Optical and Ray Tracing Mathematics

October 9, 2014

Most recent revision June 19, 2017

Abstract

This article describes the mathematical methods used in the OpticalRayTracer optical design tool, as well as general optical mathematical methods. Because OpticalRayTracer uses ray tracing to accomplish its results, ray tracing methods and mathematics are also described.

Contents

List of Figures

1 Overview

About a decade ago I wrote the first version of OpticalRayTracer[1] (hereafter ORT), intending it as a simple optical design tool for educational and entertainment purposes. Since that beginning I have received and responded to many requests for additions and corrections, with the result that recent ORT versions have become more versatile and better at imitating nature.

1.1 Ray Tracing

Using a method known as Ray Tracing[2], ORT traces light rays through optical media such as lenses and mirrors, modeling the physics of light as it does.

A note of explanation – In computer technology, ray tracing has two distinct meanings. In computer graphics, ray tracing produces very realistic images by processing each display picture element (*pixel*[3]) as a separate task – tracing the path of that location out into a virtual world where it interacts with various objects. In physics, ray tracing refers to the practice of tracing rays through physical elements and media with the aim of analyzing their interactions and behavior. ORT traces rays in the latter sense.

1.2 Snell's Law

One of ORT's primary activities is to apply Snell's Law[4], a cornerstone of modern optical design. Snell's Law describes the behavior of light as it moves through different media, each of which possesses an Index of Refraction[5] (hereafter IOR), a number that reveals the speed of light in that medium. For example, an IOR of 1.5, typical of glass, tells us that light travels only about 66% ($\frac{1}{1.5}$) as fast in that medium as it does in a vacuum (with an IOR of 1).

For those of my readers familiar with Relativity theory, who understand that the speed of light is a constant in all frames, I should add that the IOR tells us, not that light's speed has changed, but that light takes a longer path at its normal speed, with delays along the way.

1.2.1 Scalar form

Here is the classic form of Snell's law:

$$n_1 \sin\theta_1 = n_2 \sin\theta_2 \tag{1.1}$$

Where θ_1 and θ_2 are light wavefront angles and n_1 and n_2 are indices of refraction. A practical restatement of equation (1.1), meant to acquire an angle result, is:

$$\theta_2 = \sin^{-1} \frac{n_1 \sin\theta_1}{n_2} \tag{1.2}$$

1.2.2 Vector form

In order to be indifferent to the relative angle between a light ray and a lens, as well as from which side of the lens the light approaches, ORT uses a vector form to compute Snell's Law outcomes – it looks like this[6]:

$$r = \frac{n_1}{n_2} \tag{1.3}$$

$$c_1 = \pm\hat{N} \cdot \hat{I} \tag{1.4}$$

$$c_2 = \sqrt{1 - r^2(1 - c_1^2)} \tag{1.5}$$

$$\hat{T} = r\hat{I} + (rc_1 - c_2)\hat{N} \tag{1.6}$$

- $\hat{I}$ = incident light ray vector
- $\hat{N}$ = lens surface normal vector
- $\hat{T}$ = refracted light ray vector

A word of explanation: In equation (1.4) the value c_1 is the dot product of unit vectors $\hat{N}$ and $\hat{I}$, but if the result value is negative, $\hat{N}$ is negated and equation (1.4) is performed again. In this case, the negated value for $\hat{N}$ is retained for use in equation (1.6). The reason for this algebraic sign management is so that light rays can approach a lens from either side and be treated the same.

1.3 Index of Refraction

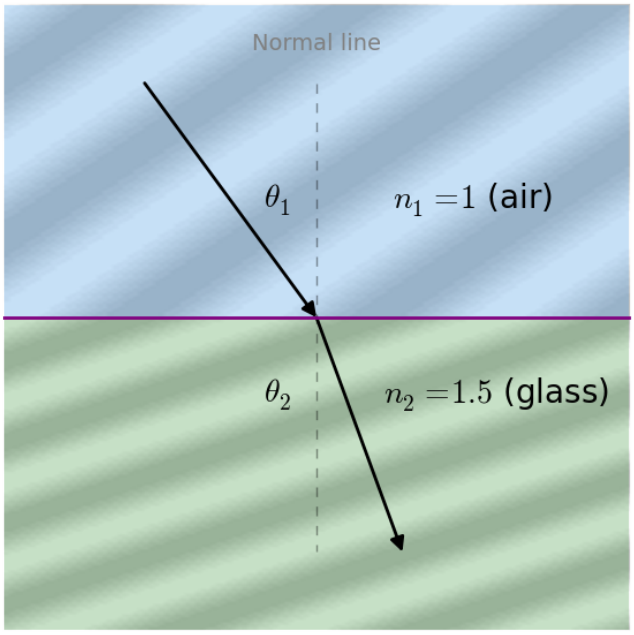

Figure 1: Index of Refraction

Figure 1 may provide an intuitive sense of why a slowing of light waves causes the light wavefront to change direction, and why a lens curved surface can change the direction of light (a direction change called *refraction*[7]). Note that the change of direction shown in Figure 1 is reversible – a light wavefront traveling in the opposite direction (from bottom to top in Figure 1) would experience the opposite deflection. Expressed another way, when moving into a medium with a higher IOR (and a slower light speed), light is deflected toward the normal line (the vertical dotted line in Figure 1), but when moving into a lower IOR, such as from glass to air, light is deflected away from the normal line (imagine reversing the arrows in Figure 1).

It's also important to understand that a light wavefront approaching perpendicular to the surface of a medium (that is, in the direction of the normal line of Figure 1) experiences no deflection.

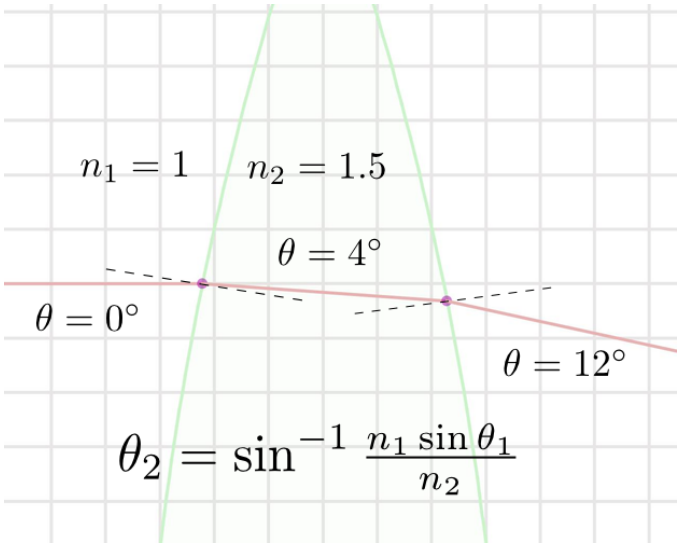

Figure 2: Snell's Law refraction example

In Figure 2 we see a lens (green outline) in ORT, with a light beam that is deflected twice, once while entering the lens, once while exiting. Even though the transitions involve opposite applications of Snell's Law (the values for refractive indices n_1 and n_2 are reversed), and because of the shape of the lens and the direction of the light beam, the beam is refracted in the same direction while entering and leaving the glass. The reason is provided by the normal (dotted) lines shown in Figure 2 – when entering, because of the higher IOR of glass and for reasons given above, the light beam is refracted toward the normal line, which rotates it clockwise (from 0° to 4°), and while leaving, because of the lower IOR of air, the beam is refracted away from the normal line, which causes it to be rotated further clockwise (from 4° to 12°).

1.4 Total Internal Reflection

In computing refraction outcomes, there are cases where a combination of entry angle and IOR produce an impossible Snell's Law equation outcome, one that (in nature) causes the light beam to reflect internally instead of exiting the medium. This is called *total internal reflection*[8]. For a given IOR, one may easily compute the critical angle past which internal reflection must take place:

$$\theta_{crit} = \sin^{-1} \frac{n_1}{n_2} \tag{1.7}$$

Here's an example critical angle calculation using the indices of refraction from our earlier example:

$$\theta_{crit} = \sin^{-1} \frac{1}{1.5} = 41.8° \tag{1.8}$$

This means that, at an interface between media having the indicated IOR differential, light beam angles greater than 41.8° will reflect internally. As it happens, when a critical angle example is submitted to ORT, it "does the right thing":

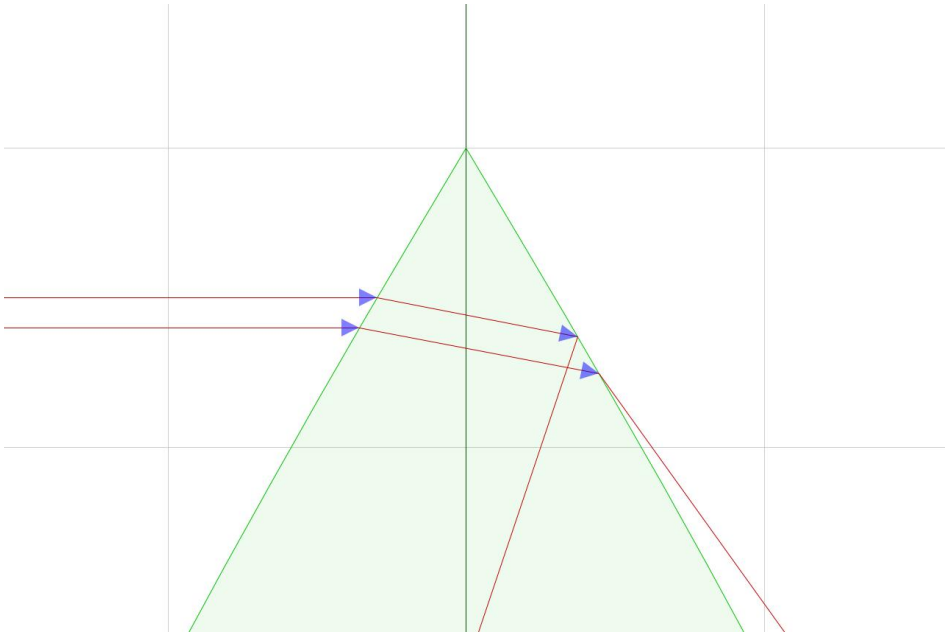

Figure 3: Total internal reflection

Figure 3 shows ORT generating two beams, one that exits the lens just below the critical angle, while the other, just above the critical angle (because of its path through the lens), instead reflects internally, just as in nature.

