ASTR 597: Optical Raytracing

This activity explores the seedy underbelly of optical raytracing—the hard-core way. Rather than rely on a high-powered, full-function optical design program (like Zemax), we'll go the "cowboy" route—in the form of a C-program (alternatively, Python). Why do it this way? Because it builds character. And sometimes it's perfectly sufficient (and faster) to bake your own than procure and learn how to use a sophisticated package.

The hard part is already done: the program is written. For reference, this activity (along with mathematical development) was completed in a single day of work. There is no chance I could have had Zemax in hand as quickly, though the custom program is *far* less versatile.

The program allows you to trace an arbitrarily large number of surfaces (currently limited to 20 in the variable declarations in C, but trivial to bump up), the surfaces being planar, spherical, ellipsoidal, parabolic, or hyperbolic. The refractive index is constant between surfaces, and ignorant of wavelength. All rays are in a 2-d plane: no skew rays allowed. The mathematical basis for the program is detailed in this document.

There are three almost identical programs that you will use:

1. one_2d.c (or one_2d.py—right-click and save): traces a single ray through the lens system. **Arguments**: *filename* z_{init} y_{init} $slope_{init}$ [z_{screen}]. The *filename* names a file with the surface parameters for the system in question (more on this later). The initial *z* and *y* positions of the ray are obvious. The *slope* is the tangent of the initial ray angle: 0.0 for level, -0.1 for descending to the right at a one-in-ten slope. The optional screen position (zero if not set) lets you evaluate the *y*-position of the final ray at $z = z_{screen}$.
2. par_2d.c (or par_2d.py—right-click and save): traces a pair of parallel rays through the lens system. **Arguments**: *filename* z_{init} y_{init} $slope_{init}$ *offset* [z_{screen}]. Most of the arguments are as above, though the initial *z* and *y* positions refer to an imaginary central ray midway between the pair. The *offset* parameter then sets the plus-and-minus offset in the *y*-direction of the two rays. For example, arguments -10.0 4.0 0.1 1.0 will set up rays at (-10, 3) and (-10, 5), both traveling with a slope of 0.1.
3. point_2d.c (or point_2d.py—right-click and save): traces a pair of rays diverging from or converging to the initial point. **Arguments**: *filename* z_{init} y_{init} $slope_{init}$ *offset_angle* [z_{screen}]. Most of the arguments are as above, though the initial *z* and *y* positions refer to the ray intersection point, and the *offset_angle* refers to the angular offset (plus and minus) about the nominal (imaginary central) ray angle. For example, arguments -100.0 0.0 0.1 0.02 will set up rays crossing at (-100, 0), sloping up at 0.08 and 0.12.

Note about Python programs: I had to rename the code to `*.pyprog` in order to defeat the scripting action of the UCSD physics server, so you will want to change the name to `*.py` once the program lands on your machine.

All programs print out the intersection points and slopes for every ray and every surface, as well as the *z*-intercept and screen-intercept of the final ray(s). The last two programs also present the point of intersection of the final two rays. Note that if the *z*-value of the intersection is to the left of the lens system, the final rays appear to diverge from this point.

Input files

The *filename* specified in the command line argument refers to a file containing the lens system parameters. An example bi-convex lens is specified by:

```
2
1.0
1.5 -5.0 150.0 0.0
1.00 10.0 -150.0 0.0
```

The first line tells us how many surfaces are described in the file. The second line is the refractive index we start in (air in this case). Each ot the following surface lines have four paramaters. The first is the refractive index you'll cross *into* when traversing the surface. The second is the distance of the vertex *from the previous vertex* (z=0 for the zeroth "vertex"). The third argument is the radius: positive means the center of curvature is to the right of the vertex. Note that planar surfaces are specified by a gargantuan radius. The final argument is the conic constant, K. For a sphere (circle), K=0.0. Ellipses are described by K>−1; parabolae have K=−1; and hyperbolae have K<−1.

An achromat (doublet) can be described by three surfaces, presuming the two lenses have a common surface with the same radius and no air gap:

```
3
1.0
1.5187 -1.5 150.0 0.0
1.624 3.0 -190.0 0.0
1.0 1.5 1.0e20 0.0
```

Two surfaces are spherical and one is planar. The refractive indices correspond to BK7 crown glass and F2 flint at 550 nm. The first lens is 3 units (mm?) thick at the center and has a bi-convex shape. The second lens is 1.5 units thick at the center and has a plano-concave shape.

