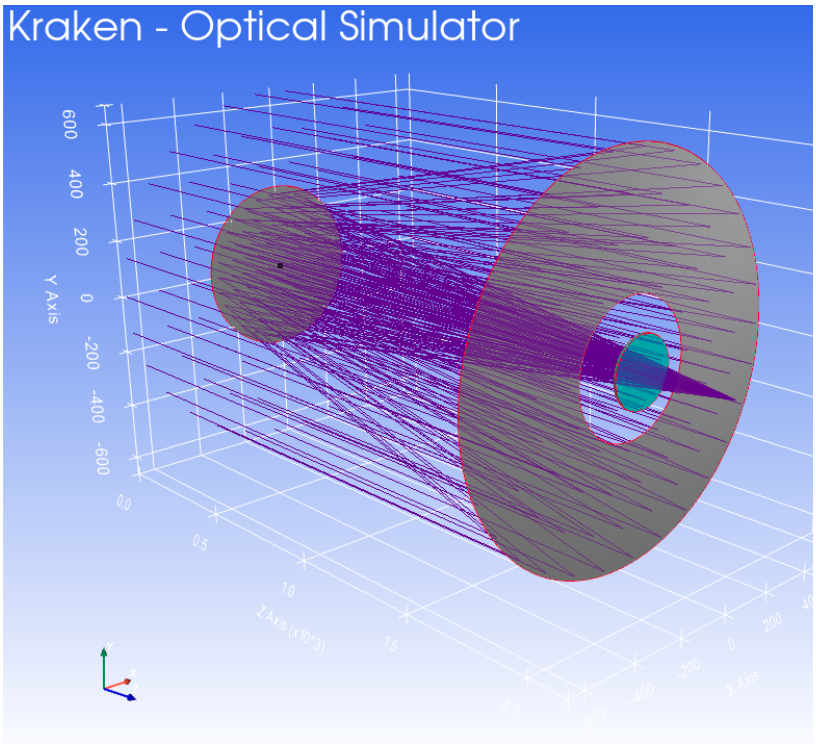
Kraken - User Manual

Joel Herrera V., Carlos Guerrero P., Morgan Rhaí Najera Roa, Anais Sotelo B., Ilse Plauchu F.
joel@astro.unam.mx



ABSTRACT:

This manual is intended for use as a technical reference by developers and users, **Kraken** is a ray tracing and optical simulation library that has been developed in the Python language, it was thought to be able to grow and adapt to different projects requirements in different optical tasks. The library was developed under the object-oriented paradigm focusing mainly its simplicity and the ability to be incorporated into new code where exact ray tracing is necessary. Being developed in Python there are many tools with which it can be combined increasing its capabilities. In the appendix at the end of this document a series of simple examples are presented.

Content

Introduction

The objective of this manual is to present the library commands and implementations, in addition, it also shows a series of examples in which all the functions are used. **Kraken** is a tool for the simulation of optical systems, it consists of a library developed in the Python 3.0, using the Numpy, PyVTK, PyVista and Matplotlib to provide a two-dimentional and three-dimensional visualization of the optical elements. This tool has been focused on the object-oriented paradigm, this gives us greater simplicity in the implementation of a system, as will be seen in this document.

1. PREREQUISITES

The library has been tested with the following packages and versions, you have to make sure you have them installed on your system.

- Python '3.7.4'
- numpy '1.18.5'
- pyvista '0.25.3'
- pyvtk '0.5.18'
- vtk '8.2'
- csv '1.0'

Place the directory Kraken in the same path where the code to be executed is located.

2. CLASSES AND ATTRIBUTES

The library has been simplified to the point of having only two classes of objects for the syatem definition, these are ***surf*** and ***system***, its instructions for use are described below.

The ***surf*** object contains all the relevant information of every optical interface, in this way, every optical interface is an object of the ***surf*** class, all interfaces, from the object plane to the image plane, have attributes of size, shape, material, orientation, etc.

Table 1. surf class atributes	
surf.Name = ""	Element name, useful only for 2D diagram or ray identification
surf.NamePos = (0,0)	Position of the note with the Name “Name” in the 2D diagram
surf.Note = "None"	Useful for user notes on that surface
surf.Rc = 9999999999.0	Paraxial radius of curvature in millimeters,
surf.Cylinder_Rxy_Ratio = 1	Ratio between axial and sagittal radius of curvature, useful for cylindrical and toroid lenses
surf.Axicon = 0	Axicon simulation, for non-zero values an axicon is generated with the entered angle.
surf.Thickness = 0.0	Gap between this surface and the next surface.

surf.Diameter = 1.0	Outside diameter of the surface.
surf.InDiameter = 0.0	Internal diameter of the surface, useful for items such as a primary mirror with a central aperture.
surf.k = 0.0	Conicity constant for classical conic surfaces, k = 0 for spherical, k = -1 for parabola, etc.
surf.DespX = 0.0	Surface displacement in the X, Y and Z axis (in millimeters)
surf.DespY = 0.0	
surf.DespZ = 0.0	
surf.TiltX = 0.0	Rotation of the surface in the X, Y and Z axis (in degrees).
surf.TiltY = 0.0	
surf.TiltZ = 0.0	
surf.Order = 0	It defines the order of the transformations. If the value is 0, the translations are performed first and then the rotations, if the value is 1 then the rotations are performed first and then the translations.
surf.AxisMove = 1	Defines what will happen to the optical axis after a coordinate transformation, if the value is zero the transformation is only carried out to the surface in question, if the value is 1 then the transformation also affects the optical axis therefore the other surfaces They will continue to transformation, if the value is different, for example 2, then the optical axis will be affected twice, the latter is useful for flat mirrors, see example.
surf.Diff_Ord = 0.0	Diffraction order, converts the element into a diffraction grating , the radius of curvature value must be omitted to make it a plane grating, it can be used in transmission or refraction.
surf.Grating_D = 0.0	Separation (microns) between the lines of the diffraction grating.
surf.Grating_Angle = 0.0	Angle of the grating lines in the plane of the surface, that is, around the optical axis. This is useful for simulating conic scattering.
surf.ZNK = np.zeros (36)	Numpy type array of 36 elements that correspond to the coefficients of the Zernike polynomials in the Noll nomenclature.
surf.ShiftX = 0	

surf.ShiftY = 0	Offset of the surface profile function on the x and y axis , this is useful for example for off-axis surfaces such as parabolas.
surf.Mask = 0	0 non masked, 1 apperture, 2 Obstruction.
surf.Mask_Shape = Object_3D	Example: 3D_Object = pv.Disc (center = [0.0,0.0,0.0], inner = 0, outer = 0.001, normal = (0,0,1), r_res = 3, c_res = 3).
surf. AspherData = np.zeros (Fix)	Arrangement of coefficients for aspheric surface
self.ExtraData = [f, coef]	User-created surface with a sagite function dependent on (x, y, V), where V can contain an array of coefficients usable by the function
Surf.Error_map = [X, Y, Z, SPACE]	It receive an error map generated with an Numpy array for the coordinates in X, and Y and the height in Z with a space value between X, Y.
surf.Draw ing = 1	1 for the element to be drawn in 3D plotting, 0 to skip drawing the element.
surf.Color = [0,0,0]	Define the color of the element in the format [1,1,1]
surf.Solid_3d_stl = "None"	Path of the 3D solid object in STL format.

The ***system*** object is a container for all interfaces, this object also provide the implementations for the ray tracing and the ray parameters calculations through the optical surfaces. The ***system*** atributes and implementations are shown below

<u>To understand how these atributes are used see: Example - -Doublet_Lens_CommandsSystem</u>

Table 2. system class atributes	
system.Trace (pS, dC, wV)	Trace is the main implementation of the *system* object , it tracesa ray through all the surfaces it finds in its path sequentially. The ray must be defined by a point of origin "pS", the directing cosines "dC" the wavelength "wV". See examples pS = [1.0, 0.0, 0.0] dC = [0.0,0.0,1.0] wV = 0.4 (Microns) for example
system.NsTrace (pS, dC, wV)	It traces rays in the same way as Trace, however, it is done in a non-sequential way, the parameters of the ray are defined in the same way.

Prx = system.Parax (w)	Returns the following paraxial calculations also accessible from system: Prx = SistemMatrix, S_Matrix, N_Matrix, a, b, c, d, EFFL, PPA, PPP, CC, N_Prec, DD
system.disable_inner	Enables and disables the center apertures, this is very useful, for example, when you want to calculate a ray trace without vignetting the aperture of a primary mirror.
system.enable_inner	Activate inner apertura if the surface have it.
system.SURFACE	Returns a list of the number of surfaces the ray passed through. Returns a list of the name of surfaces that the ray passed through, if a name was not added to the surfaces then the list will appear with empty fields. Naming surfaces is very useful, for example, to identify whether a ray strike has struck that surface.
system.NAME	
system.GLASS	Returns a list of the materials through which the ray has passed through.
system.XYZ	Returns the coordinates *[X, Y, Z]* of the ray from its origin to the image plane
system.S_XYZ	Returns the coordinates *[X, Y, Z]* of the ray from its origin and on all surfaces where this ray originates, that is, the coordinates of the image plane are exempt.
system.T_XYZ	Returns the *[X, Y, Z]* coordinates of the ray from the first surface where it intersects to the plane image.
system.OST_XYZ	Returns the coordinates *[X, Y, Z]* at which a ray intercepts the surface in interface space, that is, the coordinates with respect to a coordinate system at its vertex even if this vertex has a translation or rotation .
system.DISTANCE	Returns a list with the distances traveled by the ray between points of intersection.
system.OP	Returns a list with the optical paths traveled by the ray between intersection points, for this the dispersion of the glass and the distance between the intersection points are considered .
system.TOP	Returns the total optical path of the beam from the source to the last point of intersection.
system.TOP_S	Returns a cumulative list of the beam's optical path from the source to the last point of intersection.
system.ALPHA	Returns a list with the absorption coefficients for the materials in the intercepted surfaces considering the wavelength, these values are obtained from the materials catalog.

system.BULK_TRANS	Returns a list with the transmission through all the routes within the system, for this the optical paths and the absorption coefficients of the materials are considered.
system.S_LMN	Returns a list with the directing cosines [*L, M* , *N]* of the normals for the points of interest of a ray through all the interfaces through which it passes.
system.LMN	Returns a list with the director cosines [*L, M* , *N]* of an incident ray through all the interfaces through which it passes.
system.R_LMN	Returns a list with the directing cosines [*L, M* , *N]* of the resulting ray through all the interfaces it passes through.
system.N0	Refractive indices before and after each interface through which the ray passes. This index is calculated with the dispersion of the material and the wavelength of the ray in question.
system.N1	Returns a list with the directing cosines [*L, M* , *N]* of the resulting ray through all the interfaces it passes through.
system.WAV	Wavelength of ray in question, although the command returns a list all values are equal because the wavelength is constant for a ray, the size of the list indicates only the number of iterations with system interfaces.
system.G_LMN	Returns a list with the terms, *L, M* or *N* of the director cosines that define the lines of the diffraction grating on the plane.
system.ORDER	Returns a list with the diffraction orders associated with the ray in question.
system.GRATING	Distance between lines of the diffraction grating.
system.RP	Returns a list with the reflection and transmission Fresnel coefficients for polarization S and P
system.RS	
system.TP	Returns a list with the reflection and transmission Fresnel coefficients for polarization S and P
system.TS	
system.TTBE	Total energy transmitted or reflected per element.
system.TT	Total energy transmitted or reflected total.
system.targ_surf (int)	Limits ray tracing to defined surface (int)
system.flat_surf (int)	It is defined with the whole number of the surface that needs to be made flat, -1 to restore

system.disable_inner ()	If any surface of the system has a central hole this is disabled.
system.enable_inner ()	If any surface () has a disabled central hole, this command restores it.

3. WORKING WITH THE LIBRARY

Example: Unique Ray

Importing Kraken

```
import numpy as np
import Kraken as Kn
```

All surfaces are declared separately, each one with the possible parameters for a surface are shown in table 1.

```
P_Obj = Kn.surf()
P_Obj.Rc = 0.0
P_Obj.Thickness = 0.1
P_Obj.Glass = "AIR"
P_Obj.Diameter = 30.0

P_Obj2 = Kn.surf()
P_Obj2.Rc = 0.0
P_Obj2.Thickness = 10
P_Obj2.Glass = "AIR"
P_Obj2.Diameter = 30.0

L1a = Kn.surf()
L1a.Rc = 9.284706570002484E+001
L1a.Thickness = 6.0
L1a.Glass = "BK7"
L1a.Diameter = 30.0
L1a.Axicon = 0

L1b = Kn.surf()
L1b.Rc = -3.071608670000159E+001

L1b.Thickness = 3.0
L1b.Glass = "F2"
L1b.Diameter = 30

L1c = Kn.surf()
L1c.Rc = -7.819730726078505E+001
L1c.Thickness = 9.737604742910693E+001
L1c.Glass = "AIR"
L1c.Diameter = 30

P_Ima = Kn.surf()
P_Ima.Rc = 0.0
P_Ima.Thickness = 0.0
P_Ima.Glass = "AIR"
P_Ima.Diameter = 18.0
P_Ima.Name = "Plano imagen"
```

The ***Kraken_setup*** is used to load the glass catalog directories, This library must be modified to change the glass catalog if this is necessary. For now, a configuration object is created with the name ***configuration_1***.

```
configuration_1 = Kn.Kraken_setup()
```

An python list ***A*** is made with the objects created for all the surfaces, then an optical system is created with all of it and the ***configuration_1***.

```
A = [P_Obj, P_Obj2, L1a, L1b, L1c, P_Ima]
configuration_1 = Kn.Kraken_setup()
Doublet = Kn.system(A, configuration_1)
```

A ray is defined by three parameters, the origin coordinates called ***XYZ*** for this example, the direction defined by director cosines, named below as LMN, these two parameters are python lists with three values [x, y, z] and [l, m , n]. the third parameter is the wavelength, here expressed as ***w*** with a value of 0.4μm.

```
XYZ = [0, 14, 0]
Theta = 0.1
# Calculate the director cosines for a given angle Theta
LMN = [0.0, np.sin(np.deg2rad(Theta)), -np.cos(np.deg2rad(Theta))]
W = 0.4
```

The ray trace is made through the ***Doublet*** system in the following way

```
Doublet.Trace(XYZ, LMN, W)
```

It is possible to interrogate the ***Doublet*** system about what has happened with the ray, the communication with the Doublet system is made with the calls shown in table 2, for example, the GLASS keyword returns all the glasses through which the ray has passed by. It is used as follows.

```
print(Doublet.GLASS)
```

Returning the following list ['BK7', 'F2', 'AIR', 'AIR']. Another example is the request for the ray coordinates on all surfaces, we can also request the director cosines, for that we will use the ***XYZ*** and ***LMN*** atributes as follows:

```
print(Doublet.XYZ)
print(Doublet.LMN)
```

for which we obtain the following outputs respectively:

Doublet.XYZ=[array([0., 10., 0.]), [0.0, 10.0, 10.540090832982 81], [0.0, 9.85609739239213, 14.37575625216807], [0.0, 9.81741071793073, 18.381280697704405], [0.0, 0.05825657548273888, 116.37604742910693]]
Doublet.LMN=[array([0., 0., 1.]), array([0. , -0.03749061, 0.99929698]), array([0. , -0.00965788, 0.99995336]), array([0. , -0.09909831, 0.99507765])]

Both results are Numpy arrays, therefore, we can perform the operations we want with them directly, for example, below we see the shape of the array

```
print(np.shape(Doublet.XYZ))
print(np.shape(Doublet.LMN))
```

what returns us:

```
np.shape(Doublet.XYZ) = (5, 3)
np.shape(Doublet.LMN) = (4, 3)
```

Note that ***Doublet***.XYZ contains 5 arrays with three elements (x, y, z coordinates in three-dimensional space), while Doublet.LMN only contains 4 elements, this is because it only shows the director cosines between the

surfaces, that is, if a if stema has 5 elements from the object plane to the image plane, then there are only 4 ray segments between surfaces and therefore only 4 sets of directing cosines, exemplified below with -> between the surfaces.

A=[P_Obj -> L1a -> L1b -> L1c -> P_Ima]

After tracing a ray with the ***Trace*** option, we can request information from Doublet in the manner mentioned above, with this information, graphs or different analyzes can be performed, generally it is necessary to carry out a tracing with many rays, to preserve the results for one. or more rays we have the ray container in the ***Raykeeper*** class , with which for the example of this document we create the Rays object.

```
Rays = Kn.raykeeper(Doblete)
```

A keyword of the Raykeeper class is push (), with this we take the ray just traced inside Doublet as follows, an advantage of this is that the Doublet object does not have to save a memory with all the rays that we Trace, it We leave that task to the ray container, in this way we can have ray containers for different circumstances as we wish, for example, we can create a container for all the rays that we will trace from a field and then save in another container the rays that come of a different field, the user can find many utilities for this modality.

Each individual ray is saved by passing to the system in question as a parameter to the container in the following way, this is repeated every time a ray is traced.

```
Rays.push()
```

We also have two tools, Display2D and Display3D, these make the graph of the system, the input parameters are the system itself and a container of rays, it also receives a parameter for different considerations in the graph.

For two-dimensional graphics ***Display2D (System, Raykeeper, parameter)*** where the parameter can be the integers 0 or 1, indicating whether the graph will be in the xz or yz plane.

For graphics bidimen sional ***Display3D (System Raykeeper, parameter***) where the parameter can be the integers 0 to 2.

0	for a display of the complete elements
1	for deployment with surfaces cut ¼
2	for the deployment of cross sections.

For the example in this document we will have to replace ***System*** and ***Raykeeper*** with the objects created with them as follows

```
Display2D(Doblete,Rays,0)
```

Which generates the following graph displayed in Matplotlib. Remember that we only store a ray in the container, this is shown in blue, this depends on the wavelength used, in this case when defining the ray we use W = 0.4.

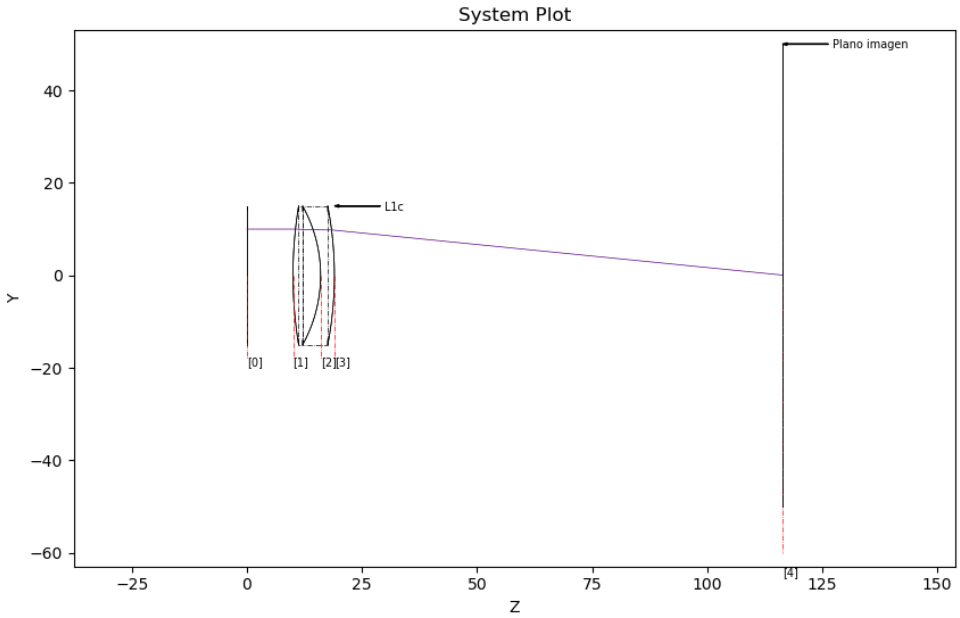


Figure 1. 2D visualization of a ray traced through a doublet

To perform the three-dimensional display we will use:

Display3D(Doblete,Rays,2)

This will open the following viewer window

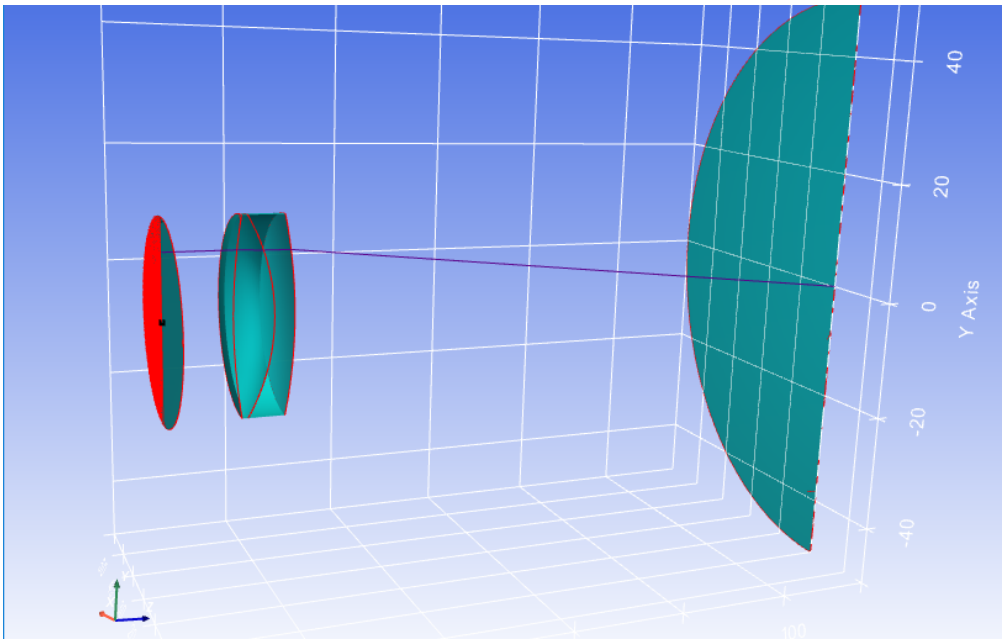


Figure 2. 3D visualization of a cross section of the doublet and a ray

We can store several rays, for example, then the creation of the ray is nested, its trace and its stored within a ***for*** loop.

```
for x in range(-10, 10):
 # Creamos un rayo paralelo al eje óptico y cambiamos la altura con "x" en el for
    XYZ = [0.0, x, 0.0]
    LMN =[0.0,0.0,1.0]
    W=0.4
    # Trazamos el rayo
    Doublet.Trace(XYZ, LMN,W)
    #Almacenamos el rayo
    Rays.push(Doblete)

# Despliega sistema con todos los Rays
Display3D(Doblete,Rays,2)
```

With which we get all the rays

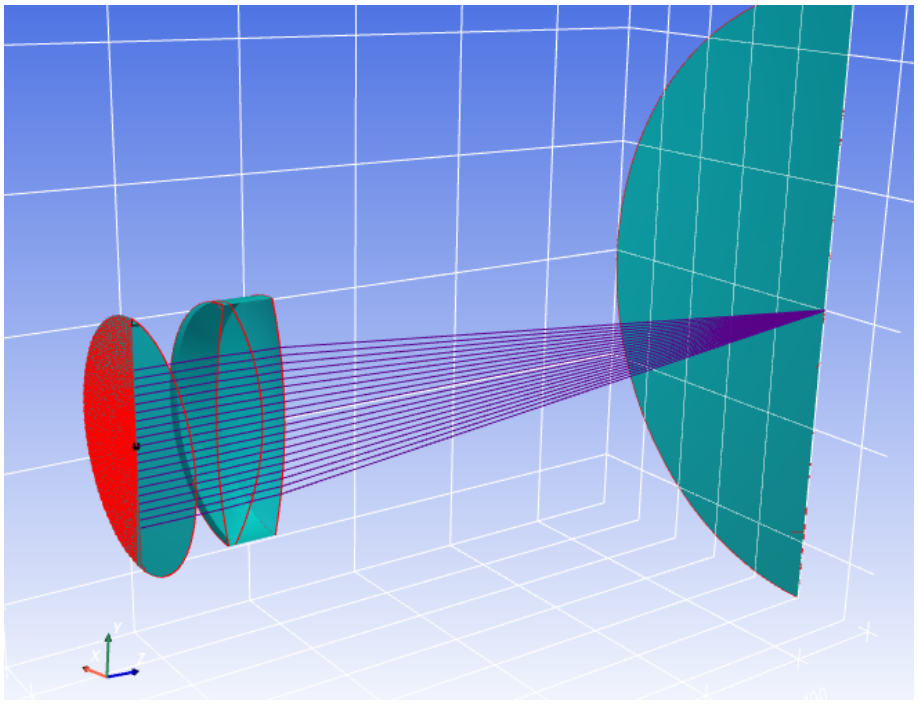


Figure 3. 3D display of the optical system with several rays

The ray container contains a list of preserves almost all the parameters that are accessible in the ***System*** class , however, now, by containing several rays, these parameters are arrays of arrays, so you have to take certain considerations into account.

By default, the ***Raykeeper*** ray container takes into account all the rays that are saved with the ***Raykeeper.push ()*** instruction , it is possible that some ray does not even enter the system due to its point of origin or its direction, anyway, the ray is taken and the existing information such as the point of origin, the director cosines and the wavelength, the rest of the data will be empty, this can also be useful for example if we want to know the characteristics of the rays that ***do not*** enter To the system. To know the number of rays stored ***Raykeeper*** has an internal variable ***nrays*** that can be called directly in this way.

```
print(Rays.nrays) Resultado: 100
```

Therefore, in this example we have 100 positions with rays about which we can request information from the container through the parameters shown in table 3 (Central column), these have the same information described in Table 2 of the ***System*** class . these calls to the ***Raykeeper*** container dep enden of the ray number defined by an integer value ***[#]*** . If the ray number is not indicated, the total data set is obtained in Numpy arrays. As already

mentioned, there are rays that did not touch any surface, if they touched a surface we call ***valid*** rays and the container can be asked for a list of their direction in the list of rays, this is shown below.

```
print(Rays.valid())
```

with which we obtain the following list.

[[25][26][27][28][29][30][31][32][33][34][35][36][37][38][39][40][41][42][43][44][45][46][47][48][49][50][51][52][53][54][55]]

We can now call the ray information using these values and the keywords from Table 3 (Central column), however, when executing the ***valid ()*** instruction on line 78 of the example, a new set of calls is created, These are the ones shown in Table 3 (right column) these are identical to those of the ***central*** column , but with the ***valid_*** prefix , it is important to clarify that if the ***valid ()*** instruction is not executed . The calls shown in the right column will be empty, so if this care is not taken, errors will be obtained.

The array that are obtained with this new set of calls contain only valid rays and can now be used directly as the empty rays have been eliminated in them, therefore, the numbering will be offset according to the number of empty spaces removed. Depended endo of the task you want to perform application could be one way or the other.

In the same way, a set of calls is created for the invalid rays, as expected, only the information of the initial ray can be obtained, these calls appear in Table 3 left column and are limited to coordinates of origin, direction and wavelength. The prefix for these calls is ***invalid_***.

