An Optical Toolbox for Astronomical Instrumentation

Brian M. Sutin, TMT Observatory Corporation

ABSTRACT

The author has open-sourced a program for optical modeling of astronomical instrumentation. The code allows for optical systems to be described in a programming language. An optical prescription may contain coordinate systems and transformations, arbitrary polynomial aspheric surfaces and complex volumes. Rather than using a plethora of rays to evaluate performance, all the derivatives along a ray are computed by automatic differentiation. By adaptively controlling the patches around each ray, the system can be modeled to a guaranteed known precision. The code currently consists of less than 10,000 lines of C++/stdlib code.

keywords: optical design, ray tracing, software tools

1. INTRODUCTION

The state of the art in professional optical raytracing has not changed significantly in more than 40 years. For example CODE V[1] is still one of the best available codes for optical design. This is despite the fact that optical journals every year have many papers on raytracing. The *skewray* optical raytracing code was created in order to test out various random ideas that might turn out to be innovations. The unfortunate side effect of this philosophy is that the resulting code is strange to use, with a steep learning curve. This may limit the resulting popularity for general usage in the design community, but popularity is not an objective.

The code itself is written in C++03 and uses the C++ Standard Library extensively. Flex & bison are the only other tools used for code generation. The code is loosely based on a previous, similar raytrace program[9,14] that was used from 1994 to 2003 at UCO/Lick Observatory and Carnegie Observatories. That code was oriented towards analysis by massive numbers of rays, and so optimized for speed. For *skewray*, no effort is spent on speed, but rather on algorithms that do not require large numbers of rays.

The previous code also had a complex, hierarchical database system. As an example, an optical glass index might be called out as “Materials/Glass/Ohara/i-line/BSL7Y/Melt/J02743/index.” Since hierarchical databases are a great way to hide data, and many people do not think hierarchically, *skewray*'s database only has two levels, objects and object properties.

This paper mainly consists of brief descriptions of various ideas that are being worked. Not all of these ideas have been implemented in code yet. A status is given at the end of the paper.

2. OPEN SOURCE

All of the *skewray* code is open sourced on Github[3]. The code is released under the GPLv3 license. This is perhaps the most important innovation implemented; most non-commercialized optical raytrace codes have disappeared at the retirement of their authors (eg, KDP[4]), or may to do so in the future (eg, OARSA[2]). Various unit tests are available so that users can confirm that the code runs correctly without trying to read opaque and confusing source code. A manual and various design file examples and unit tests are available on the Github site.

3. SYMBOLIC

Describing an optical system to an optical code can be a very frustrating user experience. For example, Zemax OpticStudio[5] software uses a spreadsheet format as the input method. Although convenient for simple models, complex designs quickly get out of hand. The approach that *skewray* takes is to use a high-level computer language to describe the optical system. “High-level” here means similar to, say, MATLAB[6]. The *skewray* language has the following features:

- Variable names can be symbolic; e.g., 'θ' can be used for angles.
- Comments and variables are internationalized, so users can comment in their native language. UTF-8, now the standard encoding for the web, is used for portability.
- Variables hold high-level objects such as coordinate systems, coordinate transformations, optical rays, surfaces, volumes, and so on.
- Counter-intuitive integers are not included; "X = 1/2" will result in X being 0.5, not zero.
- All math is done in the highest precision available, long double precision. Depending on compiler flags, this can be anywhere from 64 to 128 bits of precision.
- Almost all of the useful POSIX math functions are available, including the more esoteric functions such as the Bessel functions (eg, Airy disks).
- A macro system is used to implement system and user-defined functions, as well as multi-dimensional arrays.

The following is a code snippet showing what an input file looks like

```
#  Ref 1, eqn (9): compute β, scale height of atmosphere H_o/r_o
env.β = 0.001254 * env.Tk / Kelvin(0)
#  Ref 1, eqn (10): compute kappa, or inverse gravitational constant g_o/g
env.κ = 1 + 0.005302 * sind(env.ϕ)^2 - 0.00000583 * sind(2*env.ϕ)^2 - 0.000000315 * env.H
```

This example is taken from an input file that computes atmospheric dispersion. A nice advantage of keeping design as source code is that version control systems can be used to maintain the history and design evolution. Modern version control systems allow maintaining parts of a design separately, without adding any overhead to *skewray* code.