A beam expander, consisting of a positive plano-concave and a plano-convex lens pair is described by the following file:

```
4
1.0
1.5 -1.5 -100.0 0.0
1.0 3.0 1.0e20 0.0
1.5 198.5 1.0e20 0.0
1.0 4.0 -201.55 0.0
```

Here there are four surfaces defining two lenses surrounded by air. Two of the surfaces are planar, and two are spherical. The vertices sit at z-values of -1.5, 1.5, 200.0, and 204.0 (remember the second parameter is an z-offset *from the previous* vertex).

Example Output

Using the symmetric bi-convex lens described above, we get the following output from the three programs:

```
one_ray bicx -10.0 0.1 0.0
Surface 1 has n = 1.500000, z_vert = -5.000000, radius = 150, K = 0.000000
Surface 2 has n = 1.000000, z_vert = 5.000000, radius = -150, K = 0.000000
Ray 1 has z = -10.000000; y = 0.100000; thet = 0.000000
Ray 2 has z = -4.999967; y = 0.100000; thet = -0.000222
Ray 3 has z = 4.999968; y = 0.097778; thet = -0.000659
Ray intercepts screen at (0.000, 0.101074); z-axis at (153.314500, 0.0)
```

We used a ray parallel to the optical axis and only a tiny bit above it to find that the focal point is at z = 153.3145. (**Note that the last line contains two disconnected sets of numbers:** the y-value of the ray as it intercepts the screen (at z=0 unless specified otherwise), and the z-value of the ray when it crosses the z-axis.) Using this information, we now trace a pair of parallel rays at y = ±5.0 to check out the spherical aberration:

```
par_ray bicx -10.0 0.0 0.0 5.0 153.3145
Surface 1 has n = 1.500000, z_vert = -5.000000, radius = 150, K = 0.000000
Surface 2 has n = 1.000000, z_vert = 5.000000, radius = -150, K = 0.000000
Ray 1+ has z = -10.000000; y = 5.000000; thet = 0.000000
Ray 1- has z = -10.000000; y = -5.000000; thet = 0.000000
Ray 2+ has z = -4.916644; y = 5.000000; thet = -0.011115
Ray 2- has z = -4.916644; y = -5.000000; thet = 0.011115
Ray 3+ has z = 4.920250; y = 4.890654; thet = -0.033004
Ray 3- has z = 4.920250; y = -4.890654; thet = 0.033004
+y ray intercepts screen at (153.315, -0.008793); z-axis at (153.048188, 0.0)
-y ray intercepts screen at (153.315, 0.008793); z-axis at (153.048188, 0.0)
Rays intersect at (z,y) = (153.048188, 0.000000)
```

This time, we set the screen at the nominal focus, and see that the rays have already crossed and strike the screen at *y* = ±0.00879. The rays intersect the *z*-axis (and each other, it turns out) at *z* = 153.048188—a little shy of where they did for *y* = 0.1. This is spherical aberration at work.

We can use the same program to see the curvature of the focal plane by giving the rays an initial angle:

```
par_ray bicx 0.0 0.0 0.1 0.1 153.3145
Surface 1 has n = 1.500000, z_vert = -5.000000, radius = 150, K = 0.000000
Surface 2 has n = 1.000000, z_vert = 5.000000, radius = -150, K = 0.000000
Ray 1+ has z = 0.000000; y = 0.100000; thet = 0.099669
Ray 1- has z = 0.000000; y = -0.100000; thet = 0.099669
WARNING: ray jumped backwards!
WARNING: ray jumped backwards!
Ray 2+ has z = -4.999467; y = -0.399947; thet = 0.067278
Ray 2- has z = -4.998800; y = -0.599880; thet = 0.067725
Ray 3+ has z = 4.999750; y = 0.273799; thet = 0.100093
Ray 3- has z = 4.999980; y = 0.078322; thet = 0.101421
+y ray intercepts screen at (153.315, 15.168768); z-axis at (2.273435, 0.0)
-y ray intercepts screen at (153.315, 15.172356); z-axis at (4.230388, 0.0)
Rays intersect at (z,y) = (150.641969, 14.900371)
```