Tabla 3. Atributos del contenedor ***Raykeeper***

All rays	***Valid rays***	***Invalid rays***
Raykeeper.RayWave[#]	Raykeeper.valid_RayWave[#]	
Raykeeper.SURFACE[#]	Raykeeper.valid_SURFACE[#]	
Raykeeper.NAME[#]	Raykeeper.valid_NAME[#]	
Raykeeper.GLASS[#]	Raykeeper.valid_GLASS[#]	
Raykeeper.S_XYZ) [#]	Raykeeper.valid_S_XYZ[#]	
Raykeeper.T_XYZ[#]	Raykeeper.valid_T_XYZ[#]	
Raykeeper.XYZ[#]	Raykeeper.valid_XYZ[#]	Raykeeper.invalid_XYZ[#]
Raykeeper.OST_XYZ[#]	Raykeeper.valid_OST_XYZ[#]	
Raykeeper.S_LMN[#]	Raykeeper.valid_S_LMN[#]	
Raykeeper.LMN[#]	Raykeeper.valid_LMN[#]	Raykeeper.invalid_LMN[#]
Raykeeper.R_LMN[#]	Raykeeper.valid_R_LMN[#]	
Raykeeper.N0[#]	Raykeeper.valid_N0[#]	
Raykeeper.N1[#]	Raykeeper.valid_N1[#]	
Raykeeper.WAV[#]	Raykeeper.valid_WAV[#]	Raykeeper.invalid_WAV[#]
Raykeeper.G_LMN[#]	Raykeeper.valid_G_LMN[#]	
Raykeeper.ORDER[#]	Raykeeper.valid_ORDER[#]	
Raykeeper.GRATING[#]	Raykeeper.valid_GRATING[#]	
Raykeeper.DISTANCE[#]	Raykeeper.valid_DISTANCE[#]	
Raykeeper.OP[#]	Raykeeper.valid_OP[#]	
Raykeeper.TOP_S[#]	Raykeeper.valid_TOP_S[#]	
Raykeeper.TOP[#]	Raykeeper.valid_TOP[#]	
Raykeeper.ALPHA[#]	Raykeeper.valid_ALPHA[#]	
Raykeeper.BULK_TRANS[#]	Raykeeper.valid_BULK_TRANS[#]	
Raykeeper.RP[#]	Raykeeper.valid_RP[#]	
Raykeeper.RS[#]	Raykeeper.valid_RS[#]	

Raykeeper.TP[#]	Raykeeper.valid_TP[#]	
Raykeeper.TS[#]	Raykeeper.valid_TS[#]	
Raykeeper.TTBE[#]	Raykeeper.valid_TTBE[#]	
Raykeeper.TT[#]	Raykeeper.valid_TT[#]	

To delete the rays within a ***Raykeeper*** you can redefine the container, repeating the creation of the container, or using the internal ***clean*** implementation .

```
Rays.clean()
```

Handling the 3D viewer

• Rotate the simulation in any direction, left mouse button
• Zoom, move the mouse up and down by pressing the right mouse button
• Dragging the simulation, pressing the left mouse button and pressing the SHIFT key on the keyboard.

1. PUPILCALC TOOL

The opening of the system or as it is commonly known "Aperture Stop" is the element that defines the amount of light that passes through the entire optical system, the image of the opening of the system produced by the optical elements prior to it is known as the Entrance pupil, the image of the entrance pupil produced by the entire system is known as the exit pupil. Unlike what could first be assumed, the entrance pupil is not necessarily defined by the first element. Sometimes the element that defines the amount of light is not an optical surface but a mechanical element.

The following figure shows an example of a “Stop aperture” in an element behind a lens, specifically on surface 4. Furthermore, the figure shows two different fields, both of which also cross this aperture in an identical way, here A problem arises because to generate rays that uniformly cover the aperture of the system, it would be necessary to trace the rays towards the object plane and with this define the properties of the ray so that it passes through said area of the pupil, note in the figure that the fields have different origins, therefore generating the rays one by one can be a complicated task to perform by calculating them manually.

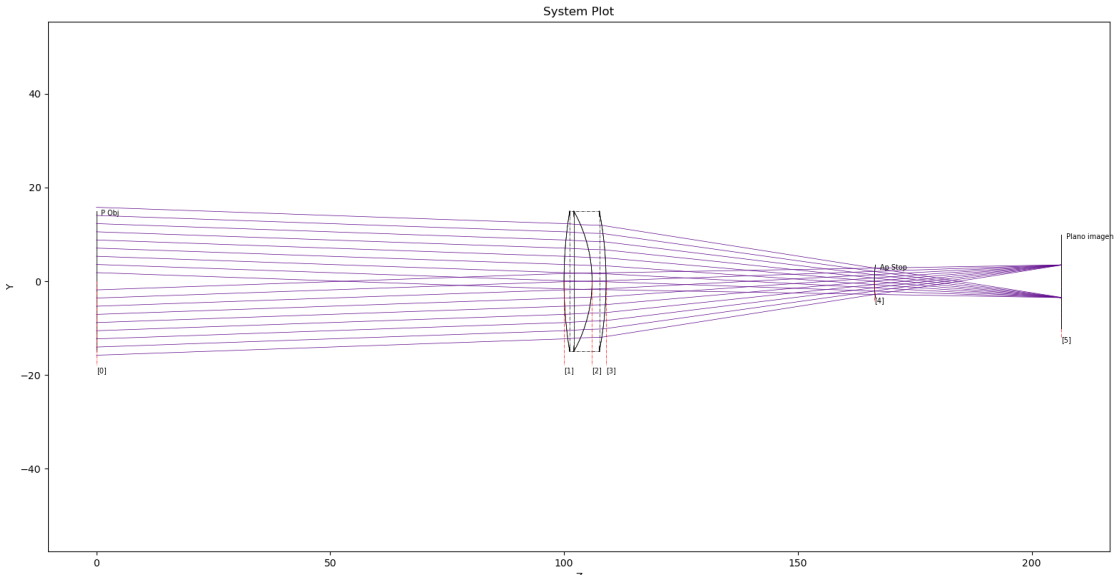


Figure 4. Visualization of two ray fanss that coincide with the position defined as the pupil.

To facilitate the generation of rays that cross the pupil, there is the ***PupilCalc*** tool , this class generates an iterative object as follows.

```
W = 0.4
Surf= 4
AperVal = 10
AperType = "EPD"
Pup = Kn.PupilCalc(Doblete, sup, W, AperType, AperVal)
```

Where *Doublet* is the system that we generate with the ***system*** class , *Surf* is the number of surface that represents the opening of the system, in the example of the image this is surface 4, and *W* is the wavelength for which the wavelength will be calculated. exit and entrance pupil.

The parameter ***AperVal*** corresponds to the diameter of the pupil, while the parameter AperType can have two different values "STOP" or "EPD" if we define as "STOP" we are indicating that the surface defined in *Surf* is the opening of the system, if the We define "EPD" as we are defining the entrance pupil diameter.

For this case where the object has been called Pup, the parameters of the pupils are obtained as shown in the following table.

Object attributes corresponding to pupil parameters	
Entrance pupil radius	Pup.RadPupInp
Entrance pupil position	Pup.PosPupInp
Exit pupil radius	Pup.RadPupOut
Exit pupil position	Pup.PosPupOut
Exit pupil position relative to the focal plane	Pup.PosPupOutFoc
Exit pupil orientation	Pup.DirPupSal

The position of the pupil is calculated even if the system has displaced and inclined elements, in that case, that displaced pupil becomes relevant in the calculation of the aberrations of the system.

Besides obtaining these parameters, you can also generate ray patterns in the pupil, calculating the director cosines and the origin coordinates defining the parameters.

Automatic ray generation based on the pupil	
Pup.Samp = #	Integer for pupil ray sampling, default value 5
Pup.Ptype = **"type of array"**	**"rteta"** Generates a ray at an angle from the unit pupil to a radial position, the radius and the angle must be defined as: **Pup.rad = n** **Pup.tet a = m** Where n is a floating number from 0-1 and theta is the angle from 0 to 360
	"chief" Main ray passing through the center of the pupil

	"hexapolar", Hexapolar ray array
	"square" , Rectangular ray array
	"fanx" , Linear **array** only on the x-axis
	"fany" , Linear array only on the y axis
	"fan" , Linear array on the x and y axes
	"rand" , Random arrangement
Pup.FieldType = **"height" or "angle"**	Define the type of field, in terms of the height of the object at the distance from the object plane with the **"height"** parameter , for parallel rays reaching the pupil from infinity the **"angle"** parameter is used
Pup.FieldY = # Pup.F ieldX = #	Field value in millimeters or degrees on the X and Y axis depending on the type of field that has been chosen in FieldType

The attributes shown in the previous table define the type of rays that we want, to obtain the array of rays, all we have to do is transfer them from the unit pupil to the real pupil, obtaining the directing cosines and the origin coordinates in the form of Numpy arrays. as follows:

```
x, y, z, L, M, N = Pup.Pattern2Field()
```

Where x, y, z are the origin coordinates and L, M, N are the directing cosines of the ray, in this way we can iterate between them and make the ray tracing through the system as shown in the following example, where the system is called **Doublet** and the RayKeeper **rays**.

```
for i in range(0, len(x)):
    pSource_0 = [x[i], y[i], z[i]]
    dCos = [L[i], M[i], N[i]]
    Doublet.Trace(pSource_0, dCos, W)
    Rays.push()
```

We can also graph the points of the rays generated in the object plane, for example, in the following image we define a "hexapolar" pattern.

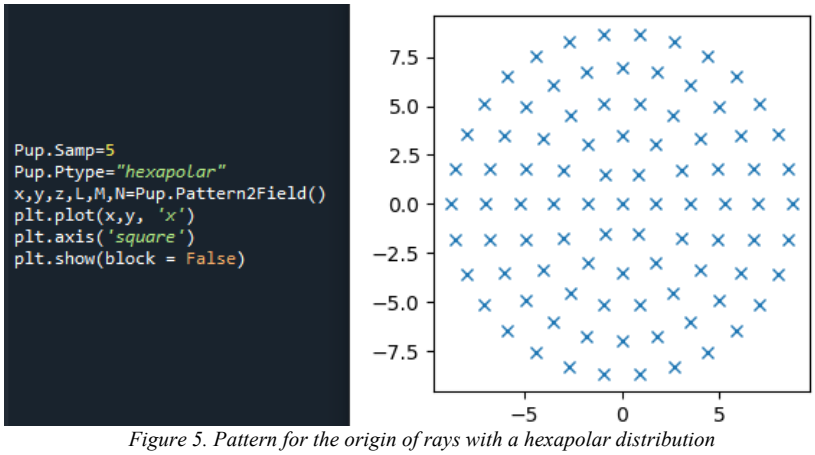


Figure 5. Pattern for the origin of rays with a hexapolar distribution

We can then generate the spot diagram:

```
Pup.Samp=10
Pup.Ptype="hexapolar"
Pup.FieldType="angle"
Pup.FieldY=2.0

x,y,z,L,M,N=Pup.Pattern2Field()


for i in range(0,len(x)):
    pSource_0 = [x[i], y[i], z[i]]
    dCos=[L[i], M[i], N[i]]

    Doblete.Trace(pSource_0, dCos,W)
    Rayos.push()

X,Y,Z,L,M,N=Rayos.pick(-1)
plt.figure(300)
plt.plot(X,Y, 'x')
plt.axis('square')
plt.show(block = False)
```

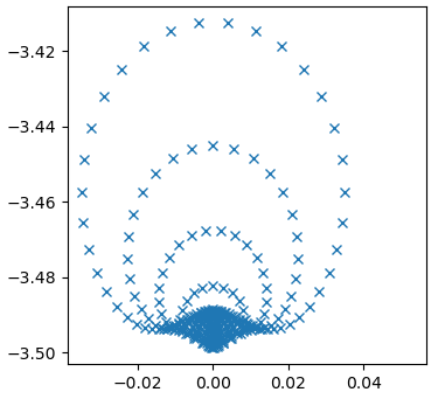


Figure 6. Spot diagram generated with a hexapolar pattern and with a field angle of 2 °

It is important to note that if it is required to analyze different fields, a matrix must be defined for each field and the ray tracing must also be saved in different containers if necessary to keep the results separate.

2. ATMOSPHERIC REFRACTION IN PUPILCALC

If ***FieldType*** is defined as *"angle"*, then since the rays come from infinity they are very suitable for use in telescopes. The open source library ***AstroAtmosphere*** has also been included to calculate de atmosphere effect in the incoming rays. For more information about this library reading this article is recommended: ***"Quantification of the expected residual dispersion of the MICADO Near-IR imaging instrument*** , van den Born & Jellema, 2020, MNRAS, DOI: 10.1093 / mnras / staa1870 ".

This library performs the calculation of the deviation of a ray depending on the physical parameters of the observatory, the reference wavelength and the zenith distance. PupilCalc uses this library internally to calculate the modification that the rays require. This function will be set as PupilCalc parameter as shown below:

```
Pup.AtmosRef = 1      # 0 to disable, 1 to enable
Pup.T    = 283.15     # Temperature (k)
Pup.P    = 101300     # Atmospheric presure(Pa)
Pup.H    = 0.5        # Humidity (0 to 1)
Pup.xc   = 400        # CO2 (ppm)
Pup.lat  = 31         # Latitude (degrees)
Pup.h    = 2800       # Observatory height (meters)
Pup.l1   = 0.60169    # Reference wavelenght micron
Pup.l2   = 0.50169    # micron
Pup.z0   = 55.0       # Zenith distance (degrees)
```

Next, an example is shown for a telescope for three groups of rays in different wavelength

```
W1 = 0.50169
Pup.12  = W1
xa,ya,za,La,Ma,Na=Pup.Pattern2Field()

W2 = 0.60169
Pup.12  = W2
xb,yb,zb,Lb,Mb,Nb=Pup.Pattern2Field()

W3 = 0.70169
Pup.12  = W3
xc,yc,zc,Lc,Mc,Nc=Pup.Pattern2Field()
```

Ray tracing is then performed for the ray groups and they are stored in different containers..

```
for i in range(0,len(xa)):
    pSource_0 = [xa[i], ya[i], za[i]]
    dCos=[La[i], Ma[i], Na[i]]
    Telescopio.Trace(pSource_0, dCos, W1)
    Rays1.push()

for i in range(0,len(xb)):
    pSource_0 = [xb[i], yb[i], zb[i]]
    dCos=[Lb[i], Mb[i], Nb[i]]
    Telescopio.Trace(pSource_0, dCos, W2)
    Rays2.push()

for i in range(0,len(xc)):
    pSource_0 = [xc[i], yc[i], zc[i]]
    dCos=[Lc[i], Mc[i], Nc[i]]
    Telescopio.Trace(pSource_0, dCos, W3)
    Rays3.push()
```

Ploting the rays intersections on the image plane we have Figure 7, a separation of 72 µm can be seen between the extreme wavelengths.

```
Extracting rays from the Three raykeepers()

X,Y,Z,L,M,N=Rays1.pick(-1)
plt.plot(X*1000.0,Y*1000.0, 'x', c="b")

X,Y,Z,L,M,N=Rays2.pick(-1)
plt.plot(X*1000.0,Y*1000.0, 'x', c="r")
```

```
X,Y,Z,L,M,N=Rays3.pick(-1)
plt.plot(X*1000.0,Y*1000.0, 'x', c="g")
```

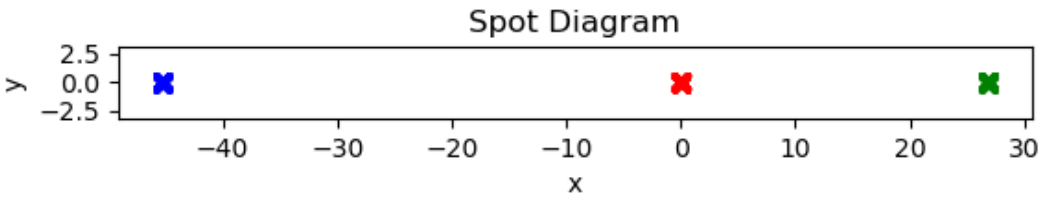


Figure 7. Images formed by a telescope which are spectrally displaced by the action of atmospheric refraction.

3. PARAX TOOL

It performs a series of paraxial calculations, basically provides all the necessary information for paraxial ray tracing in matrix form. The calculations are performed with the system object and for the requested wavelength as follows:

```
Prx = Doublet.Parax(0.4)
SistemMatrix, S_Matrix, N_Matrix, a, b, c, d, EFFL, PPA, PPP, C, N, D = Prx
```

Total system matrix:

```
SistemMatrix = [ [ 8.69329466e-01 -9.80840791e-03]
                 [ 1.00167690e+02  2.01470656e-02] ]
```

Surface by surface matrix:

```
S_Matrix = [matrix([[1., 0.],[0., 1.]]),
            matrix([[ 1.,  0.],[10.,  1.]]),
            matrix([[ 0.65323249, -0.00361793],[ 0.        ,  1.        ]]),
            matrix([[1., 0.],[6., 1.]]),
            matrix([[0.92657697, 0.00239038],[0.        , 1.        ]]),
            matrix([[1., 0.],[3., 1.]]),
            matrix([[ 1.65215474, -0.00833986],[ 0.        ,  1.        ]]),
            matrix([[ 1.        ,  0.        ],[97.37604743,  1.        ]]),
            matrix([[1., 0.],[0., 1.]]),
            matrix([[1., 0.],[0., 1.]])]
```

Surface by surface matrix names just for help:

```
N_Matrix = [ 'R sup: 0',
             'T sup: 0 to 1',
             'R sup: 1',
             'T sup: 1 to 2',
             'R sup: 2',
             'T sup: 2 to 3',
             'R sup: 3',
             'T sup: 3 to 4',
             'R sup: 4',
             'T sup: 4 to 5']
```

Values [abcd] from total system matrix:

```
a = 0.8693294662072478
b = -0.009808407908592333
```

```
c = 100.16768993870667
d = 0.02014706564149671
```

Effective focal length (EFFL), anterior principal plane (APP) and posterior principal plane (PPP):

```
EFFL = 101.9533454684305
PPA = -99.89928472490783
PPP = 13.322298074316684
```

Curvatures (C), refraction index (N) and surface separations(D):

```
C = [ 1.00000000e-12  1.04332892e-02 -3.25562791e-02 -1.27881671e-02 1.00000000e-12]
N = [1.0, 1.5308485382492993, 1.6521547400480732, 1.0, 1.0]
D = [10.          6.          3.         97.37604743  0.        ]
```

4. APPENDIX - EXAMPLES

The best way to understand the use of this library is by following the examples included in it and listed below, and its code is shown below:

Examp-Axicon.py
Examp-Axicon_And_Cylinder.py
Examp-Diffraction_Grating_Reflection.py
Examp-Diffraction_Grating_Transmission.py
Examp-Doublet_Lens-ParaxMatrix.py
Examp-Doublet_Lens.py
Examp-Doublet_Lens_3Dcolor.py
Examp-Doublet_Lens_CommandsSystem.py
Examp-Doublet_Lens_Cylinder.py
Examp-Doublet_Lens_NonSec.py
Examp-Doublet_Lens_Pupil.py
Examp-Doublet_Lens_Pupil_Seidel.py
Examp-Doublet_Lens_Tilt-Nulls.py
Examp-Doublet_Lens_Tilt.py
Examp-Doublet_Lens_Tilt_non_sec.py
Examp-Doublet_Lens_Zernike.py
Examp-ExtraShape_Micro_Lens_Array.py
Examp-ExtraShape_Radial_Sine.py
Examp-ExtraShape_XY_Cosines.py
Examp-Flat_Mirror_45Deg.py
Examp-MultiCore.py
Examp-ParaboleMirror_Shift.py
Examp-Perfect_lens.py
Examp-Ray.py
Examp-Solid_Object_STL.py
Examp-Solid_Object_STL_ARRAY.py
Examp-Sourse_Distribution_Function.py
Examp-Tel_2M.py
Examp-Tel_2M_Error_Map.py
Examp-Tel_2M_Pupila.py
Examp-Tel_2M_Spyder_Spot_Diagram.py

A. EXAMPLE - RAY

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
Examp Ray"""
import numpy as np
import Kraken as Kn


P_Obj = Kn.surf()
P_Obj.Rc = 0.0
P_Obj.Thickness = 0.1
P_Obj.Glass = "AIR"
P_Obj.Diameter = 30.0

P_Obj2 = Kn.surf()
P_Obj2.Rc = 0.0
P_Obj2.Thickness = 10
P_Obj2.Glass = "AIR"
P_Obj2.Diameter = 30.0

L1a = Kn.surf()
L1a.Rc = 9.284706570002484E+001
L1a.Thickness = 6.0
L1a.Glass = "BK7"
L1a.Diameter = 30.0
L1a.Axicon = 0

L1b = Kn.surf()
L1b.Rc = -3.071608670000159E+001
L1b.Thickness = 3.0
L1b.Glass = "F2"
L1b.Diameter = 30

L1c = Kn.surf()
L1c.Rc = -7.819730726078505E+001
L1c.Thickness = 9.737604742910693E+001
L1c.Glass = "AIR"
L1c.Diameter = 30

P_Ima = Kn.surf()
P_Ima.Rc = 0.0
P_Ima.Thickness = 0.0
P_Ima.Glass = "AIR"
P_Ima.Diameter = 18.0
P_Ima.Name = "Plano imagen"

A = [P_Obj, P_Obj2, L1a, L1b, L1c, P_Ima]
configuration_1 = Kn.Kraken_setup()

Doublet = Kn.system(A, configuration_1)
Rays = Kn.raykeeper(Doblete)

pSource_0 = [0, 14, 0]
tet = 0.1
dCos = [0.0, np.sin(np.deg2rad(tet)), -np.cos(np.deg2rad(tet))]
W = 0.4
Doublet.Trace(pSource_0, dCos, W)
Rays.push()

Kn.display3d(Doblete, Rays, 2)
```

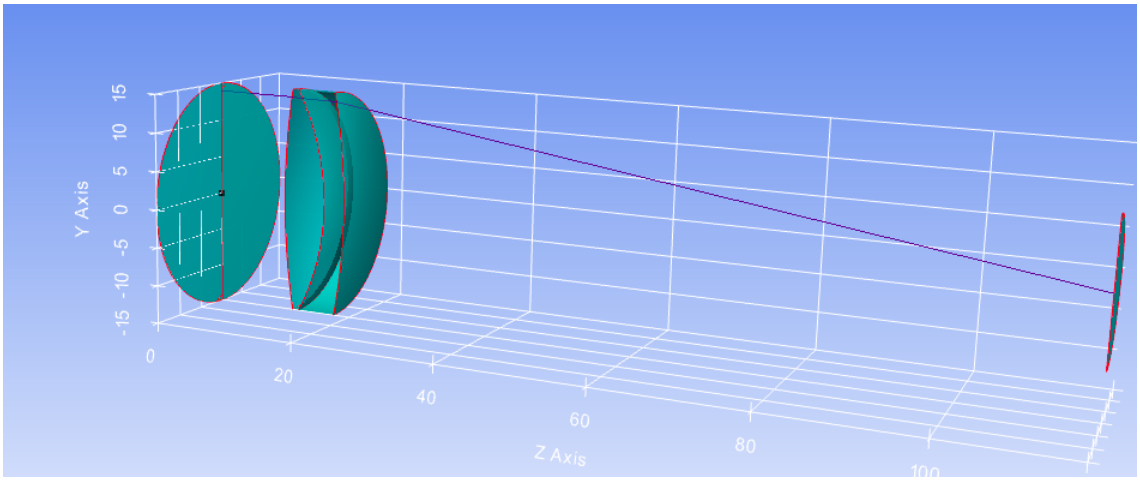
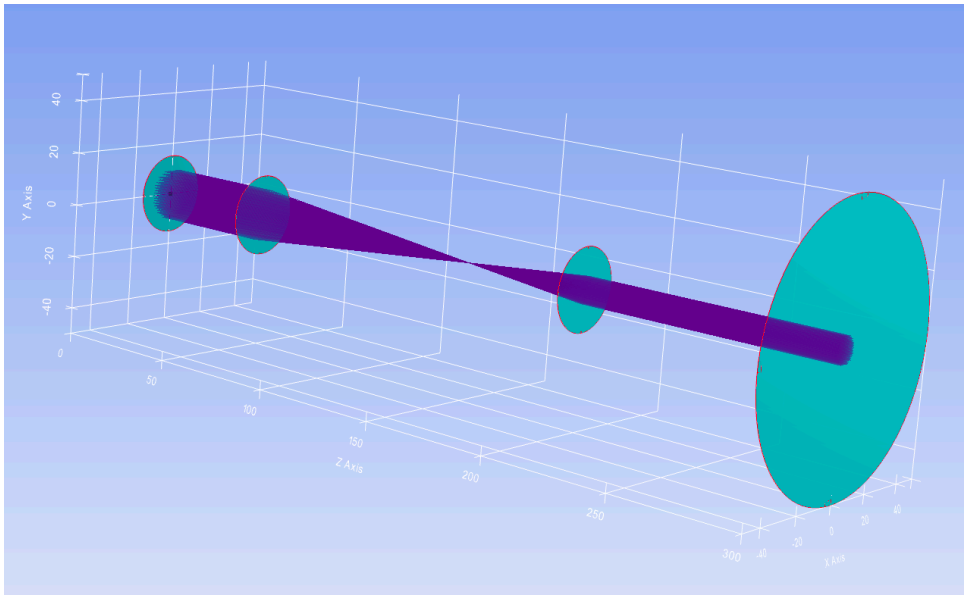


Figure 8. Example of a single ray tracing through an optical system.

B. EXAMPLE - PERFECT LENS

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
Examp Perfect Lens"""
import time
import matplotlib.pyplot as plt
import numpy as np
import Kraken as Kn

start_time = time.time()

P_Obj = Kn.surf()
P_Obj.Rc = 0.0
P_Obj.Thickness = 50
P_Obj.Glass = "AIR"
P_Obj.Diameter = 30.0

L1a = Kn.surf()
L1a.Thin_Lens = 100.
L1a.Thickness = (100 + 50)
L1a.Rc = 0.0
L1a.Glass = "AIR"
L1a.Diameter = 30.0

L1b = Kn.surf()
L1b.Thin_Lens = 50.
L1b.Thickness = 100.
L1b.Rc = 0.0
L1b.Glass = "AIR"
L1b.Diameter = 30.0

P_Ima = Kn.surf()
P_Ima.Rc = 0.0
P_Ima.Thickness = 0.0
P_Ima.Glass = "AIR"
P_Ima.Diameter = 100.0
P_Ima.Name = "Plano imagen"

A = [P_Obj, L1a, L1b, P_Ima]
config_1 = Kn.Kraken_setup()

Doublet = Kn.system(A, config_1)
Rays1 = Kn.raykeeper(Doblete)
```

```
Rays2 = Kn.raykeeper(Doblete)
Rays3 = Kn.raykeeper(Doblete)
RaysT = Kn.raykeeper(Doblete)

tam = 10
rad = 10.0
tsis = len(A) - 1
for j in range(-tam, tam + 1):
    for i in range(-tam, tam + 1):
        x_0 = (i / tam) * rad
        y_0 = (j / tam) * rad
        r = np.sqrt((x_0 * x_0) + (y_0 * y_0))
        if r < rad:
            tet = 0.0
            pSource_0 = [x_0, y_0, 0.0]
            dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]
            W = 0.4
            Doublet.Trace(pSource_0, dCos, W)
            Rays1.push()
            RaysT.push()

Kn.display3d(Doblete, RaysT, 0)
X, Y, Z, L, M, N = Rays1.pick(-1)

plt.plot(X, Y, 'x')
plt.xlabel('numbers')
plt.ylabel('values')
plt.title('Stop Diagram')
plt.axis('square')
plt.show()
print("--- %s seconds ---" % (time.time() - start_time))
```



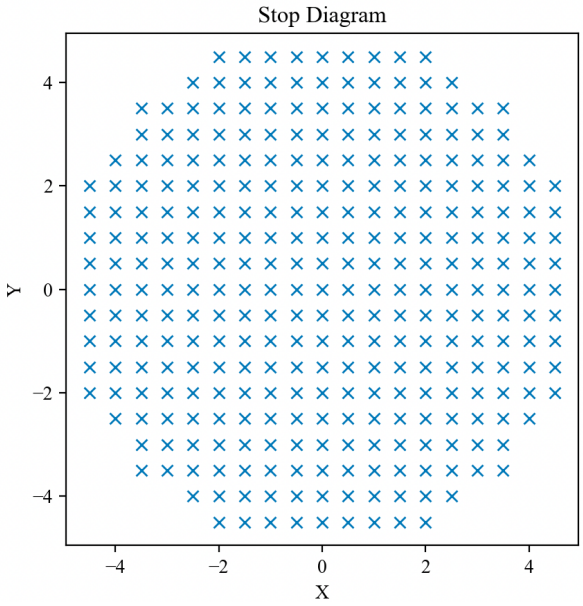


Figure 9. Example of perfect lenses that do not show aberrations.

