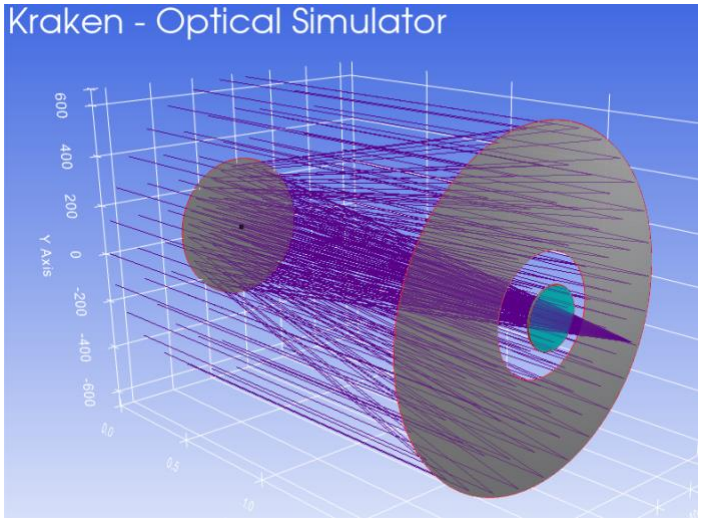
KrakenOS – A provisional user manual

Sorry for the inconvenience,

we are currently working on an english version

Joel Herrera V., Ilse Plauchu-Frayn, Carlos Guerrero P.



ABSTRACT:

This manual illustrates the use of the "Kraken-optical simulator" or "KrakenOS" ray tracing and optical simulation library. This library has been developed in the Python language and was thought to be able to grow and adapt to different user requirements in different Optics tasks. Library It was developed under the paradigm of object-oriented programming, mainly seeking its simplicity and the ability to be incorporated in analysis code or numerical simulations in where a ray tracing is necessary. Being this library developed in Python, there are many tools with which it can be combined increasing its capabilities. At the end of this manual, an appendix with a series of useful examples is made available to the user, with which the capabilities and functions of this library are show

Content

Introduction

The objective of this manual is to present the main instructions, commands, and implementations within the library, in addition, this manual also shows a series of examples in which all the functions available in it are used. KrakenOS is a tool for the simulation of optical systems, it consists of a library developed in the Python 3 programming language and using the Numpy, PyVTK and PyVista libraries, which allows it to provide three-dimensional visualization of the optical elements. This tool has focused on the paradigm of object-oriented programming, this apparently is naturally what gives us greater simplicity in the implementation of a system as will be seen in this document.

1. Prerequisites and installation

The KrakenOS library has been tested on Windows 10, MacOS 11.4, and Ubuntu 20.04LTS. In any of these operating systems, a RAM of 2Gb or higher is recommended, although this depends mainly on the simulated model. The example "Examp_Tel_2M-STL_ImageSlicer.py" is one of the KrakenOS applications that consumes the most processor memory, 2Gb, for most applications a RAM of 512Mb is sufficient.

The library has been tested with the following packages and versions:

- Python '3.7.4'
- numpy '1.18.5'
- scipy '1.7.1'
- matplotlib '3.4.3'
- pyvista '0.25.3'
- pyvtk '0.5.18'
- vtk '8.2'
- csv '1.0'

To install the KrakenOS library, it is necessary to perform a series of simple steps, which are detailed below:

- Clone the repository from the following link: https://github.com/Garchupiter/Kraken-Optical-Simulator
- Place the KrakenOS directory in the same path where the code to be executed is or, failing that, import the library from the path where the KrakenOS directory is located.
- All the files included in the directory and whose name begins with the word "Example - -" are codes with examples, which make use of the different functions of this library.

2. Classes and attributes

The library has been simplified to the extent of having only two classes of objects for the definition of a system. These objects are surf and system, the application of which is described later in the Table 1 and Table 2.

The surf object contains all the relevant information of any optical interface. In this way, every optical interface is an object of the surf class. Each optical interface, from the object plane to the image plane, contains attributes

of size, shape, material, or orientation. In the Table 1 The attributes that must be defined for the surf object are presented.

Table 1. Surf class attributes	
`surf.Name = ""`	Name of the element. Useful only for 2D diagram or ray identification.
`surf.NamePos = (0,0)`	Position of the note with the name “Name” in the 2D diagram. Default value: (0,0)
`surf.Note = "None"`	Useful for adding user notes to a surface. Default value: None.
`surf.Rc = 9999999999.0`	Paraxial radius of curvature in millimeters. Default value: 9999999999.0 or 0.0, both values will be interpreted as a plane.
`surf.Cylinder_Rxy_Ratio = 1`	Ratio between the axial and sagittal radius of curvature. Useful for cylindrical and toroid lenses. Default value: 1
`surf.Axicon = 0`	Axicon simulation, for values other than zero an axicon is generated with the angle entered. Default value: 0
`surf.Thickness = 0.0`	Distance between this surface and the next surface. Default value: 0.0, Units (mm)
`surf.Diameter = 1.0`	Outside diameter of the surface. Default value: 1.0, Units (mm)
`surf.InDiameter = 0.0`	Internal diameter of the surface. Useful for items such as a central aperture primary mirror. Default value: 0.0, Units (mm)
`surf.k = 0.0`	Conicity constant for classical conic surfaces, k = 0 for spherical, k = -1 for parabola, etc. Default value: 0.0
`surf.DespX = 0.0`	Displacement of the surface in the X, Y and Z axis (in millimeters). Default values: 0.0
`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). Default values: 0.0
`surf.TiltY = 0.0`	
`surf.TiltZ = 0.0`	

`surf.Order = 0`	Define 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. Default value: 0
`surf.AxisMove = 1`	Defines what will happen to the optical axis after a coordinate transformation. If the value is 0, 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 will follow the transformation. If the value is different, for example 2, then the optical axis will be affected twice. This is useful for flat mirrors (see Example - Flat Mirror 45 Deg). Default value: 1
`surf.Diff_Ord = 0.0`	Diffraction order. Converts the element into a diffraction grating. The radius of curvature value is automatically omitted to make it a planar diffraction grating. This option can be used in transmission or refraction. Default value: 0.0
`surf.Grating_D = 0.0`	Separation between the lines of the diffraction grating. Default value: 0.0. Units (µm)
`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 dispersion. Default value: 0.0, Units (Degrees)
`surf.ZNK = np.zeros (36)`	Numpy type array of 36 elements that correspond to the coefficients of the Zernike polynomials in the Noll nomenclature (Ref. 2)
`surf.ShiftX = 0`	Offsetting the surface profile feature on the X or Y axis. This is useful, for example, for off-axis surfaces such as parabolas. Default value: 0, Units (mm)
`surf.ShiftY = 0`	
`surf.Mask = 0`	Type of mask to apply. The different values are: (0) Do not apply mask, (1) Use mask as opening, (2) Use mask as obstruction. Default value: 0
`surf.Mask_Shape = Object_3D`	Form of the mask to apply, (Mask built with PyVista (pv) library). Pyvista (Ref 1.) and example from the appendix (7.20)
`surf.AspherData = np.zeros (Fix)`	Array of coefficients for aspherical surface. Array is a list of type Numpy.
`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: -The sagite function is defined as:

	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 -The coefficients are defined, if necessary, in the form of a list. Coef = [3.0, -3, 0] -The shape of the function is assigned to the surface L1c.ExtraData = [f, Coef]
`Surf.Error_map = [X, Y, Z, SPACE]`	It receives an error map generated with an array of type Numpy for the coordinates in X, Y and the height in Z with a space value between X, Y generated from with the following code. Example of error map generation: Example - Tel 2M Error Map
`surf.Drawing = 1`	Value equal to 1 for the element to be drawn in the 3D plot or value equal to 0 to omit the drawing of the element.
`surf.Color = [0,0,0]`	Defines the color of the element in the format [1,1,1]. RGB color percentages (red, green, blue). Default value: (0,0,0) is black. Other colors are red (1,0,0), green (0,1,0), and blue (0,0,1).
`surf.Solid_3d_stl = "None"`	Path of the 3D solid object in STL format.

The system object is intended as a container for all interfaces. This object contains implementations for ray tracing and for obtaining different parameters of the ray, which are cumulative across the surfaces through which the ray passes. To understand how these elements are called, the reader can refer to the example in Appendix7.11.

In the Table 2 The public implementations of the system object are presented.

Table 2. Implementations and attributes of the system class

`system.Trace (pS, dC, wV)`	***Trace*** is the main implementation of the system object; it traces a 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 the following examples: pS = [1.0, 0.0, 0.0] dC = [0.0,0.0,1.0] wV = 0.4
`system.NsTrace (pS, dC, wV)`	Like Trace, but non sequentially. The ray parameters are defined in the same way as in Trace.
`Prx = system.Parax (w)`	Returns the following paraxial calculations also accessible from system, these are arranged in a list with the following elements: Prx = SistemMatrix, S_Matrix, N_Matrix, a, b, c, d, EFFL, PPA, PPP, CC, N_Prec, DD
`system.disable_inner`	Enables and disables the central openings. This is very useful, for example, used to calculate a ray trace without vignetting the aperture of a primary mirror.
`system.enable_inner`	
`system.SURFACE`	Returns a list of the number of surfaces the ray passed through.
`system.NAME`	Returns a list of the names of the surfaces that the ray passed through. If no name is given to the surfaces, then the list will appear with empty fields. Naming surfaces is very useful, for example, to identify whether ray has struck that surface.
`system.GLASS`	Returns a list of the materials that the ray has pierced.
`system.XYZ`	Returns the [X, Y, Z] coordinates 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 image plane.
`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 intersection points.

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 a single value with the total optical path of the ray from the source to the last point of intersection.
system.TOP_S	Returns a cumulative list with values of the ray's optical path from the source to the last intersection point.
system.ALPHA	Returns a list with the absorption coefficients for the materials on the intercepted surfaces considering the wavelength. These values are obtained from the originally provided material catalog.
system.BULK_TRANS	Returns a list with the transmission through all the paths 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 director cosines [L, M, N] of the normal for the intersection points 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 director cosines [L, M, N] of the resulting ray, through all the interfaces through which it passes.
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	Refractive indices 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. The difference with system. N0 is that the list starts from the second refractive index, this is useful to differentiate between the middle index list before and after an iteration. Example: N0 = [n1, n2, n3, n4, n5] N1 = [n2, n3, n4, n5, n5] In such a way that if we want to know the direction of a ray in the first interface we will use N0 [0] and N1 [0]

`system.WAV`	Wavelength of the ray in question. Although this command returns a list of all the values, they are the same because the wavelength is constant for a ray. The size of the list indicates only the number of iterations with system interfaces. Units (µm)
`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_D`	Distance between lines of the diffraction grating. Units (µm)
`system.RP`	Returns a list with the Fresnel reflection and transmission coefficients for polarization S and P.
`system.RS`	
`system.TP`	
`system.TS`	
`system.TTBE`	Total energy transmitted or reflected per element.
`system.TT`	Total energy transmitted or reflected total.
`system.targ_surf (int)`	Limits the ray tracing to the defined surface, this is an integer equal to the surface number, if the value is zero (by default) it means that the ray tracing is carried out to the last surface.
`system.flat_surf (int)`	It is defined with the whole number of the surface that needs to be made flat. The value -1 is used to restore the surface to its original shape.

3. Working with the KrakenOS Library

In this section, an example is presented in which a ray is generated, which will pass through several surfaces to later extract information from it.

It is very important to mention that, in Python, classes define an object with their attributes, these, when created, acquire all the attributes available in the class. For example, if an object of the system class is created, this creation will require as a parameter a list of surfaces and a configuration, these can have the name that the user wants. The above is shown later in this manual, specifically in the Code 2. In this code, the surface list is named A, the system configuration is configuration_1, and the created optical system is named Doublet for

convenience. Said names (A, configuration_1 and Doublet) are not determined within the library, they are simply names that we have assigned to these new objects, with which we can interact through the attributes of the class used to create them.

3.1 Ray generation

Note: All rays and optical systems are drawn and traced from left to right, however KrakenOS allows rays to be traced from right to left. The separation between surfaces is defined with a positive sign from left to right and negative otherwise. The user will notice that it is not strange to find negative thickness or thickness values, the clearest example is in the use of mirrors. See Appendix (Example - Tel 2M Pupila).

To make use of the KrakenOS library, we first import them.

```
import numpy as np
import KrakenOS as Kos
```

For command line help, after importing the library, you can use the documentation in code using:

help (Kos), help (Kos.system) or help (Kos.surf)

The following example is found within the program: Example - Ray (see Sec. 5.1 and Appendix 7.1). All optical surfaces must be declared separately. When declaring a surface, use is made of all or some of the possible parameters that it has. For a description of each of these parameters see the Table 1. At Code 1 shows how five different surfaces have been defined.

Code 1.	

`P_Obj = Kos.surf()`
`P_Obj.Rc = 0.0`
`P_Obj.Thickness = 0.1`
`P_Obj.Glass = "AIR"`
`P_Obj.Diameter = 30.0`
	Here the object plane is defined as a surface of the surf type, values are assigned to the attributes of Radius of curvature (Rc), separation with the next surface (Thickness), the type of material (Glass) and its diameter (Diameter).

`L1a = Kos.surf()`
`L1a.Rc = 92,847`
`L1a.Thickness = 6.0`
`L1a.Glass = "BK7"`
`L1a.Diameter = 30.0`
`L1a.Axicon = 0`
	Here is defined the first face of a doublet and its attributes. The BK7 material is one of the most used materials in optics, it is a glass of the type ***Crown*** (low dispersion).

`L1b = Kos.surf()`
`L1b.Rc = -30,716`
`L1b.Thickness = 3.0`
`L1b.Glass = "F2"`
`L1b.Diameter = 30`
	Here is defined the second face of a doublet and its attributes. The material F2 is one of the most used materials in optics, it is a glass of the type ***Flint***, (high dispersion).
	The BK7 and F2 are widely used in the construction of achromatic doublets. (*Ref. 4*, sec 11.6)

`L1c = Kos.surf()`
`L1c.Rc = -78.19`
`L1c.Thickness = 97.37`
`L1c.Glass = "AIR"`
`L1c.Diameter = 30`
	Here the third face of a doublet and its attributes are defined.

`P_Ima = Kos.surf()`
`P_Ima.Rc = 0.0`
`P_Ima.Thickness = 0.0`
`P_Ima.Glass = "AIR"`
`P_Ima.Diameter = 18.0`
`P_Ima.Yam = "Image plane"`
	Here the surface referring to the image plane and its attributes is defined.