Note that we started the reference ray at (0,0) this time, *right in the center of the physical lens*! But the program (almost) doesn't care: it still operates sequentially, and calculates the intersection of the parallel rays with the first surface. (To be clear about this, we laid down a ray definition as a point and slope. The resulting line exists for **all** values of *z* from −∞ to +∞, and will cross the lens surface wherever it has to—even if this is to the left of the reference point used to define the ray.) Because of the refractive interface, the reference ray may not actually pass through the initial point, but they will "approach" the lens as if it is headed for this point. In this case, we can ignore the warning, because we meant to "backtrack" to the first surface. We also went back to a small parallel separation so-as not to introduce much spherical aberration into the examination of the focal plane curvature. Note that the intersection point is significantly shy of the on-axis focal plane of 153.3145.

Finally, let's examine image behavior. Noting that the lens focal length is approximately 150 units, we can guess that an object at -200 units will make an image about 600 units on the other side, and be three times larger.

```
point_ray bicx -200.0 10.0 -0.05 0.05
Surface 1 has n = 1.500000, z_vert = -5.000000, radius = 150, K = 0.000000
Surface 2 has n = 1.000000, z_vert = 5.000000, radius = -150, K = 0.000000
Ray 1+ has z = -200.000000; y = 10.000000; thet = 0.000042
Ray 1- has z = -200.000000; y = 10.000000; thet = -0.099958
Ray 2+ has z = -4.665752; y = 10.008127; thet = -0.022248
Ray 2- has z = -4.693265; y = -9.587830; thet = -0.045045
Ray 3+ has z = 4.679512; y = 9.800184; thet = -0.066274
Ray 3- has z = 4.665648; y = -10.009689; thet = -0.034174
+y ray intercepts screen at (0.000, 10.110767); z-axis at (152.337440, 0.0)
-y ray intercepts screen at (0.000, -9.850183); z-axis at (-288.123667, 0.0)
Rays intersect at (z,y) = (620.222741, -31.053950)
```

The rays indeed intersect around *z* = 600, with a *y*-value about three times (and inverted) the original. The initial ray direction was set to aim for the center of the lens: a slope of -10/200. Note that the first intersections are pretty symmetric on the lens at *y* = ±10. This is not a happy accident, but the result of planning the ray parameters. Try to be equally attentive to such things in your analyses.

Getting Started

The programs above, while primitive, open up a wealth of territory to explore. Doing so the "cowboy" way will hopefully be instructive, provide a useful tool for your future, and give you an appreciation for the bells-and-whistles programs. You can pick topics to explore from the list below to get your feet wet. Restrictions will apply to what you pick, so tune in at the beginning of lab for details.

Compiling and/or running the code

The links to the codes (for C and Python) appear above. The C code will need to be compiled. To compile on a unix/linux platform, you'll do something like: `gcc -lm -o par_2d par_2d.c`. The `-lm` flag links the math library. The output will be called `par_2d` in this case. Run from the command prompt **with command line arguments** as described above. You may need to include a `./` before the prgram name (as in `./par_2d`) if your current working directory is not in your path.

The Python code does not need to be compiled. I had to rename the code to `*.pyprog` in order to defeat the scripting action of the UCSD physics server, so you will want to change the name to `*.py` once the program lands on your machine. The fist line is configured as a shell directive that should invoke Python if it is installed on your system and if the `*.py` program is executable. Do something like: `chmod +x par_2d.py` to make the program executable. Now you should be able to run just as described in the previous paragraph (possibly needing the `./`). Alternatively, you can invoke by saying something like: `python par_2d.py` *arguments*.

Feel free to change the program to suit your special needs: look for printf() statements to change/eliminate/add any information you want. Go nuts!

