Simple Two-D Raytrace Algorithm

Tom Murphy

April, 2011

1 Introduction

This document describes the mathematical formulation implemented in the raytracing program(s) that we will use in our optical investigations. This particular algorithm is at the heart of each program, performing the mathematical operations relevant to each surface encounter. The main program does the book-keeping of initializing rays and propagating them through the system. The algorithm described here simply computes the intersection of a ray with a (generally) curved surface, computes the surface normal, and applies Snell's law to arrive at the new ray position and direction.

Following conventional practice, we will pick the z-axis as our optical axis, and use y as the cross-direction (vertical). Coordinate pairs will often be expressed as (z, y), so that drawn on a piece of paper, z acts like our more familiar x coordinate. This choice may be slightly annoying, but provides a more sensible upgrade path for moving to 3-D raytracing, if desired.

2 Defining the Surfaces

Most of the time we will use spherical surfaces, but let's allow aspherical surfaces as well—conic sections in particular. We will assume the optical axis of the system to be the z-axis, and only concern ourselves with rays in the y-z plane.

A general form for a conic section oriented axially along the z-axis, whose vertex is at z_v and has radius R at that point, is:

$$(z-z_v)^2(K+1) - 2R(z-z_v) + y^2 = 0, \tag{1}$$

where K is the conic constant. A circle has $K = 0$; $K > 0$ describes oblate ellipses; $-1 < K < 0$ describes prolate ellipses; $K = -1$ describes a parabola; and $K < -1$ describes hyperbolae (see Fig. 1). Note that the only place coordinates appear without the square also has an R. This means that flipping the sign of R also flips the curve along the z-axis. Positive R opens to the right, and negative R opens to the left. See the appendix for how to relate this to more familiar forms for the conic sections.

Since we will eventually want to describe surface normals for the application of Snell's Law, we go ahead and take the derivative here. We will go after $\frac{\partial z}{\partial y}$, but could also work with $\frac{\partial y}{\partial z}$. To start, we will develop an expression for $z(y)$ by completing the square in $(z-z_v)$. To do this, we first divide Eq. 1 by $(K+1)$, then add and subtract the constant that will complete the square.

$$(z-z_v)^2 - 2\frac{R}{K+1}(z-z_v) + \frac{y^2}{K+1} = 0$$

$$\left[(z-z_v)^2 - 2\frac{R}{K+1}(z-z_v) + \frac{R^2}{(K+1)^2}\right] - \frac{R^2}{(K+1)^2} + \frac{y^2}{K+1} = 0,$$

where the term in brackets is now a square, so that

$$\left[z - z_v - \frac{R}{K+1}\right]^2 = \frac{R^2}{(K+1)^2} - \frac{y^2}{K+1}. \tag{2}$$

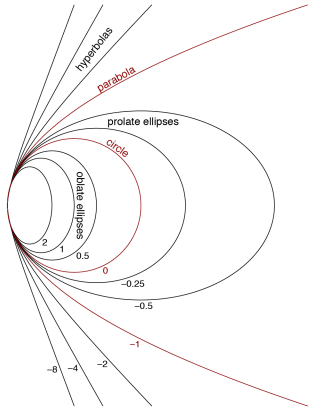

Figure 1: Various forms of conic section as a function of the conic constant, K.

Now it is straightforward to express z as a function of y:

$$z = z_v + \frac{R}{K+1} - \sqrt{\frac{R^2}{(K+1)^2} - \frac{y^2}{K+1}}, \tag{3}$$

where we have chosen the negative square root to place the curve at $z = z_v$ when $y = 0$. And finally we are in a position to evaluate derivatives:

$$z'(y) = \frac{y}{K+1}\left[\frac{R^2}{(K+1)^2} - \frac{y^2}{K+1}\right]^{-\frac{1}{2}}, \tag{4}$$

and while we're at it, we can check that the second derivative at the vertex (at $y = 0$) indeed gives us a radius of curvature of R. We simplify the derivative by noting that anything with a y out front will vanish in our evaluation at the vertex.

$$z''(y=0) = \frac{1}{K+1}\left[\frac{R^2}{(K+1)^2}\right]^{-\frac{1}{2}} = \frac{1}{R}$$

The predominance of the $K+1$ term in the denominators of these expressions indicate that we should take special care for the parabolic case when $K = -1$. For this, we are better off going back to Eq. 1 and re-deriving relationships in this simplified form.