Within the KrakenOS library there is the Setup object that only has the function of setting the environment. There are certain parameters that must be loaded from the creation of an optical system, such is the case of the glass catalogs that will be used. The content of this class (Setup) is very simple and must be modified to include the catalog of interest from the "Cat" directory. This class is stored in the SetupClass.py file in the "KrakenOS" directory. The following code fragment shows the five catalogs that are loaded by default. The "UTILIDADES.AGF" catalog must always be included in the list since the KrakenOS library depends on it to function.

```
cat1 = (route + '/KrakenOS/Cat/SCHOTT.AGF')
cat2 = (route + '/KrakenOS/Cat/TSPM.AGF')
cat3 = (route + '/KrakenOS/Cat/INFRARED.AGF')
cat4 = (route + '/KrakenOS/Cat/UTILIDADES.AGF')
filepath = [cat1, cat2, cat3, cat4]
```

In future versions of the KrakenOS library, different configurations can be established for a system, which may be useful for systems with multiple configurations with different conditions. For practical purposes in this user manual, we will load the default configuration into the variable configuration_1, which is done as follows:

```
configuration_1 = Kos.Setup ()
```

As shown in the Code 2, a list A is created with the elements declared for all surfaces, the default configuration is loaded and, later, a Doublet optical system is created with all the surfaces contained in A and configuration_1.

Code 2.	
`TO = [P_Obj, L1a, L1b, L1c, P_Ima]`	Here a list named A is created with the created surfaces.
`configuration_1 = Kos.Setup ()`	The default configuration is loaded with the catalogs in the file "KrakenOSSetupClass.py ".
`Doublet = Kos.system (TO, configuration_1)`	And the object named as Doublet is generated.

Every ray has three parameters: the origin coordinates represented in this example by the variable XYZ, which is a list of the type $[x_1 y_1 z_1]$ and a direction defined by director cosines, represented by the variable LMN, which is a list of the type [L M N]. This last variable of directing cosines, for a ray parallel to the optical axis, would have the following values [0,0,1], that is, this vector does not have an x or y component. The equation for the director cosines is given by:

$$L = \frac{x_2 - x_1}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}}$$

$$M = \frac{y_2 - y_1}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}}$$

$$N = \frac{z_2 - z_1}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}}$$

Where the intersection points of the ray on the first surface are given by $[x_2 y_2 z_2]$.

Before ray tracing, the point where the ray will hit the first surface is unknown. For this reason, it is necessary to define the director cosines as a function of the direction of the ray, or, in other words, of its angle of incidence. Suppose a ray that arrives at an angle Theta with respect to the y axis, but without any angle with respect to the x axis. So this ray will only have components on the z and y axis, and the value of L will remain equal to zero, while M will be ***Sine(Theta)*** and N will be ***Cosine(Theta)***.

The third parameter to be defined is the wavelength, which is expressed as W equal to 0.4μm Code 3.

Code 3.

```
XYZ = [0, 14, 0]
Theta= 0.1# Any field for the example
L M N = [0.0, np.without(np.deg2rad (Theta)), -np.cos (np.deg2rad (Theta))]
W = 0.4
```

At Code 3, the direction of the ray has been defined in a very general way with the angle of incidence of the Theta itself. To generate rays with different components in the x and y direction, a tool for automatic generation of rays has been integrated into the PupilCalc tool that will be seen later.

Continuing with the previous example, the ray is traced through the Doublet system as follows:

```
Doublet.Trace(XYZ, L M N, W)
```

It is possible to interrogate the Doublet system about what happened to the ray. Communication with the Doblete 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 and is used in the following way.

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

The above command returns the following array as output ['BK7', 'F2', 'AIR', 'AIR']. For this example, the ray has passed through BK7 which is L1a, F2 which is L1b and two air surfaces which are L1c and the image plane.

Another type of information that can be extracted would be the coordinates of the ray on all surfaces and / or the directing cosines. The above is obtained with the XYZ and LMN commands as follows:

```
print(Doublet.XYZ)
print(Doublet.L M N)
```

with which we obtain the following console outputs respectively:

```
Doublet.XYZ = [array ([0., 10., 0.]), [0.0, 10.0, 10.54009083298281], [0.0, 9.85609739239213,
14.37575625216807], [0.0, 9.81741071793073, 18.381280697704405], [0.0, 0.033888251065], [0.0738298297],
1167529760 ]


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 directly carry out the operations we want on them. Next, we see the form of both arrays:

```
print(np.shape (Doublet.XYZ))
print(np.shape (Doublet.L M N))
```

which produces as output:

```
np.shape (Doublet.XYZ) = (5, 3)
np.shape (Doublet.L M N) = (4, 3)
```

Note that Doublet.XYZ contains five arrays of three elements (x, y, z coordinates in three-dimensional space). On the other hand, Doublet.LMN contains only four elements, because it only shows the director cosines between the surfaces. That is, if a system has five elements from the object plane to the image plane, then there are only four ray segments between surfaces, and therefore only four sets of director cosines. In the following command line, each set of cosines between the surfaces is exemplified with the symbology "->":

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

3.2 Extraction of ray information

Once we have traced the ray with the Trace option, we can extract information from it with the attributes of the Doublet object, making use of the parameters described in the Table 2. With the information obtained it is possible to generate graphs, but generally it is necessary to have many rays. For example, a spot diagram is a tool that in Optics needs to trace many rays that come from the object, go through the pupil of the system, and reach the image plane. The intersection points of these rays are plotted and provide a great deal of information about the aberrations of the system. To preserve the results for one or more rays, we have the ray container in the Raykeeper class. Continuing with the Ray example from Sec. 5.1, we create the "Rays" object of the Raykeeper type, which will contain all the information about the ray or rays:

```
Rays = Kos.raykeeper (Doublet)
```

A keyword of the Raykeeper class is push (), with which we indicate that the ray just drawn is saved inside Doublet. An advantage of this is that the Doublet object does not have to save a memory with all the rays that we trace, this task is carried out by the ray container. In this way we can generate ray containers for different circumstances as desired. For example, we can create a container for all the rays that we will trace from one field and then store the rays that come from a different field in another container. The user will find this KrakenOS tool very useful.

The following command line indicates how this information is stored, which must be repeated every time a ray is traced by Doublet:

```
Rays.push ()
```

3.3 Generation of the optical system graph

On the other hand, if is needed system plot, there are two tools: Display2D and Display3D. The input parameters for these said tools are the system itself and a ray container.

Additionally, each of these tools receives an extra parameter. In the case of Display2D (System, Raykeeper, parameter), with which a graph of the system is generated in 2 dimensions, "parameter" takes the value 0 to indicate that the graph will be in the xz plane or the value 1 in the plane and Z. Likewise, if we want to generate a graph of the system in 3 dimensions, we will use Display3D (System, Raykeeper, parameter) where "parameter" can take the following values:

- 0 for the display of the complete elements.
- 1 for unfolding surfaces with a 1/4 cutout.

- 2 for unfolding surfaces with a 1/2 cutout.

For the example in this document, we will have to replace system and Raykeeper with the objects created with them as follows:

```
Kos.Display2D(Doublet,Ray,0)
```

The above command line will generate a graph that is displayed in Matplotlib. Let's remember that we only keep one ray in the container. This ray is shown in purple as it depends on the wavelength originally used to define the ray (ie, W = 0.4µm). If a value had been assigned to W = 0.6µm, the ray would have been automatically graphed in red. Therefore, the color of the ray is defined by the program depending on the wavelength value used. If the assigned wavelength is higher or lower than the range of the visible spectrum, then the ray will be displayed in black.

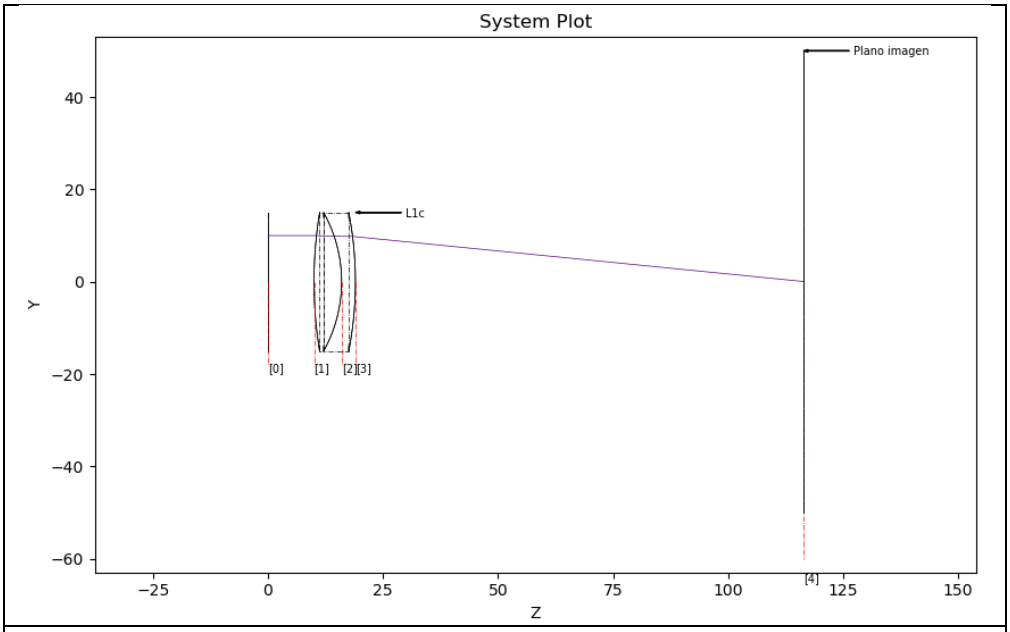


Figure 1. 2D visualization of a ray traced through a doublet

Display2d has an optional fourth parameter, whose default value is equal to zero. If this value is modified by a value greater than zero (ie, positive value), then each ray is graphed with an additional arrow. This is useful if it is needed to indicate the direction of the ray on the graph. The value of this fourth parameter defines the size of the ray arrow. The effect of using this option can be seen later in the Figure 4.

```
Kos.Display2D(Doublet,Ray,0, 1)
```

If, on the other hand, we want to display the graph of the same optical system in its three-dimensional form, we use the following command line:

```
Kos.Display3D(Doublet,Ray,2)
```

With the above command, the following viewer window will open, as shown in Figure 2.

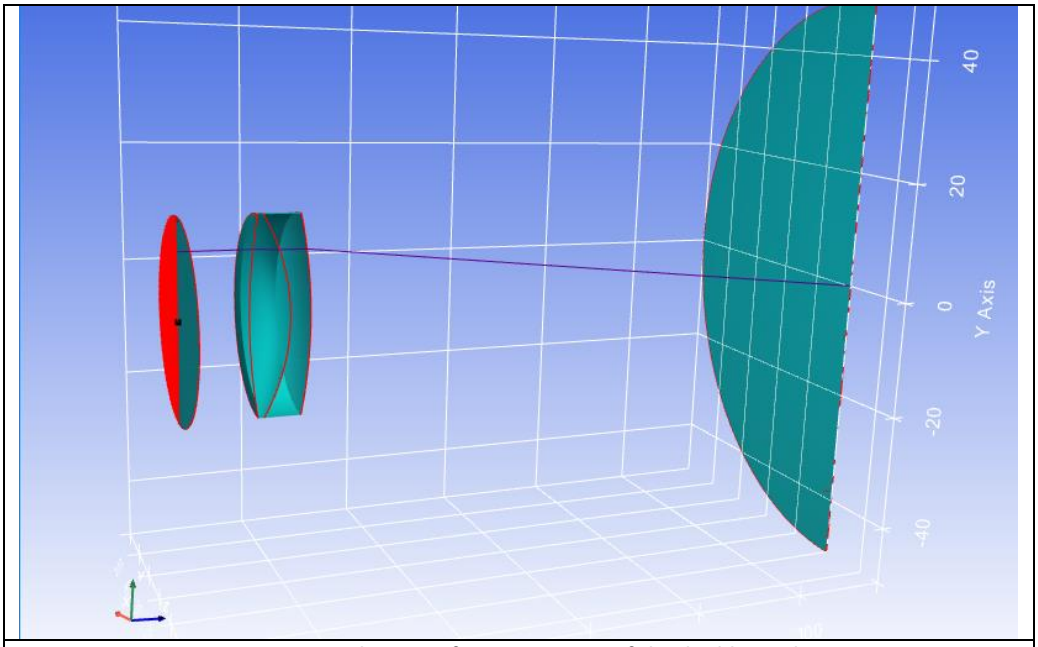


Figure 2. 3D visualization of a cross section of the doublet and a ray

It is also possible to store multiple rays. For example, in the Code 4 the creation of the ray, its traced and its storage are nested within a for loop.

Code 4	

`for y in range(-10, 10):`
`  XYZ = [0.0, y, 0.0]`
`  L M N =[0.0,0.0,1.0]`
`  W=0.4`
	Here a ray is created parallel to the optical axis within the cycle for and the height is indicated at the "y" position.

`  Doublet.Trace (XYZ, L M N,W)`	Ray tracing is done.
`  Ray.push (Doublet)`	The ray information is stored.
`Kos.Display3D(Doublet,Ray,2)`	And the system is plotted with all the rays.

With Code 4 example 20 rays within the range -10 to 10 (20 rays because Python does not perform the last element of the for loop). If the ray does not touch any surface, then this ray is not traced. In Figure 3 the 20 rays generated pass through the optical system.

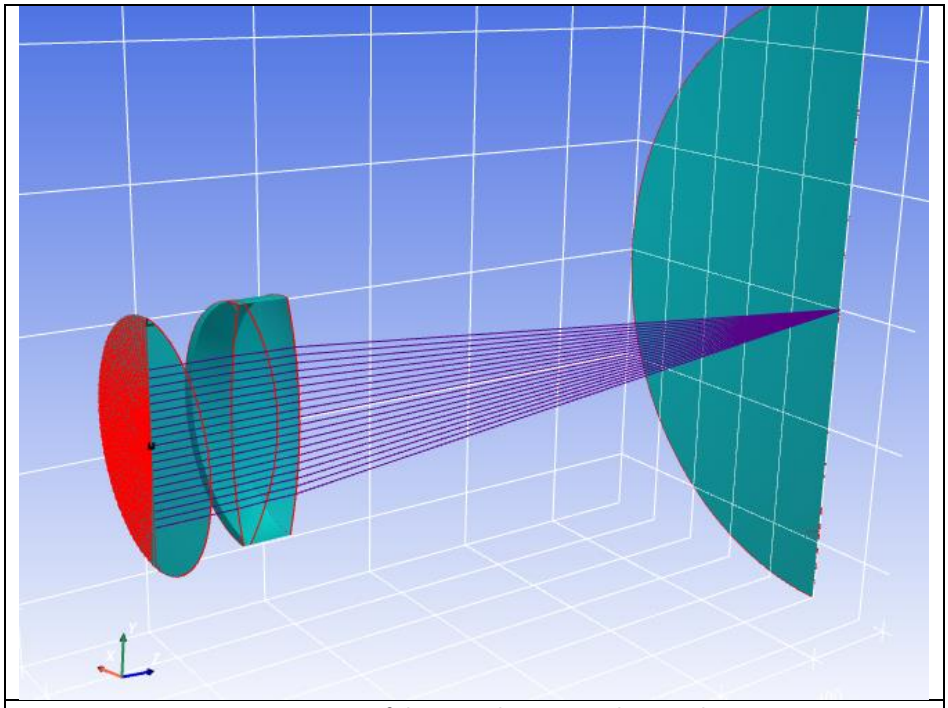


Figure 3. 3D view of the optical system with several rays

The ray container contains a list that holds almost all the parameters that are accessible in the system class. However, now, since they contain several rays, these parameters are arrays of arrays, so it is necessary to take certain considerations.

By default, the Raykeeper ray container considers 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 direction. In any case, the ray is taken and the existing information about it: point of origin, director cosines and wavelength, but the rest of the data will be empty.

The above can be useful if, for example, we launch 100 rays. To know the number of rays stored, Raykeeper has an internal variable nrays that can be called directly as follows:

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

Therefore, we would have 100 positions with rays about which we can request information from the container, this by using the parameters shown in the Table 3 (Central column). These parameters have the same information described in the Table 2 for the system class. Calls to the Raykeeper container depend on the number of rays defined by an integer value [#]. If the ray number is not indicated, the total data set is obtained in Numpy arrays. As mentioned above, there are rays that have not touched any surface and that we name "empty rays" and rays that do touch any and that we name "valid rays." We can ask the container for a list of valid rays contained in the list of ray list with the following command:

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

Thus, 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]]

Once the above is done, we can request information about the ray using the keywords of the Table 3 (Left column). However, executing the valid () statement on the (Rays.valid ()) line creates a new set of calls. These calls are the ones shown on the Table 3(central column), which are like those of the left column, but with the prefix "valid_". It is important to note that if the valid statement is not added, the calls shown in the right column will be empty and therefore errors will be obtained.

The arrays obtained with this new set of calls contain only valid rays, which can be used directly as the empty rays have been eliminated. Therefore, the numbering will be offset according to the number of empty rays discarded. Depending on the task to be carried out, it could be applied in one way or another.

In the same way, a set of calls is created for the invalid rays, which are the empty rays and whose tracing was not carried out, since they do not intersect any surface or because they do not reach the image plane. The calls to these rays are shown in the Table 3 (right column). From these calls it is only possible to extract the information of the origin of ray, which is limited to coordinates of origin, direction, and wavelength. The prefix for such calls is "invalid_".

Table 3. Raykeeper container attributes

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 remove the ray information contained within the Raykeeper container, simply redefine said container or delete its content using the internal clean implementation.