People may not realize that total internal reflection can be an everyday occurrence. For example, while swimming underwater, one cannot see through the water's surface at angles past:

$$\theta_{crit} = \sin^{-1} \frac{1}{1.333} = 48.6^\circ \tag{1.9}$$

This means a common underwater effect, known to all swimmers and divers, is explained by Snell's Law and optical refraction.

2 Basic Ray Tracing

2.1 Core algorithm

In this section we will provide a detailed explanation of ORT's method for tracing light rays. The procedure can be outlined like this:

1. Build a virtual world of lenses and mirrors, through which rays of light will pass.
2. Create a ray with an origination point and angle, that is, a polar vector with a defined origin.
3. In the direction given by the angle, make a list of all objects the ray would intersect and the details of the intersections.
4. Sort the resulting list of intersections, placing the nearest objects at the head of the list.
5. Select the nearest surface of the nearest object having a distance greater than zero (thus allowing the ray to move away from a given surface).
6. Test the surface to determine if it's part of a lens or mirror, or just part of a sphere that creates an optical object (details below).
7. If the surface is part of a lens or mirror, compute the relevant optical effects and use the result to change the ray's angle.
8. Unless the surface is a terminating plane, repeat this process from item (3) above.

The above process is repeated using as many light beams as desired to analyze the optical system.

2.2 Circle-line intersection

During ORT's tracing activity, most object detection applies a geometric equation set and method called *circle-line intersection*[9]. Here is the procedure for locating intersections between a light ray and circles representing lenses:

- In the following equations, a line of infinite extent is defined by Cartesian[10] locations x_1, y_1 and x_2, y_2, both of which lie on the line of interest.
- The specific values of x_1, y_1 and x_2, y_2 are less important than the fact that they lie on the line of interest, defined by the initial ray position and angle as explained above.
- In computer optical modeling, lenses are normally defined by two overlapping circles:

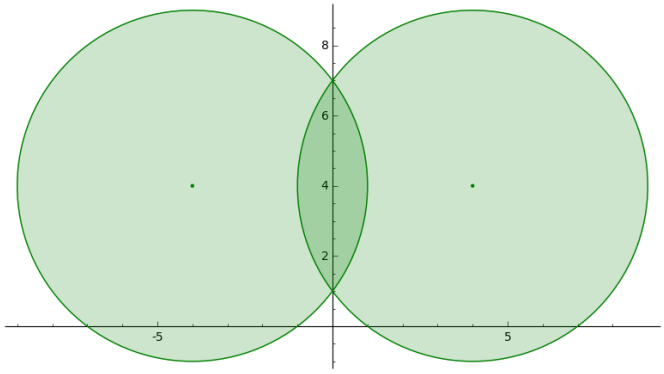

Figure 4: Lens defined by overlapping circles

- The modeled lens is defined by the area of overlap between the two circles, the dark area at the center of Figure 4.
- Each of the circles to be tested is defined by a location in Cartesian x, y coordinates, and a radius.
- The circles are each evaluated for intersections with the light ray:
  - We begin by defining some preliminary equations:

$$d_x = x_2 - x_1 \tag{2.1}$$
$$d_y = y_2 - y_1 \tag{2.2}$$
$$d_r = d_x^2 + d_y^2 \tag{2.3}$$
$$D = \begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} = x_1 y_2 - x_2 y_1 \tag{2.4}$$
$$\Delta = r^2 dr - D^2 \tag{2.5}$$

    In the above equations, as before, x_1, y_1 and x_2, y_2 describe a line for our ray of light, and r defines the radius of a tested circle.
  - With the above preliminaries, we define equations by which to locate points of intersection:

$$x_a = \frac{D d_y + \operatorname{sgn}(d_y) d_x \sqrt{r^2 d_r - D^2}}{d_r} \tag{2.6}$$
$$x_b = \frac{D d_y - \operatorname{sgn}(d_y) d_x \sqrt{r^2 d_r - D^2}}{d_r} \tag{2.7}$$
$$y_a = \frac{-D d_x + |d_y| \sqrt{r^2 d_r - D^2}}{d_r} \tag{2.8}$$
$$y_b = \frac{-D d_x - |d_y| \sqrt{r^2 d_r - D^2}}{d_r} \tag{2.9}$$

– Non-standard function $\text{sgn}(x)$ is defined as:

$$\text{sgn}(x) = \begin{cases} -1 & \text{if } x < 0 \\ 1 & \text{if } x >= 0 \end{cases} \tag{2.10}$$

- The first test is to evaluate the Δ equation (2.5) listed above – if $\Delta >= 0$, the ray doesn't cross within the target circle and can be skipped.
- if $\Delta < 0$, then the line's two points of intersection with the circle are defined by x_a, x_b and y_a, y_b (equations (2.6) - (2.9)), and the next evaluation step can be taken.

Here's a diagram of a typical field of lenses and circles, and a light ray for which we need intersection information:

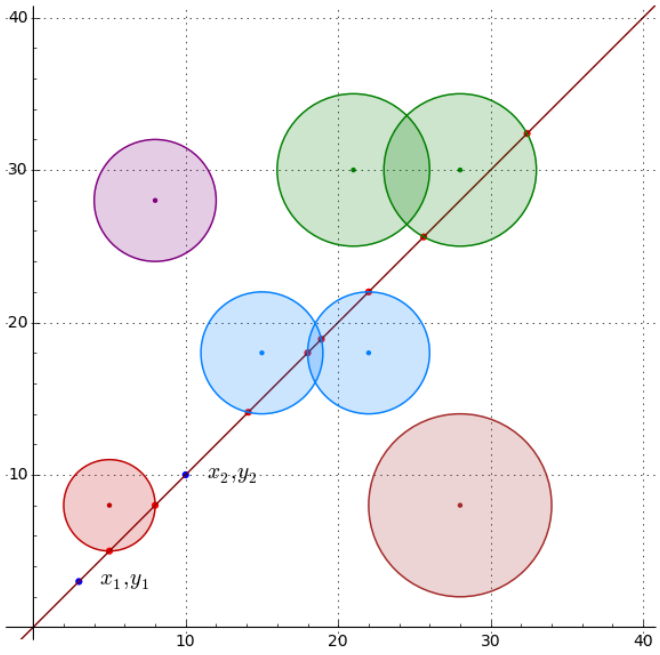

Figure 5: Ray intersection diagram

1. As explained earlier, a ray is generated and a list of intersecting targets is assembled using equations (2.1) - (2.9).
2. In the initial analysis, those circles having no intersections with the ray are eliminated by Δ (equation (2.5)).
3. The resulting intersection list is sorted by distance, so the nearest intersections are evaluated first.
4. In Figure 5, the detected points of intersection are marked with red dots.
5. Normally the only objects of interest are lenses, defined as overlapping circles, such as the blue lens at the center of Figure 5 and the larger green lens at the upper right.
6. Let's assume for this example that our ray's point of origin is x_1, y_1 at the lower left (point x_2, y_2 only serves to define the ray's angle or slope).
7. Moving toward the upper right, the first potential target is a red circle with two intersections. But this circle doesn't overlap with another circle, therefore it's not a lens, so it's bypassed.
8. The next surface is part of a blue circle that represents the left part of a lens, but this specific surface is not within the shared area of the lens, so it's also bypassed.
9. The next intersection is the near side of a lens defined by two overlapping blue circles. Therefore the search stops, Snell's Law is applied to the surface (as shown in Figure 1 above), a new angle is computed, and the entire search process repeats from step (1) above using this new origin point and angle.

For each lens surface, in other words when both entering and exiting a lens, the Snell's Law analysis procedure described above is applied and, because this changes the ray's direction, the search process begins anew with the present surface as the ray's point of origin.

For a given ray of light, the above process is repeated until there are no more targets in the field of view or a terminating plane is encountered. And the above-described process is repeated for as many light rays as the user needs to properly evaluate the optical configuration under test.

2.3 Mirror Reflection

ORT uses mirror reflection in two circumstances – directly, when an object is specified as reflecting, and in the case of total internal reflection, when a light beam exiting an optical medium exceeds the critical angle (discussed above), as a result of which the beam is reflected back into the medium.