3 Ray Intersection

We will define the ray segment, labeled by the subscript i as we must deal with multiple segments, as passing through the point (z_i, y_i), and traveling at an angle θ_i relative to the z-axis. It is assumed that the ray is traveling from left-to-right, and θ_i is positive if the ray's y-value increases as it travels to the right (to larger z-values). The ray therefore has a slope:

$$m_i = \tan\theta_i. \tag{5}$$

The ray segment's position (backward or forward) therefore satisfies

$$y = y_i + m_i(z - z_i). \tag{6}$$

The intersection of this ray segment with the optical curve of interest is given by the joint solution of Equation 1 and Equation 6. To this end, we substitute the y-value determined from Equation 6 into Equation 1 to get

$$y_i^2 + 2m_i(z-z_i)y_i + m_i^2(z-z_i)^2 - 2R(z-z_v) + (K+1)(z-z_v)^2, \tag{7}$$

for which we consolidate terms in z^2, z, and constants:

$$(K+1+m_i^2)z^2+2[m_iy_i-m_i^2z_i-R-(K+1)z_v]z+y_i^2-2m_iz_iy_i+m_i^2z_i^2+2Rz_v+(K+1)z_v^2=0,$$

then divide by $(K+1+m_i^2)$ to put in the form $z^2+2bz+c=0$, which has the familiar quadratic solution:

$$z=-b\pm\sqrt{b^2-c}, \tag{8}$$

with

$$b=\frac{m_iy_i-m_i^2z_i-R-(K+1)z_v}{K+1+m_i^2}, \tag{9}$$

and

$$c=\frac{(K+1)z_v^2+y_i^2+m_i^2z_i^2+2Rz_v-2m_iz_iy_i}{K+1+m_i^2}. \tag{10}$$

Now one simply inserts the solutions for b and c into Equation 8 to find the surface intersections. But notice there are two roots. This should be the case for a line intersecting a circle, ellipse, or hyperbola (recall there are two arcs): provided there is any non-tangent intersection there will be two. Which one do we want? For $K>-1$, we want the most negative root when $R>0$, and the most positive root when $R<0$ (see Fig. 1 for visual sense of $R>0$ case). For hyperbolae, we want the curve whose vertex is at z_v, and not the other curve. When $R>0$, the other curve will be at more negative values of z, so we must flip the rule established above in the case where $K<-1$.

So the rule becomes:

- if $K>-1$, $z=-b-\frac{R}{|R|}\sqrt{b^2-c}$;
- if $K<-1$, $z=-b+\frac{R}{|R|}\sqrt{b^2-c}$,

where the sign of R is used to pick which root to use.

Note that the math still works for $R\to\infty$ to represent a planar interface. However, a computer will have numerical roundoff problems, so it is best to detect $R>R_{max}$ and treat such cases as planar. The intersection solution in this case is a trivial calculation of Equation 6, with z set to the z-value of the plane.

4 Surface Normal

Now that we have established the (z, y) value of the ray intersection, we must determine the angle of the surface normal at this location. For the general case, we can use Equation 4, re-expressed as:

$$z'(y)=\frac{y}{\sqrt{R^2-(K+1)y^2}}\frac{R}{|R|},$$

where the last factor accommodates different signs for the radius. The surface normal is perpendicular to this slope (recall that a slope perpendicular to m is $-\frac{1}{m}$), but we want to compare to our ray, whose slope is expressed as $m=y'(z)$. So by the time we accommodate the coordinate change and flip signs, we find the surface normal in y-z land to be

$$m_\perp=\tan\theta_n=-\frac{y}{\sqrt{R^2-(K+1)y^2}}\frac{R}{|R|}. \tag{11}$$

For the special case of the parabola, $m_\perp=-y/R$.

5 Snell's Law

Relating the incident ray angle, θ_i to the outgoing ray angle, θ_{i+1}, using the surface normal angle, θ_n, obtained from Equation 11, we first define

$$\theta_{in}=\theta_i-\theta_n. \tag{12}$$

Then, we apply Snell's law:

$$n_i \sin\theta_{in} = n_{i+1} \sin\theta_{out}, \tag{13}$$

or explicitly,

$$\theta_{out} = \sin^{-1}\left(\frac{n_i}{n_{i+1}} \sin\theta_{in}\right). \tag{14}$$

And finally, we re-transform the out ray with respect to the surface normal:

$$\theta_{i+1} = \theta_{out} + \theta_{norm}. \tag{15}$$

6 The Final Ray

The new ray segment passes through the intersection found via Equation 8. We call the z solution z_{i+1}, so that $y_{i+1} = y_i + m_i(z_{i+1} - z_i)$. Together with the angle θ_{i+1} from Equation 15, we have a full description of the new ray in space. From this point, we can either iterate for the next surface, or describe where this ray intersects the z-axis, a selected z-plane, or another ray of choice.

7 Notes on Subtleties

Each intersection is computed without reference to whether the designated "pass-through" point is to the left or right of the surface. An example of how this can be bad is if a bi-convex (positive) lens is too thin, such that the spheres occur in the wrong order far from the optical axis. The ray will blithely intersect them in this wrong order, dutifully following Snell's Law, producing an unphysical ray path. A warning in the algorithm can alert you to any ray whose x-intersection jumps backward from one surface to the next.