```
Ray.clean ()
```

4. Parax Tool

The Parax tool is an implementation of the system class. This tool performs a series of paraxial calculations, which basically provide all the information necessary for paraxial ray tracing in matrix form. These calculations are carried out with the object of the system and for the requested wavelength as follows:

```
Prx = Doublet.Parax (0.4)
SistemMatrix, S_Matrix, N_Matrix, to, 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 matrix element by element:

```
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.]])]
```

Name of the matrix to support identification by the user:

```
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 the total system matrix:

```
a = 0.8693294662072478
b = -0.009808407908592333
c = 100.16768993870667
d = 0.02014706564149671
```

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

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

List of curvatures (C), List of refractive indices (N) and list of separations between surfaces (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.]
```

5. PupilCalc Tool

The system aperture, or as it is commonly known "Aperture Stop" (Ref. 5, section 6.2), is the element that defines the amount of light that passes through the entire optical system. The image of the aperture of the optical system, produced by the optical elements before it, is known as the entrance pupil. Likewise, the image of the entrance pupil, produced by the entire optical system, is known as the exit pupil. Unlike what could initially be assumed, the entrance pupil is not necessarily defined by the first element of the optical system, since sometimes the element that defines the amount of light that enters the system is not an optical surface but an element mechanical.

In the Figure 4 an example of an “Aperture stop” or “System aperture” is shown on an element behind a lens, specifically on surface 4 (see in more detail in Appendix 7.10 Examp_Doublet_Lens_Pupil.py). Similarly, on the left side of this figure, two beams of rays are shown at two different field angles (+ 2 ° and -2 °). Said rays come from two objects (from minus infinity) and therefore, the rays are collimated and simultaneously cross this aperture (surface 4). (Ref. 5, section 6.2).

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 Figure 4 that the fields have different origins in the object plane,

therefore, generating the rays one by one can be a complicated task if one by one is calculated as it was done in the Code 3.

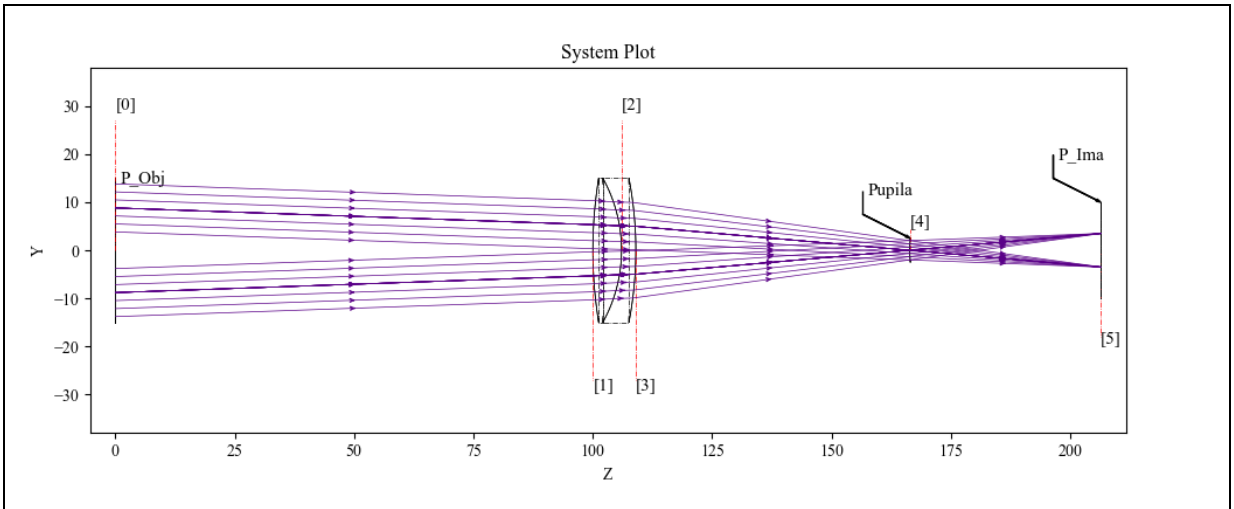


Figure 4. Visualization of two beams that coincide in the position defined as the pupil.

Therefore, and to facilitate the generation of rays that cross the pupil, the PupilCalc tool is counted, this is a class, which generates an interactive object as shown in the Code 5:

Code 5.

```
W = 0.4

Surf= 4

AperVal = 10

AperType = "EPD"

Pup = Kos.PupilCalc (Doublet, His p, W, AperType, AperVal)
```

Where:

- **Doublet** is the optical system that we generate with the system class
- **Surf** is the number of the surface that represents the system aperture (surface 4 in the Figure 4).
- **W** is the wavelength for which the exit and entrance pupil will be calculated.
- **AperType** It is a parameter that can have the following two values: “STOP” or “EPD”. By using the value “STOP” it will be indicating that the surface defined in Surf is the system aperture, while using “EPD” it will be defining the diameter of the entrance pupil.
- **AperVal** is he diameter of the system aperture or entrance pupil, depending on how AperType is configured.

At Code 5, for the object named Pup, the parameters of the pupils are obtained as shown in the **Error! Reference source not found.**:

Table 4. Attributes object corresponding to pupil parameters.	
`Pup.RadPupInp`	Entrance pupil radius.
`Pup.PosPupInp`	Entrance pupil position.
`Pup.RadPupOut`	Exit pupil radius.
`Pup.PosPupOut`	Exit pupil position.
`Pup.PosPupOutFoc`	Exit pupil position with respect to the focal plane.
`Pup.DirPupSal`	Exit pupil orientation.

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

In addition to obtaining these parameters, it is also possible to generate ray patterns in the pupil, by calculating the director cosines and the origin coordinates, from the definition of the parameters.

Table 5. Generation automatic ray based on the pupil	
`Pup.Samp = #`	Integer number for pupil ray sampling. The default value is 5.
`Pup.Ptype = "type of array"`	Where "Type of array" can take the following values: **"rtheta"**generates a beam at an angle from the unitary pupil to a radial position. The radius and angle must be defined as follows: **`Pup.rad = n`** **`Pup.theta = m`** Where n is a floating number between 0 and 1 and theta an angle between 0 and 360.
	"chief": Chief 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 array.

`Pup.FieldType = "height" or "angle"`	Defines the field type, this in terms of the height of the object and the distance from the object plane. "Height" is used for parallel rays reaching the pupil and from infinity the "angle" parameter is used.
`Pup.FieldY = #` `Pup.FieldX = #`	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 Table 5 define the type of rays desired. To obtain the array of rays, all are transferred from the unitary pupil to the real pupil, obtaining the directing cosines and the origin coordinates in the form of arrays. This is done as follows:

```
x, and, 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, it is possible to iterate between them and make the ray tracing through the system as shown in the Code 6., where we continue to work with the Doublet optical system and the Rays container.

Code 6.

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

On the other hand, it is also possible to plot the points of the rays generated in the object plane. For example, in the Figure 5 a "hexapolar" pattern has been defined.

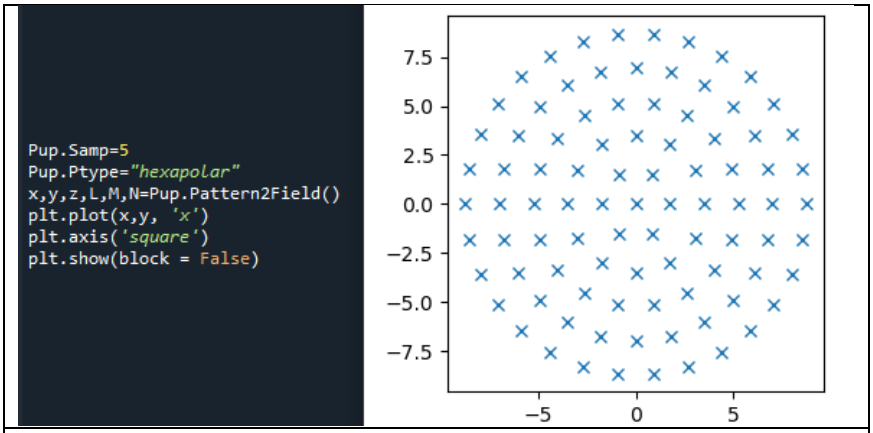


Figure 5. Generated code and pattern for the origin of rays with a hexapolar distribution

Likewise, the spot diagram can be generated (Figure 6):

```
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)
```

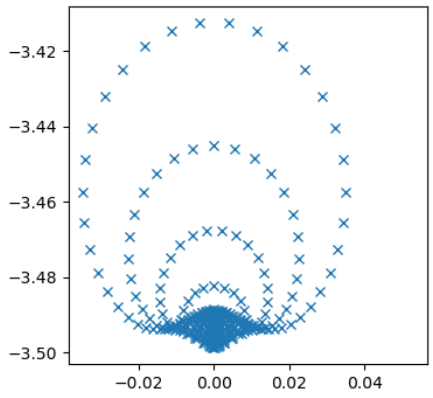


Figure 6. Code and spot diagram generated with a hexapolar pattern and with a field angle of 2 °

It is important to note that to analyze different fields, it is needed to make arrays for each field and also save the ray tracing in different containers in order to avoid combined the results.

5.1 Atmospheric Refraction in PupilCalc

If the FieldType parameter (see Table 5.) is defined as “angle”, its use will be appropriate for optical systems such as telescopes, where the rays are considered to come from infinity. With this idea in mind, the open source AstroAtmosphere library (*Ref. 3*).

The AstroAtmosphere library performs the calculation of the deviation of a ray, depending on the physical parameters of the observatory, the geographical latitude in degrees, the height of the site in meters, the humidity, the amount of CO2 in the atmosphere in parts per million, the reference wavelength and the zenith distance at which the object is being observed in degrees. The PupilCalc tool uses this library internally, to calculate the modification that the rays require. This function is configured as a parameter of PupilCalc as shown in the Code 7:

Code 7.

```
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.The t = 31 # Latitude (degrees)
Pup.h = 2800 # Observatory height (meters)
Pup.l1 = 0.60169 # Reference wavelength(micron)
Pup.l2 = 0.50169 # Wavelenght of interest(micron)
Pup.z0 = 55.0 # Zenith distance (degrees)
```

Then in the Code 8 an example is shown using a telescope where 3 groups of rays are generated, only changing the wavelength between them:

Code 8.

```
W1 = 0.50169
Pup.l2 = W1
for,already,za,The,Ma,Na=Pup.Pattern2Field
()

W2 = 0.60169
Pup.l2 = W2
xb,and b,zb,Lb,Mb,Nb=Pup.Pattern2Field ()

W3 = 0.70169
Pup.l2 = W3
xc,and c,zc,Lc,Mc,Nc=Pup.Pattern2Field ()
```

At Code 9, the ray tracing is then performed for the three groups and they are stored in different containers.

Code 9.

```
for i in range(0,len(for)):
 pSource_0 = [for[i], already[i], za[i]]
 dCos=[The[i], Ma[i], Na[i]]
 Telescope.Trace (pSource_0, dCos, W1)
 Rays 1.push ()

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

for i in range(0,len(xc)):
 pSource_0 = [xc[i], and c[i], zc[i]]
 dCos=[Lc[i], Mc[i], Nc[i]]
 Telescope.Trace (pSource_0, dCos, W3)
 Rays 3.push ()
```

As a result of making the spot diagrams we obtain the graph of the Figure 7, generated with code 10, where can be seen a separation of approximately 72 µm between the extreme wavelengths (blue and green crosses in Figure 7).

Code 10.

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


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


X,AND,Z,L,M,N=Rays 3.pick (-1)
plt.plot   (X   *1000.0,AND*1000.0,   'x',
c="g")
```

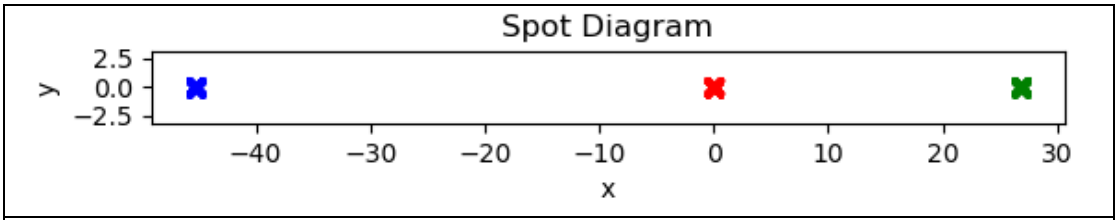


Figure 7. Images formed by a telescope which are spectrally displaced by the action of atmospheric refraction.