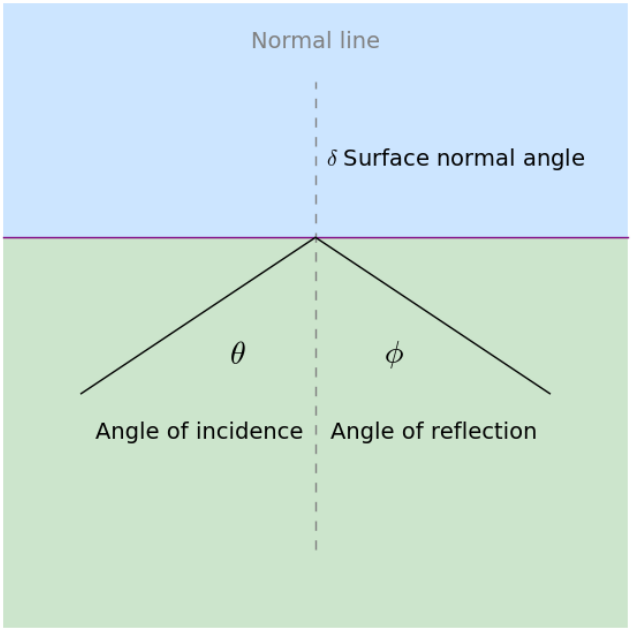

Figure 6: Mirror reflection

2.3.1 Scalar form

Using the variable names from Figure 6 and assuming units of radians, the equation for reflection is:

$$\phi = \pi - 2(\theta - \delta) + \theta \tag{2.11}$$

2.3.2 Vector form

To be indifferent to relative angles and to accommodate a vector-based approach, ORT uses a vector form for mirror reflection, using terminology from the above-described Snell's Law vector-form equation:

$$\hat{R} = \hat{I} - 2(\hat{I} \cdot \hat{N})\hat{N} \tag{2.12}$$

- $\hat{I}$ = incident light ray vector
- $\hat{N}$ = lens/mirror surface normal vector
- $\hat{R}$ = reflected light ray vector

Mirror reflection follows these rules[11]:

- The three angles in equations (2.11) and (2.12) all lie in the same plane.

- The angle of reflection and the angle of incidence bear the same magnitude relationship to the surface normal.
- The angle of reflection and the angle of incidence lie on opposite sides of the surface normal.

2.4 Distance from point to line

This surprisingly complex procedure is used to detect the nearest line segment when the user clicks the graphic display for information about a ray tracing line. Unlike other geometric procedures used by ORT, this one is meant for line segments, finite-length lines defined by Cartesian position pairs x_1, y_1 - x_2, y_2.

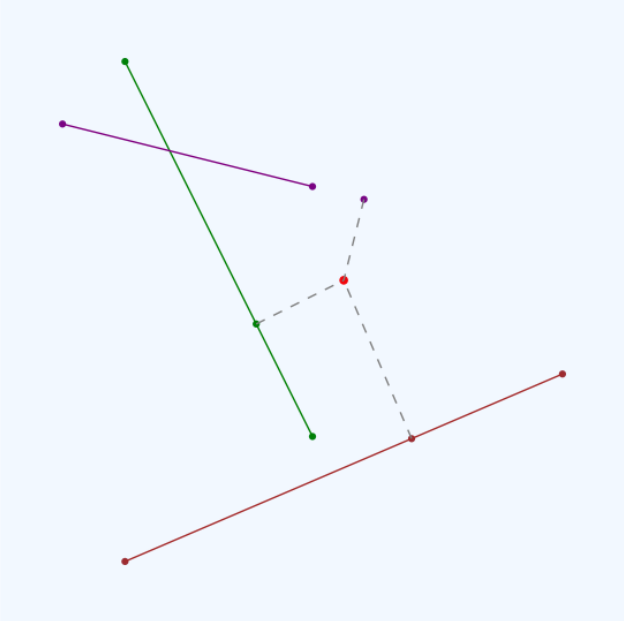

Figure 7: Point-line proximity calculation

Referring to Figure 7, the red point is the location x, y for which the closest line is sought. For each candidate line segment defined by x_1, y_1 - x_2, y_2, the following equations provide the point of closest proximity px, py:

$$p_x = \frac{(x_1 - x_2)y + x_2 y_1 - x_1 y_2}{y_1 - y_2} \tag{2.13}$$

$$p_y = \frac{(x - x_2)y_1 - (x - x_1)y_2}{x_1 - x_2} \tag{2.14}$$

Again referring to Figure 7, the purple line at the upper left yields a point of closest approach, but this result is for a line of infinite extent, not a line segment, and that line segment isn't actually adjacent to the test position. To guard against this outcome, after a point of closest proximity is computed, an additional test is performed to assure that the line segment x_1, y_1 - x_2, y_2 is adjacent to the test position x, y:

$$\text{adjacent} = \begin{cases} x_1 \le x \wedge x \le x_2 \vee x_2 \le x \wedge x \le x_1 & \text{if } x_1 \ne x_2 \\ y_1 \le y \wedge y \le y_2 \vee y_2 \le y \wedge y \le y_1 & \text{otherwise} \end{cases} \tag{2.15}$$

Line segments that pass the above tests are placed in an array. At the end of the evaluation the array is sorted by distance and the closest line is accepted as the user's target.

3 Advanced Topics

3.1 Dispersion computation

ORT knows how to compute spectral dispersion[12], a property of many optical materials in which the index of refraction depends on wavelength.

3.1.1 Abbe number

In dispersion calculations, an Abbe number[13] is acquired through laboratory studies of material dispersion. The Abbe number is calculated in different ways, here is one:

$$V_d = \frac{n_D - 1}{n_F - n_C} \tag{3.1}$$

- V_d: Abbe number
- Spectral lines:
  - n_D: Fraunhofer D[14], Sodium, 589.3 nm
  - n_F: Fraunhofer F, Hydrogen β, 486.1 nm
  - n_C: Fraunhofer C, Hydrogen α, 656.3 nm

  The Abbe number quantifies differences in Index of Refraction (IOR) for different wavelengths.

3.1.2 Sellmeier equation

To produce an effective IOR for a given material, one applies an equation based on empirically derived constants. One example is the Sellmeier equation[15], which bypasses the Abbe number scheme and derives its results from laboratory measurements:

$$n^2(\lambda) = 1 + \frac{B_1\lambda^2}{\lambda^2 - C_1} + \frac{B_2\lambda^2}{\lambda^2 - C_2} + \frac{B_3\lambda^2}{\lambda^2 - C_3} \tag{3.2}$$

The Sellmeier equation can be more concisely expressed this way:

$$n^2(\lambda) = 1 + \sum_{i=1}^{3} \frac{B_i\lambda^2}{\lambda^2 - C_i} \tag{3.3}$$

- n: Refractive index
- λ: Wavelength in micrometers
- $B_{1,2,3}, C_{1,2,3}$: Empirically derived coefficients for each material of interest

3.1.3 ORT/Sellmeier comparison

In ORT I use a much simpler (and less accurate) equation to produce a graphic representation of dispersion:

$$n' = n + \frac{(p - \lambda)c)}{a\,p\,\lambda^2} \tag{3.4}$$

- λ: Wavelength in nm
- n: Material index of refraction
- n': Dispersion-adjusted index of refraction at λ
- a: Abbe number for material
- p: Wavelength pivot, set to 589.3 nm
- c: An empirical constant equal to 5×10^5

Because ORT's dispersion function is meant only to represent dispersion in a general way, and because it's expected to function without knowledge of material properties apart from IOR and Abbe number, the above equation produces satisfactory results. Here's an example:

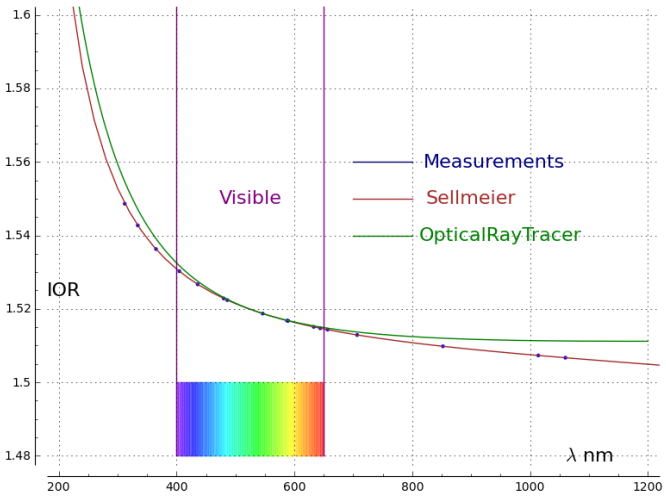

Figure 8: Dispersion equation comparison

Figure 8 shows an outcome based on measurements of the properties of Schott BK7 glass[16] and published Sellmeier coefficients for two of its traces (showing excellent agreement between prediction and measurement), and just the BK7 Abbe number and IOR[17] for the ORT result.

3.2 Hyperbolic Lenses

Even though it's based on a simple premise, in terms of theoretical complexity the hyperbolic lens model is by far the most complex part of ORT. The complexity doesn't arise because dealing with hyperbolic curves is innately complex, because that's not so. It arises because to model a hyperbolic lens, ORT must join two hyperbolic curves in a distinctly unnatural way, then find points of intersection between the modeled lens and a beam of light.