Example Activities

1. **Thick Lens Focal Length**: Look at the focal length of a symmetric bi-convex lens as a function of thickness. Use paraxial rays to be insensitive to spherical aberration.
2. **Plate Glass Spherical Aberration**: Investigate the spherical aberration produced by a converging beam going through a plate of glass. **Use analytic methods** to determine focus shift as a function of thickness and refractive index, using the simplifying approximation that $\sin x = \tan x = x$. Use the **point_ray** program, and set the initial intersection at the place the focus *would* have come to without the glass there. Verify that the focus shift agrees with the analytic calculation, and explore the second-order effect of focus shift as a function of *f*-number (see tip below). See the tips below on ways to explore spherical aberration. This problem is relevant for imaging systems behind a window--as often is the case for CCDs.
3. **Spherical Aberration of a Lens**:Investigate the spherical aberration of a symmetric bi-convex positive lens. when the rays are parallel to the optical axis. See tips below for ways to look at spherical aberration. Use **par_ray** for this one.
4. **Image Quality**: Look at the image quality of a symmetric bi-convex lens when the image and object distances (s and s') are the same. Chose a point on the optical axis, establish the conjugate image point with a pair of rays with a very small angular offset (using **point_ray**), then putting the screen here look at the focal/position/blur as a function of divergence angle (offset_angle).
5. **Plano-convex Orientation**: Make a plano-convex lens (plane on one side, convex on the other) lens and compare the on-axis spherical aberration (sufficient to look at focus shift) as a function of which way the curved side is oriented relative to the oncoming parallel beams. One is better than the other (less aberration). Can you think of a reason why one is better than the other? This is a highly useful thing to know: plano-convex lenses are common, and it is nice to know what a difference orientation makes (and which one is best and why).
6. **Optimum Lens Shape**: As a generalization of the previous problem, design a positive lens whose radii are allowed to change, but whose focal length remains the same. Use the lens-maker's formula to establish this condition. Parameterize your shapes by $s = f\times(n - 1)\times(1/R_1 + 1/R_2)$. A plano-convex lens will have a shape parameter of ±1; symmetric lenses will have $s = 0$. Note that both R-values can have the same sign and become a meniscus lens. For a given parallel ray offset (e.g., ±10 or equivalent *f*-number), plot the focus shift (relative to focus of tiny offset) as a function of s. You should see a minimum s-value, suggesting this is the best shape for a spherical singlet lens. Another handy thing to know!
7. **Field Curvature**: Orient a plano-convex lens in the orientation suggested by the item two entries above, and send small-offset ray pairs (parallel) through the center of the lens at a variety of angles. Map out the intersection of rays in the z-y plane. Is the focal surface flat or curved? Which way? Is it roughly circular, or some other shape? Use the trick of initiating rays from the lens center.
8. **Cat's Eye**: Send parallel rays into a single spherical surface. Using paraxial rays, determine the index of refraction that would bring rays to focus on the back wall of the sphere (don't need to put this into model). This is called a cat's eye (though not an actual *cat's* eye). If the back surface were coated to be

reflective, then a parallel ray bundle entering the face of the sphere would come to a focus on the back, get reflected exactly back along the same set of paths, and emerge traveling back to the source. Given the spherical symmetry, it would work regardless of incident angle (in the un-coated hemisphere). So it's a sort of retroreflector--like a glowing cat's eye in the headlights!