6. Handling the 3D viewer

When generating a three-dimensional graphic, it is also possible to perform the following actions in it for a better visualization of it:

- The viewer takes control of the execution in Python, so Display3D must be the last instruction of a script, if you place Display3D before another calculation, the Python kernel will pause until the 3D viewer window is closed with the close button.
- Rotate the simulation in any direction. To do this, move the mouse while holding down the left mouse button.
- Zoom out or zoom in. To do this, move the mouse up and down, while holding down the right mouse button.
- Drag the simulation. To do this, move the mouse while simultaneously pressing the left mouse button and the SHIFT key.

References:

Ref. 1: Ref: https://docs.pyvista.org/examples/00-load/create-geometric-objects.html#sphx-glr-examples-00-load-create-geometric-objects-py

Ref. 2: "Zernike polynomials and atmospheric turbulence", R. Noll, J. Opt. Soc. Am. 66, 207-211 (1976).

Ref. 3: "Quantification of the expected residual dispersion of the MICADO Near-IR imaging instrument", by the authors van den Born & Jellema, 2020, MNRAS, DOI:10.1093 / mnras / staa1870

Ref. 4: "Handbook of Optical Design", Daniel Malacara-Hernández, Zacarías Malacara-Hernández, Zacarias Malacara, Edition 2, CRC Press, 2003, ISBN: 0203912942, 9780203912942

Ref. 5: "Modern Optical Engineering The Design of Optical Systems", Warren J. Smith, Third Edition, McGraw-Hill, 2000. ISBN 0-07-136360-2

7. Appendix - Examples

The authors of this manual believe that the best way to understand the use of the KrakenOS library is to practice with the examples included within it. The file names for each available example are listed below, followed by the code for each.

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
Examp-Tel_2M_Spyder_Spot_RMS.py
Examp-Tel_2M_Spyder_Spot_Tilt_M2.py
Examp-Tel_2M_Atmospheric_Refraction.py
Examp-Tel_2M_Wavefront_Fitting.py

7.1 Example - Ray

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


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

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

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

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

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

P_Ima = Kos.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]
configuracion_1 = Kos.Setup()

Doblete = Kos.system(A, configuracion_1)
Rayos = Kos.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
Doblete.Trace(pSource_0, dCos, W)
Rayos.push()

Kos.display3d(Doblete, Rayos, 2)
```

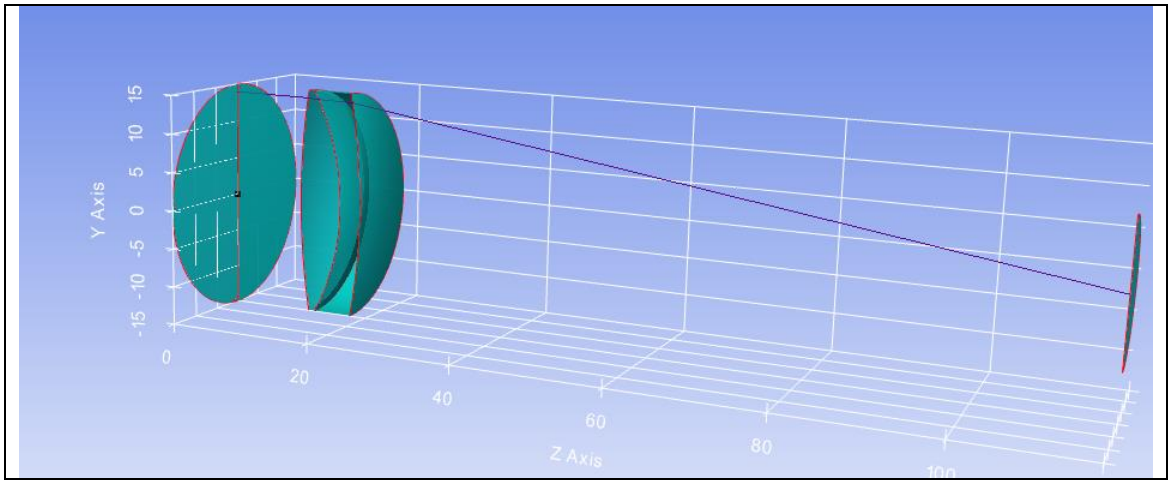
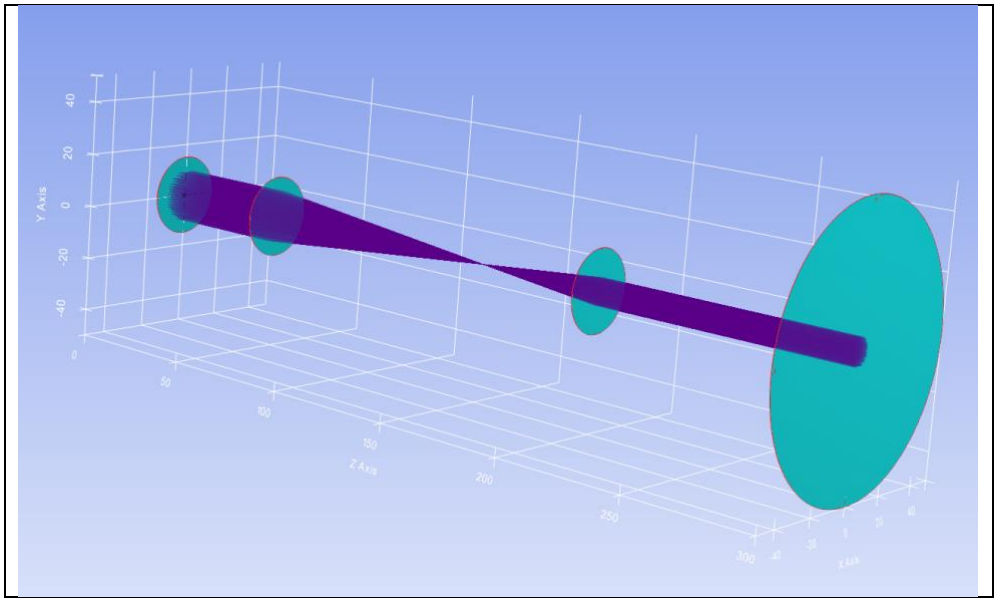


Figura 8. Example de el trazo de un único rayo a través de un sistema óptico.

7.2 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 KrakenOS as Kos

start_time = time.time()

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

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

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

P_Ima = Kos.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 = Kos.Setup()

Doblete = Kos.system(A, config_1)
Rayos1 = Kos.raykeeper(Doblete)
Rayos2 = Kos.raykeeper(Doblete)
Rayos3 = Kos.raykeeper(Doblete)
RayosT = Kos.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
            Doblete.Trace(pSource_0, dCos, W)
            Rayos1.push()
            RayosT.push()

Kos.display3d(Doblete, RayosT, 0)
X, Y, Z, L, M, N = Rayos1.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))
```



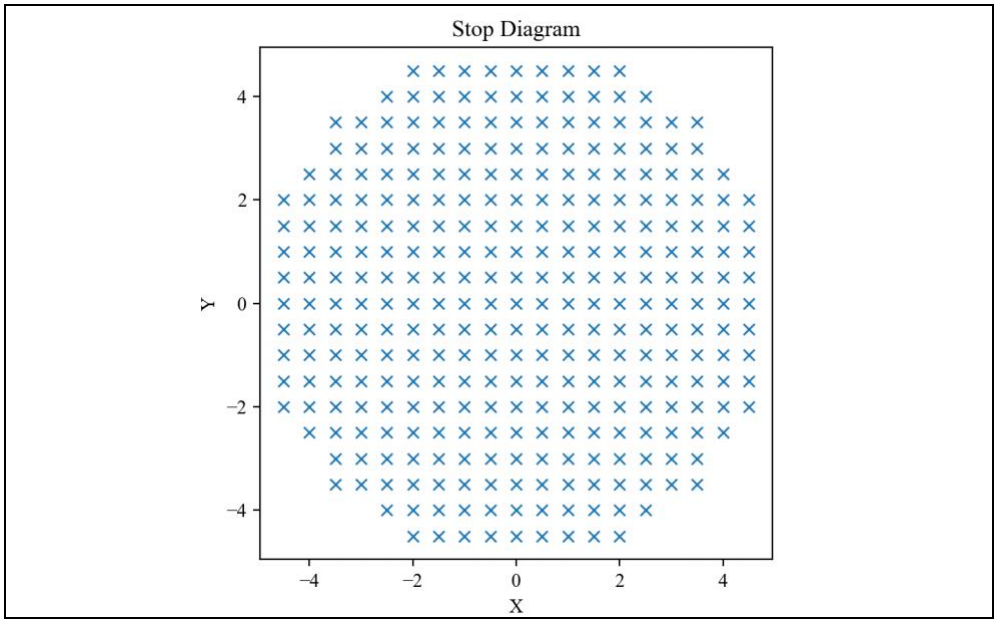


Figura 9. Example de lentes perfectas que no presentan aberraciones.

7.3 Example - Doublet Lens 3D color

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


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

L1a = Kos.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 = Kos.surf()
L1b.Rc = -3.071608670000159E+001
L1b.Thickness = 3.0
L1b.Glass = "F2"
L1b.Diameter = 30
L1b.Color = [.7, .4, .4]

L1c = Kos.surf()
L1c.Rc = -7.819730726078505E+001
L1c.Thickness = 9.737604742910693E+001
```

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

P_Ima = Kos.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]
configuracion_1 = Kos.Setup()

Doblete = Kos.system(A, configuracion_1)
Rayos = Kos.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
            Doblete.Trace(pSource_0, dCos, W)
            Rayos.push()
            W = 0.5
            Doblete.Trace(pSource_0, dCos, W)
            Rayos.push()
            W = 0.6
            Doblete.Trace(pSource_0, dCos, W)
            Rayos.push()

Kos.display3d(Doblete, Rayos, 1)
```

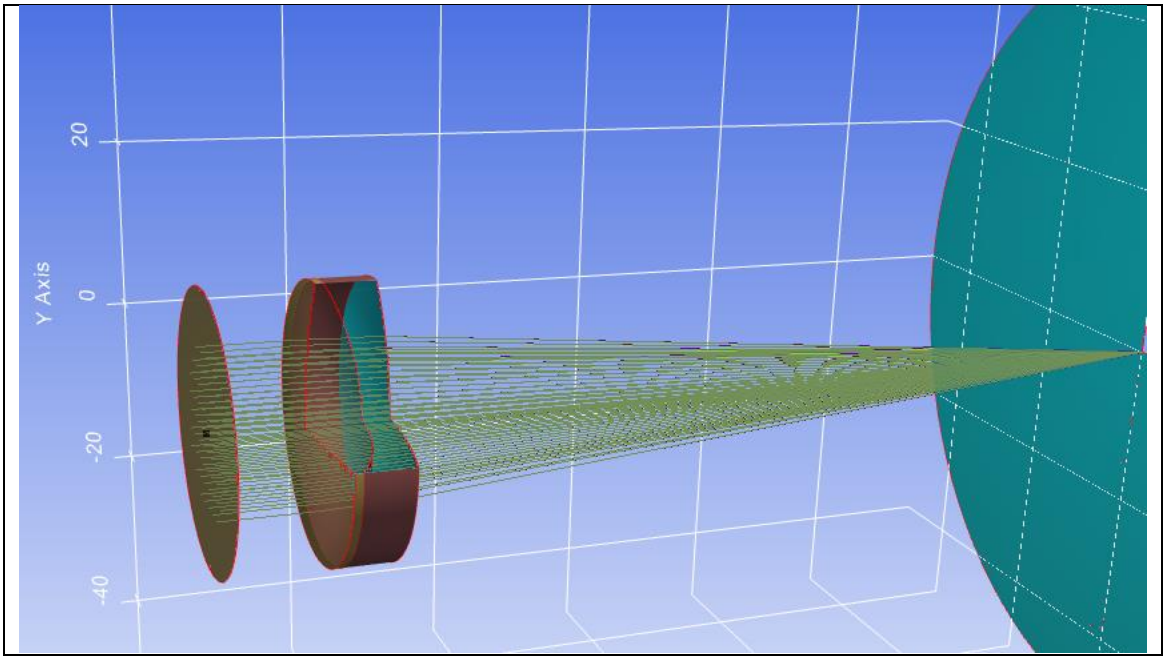


Figura 10. Vista de un doblete donde se ha definido un cambio en el color de la superficie.

7.4 Example - Doublet Lens Tilt

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


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

L1a = Kos.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 = Kos.surf()
L1b.Rc = (-3.071608670000159E+001)
L1b.Thickness = 3.0
L1b.Glass = "F2"
L1b.Diameter = 30

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

P_Ima = Kos.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]
configuracion_1 = Kos.Setup()

Doblete = Kos.system(A, configuracion_1)
Rayos = Kos.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
            Doblete.Trace(pSource_0, dCos, W)
            Rayos.push()
            W = 0.5
            Doblete.Trace(pSource_0, dCos, W)
            Rayos.push()
            W = 0.6
            Doblete.Trace(pSource_0, dCos, W)
            Rayos.push()

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

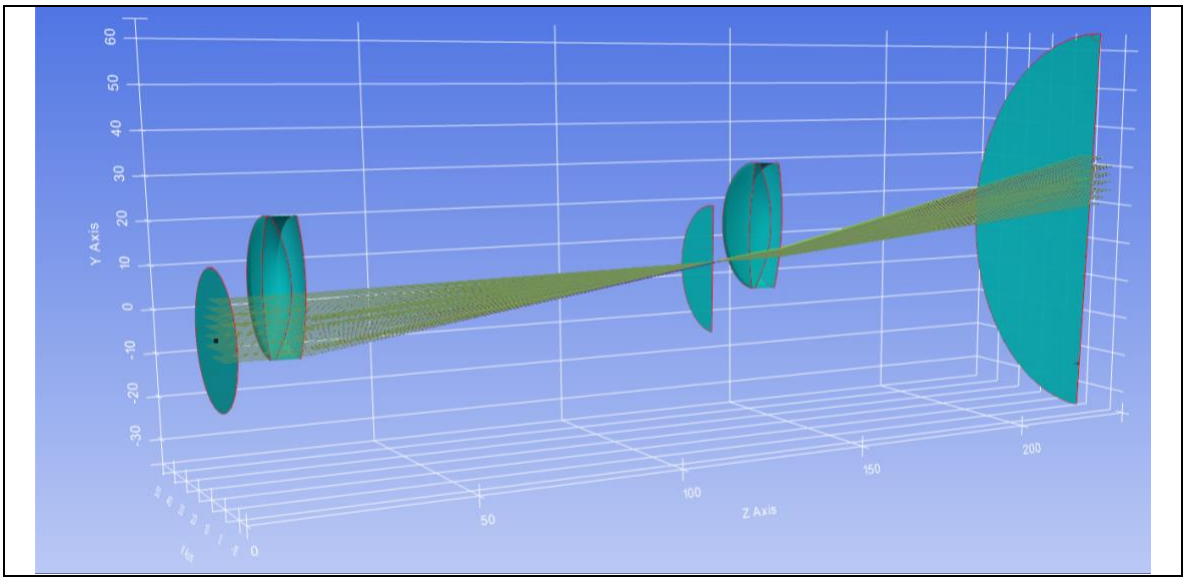


Figura 11. Visualización 3D de un sistema fuera de eje generado con repetición de superficies, la modificación de una superficie afecta a todas las partes del diseño en donde esta sea utilizada.