3.2.1 Advantages of hyperbolic lenses

Hyperbolic lenses are remarkable elements that overcome many of the drawbacks of spherical lenses. A short focal length spherical lens (i.e. an ordinary lens, a lens based on the intersection of two imaginary spheres) suffers from spherical aberration[18]:

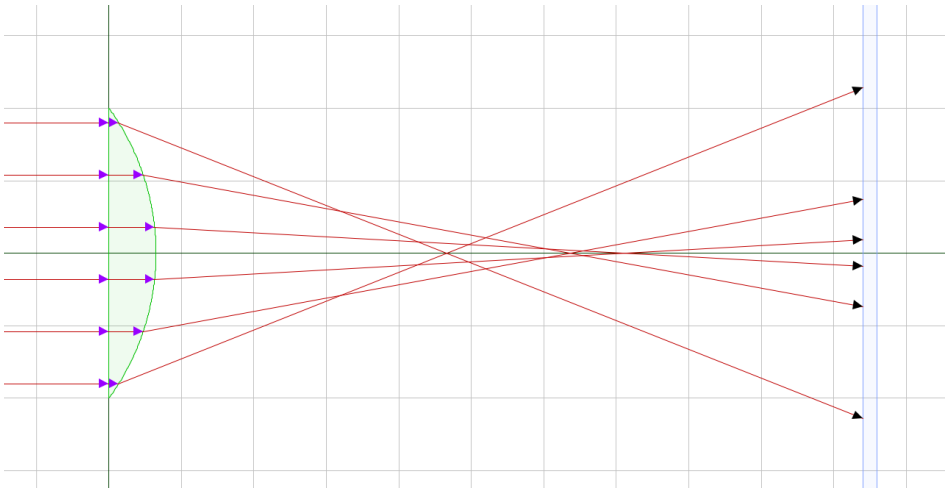

Figure 9: Spherical aberration

By comparison, a carefully configured hyperbolic lens of the same overall shape, material and design, a lens that differs only in the curvature of its surfaces, produces this result when modeled in ORT:

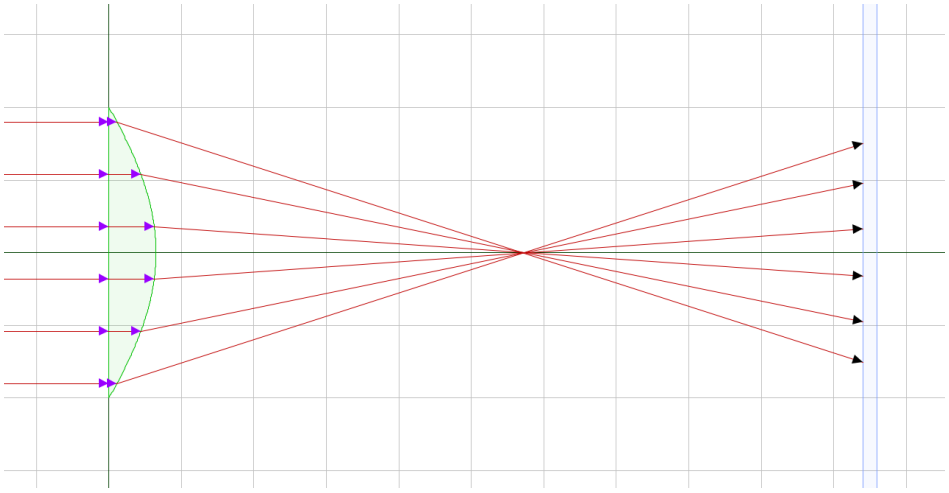

Figure 10: Optimized hyperbolic lens

3.2.2 Parabolic curve option

When presented with a large numeric value such as 1×10^7, the ORT hyperbolic curve generator creates an approximate parabolic curve. This is a side effect of the mathematical method used to generate a range of hyperbolic curves, and may be useful in designing telescope mirrors as one example. Here's an ORT image showing an approximate parabolic curve:

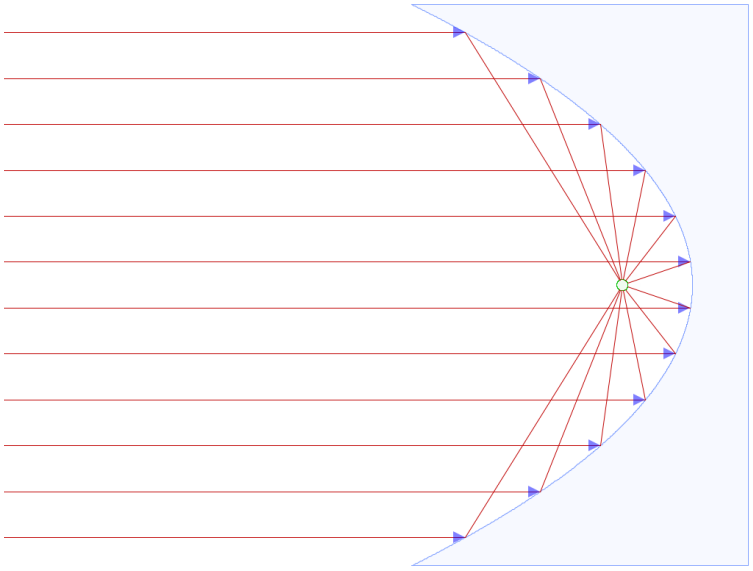

Figure 11: Parabolic mirror

Interestingly, hyperbolic curves find useful application in lenses, not mirrors, and parabolic curves have the opposite application. More recent ORT versions support an exact parabolic curve model, which in most cases differs academically from the above example, except when designing telescope mirrors.

3.2.3 Conic sections

Before describing how one models spherical, hyperbolic and parabolic optical elements, we need to digress into an explanation of conic sections and how they relate to these curves.

A hyperbola is one of three classes of conic section[19]. As it turns out, it's productive to think of circles and ellipses (with eccentricity[20] $\epsilon < 1$), parabolas ($\epsilon = 1$), and hyperbolas ($\epsilon > 1$) as representing open or closed curves of intersection between a plane and a cone in three dimensions. Here's an example – an ellipse as a conic section:

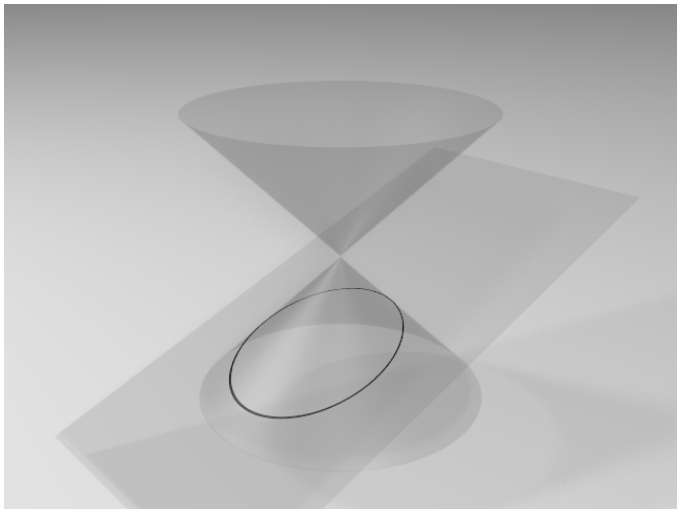

Figure 12: Elliptical conic section

Here's a commonly seen kind of elliptic curve:

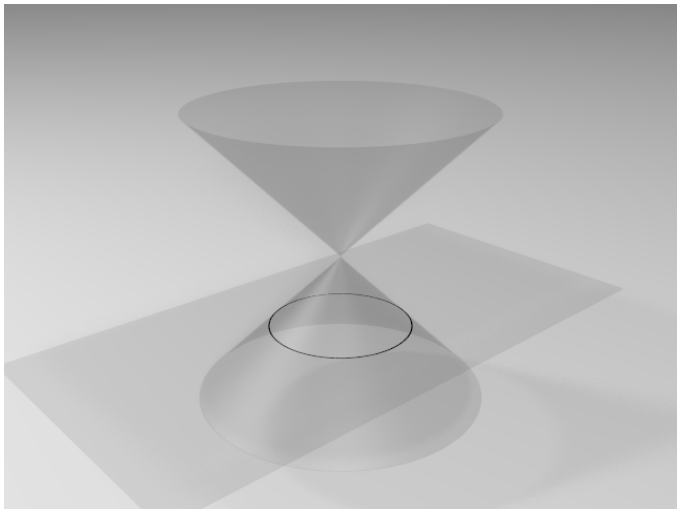

Figure 13: Circular conic section

Remember that a circle is an ellipse with an eccentricity of zero ($\epsilon = 0$). Ellipses have eccentricities between zero and one, parabolas have an eccentricity of precisely one, and hyperbolas have eccentricities greater than one. Here's a parabolic conic section:

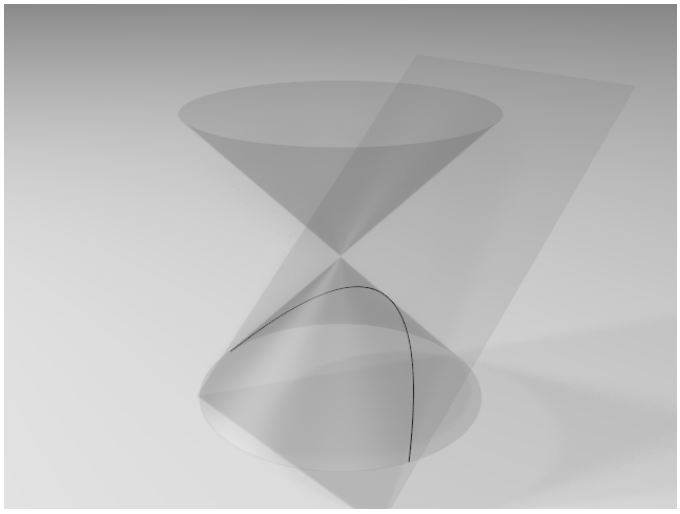

Figure 14: Parabolic conic section

And a hyperbola:

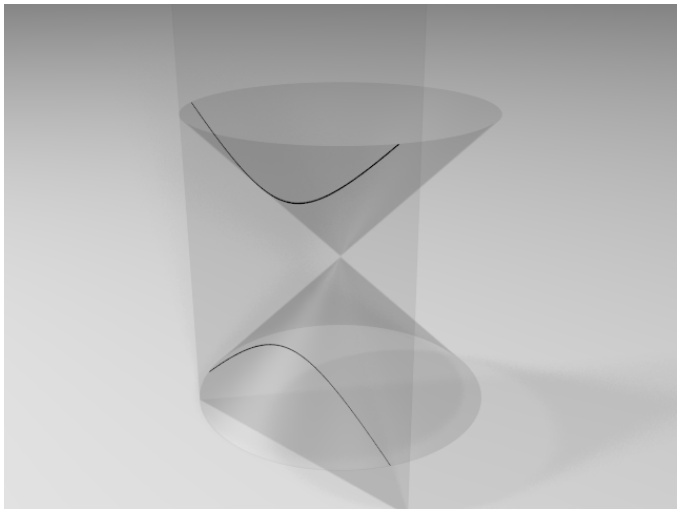

Figure 15: Hyperbolic conic section, $z = .5$

Here's an image for those who prefer a true three-dimensional view, and who happen to have a pair of anaglyphic[21] (red-blue) glasses lying about:

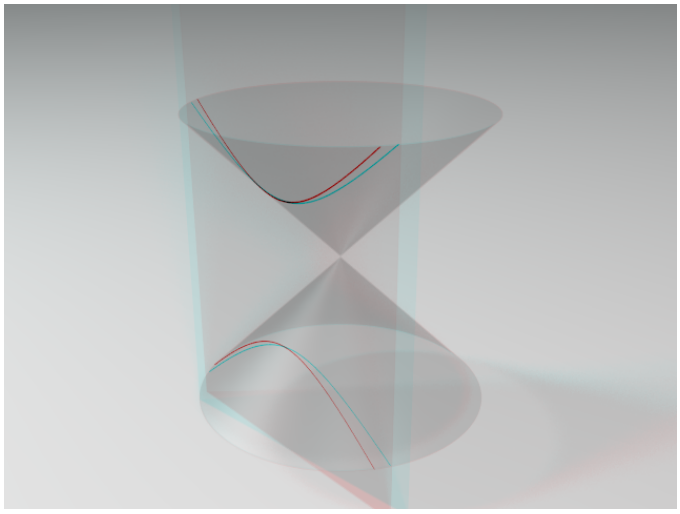

Figure 16: Hyperbolic conic section (anaglyph), $z = .5$

Ellipses are closed curves, while parabolas and hyperbolas are open curves – they extend to infinity without closing. Interestingly, these terms are used to describe planetary orbits, which, depending on their kinetic and potential energy, are classed as ellipses, parabolas or hyperbolas. An ellipse is a stable planetary orbit, a parabola is the path taken by a body possessing exactly escape velocity (its orbital path never closes, but its velocity declines and becomes zero at an infinite distance), and a hyperbola is the path taken by a body with greater than escape velocity.

I ask my readers to imagine what would happen if the intersecting plane were to be moved toward the apex of the cones. It turns out that the resulting curve is still hyperbolic, but its shape becomes more acute, and finally, at z = 0, it becomes triangular:

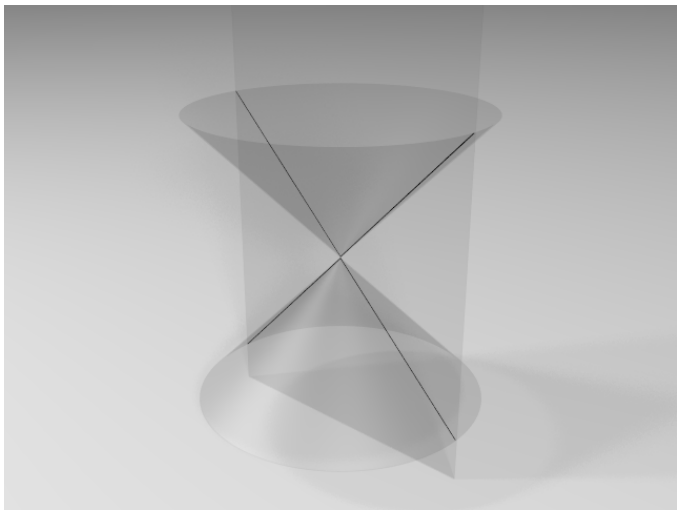

Figure 17: Hyperbolic conic section, $z = 0$

Figure 17 shows that, by changing the distance between the plane and the apex of the cones, one can control the shape of the hyperbolic curve. This point becomes important as we move into a description of hyperbolic modeling, so let's take a moment to orient ourselves – the intersecting plane's motion is in the third or Z dimension:

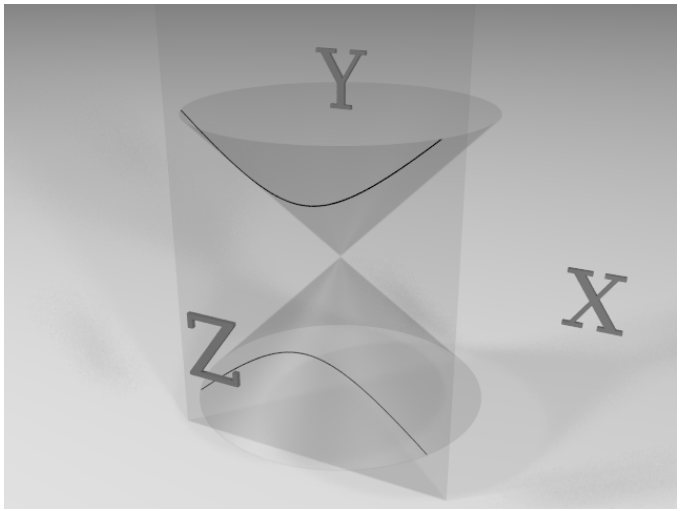

Figure 18: 3D orientation diagram

Here's the same image for those with anaglyphic glasses:

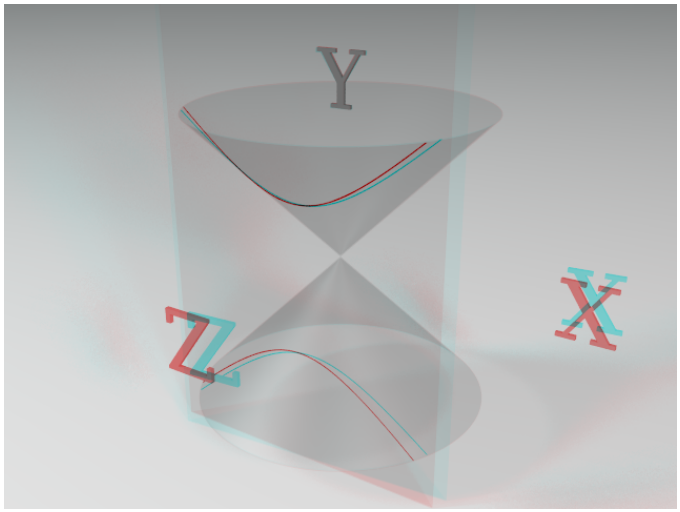

Figure 19: 3D orientation diagram (anaglyphic)

In the next section, the names of the dimensions shown in Figures 18 and 19 will be used regularly, so remember that X is the left-right dimension, Y is the up-down dimension, and Z is the third dimension, which can be thought of as an increase or decrease in radial distance.

3.2.4 Modeling challenge

ORT models hyperbolic lenses in such a way that the lenses are easy to explain and characterize, but this external simplicity conceals some internal complexity. Here are the goals set out for the hyperbolic modeling task:

- To base a hyperbolic lens model on the simplest mathematics.
- To be able to describe the resulting lenses with a minimum of mathematical complexity, so that they might actually be built.
- To solve a number of conflicting issues, among which are finding points of intersection between lines and hyperbolic curves (not difficult), after the curves have been translated in such a way that they model a solid lens, one with specific user-controlled properties (not easy).
- To model a lens whose dimensions (height and width) can be defined independently of the chosen hyperbolic curvature. Not easy at all.

3.2.5 Ellipse-line intersection

In a preliminary step, we write equations able to find points of intersection between an ellipse and a line[22]. The first equation describes an ellipse in terms of Cartesian[23] coordinates x, y and the ellipse's semiaxes a, b:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \tag{3.5}$$

The next equation describes an intersecting line:

$$y = \frac{y_0}{x_0} x \tag{3.6}$$

For equations (3.5) and (3.6):

- x: A horizontal position in Cartesian coordinates
- y: A vertical position in Cartesian coordinates
- a: The length of the ellipse's horizontal semiaxis
- b: The length of the ellipse's vertical semiaxis
- $x0$: The X coordinate of a point on the intersecting line
- $y0$: The Y coordinate of a point on the intersecting line

To clarify, the intersecting line is defined as passing through $(0, 0)$ and $(x0, y0)$.

If the above equations are solved simultaneously and for both x and y, they produce equations for two points of intersection (a total of four resulting values, two for x and two for y):

$$x = \pm \frac{abx_0}{\sqrt{b^2 x_0^2 + a^2 y_0^2}} \tag{3.7}$$

$$y = \pm \frac{aby_0}{\sqrt{b^2 x_0^2 + a^2 y_0^2}} \tag{3.8}$$

Here's an example outcome that uses equations (3.7) and (3.8) to compute points of intersection with the arguments $a = 3, b = 2, x0 = 5, y0 = 1$:

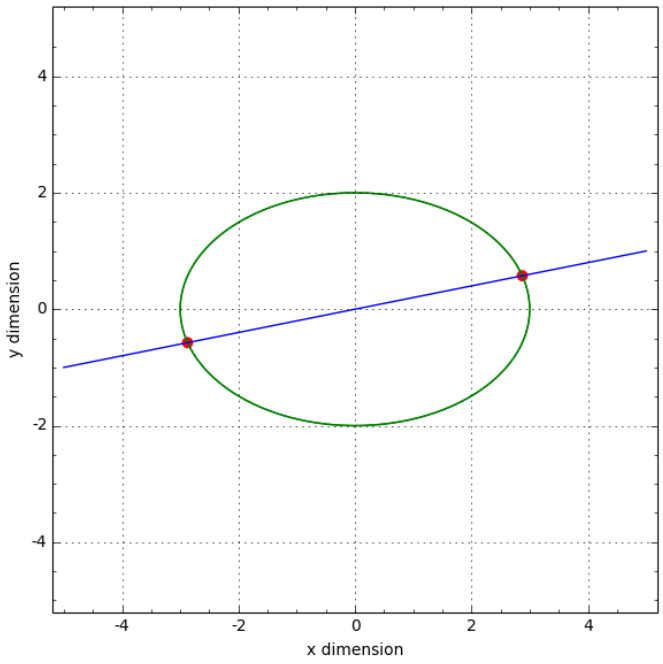

Figure 20: Ellipse-line intersection

The points of intersection (the two roots of the quadratic equations that define the ellipse) are marked with red dots. Notice about Figure 20 that the ellipse's width is equal to $2a$ and its height is equal to $2b$, the meaning of "semiaxis" in this context. This will be important to remember later on.