4. OVERLOADED DERIVATIVES

All floating point variables are considered as independent variables, and all subsequent computations depending on a variable will also carry along the first derivatives with respect to that variable. This includes any object that depends on that variable, such as coordinate systems, rays, surfaces, and the more complex results of ray tracing, such as wavefront error integrated over an aperture. If the user wants to know how sensitive anything is to an input, the answer is trivially available. These first derivatives are available as symbolic inputs to consequent computations.

As an example, consider a light ray which is propagated through a system to some new location. All the parameters of the ray are overloaded such that all of the first derivatives are computed automatically. If a ray described by $\{q_i,p_i\}$ is traced, then the complete list of information available after tracing a ray is

$$\begin{array}{cccc} t & \left.\frac{\partial t}{\partial q_i}\right|_{p_i,\alpha} & \left.\frac{\partial t}{\partial p_i}\right|_{q_i,\alpha} & \left.\frac{\partial t}{\partial \alpha}\right|_{q_i,p_i} \\ q_o & \left.\frac{\partial q_o}{\partial q_i}\right|_{p_i,\alpha} & \left.\frac{\partial q_o}{\partial p_i}\right|_{q_i,\alpha} & \left.\frac{\partial q_o}{\partial \alpha}\right|_{q_i,p_i} \\ p_o & \left.\frac{\partial p_o}{\partial q_i}\right|_{p_i,\alpha} & \left.\frac{\partial p_o}{\partial p_i}\right|_{q_i,\alpha} & \left.\frac{\partial p_o}{\partial \alpha}\right|_{q_i,p_i} \end{array}. \tag{1}$$

The α represents any other independent variables that the ray might depend on, such as wavelength, surface curvatures, &c.

If the final time value t does not depend on $\{q_i,p_i\}$, then the matrix of $\{q_o,p_o\}$ partial derivatives for a conservative system will have only 18 degrees of freedom, rather than 36 (see Generating Functions below). This interdependence of the derivatives can be used to confirm that ray tracing is obeying physical principles, as well as confirm computational accuracy.

5. TRANSFORMATIONS & COORDINATES

Even for very simple optical systems, multiple coordinate systems can be useful. Not only do optical elements need to be moved into position, but sometimes an analysis needs to be done at an entrance or exit pupil, or an analysis depends on the locations of the elements. Telescopes are full of moving parts, and these coordinate systems can be dependent on variables as well.

A relatively complete set of operations are available for handling of transformations and coordinate systems. These include translations, rotations, multiplication, inverse, division both on the right and on the left, and rotation about an arbitrary axis. For example, a code snippet from model of TMT:

```
%number θ = 2 * atand( cosd(ζ) * cosd(IEA) * sind(IBA) + sind(ζ) * sind(IEA), cosd(IEA) * cosd(IBA) )
%number Φ = acosd( -sind(ζ) * cosd(IEA) * sind(IBA) + cosd(ζ) * sind(IEA) ) / 2
%coordinate M1CRS = ECRS * zmov(-D_EL_M1)                          # Primary Mirror vertex
%coordinate M2CRS = M1CRS * zmov(D_M2_M1) * xrot(180)              # Secondary Mirror
%coordinate M3CRS = ECRS * zrot(90+θ) * xrot(Φ)                    # Tertiary Mirror
```

6. SURFACES & VOLUMES

Many ways exist to define optical surfaces[10]. The approach *skewray* takes is to define the shape of a surface as a function over $\mathbb{R}^3$. The actual surface consists of the points where the function is zero. The shape of a surface 'S' is, in set notation, $\{r \in \mathbb{R}^3 \,|\, S(r) = 0\}$. For example, $\{x^2 + y^2 + z^2 - 1 = 0\}$ is the unit sphere. The most common surfaces are supplied as macros.