7.5 Example - Doublet Lens (Cálculos paraxiales)

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

start_time = time.time()


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

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

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

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

P_Ima = Kos.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 = Kos.Setup()

Doblete = Kos.system(A, config_1)
print("Calculando para cierta wave --------")
Prx = Doblete.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
Doblete.ResetData()
print("Calculando para cierta wave --------")
Prx = Doblete.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
Doblete.ResetData()
print("Calculando para cierta wave --------")
Prx = Doblete.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
Doblete.ResetData()
print("Calculando para cierta wave --------")
Prx = Doblete.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
```

Figura 12. Distancias focales resultantes para un incremento repetido de 1mm entre cálculos

7.6 Example - Doublet Lens Tilt Nulls

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


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

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

Null1_L1a = Kos.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 = Kos.surf()
Null2_L1a.Thickness = L1a.Thickness
Null2_L1a.Glass = "NULL"
Null2_L1a.Diameter = L1a.Diameter

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

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

P_Ima = Kos.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]
configuracion_1 = Kos.Setup()

Doblete = Kos.system(A, configuracion_1)
Rayos = Kos.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
            Doblete.Trace(pSource_0, dCos, W)
            Rayos.push()
            W = 0.5
            Doblete.Trace(pSource_0, dCos, W)
            Rayos.push()
            W = 0.6
            Doblete.Trace(pSource_0, dCos, W)
            Rayos.push()

Kos.display2d(Doblete, Rayos, 0)
```

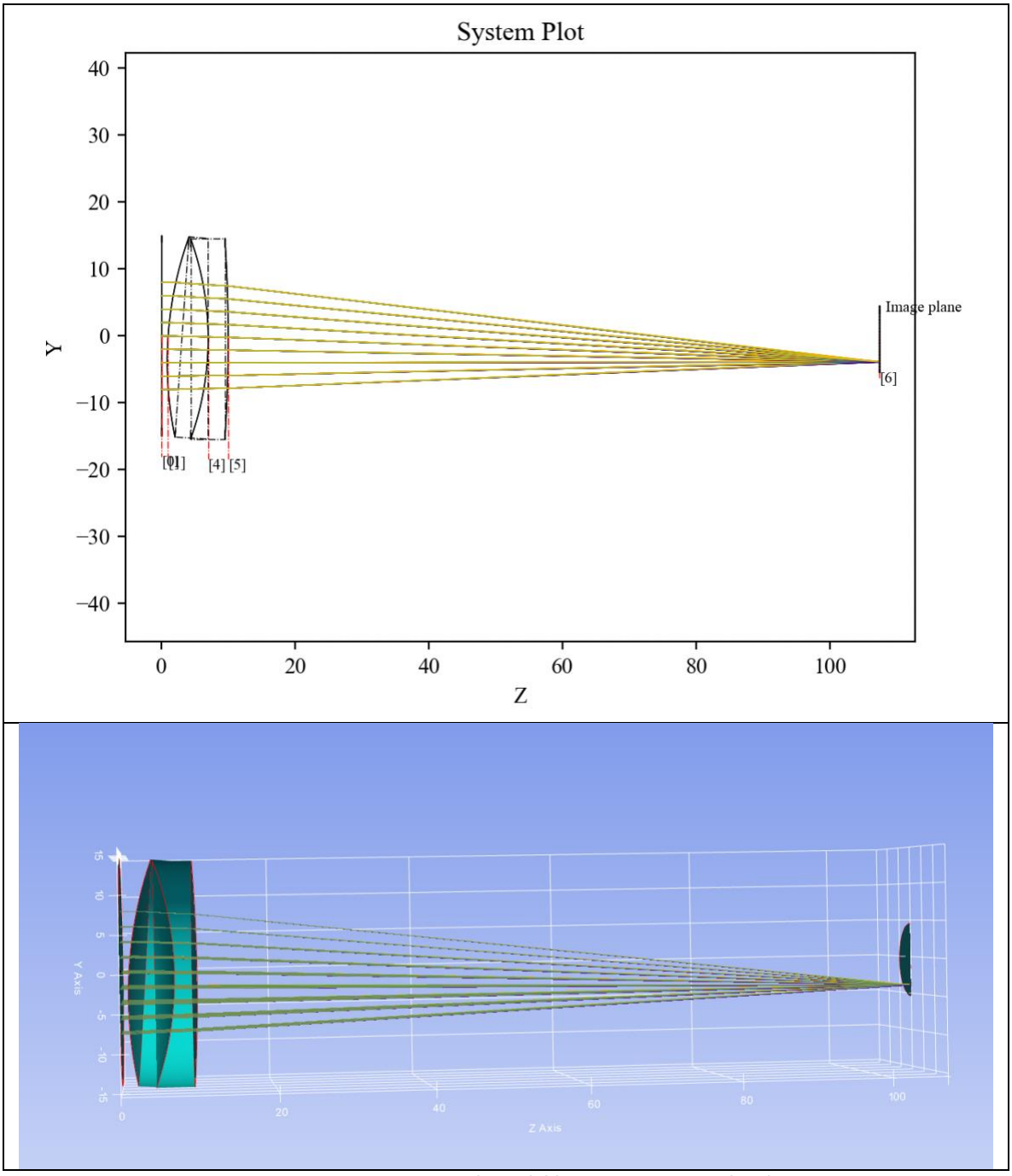


Figura 13. Vista 2D y 3D de un doblete con una cara inclinada.

7.7 Example - Doublet Lens NonSec

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

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

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

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

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

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

P_Ima = Kos.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]
configuracion_1 = Kos.Setup()

Doblete = Kos.system(A, configuracion_1)
Rayos = Kos.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
            Doblete.NsTrace(pSource_0, dCos, W)
            Rayos.push()


Kos.display2d(Doblete, Rayos, 0)
```

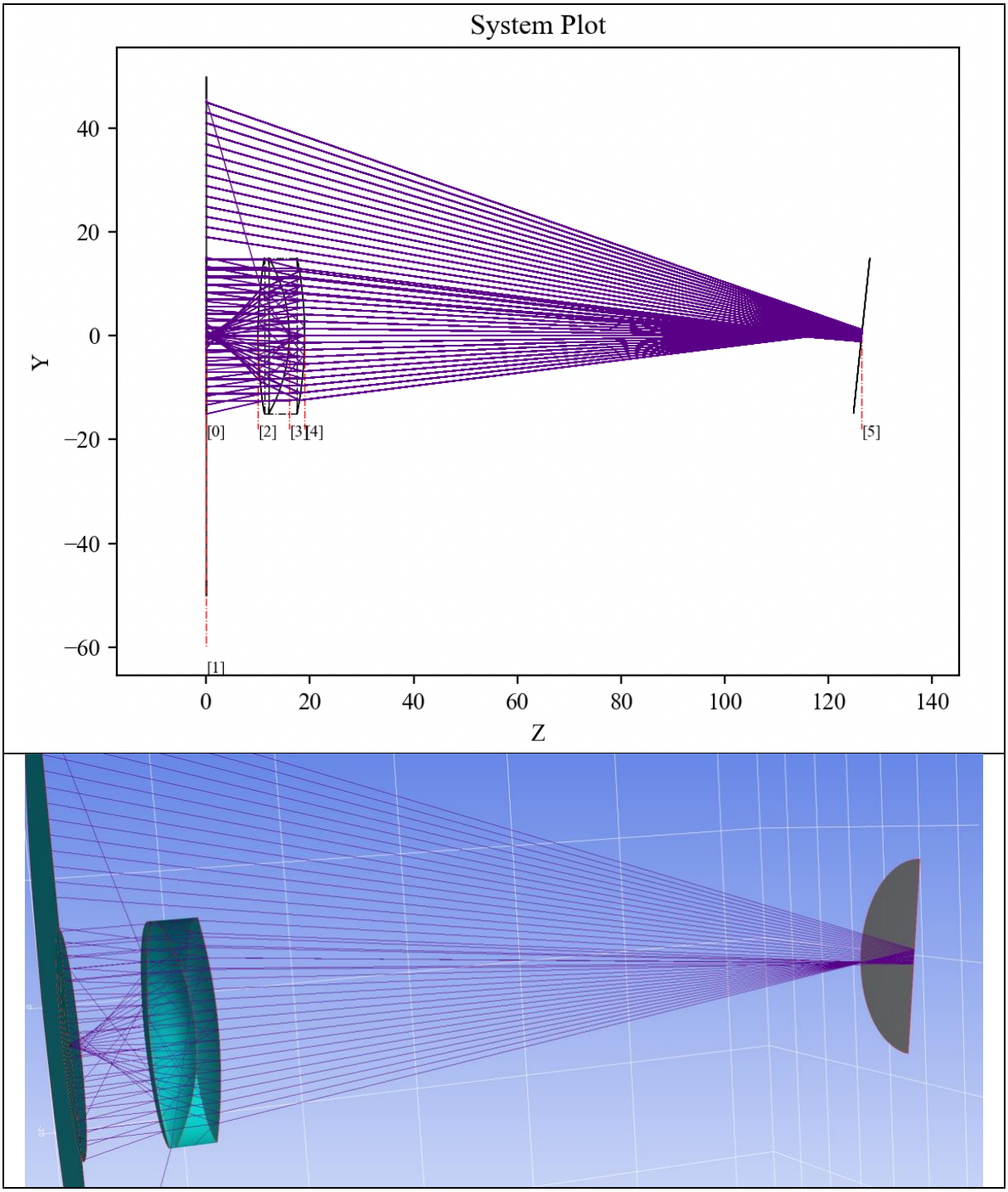


Figura 14. Visualización 2D y 3D de un trazado de rayos no secuencial, algunos rayos son reflejados de regreso dependiendo del valor en los coeficientes e Fresnel, estos dependen de la longitud de onda, el material y el ángulo del rayo con respecto a la normal de la superficie en el punto de intersección.

7.8 Example - Doublet Lens Zernike

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""Doublet Lens Zernike"""
import KrakenOS as Kos
import numpy as np
import matplotlib.pyplot as plt
import time

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

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

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

L1c = Kos.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 = Kos.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]
configuracion_1 = Kos.Setup()

Doblete = Kos.system(A, configuracion_1)
Rayos = Kos.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
            Doblete.Trace(pSource_0, dCos, W)
            Rayos.push()

Kos.display3d(Doblete, Rayos, 2)
Kos.display2d(Doblete, Rayos, 0)
```

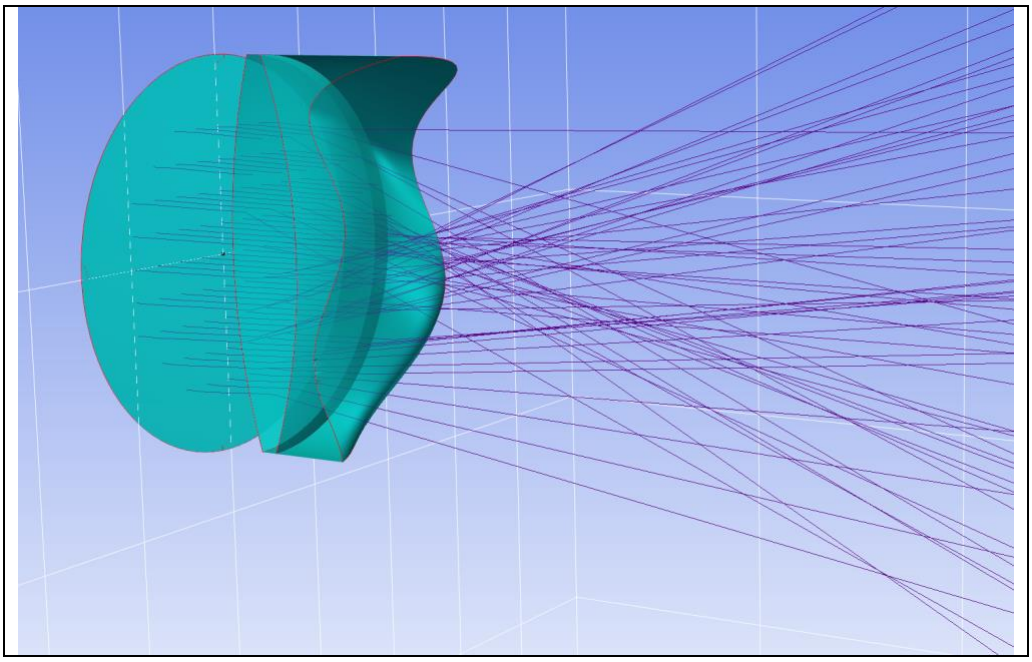


Figura 15. Example de lente con una superficie definida por coeficientes de los polinomios de Zernike.

7.9 Example - Doublet Lens Tilt nonSec

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


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

L1a = Kos.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 = Kos.surf()
L1b.Rc = (-3.071608670000159E+001)
L1b.Thickness = 3.0
L1b.Glass = "F2"
L1b.Diameter = 30

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

P_Ima = Kos.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]
configuracion_1 = Kos.Setup()

Doblete = Kos.system(A, configuracion_1)
Rayos = Kos.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
        Doblete.NsTrace(pSource_0, dCos, W)
        Rayos.push()
        W = 0.5
        Doblete.NsTrace(pSource_0, dCos, W)
        Rayos.push()
        W = 0.6
        Doblete.NsTrace(pSource_0, dCos, W)
        Rayos.push()

Kos.display3d(Doblete, Rayos, 2)
```

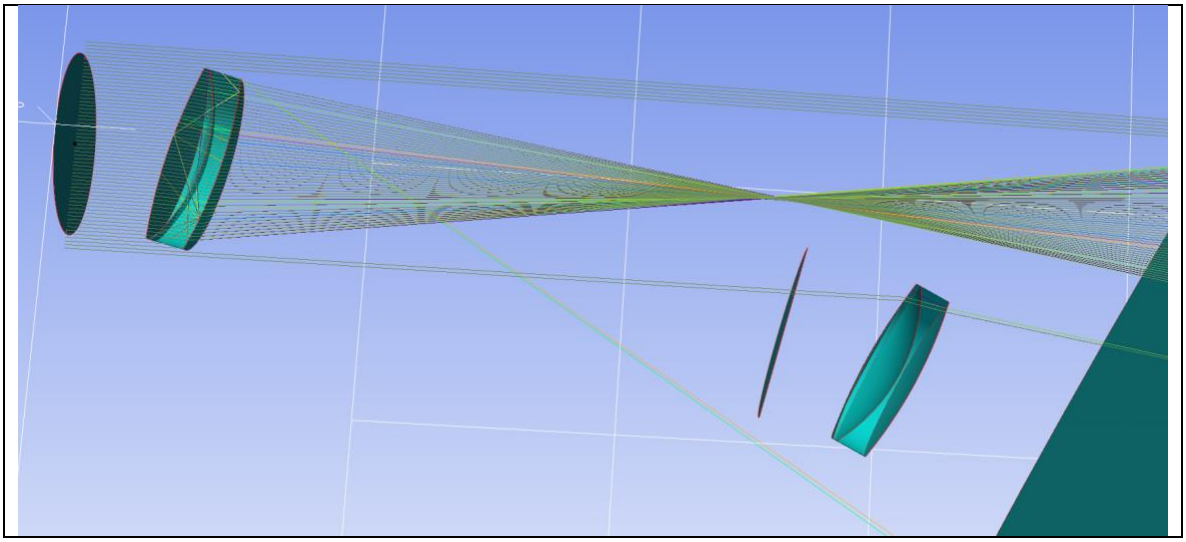


Figura 16. Example de trazado de rayos no secuencial, los rayos que no tocan al primer elemento son tazados hasta un segundo elemento donde si tocan la superficie.

7.10 Example - Doublet Lens Pupil

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
Examp Doublet Lens Pupil"""
import KrakenOS as Kos

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

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

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

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

P_Ima = Kos.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 = Kos.Setup()

Doblete = Kos.system(A, config_1)
Rayos = Kos.raykeeper(Doblete)

W = 0.4
Surf= 4
AperVal = 10
AperType = "EPD"
Pup = Kos.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]]
    Doblete.Trace(pSource_0, dCos, W)
    Rayos.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]]
    Doblete.Trace(pSource_0, dCos, W)
    Rayos.push()

Kos.display2d(Doblete, Rayos, 0)
```

