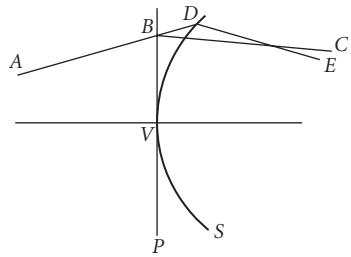

FIGURE 2 Showing the difference between a paraxial ray and a real ray. The paraxial ray propagates along *ABC*, while the real ray propagates along *ADE*.

quantities in a way that does not depend on the position of the final image surface. Let the object height be h, the entrance pupil radius be r, the axial ray data in object and image spaces be y, u and y', u', the chief ray data be $\bar{y}$, $\bar{u}$ and $\bar{y}'$, $\bar{u}'$, and the refractive indices be n and n'.

The above-mentioned paraxial constants are then given by

$$\text{efl} = \frac{-rh}{\bar{u}'r + u'h}$$

$$\text{FN} = -\frac{n}{2n'u'}$$

$$\text{PIV} = n(\bar{y}u - y\bar{u})$$

$$m = \frac{nu}{n'u'}$$

In addition to the paraxial constants, most programs display the locations of the entrance and exit pupils, which are easily determined using chief-ray data. Surprisingly, most optical design programs do not explicitly show the locations of the principal planes. In addition, although most programs have the capability to display "$y - \bar{y}$" plots, few have integrated this method into the main lens entry routine.

Aberrations

Although most optical designs are based on exact ray data, virtually all programs have the capability to compute and display first-order chromatic aberrations and third-order monochromatic (Seidel) aberrations. Many programs can compute fifth-order aberrations as well. The form in which aberrations are displayed depends on the program and the type of system under study, but as a general rule, for focal systems aberrations are displayed as equivalent ray displacements in the paraxial image plane.

In the case of the chromatic aberrations, the primary and secondary chromatic aberration of the axial and chief rays are computed. In a system for which three wavelengths are defined, the primary aberration is usually taken between the two outer wavelengths, and the secondary aberration between the central and short wavelengths.

The Seidel aberrations are computed according to the usual aberration polynomial. If we let ϵ be the displacement of a ray from the chief ray, then

$$\epsilon_y = \epsilon_{3y} + \epsilon_{5y} + \cdots$$

$$\epsilon_x = \epsilon_{3x} + \epsilon_{5x} + \cdots$$

For a relative field height h and normalized entrance pupil coordinates r and θ, the third-order terms are

$$\epsilon_{3y} = \sigma_1 \cos\theta r^3 + \sigma_2 (2 + \cos 2\theta) r^2 h + (3\sigma_3 + \sigma_4) \cos\theta r h^2 + \sigma_5 h^3$$

$$\epsilon_{3x} = \sigma_1 \sin\theta r^3 + \sigma_2 \sin 2\theta r^2 h + (\sigma_3 + \sigma_4) \sin\theta r h^3$$

The interpretation of the coefficients is generally as follows, but several optical design programs display tangential coma, rather than the sagittal coma indicated in the table.

σ_1	Spherical aberration
σ_2	Coma
σ_3	Astigmatism
σ_4	Petzval blur
σ_5	Distortion

The fifth-order terms are

$$\epsilon_{5y} = \mu_1 \cos\theta r^5 + (\mu_2 + \mu_3 \cos 2\theta) r^4 h + (\mu_4 + \mu_6 \cos^2\theta) \cos\theta r^2 h^2$$
$$+ (\mu_7 + \mu_8 \cos 2\theta) r^2 h^3 + \mu_{10} \cos\theta r h^4 + \mu_{12} h^5$$

$$\epsilon_{5x} = \mu_1 \sin\theta r^5 + \mu_3 \sin 2\theta r^4 h + (\mu_5 + \mu_6 \cos^2\theta) \sin\theta r^3 h^2$$
$$+ \mu_9 \sin 2\theta r^2 h^3 + \mu_{11} \sin\theta r h^4$$

These equations express the fifth-order aberration in terms of the Buchdahl μ coefficients. In systems for which the third-order aberrations are corrected, the following identities exist:

$$\mu_2 = \frac{3}{2}\mu_3$$

$$\mu_4 = \mu_5 + \mu_6$$

$$\mu_7 = \mu_8 + \mu_9$$

μ_1	Spherical aberration
μ_3	Coma
$(\mu_{10} - \mu_{11})/4$	Astigmatism
$(5\mu_{11} - \mu_{10})/4$	Petzval blur

$\mu_4 + \mu_6$	Tangential oblique spherical aberration
μ_5	Sagittal oblique spherical aberration
$\mu_7 + \mu_8$	Tangential elliptical coma
μ_9	Sagittal elliptical coma
μ_{12}	Distortion

Some programs display only the aberrations that have corresponding third-order coefficients, omitting oblique spherical aberration and elliptical coma.

The formulas needed to calculate the chromatic and third-order aberrations are given in the *U.S. Military Handbook of Optical Design.* The formulas for calculating the fifth-order aberrations are given in Buchdahl's book.[9]

Aberration coefficients are useful in optical design because they characterize the system in terms of its symmetries, allow the overall performance to be expressed as a sum of surface contributions, and are calculated quickly. On the negative side, aberration coefficients are not valid for systems that have tilted and decentered elements for systems that cover an appreciable field of view, and the accuracy of aberration coefficients in predicting performance is usually inadequate. Moreover, for systems that include unusual elements like diffractive surfaces and gradient index materials, the computation of aberration coefficients is cumbersome at best.