3.2.6 Hyperbolic-line intersection

In the next step, we rewrite equation (3.5) to define a hyperbola instead of an ellipse. The new equation:

$$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1 \tag{3.9}$$

Notice about the new equation that it's identical to equation (3.5) except that the terms are subtracted from each other. Here are the derived equations when prior equations (3.9) and (3.6) are (as before) solved simultaneously and for both x and y:

$$x = \pm\sqrt{\frac{1}{b^2x_0^2 - a^2y_0^2}}abx_0 \tag{3.10}$$

$$y = \pm\frac{aby_0}{\sqrt{bx_0 + ay_0}\sqrt{bx_0 - ay_0}} \tag{3.11}$$

Here's an example result, using the same values as in the previous example:

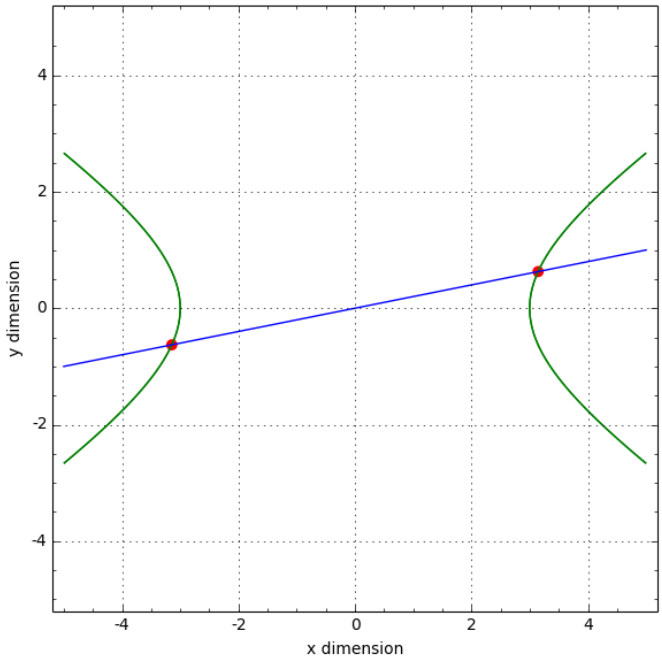

Figure 21: Hyperbola-line intersection

In Figure 21, notice that the vertices of the hyperbolic curves (i.e. their closest approach to the origin) are located at $\pm a$.

3.2.7 Z coordinate

First and perhaps most important, the value entered into the ORT "Hyperbolic Curvature" entry field is a bisecting plane's coordinate in the Z dimension, through a unit hyperbola (a hyperbola with a slope of 1). This arrangement is meant to simplify the technical description of a hyperbolic lens modeled by ORT.

In the previous section we showed that hyperbolic curvature is controlled by the intersecting plane's position in the Z dimension. With this in mind, and with the intent to write a more general equation, let's rewrite equation (3.5), eliminate the scaling variables a, b and add a variable that specifies a position in the Z dimension:

$$x^2 - y^2 = z \tag{3.12}$$

In this simplified equation, z specifies the location of the intersecting plane in the Z dimension (see Figure 18). We can use this equation to explore the relationship between plane position and curvature:

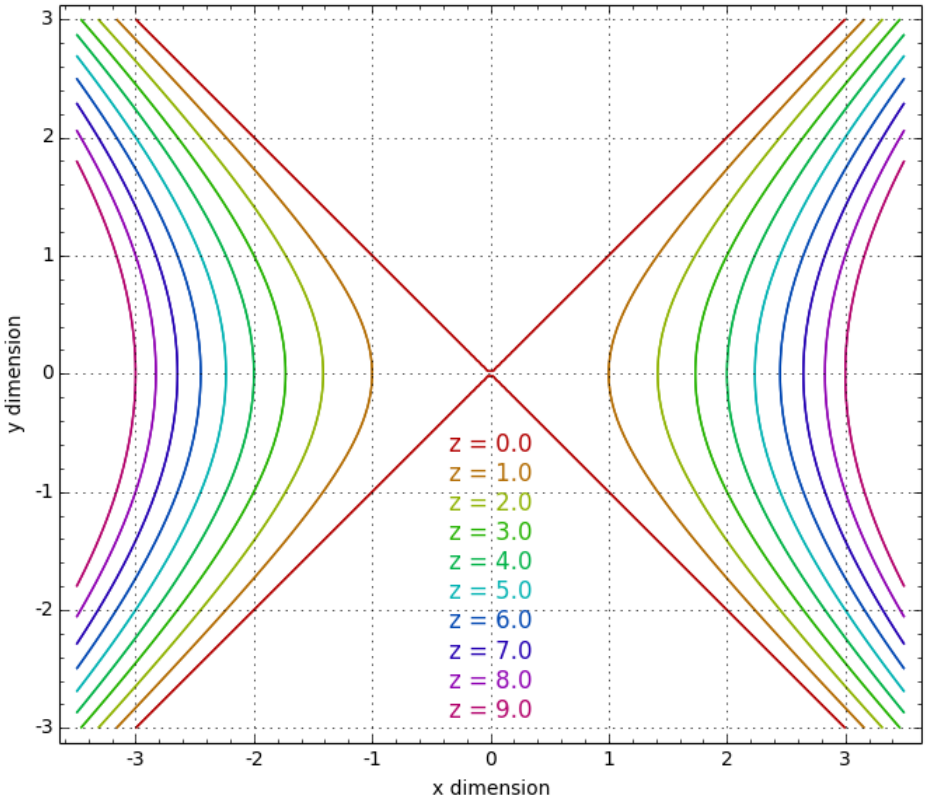

Figure 22: Hyperbolic curvature as a function of z

I emphasize this point because, to successfully model hyperbolic lenses with different amounts of curvature, we need to be able to control the position of the intersecting plane in the Z dimension, and the z variable in equation (3.12) provides this ability.

For a practical solution we need functions able to describe the profile of a hyperbolic lens, and we need a function that provides the first derivative[24] of that profile, or rate of change, used to compute surface angles, a requirement for Snell's Law calculations.

Here are example functions derived from equation (3.12) that produce hyperbolic profiles and first derivatives of those profiles:

- Profile:

$$x = \pm\sqrt{y^2 + z} \tag{3.13}$$

- First derivative (surface curvature):

$$x = \pm\frac{y}{\sqrt{y^2 + z}} \tag{3.14}$$

Here are example outcomes for these functions:

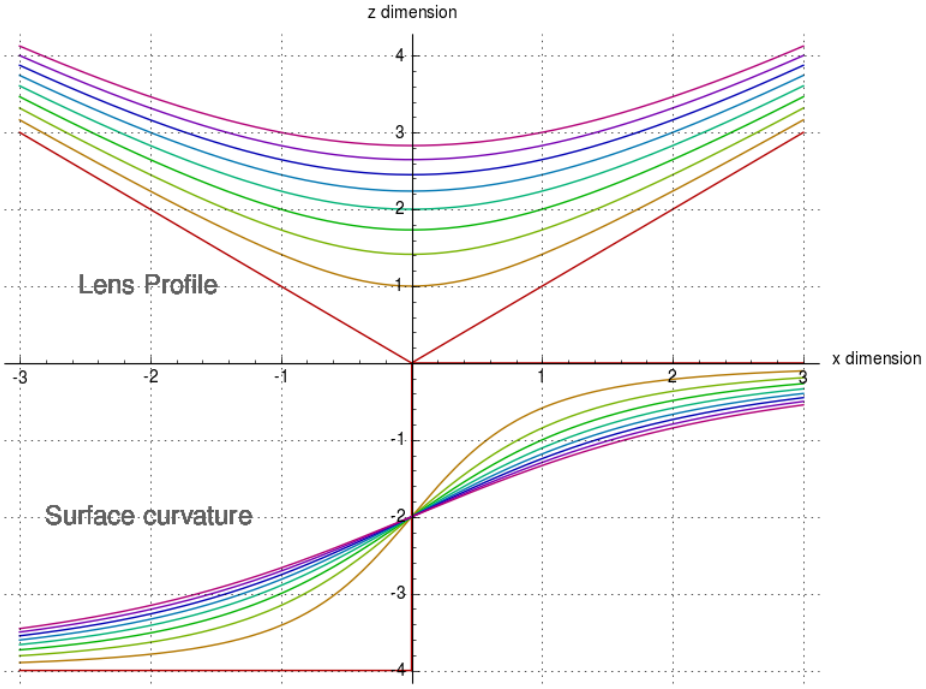

Figure 23: Hyperbolic curvature and its derivative as a function of z

It can be seen that, as the Z coordinate moves away from zero, the surface profile becomes less extreme and the first derivative declines in proportion. An ideal hyperbolic lens equation would provide a way to control this curvature without disrupting any other properties such as lens size.