An example of where this might be good is if you want to define your initial ray at the center of the lens rather than far out to the left, there is no penalty. For example, if I want a ray whose angle to the optical axis is 0.1 radians, and I want this ray to encounter the first lens (at $z = 0$, say) at a height of 5 mm, then I can specify the initial position as (0.0,5.0) with an angle of 0.1, rather than at (-100.0, -5.0) with an angle of 0.1. The math is simplified. Likewise, if I want the original ray to head for a specified point to the right of the optic (in the absence of the optic), then I simply specify the ray through that point. The calculation will "backtrack" to the appropriate first-surface intersection.

8 Computer Code and Example

The following Python listing traces a single ray through the optical system. Command line arguments are surface (lens system) filename, followed by the ray's initial z-position, initial y-position, and initial slope (tangent to the angle) relative to the optical axis. Optionally, the z-value of a final screen may be entered to probe the y-value at which the final ray pierces this screen (otherwise defaults to $z = 0$).

```
from math import *
import sys

def ray_step(in_ray, surface):
  out_ray = [0.0]*3

  # rename in_ray components for clarity
  z0 = in_ray[0]
  y0 = in_ray[1]
  thet0 = in_ray[2]

  # rename surface params for clarity
  n0 = surface[0]
  z_v = surface[1]
  R = surface[2]
  K = surface[3]
  n_new = surface[4]
```

```
  # establish sign variable for sign flips
  Rsign = R/fabs(R)
  Ksign = 1.0
  if (K < -1.0):
    Ksign *= -1.0

  # compute solution to quadratic formula
  m = tan(thet0)
  if (K == -1.0 and m == 0.0):  # would make denom = 0
    z_new = z_v + y0*y0/(2.0*R) # parabolic case
  else:
    denom = 1.0 + K + m*m
    b = (m*y0 - m*m*z0 - R - (K+1)*z_v)/denom
    c = (y0*y0 + m*m*z0*z0 - 2.0*m*z0*y0 + 2.0*R*z_v + (K+1)*z_v*z_v)/denom
    z_new = -b - Rsign*Ksign*sqrt(b*b - c)
  if (fabs(R) > 1.0e10):
    z_new = z_v # planar case

  y_new = y0 + m*(z_new - z0)
thet_norm = atan(-y_new*Rsign/sqrt(R*R - (K+1)*y_new*y_new))
  thet_in = thet0 - thet_norm
  thet_out = asin(n0*sin(thet_in)/n_new)
  thet_new = thet_out + thet_norm
  #print "%f %f %f %f" % (z_v,z_new,y_new,thet_norm)

  if (z_new < z0):
    print "WARNING: ray jumped backwards!''

out_ray[0] = z_new
  out_ray[1] = y_new
  out_ray[2] = thet_new


  return out_ray

narg = len(sys.argv)
z = []
y = []
thet = []
n = []
z_vert = [0.0]
R = [0.0]
K = [0.0]
in_ray = [0.0]*3
out_ray = [0.0]*3
surf_params = [0.0]*5
screen_pos=0.0

if (narg > 4):                  # must have at least these four
  filename = sys.argv[1]
  z.append(float(sys.argv[2]))
  y.append(float(sys.argv[3]))
  slope = float(sys.argv[4])
else:
  print "Must supply lens_file_name z0, y0, slope arguments"
  sys.exit()

if (narg > 5):          # optionally, put a screen somewhere
  screen_pos = float(sys.argv[5])


thet.append(atan(slope))        # input slope --> angle in radians

lens_file = open(filename,'r'); # grab lens surface parameters
```

```
n_surf = int(lens_file.readline().strip())      # number of surfaces (1st line)
n.append(float(lens_file.readline().strip()))   # initial refr. index (2nd line)
current_z = 0.0
for i in range(n_surf):                 # and n_surf additional lines...
  # read in lens file and verify results
  line = lens_file.readline()
  Slist = line.split()
  n.append(float(Slist[0]))
  z_vert.append(float(Slist[1]))
  R.append(float(Slist[2]))
  K.append(float(Slist[3]))
  current_z += z_vert[i+1]
  print "Surface %d has n = %f, z_vert = %f, radius = %g, K = %f" % \
         (i+1,n[i+1],current_z,R[i+1],K[i+1])

lens_file.close()

print "Ray 1 has z = %f; y = %f; thet = %f" % (z[0],y[0],thet[0])

current_z = 0.0
for i in range(n_surf):         # now propagate surface-at-a-time
                                # begin ray propagation for loop
  # populate surface parameters array
  current_z += z_vert[i+1]
  surf_params[0] = n[i]
  surf_params[1] = current_z
  surf_params[2] = R[i+1]
  surf_params[3] = K[i+1]
  surf_params[4] = n[i+1]

  # populate in_ray array for +y ray
  in_ray[0] = z[i]
  in_ray[1] = y[i]
  in_ray[2] = thet[i]

  # carry out calculation
  out_ray = ray_step(in_ray,surf_params)

  # stow out_ray into approp. arrays
  z.append(out_ray[0])
  y.append(out_ray[1])
  thet.append(out_ray[2])

  print "Ray %d has z = %f; y = %f; thet = %f" % \
              (i+2,z[i+1],y[i+1],thet[i+1])

# final +y ray parameters
zf = z[n_surf]
yf = y[n_surf]
thetf = thet[n_surf]
mf = tan(thetf)

# compute intercepts
screen_intercept = yf + mf*(screen_pos - zf)
z_intercept = zf - yf/mf

print "+Ray intercepts screen at (%.3f, %f); z-axis at (%f, 0.0)" % \
        (screen_pos,screen_intercept, z_intercept)
```