C. EXAMPLE - DOUBLET LENS 3D COLOR

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
Examp Doublet Lens 3D color"""
import numpy as np
import Kraken as Kn


P_Obj = Kn.surf()
P_Obj.Rc = 0.0
P_Obj.Thickness = 10
P_Obj.Glass = "AIR"
P_Obj.Diameter = 30.0

L1a = Kn.surf()
L1a.Rc = 9.284706570002484E+001
L1a.Thickness = 6.0
L1a.Glass = "BK7"
L1a.Diameter = 30.0
L1a.Axicon = 0
L1a.Color = [.8, .7, .4]

L1b = Kn.surf()
L1b.Rc = -3.071608670000159E+001
L1b.Thickness = 3.0
L1b.Glass = "F2"
L1b.Diameter = 30
L1b.Color = [.7, .4, .4]

L1c = Kn.surf()
L1c.Rc = -7.819730726078505E+001
L1c.Thickness = 9.737604742910693E+001
L1c.Glass = "AIR"
```

```
L1c.Diameter = 30

P_Ima = Kn.surf()
P_Ima.Rc = 0.0
P_Ima.Thickness = 0.0
P_Ima.Glass = "AIR"
P_Ima.Diameter = 100.0
P_Ima.Name = "Plano imagen"

A = [P_Obj, L1a, L1b, L1c, P_Ima]
configuration_1 = Kn.Kraken_setup()

Doublet = Kn.system(A, configuration_1)
Rays = Kn.raykeeper(Doblete)

tam = 5
rad = 10.0
tsis = len(A) - 1
for i in range(-tam, tam + 1):
    for j in range(-tam, tam + 1):
        x_0 = (i / tam) * rad
        y_0 = (j / tam) * rad
        r = np.sqrt((x_0 * x_0) + (y_0 * y_0))
        if r < rad:
            tet = 0.0
            pSource_0 = [x_0, y_0, 0.0]
            dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]
            W = 0.4
            Doublet.Trace(pSource_0, dCos, W)
            Rays.push()
            W = 0.5
            Doublet.Trace(pSource_0, dCos, W)
            Rays.push()
            W = 0.6
            Doublet.Trace(pSource_0, dCos, W)
            Rays.push()

Kn.display3d(Doblete, Rays, 1)
```

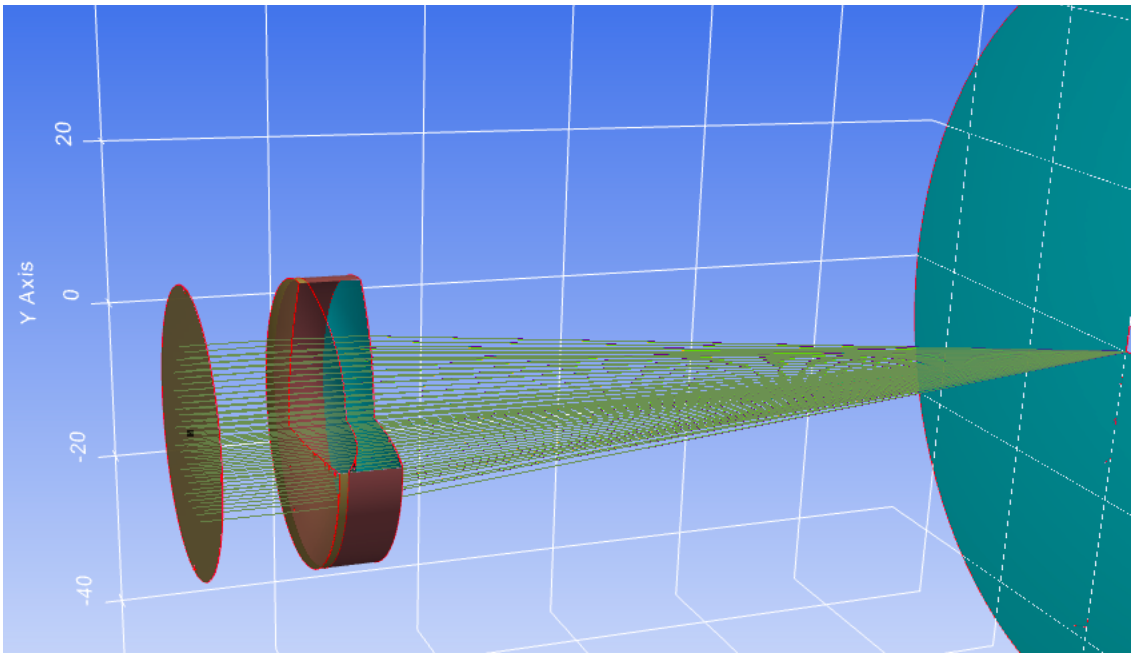


Figure 10. View of a doublet where a change in surface color has been defined

D. EXAMPLE - DOUBLET LENS TILT

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
Examp Doublet Lens Tilt"""
import numpy as np
import Kraken as Kn


P_Obj = Kn.surf()
P_Obj.Rc = 0.0
P_Obj.Thickness = 10
P_Obj.Glass = "AIR"
P_Obj.Diameter = 30.0

L1a = Kn.surf()
L1a.Rc = 9.284706570002484E+001
L1a.Thickness = 6.0
L1a.Glass = "BK7"
L1a.Diameter = 30.0
L1a.AxisMove = 1
L1a.TiltX = 1
L1a.DespY = 10.

L1b = Kn.surf()
L1b.Rc = (-3.071608670000159E+001)
L1b.Thickness = 3.0
L1b.Glass = "F2"
L1b.Diameter = 30

L1c = Kn.surf()
L1c.Rc = (-7.819730726078505E+001)
L1c.Thickness = 9.737604742910693E+001
L1c.Glass = "AIR"
L1c.Diameter = 30

P_Ima = Kn.surf()
P_Ima.Rc = 0.0
P_Ima.Thickness = 0.0
P_Ima.Glass = "AIR"
P_Ima.Diameter = 100.0
P_Ima.Name = "Plano imagen"

A = [P_Obj, L1a, L1b, L1c, P_Obj, L1a, L1b, L1c, P_Ima]
configuration_1 = Kn.Kraken_setup()

Doublet = Kn.system(A, configuration_1)
Rays = Kn.raykeeper(Doblete)

tam = 5
rad = 10.0
tsis = len(A) - 1
for i in range(-tam, tam + 1):
    for j in range(-tam, tam + 1):
        x_0 = (i / tam) * rad
        y_0 = (j / tam) * rad
        r = np.sqrt((x_0 * x_0) + (y_0 * y_0))
        if r < rad:
            tet = 0.0
            pSource_0 = [x_0, y_0, 0.0]
            dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]
            W = 0.4
            Doublet.Trace(pSource_0, dCos, W)
            Rays.push()
            W = 0.5
            Doublet.Trace(pSource_0, dCos, W)
            Rays.push()
            W = 0.6
            Doublet.Trace(pSource_0, dCos, W)
            Rays.push()

Kn.display3d(Doblete, Rays, 2)
Kn.display2d(Doblete, Rays, 0)# !/usr/bin/env python3
```

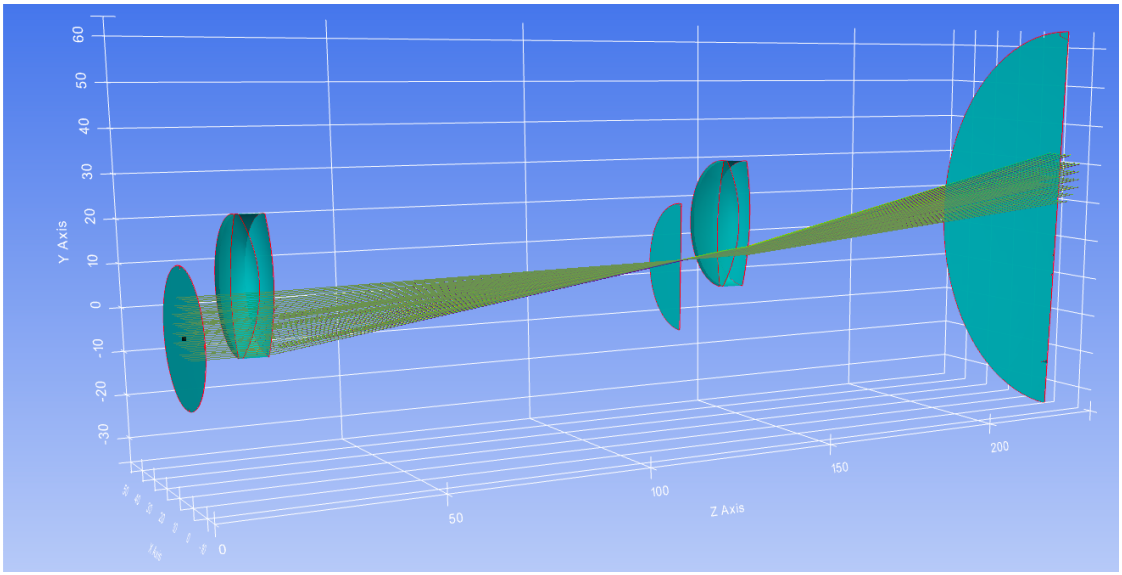


Figure 11. 3D visualization of an off-axis system generated with repetition of surfaces, the modification of a surface affects all parts of the design where it is used.

E. EXAMPLE - DOUBLET LENS (CÁLCULOS PARAXIALES)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
Examp Doublet Lens Para Matrix"""
import time
import Kraken as Kn

start_time = time.time()


P_Obj = Kn.surf()
P_Obj.Rc = 0.0
P_Obj.Thickness = 10
P_Obj.Glass = "AIR"
P_Obj.Diameter = 30.0

L1a = Kn.surf()
L1a.Rc = 9.284706570002484E+001
L1a.Thickness = 6.0
L1a.Glass = "BK7"
L1a.Diameter = 30.0
L1a.Axicon = 0

L1b = Kn.surf()
L1b.Rc = -3.071608670000159E+001
L1b.Thickness = 3.0
L1b.Glass = "F2"
L1b.Diameter = 30

L1c = Kn.surf()
L1c.Rc = -7.819730726078505E+001
L1c.Thickness = 9.737604742910693E+001
L1c.Glass = "AIR"
L1c.Diameter = 30

P_Ima = Kn.surf()
P_Ima.Rc = 0.0
P_Ima.Thickness = 0.0
P_Ima.Glass = "AIR"
P_Ima.Diameter = 3.0
```

```
P_Ima.Name = "Plano imagen"

A = [P_Obj, L1a, L1b, L1c, P_Ima]
config_1 = Kn.Kraken_setup()

Doublet = Kn.system(A, config_1)
print("Calculando para cierta wave --------")
Prx = Doublet.Parax(0.4)
SistemMatrix, S_Matrix, N_Matrix, a, b, c, d, EFFL, PPA, PPP, CC, N_Prec, DD = Prx
print(EFFL)

L1a.Rc = L1a.Rc + 1
Doublet.ResetData()
print("Calculando para cierta wave --------")
Prx = Doublet.Parax(0.4)
SistemMatrix, S_Matrix, N_Matrix, a, b, c, d, EFFL, PPA, PPP, CC, N_Prec, DD = Prx
print(EFFL)

L1a.Rc = L1a.Rc + 1
Doublet.ResetData()
print("Calculando para cierta wave --------")
Prx = Doublet.Parax(0.4)
SistemMatrix, S_Matrix, N_Matrix, a, b, c, d, EFFL, PPA, PPP, CC, N_Prec, DD = Prx
print(EFFL)

L1a.Rc = L1a.Rc + 1
Doublet.ResetData()
print("Calculando para cierta wave --------")
Prx = Doublet.Parax(0.4)
SistemMatrix, S_Matrix, N_Matrix, a, b, c, d, EFFL, PPA, PPP, CC, N_Prec, DD = Prx
print(EFFL)
```

```
100.18502274454893
100.78009773058245
101.3695118403712
101.9533454684305
```

Figure 12. Resulting focal lengths for a repeated 1mm increment between calculations.

F. EXAMPLE - DOUBLET LENS TILT NULLS

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
Examp Doublet Lens Tilt Nulls"""
import numpy as np
import Kraken as Kn


P_Obj = Kn.surf()
P_Obj.Rc = 0.0
P_Obj.Thickness = 1
P_Obj.Glass = "AIR"
P_Obj.Diameter = 30.0

L1a = Kn.surf()
L1a.Rc = 5.513435044607768E+001
L1a.Thickness = 6.0
L1a.Glass = "BK7"
L1a.Diameter = 30.0
L1a.TiltX = 4

Null1_L1a = Kn.surf()
Null1_L1a.Thickness = -L1a.Thickness
Null1_L1a.Glass = "NULL"
Null1_L1a.Diameter = L1a.Diameter
Null1_L1a.TiltX = -L1a.TiltX
Null1_L1a.Order = 1

Null2_L1a = Kn.surf()
```

```
Null2_L1a.Thickness = L1a.Thickness
Null2_L1a.Glass = "NULL"
Null2_L1a.Diameter = L1a.Diameter

L1b = Kn.surf()
L1b.Rc = -4.408716526030626E+001
L1b.Thickness = 3.0
L1b.Glass = "F2"
L1b.Diameter = 30

L1c = Kn.surf()
L1c.Rc = -2.246906271406796E+002
L1c.Thickness = 9.737871661422000E+001
L1c.Glass = "AIR"
L1c.Diameter = 30

P_Ima = Kn.surf()
P_Ima.Rc = 0.0
P_Ima.Thickness = 0.0
P_Ima.Glass = "AIR"
P_Ima.Diameter = 100.0
P_Ima.Name = "Plano imagen"

A = [P_Obj, L1a, Null1_L1a, Null2_L1a, L1b, L1c, P_Ima]
configuration_1 = Kn.Kraken_setup()

Doublet = Kn.system(A, configuration_1)
Rays = Kn.raykeeper(Doblete)

tam = 5
rad = 10.0
tsis = len(A) - 1
for i in range(-tam, tam + 1):
    for j in range(-tam, tam + 1):
        x_0 = (i / tam) * rad
        y_0 = (j / tam) * rad
        r = np.sqrt((x_0 * x_0) + (y_0 * y_0))
        if r < rad:
            tet = 0.0
            pSource_0 = [x_0, y_0, 0.0]
            dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]
            W = 0.4
            Doublet.Trace(pSource_0, dCos, W)
            Rays.push()
            W = 0.5
            Doublet.Trace(pSource_0, dCos, W)
            Rays.push()
            W = 0.6
            Doublet.Trace(pSource_0, dCos, W)
            Rays.push()

Kn.display2d(Doblete, Rays, 0)
```

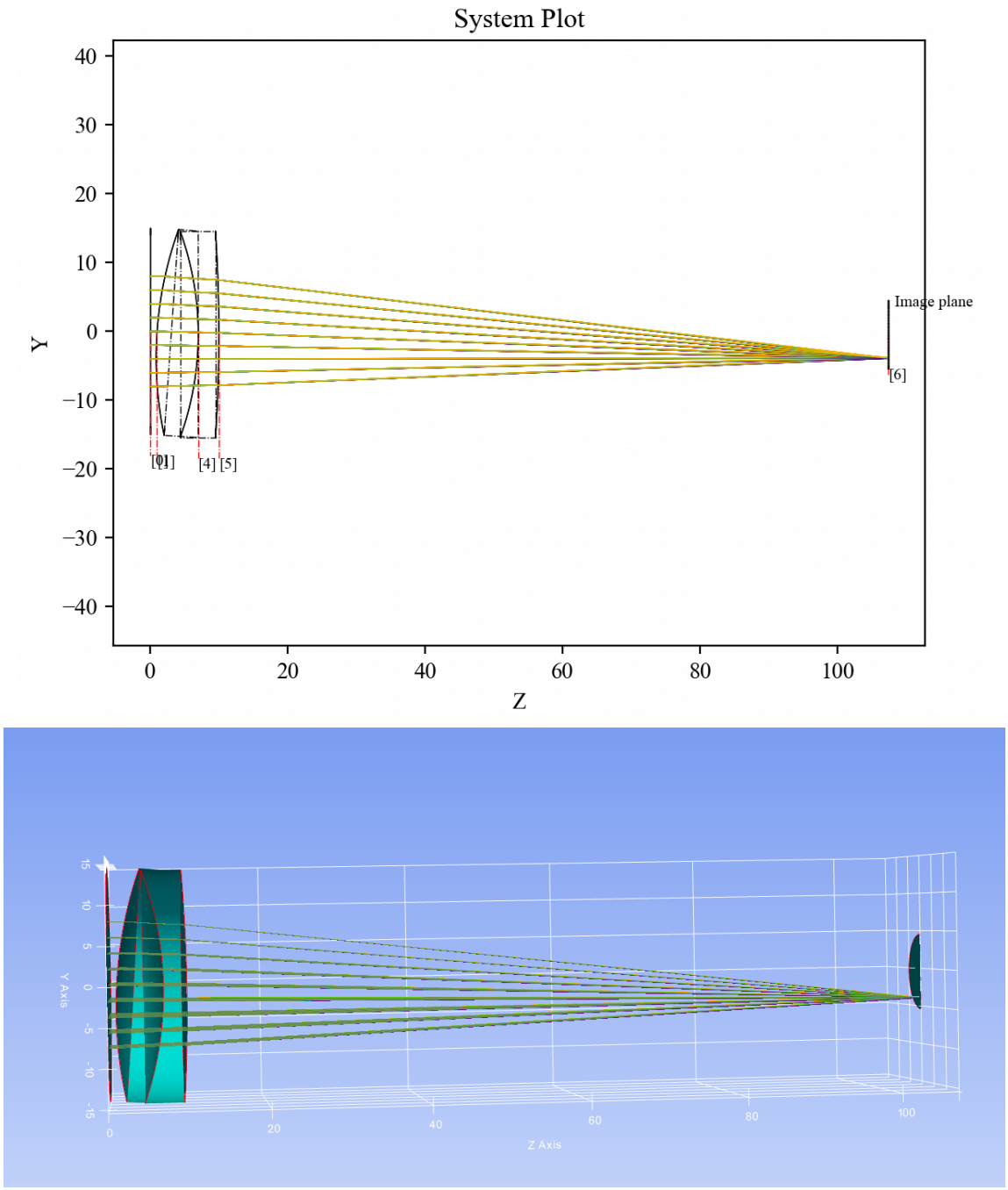


Figure 13. 2D and 3D view of a doublet with a tilted face.

G. EXAMPLE - DOUBLET LENS NONSEC

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
Examp Doublet Lens NonSec"""
import numpy as np
import Kraken as Kn

P_Obj = Kn.surf()
P_Obj.Rc = 0.0
P_Obj.Thickness = 0
P_Obj.Glass = "AIR"
P_Obj.Diameter = 30.0

P_Obj2 = Kn.surf()
P_Obj2.Rc = 0.0
P_Obj2.Thickness = 10
P_Obj2.Glass = "AIR"
P_Obj2.Diameter = 100.0

L1a = Kn.surf()
L1a.Rc = 9.284706570002484E+001
L1a.Thickness = 6.0
L1a.Glass = "BK7"
L1a.Diameter = 30.0
L1a.Axicon = 0

L1b = Kn.surf()
L1b.Rc = -3.071608670000159E+001
L1b.Thickness = 3.0
L1b.Glass = "F2"
L1b.Diameter = 30

L1c = Kn.surf()
L1c.Rc = -7.819730726078505E+001
L1c.Thickness = 9.737604742910693E+001
L1c.Glass = "AIR"
L1c.Diameter = 30

P_Ima = Kn.surf()
P_Ima.Rc = 0.0
P_Ima.Thickness = 0.0
P_Ima.Glass = "MIRROR"
P_Ima.Diameter = 30.0
P_Ima.Name = "Plano imagen"
P_Ima.DespZ = 10
P_Ima.TiltX = 6.

A = [P_Obj, P_Obj2, L1a, L1b, L1c, P_Ima]
configuration_1 = Kn.Kraken_setup()

Doublet = Kn.system(A, configuration_1)
Rays = Kn.raykeeper(Doblete)

tam = 10
rad = 14.0
tsis = len(A) - 1
for nsc in range(0, 100):
    for j in range(-tam, tam + 1):
        x_0 = (0 / tam) * rad
        y_0 = (j / tam) * rad
        r = np.sqrt((x_0 * x_0) + (y_0 * y_0))
        if r < rad:
            tet = 0.0
            pSource_0 = [x_0, y_0, 0.0]
            dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]
            W = 0.4
            Doublet.NsTrace(pSource_0, dCos, W)
            Rays.push()


Kn.display2d(Doblete, Rays, 0)
```

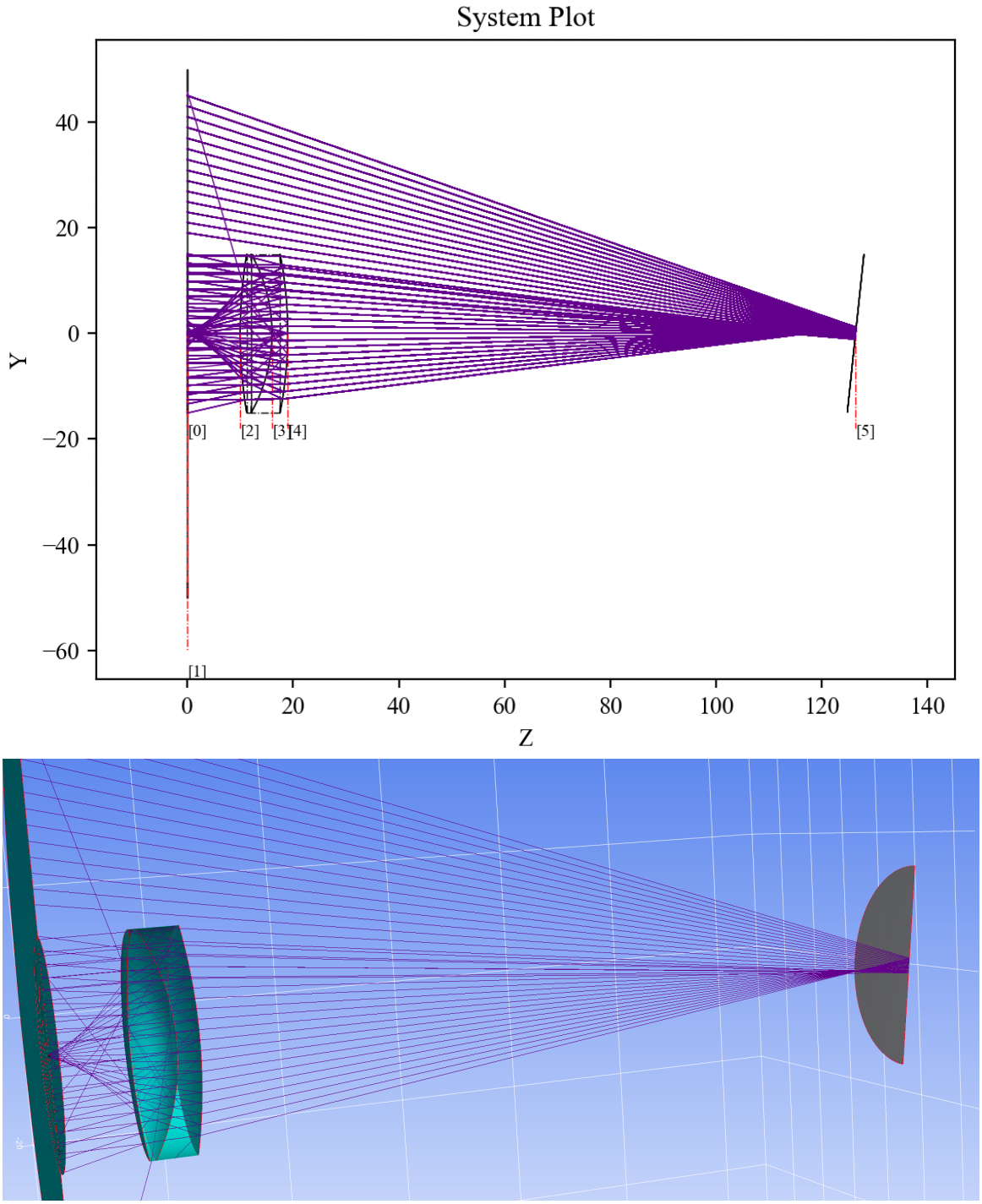


Figure 14. 2D and 3D display of a raytracing nonsequential, some rays are reflected back depending on the value in the coefficients and Fresnel, these depend on the wavelength, the material and the angle of the beam with respect to the normal l to surface at the point of intersection.

H. EXAMPLE - DOUBLET LENS ZERNIKE

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""Doublet Lens Zernike"""
```

```
import Kraken as Kn
import numpy as np
import matplotlib.pyplot as plt
import time

P_Obj = Kn.surf()
P_Obj.Rc = 0.0
P_Obj.Thickness = 1
P_Obj.Glass = "AIR"
P_Obj.Diameter = 30.0

L1a = Kn.surf()
L1a.Rc = 5.513435044607768E+001
L1a.Thickness = 6.0
L1a.Glass = "BK7"
L1a.Diameter = 30.0

L1b = Kn.surf()
L1b.Rc = -4.408716526030626E+001
L1b.Thickness = 3.0
L1b.Glass = "F2"
L1b.Diameter = 30

L1c = Kn.surf()
L1c.Rc = -2.246906271406796E+002
L1c.Thickness = 9.737871661422000E+001
L1c.Glass = "AIR"
L1c.Diameter = 30

Z = np.zeros(36)
Z[8] = 0.5
Z[9] = 0.5
Z[10] = 0.5
Z[11] = 0.5
Z[12] = 0.5
Z[13] = 0.5
Z[14] = 0.5
Z[15] = 0.5
L1c.ZNK = Z

P_Ima = Kn.surf()
P_Ima.Rc = 0.0
P_Ima.Thickness = 0.0
P_Ima.Glass = "AIR"
P_Ima.Diameter = 100.0
P_Ima.Name = "Plano imagen"

A = [P_Obj, L1a, L1b, L1c, P_Ima]
configuration_1 = Kn.Kraken_setup()

Doublet = Kn.system(A, configuration_1)
Rays = Kn.raykeeper(Doblete)

tam = 5
rad = 12.0
tsis = len(A) - 1
for i in range(-tam, tam + 1):
    for j in range(-tam, tam + 1):
        x_0 = (i / tam) * rad
        y_0 = (j / tam) * rad
        r = np.sqrt((x_0 * x_0) + (y_0 * y_0))
        if r < rad:
            tet = 0.0
            pSource_0 = [x_0, y_0, 0.0]
            dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]
            W = 0.4
            Doublet.Trace(pSource_0, dCos, W)
            Rays.push()

Kn.display3d(Doblete, Rays, 2)
Kn.display2d(Doblete, Rays, 0)
```