9. **Hyperbolic Lens**: Make a plano-convex lens with a hyperbolic curved surface. The right shape constant will eliminate spherical aberration for parallel, on-axis light incident on the planar face (which thus does not bend any light: it's all up to the hyperbola). First use paraxial rays to determine the focal length, then open up to large offsets and seek the shape constant that results in convergence to the same focus. You can find the right shape constant by trial and error (hint: pick a rational refractive index). Or you can think about what would happen for extreme offsets: where the hyperbola would be flat and asymptotic. What asymptotic slope (related to shape constant) would direct rays along the asymptote toward the distant focus? The answer yields a relatively simple relation of the refractive index. It's good to know that a hyperbola can eliminate spherical aberration in this case (but it may not do so well at other aberrations).
10. **Elliptical Surface**: Make a single refractive surface of elliptical shape, and allow incident parallel rays to come to focus within the medium. Find the focus using paraxial rays, then open up to large offsets and find the shape constant that eliminates spherical aberration. The number is not entirely unrelated to the number above for the hyperbola (but not simply the negative). So pick a rational refractive index to have a better chance of hitting on the right value.
11. **Elliptical Meniscus**: Using knowledge gained above, make a meniscus positive lens of an elliptical and spherical surface (in that order) that is free of spherical aberration. The hint is that you must preserve the rays coming off the ellipse, so that the rear surface does not change the ray directions. Verify that the resulting lens has no on-axis spherical aberration.
12. **Coma**: Use the hyperbolic plano-convex lens from above (which has a q-value that eliminates on-axis spherical aberration). Now send in parallel rays at an angle (through the lens center) and see how good the focus is. To compensate for field curvature, start with a tiny offset, and place the screen at this intersection for the rest of the rays at this angle. Now try a variety of parallel ray offsets (still through lens center) and see where they hit the screen. The y-values of the says on the screen tell you about the blur. The offset from the nominal focal point is the tell-tale mark of coma. Examine a few different input angles in this way.
13. **Chromatic Aberration**: Typical lenses are made of BK7 glass. This has refractive indices at 450, 550, and 650 nm of 1.5252, 1.5187, and 1.5145, respectively. Make a lens (pick your style) and examine the focus at these three wavelengths for the same pair of parallel, on-axis rays. You'll have to generate three lens files, with the three refractive indices. If you set the screen at the green (middle) focus, what is the blur size at red and blue? Compare this to an angle of incoming light: that is, lenses convert directions to focal plane distances. What is the angular spread associated with the chromatic blur?
14. **Achromatic Lens**: Build an achromatic doublet out of two types of glass: BK7 crown (properties listed above), and F2 flint (1.639, 1.624, 1.6154 at 450, 550, 650, respectively). There will only be three surfaces: the two lenses meet with no air gap. Start by making the first lens (BK7) a symmetric bi-convex 3 mm thick and radii of 150 mm. Make the second lens 1.5 mm thick, with a planar rear surface. Vary the radius of the common surface (starts at -150) until the red and blue focus at the same place. Don't worry about the constant shifting of focus as this parameter changes: you just care about the relative focus of red and blue. Once you have this dialed in (remember to change the parameter in all three lens files), see how far the green focus is from the other two (inside or outside?). Also, if you put the screen at the red/blue focus, what is the green blur diameter, expressed in effective angle (see above). If you compare the blur to the singlet above using the same rays, you'll see how good the achromat is.
15. **Apochromatic Lens**: An apochromatic lens uses three lenses of three materials to further compensate chromatic aberration. We will use a design that has two lenses in contact (share one surface) plus a free-floater. Use this prescription as a starting point. The glasses are N-F2, KZFSN5, and N-KF51A (design inspired by: this page, but tweaked and with slightly different glasses). The refractive indices are included in the prescription file itself. Note there is a formula for computing the refractive index at any wavelength in addition to the numbers for five different wavelengths. Using similar techniques to the previous entry (just working with 450, 550, 650 nm), tweak the second ant third radius to give you the flattest focus you can get (within a few microns), looking at slope and curvature. We may not simultaneously achieve the best spherical aberration and coma, etc. in just these two parameters, but we can at least stomp out chromatic aberration. As in the previous exercises, put the results in some meaningful context.