Ray Tracing

Exact ray tracing is the foundation of an optical design program, serving as a base for both evaluation and optimization. From the programmer's standpoint, the exact ray-trace routines must be accurate and efficient. From the user's viewpoint, the data produced by the ray-trace routines must be accurate and comprehensible. Misunderstanding the meaning of ray-trace results can be the source of costly errors in design.

To trace rays in an optical design program, it is necessary to understand how exact rays are specified. Although the details may vary from one program to the next, many programs define a ray by a two-step process. In the first step, an object point is specified. Once this has been done, all rays are assumed to originate from this point until a new object point is specified. The rays themselves are then specified by aperture coordinates and wavelength.

Exact ray starting data is usually normalized to the object and pupil coordinates specified by the axial and chief rays. That is, the aperture coordinates of a ray are specified as a fractional number, with 0.0 representing a point on the vertex of the entrance pupil, and 1.0 representing the edge of the pupil. Field angles or object heights are similarly described, with 0.0 being a point on the axis, and 1.0 being a point at the edge of the field of view.

Although the above normalization is useful when the object plane is at infinity, it is not so good when the object is at a finite distance and the numerical aperture in object space is appreciable. Then, fractional aperture coordinates should be chosen proportional to the direction cosines of rays leaving an object point. There are two reasons for this. One is that it allows an object point to be considered a point source, so that the amount of energy is proportional to the "area" on the entrance pupil. The other is that for systems without pupil aberrations, the fractional coordinates on the second principal surface should be the same as those on the first principal surface. Notwithstanding these requirements, many optical design programs do not define fractional coordinates proportional to direction cosines.

It is sometimes a point of confusion that the aperture and field of view of a system are specified by paraxial quantities, when the actual performance is determined by exact rays. In fact, the paraxial specifications merely establish a normalization for exact ray data. For example, in a real system the field of view is determined not by the angle of the paraxial chief ray, but by the angle at which exact rays blocked by actual apertures just fail to pass through the system. Using an iterative procedure, it is not too hard to find this angle, but because of the nonlinear behavior of Snell's law, it does not provide a convenient reference point.

There are two types of exact rays: *ordinary* or *lagrangian* rays, and *iterated* or *hamiltonian* rays. The designation of rays as lagrangian or hamiltonian comes from the analogy to the equations of motion of a particle in classical mechanics. Here we use the more common designation as ordinary or iterated rays. An ordinary ray is a ray that starts from a known object point in a known direction. An iterated ray also starts from a known object point, but its direction is not known at the start. Instead, it is known that the ray passes through some known (nonconjugate) point inside the system, and the initial ray direction is determined by an iterative procedure.

Iterated rays have several applications in optical design programs. For example, whenever a new object point is specified, it is common to trace an iterated ray through the center of the aperture stop (or some other point) to serve as a reference ray, or to trace several iterated rays through the edges of limiting apertures to serve as reference rays. In fact, many programs use the term *reference ray* to mean iterated ray (although in others, reference rays are ordinary rays). Iterated rays are traced using differentially displaced rays to compute corrections to the initial ray directions. Because of this, they are traced slower than ordinary rays. On the other hand, they carry more information in the form of the differentials, which is useful for computing ancillary data like field sags.

Reference rays are used as base rays in the interpretation of ordinary ray data. For example, the term *ray displacement* often refers to the difference in coordinates on the image surface of a ray from those of the reference ray. Similarly, the *optical path difference* of a ray may compare its phase length to that of the corresponding reference ray. The qualifications expressed in the preceding sentences indicate that the definitions are not universal. Indeed, although the terms *ray displacement* and *optical path difference* are very commonly used in optical design, they are not precisely defined, nor can they be. Let us consider, for example, the optical path difference.

Imagine a monochromatic wavefront from a specified object point that passes through an optical system. Figure 3 shows the wavefront *PE* emerging in image space, where it is labeled "actual wavefront." Because of aberrations, an ordinary ray perpendicular to the actual wavefront will not intersect the final image surface at the ideal image point *I*, but at some other point *Q*. The optical path difference (OPD) may be defined as the optical path measured along the actual ray between the actual wavefront and a reference sphere centered on the ideal image point.

Unfortunately, the ideal image point is not precisely defined. In the figure, it is shown as the intersection of the reference ray with the image surface, but the reference ray itself may not be precisely defined.

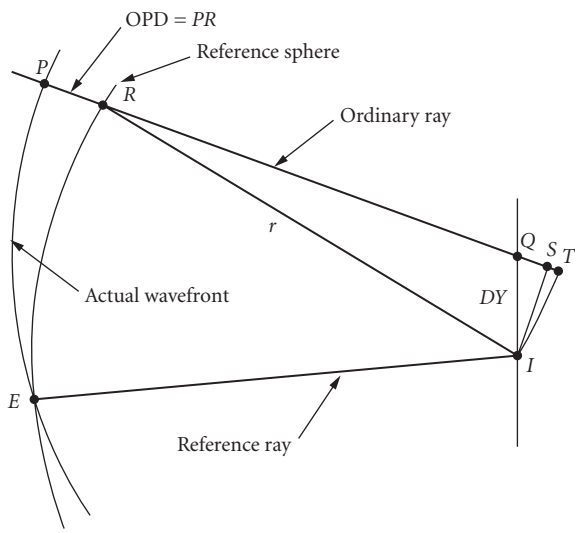

FIGURE 3 The relation between ray trajectories and optical path difference (OPD). See text.

Is it the ray through the center of the aperture stop, or perhaps the ray through the center of the actual vignetted aperture? These two definitions will result in different reference rays, and correspondingly different values for the optical path difference. In fact, in many practical applications neither definition is used, and the actual ideal image point is defined to be the one that minimizes the variance of the optical path difference (and hence maximizes the peak intensity of the diffraction image).