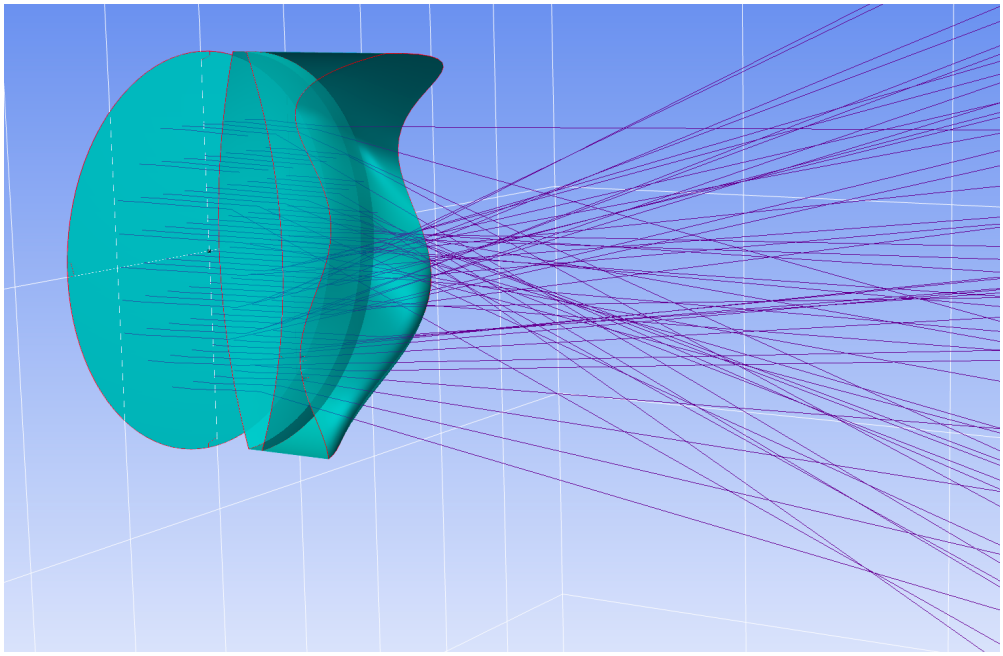


Figure 15. Example of a lens with a surface defined by coefficients of the Zernike polynomials 16 . Non-sequential ray tracing example, the rays that do not touch the first element are crossed to a second element where they do touch the surface.

I. EXAMPLE - DOUBLET LENS TILT NONSEC

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
Examp Doublet Lens Tilt nonSec"""
import numpy as np
import Kraken as Kn


P_Obj = Kn.surf()
P_Obj.Rc = 0.0
P_Obj.Thickness = 10
P_Obj.Glass = "AIR"
P_Obj.Diameter = 30.0

L1a = Kn.surf()
L1a.Rc = 9.284706570002484E+001
L1a.Thickness = 6.0
L1a.Glass = "BK7"
L1a.Diameter = 30.0
L1a.AxisMove = 1
L1a.TiltX = 13.0
L1a.DespZ = 5.0

L1b = Kn.surf()
L1b.Rc = (-3.071608670000159E+001)
L1b.Thickness = 3.0
L1b.Glass = "F2"
L1b.Diameter = 30

L1c = Kn.surf()
L1c.Rc = (-7.819730726078505E+001)
L1c.Thickness = 9.737604742910693E+001
L1c.Glass = "AIR"
L1c.Diameter = 30

P_Ima = Kn.surf()
P_Ima.Rc = 0.0
P_Ima.Thickness = 0.0
```

```
P_Ima.Glass = "AIR"
P_Ima.Diameter = 1000.0
P_Ima.Name = "Plano imagen"

A = [P_Obj, L1a, L1b, L1c, P_Obj, L1a, L1b, L1c, P_Ima]
configuration_1 = Kn.Kraken_setup()

Doublet = Kn.system(A, configuration_1)
Rays = Kn.raykeeper(Doblete)

tam = 30
rad = 18.0
tsis = len(A) - 1
for j in range(-tam, tam + 1):
    i = 0
    x_0 = (i / tam) * rad
    y_0 = (j / tam) * rad
    r = np.sqrt((x_0 * x_0) + (y_0 * y_0))
    if r < rad:
        tet = 0.0
        pSource_0 = [x_0, y_0, 0.0]
        dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]
        W = 0.4
        Doublet.NsTrace(pSource_0, dCos, W)
        Rays.push()
        W = 0.5
        Doublet.NsTrace(pSource_0, dCos, W)
        Rays.push()
        W = 0.6
        Doublet.NsTrace(pSource_0, dCos, W)
        Rays.push()

Kn.display3d(Doblete, Rays, 2)
```

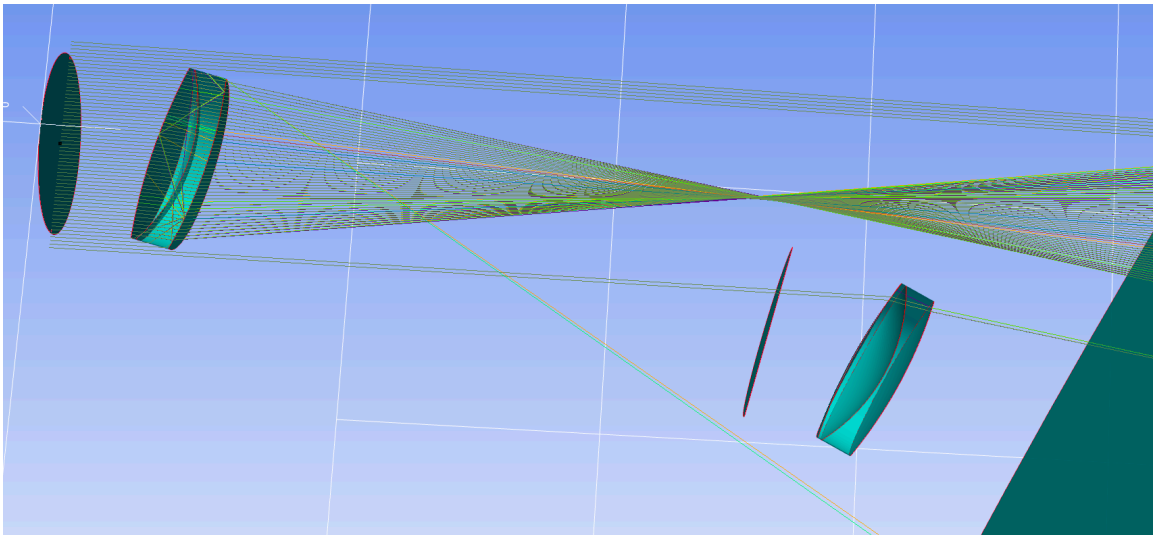


Figure 16. Non-sequential ray tracing example, the rays that do not touch the first element are crossed to a second element where they do touch the surface.

J. EXAMPLE - DOUBLET LENS PUPIL

```
#!/usr/bin/env python3
```

```
# -*- coding: utf-8 -*-
Examp Doublet Lens Pupil"""
import Kraken as Kn

P_Obj = Kn.surf()
P_Obj.Rc = 0.0
P_Obj.Thickness = 100
P_Obj.Glass = "AIR"
P_Obj.Diameter = 30.0
P_Obj.Name = "P Obj"

L1a = Kn.surf()
L1a.Rc = 9.284706570002484E+001
L1a.Thickness = 6.0
L1a.Glass = "BK7"
L1a.Diameter = 30.0
L1a.Axicon = 0

L1b = Kn.surf()
L1b.Rc = -3.071608670000159E+001
L1b.Thickness = 3.0
L1b.Glass = "F2"
L1b.Diameter = 30

L1c = Kn.surf()
L1c.Rc = -7.819730726078505E+001
L1c.Thickness = 9.737604742910693E+001 - 40
L1c.Glass = "AIR"
L1c.Diameter = 30

pupila = Kn.surf()
pupila.Rc = 30
pupila.Thickness = 40.
pupila.Glass = "AIR"
pupila.Diameter = 5
pupila.Name = "Ap Stop"
pupila.DespY = 0.

P_Ima = Kn.surf()
P_Ima.Rc = 0.0
P_Ima.Thickness = 0.0
P_Ima.Glass = "AIR"
P_Ima.Diameter = 20.0
P_Ima.Name = "Plano imagen"

A = [P_Obj, L1a, L1b, L1c, pupila, P_Ima]
config_1 = Kn.Kraken_setup()

Doublet = Kn.system(A, config_1)
Rays = Kn.raykeeper(Doblete)

W = 0.4
Surf= 4
AperVal = 10
AperType = "EPD"
Pup = Kn.PupilCalc(Doblete, sup, W, AperType, AperVal)

print("Radio pupila de entrada: ")
print(Pup.RadPupInp)
print("Posición pupila de entrada: ")
print(Pup.PosPupInp)
print("Rádio pupila de salida: ")
print(Pup.RadPupOut)
print("Posicion pupila de salida: ")
print(Pup.PosPupOut)
print("Posicion pupila de salida respecto al plano focal: ")
print(Pup.PosPupOutFoc)
print("Orientación pupila de salida")
print(Pup.DirPupSal)

[L, M, N] = Pup.DirPupSal
print(L, M, N)

Pup.Samp = 5
Pup.Ptype = "fan"
```

```
Pup.FieldType = "angle"
Pup.FieldY = 2.0
x, y, z, L, M, N = Pup.Pattern2Field()
for i in range(0, len(x)):
    pSource_0 = [x[i], y[i], z[i]]
    dCos = [L[i], M[i], N[i]]
    Doublet.Trace(pSource_0, dCos, W)
    Rays.push()

Pup.FieldY = -2.0
x, y, z, L, M, N = Pup.Pattern2Field()
for i in range(0, len(x)):
    pSource_0 = [x[i], y[i], z[i]]
    dCos = [L[i], M[i], N[i]]
    Doublet.Trace(pSource_0, dCos, W)
    Rays.push()

Kn.display2d(Doblete, Rays, 0)
```

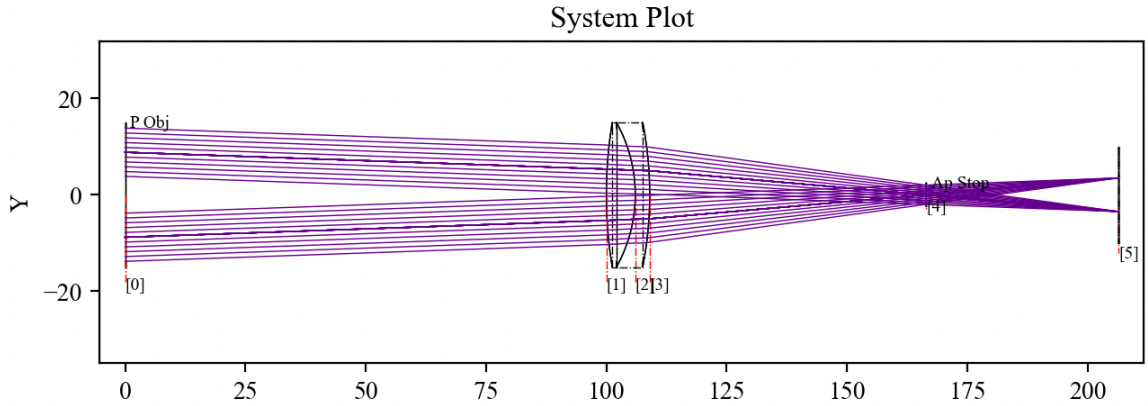


Figure 17. Example of beams of rays generated with the pupil function, in which the place where the pupil or the opening of the system is located is defined.

K. EXAMPLE - DOUBLET LENS COMMANDS SYSTEM

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
Examp Doublet Lens Commands System"""
import numpy as np
import Kraken as Kn


P_Obj = Kn.surf()
P_Obj.Rc = 0.0
P_Obj.Thickness = 0.1
P_Obj.Glass = "AIR"
P_Obj.Diameter = 30.

P_Obj2 = Kn.surf()
P_Obj2.Rc = 0.0
P_Obj2.Thickness = 10
P_Obj2.Glass = "AIR"
P_Obj2.Diameter = 30.0

L1a = Kn.surf()
L1a.Rc = 9.284706570002484E+001
L1a.Thickness = 6.0
L1a.Glass = "BK7"
L1a.Diameter = 30.0
L1a.Axicon = 0

L1b = Kn.surf()
L1b.Rc = -3.071608670000159E+001
```

```
L1b.Thickness = 3.0
L1b.Glass = "F2"
L1b.Diameter = 30

L1c = Kn.surf()
L1c.Rc = -7.819730726078505E+001
L1c.Thickness = 9.737604742910693E+001
L1c.Glass = "AIR"
L1c.Diameter = 30

P_Ima = Kn.surf()
P_Ima.Rc = 0.0
P_Ima.Thickness = 0.0
P_Ima.Glass = "AIR"
P_Ima.Diameter = 18.0
P_Ima.Name = "Plano imagen"

A = [P_Obj, P_Obj2, L1a, L1b, L1c, P_Ima]
configuration_1 = Kn.Kraken_setup()

Doublet = Kn.system(A, configuration_1)
Rays = Kn.raykeeper(Doblete)

pSource_0 = [0, 14, 0]
tet = 0.1
dCos = [0.0, np.sin(np.deg2rad(tet)), -np.cos(np.deg2rad(tet))]
W = 0.4
Doublet.Trace(pSource_0, dCos, W)
Rays.push()

Kn.display3d(Doblete, Rays, 2)

print("Distancia focal efectiva")
print(Doublet.EFFL)
print("Plano principal anterior")
print(Doublet.PPA)
print("Plano principal posterior")
print(Doublet.PPP)
print("Superficies tocadas por el rayo")
print(Doublet.SURFACE)
print("Nombre de la superficie")
print(Doublet.NAME)
print("Vidrio de la superficie")
print(Doublet.GLASS)
print("Coordenadas del rayo en las superficies")
print(Doublet.XYZ)
print("Etc, ver documentaciòn")
print(Doublet.S_XYZ)
print(Doublet.T_XYZ)
print(Doublet.OST_XYZ)
print(Doublet.DISTANCE)
print(Doublet.OP)
print(Doublet.TOP)
print(Doublet.TOP_S)
print(Doublet.ALPHA)
print(Doublet.S_LMN)
print(Doublet.LMN)
print(Doublet.R_LMN)
print(Doublet.N0)
print(Doublet.N1)
print(Doublet.WAV)
print(Doublet.G_LMN)
print(Doublet.ORDER)
print(Doublet.GRATING)
print(Doublet.RP)
print(Doublet.RS)
print(Doublet.TP)
print(Doublet.TS)
print(Doublet.TTBE)
print(Doublet.TT)
print(Doublet.BULK_TRANS)
```

L. EXAMPLE - DOUBLET LENS PUPIL SEIDEL

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
Examp Doublet Lens Pupil Seidel"""
import Kraken as Kn
import numpy as np


P_Obj = Kn.surf()
P_Obj.Rc = 0.0
P_Obj.Thickness = 100
P_Obj.Glass = "AIR"
P_Obj.Diameter = 30.0
P_Obj.Name = "P Obj"

L1a = Kn.surf()
L1a.Rc = 9.284706570002484E+001
L1a.Thickness = 6.0
L1a.Glass = "N-BK7"
L1a.Diameter = 30.0
L1a.Axicon = 0

L1b = Kn.surf()
L1b.Rc = -3.071608670000159E+001
L1b.Thickness = 3.0
L1b.Glass = "F2"
L1b.Diameter = 30

L1c = Kn.surf()
L1c.Rc = -7.819730726078505E+001
L1c.Thickness = 9.737604742910693E+001 - 40
L1c.Glass = "AIR"
L1c.Diameter = 30

pupila = Kn.surf()
pupila.Rc = 0
pupila.Thickness = 40.
pupila.Glass = "AIR"
pupila.Diameter = 15.0
pupila.Name = "Ap Stop"

P_Ima = Kn.surf()
P_Ima.Rc = 0.0
P_Ima.Thickness = 0.0
P_Ima.Glass = "AIR"
P_Ima.Diameter = 20.0
P_Ima.Name = "Plano imagen"

A = [P_Obj, L1a, L1b, L1c, pupila, P_Ima]
config_1 = Kn.Kraken_setup()

Doublet = Kn.system(A, config_1)

W = 0.6
Surf= 4
AperVal = 3
AperType = "EPD"
field = 3.25
fieldType = "angle"

AB = Kn.Seidel(Doblete, sup, W, AperType, AperVal, field, fieldType)
print( AB[0][0])
print(np.sum(AB[1][0]), np.sum(AB[1][1]), np.sum(AB[1][2]), np.sum(AB[1][3]), np.sum(AB[1][4]))
```

```
    j=1
    print( AB[0][0+j])
    print(np.sum(AB[1+j][0]), np.sum(AB[1+j][1]), np.sum(AB[1+j][2]), np.sum(AB[1+j][3]),
np.sum(AB[1+j][4]))
    j=2
    print( AB[0][0+j])
    print(np.sum(AB[1+j][0]), np.sum(AB[1+j][1]), np.sum(AB[1+j][2]), np.sum(AB[1+j][3]),
np.sum(AB[1+j][4]))
    j=3
    print( AB[0][0+j])
    print(np.sum(AB[1+j][0]), np.sum(AB[1+j][1]), np.sum(AB[1+j][2]), np.sum(AB[1+j][3]),
np.sum(AB[1+j][4]))

    Pup = Kn.PupilCalc(Doblete, sup, W, AperType, AperVal)
    Pup.Samp = 25
    Pup.Ptype = "fan"
    Pup.FieldY = field
    x, y, z, L, M, N = Pup.Pattern2Field()
    Rays = Kn.raykeeper(Doblete)

    for i in range(0, len(x)):
        pSource_0 = [x[i], y[i], z[i]]
        dCos = [L[i], M[i], N[i]]
        Doublet.Trace(pSource_0, dCos, W)
        Rays.push()

    Kn.display2d(Doblete, Rays, 0)
```

```
--------------------------------------
Seidel Aberration Coefficents
['si', 'sii', 'siii', 'siv', 'sv']
[-1.924670463514677e-06, -2.7721266737423735e-05, -0.0003183543169943356,
-5.20847647729677e-05, -0.0033844088571035732]
--------------------------------------
Seidel coefficients in waves
['W040', 'W131', 'W222', 'W220', 'W311']
[-0.0004009730132322241, -0.023101055614519778, -0.2652952641619465,
-0.02170198532206988, -2.8203407142529766]
--------------------------------------
Transverse Aberration Coefficents
['TSPH', 'TSCO', 'TTCO', 'TAST', 'TPFC', 'TSFC', 'TTFC', 'TDIS']
[6.41640871597908e-05, 0.0009241632834494345, 0.0027724898503483017,
-0.021226401641783688, -0.0017363862801152308, 0.012349587101007076,
0.03357598874279076, -0.11282840829541621]
--------------------------------------
Longitudinal Aberration Coefficents
['LSPH', 'LSCO', 'LTCO', 'LAST', 'LPFC', 'LSFC', 'LTFC', 'LDIS']
[-0.0042781662202380916, -0.06161895721186755, -0.1848568716356025,
-1.4152788343257612, -0.11577425095091264, -0.8234136681137931,
-2.2386925024395543, 7.522879330467116]
--------------------------------------
```

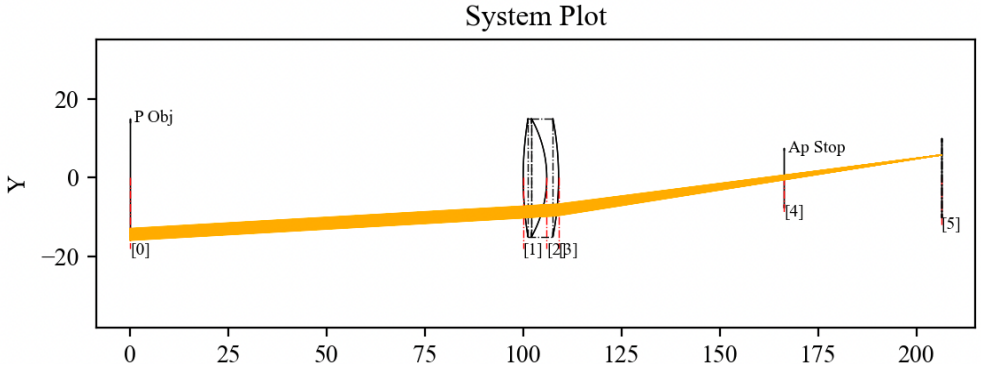
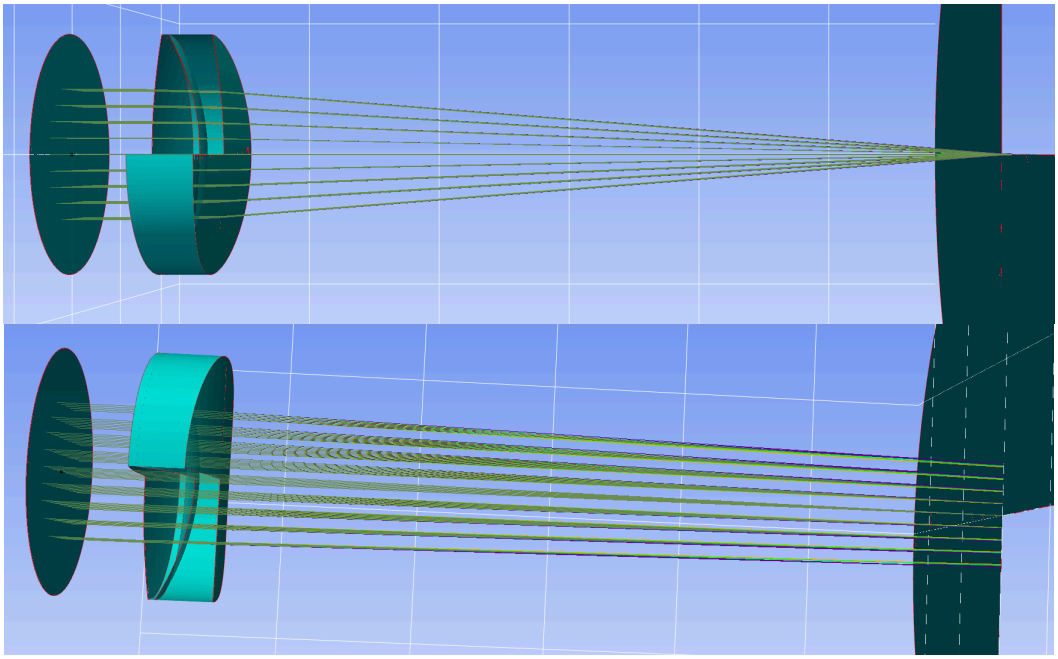


Figure 18. (Above) Seidel sums values. (Bottom) Doublet with which aberrations were calculated using that field delimited by the image of the pupil.

M. EXAMPLE - DOUBLET LENS CYLINDER

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
Examp Doublet Lens Cylinder"""
import numpy as np
import Kraken as Kn


P_Obj = Kn.surf()
P_Obj.Rc = 0.0
P_Obj.Thickness = 10
P_Obj.Glass = "AIR"
P_Obj.Diameter = 30.0

L1a = Kn.surf()
L1a.Rc = 9.284706570002484E+001
L1a.Thickness = 6.0
L1a.Glass = "BK7"
L1a.Diameter = 30.0

L1b = Kn.surf()
L1b.Rc = -3.071608670000159E+001
L1b.Thickness = 3.0
L1b.Glass = "F2"
L1b.Diameter = 30
L1b.TiltZ = 30
L1b.AxisMove = 0

L1c = Kn.surf()
L1c.Rc = -7.819730726078505E+001
L1c.Thickness = 9.737604742910693E+001
L1c.Glass = "AIR"
L1c.Diameter = 30
L1c.Cylinder_Rxy_Ratio = 0
L1c.TiltZ = 45
L1c.AxisMove = 0

P_Ima = Kn.surf()
P_Ima.Rc = 0.0
P_Ima.Thickness = 0.0
P_Ima.Glass = "AIR"
P_Ima.Diameter = 100.0
P_Ima.Name = "Plano imagen"

A = [P_Obj, L1a, L1b, L1c, P_Ima]
configuration_1 = Kn.Kraken_setup()

Doublet = Kn.system(A, configuration_1)
Rays = Kn.raykeeper(Doblete)

tam = 5
rad = 10.0
tsis = len(A) - 1
for i in range(-tam, tam + 1):
    for j in range(-tam, tam + 1):
        x_0 = (i / tam) * rad
        y_0 = (j / tam) * rad
        r = np.sqrt((x_0 * x_0) + (y_0 * y_0))
        if r < rad:
            tet = 0.0
```

```
            pSource_0 = [x_0, y_0, 0.0]
            dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]
            W = 0.4
            Doublet.Trace(pSource_0, dCos, W)
            Rays.push()
            W = 0.5
            Doublet.Trace(pSource_0, dCos, W)
            Rays.push()
            W = 0.6
            Doublet.Trace(pSource_0, dCos, W)
            Rays.push()

Kn.display3d(Doblete, Rays, 1)
```



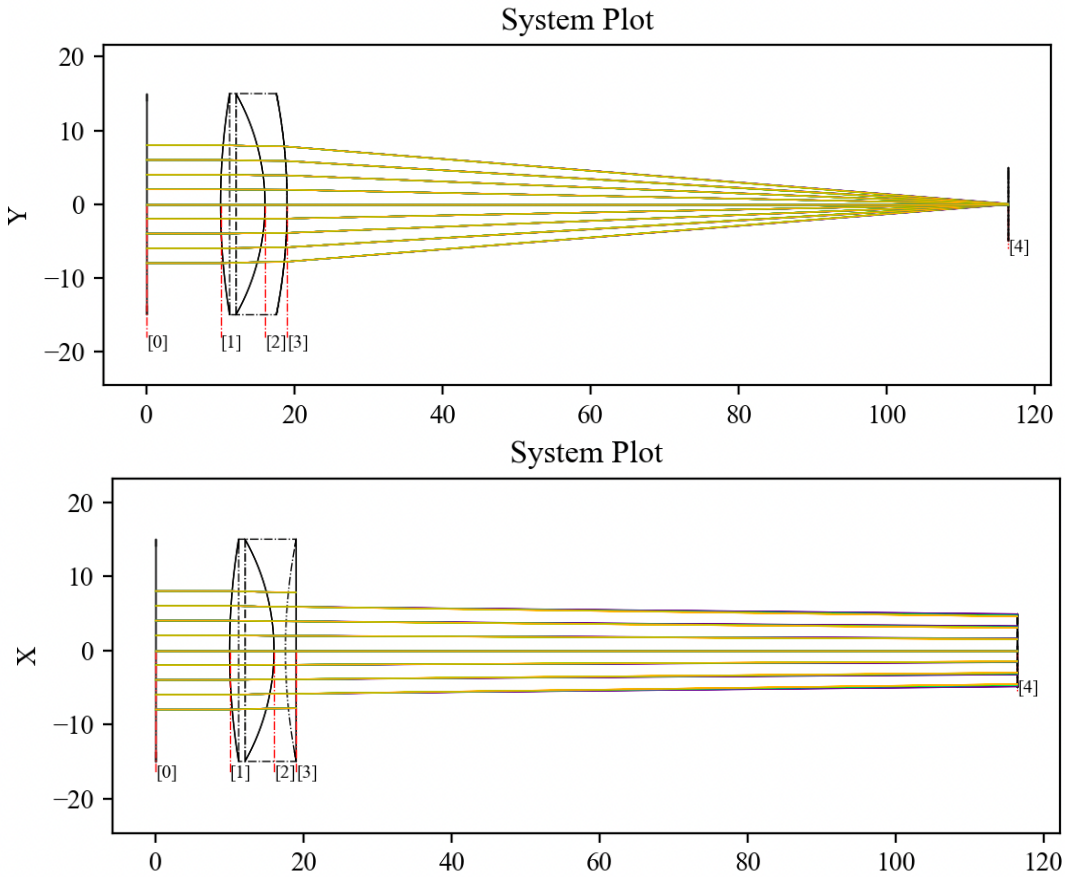


Figure 19. 2D and 3D visualization seen in the direction of the X axis and later the Y axis. The last face of the lens has a radius of curvature that can be seen from the Y axis, this is flat when viewed from the X axis..