8.1 Example Input File

The input file for a beam expander looks like:

```
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
```

The first line indicates the number of surfaces. The second line indicates the initial refractive index. Each following line represents a surface. For each, the fields are as follows:

- the refractive index that one passes into by traversing the surface
- The z-displacement from the last vertex (initialized to zero)
- the radius of curvature: positive means center of curvature is to the right
- the K-constant describing the shape

In this example, the vertices lie at z-values of -1.5, 1.5, 200.0, and 204.0, respectively (the numbers in the file represent separation from the last vertex). The first lens is a negative lens, and the second a positive lens. Surfaces are either spherical (all have $K = 0.0$) or planar (huge R).

8.2 Example Output

Example output of the code looks like the following:

```
./one_2d beam_expanderK -10.0 5.0 0.0 300.0

Surface 1 has n = 1.500000, z_vert = -1.500000, radius = -100, K = 0.000000
Surface 2 has n = 1.000000, z_vert = 1.500000, radius = 1e+20, K = 0.000000
Surface 3 has n = 1.500000, z_vert = 200.000000, radius = 1e+20, K = 0.000000
Surface 4 has n = 1.000000, z_vert = 204.000000, radius = -201.55, K = 0.000000
Ray 1 has z = -10.000000; y = 5.000000; thet = 0.000000
Ray 2 has z = -1.625078; y = 5.000000; thet = 0.016681
Ray 3 has z = 1.500000; y = 5.052135; thet = 0.025023
Ray 4 has z = 200.000000; y = 10.020332; thet = 0.016681
Ray 5 has z = 203.747637; y = 10.082853; thet = -0.000013
Ray intercepts screen at (300.000, 10.081571); z-axis at (756938.486327, 0.0)
```

The ray starts at $(-10, 5)$, initially horizontal. A screen has been placed at $z = 300$. The final ray leaves the lens at (203.747637, 10.082853) at an angle of -0.000013. This ray will cross the z-axis at $z = 756938.5$ (far away: nearly level).

Appendix: Relating to Familiar Conic Forms

Starting with Eq. 1:

$$(z-z_v)^2(K+1) - 2R(z-z_v) + y^2 = 0,$$

and completing the square as we did before for Eq. 2, we end up with

$$\left[z - z_v - \frac{R}{K+1}\right]^2 + \frac{y^2}{K+1} = \frac{R^2}{(K+1)^2}.$$

Now for the sake of two-dimensional familiarity, we will rename the term in brackets x, which is just a shift of the coordinate origin along the z-axis, followed by a renaming of the z-axis to the x-axis. This leaves:

$$x^2 + \frac{y^2}{K+1} = \frac{R^2}{(K+1)^2}.$$

Dividing by the right-hand side, we get:

$$\frac{x^2(K+1)^2}{R^2} + \frac{y^2(K+1)}{R^2} = 1.$$

Now if we define constants a and b according to:

$$a^2 \equiv \frac{R^2}{(K+1)^2},$$

$$b^2 \equiv a^2(K+1),$$

we can re-express in a very familiar form:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1. \tag{16}$$

When $K = 0$, for a circle, $a = b = R$, and we can re-express Eq. 16 as the more familiar $x^2 + y^2 = R^2$. For ellipses, Eq. 16 is already in the most familiar Cartesian form with semi-major and semi-minor axes a and b. Note that in the case of ellipses, $K+1$ is always positive. In familiar terms, the eccentricity, e, of an ellipse satisfies: $a^2(1-e^2) = b^2$, from which we learn that $K = -e^2$. For hyperbolae, $K+1 < 0$, so that a^2 and b^2 differ in sign. If we associate $\alpha^2 = a^2$ and $\beta^2 = -b^2$, we could re-write Eq. 16 in the more familiar form for hyperbolae in Cartesian coordinates. The asymptotic slope is then β/α.