Any surface shape that is polynomial in x, y, and z can be modeled. This may seem like a significant limitation, but the vast majority of surface shapes used in optics are polynomial. These include planes, spheres, paraboloids, hyperboloids, ellipsoids, cones, cylinders, toroids, parabolic concentrators, Cartesian ovals, standard aspheres, Zernike polynomials, and so on.

Note that the above paragraphs describe the shapes of surfaces, not the actual surface itself. If surfaces were infinitely thin 2-dimensional manifolds, then a point described by a floating point number will almost always not be exactly on the surface. Instead, surfaces in *skewray* are fuzzy or spongy; a point associated with a ray is on a surface if the point is within some fraction of the ray wavelength. The wavelength fraction is chosen by the user.

A volume is defined as the point in space where a surface is greater than zero. Complex volumes can be built up by boolean logic. Shapes of almost arbitrary complexity can be built up this way. This is analogous to the way that the 3D modeling program OpenSCAD[7] works.

Fuzzy surfaces add an interesting aspect to determining if a ray in inside a volume. A ray near the boundary surface of a volume might or might not be inside the volume, so trinary logic ('true', 'false', and 'maybe') is required.

7. EULERIAN

Fluid mechanics has two ways of describing how simulations are done, Eulerian and Lagrangian. In Eulerian, the equations are written with the walls stationary and the fluid flowing through a grid. In Lagrangian, the equations move with the fluid, and the walls move past at the fluid velocity. Smoothed Particle Hydrodynamics (SPH) is an example of a Lagrangian implementation.

The same analogy can be used for ray tracing. 'Sequential' ray tracing is often done using the Lagrangian paradigm, where the the surfaces are always at the origin and the rays are translated and rotated to approach each surface. This is done this way because otherwise complex aspheric surfaces would need to be transformed to arbitrary locations in space. 'Non-sequential' ray tracing is done using the Eulerian paradigm, where the “next” surface is not clear until the ray has been traced to all contending surfaces. Surfaces are often approximated by a mesh, NURBS, or similar.

The approach taken by *skewray* is to use the Eulerian paradigm for sequential and non-sequential ray tracing. This is done by transforming surfaces to their true location, since a linear coordinate transformation of a polynomial is another polynomial.

8. APERTURES & STOPS

Since *skewray* is not built on the idea of a paraxial system, the difference between a stop and an aperture is not entirely clear. We shall use 'stop' for something that prevents a ray from proceeding, at the place where it occurs, and use 'aperture' for a stop or the image of a stop projected by optics in the system. The kinds of stops that we consider are

- physical stops; eg, lens and mirror edges
- negative spacings such as feathered edges on lenses
- missing a surface
- total internal reflection boundary
- non-existent grating orders

Because derivatives are carried during ray tracing, all of these stops can be linearly (or quadratically) approximated when a ray passes nearby. Since the ray carries derivatives, the linear transformation from any place to another in the optical system is available as well. Consequently, every ray through the system has along with it a complete set of apertures corresponding to every possible way the ray might be stopped. These apertures are then used as the boundaries of integrals over bundles of rays.

9. GENERATING FUNCTIONS

As long as the physics used to trace rays is somewhat realistic and does not include anything dissipative (such as scatter), then the system is Hamiltonian. What that means in practice is that the transformation that takes a ray from one part of an optical system to another is a symplectic or canonical transformation. These transformations are special in that each transformation has a generating function, usually called a characteristic function or eikonal in optics. The upshot of all these buzzwords is merely the existence of generating functions.

A generating function is a function of half of the input variables $\{q_i,p_i\}$ and half of the output variables $\{q_o,p_o\}$. If the variables are chosen to be the input and output ray positions, then then a quadratic approximation to the exact generating function would be

$$G(q_i,q_o)=G_{const}+G_i\cdot q_i+G_o\cdot q_o+q_i\,G_{ii}\,q_i+q_i\,G_{io}\,q_o+q_o\,G_{oo}\,q_o. \tag{2}$$