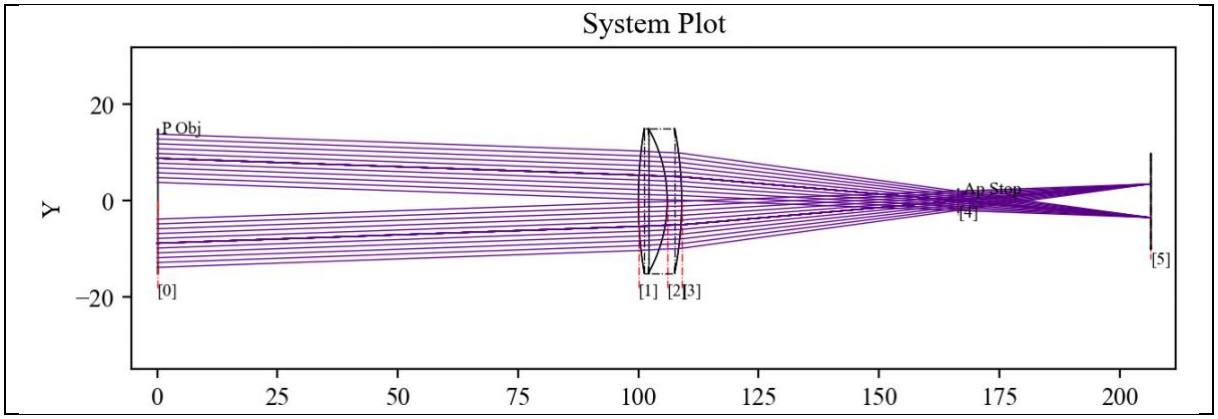


Figura 17. Example de haces de rayos generados con la función de pupila, en esta se define el lugar en donde se encuentra la pupila o la apertura del sistema.

7.11 Example - Doublet Lens Commands System

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


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

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

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

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

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

P_Ima = Kos.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]
configuracion_1 = Kos.Setup()

Doblete = Kos.system(A, configuracion_1)
Rayos = Kos.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
Doblete.Trace(pSource_0, dCos, W)
Rayos.push()

Kos.display3d(Doblete, Rayos, 2)

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

7.12 Example - Doublet Lens Pupil Seidel

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


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

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

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

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

P_Ima = Kos.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 = Kos.Setup()

Doblete = Kos.system(A, config_1)

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

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

    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()

    Kos.display2d(Doblete, Rayos, 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]
------------------------------------
```

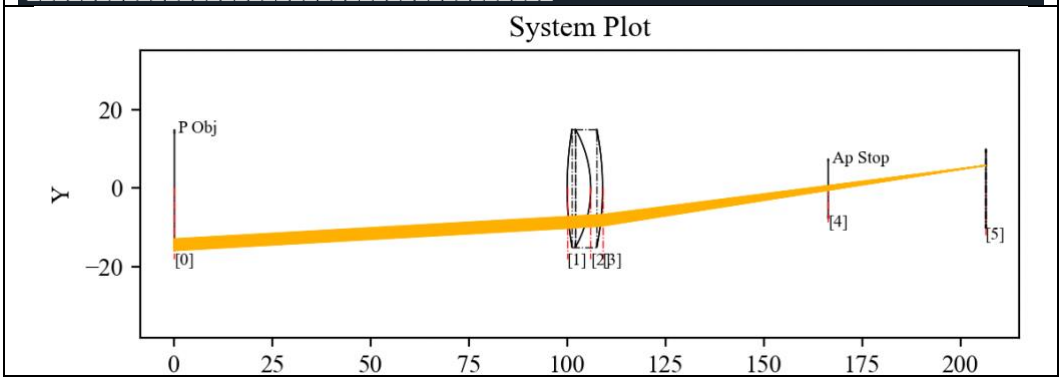
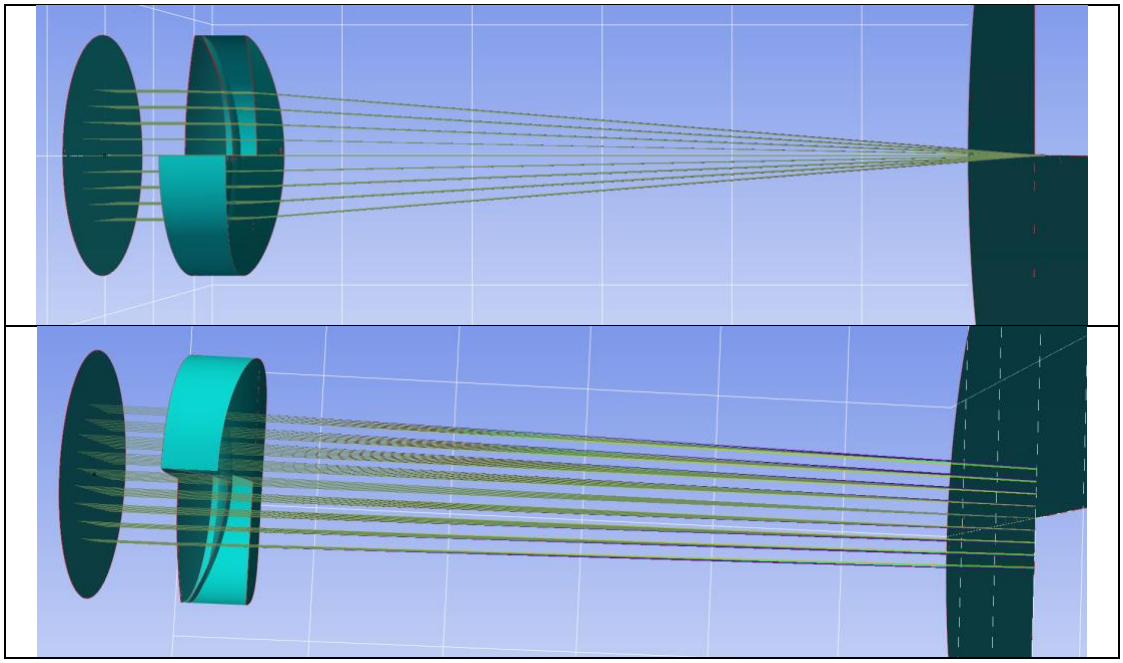


Figura 18. (Arriba) Valores de las sumas de Seidel. (Abajo) Doblete con el que se calcularon las aberraciones utilizando ese campo delimitado por la imagen de la pupila.

7.13 Example - Doublet Lens Cylinder

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


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

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

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

L1c = Kos.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 = Kos.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]
configuracion_1 = Kos.Setup()

Doblete = Kos.system(A, configuracion_1)
Rayos = Kos.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
                Doblete.Trace(pSource_0, dCos, W)
                Rayos.push()
                W = 0.5
                Doblete.Trace(pSource_0, dCos, W)
                Rayos.push()
                W = 0.6
                Doblete.Trace(pSource_0, dCos, W)
                Rayos.push()

Kos.display3d(Doblete, Rayos, 1)
```



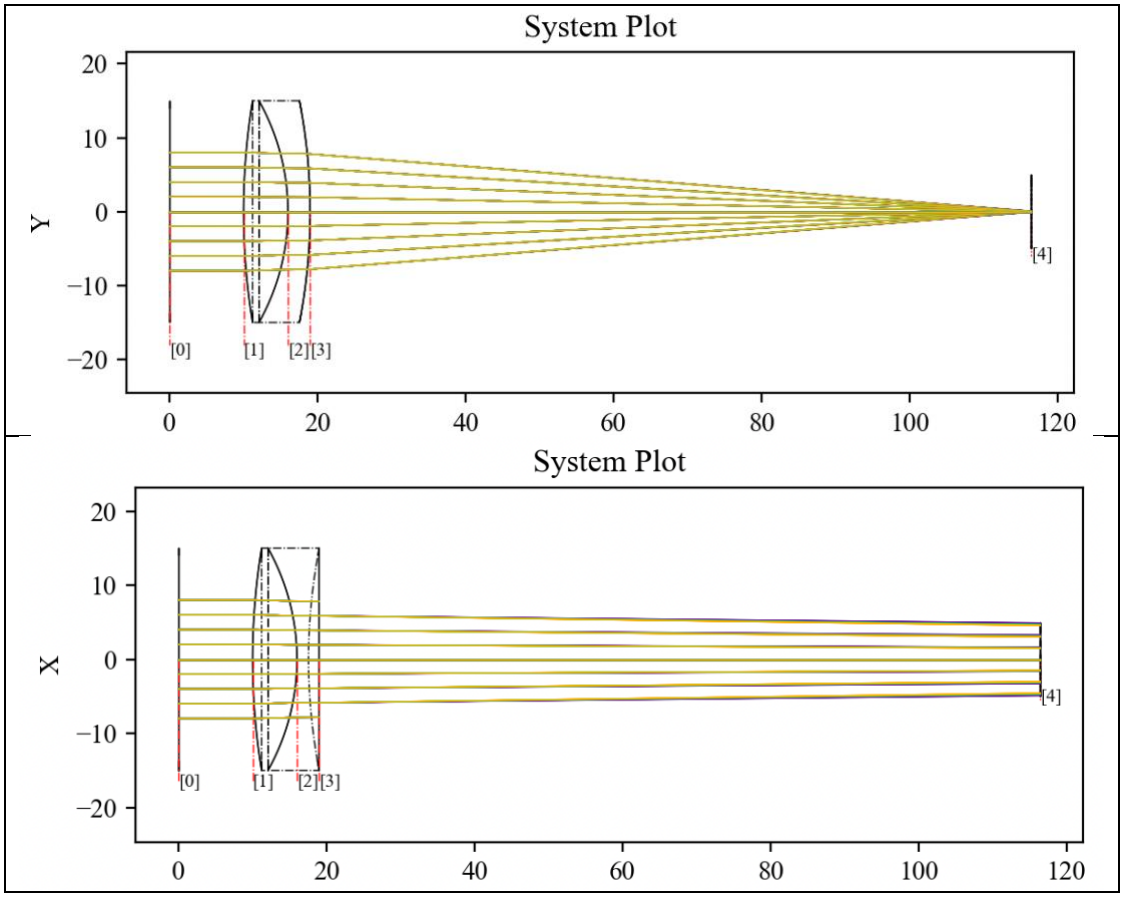


Figura 19. Visualización 2D y 3D vistos en la dirección del eje X y posteriormente del eje Y. La ultima cara de la lente tiene un radio de curvatura que puede verse desde el eje Y, esta es plana al verla desde el eje X.

7.14 Example - Axicon

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

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

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

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

```
L1c.Diameter = 30

P_Ima = Kos.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"

configuracion_1 = Kos.Setup()
A = [P_Obj, L1a, L1c, P_Ima]

Doblete = Kos.system(A, configuracion_1)
Rayos = Kos.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
            Doblete.Trace(pSource_0, dCos, W)
            Rayos.push()
            W = 0.5
            Doblete.Trace(pSource_0, dCos, W)
            Rayos.push()
            W = 0.6
            Doblete.Trace(pSource_0, dCos, W)
            Rayos.push()

Kos.display3d(Doblete, Rayos, 2)
```

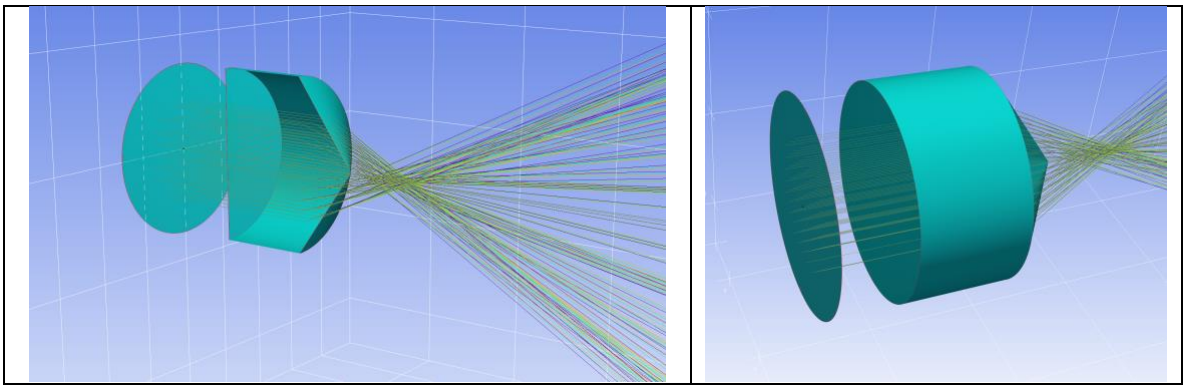


Figura 20. Vista 3D de un Axicon, vista en sección transversal y completa.