Moreover, the figure shows that even if the ideal image point is precisely defined, the value of the optical path difference depends on the point *E* where the actual wavefront interesects the reference sphere. For the particular point shown, the optical path difference is the optical length along the ordinary ray from the object point to the point *T,* less the optical length along the reference ray from the object point to the point *I*. As the radius of the reference sphere is increased, the point *T* merges with the point *S,* where a perpendicular from the ideal image point intersects the ordinary ray.

The above somewhat extended discussion is meant to demonstrate that even "well-known" optical terms are not always precisely defined. Not surprisingly, various optical design programs in common use produce different values for such quantities. There has been little effort to standardize the definitions of many terms, possibly because one cannot legislate physics. In any case, it is important for the user of an optical design program to understand precisely what the program is computing.

Virtually all optical design programs can trace single rays and display the ray heights and direction cosines on each surface. Other data, such as the path length, angles of incidence and refraction, and direction of the normal vector, are also commonly computed. Another type of ray-data display that is nearly universal is the ray-intercept curve, which shows ray displacement on the final image surface versus (fractional) pupil coordinates. A variation plots optical path difference versus pupil coordinates.

In addition to the uncertainty concerning the definition of ray displacement and optical path difference, there are different methods for handling the pupil coordinates. Some programs use entrance pupil coordinates, while others use exit pupil coordinates. In most cases, there is not a significant difference, but in the case of systems containing cylindrical lenses, for example, there are major differences.

Another consideration relating to ray-intercept curves is the way in which vignetting is handled. This is coupled to the way the program handles apertures. As mentioned before, apertures have a special status in many optical design programs. Rays can be blocked by apertures, but this must be handled as a special case by the program, because there is nothing inherent in the ray-trace equations that prevents a blocked ray from being traced, in contrast to a ray that misses a surface or undergoes total internal reflection.

Even though a surface may have a blocking aperture, it may be desirable to let the ray trace proceed anyway. As mentioned before, blocking rays in optimization can produce instabilities that prevent convergence to a solution even though all the rays in the final solution are contained within the allowed apertures. Another situation where blocking can be a problem concerns central obstructions. In such systems, the reference ray may be blocked by an obstruction, so its data are not available to compute the displacement or optical path difference of an ordinary ray (which is not blocked). The programmer must anticipate such situations and build in the proper code to handle them.

In the case of ray-intercept curves, it is not unusual for programs to display data for rays that are actually blocked by apertures. The user is expected to know which rays get through, and ignore the others, a somewhat unreasonable expectation. The justification for allowing it is that the designer can see what would happen to the rays if the apertures were increased.

In addition to ray-intercept curves, optical design programs usually display field sag plots showing the locations of the tangential and sagittal foci as a function of field angle and distortion curves. In the case of distortion, there is the question of what to choose as a reference height. It is generally easiest to refer distortion to the paraxial chief ray height in the final image surface, but in many cases it is more meaningful to refer it to the centroid height of a bundle of exact rays from the same object point. Again, it is important for the user to know what the program is computing.

Spot-Diagram Analysis

Spot diagrams provide the basis for realistic modeling of optical systems in an optical design program. In contrast to simple ray-trace evaluation, which shows data from one or a few rays, spot

diagrams average data from hundreds or thousands of rays to evaluate the image of a point source. Notwithstanding this, it should be understood that the principal purpose of an optical design program is to design a system, not to simulate its performance. It is generally up to the designer to understand whether or not the evaluation model of a system is adequate to characterize its real performance, and the prudent designer will view unexpected results with suspicion.

From a programmer's point of view, the most difficult task in spot-diagram analysis is to accurately locate the aperture of the system. For systems that have rotational symmetry, this is not difficult, but for off-axis systems with vignetted apertures it can be a challenging exercise. However, the results of image evaluation routines are often critically dependent on effects that occur near the edges of apertures, so particular care must be paid to this problem in writing optical design software. Like many other aspects of an optical design program, there is a trade-off between efficiency and accuracy.

A spot diagram is an assemblage of data describing the image-space coordinates of a large number of rays traced from a single object point. The data may be either monochromatic or polychromatic. Each ray is assigned a weight proportional to the fractional energy that it carries. Usually, the data saved for each ray include its *xyz* coordinates on the image surface, the direction cosines *klm,* and the optical path length or optical path difference from the reference ray. The ray coordinates are treated statistically to calculate root-mean-square spot sizes. The optical path lengths yield a measure of the wavefront quality, expressed through its variance and peak-to-valley error.

To obtain a spot diagram, the entrance pupil must be divided into cells, usually of equal area. Although for many purposes the arrangement of the cells does not matter, for some computations (e.g., transfer functions) it is advantageous to have the cells arranged on a rectangular grid. To make the computations have the proper symmetry, the grid should be symmetrical about the *x* and *y* axis. The size of the grid cells determines the total number of rays in the spot diagram.