3.2.8 Hyperbolic model

The ultimate goal is a mathematical treatment that will allow a modeled lens that can be scaled and positioned like a spherical lens, but that has hyperbolic curvature:

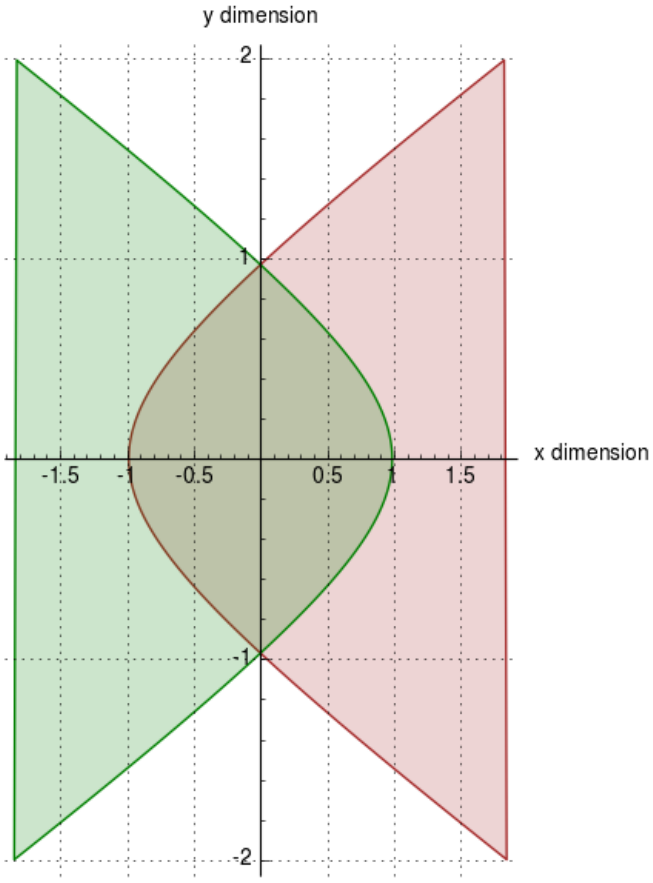

Figure 24: Hyperbolic overlapping curves

Having achieved the result shown in Figure 24, we should be able to compute points of intersection between lines and the modeled lens, just as with a spherical lens:

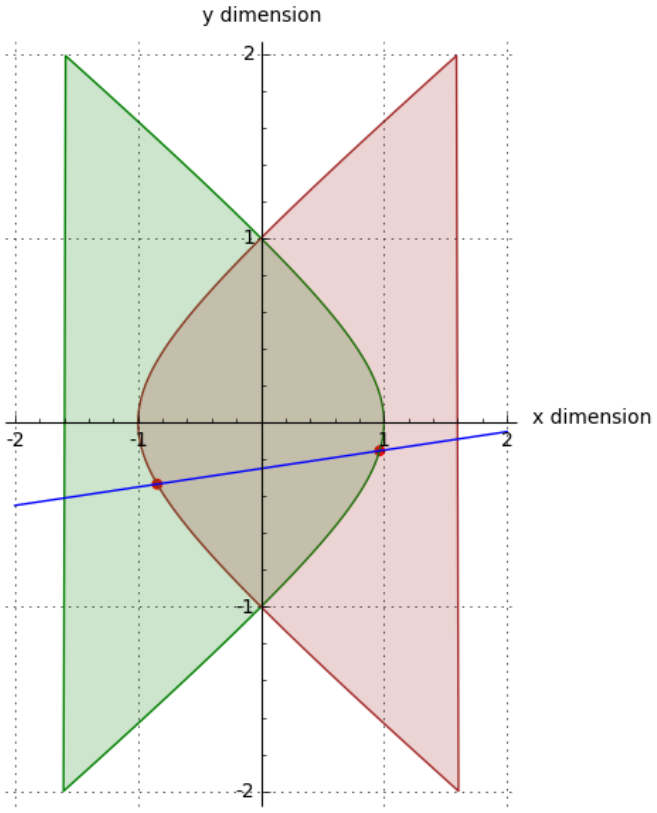

Figure 25: Hyperbolic intersection

Notice about figures 24 and 25 that the modeled lens has normalized dimensions – its width and height are equal to $\pm$ 1 unit. This means the basic model can be scaled to any required size because, independent of its curvature, the model has uniform and consistent dimensions. Achieving this result requires substantial modification of the earlier hyperbolic equations, as well as some algorithmic treatment of intermediate results.

Taking as our starting point the simplified equation (3.12) responsible for the above curvature examples, we add variables that define the x, y origin point, to allow lines to cross through points other than $(0, 0)$:

$$(x + p_x)^2 - (y + p_y)^2 = z \tag{3.15}$$

Finally, we add one more variable to control the curve's scale independent of its curvature:

$$(x + p_x)^2 - (m(y + p_y))^2 = z \tag{3.16}$$

To summarize these values:

- x, y: These are the unknown result values for a line-hyperbola intersection calculation.
- p_x, p_y: This specifies a *position* so that lines may intersect the lens at positions other than the origin.
- m: This value controls the overall *scale* of the lens, independent of the other values.
- z: This moves the intersecting plane in the Z axis, providing control over the hyperbola's *curvature* as shown in the earlier figures.

We now define an equation that describes an intersecting line:

$$sx = y \tag{3.17}$$

In equation (3.17) the s value represents the slope of a line whose points of intersection are being sought. For a given line defined by $x_1, y_1 - x_2, y_2$:

$$s = \frac{y_2 - y_1}{x_2 - x_1} \tag{3.18}$$

In practice, for a given line whose points of intersection are sought and that is defined by $x_1, y_1 - x_2, y_2$, one coordinate pair is assigned to p_x, p_y of equation (3.16) and the s value computed by equation (3.18) is applied to equation (3.17). This defines the line with respect to the hyperbolic profiles of the modeled lens.

At this point we've prepared the basic equations for a modeled hyperbola whose position and shape we control. As before, to model our lens we need equations for x-axis profile and first derivative of profile, based on solutions to equation (3.16):

- Profile:

$$x = -p_x \pm \sqrt{m^2 p_y^2 + 2m^2 p_y y + m^2 y^2 + z} \tag{3.19}$$

- First derivative (surface curvature):

$$x = \pm \frac{m^2 p_y + m^2 y}{\sqrt{m^2 p_y^2 + 2m^2 p_y y + m^2 y^2 + z}} \tag{3.20}$$

Finally, to compute points of intersection between a line and the hyperbolic curve, a simultaneous solution for equations (3.16) and (3.17) is acquired for both x and y, but unlike the earlier examples and because of the number of terms in the new equations, the derivations are a bit more complex:

$$x = \frac{1}{s\left(m^2 s^2 - 1\right)} \left(-m^2 p_y s^2 + p_x s \pm \sqrt{s^2 \left(m^2 p_x^2 s^2 - 2m^2 p_x p_y s + m^2 p_y^2 - m^2 s^2 z + z \right)} \right) \tag{3.21}$$

$$y = \frac{1}{m^2 s^2 - 1} \left(-m^2 p_y s^2 + p_x s \pm \sqrt{s^2 \left(m^2 p_x^2 s^2 - 2m^2 p_x p_y s + m^2 p_y^2 - m^2 s^2 z + z \right)} \right) \tag{3.22}$$

The m value in the above equations, responsible for normalizing the curve's scale, is derived from a given z argument in this way:

$$m = -\sqrt{2\sqrt{z} + 1} \tag{3.23}$$

Having acquired points of intersection from equations (3.21) and (3.22), we can move on to a more conventional optical computation such as a Snell's Law result.

4 Conclusion

4.1 Role of computers in optics

In a modest way ORT shows the power of modern, computer-aided, applied mathematics. In reading the literature of the optical field while preparing this article, I became impressed by the fact that, until recently, practical optics has relied on either no mathematical results or very crude results. Programs like ORT can greatly simplify optical design and prototyping, as well as increase the amount of information available about a given design or configuration.

This isn't to say that practical lab testing has no role, but the threshold is higher in modern times – many kinds of preliminary experiments can (and should) be conducted using a computer instead of an optical bench, for both economic and practical reasons. And to a greater degree than in the past, modern optical manufacturing is controlled by computers, which increases the value of accurate mathematical models.

Put simply, these changes mean workers in the field of optics need to learn about computers or be replaced by those who have done so.

4.2 Tools

This article was composed on Texmaker[25], an excellent, free, LaTeX editor.

The mathematical results used in ORT derive from many sources. In the project's early years Mathematica[26] was the primary source for equation derivation, but over time its expense, and the requirement for frequent and costly updates to keep the program working, led me to explore alternatives. More recently I've begun using Sage[27], the iPython[28] environment, and the Python library SymPy[29] to produce results and derivations. One advantage of these newer tools is that they're all free. Another is that over time, very skilled and enthusiastic people create improvements and fix bugs.

Most of the graphic images were created in the Sage mathematical environment (discussed below), but the conic section images were created using Pov-Ray[30], a ray tracing program of the other kind (the kind meant to create pretty pictures).

4.2.1 SymPy

I've been recenty exploring SymPy's symbolic math abilities – it has occurred to me that I should be able to write a single Python script to derive all required equations as well as format results for both the ORT Java source (as Java source code) and for articles like this one (as LaTeX text). The advantage is that I can generate all these results in a Python script that accepts a set of preliminary equations I specify (and that change over time) and automatically produce all the required derivations and text forms.