N. EXAMPLE - AXICON

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
Examp Axicon"""
import numpy as np
import Kraken as Kn

P_Obj = Kn.surf()
P_Obj.Rc = 0.0
P_Obj.Thickness = 10
P_Obj.Glass = "AIR"
P_Obj.Diameter = 30.0

L1a = Kn.surf()
L1a.Rc = 0
L1a.Thickness = 26.0
L1a.Glass = "BK7"
L1a.Diameter = 30.0

L1c = Kn.surf()
L1c.Rc = 0
L1c.Thickness = 9.737604742910693E+001
L1c.Axicon = -35.0
L1c.ShiftY = 0
L1c.Glass = "AIR"
L1c.Diameter = 30
```

```
P_Ima = Kn.surf()
P_Ima.Rc = 0.0
P_Ima.Thickness = 0.0
P_Ima.Glass = "AIR"
P_Ima.Diameter = 100.0
P_Ima.Name = "Plano imagen"

configuration_1 = Kn.Kraken_setup()
A = [P_Obj, L1a, L1c, P_Ima]

Doublet = Kn.system(A, configuration_1)
Rays = Kn.raykeeper(Doblete)

tam = 5
rad = 10.0
tsis = len(A) - 1
for i in range(-tam, tam + 1):
    for j in range(-tam, tam + 1):
        x_0 = (i / tam) * rad
        y_0 = (j / tam) * rad
        r = np.sqrt((x_0 * x_0) + (y_0 * y_0))
        if r < rad:
            tet = 0.0
            pSource_0 = [x_0, y_0, 0.0]
            dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]
            W = 0.4
            Doublet.Trace(pSource_0, dCos, W)
            Rays.push()
            W = 0.5
            Doublet.Trace(pSource_0, dCos, W)
            Rays.push()
            W = 0.6
            Doublet.Trace(pSource_0, dCos, W)
            Rays.push()

Kn.display3d(Doblete, Rays, 2)
```

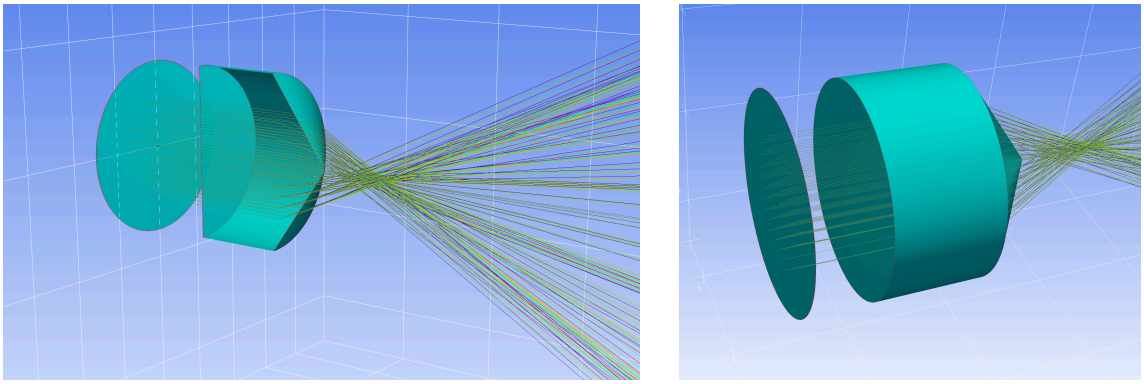


Figure 20. 3D view of an Axicon, complete and cross-sectional view.

O. EXAMPLE - AXICON AND CYLINDER

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
Examp Axicon and Cylinder"""
import numpy as np
import Kraken as Kn

configuration_1 = Kn.Kraken_setup()

P_Obj = Kn.surf()
P_Obj.Rc = 0.0
P_Obj.Thickness = 10
P_Obj.Glass = "AIR"
```

```
P_Obj.Diameter = 30.0

L1a = Kn.surf()
L1a.Rc = 0
L1a.Thickness = 26.0
L1a.Glass = "BK7"
L1a.Diameter = 30.0


L1c = Kn.surf()
L1c.Rc = 0.
L1c.K = -1
L1c.Thickness = 9.737604742910693E+001
L1c.Axicon = (-35.0)
L1c.ShiftY = 0
L1c.Cylinder_Rxy_Ratio = 0
L1c.Glass = "AIR"
L1c.Diameter = 30

P_Ima = Kn.surf()
P_Ima.Rc = 0.0
P_Ima.Thickness = 0.0
P_Ima.Glass = "AIR"
P_Ima.Diameter = 100.0
P_Ima.Name = "Plano imagen"

A = [P_Obj, L1a, L1c, P_Ima]

Doublet = Kn.system(A, configuration_1)
Rays = Kn.raykeeper(Doblete)

tam = 5
rad = 10.0
tsis = len(A) - 1
for i in range(-tam, tam + 1):
    for j in range(-tam, tam + 1):
        x_0 = (i / tam) * rad
        y_0 = (j / tam) * rad
        r = np.sqrt((x_0 * x_0) + (y_0 * y_0))
        if r < rad:
            tet = 0.0
            pSource_0 = [x_0, y_0, 0.0]
            dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]
            W = 0.4
            Doublet.Trace(pSource_0, dCos, W)
            Rays.push()
            W = 0.5
            Doublet.Trace(pSource_0, dCos, W)
            Rays.push()
            W = 0.6
            Doublet.Trace(pSource_0, dCos, W)
            Rays.push()

Kn.display3d(Doblete, Rays, 0)
```

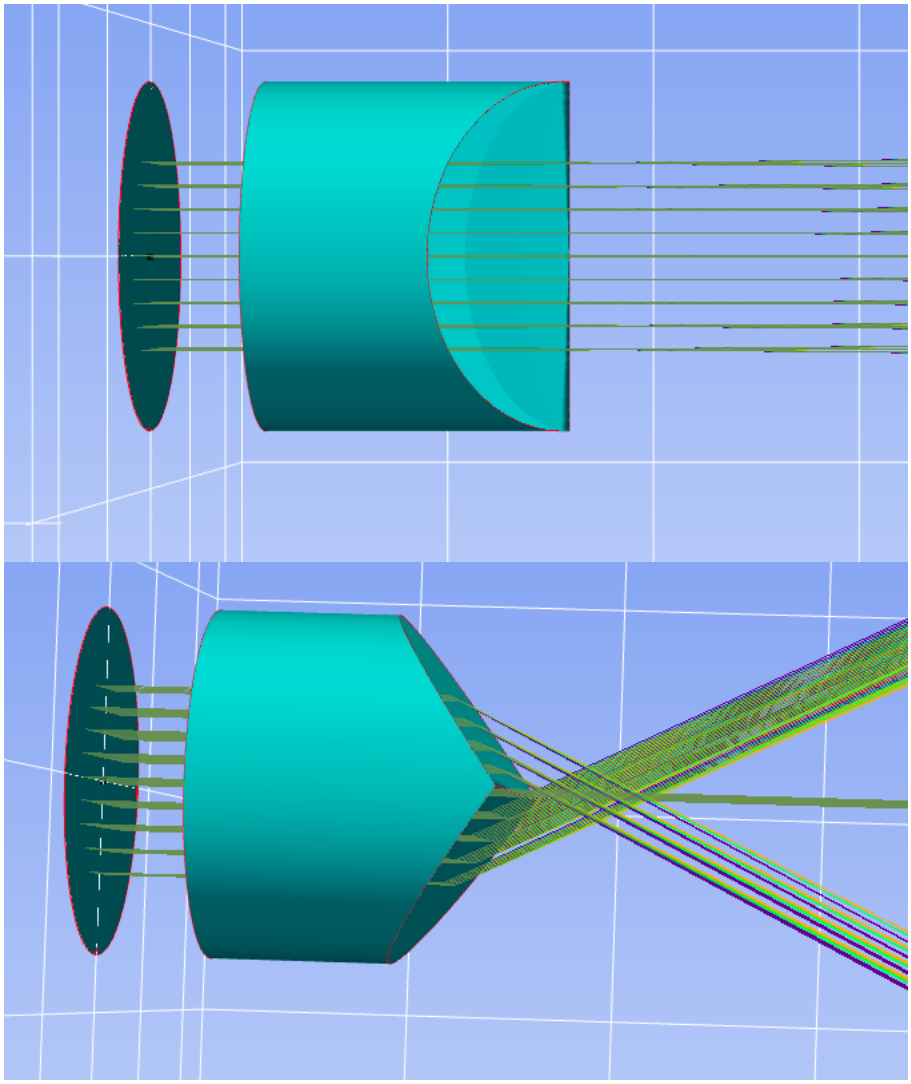


Figure 21. Example of a cylindrical lens combined with axicon.

P. EXAMPLE - FLAT MIRROR 45 DEG

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
Examp Flat Mirror 45 Deg"""
import time
import matplotlib.pyplot as plt
import numpy as np
import Kraken as Kn

start_time = time.time()


P_Obj = Kn.surf()
P_Obj.Rc = 0.0
P_Obj.Thickness = 10
P_Obj.Glass = "AIR"
P_Obj.Diameter = 30.0

L1a = Kn.surf()
L1a.Rc = 9.284706570002484E+001
L1a.Thickness = 6.0
L1a.Glass = "BK7"
```

```
L1a.Diameter = 30.0
L1a.Axicon = 0

L1b = Kn.surf()
L1b.Rc = -3.071608670000159E+001
L1b.Thickness = 3.0
L1b.Glass = "F2"
L1b.Diameter = 30

POS_ESP = -40
L1c = Kn.surf()
L1c.Rc = -7.819730726078505E+001
L1c.Thickness = 9.737604742910693E+001 + POS_ESP
L1c.Glass = "AIR"
L1c.Diameter = 30

Esp90 = Kn.surf()
Esp90.Thickness = POS_ESP
Esp90.Glass = "MIRROR"
Esp90.Diameter = 30.0
Esp90.Name = "Espejo a 90 grados"
Esp90.TiltX = 45.
Esp90.AxisMove = 2.

P_Ima = Kn.surf()
P_Ima.Rc = 0.0
P_Ima.Thickness = 0.0
P_Ima.Glass = "AIR"
P_Ima.Diameter = 3.0
P_Ima.Name = "Plano imagen"

A = [P_Obj, L1a, L1b, L1c, Esp90, P_Ima]
config_1 = Kn.Kraken_setup()

Doublet = Kn.system(A, config_1)
Rays1 = Kn.raykeeper(Doblete)
Rays2 = Kn.raykeeper(Doblete)
Rays3 = Kn.raykeeper(Doblete)
RaysT = Kn.raykeeper(Doblete)

tam = 10
rad = 10.0
tsis = len(A) - 1
for j in range(-tam, tam + 1):
    for i in range(-tam, tam + 1):
        x_0 = (i / tam) * rad
        y_0 = (j / tam) * rad
        r = np.sqrt((x_0 * x_0) + (y_0 * y_0))
        if r < rad:
            tet = 0.0
            pSource_0 = [x_0, y_0, 0.0]
            dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]
            W = 0.4
            Doublet.Trace(pSource_0, dCos, W)
            Rays1.push()
            RaysT.push()
            W=0.5
            Doublet.Trace(pSource_0, dCos,W)
            Rays2.push()
            RaysT.push()
            W=0.6
            Doublet.Trace(pSource_0, dCos,W)
            Rays3.push()
            RaysT.push()

Kn.display3d(Doblete, RaysT, 2)

X, Y, Z, L, M, N = Rays1.pick(-1)
plt.plot(X, Z, 'x')
X, Y, Z, L, M, N = Rays2.pick(-1)
plt.plot(X, Z, 'x')
X, Y, Z, L, M, N = Rays3.pick(-1)
plt.plot(X, Z, 'x')
plt.xlabel('numbers')
plt.ylabel('values')
```

```
plt.title('Spot Diagram')
plt.axis('square')
plt.show()
```

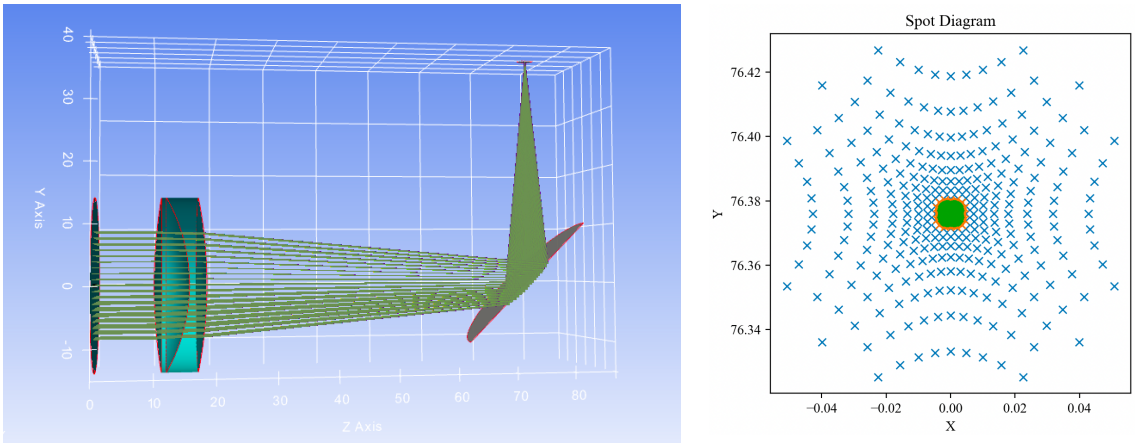


Figure 22. (Left) Example of a diagonal mirror rotated by 45 °, (Right) Spot diagram for three different wavelengths, 0.4 µm, 0.5 µm and 0.6 µm.

Q. EXAMPLE - PARABOLE MIRROR SHIFT

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
Examp Parabole Mirror Shift"""
import numpy as np
import Kraken as Kn


P_Obj = Kn.surf()
P_Obj.Thickness = 1000.0
P_Obj.Diameter = 300
P_Obj.Drawing = 0

M1 = Kn.surf()
M1.Rc = -2000.0
M1.Thickness = M1.Rc / 2
M1.k = -1.0
M1.Glass = "MIRROR"
M1.Diameter = 300
M1.ShiftY = 200

P_Ima = Kn.surf()
P_Ima.Glass = "AIR"
P_Ima.Diameter = 1600.0
P_Ima.Drawing = 0
P_Ima.Name = "Plano imagen"

A = [P_Obj, M1, P_Ima]
configuration_1 = Kn.Kraken_setup()

Espejo = Kn.system(A, configuration_1)
Rays = Kn.raykeeper(Espejo)

tam = 5
rad = 150.0
tsis = len(A) - 1
for i in range(-tam, tam + 1):
    for j in range(-tam, tam + 1):
        x_0 = (i / tam) * rad
        y_0 = (j / tam) * rad
        r = np.sqrt((x_0 * x_0) + (y_0 * y_0))
        if r < rad:
            tet = 0.0
            pSource_0 = [x_0, y_0, 0.0]
            dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]
```

```
            W = 0.4
            Espejo.Trace(pSource_0, dCos, W)
            Rays.push()

Kn.display3d(Espejo, Rays, 0)
```

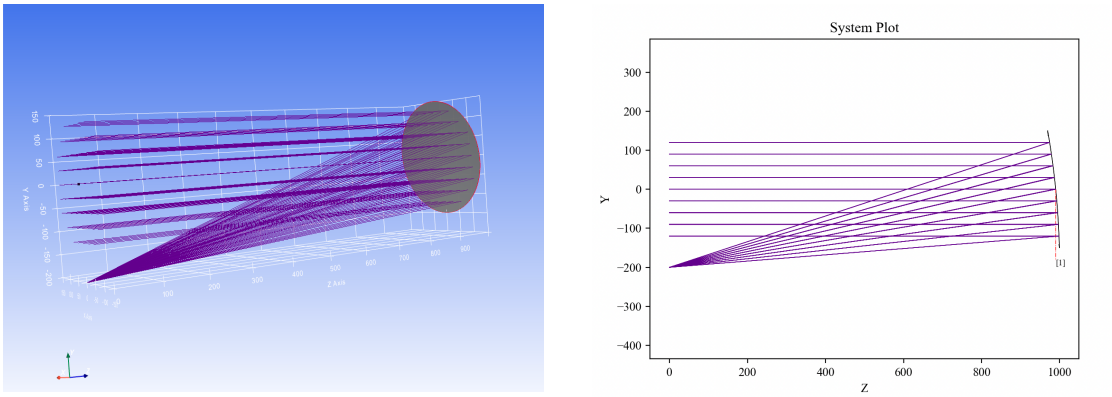


Figure 23. 3D and 2D view of an off-axis parabola..

R. EXAMPLE - DIFFRACTION GRATING TRANSMISSION

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
Examp Diffraction Grating Transmission"""
import numpy as np
import Kraken as Kn

P_Obj = Kn.surf()
P_Obj.Rc = 0.0
P_Obj.Thickness = 10
P_Obj.Glass = "AIR"
P_Obj.Diameter = 30.0

Dif_Obj_c1 = Kn.surf()
Dif_Obj_c1.Rc = 0.0
Dif_Obj_c1.Thickness = 1
Dif_Obj_c1.Glass = "BK7"
Dif_Obj_c1.Diameter = 30.0
Dif_Obj_c1.Grating_D = 1.0
Dif_Obj_c1.Diff_Ord = 1.
Dif_Obj_c1.Grating_Angle = 45.

Dif_Obj_c2 = Kn.surf()
Dif_Obj_c2.Rc = 0.0
Dif_Obj_c2.Thickness = 10
Dif_Obj_c2.Glass = "AIR"
Dif_Obj_c2.Diameter = 30.0

L1a = Kn.surf()
L1a.Rc = 5.513435044607768E+001
L1a.Thickness = 6.0
L1a.Glass = "BK7"
L1a.Diameter = 30.0

L1b = Kn.surf()
L1b.Rc = -4.408716526030626E+001
L1b.Thickness = 3.0
L1b.Glass = "F2"
L1b.Diameter = 30

L1c = Kn.surf()
L1c.Rc = -2.246906271406796E+002
L1c.Thickness = 9.737871661422000E+001
```

```
L1c.Glass = "AIR"
L1c.Diameter = 30

P_Ima = Kn.surf()
P_Ima.Name = "Plano imagen"
P_Ima.Rc = 0.0
P_Ima.Thickness = 0.0
P_Ima.Glass = "AIR"
P_Ima.Diameter = 300.0

A = [P_Obj, Dif_Obj_c1, Dif_Obj_c2, L1a, L1b, L1c, P_Ima]
configuration_1 = Kn.Kraken_setup()

Doublet = Kn.system(A, configuration_1)
Rays = Kn.raykeeper(Doblete)

tam = 5
rad = 10.0
tsis = len(A) - 1
for i in range(-tam, tam + 1):
    for j in range(-tam, tam + 1):
        x_0 = (i / tam) * rad
        y_0 = (j / tam) * rad
        r = np.sqrt((x_0 * x_0) + (y_0 * y_0))
        if r < rad:
            tet = 0.0
            pSource_0 = [x_0, y_0, 0.0]
            dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]
            W = 0.4
            Doublet.Trace(pSource_0, dCos, W)
            Rays.push()
            W = 0.5
            Doublet.Trace(pSource_0, dCos, W)
            Rays.push()
            W = 0.6
            Doublet.Trace(pSource_0, dCos, W)
            Rays.push()


Kn.display2d(Doblete, Rays, 0)
```

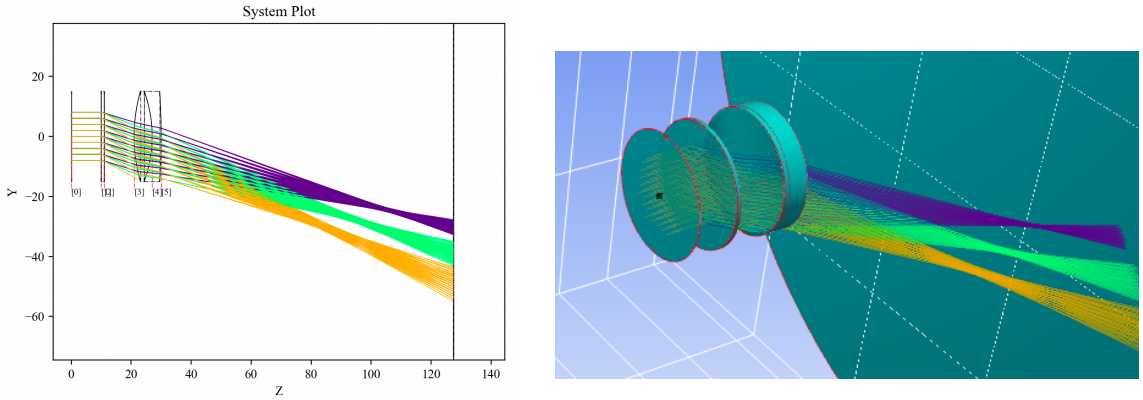


Figure 24. V3D and 2D visualization of a system that has a transmission difaction grating.

S. EXAMPLE - DIFFRACTION GRATING REFLECTION

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
Examp Diffraction Grating Reflection"""
import numpy as np
import Kraken as Kn

P_Obj = Kn.surf()
```

```
P_Obj.Rc = 0.0
P_Obj.Thickness = 10
P_Obj.Glass = "AIR"
P_Obj.Diameter = 30.0

L1a = Kn.surf()
L1a.Rc = 5.513435044607768E+001
L1a.Thickness = 6.0
L1a.Glass = "BK7"
L1a.Diameter = 30.0

L1b = Kn.surf()
L1b.Rc = -4.408716526030626E+001
L1b.Thickness = 3.0
L1b.Glass = "F2"
L1b.Diameter = 30

L1c = Kn.surf()
L1c.Rc = -2.246906271406796E+002
L1c.Thickness = 9.737871661422000E+001 - 50.0
L1c.Glass = "AIR"
L1c.Diameter = 30

Dif_Obj = Kn.surf()
Dif_Obj.Rc = 0.0
Dif_Obj.Thickness = -50
Dif_Obj.Glass = "MIRROR"
Dif_Obj.Diameter = 30.0
Dif_Obj.Grating_D = 1.0
Dif_Obj.Diff_Ord = 1
Dif_Obj.Grating_Angle = 45.0
Dif_Obj.Surface_type = 1

P_Ima = Kn.surf()
P_Ima.Rc = 0.0
P_Ima.Name = "Plano imagen"
P_Ima.Thickness = 0.0
P_Ima.Glass = "AIR"
P_Ima.Diameter = 300.0
P_Ima.Drawing = 0

A = [P_Obj, L1a, L1b, L1c, Dif_Obj, P_Ima]
configuration_1 = Kn.Kraken_setup()

Doublet = Kn.system(A, configuration_1)
Rays = Kn.raykeeper(Doblete)

tam = 5
rad = 10.0
tsis = len(A) - 1
for i in range(-tam, tam + 1):
    for j in range(-tam, tam + 1):
        x_0 = (i / tam) * rad
        y_0 = (j / tam) * rad
        r = np.sqrt((x_0 * x_0) + (y_0 * y_0))
        if r < rad:
            tet = 0.0
            pSource_0 = [x_0, y_0, 0.0]
            dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]
            W = 0.4
            Doublet.Trace(pSource_0, dCos, W)
            Rays.push()
            W = 0.5
            Doublet.Trace(pSource_0, dCos, W)
            Rays.push()
            W = 0.6
            Doublet.Trace(pSource_0, dCos, W)
            Rays.push()

Kn.display2d(Doblete, Rays, 1)
```

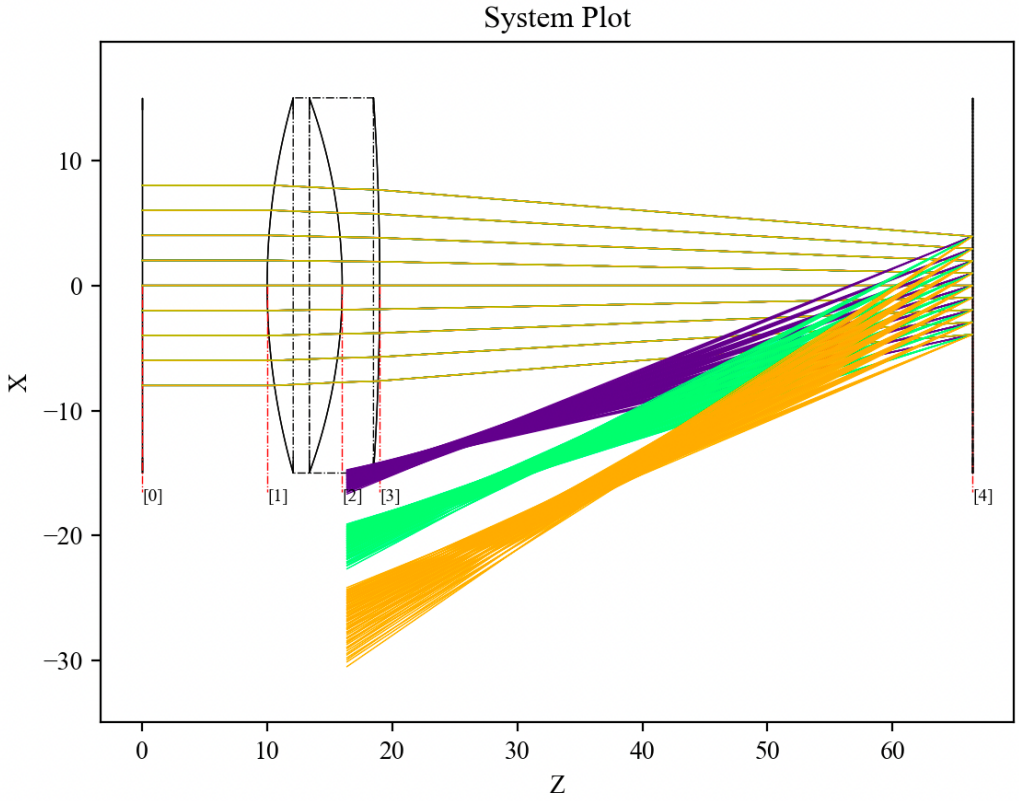


Figure 25. 2D view of a system with a reflection diffraction grating.