In computing spot diagrams, the same considerations concerning the reference point appear as for ray fans. That is, it is possible to define ray displacements with respect to the chief-ray, the paraxial ray height, or the centroid of the spot diagram. However, for spot diagrams it is most common to use the centroid as the reference point, both because many image evaluation computations require this definition, and also because the value for the centroid is readily available from the computed ray data.

$$a_i = \epsilon_{xi} = \text{ray displacement in the } x \text{ direction}$$
$$b_i = \epsilon_{yi} = \text{ray displacement in the } y \text{ direction}$$
$$c_i = k_i/m_i = \text{ray slope in the } x \text{ direction}$$
$$d_i = l_i/m_i = \text{ray slope in the } y \text{ direction}$$
$$w_i = \text{weight assigned to ray}$$

The displacements of rays on a plane shifted in the *z* direction from the nominal image plane by an amount Δz are given by

$$\delta x_i = a_i + b_i \Delta z$$

$$\delta y_i = c_i + d_i \Delta z$$

If there are *n* rays, the coordinates of the centroid of the spot diagram are

$$\delta \bar{x} = \frac{1}{W} \sum_{i=1}^{n} w_i \delta x_i = A + B \Delta z$$

$$\delta \bar{y} = \frac{1}{W} \sum_{i=1}^{n} w_i \delta y_i = C + D \Delta z$$

OPTICAL DESIGN SOFTWARE

where W is a normalizing constant that ensures that the total energy in the image adds up to 100 percent, and

$$A = \frac{1}{W}\sum_{i=1}^{n} w_i a_i$$

$$B = \frac{1}{W}\sum_{i=1}^{n} w_i b_i$$

$$C = \frac{1}{W}\sum_{i=1}^{n} w_i c_i$$

$$D = \frac{1}{W}\sum_{i=1}^{n} w_i d_i$$

The mean-square spot size can then be written as

$$\text{MSS} = \frac{1}{W}\sum_{i=1}^{n} w_i\{(\delta x_i - \delta\bar{x})^2 + (\delta y_i - \delta\bar{y})^2\}$$

Usually, the root-mean-square (rms) spot size, which is the square root of this quantity, is reported. Since the MSS has a quadratic form, it can be written explicitly as a function of the focus shift by

$$\text{MSS} = P + 2Q\,\Delta z + R(\Delta z)^2$$

where

$$P = \frac{1}{W}\sum_{i=1}^{n} w_i\{(a_i^2 + c_i^2) - (A^2 + C^2)\}$$

$$Q = \frac{1}{W}\sum_{i=1}^{n} w_i\{(a_i b_i + c_i d_i) - (AB + CD)\}$$

$$R = \frac{1}{W}\sum_{i=1}^{n} w_i\{(b_i^2 d_i^2) - (B^2 + D^2)\}$$

Differentiating this expression for the MSS with respect to focus shift, then setting the derivative to zero, determines the focus shift at which the rms spot size has its minimum value:

$$\Delta z_{\text{opt}} = -Q/R$$

Although the above equations determine the rms spot size in two dimensions, similar one-dimensional equations can be written for x and y separately, allowing the ready computation of the tangential and sagittal foci from spot-diagram data. In addition, it is straightforward to carry out the preceding type of analysis using optical path data, which leads to the determination of the center of the reference sphere that minimizes the variance of the wavefront.

Beyond the computation of the statistical rms spot size and the wavefront variance, most optical design programs include a variety of image evaluation routines that are based on spot diagram data. It is useful to characterize them as belonging to geometrical optics or physical optics, according to whether they are based on ray displacements or wavefronts, although, of course, all are based on the results of geometrical ray tracing.

Geometrical Optics Most optical design programs provide routines for computing radial diagrams and knife-edge scans. To compute a radial energy diagram, the spot-diagram data are sorted according to increasing ray displacement from the centroid of the spot. The fractional energy is then plotted as a function of spot radius. The knife-edge scan involves a similar computation, except that the spot-diagram data are sorted according to x or y coordinates, instead of total ray displacement.

Another type of geometrical image evaluation based on spot-diagram data is the so-called *geometrical optical transfer function* (GOTF). This function can be developed as the limiting case, as the wavelength approaches zero, of the actual diffraction MTF, or, alternately, in a more heuristic way as the Fourier transform of a line spread function found directly from spot-diagram ray displacements (see, for example, Smith's book[10]). From a programming standpoint, computation of the GOTF involves multiplying the ray displacements by 2π times the spatial frequency under consideration, forming cosine and sine terms, and summing over all the rays in the spot diagram. The computation is quick, flexible, and if there are more than a few waves of aberration, accurate. The results of the GOTF computation are typically shown as either plots of the magnitude of the GOTF as a function of frequency, or alternately in the form of what is called a "through-focus" MTF, in which the GOTF at a chosen frequency is plotted as a function of focus shift from the nominal image surface.

Physical Optics The principal physical optics calculations based on spot-diagram data are the modulation transfer function, sometimes called the "diffraction" MTF, and the point spread function (PSF). Both are based on the wavefront derived from the optical path length data in the spot diagram. There are various ways to compute the MTF and PSF, and not all programs use the same method. The PSF, for example, can be computed from the pupil function using the fast Fourier transform algorithm or, alternately, using direct evaluation of the Fraunhofer diffraction integral. The MTF can be computed either as the Fourier transfer of the PSF or, alternately, using the convolution of the pupil function.[11] The decision as to which method to use involves speed, accuracy, flexibility, and ease of coding.

In physical-optics-based image evaluation, accuracy can be a problem of substantial magnitude. In many optical design programs, diffraction-based computations are only accurate for systems in which diffraction plays an important role in limiting the performance. Systems that are limited primarily by geometrical aberrations are difficult to evaluate using physical optics, because the wavefront changes so much across the pupil that it may be difficult to sample it sufficiently using a reasonable number of rays. If the actual wavefront in the exit pupil is compared to a reference sphere, the resultant fringe spacing defines the size required for the spot diagram grid, since there must be several sample points per fringe to obtain accurate diffraction calculations. To obtain a small grid spacing, one can either trace many rays, or trace fewer rays but interpolate the resulting data to obtain intermediate data.