Here G_i and G_o are row vectors and G_{ii}, G_{oo}, and G_{io} are matrices. Since *skewray* computes all of the first-order derivatives, tracing a single ray gives enough information to compute a generating function out to second order, or to first order along with derivatives with respect to independent variables. The generating function is then a description of the optical system near the ray. Any combination of input and output variables may be chosen, but it so happens that if the two spatial points are chosen that are not conjugate to each other, then the generating function is the optical path length. The input and output ray angles can be found by taking the partial derivatives,

$$\frac{\partial G}{\partial q_i}=p_i \quad \frac{\partial G}{\partial q_o}=-p_o. \tag{3}$$

The Seidel aberrations are nothing more than the third-order terms of the polynomial expansion of G, assuming that the system is axially symmetric. The Seidel aberrations can be computed by using the independent variables $u=q\cdot q$, $v=p\cdot p$, and $w=q\cdot p$ rather than q and p, and then using a generating function with a quadratic approximation in u, v, and w.

From the generating function, quantities such as RMS wavefront error (WFE) and RMS spot size can be computed analytically. For diffraction-limited applications, RMS wavefront error is a reasonable measure of performance. RMS spot size, however, is not guaranteed to converge (eg, diffraction) and is not a particularly good measure of seeing-limited performance. Better are integrals over the Point Spread Function (PSF), such as Point Source Sensitivity (PSS)[12] or Equivalent Noise Area (ENA)[13]. Since the final spot at the image surface can be folded over, computing the PSF becomes a topological problem. Creating an algorithm to compute the PSF from a generating function is possible, but not trivial.

Generating functions cannot be used for all optical raytrace problems, Scattering, for example, is a non-conservative process, so the generating function approach is not applicable. However, overloaded derivatives can be used to accelerate the modeling of scattering. An outline of such an approach is given in reference [11].

10. TESSELLATION AND GUARANTEED PERFORMANCE

When the user wants to know something about the system such as wavefront error or spot size, having the value to a guaranteed precision is very useful. For a very crude system which may be evaluated repeatedly by an optimization routine, very crude precision is useful in order to avoid wasting computational effort. For a system being evaluated for performance, the program should dynamically choose the number of rays used in order to achieve the desired precision.

This precision can be obtained by tessellation. The space of independent variables (aperture coordinates, wavelength, &c) is divided up into smaller and smaller regions. For each region, an estimate is made of the error, and if this error is too large, the region is broken up. Since tessellation in multiple dimensions is already confusing enough, *skewray* forces the regions to be n-rectangles. N-triangles (eg, 3-tetrahedrons) are more efficient, but considerably more complex to tessellate and integrate over.

When evaluating integrals over generating functions (eg, WFE, spot size), the first-order generating function gives integrals with derivatives, while the second-order generating function gives a more accurate estimate of the integral value, as well as an estimate of the precision being obtained by the first-order integral.

The same tessellation algorithm can be used to generate graphical output, either to the screen or to an external CAD program. Surfaces are broken up into smaller and smaller regions until the surfaces appear smooth to the eye.

11. OPTIMIZATION

Optimization is one of the most important aspects of optical design software. Designers want a magic button that finds the best possible design, given various free parameters. There are several reasons why this approach may not be the best. First, the "No Free Lunch" theorems[8] state roughly that all optimization algorithms are mediocre over all problems. So a particular algorithm might be good for most optical design problems, but there is no guarantee that some design problem outside of the typical will be optimizable. Second, having a requirement that a particular design be the best possible is not verifiable; proving that a design is optimal is probably more difficult than finding the optimum in the first place. Third, a requirement that a design be optimal is not a SMART goal; a designer tasked with finding the best solution has no stopping criteria.

Better is to design to a set of specific, verifiable requirements. For a set of requirements $\{R_m\}$, functions of the independent variables V_n, The requirements are

$$\begin{aligned} R_0(V_0, V_1 \ldots V_N) &\le 0 \\ R_1(V_0, V_1 \ldots V_N) &\le 0 \\ &\ldots \\ R_M(V_0, V_1 \ldots V_N) &\le 0 . \end{aligned} \tag{4}$$

So rather than require that, say, the RMS spot size be small, we require that the RMS spot size be smaller than some reasonable value, such as two or three pixels.