T. EXAMPLE - TEL 2M SPYDER SPOT DIAGRAM

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
Examp Tel 2M Spyder Spot Diagram"""
import os
import numpy as np
import Kraken as Kn

P_Obj = Kn.surf()
P_Obj.Thickness = 2.000000000000000E+003
P_Obj.Glass = "AIR"
P_Obj.Diameter = 6.796727741707513E+002 * 2.0
P_Obj.Drawing = 0

M1 = Kn.surf()
M1.Rc = -6.06044E+003
M1.Thickness = -1.774190000000000E+003 + 1.853722901194000E+000
M1.k = -1.637E+000
M1.Glass = "MIRROR"
M1.Diameter = 6.63448E+002 * 2.0
M1.InDiameter = 228.6 * 2.0
M1.DespY = 0.0
M1.TiltX = 0.0000
M1.AxisMove = 1

M2 = Kn.surf()
M2.Rc = -6.06044E+003
M2.Thickness = -M1.Thickness
M2.k = -3.5782E+001
```

```
M2.Glass = "MIRROR"
M2.Diameter = 2.995730651164167E+002 * 2.0
ED0 = np.zeros(20)
ED0[2] = 4.458178314555000E-018
M2.AspherData = ED0

Vertex = Kn.surf()
Vertex.Thickness = 130.0
Vertex.Glass = "AIR"
Vertex.Diameter = 600.0
Vertex.Drawing = 0

currentDirectory = os.getcwd()
direc = r"Prisma.stl"

objeto = Kn.surf()
objeto.Diameter = 118.0 * 2.0
objeto.Solid_3d_stl = direc
objeto.Thickness = 600
objeto.Glass = "BK7"
objeto.TiltX = 55
objeto.TiltY = 0
objeto.TiltZ = 45
objeto.DespX = 0
objeto.DespY = 0
objeto.AxisMove = 0

P_Ima = Kn.surf()
P_Ima.Rc = 0
P_Ima.Thickness = 100.0
P_Ima.Glass = "BK7"
P_Ima.Diameter = 500.0
P_Ima.Drawing = 1

A = [P_Obj, M1, M2, Vertex, objeto, P_Ima]
configuration_1 = Kn.Kraken_setup()

Telescope = Kn.system(A, configuration_1)
Rays = Kn.raykeeper(Telescope)

W = 0.633
tam = 5
rad = 6.56727741707513E+002
tsis = len(A) + 2
for gg in range(0, 10):
    for j in range(-tam, tam + 1):
        # j=0
        for i in range(-tam, tam + 1):
            x_0 = (i / tam) * rad
            y_0 = (j / tam) * rad
            r = np.sqrt((x_0 * x_0) + (y_0 * y_0))
            if r < rad:
                tet = 0.0
                pSource_0 = [x_0, y_0, 0.0]
                # print("-..............")
                dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]
                W = 0.633
                Telescope.NsTrace(pSource_0, dCos, W)
                Rays.push()

Kn.display3d(Telescope, Rays, 0)
print(Telescope.EFFL)
```

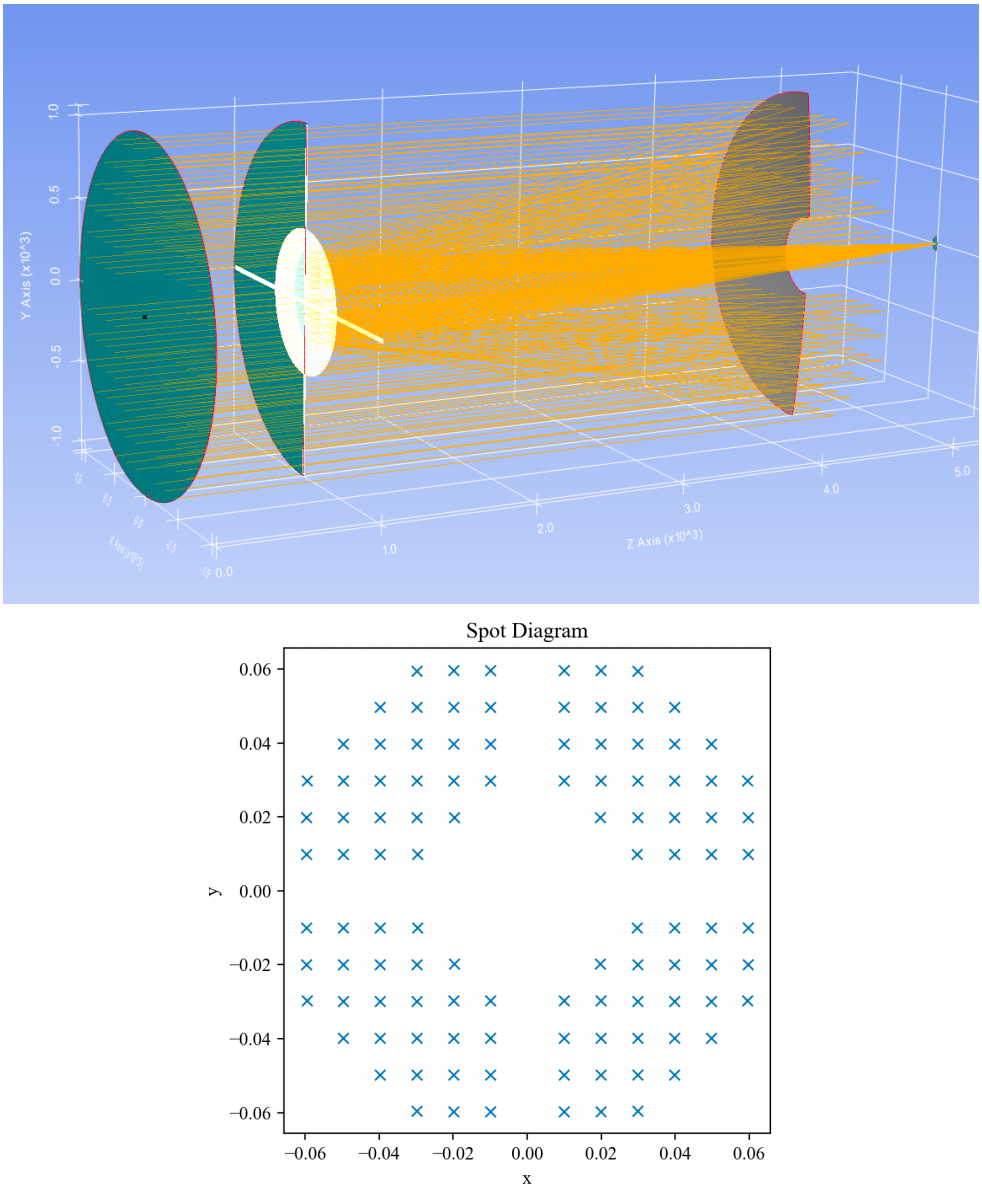


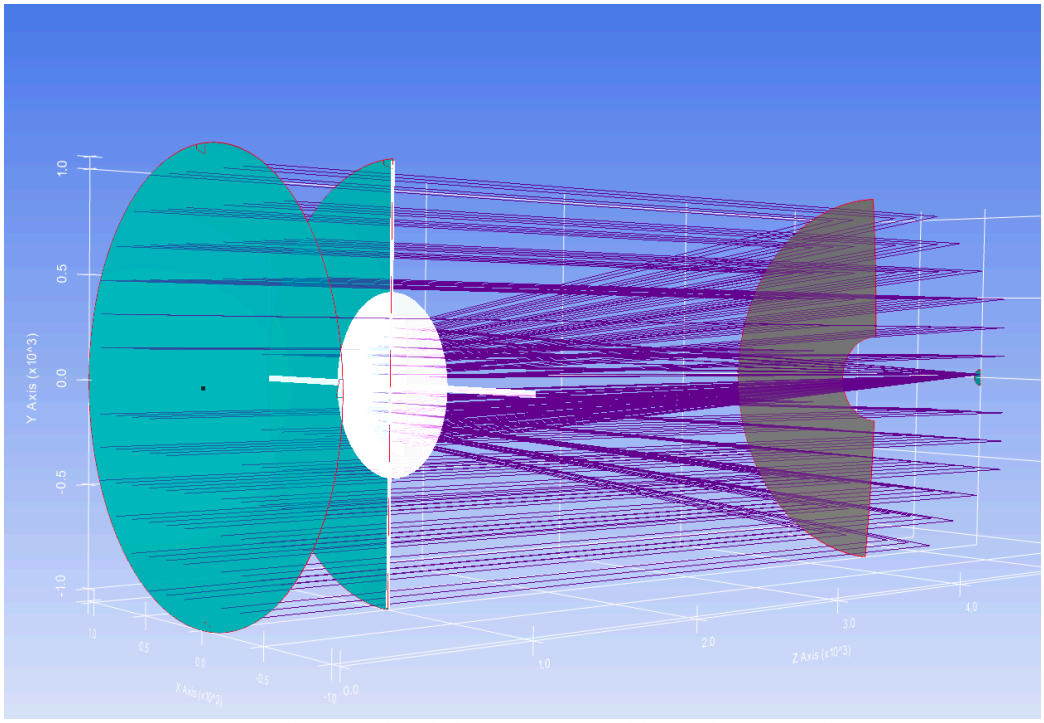
Figure 26. (Top) View of the telescope with the shadow of the spider, (Bottom) diagram of spots with a rectangular pattern, you can notice the lack of the rays that were obstructed by the spider.

U. EXAMPLE - TEL 2M SPYDER SPOT TILT M2

```
#             Rays.push()
#Kn.display3d(Telescope, Rays, 0)
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
Examp Tel 2M Spyder Spot Tilt M2"""
import matplotlib.pyplot as plt
import numpy as np
import pyvista as pv
import Kraken as Kn

P_Obj = Kn.surf()
P_Obj.Rc = 0
P_Obj.Thickness = 1000
P_Obj.Glass = "AIR"
P_Obj.Diameter = 1.059E+003 * 2.0
```

```
    Spider = Kn.surf()
    Spider.Rc = 999999999999.0
    Spider.Thickness = 3.452229924716749E+003 + 100.0
    Spider.Glass = "AIR"
    Spider.Diameter = 1.059E+003 * 2.0

    plane1 = pv.Plane(center=[0, 0, 0], direction=[0, 0, 1], i_size=30, j_size=2100, i_resolution=10,
j_resolution=10)
    plane2 = pv.Plane(center=[0, 0, 0], direction=[0, 0, 1], i_size=2100, j_size=30, i_resolution=10,
j_resolution=10)
    Baffle1 = pv.Disc(center=[0.0, 0.0, 0.0], inner=0, outer=875 / 2.0, normal=[0, 0, 1], r_res=1,
c_res=100)
    Baffle2 = Baffle1.boolean_add(plane1)
    Baffle3 = Baffle2.boolean_add(plane2)

    AAA = pv.MultiBlock()
    AAA.append(plane1)
    AAA.append(plane2)
    AAA.append(Baffle1)

    Spider.Mask_Shape = AAA
    Spider.Mask_Type = 2
    Spider.TiltZ = 0

    Thickness = 3.452200000000000E+003
    M1 = Kn.surf()
    M1.Rc = -9.638000000004009E+003
    M1.Thickness = -Thickness
    M1.k = -1.077310000000000E+000
    M1.Glass = "MIRROR"
    M1.Diameter = 1.059E+003 * 2.0
    M1.InDiameter = 250 * 2.0

    M2 = Kn.surf()
    M2.Rc = -3.93E+003
    M2.Thickness = Thickness + 1.037353522418897E+003
    M2.k = -4.328100000000000E+000
    M2.Glass = "MIRROR"
    M2.Diameter = 3.365E+002 * 2.0
    M2.TiltX = -9.657878504276254E-002
    M2.DespY = -2.000000000000000E+000
    M2.AxisMove = 0

    P_Ima = Kn.surf()
    P_Ima.Diameter = 100.0
    P_Ima.Glass = "AIR"
    P_Ima.Name = "Plano imagen"

    A = [P_Obj, Spider, M1, M2, P_Ima]
    configuration_1 = Kn.Kraken_setup()

    Telescopio = Kn.system(A, configuration_1)
    Rays = Kn.raykeeper(Telescopio)

    tam = 7
    rad = 2200 / 2
    tsis = len(A) - 1
    for i in range(-tam, tam + 1):
        for j in range(-tam, tam + 1):
            x_0 = (i / tam) * rad
            y_0 = (j / tam) * rad
            r = np.sqrt((x_0 * x_0) + (y_0 * y_0))
            if r < rad:
                tet = 0.0
                pSource_0 = [x_0, y_0, 0.0]
                dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]
                W = 0.4
                Telescopio.Trace(pSource_0, dCos, W)
                Rays.push()

    Kn.display3d(Telescopio, Rays, 2)
    X, Y, Z, L, M, N = Rays.pick(-1)

    plt.plot(X, Y, 'x')
```

```
plt.xlabel('x')
plt.ylabel('y')
plt.title('Spot Diagram')
plt.axis('square')
plt.show()
```



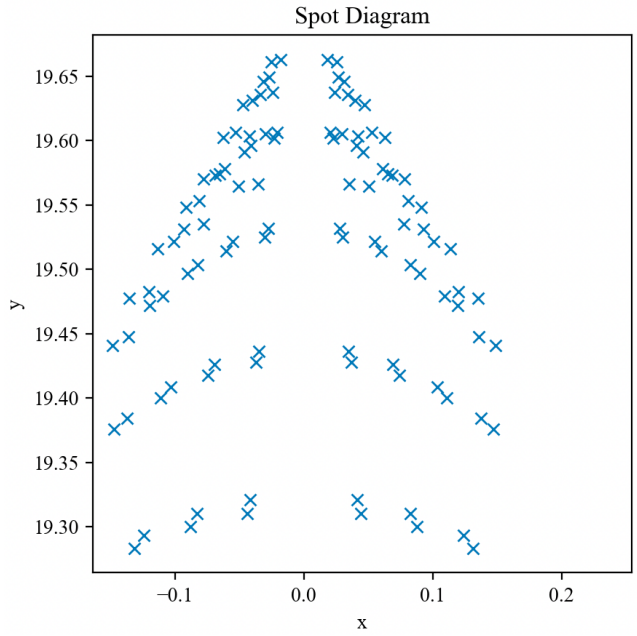


Figure 27. of the telescope with a spider and a spot diagram showing coma aberration from tilting the secondary mirror.

V. EXAMPLE - TEL 2M PUPILA

```
# !/usr/bin/env python3
# -*- coding: utf-8 -*-
Examp  TEl 2M Pupila"""
import matplotlib.pyplot as plt
import numpy as np
import Kraken as Kn

P_Obj = Kn.surf()
P_Obj.Rc = 0
P_Obj.Thickness = 1000 + 3.452200000000000E+003
P_Obj.Glass = "AIR"
P_Obj.Diameter = 1.059E+003 * 2.0

Thickness = 3.452200000000000E+003
M1 = Kn.surf()
M1.Rc = -9.638000000004009E+003
M1.Thickness = -Thickness
M1.k = -1.077310000000000E+000
M1.Glass = "MIRROR"
M1.Diameter = 1.059E+003 * 2.0
M1.InDiameter = 250 * 2.0

M2 = Kn.surf()
M2.Rc = -3.93E+003
M2.Thickness = Thickness + 1.037525880125084E+003
M2.k = -4.328100000000000E+000
M2.Glass = "MIRROR"
M2.Diameter = 3.365E+002 * 2.0
M2.TiltY = 0.1
M2.TiltX = 0.1
M2.AxisMove = 0

P_Ima = Kn.surf()
P_Ima.Diameter = 300.0
P_Ima.Glass = "AIR"
P_Ima.Name = "Plano imagen"

A = [P_Obj, M1, M2, P_Ima]
configuration_1 = Kn.Kraken_setup()
Telescopio = Kn.system(A, configuration_1)

W = 0.4
Surf= 1
AperVal = 2010
AperType = "EPD" # "STOP"
Pup = Kn.PupilCalc(Telescopio, sup, W, AperType, AperVal)

print("Radio pupila de entrada: ")
print(Pup.RadPupInp)
print("Posicion pupila de entrada: ")
print(Pup.PosPupInp)
print("Radio pupila de salida: ")
print(Pup.RadPupOut)
print("Posicion pupila de salida: ")
print(Pup.PosPupOut)
print("Posicion pupila de salida respecto al plano focal: ")
print(Pup.PosPupOutFoc)
print("Orientación pupila de salida")
print(Pup.DirPupSal)
[L, M, N] = Pup.DirPupSal
print(L, M, N)
TetX = np.rad2deg(np.arcsin(-M))
TetY = np.rad2deg(np.arcsin(L / np.cos(np.arcsin(-M))))
print(TetX, TetY)
print("-----------------------------------------------------------------")

Pup.Samp = 10
Pup.Ptype = "hexapolar"
Pup.FieldY = 0.0
Pup.FieldType = "angle"
x, y, z, L, M, N = Pup.Pattern2Field()
Rays = Kn.raykeeper(Telescopio)
```

```
for i in range(0, len(x)):
    pSource_0 = [x[i], y[i], z[i]]
    dCos = [L[i], M[i], N[i]]
    W = 0.4
    Telescopio.Trace(pSource_0, dCos, W)
    Rays.push()

Kn.display3d(Telescopio, Rays, 2)

X, Y, Z, L, M, N = Rays.pick(-1)
plt.figure(300)
plt.plot(X, Y, 'x')
plt.axis('square')
plt.show(block=False)
```

```
Radio pupila de entrada:
1005.0
Posicion pupila de entrada:
[   0.     0.  4452.2]
Radio pupila de salida:
384.13536608352524
Posicion pupila de salida:
[  -4.37108607    4.37107941 -252.21310955]
Posicion pupila de salida respecto al plano focal:
[-4.37108607e+00  4.37107941e+00 -5.74193899e+03]
Orientación pupila de salida
[ 0.00349065 -0.00349064  0.99998782]
```

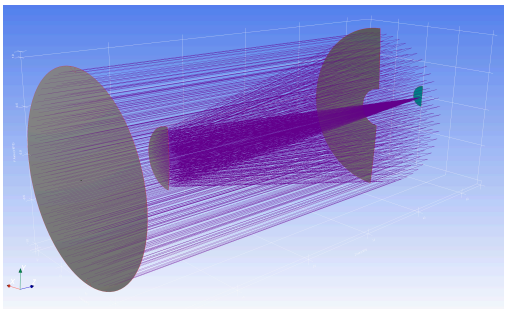


Figure 28. (Left) Obtaining the pupil data. (Right) 3D visualization of the telescope.

W. EXAMPLE - TEL 2M ERROR MAP

```
# -*- coding: utf-8 -*-
Examp Tel 2M Error Map"""
import matplotlib.pyplot as plt
import numpy as np
import Kraken as Kn
import time

def ErrorGen():
    L = 1000.
    N = 20.
    hight = 0.001
    SPACE = 2 * L / N
    x = np.arange(-L, L + SPACE, SPACE)
    y = np.arange(-L, L + SPACE, SPACE)
    gx, gy = np.meshgrid(x, y)
    R = np.sqrt((gx * gx) + (gy * gy))
    arg = np.argwhere(R < L)
    Npoints = np.shape(arg)[0]
    X = np.zeros(Npoints)
    Y = np.zeros(Npoints)
    i = 0
    for [a, b] in arg:
        X[i] = gx[a, b]
        Y[i] = gy[a, b]
        i = i + 1
    spa = 10000000
    Z = hight * (np.random.randint(-spa, spa, Npoints)) / (spa * 2.0)
    return [X, Y, Z, SPACE]

P_Obj = Kn.surf()
P_Obj.Rc = 0
P_Obj.Thickness = 3500
```

```
P_Obj.Glass = "AIR"
P_Obj.Diameter = 1.059E+003 * 2.0

Thickness = 3.452200000000000E+003
M1 = Kn.surf()
M1.Rc = -9.638000000004009E+003
M1.Thickness = -Thickness
M1.k = -1.077310000000000E+000
M1.Glass = "MIRROR"
M1.Diameter = 1.059E+003 * 2.0
M1.InDiameter = 250 * 2.0
M1.Error_map = ErrorGen()

M2 = Kn.surf()
M2.Rc = -3.93E+003
M2.Thickness = Thickness + 1.037525880125084E+003
M2.k = -4.328100000000000E+000
M2.Glass = "MIRROR"
M2.Diameter = 3.365E+002 * 2.0
M2.AxisMove = 0

P_Ima = Kn.surf()
P_Ima.Diameter = 1000.0
P_Ima.Glass = "AIR"
P_Ima.Name = "Plano imagen"

A = [P_Obj, M1, M2, P_Ima]
configuration_1 = Kn.Kraken_setup()

Telescopio = Kn.system(A, configuration_1)
Rays = Kn.raykeeper(Telescopio)

tam = 9
rad = 2100 / 2
tsis = len(A) - 1

start_time = time.time()


for i in range(-tam, tam + 1):
    for j in range(-tam, tam + 1):
        x_0 = (i / tam) * rad
        y_0 = (j / tam) * rad
        r = np.sqrt((x_0 * x_0) + (y_0 * y_0))
        if r < rad:
            print(i)
            tet = 0.0
            pSource_0 = [x_0, y_0, 0.0]
            dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]
            W = 0.4
            Telescopio.Trace(pSource_0, dCos, W)
            Rays.push()

print("--- %s seconds ---" % (time.time() - start_time))
Kn.display3d(Telescopio, Rays, 2)
print(Telescopio.EFFL)
X, Y, Z, L, M, N = Rays.pick(-1)

plt.plot(X, Y, 'x')
plt.xlabel('numbers')
plt.ylabel('values')
plt.title('Spot Diagram')
plt.axis('square')
plt.show()
```

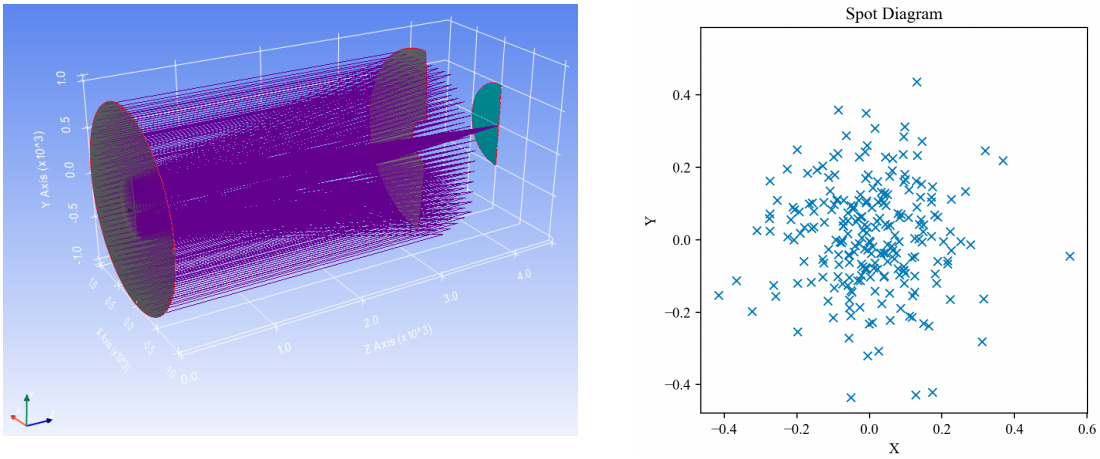


Figure 29. (Left) 3D visualization of a telescope generated with a deformation map added to the shape function of the primary mirror. (Right) Spot diagram originated with this system.

X. EXAMPLE - TEL 2M WAVEFRONT FITTING

```
# !/usr/bin/env python3
# -*- coding: utf-8 -*-
Examp Tel 2M Wavefront Fitting"""
import os
import sys
import matplotlib.pyplot as plt
import numpy as np
import Kraken as Kn
from PhaseCalc import Phase

currentDirectory = os.getcwd()
sys.path.insert(1, currentDirectory + '/library')

P_Obj = Kn.surf()
P_Obj.Rc = 0
P_Obj.Thickness = 1000 + 3.4522000000000000E+003
P_Obj.Glass = "AIR"
P_Obj.Diameter = 1.059E+003 * 2.0

Thickness = 3.4522000000000000E+003
M1 = Kn.surf()
M1.Rc = -9.638000000004009E+003
M1.Thickness = -Thickness
M1.k = -1.077310000000000E+000
M1.Glass = "MIRROR"
M1.Diameter = 1.059E+003 * 2.0
M1.InDiameter = 250 * 2.0
M1.TiltY = 0.0
M1.TiltX = 0.0

M1.AxisMove = 0
M2 = Kn.surf()
M2.Rc = -3.93E+003
M2.Thickness = Thickness + 1037.525880
M2.k = -4.328100000000000E+000
M2.Glass = "MIRROR"
M2.Diameter = 3.365E+002 * 2.0
M2.TiltY = 0.0
M2.TiltX = 0.0
M2.DespY = 0.0
M2.DespX = 0.0
M2.AxisMove = 0

P_Ima = Kn.surf()
```

```
    P_Ima.Diameter = 300.0
    P_Ima.Glass = "AIR"
    P_Ima.Name = "Plano imagen"

    A = [P_Obj, M1, M2, P_Ima]
    configuration_1 = Kn.Kraken_setup()
    Telescopio = Kn.system(A, configuration_1)

    W = 0.4
    Surf= 1
    Samp = 10
    Ptype = "hexapolar"
    FieldY = 0.1
    FieldX = 0.0
    FieldType = "angle"

    AperType = "STOP"
    fieldType = "angle"
    AperVal = 2100.