Diffraction calculations are necessarily restricted to one wavelength. To obtain polychromatic diffraction results it is necessary to repeat the calculations in each color, adding the results while keeping track of the phase shifts caused by the chromatic aberration.

3.5 OPTIMIZATION

The function of the optimization part of the program is to take a *starting design* and modify its construction so that it meets a given set of specifications. The starting design may be the result of a previous design task, a lens from the library, or a new design based on general optical principles and the designer's intuition.

OPTICAL DESIGN SOFTWARE

The performance of the design must be measured by a single number, often known in optics as the *merit function,* although the term *error function* is more descriptive and will be used here. The error function is the sum of squares of quantities called *operands* that characterize the desired attributes. Examples of typical operands include paraxial constants, aberration coefficients, and exact ray displacements. Sometimes, the operands are broken into two groups: those that must be satisfied exactly, which may be called *constraints,* and others that must be minimized. Examples of constraints might include paraxial conditions such as the focal length or numerical aperture.

The constructional parameters to be adjusted are called *variables,* which include lens curvatures, thicknesses, refractive indices, etc. Often the allowed values of the variables are restricted, either by requirements of physical reality (e.g., positive thickness) or the given specifications (e.g., lens diameters less than a prescribed value). These restrictions are called *boundary conditions,* and represent another form of constraint.

Usually, both the operands and constraints are nonlinear functions of the variables, so optical design involves nonlinear optimization with nonlinear constraints, the most difficult type of problem from a mathematical point of view. A great deal of work has been carried out to develop efficient, general methods to solve such problems. Detailed consideration of these methods is beyond the scope of this chapter, and the reader is referred to a paper by Hayford.[12]

In a typical optical design task, there are more operands than variables. This means that there is, in general, no solution that makes all of the operands equal to their target values. However, there is a well-defined solution called the *least-squares* solution, which is the state of the system for which the operands are collectively as close to their targets as is possible. This is the solution for which the error function is a minimum.

The Damped Least-Squares Method

Most optical design programs utilize some form of the *damped least-squares* (DLS) method, sometimes in combination with other techniques. DLS was introduced to optics in about 1960, so it has a history of 50 years of (usually) successful application. It is an example of what is known as a *downhill* optimizer, meaning that in a system with multiple minima, it is supposed to find the nearest local minimum. In practice, it sometimes suffers from *stagnation,* yielding slow convergence. On the other hand, many designers over the years have learned to manipulate the damping factor to overcome this deficiency, and even in some cases to find solutions beyond the local minimum.

We consider first the case of unconstrained optimization. Let the system have M operands f_i and N variables x_i. The error function ϕ is given by

$$\phi = f_1^2 + f_2^2 + \cdots + f_M^2$$

Define the following:

$$\mathbf{A} = \text{derivative matrix, } A_{ij} \equiv \frac{\partial f_i}{\partial x_j}$$

$$\mathbf{G} = \text{gradient vector, } G_k \equiv \frac{1}{2}\frac{\partial \phi}{\partial x_k}$$

$$\mathbf{x} = \text{change vector}$$

$$\mathbf{f} = \text{error vector}$$

With these definitions, we have

$$\mathbf{G} = \mathbf{A}^T\mathbf{f}$$

If we assume that the changes in the operands are linearly proportional to the changes in the variables, we have

$$\mathbf{f} = \mathbf{A}\mathbf{x} + \mathbf{f}_0$$

$$\mathbf{G} = \mathbf{A}^T\mathbf{A}\mathbf{x} + \mathbf{G}_0$$

At the solution point, the gradient vector is zero, since the error function is at a minimum. The change vector is thus

$$\mathbf{x} = -(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{G}_0$$

These are called the least-squares *normal equations,* and are the basis for linear least-squares analysis. When nonlinear effects are involved, repeated use of these equations to iterate to a minimum often leads to a diverging solution. To prevent such divergence, it is common to add another term to the error function, and this limits the magnitude of the change vector **x**. In the DLS method, this is accomplished by defining a new error function

$$\varphi = \phi + p\mathbf{x}^T\mathbf{x}$$

A key property of DLS is that the minimum of φ is the same as the minimum of ϕ since, at the minimum, the change vector **x** is zero. By differentiating and setting the derivative equal to zero at the minimum, we arrive at the damped least-squares equations

$$\mathbf{x} = -(\mathbf{A}^T\mathbf{A} + p\mathbf{I})^{-1}\mathbf{G}_0$$

which look like the normal equations with terms added along the diagonal. These terms provide the damping, and the factor p is called the *damping factor.* This particular choice of damping is called *additive damping* but, more generally, it is possible to add any terms to the diagonal and still maintain the same minimum. Some optical design programs multiply the diagonal elements of the $\mathbf{A}^T\mathbf{A}$ matrix by a damping factor, while others make them proportional to the second derivative terms. Although theoretical arguments are sometimes advanced to support the choice of a particular method of damping, in practice the choice of damping factor is an ad hoc way to accelerate convergence to a solution by limiting the magnitude (and changing the direction) of the change vector found from the normal equations.

In practical optical design work, it has been found that no single method for choosing the damping factor works best in all cases. In a particular problem, one method may be dramatically better than another, but in a different problem, the situation may be completely reversed. Every optical design program has its unique way of choosing the optimum damping, which makes each program different from the others, and gives it a raison d'être.

Although the principal use of the damping factor is to accelerate convergence by limiting the magnitude of the change vector, the damping factor has also been used routinely to increase the magnitude of the change vector to escape a local minimum. During the course of a minimization task, if the solution stagnates, or does not converge to what the designer believes to be an acceptable configuration, it may be possible to force the solution into another region by running one or more iterations with reduced damping in which the error function increases.