We could use these requirements to form a "merit" function (really a "demerit" function - big is bad), by forming a single function out of weighted least squares,

$$M(V) = \sum_{m|R_m>0} W_m R_m^2 . \tag{5}$$

The issue with this concept is that the demerit function is both overly restrictive and hides the structure of the problem. Which of the R_m requirements are not being satisfied? The set of conditions $\{R_m\}$ define a volume of the solution space

that meet requirements. The desired outcome of the process is not to find the best possible solution, but to find a solution that meets all of the requirements by being inside the specified volume.

The proposed optimization algorithm is a combination of a simplex method and Levenberg-Marquardt. First, consider the information gleaned from a single sample point,

$$\begin{array}{cccccc} R_0 & \frac{\partial R_0}{\partial V_0} & \frac{\partial R_0}{\partial V_1} & \cdots & \frac{\partial R_0}{\partial V_N} \\ R_1 & \frac{\partial R_1}{\partial V_0} & \frac{\partial R_1}{\partial V_1} & \cdots & \frac{\partial R_1}{\partial V_N} \\ & & \cdots & & \\ R_M & \frac{\partial R_M}{\partial V_0} & \frac{\partial R_M}{\partial V_1} & \cdots & \frac{\partial R_M}{\partial V_N}. \end{array} \tag{6}$$

Enough information exists from this single sample point to create a trivial solution-finding algorithm. Form the partial derivatives into an $M \times N$ matrix, find a Penrose inverse (eg, singular value decomposition), and use this to do a Newton-Raphson zero-find,

$$V_{new} = V_{old} - \alpha \left(\frac{\partial R}{\partial V} \right)^{-1} R \qquad \text{for } \{ m | R_m > 0 \}, \alpha < 1 . \tag{7}$$

This method is almost guaranteed to fail, but it does show the power of overloaded derivatives. The linear approximation can be graphically imagined by considering Figure 1, where each requirement, once linearized, is a half-plane, and the solution is a region where all the half-plans overlap. An improvement is to use quadratic approximations to the requirements, in which case the requirements appear as ellipsoids, as in Figure 2.

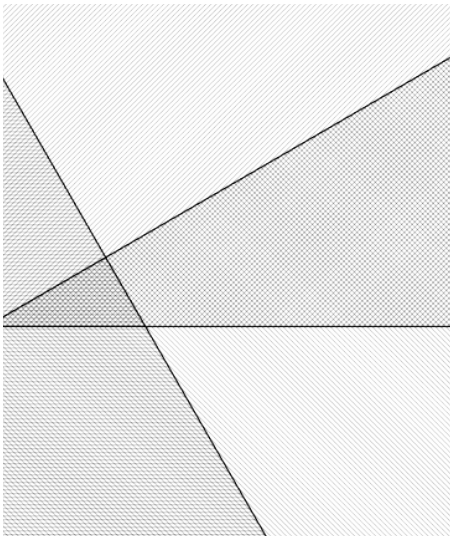

Figure 1: 3 linearized constraints, shown in 2-D space

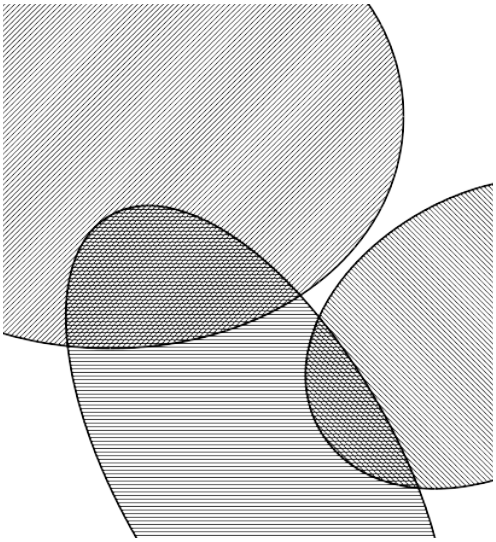

Figure 2: 3 quadratic constraints, shown in 2-D space

The second derivatives of the $\{R_m\}$, needed to make a quadratic fit, can be approximated given a sufficient number of sample points, at least M/2. Positive-definite (or negative-definite) second derivative matrices create a N-ellipsoids in $\mathbb{R}^N$, and finding the intersections of ellipsoids is relatively easy, while non-definite matrices make hyperboloids that are difficult to work with. The algorithm is then as follows:

Step 1: Use the method of equation (7) to find a set of initial sample points (the simplex).