    Z, X, Y, P2V = Phase(Telescopio, sup, W, AperType, AperVal, configuration_1, Samp, Ptype, FieldY,
FieldX, FieldType)
    NC = 38
    A = np.ones(NC)

    z_coeff, MatNotation, w_rms, fitt_error = Kn.Zernike_Fitting(X, Y, Z, A)
    A = np.abs(z_coeff)
    Zeros = np.argwhere(A > 0.0001)
    AA = np.zeros_like(A)
    AA[Zeros] = 1
    A = AA
    z_coeff, MatNotation, w_rms, fitt_error = Kn.Zernike_Fitting(X, Y, Z, A)
    print("Peak to valley: ", P2V)

    for i in range(0, NC):
        print("z ", i + 1, "  ", "{0:.6f}".format(float(z_coeff[i])), "  :  ", MatNotation[i])

    print("RMS: ", "{:.4f}".format(float(w_rms)), " Error del ajuste: ", fitt_error)
    z_coeff[0] = 0
    print("RMS to chief: ", np.sqrt(np.sum(z_coeff * z_coeff)))
    z_coeff[1] = 0
    z_coeff[2] = 0
    print("RMS to centroid: ", np.sqrt(np.sum(z_coeff * z_coeff)))

    RR = Kn.raykeeper(Telescopio)
    Pup = Kn.PupilCalc(Telescopio, sup, W, AperType, AperVal)
    Pup.FieldX = FieldX
    Pup.FieldY = FieldY
    x, y, z, L, M, N = Pup.Pattern2Field()

    for i in range(0, len(x)):
        pSource_0 = [x[i], y[i], z[i]]
        dCos = [L[i], M[i], N[i]]
        Telescopio.Trace(pSource_0, dCos, W)
        RR.push()

    Kn.display3d(Telescopio, RR, 2)
    X, Y, Z, L, M, N = RR.pick(-1)

    plt.plot(X, Y, 'x')
    plt.xlabel('numbers')
    plt.ylabel('values')
    plt.title('spot Diagram')
    plt.axis('square')
    plt.show()
```

```
Peak to valley:  -4.6972543604773245
z 1    0.562333 : 1^(1/2)(1.0)   Piston
z 2    -0.663146 : 4^(1/2)(1.0r^1)cos(T)   Tilt x, (about y axis)
z 3    0.022264 : 4^(1/2)(1.0r^1)sin(T)   Tilt y, (about x axis)
z 4    0.327227 : 3^(1/2)(-1.0+2.0r^2)   Power or Focus
z 5    0.016728 : 6^(1/2)(1.0r^2)sin(2T)   Astigmatism y, (45deg)
z 6    -0.087753 : 6^(1/2)(1.0r^2)cos(2T)   Astigmatism x, (0deg)
z 7    0.018916 : 8^(1/2)(-2.0r^1+3.0r^3)sin(T)   Coma y
z 8    -0.232776 : 8^(1/2)(-2.0r^1+3.0r^3)cos(T)   Coma x
z 9    0.000000 : 8^(1/2)(1.0r^3)sin(3T)   Trefoil y
z 10    0.000000 : 8^(1/2)(1.0r^3)cos(3T)   Trefoil x
z 11    0.000729 : 5^(1/2)(1.0+-6.0r^2+6.0r^4)   Primary Spherical
z 12    0.000155 : 10^(1/2)(-3.0r^2+4.0r^4)cos(2T)   Secondary Astigmatism x
z 13    0.000000 : 10^(1/2)(-3.0r^2+4.0r^4)sin(2T)   Secondary Astigmatism y
z 14    0.000000 : 10^(1/2)(1.0r^4)cos(4T)   Tetrafoil x
z 15    0.000000 : 10^(1/2)(1.0r^4)sin(4T)   Tetrafoil y
z 16    0.001140 : 12^(1/2)(3.0r^1+-12.0r^3+10.0r^5)cos(T)   Secondary Coma x
z 17    -0.000999 : 12^(1/2)(3.0r^1+-12.0r^3+10.0r^5)sin(T)   Secondary Coma y
z 18    0.000000 : 12^(1/2)(-4.0r^3+5.0r^5)cos(3T)   Secondary Trefoil x
```

Figure 30. Value of the coefficients of the Zernike polynomials fitted to the wavefront of the system for a given field. The result also includes the mathematical expression of the polynomial.

Y. EJEMPLO – TEL 2M-STL_IMAGESLICER.PY

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Example - -2M-STL_ImageSlicer.py
"""
import Kraken as Kn
import numpy as np
import matplotlib.pyplot as plt
import scipy
import os
A1=1
if A1==0:

    P_Obj=Kn.surf()
    P_Obj.Rc=0
    P_Obj.Thickness=1000+3.452200000000000E+003
    P_Obj.Glass="AIR"
    P_Obj.Diameter=1.059E+003*2.0

    Thickness=3.452200000000000E+003
    M1=Kn.surf()
    M1.Rc=-9.638000000004009E+003
    M1.Thickness=-Thickness
    M1.k=-1.077310000000000E+000
    M1.Glass="MIRROR"
    M1.Diameter=1.059E+003*2.0
    M1.InDiameter=250*2.0

    M2=Kn.surf()
    M2.Rc=-3.93E+003
    M2.Thickness=Thickness+1.037525880125084E+003
    M2.k=-4.328100000000000E+000
    M2.Glass="MIRROR"
    M2.Diameter=3.365E+002*2.0
    M2.AxisMove=0

    P_Image_A=Kn.surf()
    P_Image_A.Diameter=10.0
    P_Image_A.Glass="AIR"
    P_Image_A.Thickness=10
    P_Image_A.Name="Image plane Tel"
    P_Image_A.DespZ=-100.0

    A=[P_Obj,M1,M2,P_Image_A]



    configuration_1=Kn.Kraken_setup()
```

```
Telescopio=Kn.system(A,configuration_1)
Rays=Kn.raykeeper(Telescopio)


# Gaussian
def f(x):
    x=np.rad2deg(x)
    seing=1.2/3600.0
    sigma=seing/2.3548
    mean = 0
    standard_deviation = sigma
    y=scipy.stats.norm(mean, standard_deviation)
    res=y.pdf(x)
    return res


Sun = Kn.SourceRnd()
Sun.field=4*1.2/(2.0*3600.0)
Sun.fun = f
Sun.dim = 2100
Sun.num = 100000
L, M, N, X, Y, Z = Sun.rays()


Xr=np.zeros_like(L)
Yr=np.zeros_like(L)
Zr=np.zeros_like(L)

Lr=np.zeros_like(L)
Mr=np.zeros_like(L)
Nr=np.zeros_like(L)

NM=np.zeros_like(L)

con=0
con2=0
W = 0.6

for i in range(0,Sun.num):
    if con2==10:
        print(100.*i/Sun.num)
        con2=0

    pSource_0 = [X[i], Y[i], Z[i]]
    dCos = [L[i], M[i], N[i]]
    Telescopio.Trace(pSource_0, dCos, W)

    x,y,z=Telescopio.XYZ[-1]
    l,m,n=Telescopio.LMN[-1]
    Xr[con]=x
    Yr[con]=y
    Zr[con]=z
    Lr[con]=l
    Mr[con]=m
    Nr[con]=n


    if Telescopio.NAME[-1]=="Image plane Tel":
        NM[con]=i
    else:
        NM[con]=-1

    con=con+1
    con2=con2+1
    # Rays.push()


args=np.argwhere(NM!=-1)


X=Xr[args]
Y=Yr[args]
Z=Zr[args]

L=Lr[args]
```

```
    M=Mr[args]
    N=Nr[args]
    W=W*np.ones_like(N)



    Rays=np.hstack((X,Y,Z,L,M,N,W))
    outfile="savedRays.npy"
    np.save(outfile, Rays)



#####################################################################
else:
    P_Obj=Kn.surf()
    P_Obj.Rc=0
    P_Obj.Thickness=100.+0.5
    P_Obj.Glass="AIR"
    P_Obj.Diameter=10



    currentDirectory = os.getcwd()
    direc = r"Jherrera-ImageSlicerBW-00.stl"
    P_ImageSlicer=Kn.surf()
    P_ImageSlicer.Diameter=10.0
    P_ImageSlicer.Glass="BK7"
    P_ImageSlicer.Name="Image slicer"
    P_ImageSlicer.Solid_3d_stl = direc
    P_ImageSlicer.Thickness = 13
    P_ImageSlicer.TiltX=180.0
    P_ImageSlicer.DespX=-0.55
    P_ImageSlicer.DespY=-0.03
    P_ImageSlicer.AxisMove=0

    P_Ima=Kn.surf()
    P_Ima.Diameter=10.0
    P_Ima.Glass="AIR"
    P_Ima.Name="Plano imagen"




    A=[P_Obj, P_ImageSlicer, P_Ima]
    configuration_1 = Kn.Kraken_setup()
    ImageSlicer = Kn.system(A,configuration_1)
    Rays=Kn.raykeeper(ImageSlicer)



    outfile="savedRays.npy"
    R=np.load(outfile)
    print(np.shape(R))
    X,Y,Z,L,M,N,W=R[:,0],R[:,1],R[:,2],R[:,3],R[:,4],R[:,5],R[:,6]



    nrays=2000
    Xr=np.zeros(nrays)
    Yr=np.zeros(nrays)
    Zr=np.zeros(nrays)
    Lr=np.zeros(nrays)
    Mr=np.zeros(nrays)
    Nr=np.zeros(nrays)
    NM=np.zeros(nrays)

    con=0
    con2=0


    for i in range(0,nrays):
        if con2==10:
            print(100.*i/nrays)
            con2=0
```

```
        pSource_0 = [X[i], Y[i], Z[i]*0]
        dCos = [L[i], M[i], N[i]]
        ImageSlicer.NsTrace(pSource_0, dCos, W[i])

        x,y,z=ImageSlicer.XYZ[-1]
        l,m,n=ImageSlicer.LMN[-1]
        Xr[con]=x
        Yr[con]=y
        Zr[con]=z
        Lr[con]=l
        Mr[con]=m
        Nr[con]=n

        AA=ImageSlicer.SURFACE
        AA=np.asarray(AA)
        AW=np.argwhere(AA==1)
        if ImageSlicer.NAME[-1]=="Plano imagen" and len(AW)<10 and ImageSlicer.TT<0.9:


            # and ImageSlicer.TT<0.9 and ImageSlicer.TT>0.4


            NM[con]=i
            Rays.push()
        else:
            NM[con]=-1

        con=con+1
        con2=con2+1



    args=np.argwhere(NM!=-1)

    X=Xr[args]
    Y=Yr[args]
    Z=Zr[args]

    L=Lr[args]
    M=Mr[args]
    N=Nr[args]
    W=W*np.ones_like(N)


###################
    plt.plot(X, Y, '.', c="r", markersize=1)

    # axis labeling
    plt.xlabel('x')
    plt.ylabel('y')

    # figure name
    plt.title('Dot Plot')
    plt.axis('square')
    plt.show()


    #             Rays.push()
    Kn.display3d(ImageSlicer, Rays, 0)
```

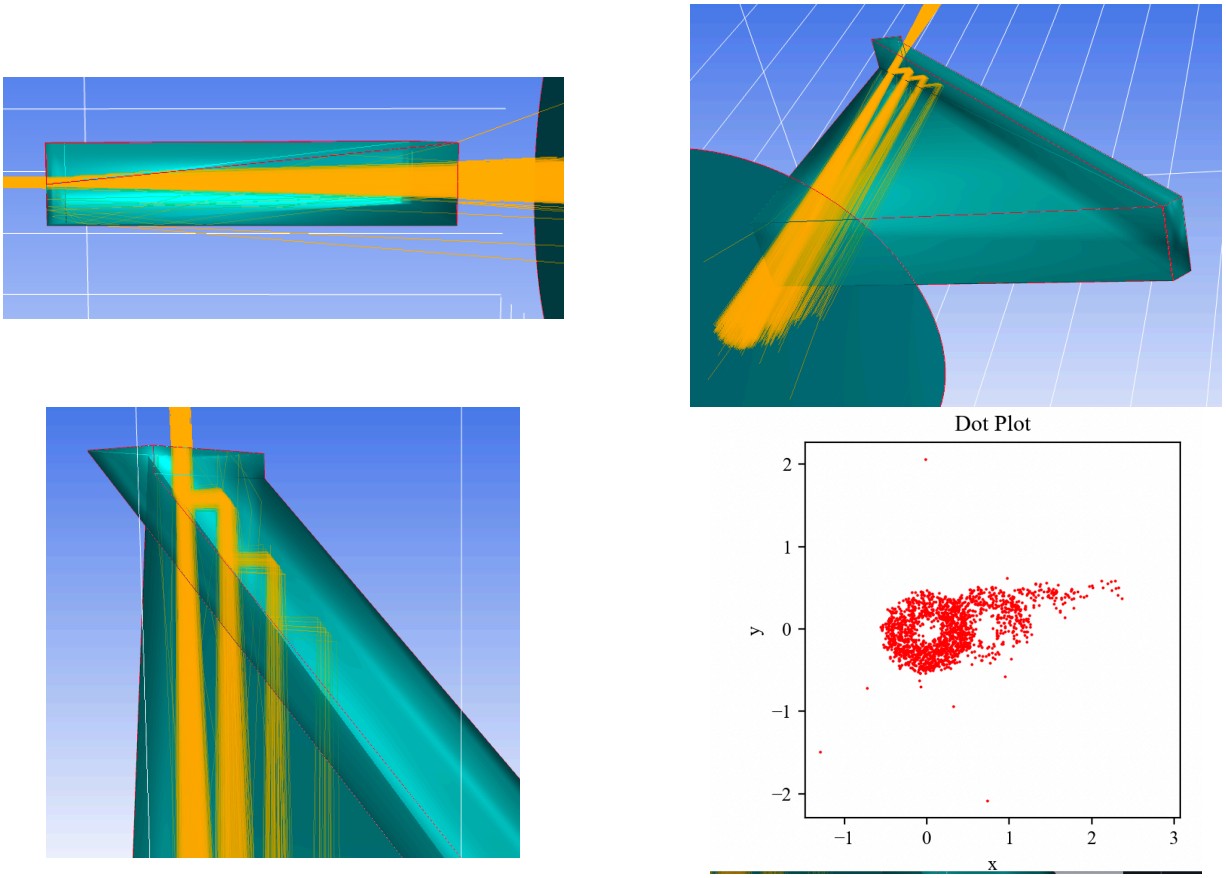


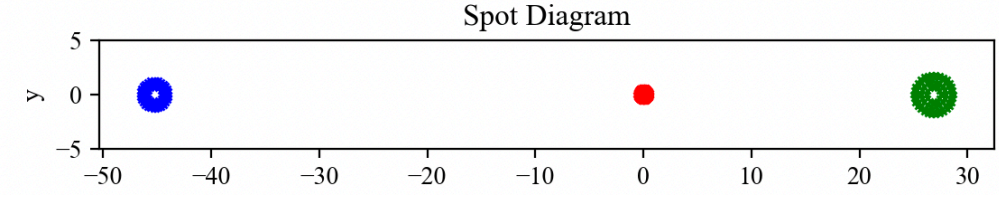
Figure 31. Example of Image Slicer Bowel Walraven from an STL model generated in OpenScad.

Z. EJEMPLO – TEL 2M_ATMOSPHERIC_REFRACTION_CORRECTOR.PY

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Aug  2 12:04:14 2020
Example - -Tel_2M_Atmospheric_Refraction_Corrector.py

@author: joelherreravazquez
"""
import Kraken as Kn
import numpy as np
import matplotlib.pyplot as plt
import time


P_Obj=Kn.surf()
P_Obj.Rc=0
P_Obj.Thickness=1000+3.452200000000000E+003
P_Obj.Glass="AIR"
P_Obj.Diameter=1.059E+003*2.0

Thickness=3.452200000000000E+003
M1=Kn.surf()
M1.Rc=-9.638000000004009E+003
M1.Thickness=-Thickness
M1.k=-1.077310000000000E+000
M1.Glass="MIRROR"
M1.Diameter=1.059E+003*2.0
M1.InDiameter=250*2.0
```

```
M2=Kn.surf()
M2.Rc=-3.93E+003
M2.Thickness=Thickness+1.037525880125084E+003
M2.k=-4.328100000000000E+000
M2.Glass="MIRROR"
M2.Diameter=3.365E+002*2.0
M2.AxisMove=0

P_Ima=Kn.surf()
P_Ima.Diameter=1000.0
P_Ima.Glass="AIR"
P_Ima.Name="Plano imagen"
A=[P_Obj,M1,M2,P_Ima]

configuration_1=Kn.Kraken_setup()
Telescopio=Kn.system(A,configuration_1)
Rays1=Kn.raykeeper(Telescopio)
Rays2=Kn.raykeeper(Telescopio)
Rays3=Kn.raykeeper(Telescopio)

W = 0.4
Surf= 1
AperVal = 2010
AperType = "EPD" # "STOP"
Pup = Kn.PupilCalc(Telescopio, sup, W, AperType, AperVal)
Pup.Samp=11
Pup.FieldType = "angle"
Pup.AtmosRef = 1
Pup.T    = 283.15     # k
Pup.P    = 101300     # Pa
Pup.H    = 0.5        # Humidity ratio 1 to 0
Pup.xc   = 400        # ppm
Pup.lat  = 31         # degrees
Pup.h    = 2800       # meters
Pup.l1   = 0.60169    # micron
Pup.l2   = 0.50169    # micron
Pup.z0   = 55.0       # degrees
Pup.Ptype = "hexapolar"
Pup.FieldX = 0.0
W1 = 0.50169
Pup.l2   = W1
xa,ya,za,La,Ma,Na=Pup.Pattern2Field()
W2 = 0.60169
Pup.l2   = W2
xb,yb,zb,Lb,Mb,Nb=Pup.Pattern2Field()
W3 = 0.70169
Pup.l2   = W3
xc,yc,zc,Lc,Mc,Nc=Pup.Pattern2Field()
###############################################
for i in range(0,len(xa)):
    pSource_0 = [xa[i], ya[i], za[i]]
    dCos=[La[i], Ma[i], Na[i]]
    Telescopio.Trace(pSource_0, dCos, W1)
    Rays1.push()
for i in range(0,len(xb)):
    pSource_0 = [xb[i], yb[i], zb[i]]
    dCos=[Lb[i], Mb[i], Nb[i]]
    Telescopio.Trace(pSource_0, dCos, W2)
    Rays2.push()
for i in range(0,len(xc)):
    pSource_0 = [xc[i], yc[i], zc[i]]
    dCos=[Lc[i], Mc[i], Nc[i]]
    Telescopio.Trace(pSource_0, dCos, W3)
    Rays3.push()


###############################################

# Kn.display3d(Telescopio,Rays,2)
X,Y,Z,L,M,N=Rays1.pick(-1)
plt.plot(X*1000.0,Y*1000.0, 'x', c="b")
X,Y,Z,L,M,N=Rays2.pick(-1)
plt.plot(X*1000.0,Y*1000.0, 'x', c="r")
X,Y,Z,L,M,N=Rays3.pick(-1)
plt.plot(X*1000.0,Y*1000.0, 'x', c="g")
```

```
# axis labeling
plt.xlabel('x')
plt.ylabel('y')
# figure name
plt.title('Spot Diagram')
plt.axis('square')
plt.ylim(-np.pi, np.pi)
plt.show()
```



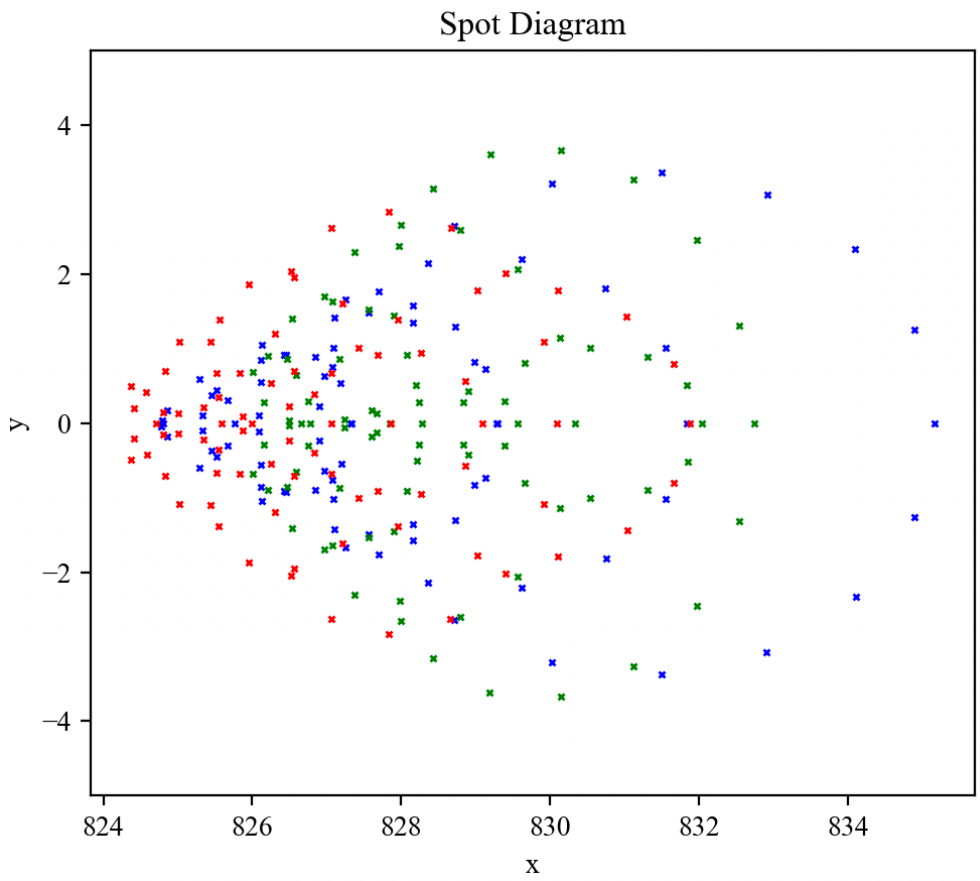


Figure 32. (Above) Example of the effect of the atmosphere in three different wavelengths and the implementation of an atmospheric dispersion corrector (Below).

AA. EXAMPLE - EXTRA SHAPE MICRO LENS ARRAY

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
Examp Extra Shape Micro Lens Array"""
import Kraken as Kn
import numpy as np
import matplotlib.pyplot as plt

P_Obj = Kn.surf()
P_Obj.Rc = 0.0
P_Obj.Thickness = 10
P_Obj.Glass = "AIR"
P_Obj.Diameter = 30.0

L1a = Kn.surf()
L1a.Rc = 55.134*0
L1a.Thickness = 2.0
L1a.Glass = "BK7"
L1a.Diameter = 30.0
```

```
L1c = Kn.surf()
L1c.Thickness = 40
L1c.Glass = "AIR"
L1c.Diameter = 30

def f(x,y,E):
    DeltaX=E[0]*np.rint(x/E[0])
    DeltaY=E[0]*np.rint(y/E[0])
    x=x-DeltaX
    y=y-DeltaY
    s = np.sqrt((x * x) + (y * y))
    c = 1.0 / E[1]
    InRoot = 1 - (E[2] + 1.0) * c * c * s * s
    z = (c * s * s / (1.0 + np.sqrt(InRoot)))
    return z

coef=[3.0, -3, 0]
L1c.ExtraData=[f, coef]
L1c.Res=2

P_Ima = Kn.surf()
P_Ima.Rc = 0.0
P_Ima.Thickness = 0.0
P_Ima.Glass = "AIR"
P_Ima.Diameter = 300.0
P_Ima.Name = "Image plane"

A = [P_Obj, L1a,  L1c, P_Ima]
Config_1 = Kn.Kraken_setup()

Lens = Kn.system(A, Config_1)
Rays = Kn.raykeeper(Lens)

Wav = 0.45
for i in range(-100, 100+1):
    pSource = [0.0, i/10., 0.0]
    dCos = [0.0, 0.0, 1.0]
    Lens.Trace(pSource, dCos, Wav)
    Rays.push()


Kn.display3d(Lens, Rays, 1)
Kn.display2d(Lens, Rays, 0)
```

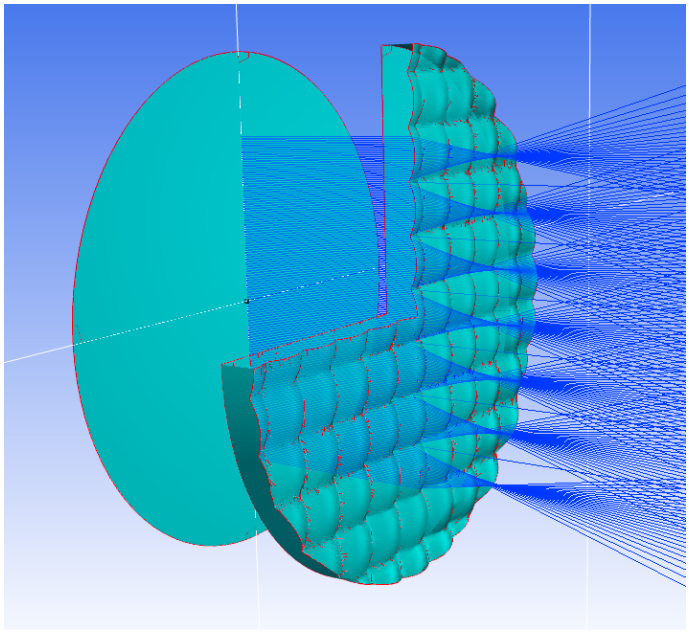


Figure 33. Example of a user-defined micro lens arrangement on an ExtraShape type surface.