7.15 Example - Axicon and Cylinder

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

configuracion_1 = Kos.Setup()

P_Obj = Kos.surf()
P_Obj.Rc = 0.0
```

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

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


L1c = Kos.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 = Kos.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]

Doblete = Kos.system(A, configuracion_1)
Rayos = Kos.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
            Doblete.Trace(pSource_0, dCos, W)
            Rayos.push()
            W = 0.5
            Doblete.Trace(pSource_0, dCos, W)
            Rayos.push()
            W = 0.6
            Doblete.Trace(pSource_0, dCos, W)
            Rayos.push()

Kos.display3d(Doblete, Rayos, 0)
```

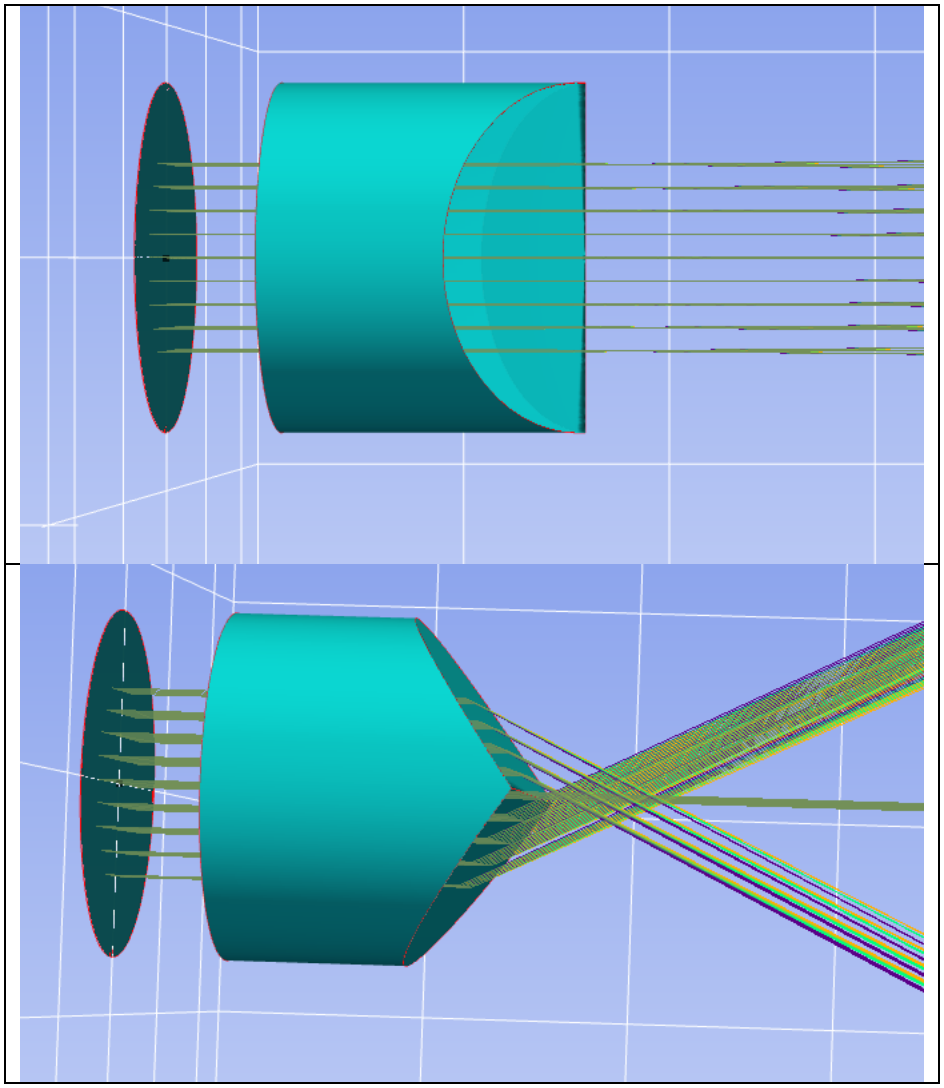


Figura 21. Example de lente cilíndrica combinada con axicon.

7.16 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 KrakenOS as Kos

start_time = time.time()


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

L1a = Kos.surf()
L1a.Rc = 9.284706570002484E+001
L1a.Thickness = 6.0
```

```
L1a.Glass = "BK7"
L1a.Diameter = 30.0
L1a.Axicon = 0

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

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

Esp90 = Kos.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 = Kos.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 = Kos.Setup()

Doblete = Kos.system(A, config_1)
Rayos1 = Kos.raykeeper(Doblete)
Rayos2 = Kos.raykeeper(Doblete)
Rayos3 = Kos.raykeeper(Doblete)
RayosT = Kos.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
            Doblete.Trace(pSource_0, dCos, W)
            Rayos1.push()
            RayosT.push()
            W=0.5
            Doblete.Trace(pSource_0, dCos,W)
            Rayos2.push()
            RayosT.push()
            W=0.6
            Doblete.Trace(pSource_0, dCos,W)
            Rayos3.push()
            RayosT.push()

Kos.display3d(Doblete, RayosT, 2)

X, Y, Z, L, M, N = Rayos1.pick(-1)
plt.plot(X, Z, 'x')
X, Y, Z, L, M, N = Rayos2.pick(-1)
plt.plot(X, Z, 'x')
X, Y, Z, L, M, N = Rayos3.pick(-1)
plt.plot(X, Z, 'x')
plt.xlabel('numbers')
```

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

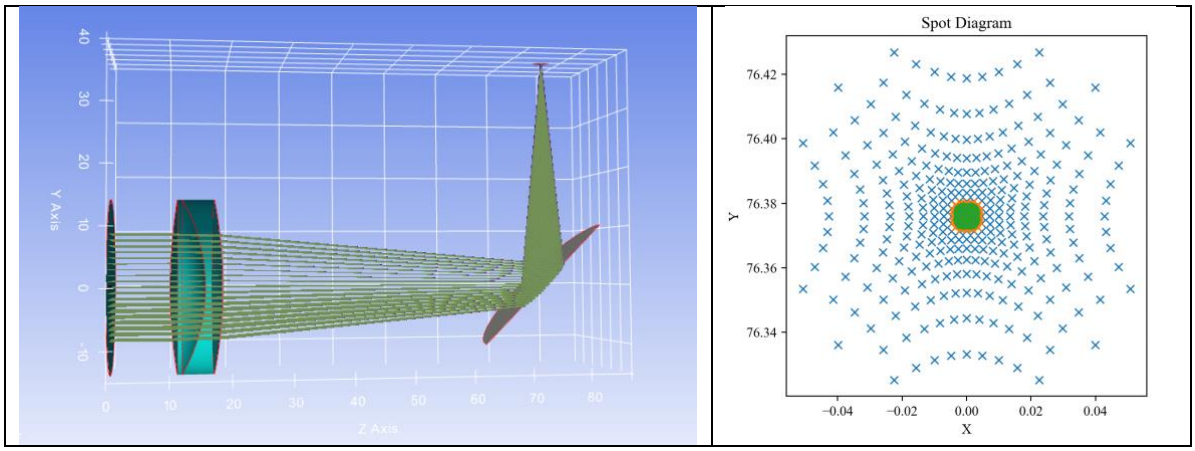


Figura 22. (Izquierda) Example de espejo diagonal con una rotación de 45°, (Derecha) Diagrama de manchas para tres distintas longitudes de onda, 0.4 µm, 0.5 µm y 0.6µm.

7.17 Example - Parabole Mirror Shift

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


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

M1 = Kos.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 = Kos.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]
configuracion_1 = Kos.Setup()

Espejo = Kos.system(A, configuracion_1)
Rayos = Kos.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)
            Rayos.push()

Kos.display3d(Espejo, Rayos, 0)
```

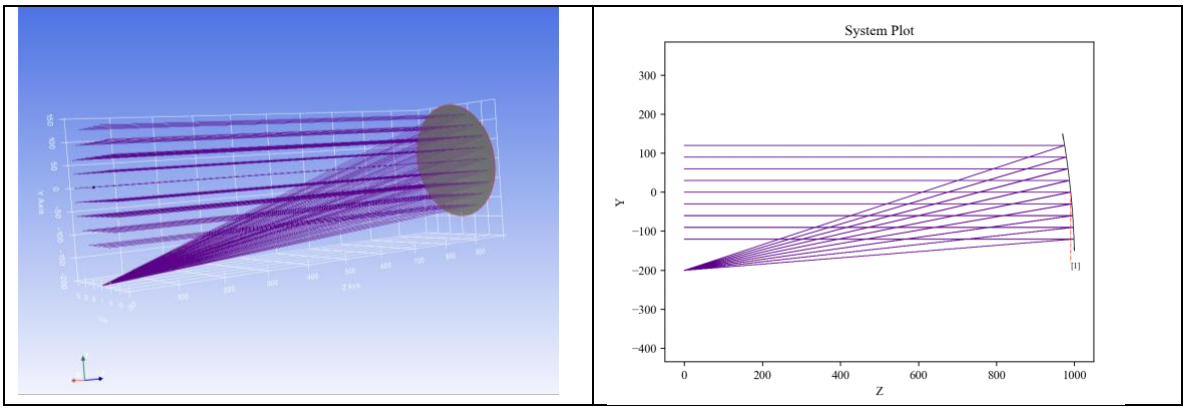


Figura 23. Vista 3D y 2D de una parábola fuera de eje.

7.18 Example - Diffraction Grating Transmission

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

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

Dif_Obj_c1 = Kos.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 = Kos.surf()
Dif_Obj_c2.Rc = 0.0
Dif_Obj_c2.Thickness = 10
Dif_Obj_c2.Glass = "AIR"
Dif_Obj_c2.Diameter = 30.0

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

L1b = Kos.surf()
L1b.Rc = -4.408716526030626E+001
L1b.Thickness = 3.0
```

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

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

P_Ima = Kos.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]
configuracion_1 = Kos.Setup()

Doblete = Kos.system(A, configuracion_1)
Rayos = Kos.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
            Doblete.Trace(pSource_0, dCos, W)
            Rayos.push()
            W = 0.5
            Doblete.Trace(pSource_0, dCos, W)
            Rayos.push()
            W = 0.6
            Doblete.Trace(pSource_0, dCos, W)
            Rayos.push()


Kos.display2d(Doblete, Rayos, 0)
```

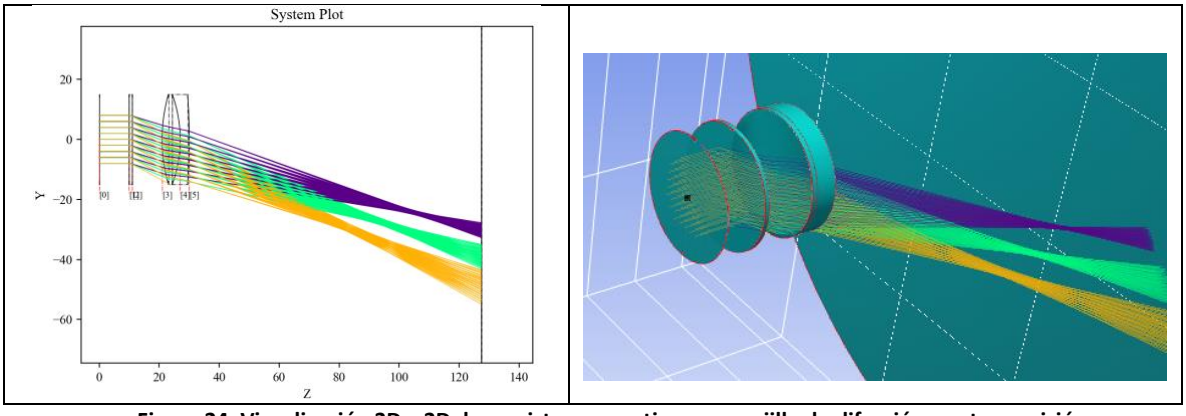


Figura 24. Visualización 3D y 2D de un sistema que tiene una rejilla de difacción por transmisión.

7.19 Example - Diffraction Grating Reflection

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

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

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

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

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

Dif_Obj = Kos.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 = Kos.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]
configuracion_1 = Kos.Setup()

Doblete = Kos.system(A, configuracion_1)
Rayos = Kos.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
            Doblete.Trace(pSource_0, dCos, W)
            Rayos.push()
            W = 0.5
            Doblete.Trace(pSource_0, dCos, W)
            Rayos.push()
            W = 0.6
            Doblete.Trace(pSource_0, dCos, W)
            Rayos.push()
```