Constraints and Boundary Conditions There are two general methods used in optical design programs for handling constraints and boundary conditions. The first is to add a term (called a *penalty* function) to the error function that targets the constraint to its desired value. In the case of boundary violations, "one-sided" terms can be added, or special weighting functions can be constructed that increase in magnitude as a violation goes farther into a forbidden region. The other method augments

the number of equations by the number of constraints and solves the resulting equations using the Lagrange multiplier method. This produces a minimum that satisfies the constraints exactly and minimizes the remaining error function.

The penalty function method is more flexible and faster (since there are fewer equations) than the Lagrange multiplier method. On the other hand, the Lagrange multiplier method gives more precise control over the constraints. Both are commonly used in optical design software.

Other Methods

Although DLS is used in the vast majority of optical design applications, other methods are occasionally used,[12] and two warrant mention. These are *orthonormalization*, which has been used to overcome stagnation in some DLS problems, and *simulated annealing*, which has been used for global optimization.

Orthonormalization The technique of orthonormalization for the solution of optical design problems was introduced by Grey.[2] Although it solves the same problem as DLS does, it proceeds in a very different fashion. Instead of forming the least-squares normal equations, Grey works directly with the operand equations

$$\mathbf{Ax} = -\mathbf{f}$$

To understand Grey's method, it is best to forget about optics and consider the solution of these equations strictly from a mathematical point of view. The point of view that Grey uses is that **f** represents a vector in m-dimensional space. The columns of **A** can be regarded as basis vectors in this m-dimensional space. Since there are only n columns, the basis vectors do not span the space. The change vector **x** represents a projection of **f** on the basis vectors defined by **A**. At the solution point, the residual part of **f** will be orthogonal to its projection on the basis vectors.

In Grey's orthonormalization method, the solution of the equations is found by a technique similar to Gram-Schmidt orthogonalization, but during the solution process, the actual error function is evaluated several times in an effort to use the best variables to maximum advantage. Because of this, the method is computationally intensive compared to DLS. However, the extra computation is justified by a more accurate solution. The common wisdom is that orthogonalization is superior to DLS near a solution point, and inferior to DLS when the solution is far removed from the starting point.

Simulated Annealing Simulated annealing has been applied to optical design optimization, chiefly in problems where the task is to find a global minimum. The method varies drastically from other techniques. It makes no use of derivative information, and takes random steps to form trial solutions. If a trial solution has a lower error function than the current system, the new system replaces the old. If a trial solution has a higher error function than the current system, it may be accepted, depending on how much worse it is. The probability of acceptance is taken to be $\exp(-\Delta\phi/T)$, where T is an experimentally determined quantity. In general simulated annealing, T is provided by the user. In adaptive simulated annealing, T is reduced automatically according to algorithms that hold the system near statistical equilibrium.

Error Functions

Obviously, the choice of an error function has a major impact on the success of an optical design task. There are a number of requirements that an error function should meet. Most importantly, the error function should accurately characterize the desired properties of the system under design. There is little chance of success if the program is optimizing the wrong thing. Yet this is an area of great difficulty in computer-aided optical design, because it is at odds with efficiency. In order to obtain more accuracy, more extensive computations should be carried out, but this takes time.

There are two schools of thought concerning the implementation of error functions in optical design programs. The first holds that the designer should have complete control over the items included in the error function, while the second holds that the program itself should set up the basic error function, allowing the designer some degree of control through weighting functions. Neither school has demonstrated superiority, but the approach to error function construction taken by various optical design programs accounts for user allegiances that are sometimes remarkably strong.

The different ways that optical design programs handle error functions makes it difficult to discuss the topic here in anything other than broad detail. At one extreme are programs that provide practically no capability for the user to insert operands, displaying only the value of the overall error function, while at the other extreme are programs that make the user enter every operand individually. Regardless of the user interface, however, there are some general concepts that are universally relevant.

Error functions can be based on either aberration coefficients or exact-ray data (or both). In the early stages of design, aberration coefficients are sometimes favored because they provide insight into the nature of the design, and do not suffer ray failures. However, the accuracy of aberration coefficients for evaluating complex systems is not very good, and exact-ray data are used in virtually all final optimization work.

So far as exact-ray error functions are concerned, there is the question of whether to use ray displacements or optical path difference (or both). This is a matter of user (or programmer) preference. The use of ray displacements leads to minimizing geometrical spot sizes, while the use of optical path difference leads to minimizing the wavefront variance.

For exact-ray error functions, a suitable pattern of rays must be set up. This is often called a *ray set.* There are three common methods for setting up a ray set. The first is to allow the designer to specify the coordinates (object, pupil, wavelength, etc.) for a desired set of rays. This gives great flexibility, but demands considerable skill from the user to ensure that the resulting error function accurately characterizes performance.

The other two methods for setting up ray sets are more automatic. The first is to allow the user to specify object points, and have the program define a rectangular grid of rays in the aperture for each point. The second uses a Gaussian integration scheme proposed by Forbes to compute the rms spot size, averaged over field, aperture, and wavelength.[13] The Forbes method, which is restricted to systems having plane symmetry, leads to dividing the aperture into *rings* and *spokes.* For systems having circular pupils, the Forbes method has both superior accuracy and efficiency, but for vignetted pupils, there is little difference between the two.

Multiconfiguration Optimization

Multiconfiguration optimization refers to a process in which several systems having some common elements are optimized jointly, so that none of the individual systems but the *ensemble* of all of the systems is optimized. The archetype of multiconfiguration systems is the *zoom* system in which the focal length is changed by changing the separation between certain elements. The system is optimized simultaneously at high, medium, and low magnifications to produce the best overall performance.