Step 2: Use the sample points with derivatives to approximate a quadratic fit to each $\{R_m\}$ (least squares).

Step 3: Add a constant diagonal term to any non-positive-definite (or non-negative definite) quadratic R_m fit (Levenberg-Marquardt).

Step 4: For each sample point, create an M-digit binary number, where the m^{th} bit is 0 if the m^{th} requirement is met (1 for a negative-definite ellipsoid).

Step 5: Consider each R_m as an N-ellipsoid. New sample points are proposed by looking for intersections between the ellipsoids of simplex points with one bit set to zero.

Step 6: Sample points with a large number of bits or duplicate binary numbers can be discarded from the simplex.

Step 7: Re-adjust any sample points having binary digits that changed due to derivatives depending on a changing simplex.

Step 8: If no binary number is zero, go to Step 2.

A nice aspect of this method is that conflicting requirements are highlighted, which is something not necessarily evident in standard optimization methods. The advantage of this algorithm is not that it is a better optimizer, but that the trail of bread crumbs left behind after a failure to find a solution are very informative as to why, and can lead the designer to a better topology for the design in question.

12. STATUS

Major Task	**Status**
open source	done
high-level language for describing physical models	done
internationalization - native language variable names and comments	done
overloaded derivatives	done
arbitrary coordinate systems and transformations	done
arbitrary polynomial surfaces	done
sequential and non-sequential ray tracing	in progress
tracing with guaranteed tolerance	in progress
tracking of all apertures	not started
requirement satisfaction, rather than optimization	not started

13. REFERENCES

[1] Synopsys, Inc., CODE V, <https://optics.synopsys.com/codev/>, (2016).
[2] Epps, H. W., OARSA, (private communication).
[3] Sutin, B. M., Skewray, <https://github.com/skewray/skewray>, (2016).
[4] Klein, J. E., KDP, <http://www.ecalculations.com/>, (2016).
[5] Zemax, LLC, Zemax OpticSudio, <http://zemax.com/os/opticstudio>, (2016).
[6] The MathWorks, Inc., MATLAB, <http://www.mathworks.com/products/matlab/>, (2016).
[7] Openscad.org, OpenSCAD, <http://www.openscad.org/>, (2016).
[8] Wolpert, D. H. and Macready, W. G., "No free lunch theorems for optimization", IEEE Trans. Evol. Comp., **1**, #1 (1997).
[9] Sutin, B. M., Skewray, <https://github.com/skewray/skewray_old>, (2016).
[10] Stavroudis, O. N., [The Mathematics of Geometrical and Physical Optics], Wiley-VCH, Weinheim, p64 (2006).
[11] Rock, D. F., "Using differential ray tracing in stray light analysis", Proc. SPIE, 8495 (2012).

[12] Seo, B. *et al*, “Analysis of normalized point source sensitivity as a performance metric for large telescopes”, Appl. Optics, **48**, issue 31, p5997 (2009).
[13] King, I. R., “Accuracy of measurement of star images on a pixel array”, PASP, **95**, p163 (1983).
[14] Sutin, B. M., “New Methods of Optical Modeling for Astronomical Instrumentation”, Bull. AAS, **28**(N2):905 (1996).

15. ACKNOWLEDGEMENTS

The TMT Project gratefully acknowledges the support of the TMT collaborating institutions. They are the Association of Canadian Universities for Research in Astronomy (ACURA), the California Institute of Technology, the University of California, the National Astronomical Observatory of Japan, the National Astronomical Observatories of China and their consortium partners, and the Department of Science and Technology of India and their supported institutes. This work was supported as well by the Gordon and Betty Moore Foundation, the Canada Foundation for Innovation, the Ontario Ministry of Research and Innovation, the National Research Council of Canada, the Natural Sciences and Engineering Research Council of Canada, the British Columbia Knowledge Development Fund, the Association of Universities for Research in Astronomy (AURA) and the U.S. National Science Foundation.