Here's an example that shows how powerful and accessible SymPy is. In this example SymPy takes the simple preliminary equations required for hyperbolic lens analysis (equations (3.16) and (3.17)) and produces all the forms required by ORT and by articles like this.

```
from sympy import *
var('x y p_x p_y z m s')
exp1 = (x+p_x)**2 - (m*(y+p_y))**2 - z
exp2 = s*x-y
qa = solve((exp1,exp2),x,y)
qa = map(simplify,qa)
print("x = " + latex(qa[0][0]))
```

$$x = \frac{1}{s\left(m^2 s^2 - 1\right)}\left(-m^2 p_y s^2 + p_x s - \sqrt{s^2\left(m^2 p_x^2 s^2 - 2m^2 p_x p_y s + m^2 p_y^2 - m^2 s^2 z + z\right)}\right) \tag{4.1}$$

That's one of the four forms of the line-hyperbolic intersection equation, the "heavy lifting". Now for the profile form:

```
qb = solve(exp1,x)
qb = map(simplify,qb)
print("x = " + latex(qb[0]))
```

$$x = -p_x - \sqrt{m^2 p_y^2 + 2m^2 p_y y + m^2 y^2 + z} \tag{4.2}$$

Now the derivative form, used for surface normal angle calculations:

```
qc = diff(qb[0],y)
qc = simplify(qc)
print("x' = " + latex(qc))
```

$$x' = -\frac{m^2\left(p_y + y\right)}{\sqrt{m^2 p_y^2 + 2m^2 p_y y + m^2 y^2 + z}} \tag{4.3}$$

Simple, efficient, repeatable. I can create Java forms and LaTeX forms with a few keystrokes. To make a change, I need only edit a few lines and run the script again.

Deriving equation (3.23) was a bit more complicated – it's the equation that normalizes the lens size for any Z coordinate value. To produce this result, I need to assign a SymPy "solve()" result to a function, then set particular values for that function, then solve for those particular arguments. The idea is that the m value should produce a lens semidiameter of 1.0 regardless of the z argument that represents the Z dimension coordinate. Therefore to find the required m value for any z value, I needed to derive a special form of the profile equation. Here's how I presented the problem to SymPy, using the results acquired above:

```
pe = lambdify((y,z,p_x,p_y,m), qb[1],'sympy')
qd = solve(pe(0,z,0,0,m)-pe(1,z,0,0,m)-1,m)[2]
print("m = " + latex(qd))
pf = lambdify( z, qd,'sympy')
z = 10
print(N(pf(z)))
```

$$m = -\sqrt{2\sqrt{z}+1} \tag{4.4}$$

```
2.70639156818387
```

An earlier version of ORT acquired the m value by iteration, using a slow, inefficient binary search. Eventually I decided to look for a closed-form solution, but this required me to figure out how to express the problem in a way comprehensible to SymPy's equation solver. Simply put, I required an m value that produced an exact differential of one unit between the edge and center of a lens with any curvature, any z value.

When I first saw the result ($m = -\sqrt{2\sqrt{z}+1}$) I thought the solver had failed and was emitting bogus math (which sometimes happens) – I had rarely seen one square root nested within another square root before. But a few quick tests comparing results with the iterative method (see example result above) proved that the odd-looking equation was exactly right, indeed because it produced immediate numerical results of 16-place accuracy, it was able to draw lenses more precisely than the prior iterative method. The result was that I was able to abandon the old method and see ORT run faster and more reliably.

As time passes, SymPy is becoming more powerful and able to solve more kinds of equations, consequently if I need an equation generator that's reliable and repeatable, if I need a result that doesn't change until I want it to (such as the equation set that defines ORT), it's definitely the tool of choice.

4.2.2 IPython

The IPython environment is more interactive and immediate than SymPy (even though it relies on SymPy internally), so for rapid experimental work it excels. In its current incarnation it's hosted in a browser. Here's what it looks like in action:

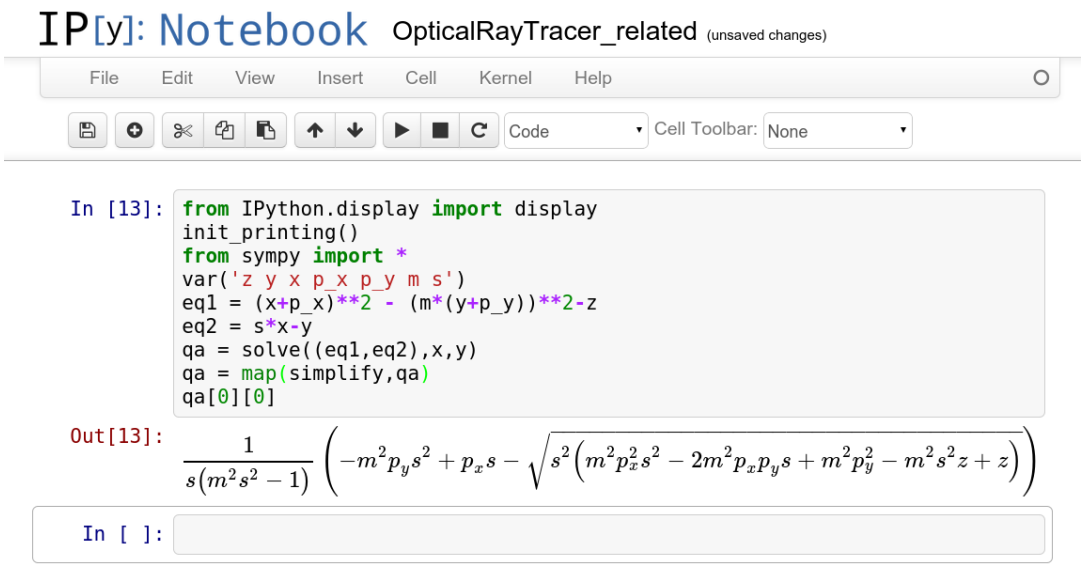

Figure 26: IPython work session

The difference between IPython and SymPy is that SymPy is a symbolic library accessible through Python scripting, therefore it's the right choice for a script that produces a given kind of result perpetually, but IPython is an interactive environment intended for the kind of experimentation that might lead to a more permanent solution.

4.2.3 Sage

Sage is a more powerful mathematical environment than either SymPy or IPython, one that manages certain symbolic issues more cleanly than SymPy or IPython do, but it's very large and for structural reasons pretty much limited to operating under Linux. Ordinarily I would be happy about that –I'm a longstanding advocate of Linux – but in this case the program's Linux base represents a limitation because it prevents wider adoption and possibly fresh ideas.

Earlier in the Sage project there were a few efforts to create a native Windows install package, but as Sage grew larger these efforts gradually became unworkable. Now that Sage's typical installed size exceeds 4 gigabytes, now

that it won't run under Windows without a virtual host environment and a rather complex installation procedure, it's no longer an end-user program.

Notwithstanding these issues, many of this article's graphic images were created in Sage. Sage plotting is easy and flexible compared to its alternatives, and Sage is immediate in much the way IPython is.

4.2.4 Pov-Ray

I can't end a list of development tools without mentioning Pov-Ray. I used Pov-Ray to create the three-dimensional conic section graphics, including the anaglyphic images. Pov-Ray has been around for a long time, and one would think by now it would be paired with a reliable, open-source, cross-platform, undead modeling tool. There's a nice tool called KPovModeler[31], unfortunately no one is maintaining it, which means over time it becomes increasingly hard to compile it and get it running.

References

[1] OpticalRayTracer – A virtual lens/mirror design workshop.
[2] Ray Tracing (physics) – A physics analysis method that examines the paths of rays through various media.
[3] Pixel – picture element, part of a computer display representing a single screen location.
[4] Snell's Law – a cornerstone of optical science and design.
[5] Index of Refraction – a convenient measure of the speed of light in a given material.
[6] Derivation of Refraction Formulas – author Paul S. Heckbert
[7] Refraction – a change in the direction of a light wavefront, often by means of a lens.
[8] Total Internal Reflection – a case where Snell's Law dictates that a light beam will be reflected internally rather than exiting the medium.
[9] Circle-line intersection – a mathematical method for locating points of intersection between a circle and line.
[10] Cartesian Coordinates – orthogonal coordinates that define points on a plane or in higher dimensions.
[11] Laws of Reflection – a description of the physical circumstances of planar reflection.
[12] Dispersion (Optics) – a property of many optical materials in which the index of refraction is wavelength-dependent.
[13] Abbe number – a numeric value proportional to a material's spectral dispersion.
[14] Fraunhofer Lines – spectral lines measured in the early days of observational astronomy, now often used as physical constants in optical work.
[15] Sellmeier equation – an equation that relates Abbe numbers to optical dispersion.
[16] SChot Optical Glass – Page 12 for BK7 glass properties
[17] Schott BK7 Refractive Index info – Source for BK7's Abbe number and other properties.
[18] Sperical Aberration – A persistent and unavoidable defect in spherical lenses.
[19] Conic section – A geometric interpretation of certain mathematical notions.
[20] Ecentricity (Mathematics) – a defining property of all conic sections.
[21] Anaglyph 3D – A method for producing stereoscopic images by means of color filters.
[22] Ellipse-Line Intersection – Equations able to find points of intersection between an ellipse and a line
[23] Cartesian coordinate system – A system of spatial orientation using orthogonal coordinates.
[24] Derivative (mathematics) – a rate of change in a function, and a cornerstone of Calculus.
[25] Texmaker – An excellent, free, LaTeX editor.
[26] Mathematica – An excellent, but expensive, mathematical environment.
[27] Sage (mathematics) – A free open-source mathematics software system.
[28] iPython – A Python-based interactive computing environment.
[29] SymPy – A Python library for symbolic mathematics.
[30] Pov-Ray – A powerful raytracing tool of the other kind (for generating pretty images).
[31] KPovModeler – A nice modeling tool for Pov-Ray, now unfortunately abandoned.