Most of the larger optical design programs have the capability to carry out multiconfiguration optimization, and this capability is probably used more for non-zoom systems than for zoom systems. A common use of this feature is to optimize a focal system for through-focus performance in order to minimize sensitivity to image plane shifts. In fact, multiconfiguration optimization is used routinely to control tolerances.

Tolerancing

Beyond the task of desensitizing a given design, considerations of manufacturing tolerances become increasingly important as the complexity of optical designs increases. It is quite easy to

design optical systems that cannot be built because the fabrication tolerances are beyond the capability of optical manufacturing technology. In any case, specifying tolerances is an integral part of optical design, and a design project cannot be considered finished until appropriate tolerances are established.

Tolerancing is closely related to optimization. The basic tolerance computation is to calculate how much the error function changes for a small change in a construction parameter, which is the same type of computation carried out when computing a derivative matrix. Even more relevant, however, is the use of *compensators,* which requires reoptimization. A compensator is a construction parameter that can be adjusted to compensate for an error introduced by another construction parameter. For example, a typical compensator would be the image distance, which could be adjusted to compensate for power changes introduced by curvature errors.

There is considerable variation in how different optical design programs handle tolerancing. Some use the reoptimization method described here, while others use Monte Carlo techniques. Some stress interaction with the designer, while others use defaults for more automatic operation.

3.6 OTHER TOPICS

Of course, many other topics would be included in a full discussion of optical design software. Space limitations and our intended purpose prevents any detailed consideration, but a few of the areas where there is still considerable interest are the following.

Simulation

There is increased interest in using optical design programs to simulate the performance of actual systems. The goal is often to be able to calculate radiometric throughput of a system used in conjunction with a real extended source. It is difficult to provide software to do this with much generality, because brute force methods are very inefficient and hard to specify, while elegant methods tend to have restricted scope, and demand good judgment by the person modeling the physical situation. Nevertheless, with the increase in the speed of computers, there is bound to be an increasing use of optical design software for evaluating real systems.

Global Optimization

After several years during which there was little interest in optimization methodology, the tremendous increase in the speed of new computers has spawned a renewal of efforts to find global, rather than local, solutions to optical design problems. Global optimization is a much more difficult problem than local optimization. In the absence of an analytic solution, one never knows whether a global optimum has been achieved. All solution criteria must specify a region of interest and a time limit, and the method cannot depend on the starting point. The simulated annealing method described above is one area of continuing interest. Several methods for what might be called *pseudo-global* optimization have been used in commercial optical design programs, combining DLS with algorithms that allow the solution to move away from the current local minimum.

Computing Environment

Increasingly, optical design programs are used in conjunction with other software. Drawing programs, manufacturing inventory software, and intelligent databases are all relevant to optical design. While the conventional optical design program has been a *stand-alone* application, there is increasing demand for integrating optical design into more general design tasks.

3.7 BUYING OPTICAL DESIGN SOFTWARE

The complexity of the optical design process, together with the breadth of applications of optics, has created an ongoing market for commercial optical design software. For people new to optical design, however, the abundance of advertisements, feature lists, and even technical data sheets doesn't make purchasing decisions easy. The following commentary, adapted from an article, may be helpful in selecting an optical design program.[14] It considers five key factors: hardware, features, user interface, cost, and support.

Hardware

It used to be that the choice of an optical design program was governed by the computer hardware available to the designer. Of course, when the hardware cost was many times higher than the software cost, this made a great deal of sense. Today, however, the software often costs more than the hardware, and many programs can be run on several different computer platforms, so the choice of computer hardware is less important. The hardware currently used for optical design is principally IBM-PC compatible.

To run optical design software, the fastest computer that can be obtained easily is recommended. The iterative nature of optical design makes the process interminable. There is a rule, sometimes called the Hyde maxim, that states that an optical design is finished when the time or money runs out. Notwithstanding this, the speed of computers has ceased to be a significant impediment to ordinary optical design. Even low-cost computers now trace more than 1000 ray-surfaces/s, a speed considered the minimum for ordinary design work, and create the potential for solving new types of problems formerly beyond the range of optical design software.

Before desktop computers, optical design software was usually run on time-shared central computers accessed by terminals, and some programs are still in that mode. There seems to be general agreement, however, that the memory-mapped display found on PCs provide a superior working environment and dedicated desktop computer systems are currently most popular.

Features

If you need a particular feature to carry out your optical design task, then it is obviously important that your optical design program have that feature. But using the number of features as a way to select an optical design program is probably a mistake. There are more important factors, such as cost, ease of use, and scope. Moreover, you might assume that all the features listed for a program work simultaneously, which may not be true. For example, if a vendor states that its program handles holograms and toric surfaces, you might assume that you can work with holographic toroids, but this may not be true.

The continuing growth of optics and the power of desktop computers has put heavy demands on software vendors to keep up with the development of new technology. Moreover, since the customer base is small and most vendors now support the same computer hardware, the market has become highly competitive. These factors have led to a "feature" contest in which software suppliers vie to outdo each other. While this is generally good for the consumer, the introduction of a highly visible new feature can overshadow an equally important but less obvious improvement (for example, fewer bugs or better documentation). In addition, the presence of a number of extra features is no guarantee that the underlying program is structurally sound.

User Interface

There is very little in common between the user interfaces used by various optical design programs. Each seems to have its own personality. The older programs, originally designed to run in batch

mode on a large computer, are usually less interactive than ones that were written specifically for desktop computers. Batch programs tend to be built more around default actions than interactive programs, which require more user input. It would be hard to put any of today's major optical design programs in a box classified as either batch or interactive, but the look and feel of a program has a strong influence on its usefulness.