16. **Beam Expander**: Design a 3× beam expander using a plano-concave and a plano-convex lens. The general idea is that a parallel (collimated) beam incident on the negative lens diverges, is caught by the positive lens, and re-collimated at three times the original beam diameter. The virtual focus of the negative lens wants to be coincident with the real focus of the positive lens (draw a diagram to see). Let's say the initial beam diameter is 10 mm, and you want the outgoing beam diameter to be 30 mm. Aim for a lens separation of at least 150 mm so that the *f*-number is not too extreme. When you've got the lens parameters and separations right so that the outgoing beams are perfectly parallel, see what happens when the incident angle is non-zero. Are the outgoing rays still parallel? Is the angle the same, or different? If different, by what factor?
17. **Telescope with Positive Eyepiece**: We've seen telescope systems in class. Let's make a telescope with a 60 mm diameter lens, a focal length of 900 mm, using an eyepiece of 18 mm (positive) focal length. Predict the magnification and lens separations based on a drawing of the geometry and associated raytraces. Verify the magnification by raytracing the system with on-axis rays at ±30 mm. Adjust focus for parallel rays at the output (and still parallel to the optical axis). The point of ray intersection is a sensitive measure of this: it should tend toward infinity (negative infinity okay: just means slight *divergence*, rather than positive infinity slight *convergence*). The separation between the rays should be smaller than the entrance aperture (60 mm) by a factor of the magnification. Is this true? Once you have the heights of the emerging parallel rays, put in parallel ±30 mm rays at an angle of 0.005 radians (initiate rays at objective lens center). Is the magnification the same? Does the emergent angle correspond to the magnification? Where do you have to put the screen so the screen heights of the emergent rays match the ray heights of the original on-axis rays? This is the pupil location. Does it match where the image/lens formula tells you the image of the objective should be through the eyepiece lens? Finally, does this telescope invert the image? In order to answer this, ask what direction an observer would have to look through a telescope to see "down" on the outside. If the answer is "up," then the image is inverted.
18. **Telescope with Negative Eyepiece**: Perform the same set of exercises as above, but with a negative eyepiece instead of a positive one.
19. **Complex Lens System**: "Real" lenses—such as may be found on cameras— are more complex than the ones we have thus far modeled. By having many spherical surfaces, different materials, and lens clusters whose intervening distance may be modified, it is possible to correct for many aberrations. The Petzval lens is such an example. Five lenses of four different materials in three clusters helps eliminate nastiness. The lens prescription is here. **You may use this lens to contrast properties studied above,** such as spherical aberration, coma, chromatic aberation, and field curvature. Some things to note:
    a. You must find the nominal focus position in the usual way: with paraxial rays near the axis.
    b. The lens aperture is 50 units in diameter, so keep rays within (but up to) ±25 units at the front surface.
    c. The focal plane is meant to be about 15 units across, so keep your entrance angle small enough to hit the focal plane (screen) within $y = \pm 7.5$ units.
    d. The five lenses are of the materials: Balkn3, F4, BK7, F2, and F2, so that the refractive indices in green/yellow are: 1.518489, 1.616592, 1.5168, and 1.620040, respectively. If exploring chromatic aberration, you will need to explore the blue, where n = 1.523618, 1.626627, 1.521601, and 1.630203; and red, where n = 1.515014, 1.609793, 1.513547, and 1.613154, respectively. The prescription must be modified accordingly.

    In checking out the Petzval's aberrations, you *will* see them, but be smart about assessing the magnitude of the problems—not only in relation to the simple lens example, but also in absolute terms. If we say the units are millimeters (reasonable), then how big are the errors in wavelengths? When errors at the screen are smaller than a wavelength, they can be practically ignored. Or it may even be pixel size that matters (maybe 5 or 10 microns). Put the deviations in context.

Tips

- "On-axis" means parallel to the optical axis (not on top of the optical axis itself).
- Paraxial rays means rays very close to the optical axis: small offset.
- An *f*/5 beam corresponds to slopes of the exterior rays of ±0.1. *f*/8 means ±0.0625, etc. This is handy considering that the program arguments are angles, but you're asked to evaluate some things as a function of *f*-number.

- When defining a lens, **make sure its thickness is appropriate** for the *y*-values you want to use. The sag of a circle is $R - \text{sqrt}(R^2 - h^2)$. For example, if we want a lens with a 150 mm radius of curvature to span ±30 mm in *y*, then we need a depth allowance of just over 3.0 mm.
- When **characterizing spherical aberration**, there are several ways to go. You can note the shift in focus as a function of beam angle (or *f*-number) or of beam height (also convertible into *f*-number). You can plot a side-view of the rays coming at their different angles into their different foci. In doing so, you will see where the best focus lies, and even be able to estimate blur size. You can also place the screen at some representative focus and let the screen-intercept values tell you something about blur size.
- If you're exploring the impact of the shape of a lens on properties like spherical aberration, use the lensmaker's formula: $1/f = (n - 1)(1/R_1 - 1/R_2)$, setting R_1 to some fraction or multiple of R_2. This way, you can vary the fraction (thus the shape) while preserving the focal length.
- When looking for aberrations, always trace a set of rays with very little deviation (either closely parallel or close in angle) to establish a baseline of what the lens would do in the ideal.

[Back to ASTR 597 page]