BB. EXAMPLE - EXTRA SHAPE RADIAL SINE

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
Examp Extra Shape Radial Sine"""
import Kraken as Kn
import numpy as np
import matplotlib.pyplot as plt

P_Obj = Kn.surf()
P_Obj.Rc = 0.0
P_Obj.Thickness = 10
P_Obj.Glass = "AIR"
P_Obj.Diameter = 30.0
P_Obj.Drawing = 0

L1a = Kn.surf()
L1a.Rc = 55.134
L1a.Thickness = 9.0
L1a.Glass = "BK7"
L1a.Diameter = 30.0

L1c = Kn.surf()
L1c.Rc = -224.69
L1c.Thickness = 40
L1c.Glass = "AIR"
L1c.Diameter = 30

def f(x,y,E):
    r=np.sqrt(x*x+y*y)
    r=np.asarray(r)
    H=2.0*np.pi*r/E[0]
    z=np.sin(H) * E[1]
    return z

coef = np.zeros(36)
coef[0]=5
coef[1]=.5
ES = [f, coef]
L1c.ExtraData=ES

P_Ima = Kn.surf()
P_Ima.Rc = 0.0
P_Ima.Thickness = 0.0
P_Ima.Glass = "AIR"
P_Ima.Diameter = 200.0
P_Ima.Name = "Image plane"

A = [P_Obj, L1a,  L1c, P_Ima]
Config_1 = Kn.Kraken_setup()

Lens = Kn.system(A, Config_1)
Rays = Kn.raykeeper(Lens)

Wav = 0.45
for i in range(-100, 100+1):
    pSource = [0.0, i/10., 0.0]
    dCos = [0.0, 0.0, 1.0]
    Lens.Trace(pSource, dCos, Wav)
    Rays.push()


Kn.display3d(Lens, Rays, 2)
Kn.display2d(Lens, Rays, 0)
```

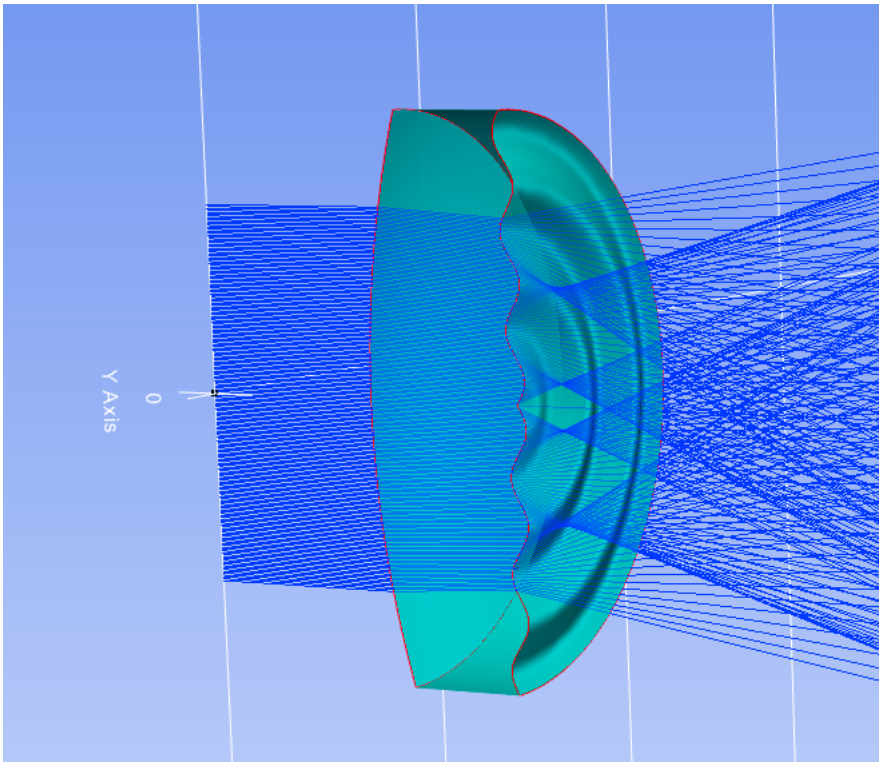


Figure 34. View of a lens with a surface defined by the user by means of a radial function.

CC. EXAMPLE - EXTRA SHAPE XY COSINES

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
Examp Extra Shape XY Cosines"""
import Kraken as Kn
import numpy as np
import matplotlib.pyplot as plt

P_Obj = Kn.surf()
P_Obj.Rc = 0.0
P_Obj.Thickness = 10
P_Obj.Glass = "AIR"
P_Obj.Diameter = 30.0

L1a = Kn.surf()
L1a.Rc = 55.134*0
L1a.Thickness = 9.0
L1a.Glass = "BK7"
L1a.Diameter = 30.0

L1c = Kn.surf()
L1c.Thickness = 40
L1c.Glass = "AIR"
L1c.Diameter = 30

def f(x,y,E):
    r = np.sqrt((x * x) + (y * y * 0))
```

```
    H=2.0*np.pi*r/E[0]
    zx=np.abs(np.cos(H) * E[1])
    r = np.sqrt((x * x * 0) + (y * y))
    H=2.0*np.pi*r/E[0]
    zy=np.abs(np.cos(H) * E[1])
    return zx+zy

coef=[10.0,1.]
L1c.ExtraData=[f, coef]
L1c.Res=1

P_Ima = Kn.surf()
P_Ima.Rc = 0.0
P_Ima.Thickness = 0.0
P_Ima.Glass = "AIR"
P_Ima.Diameter = 300.0
P_Ima.Name = "Image plane"

A = [P_Obj, L1a,  L1c, P_Ima]
Config_1 = Kn.Kraken_setup()

Lens = Kn.system(A, Config_1)
Rays = Kn.raykeeper(Lens)

Wav = 0.45
for i in range(-100, 100+1):
    pSource = [0.0, i/10., 0.0]
    dCos = [0.0, 0.0, 1.0]
    Lens.Trace(pSource, dCos, Wav)
    Rays.push()


Kn.display3d(Lens, Rays, 2)
Kn.display2d(Lens, Rays, 0)
```

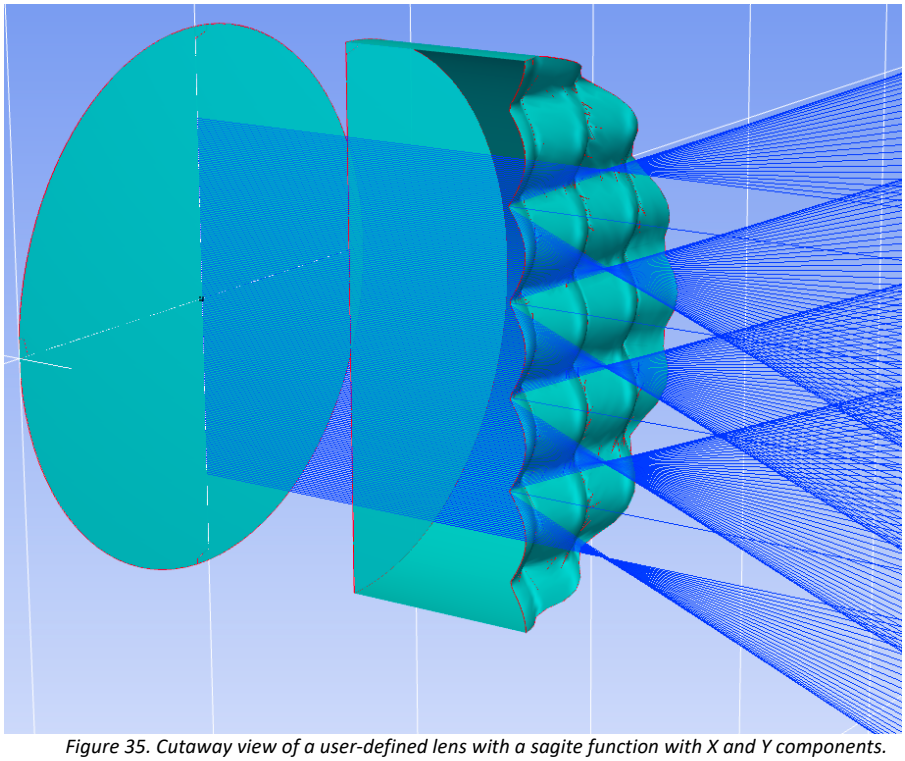


Figure 35. Cutaway view of a user-defined lens with a sagite function with X and Y components.

DD. EXAMPLE - MULTICORE

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
Examp Multicore"""
import multiprocessing
import time
import numpy as np
import Kraken as Kn

start_time = time.time()

P_Obj = Kn.surf()
P_Obj.Rc = 0.0
P_Obj.Thickness = 10
P_Obj.Glass = "AIR"
P_Obj.Diameter = 30.0

L1a = Kn.surf()
L1a.Rc = 9.284706570002484E+001
L1a.Thickness = 6.0
L1a.Glass = "BK7"
L1a.Diameter = 30.0
L1a.Axicon = 0

L1b = Kn.surf()
L1b.Rc = -3.071608670000159E+001
L1b.Thickness = 3.0
L1b.Glass = "F2"
L1b.Diameter = 30

L1c = Kn.surf()
L1c.Rc = -7.819730726078505E+001
L1c.Thickness = 9.737604742910693E+001
L1c.Glass = "AIR"
L1c.Diameter = 30

P_Ima = Kn.surf()
P_Ima.Rc = 0.0
P_Ima.Thickness = 0.0
P_Ima.Glass = "AIR"
P_Ima.Diameter = 3.0
P_Ima.Name = "Plano imagen"

A = [P_Obj, L1a, L1b, L1c, P_Ima]
config_1 = Kn.Kraken_setup()

Doblete1 = Kn.system(A, config_1)

def trax1(xyz, lmn, w, q):
    Rays = Kn.raykeeper(Doblete1)
    start_time = time.time()
    for i in range(0, 700):
        Doblete1.Trace(xyz, lmn, w)
        Rays.push()
    A = Rays.pick(-1)
    Rays.clean()
    q.put(A[0])
    print("--- %s seconds ---" % (time.time() - start_time))

if __name__ == '__main__':
    start_time = time.time()
    pSource_0 = [1, 0, 0.0]
    dCos = [0.0, np.sin(np.deg2rad(0)), np.cos(np.deg2rad(0))]
    w = 0.5
    q = multiprocessing.Queue()
    p1 = multiprocessing.Process(target=trax1, args=(pSource_0, dCos, w + 0.1, q))
    p2 = multiprocessing.Process(target=trax1, args=(pSource_0, dCos, w + 0.2, q))
    p3 = multiprocessing.Process(target=trax1, args=(pSource_0, dCos, w + 0.3, q))
    p4 = multiprocessing.Process(target=trax1, args=(pSource_0, dCos, w + 0.4, q))
    p5 = multiprocessing.Process(target=trax1, args=(pSource_0, dCos, w + 0.5, q))
    p6 = multiprocessing.Process(target=trax1, args=(pSource_0, dCos, w + 0.6, q))
    p7 = multiprocessing.Process(target=trax1, args=(pSource_0, dCos, w + 0.7, q))
    p8 = multiprocessing.Process(target=trax1, args=(pSource_0, dCos, w + 0.8, q))
```

```
    p9 = multiprocessing.Process(target=trax1, args=(pSource_0, dCos, w + 0.1, q))
    p10 = multiprocessing.Process(target=trax1, args=(pSource_0, dCos, w + 0.2, q))

    p1.start()
    p2.start()
    p3.start()
    p4.start()
    p5.start()
    p6.start()
    p7.start()
    p8.start()
    p9.start()
    p10.start()

    p1.join()
    p2.join()
    p3.join()
    p4.join()
    p5.join()
    p6.join()
    p7.join()
    p8.join()
    p9.join()
    p10.join()
    print(".....................................................")
    print("Total time :")
    print("--- %s seconds ---" % (time.time() - start_time))

for i in range(0,10):

        A = q.get()
        print(len(A))
```

```
--- 2.0178701877593994 seconds ---
Loading glass calatogs:
--- 2.0388669967651367 seconds ---
Loading glass calatogs:
--- 2.029503107070923 seconds ---
Loading glass calatogs:
--- 2.0563321113586426 seconds ---
Loading glass calatogs:
--- 2.106066942214966 seconds ---
Loading glass calatogs:
--- 2.1395950317382812 seconds ---
Loading glass calatogs:
--- 2.187788724899292 seconds ---
Loading glass calatogs:
--- 2.1048178672790527 seconds ---
Loading glass calatogs:
--- 2.1402010917663574 seconds ---
Loading glass calatogs:
--- 2.1489808559417725 seconds ---
.....................................
Total time :
--- 5.069846153259277 seconds ---
700
700
700
700
700
700
700
700
700
700
```

Figure 36. Console output when running the same ray tracing with 8 processor cores simultaneously.

EE.EXAMPLE - SOLID OBJECTS STL ARRAY

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
Examp Solid Objects STL ARRAY"""
import matplotlib.pyplot as plt
import numpy as np
import pyvista as pv
import Kraken as Kn

P_Obj = Kn.surf()
P_Obj.Thickness = 5000.0
P_Obj.Glass = "AIR"
P_Obj.Diameter = 6.796727741707513E+002 * 2.0
P_Obj.Drawing = 0

FOV = 0.5
Ref_esp = 0.8
Tf = 200  # 40
A1 = Tf * Tf
Conc = 800
Conc = Conc / 0.8
fpl = int(np.round(np.sqrt(Conc) / 2.0))
print("Numero de facetas por lado (Calculo 1)", (fpl * 2.0) + 1.0)
print("Tamanio por lado(mm) ", Tf * ((fpl * 2.0) + 1.0))

n = fpl
FN = 1
focal = Tf * (fpl * 2.0 + 1.0) * FN
print("Distancia focal(mm) ", focal)

sobredim = 2.0 * focal * np.tan(np.deg2rad(FOV / 2.0))
print("Sobredim (mm): ", sobredim)

Tf = Tf - sobredim
A2 = Tf * Tf
RA = A1 / A2
Conc = Conc * RA
print("Nuevo tamaño de las facetas: ", Tf)

fpl = int(np.round(np.sqrt(Conc) / 2.0))
print("Nuevo numero de facetas por lado (Calculo 2)", (fpl * 2.0) + 1.0)
print("Tamanio por lado 2a (mm) ", Tf * ((fpl * 2.0) + 1.0))

n = fpl
Cx = Tf
Cy = Tf
Cz = 0
Lx = Tf
Ly = Tf
Lz = 1.0

element0 = pv.Cube(center=(0.0, 0.0, 0.0), x_length=0.1, y_length=0.1, z_length=0.1, bounds=None)
for A in range(-n, n + 1):
    for B in range(-n, n + 1):
        Ty = 0.5 * np.rad2deg(np.arctan2(Cx * A, focal))
        Tx = -0.5 * np.rad2deg(np.arctan2(Cy * B, focal))
        element1 = pv.Cube(center=(0.0, 0.0, 0.0), x_length=Lx, y_length=Ly, z_length=Lz,
bounds=None)
        element1.rotate_x(Tx)
        element1.rotate_y(Ty)
        v = [-Cx / 2.0, -Cy / 2.0, 0]
        element1.translate(v)
        v = [Cx * A, Cy * B, Cz]
        element1.translate(v)
        element0 = element0 + element1
element0.save("salida.stl")
direc = r"salida.stl"
```

```
objeto = Kn.surf()
objeto.Diameter = 118.0 * 2.0
objeto.Solid_3d_stl = direc
objeto.Thickness = -6000
objeto.Glass = "MIRROR"
objeto.TiltX = 0
objeto.TiltY = 0
objeto.DespX = 0
objeto.DespY = 0
objeto.AxisMove = 0

P_Ima = Kn.surf()
P_Ima.Rc = 0
P_Ima.Thickness = -1.0
P_Ima.Glass = "AIR"
P_Ima.Diameter = 2000.0
P_Ima.Drawing = 1
P_Ima.Name = "Plano imagen"

A = [P_Obj, objeto, P_Ima]
configur = Kn.Kraken_setup()

Telescope = Kn.system(A, configur)
Rays = Kn.raykeeper(Telescope)

W = 0.633
tam = 25
rad = 5500.0
tsis = len(A) + 2
for j in range(-tam, tam + 1):
    for i in range(-tam, tam + 1):
        x_0 = (i / tam) * rad
        y_0 = (j / tam) * rad
        r = np.sqrt((x_0 * x_0) + (y_0 * y_0))
        if r < rad:
            tet = 0.0
            pSource_0 = [x_0, y_0, 0.0]
            dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]
            Telescope.NsTrace(pSource_0, dCos, W)
            if np.shape(Telescope.NAME)[0] != 0:
                if Telescope.NAME[-1] == "Plano imagen":
                    plt.plot(Telescope.Hit_x[-1], Telescope.Hit_y[-1], '.', c="g")
                    Rays.push()

plt.axis('square')
plt.show()
Kn.display3d(Telescope, Rays, 0)
```

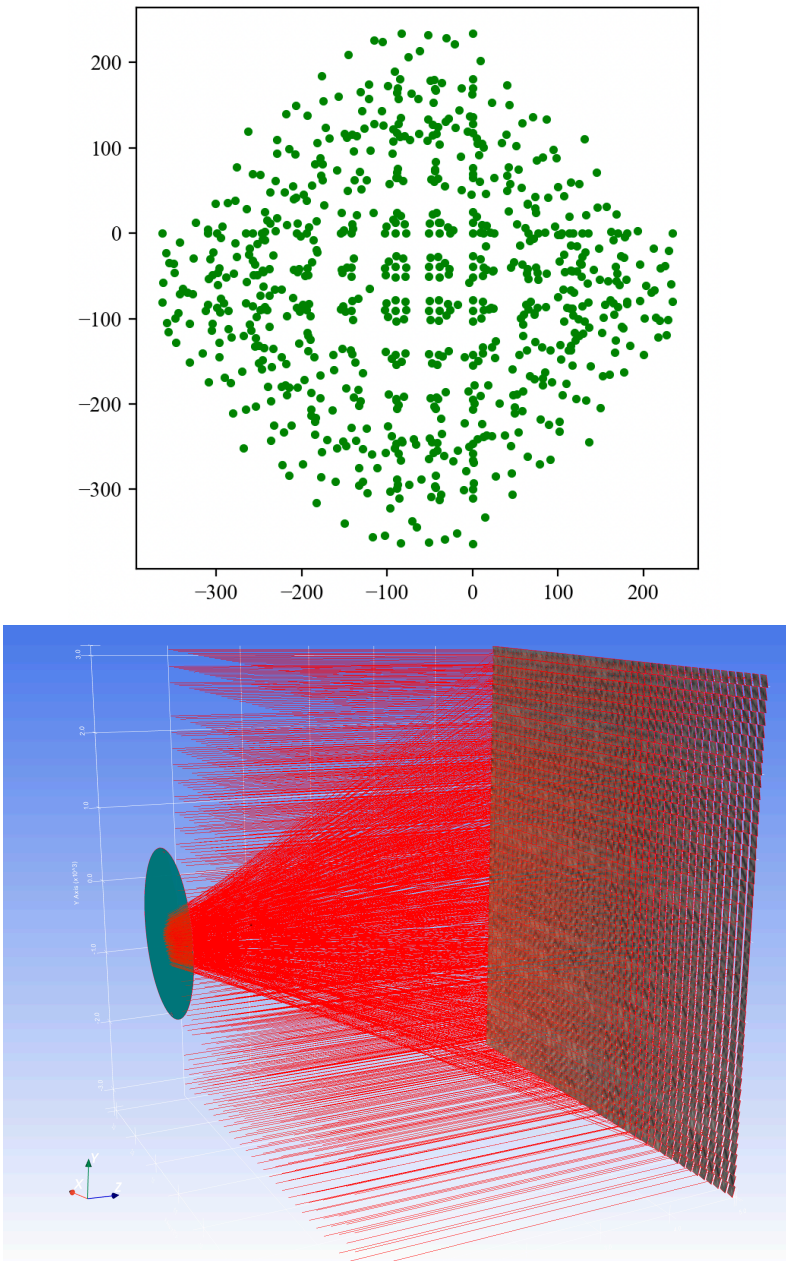


Figure 37. (Top) Spot diagram generated by an arrangement of flat mirrors shown in the figure below.

FF. EXAMPLE - SOURSE_DISTRIBUTION_FUNCTION

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Aug  2 12:04:14 2020
Example - -Sourse_Distribution_Function.py
"""
```

```
import matplotlib.pyplot as plt
import numpy as np
import pyvista as pv
import scipy
import Kraken as Kn

P_Obj = Kn.surf()
P_Obj.Thickness = 5000.0
P_Obj.Glass = "AIR"
P_Obj.Diameter = 6.796727741707513E+002 * 2.0
P_Obj.Drawing = 0
##############################################################################################
objeto = Kn.surf()
objeto.Rc=-12000
objeto.k=-1
objeto.Diameter=2500
objeto.Thickness = -6000
objeto.Glass = "MIRROR"
P_Ima = Kn.surf()
P_Ima.Rc = 0
P_Ima.Thickness = -1.0
P_Ima.Glass = "AIR"
P_Ima.Diameter = 6000.0
P_Ima.Drawing = 1
P_Ima.Name = "Plano imagen"
A = [P_Obj, objeto, P_Ima]
configur = Kn.Kraken_setup()
Telescope = Kn.system(A, configur)
Rays = Kn.raykeeper(Telescope)
W = 0.633
Sun = Kn.SourceRnd()
Example - =4
if Example -  == 0:
    # Sun distribution
    def f(x):
        teta=x
        FI=np.zeros_like(teta)
        Arg2=np.argwhere(teta>(4.65/1000.0))
        FI=np.cos(0.326 * teta)/np.cos(0.308*teta)

        Chi2=.03
        k=0.9* np.log(13.5*Chi2)*np.power(Chi2,-0.3)
        r=(2.2* np.log(0.52*Chi2)*np.power(Chi2,0.43))-1.0
        FI[Arg2]= np.exp(k)*np.power(teta[Arg2] * 1.0e3 , r)
        return FI
    Sun.field =20*np.rad2deg((4.65/1000.0))
if Example -  == 1:
    # Sinc cunction
    def f(x):
        Wh=0.025
        r=(x*90.0/0.025)*np.pi
        res=np.sin(r)/r
        return res
    Sun.field =0.025*3
if Example -  == 2:
    #Flat
    def f(x):
        res=1
        return res
    Sun.field =1.2/(2.*3600.)
if Example -  == 3:
    # Parabolic
    def f(x):
        r=(x*90.0/0.025)
        res=r**2
        return res
    Sun.field =1.2/(2.*3600.)

if Example -  == 4:
    # Gaussian (Seeing)
    def f(x):
        x=np.rad2deg(x)
        seing=1.2/3600.0
        sigma=seing/2.3548
        mean = 0
```

```
        standard_deviation = sigma
        y=scipy.stats.norm(mean, standard_deviation)
        res=y.pdf(x)
        return res
    Sun.field=4*1.2/(2.0*3600.0)

Sun.fun = f
Sun.dim = 3000
Sun.num = 100000
L, M, N, X, Y, Z = Sun.rays()

Xr=np.zeros_like(L)
Yr=np.zeros_like(L)
Nr=np.zeros_like(L)
con=0
con2=0
for i in range(0,Sun.num):
    if con2==10:
        print(100.*i/Sun.num)
        con2=0

    pSource_0 = [X[i], Y[i], Z[i]]
    dCos = [L[i], M[i], N[i]]
    Telescope.Trace(pSource_0, dCos, W)
    Xr[con]=Telescope.Hit_x[-1]
    Yr[con]=Telescope.Hit_y[-1]
    Nr[con]=Telescope.SURFACE[-1]
    con=con+1
    con2=con2+1
    #Rays.push()
args=np.argwhere(Nr==2)
plt.plot(Xr[args], Yr[args], '.', c="g", markersize=1)
# axis labeling
plt.xlabel('x')
plt.ylabel('y')
# figure name
plt.title('Dot Plot')
plt.axis('square')
plt.show()
```

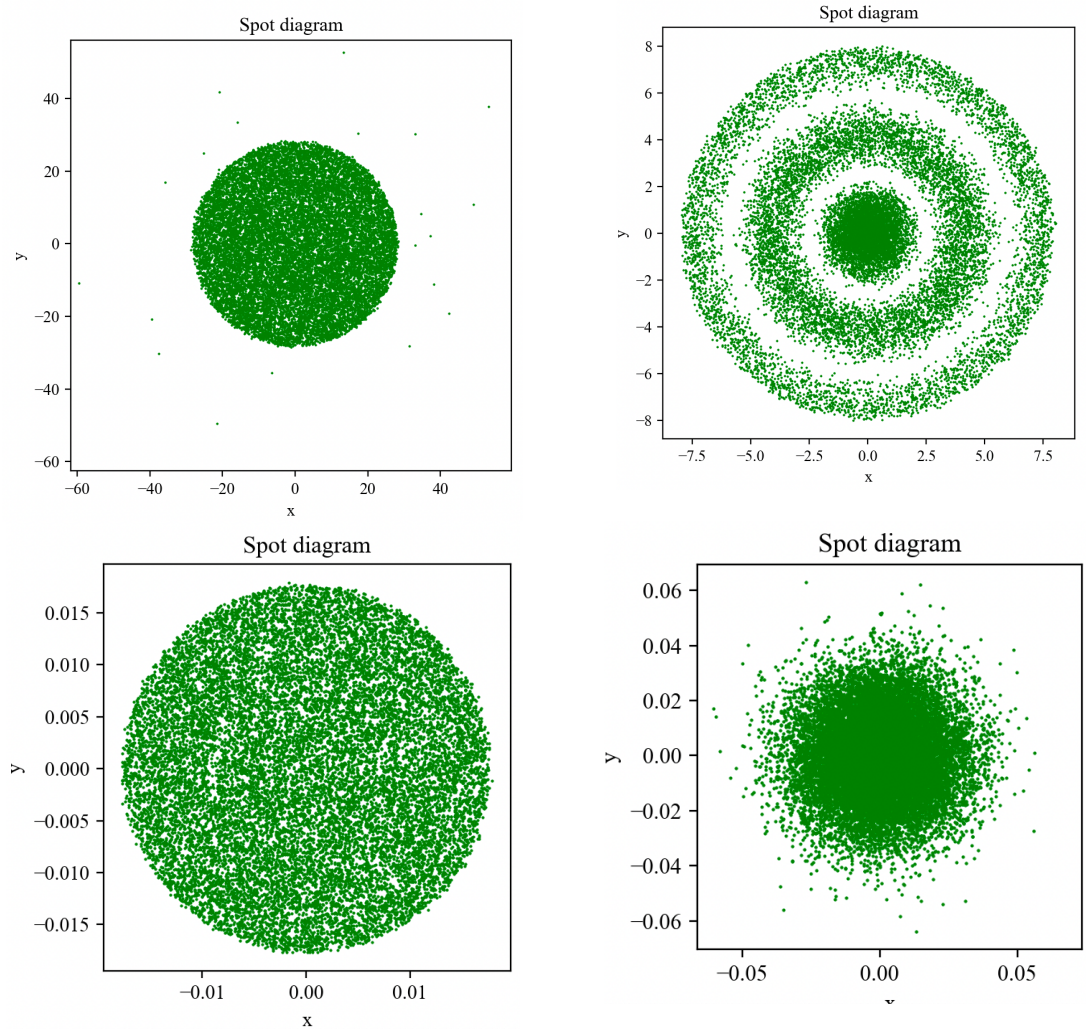


Figure 38. Examples of images generated with rays that come from 4 different sources whose distribution function is a mathematical function. (Top left) Distribution defined by the sun. (Top right) Sync function (Bottom left) constant. (From right to bottom) Gaussian distribution.