Many people don't realize that the most important benefit of using an optical design program is often the understanding that it provides the user about how a particular design works. It's often tempting to think that if the computer could just come up with a satisfactory solution, the design would be finished. In practice, it is important to know the trade-offs that are made during a design project. This is where the judgment of the optical engineer comes in, knowing whether to make changes in mechanical or electrical specifications to achieve the optimum balance in the overall system. Lens designers often say that the easiest lens to design is one that has to be diffraction-limited, because it is clear when to stop. If the question of how to fit the optics together with other system components is important, then the ability of the user to work interactively with the design program can be a big help.

Cost

In today's market, there is a wide range of prices for optical design software. This can be very confusing for the first-time buyer, who often can't see much difference in the specifications. The pricing of optical design software is influenced by (at least) three factors.

First, the range of tasks that can be carried out using an optical design program is enormous. The difference in complexity between the job of designing a singlet lens for a simple camera, and that of designing a contemporary objective for a microlithographic masking camera is somewhat akin to the difference between a firecracker and a hydrogen bomb.

Second, all software is governed by the factors originally studied in F. P. Brooks' famous essay *The Mythical Man-Month.*[15] Brooks was director of the group that developed the operating system for the IBM 360, a mainframe computer introduced in the 1960s. Despite its provocative title, Brooks' essay is a serious work that has become a standard reference for software developers. In it, he notes that if the task of developing a program to be used on a single computer by its author has a difficulty of 1, then the overall difficulty of producing integrated software written by a group of people and usable by anyone on a wide range of computers may be as high as 10. In recent years, the scope of the major optical design programs has grown too big for a single programmer to develop and maintain, which raises costs.

Third, there are structural differences in the way optical design software is sold. The original mainframe programs were rented, not sold. If the user did not want to continue monthly payments, the software had to be returned. PC programs, on the other hand, are usually sold with a one-time fee. In the optical design software business, several vendors offer a compromise policy, combining a permanent license with an optional ongoing support fee.

It would be nice if the buyer could feel comfortable that "you get what you pay for," but unfortunately this view is too simplistic. One program may lack essential capabilities, another may contain several unnecessary features when evaluated for a particular installation. Buying on the basis of cost, like features, is probably not a good idea.

Support

Support is an important aspect to consider in selecting an optical design program, and it is often difficult to know what is included in support. Minimal support consists of fixing outright bugs in the program. More commonly, support includes software updates and phone or email assistance in working around problems.

Optical design programs are typically not bug-free. Unlike simple programs like word processors, optimization programs cannot be fully tested, because they generate their own data. One result of this is that software vendors are generally reluctant to offer any warranty beyond a "best-effort"

attempt to fix reported problems. Unfortunately, there is no good way for buyers to know whether and when their particular problems may be fixed; the best approach is probably to assess the track record of the vendor by talking to other users.

Coupled with support is user training. Although it should be possible to use a program by studying the documentation, the major optical design software vendors offer regular seminars, often covering not only the mechanics of using their program, but also general instruction in optical design. For new users, this can be a valuable experience.

3.8 SUMMARY

As stated in the introduction, this chapter is intended as a survey for readers who are not regular users of optical design software. The form of an optical design program described here, consisting of lens entry, evaluation, and optimization sections, is used in many different programs. There has been little standardization in this field, so the "look and feel," performance features and extent of various programs are quite different. Nonetheless, it is hoped that with a knowledge of the basic features described here, the reader will be in a good position to judge whether an optical design program is of use, and to make an informed decision about whether one particular program is better than another.

3.9 REFERENCES

1. D. P. Feder, "Automatic Optical Design," *Appl. Opt.* **2**:1209–1226 (1963).
2. D. S. Grey, "Aberration Theories for Semiautomatic Lens Design by Electronic Computers," *J. Opt. Soc. Am.* **53**:672–680 (1963).
3. G. H. Spencer, "A Flexible Automatic Lens Correction Procedure," *Appl. Opt.* **2**:1257–1264 (1963).
4. C. G. Wynne and P. Wormell, "Lens Design by Computer," *Appl. Opt.* **2**:1233–1238 (1963).
5. See, for example, *The Photonic Industry Buyer's Guide,* Laurin Publishing, Pittsfield, MA 01202.
6. *U.S. Military Handbook for Optical Design,* republished by Sinclair Optics, Fairport, NY 14450 (1987).
7. G. H. Spencer and M. V. R. K. Murty, "Generalized Ray-Tracing Procedure," *J. Opt. Soc. Am.* **52**:672–678 (1962).
8. D. C. O'Shea, *Elements of Modern Optical Design,* John Wiley & Sons, New York (1985).
9. H. A. Buchdahl, *Optical Aberration Coefficients,* Oxford Press, London (1954).
10. W. J. Smith, *Modern Optical Engineering,* McGraw-Hill, New York (1990).
11. H. H. Hopkins, "Numerical Evaluation of the Frequency Response of Optical Systems," *Proc. Phys. Soc. B* **70**:1002–1005 (1957).
12. M. J. Hayford, "Optimization Methodology," *Proc. SPIE* **531:** 68–81 (1985).
13. G. W. Forbes, "Optical System Assessment for Design: Numerical Ray Tracing in the Gaussian Pupil," *J. Opt. Soc. Am. A* **5**:1943–1956 (1988).
14. D. C. Sinclair, "Optical Design Software: What to Look For in a Program," *Photonics Spectra,* Nov. 1991.
15. F. P. Brooks, Jr., *The Mythical Man-Month,* Addison-Wesley, Reading, MA (1975).