```
Kos.display2d(Doblete, Rayos, 1)
```

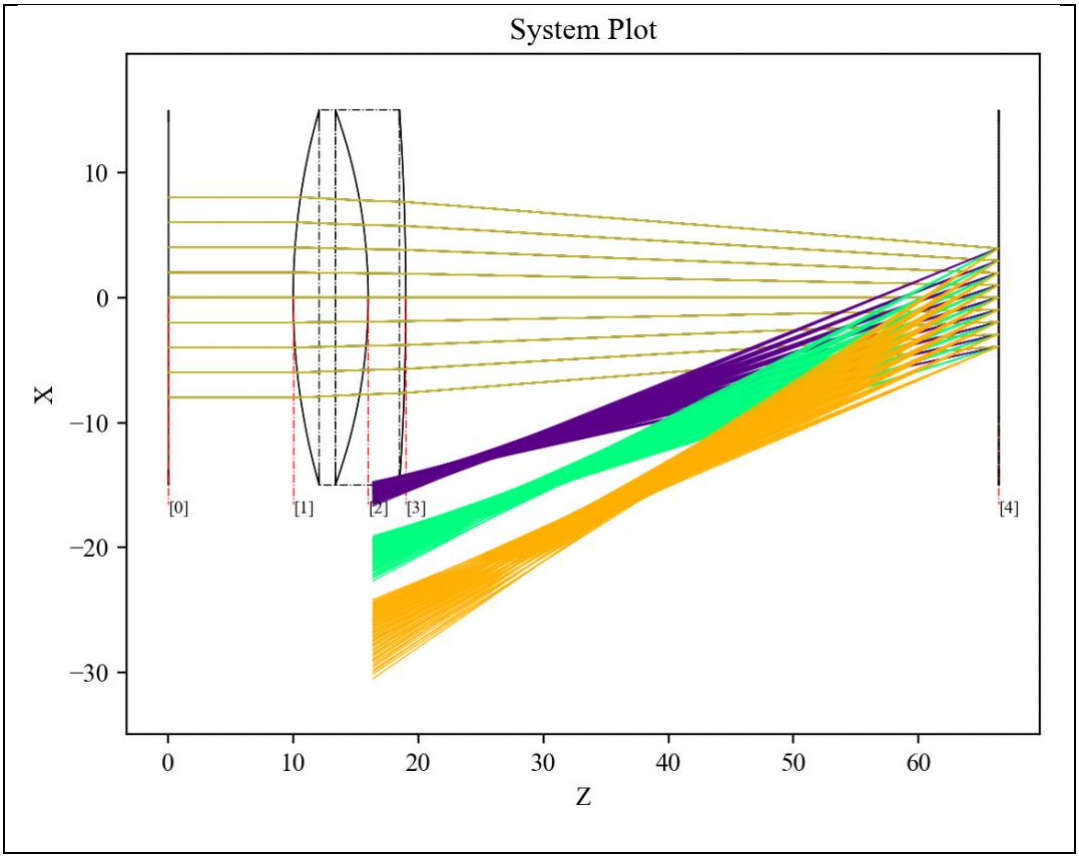


Figura 25. Vista 2D de un sistema con rejilla de difracción por reflexión.

7.20 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 KrakenOS as Kos

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

M1 = Kos.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 = Kos.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 = Kos.surf()
Vertex.Thickness = 130.0
Vertex.Glass = "AIR"
Vertex.Diameter = 600.0
Vertex.Drawing = 0

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

objeto = Kos.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 = Kos.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]
configuracion_1 = Kos.Setup()

Telescope = Kos.system(A, configuracion_1)
Rays = Kos.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()

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

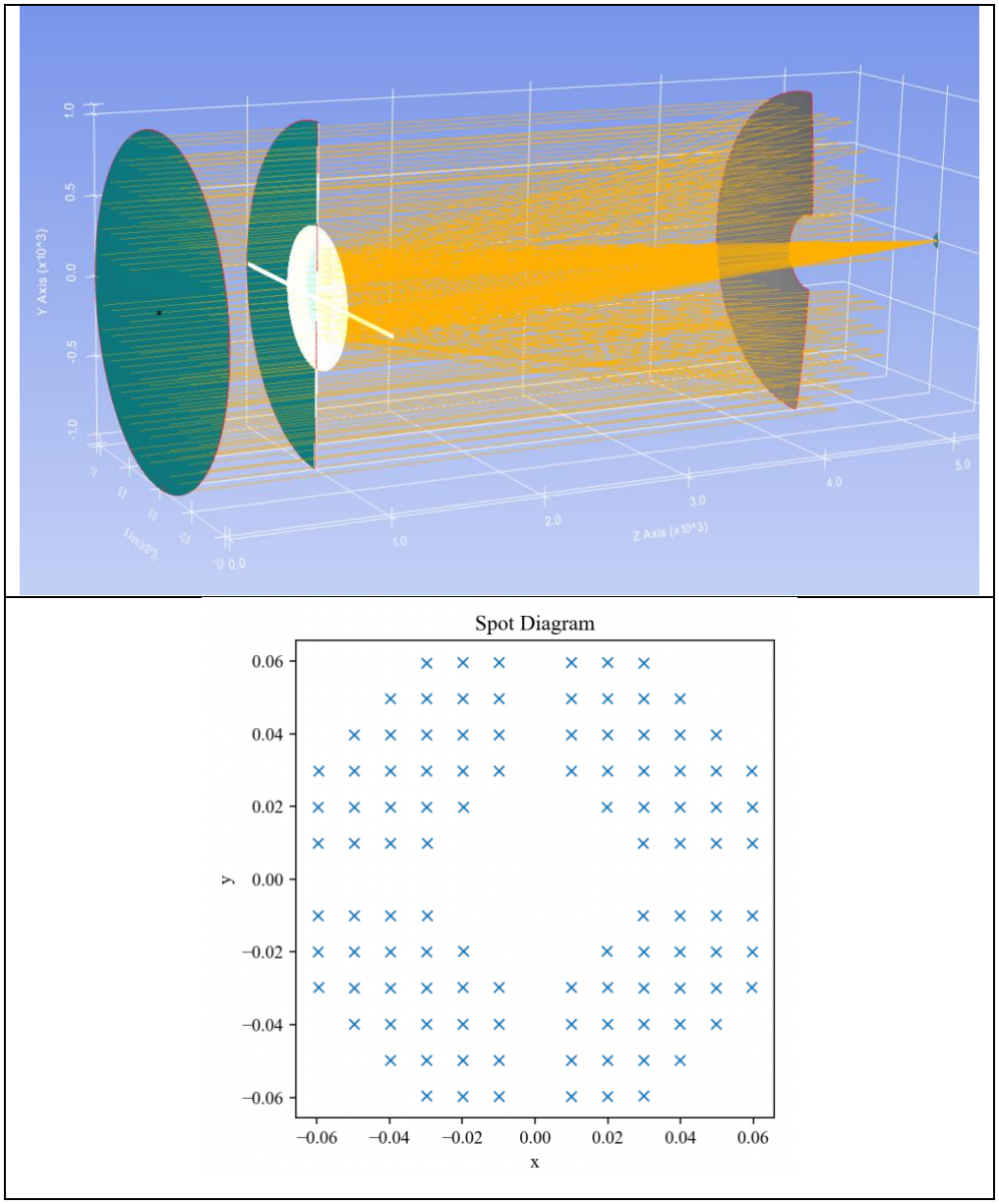


Figura 26. (Superior) Vista del telescopio con la sombra de la araña, (Inferior) diagrama de manchas con un patrón rectangular, se puede notar la falta de los rayos que fueron obstruidos por la araña.

7.21 Example - Tel 2M Spyder Spot Tilt M2

```
#            Rays.push()
#Kos.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 KrakenOS as Kos

P_Obj = Kos.surf()
P_Obj.Rc = 0
P_Obj.Thickness = 1000
```

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

    Spider = Kos.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 = Kos.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 = Kos.surf()
    M2.Rc = -3.93E+003
    M2.Thickness = Thickness + 1.037535322418897E+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 = Kos.surf()
    P_Ima.Diameter = 100.0
    P_Ima.Glass = "AIR"
    P_Ima.Name = "Plano imagen"

    A = [P_Obj, Spider, M1, M2, P_Ima]
    configuracion_1 = Kos.Setup()

    Telescopio = Kos.system(A, configuracion_1)
    Rayos = Kos.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)
                Rayos.push()

    Kos.display3d(Telescopio, Rayos, 2)
    X, Y, Z, L, M, N = Rayos.pick(-1)
```

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

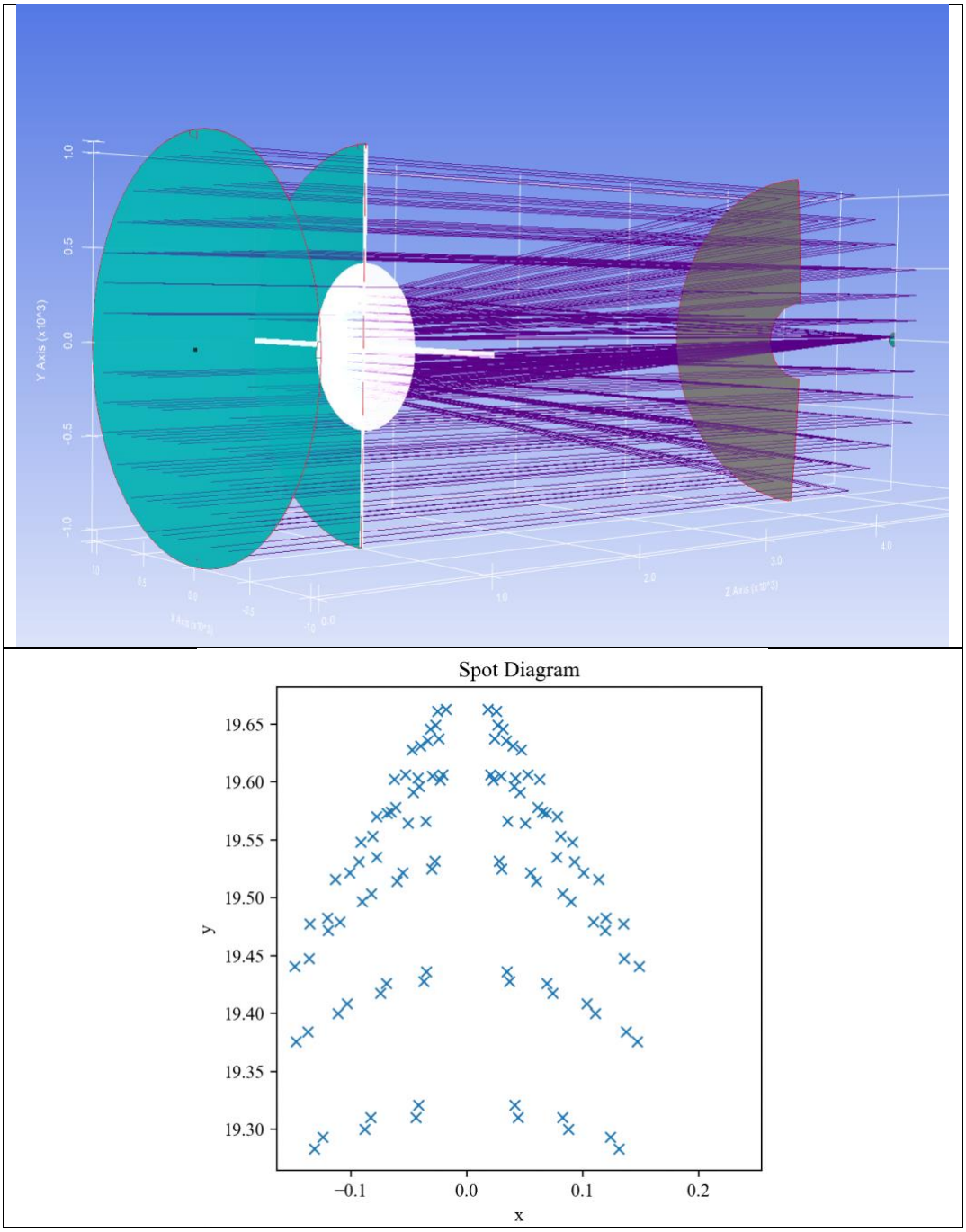


Figura 27. Vista del telescopio con araña y un diagrama de manchas que muestra aberración de coma por el hecho de inclinar el espejo secundario.

7.22 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 KrakenOS as Kos

P_Obj = Kos.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 = Kos.surf()
M1.Rc = -9.6380000000004009E+003
M1.Thickness = -Thickness
M1.k = -1.0773100000000000E+000
M1.Glass = "MIRROR"
M1.Diameter = 1.059E+003 * 2.0
M1.InDiameter = 250 * 2.0

M2 = Kos.surf()
M2.Rc = -3.93E+003
M2.Thickness = Thickness + 1.0375258801250084E+003
M2.k = -4.3281000000000000E+000
M2.Glass = "MIRROR"
M2.Diameter = 3.365E+002 * 2.0
M2.TiltY = 0.1
M2.TiltX = 0.1
M2.AxisMove = 0

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

A = [P_Obj, M1, M2, P_Ima]
configuracion_1 = Kos.Setup()
Telescopio = Kos.system(A, configuracion_1)

W = 0.4
Surf= 1
AperVal = 2010
AperType = "EPD" # "STOP"
Pup = Kos.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()
Rayos = Kos.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)
    Rayos.push()

Kos.display3d(Telescopio, Rayos, 2)

X, Y, Z, L, M, N = Rayos.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]
```

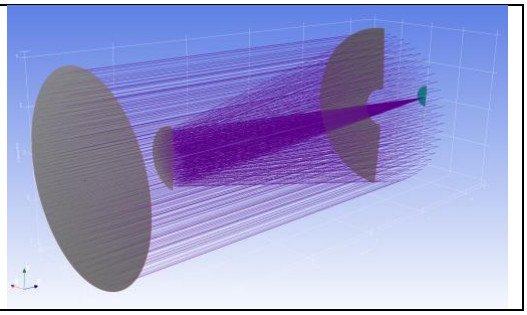


Figura 28. (Izquierda) Obtención de los datos de la pupila. (Derecha) Visualización 3D del telescopio.

7.23 Example - Tel 2M Error Map

```
# -*- coding: utf-8 -*-
Examp Tel 2M Error Map"""
import matplotlib.pyplot as plt
import numpy as np
import KrakenOS as Kos
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 = Kos.surf()
P_Obj.Rc = 0
P_Obj.Thickness = 3500
P_Obj.Glass = "AIR"
P_Obj.Diameter = 1.059E+003 * 2.0

Thickness = 3.4522000000000000E+003
M1 = Kos.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 = Kos.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 = Kos.surf()
P_Ima.Diameter = 1000.0
P_Ima.Glass = "AIR"
P_Ima.Name = "Plano imagen"

A = [P_Obj, M1, M2, P_Ima]
configuracion_1 = Kos.Setup()

Telescopio = Kos.system(A, configuracion_1)
Rayos = Kos.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)
            Rayos.push()

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

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

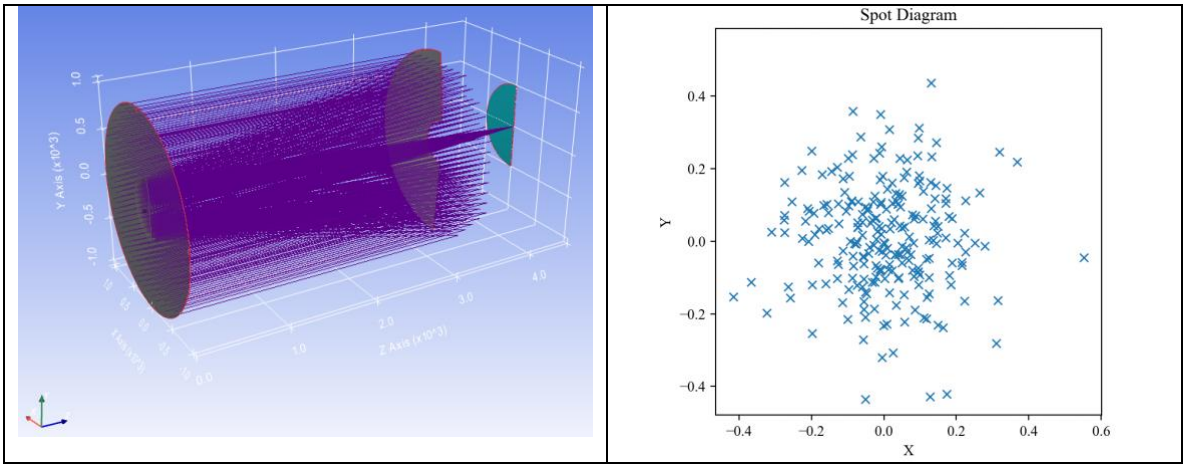


Figura 29. (Izquierda) Visualización 3D de un telescopio generado con un mapa de deformaciones sumado a la función de forma del espejo primario. (Derecha) Diagrama de manchas originado con este sistema.

7.24 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 KrakenOS as Kos
from PhaseCalc import Phase

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

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

    A = [P_Obj, M1, M2, P_Ima]
    configuracion_1 = Kos.Setup()
    Telescopio = Kos.system(A, configuracion_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, configuracion_1, Samp, Ptype, FieldY,
FieldX, FieldType)
    NC = 38
    A = np.ones(NC)

    z_coeff, MatNotation, w_rms, fitt_error = Kos.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 = Kos.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 = Kos.raykeeper(Telescopio)
    Pup = Kos.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()

    Kos.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
```

Figura 30. Valor de los coeficientes de los polinomios de Zernike ajustados al frente de onda del sistema para un campo dado. El resultado también incluye la expresión matemática del polinomio.

7.25 Example – Tel 2M-STL_ImageSlicer.py

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

    P_Obj=Kos.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=Kos.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=Kos.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=Kos.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]



configuracion_1=Kos.Setup()
Telescopio=Kos.system(A,configuracion_1)
Rayos=Kos.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 = Kos.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
    # Rayos.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=Kos.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=Kos.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=Kos.surf()
    P_Ima.Diameter=10.0
    P_Ima.Glass="AIR"
    P_Ima.Name="Plano imagen"




    A=[P_Obj, P_ImageSlicer, P_Ima]
    configuracion_1 = Kos.Setup()
    ImageSlicer = Kos.system(A,configuracion_1)
    Rayos=Kos.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
            Rayos.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()
    Kos.display3d(ImageSlicer, Rayos, 0)
```

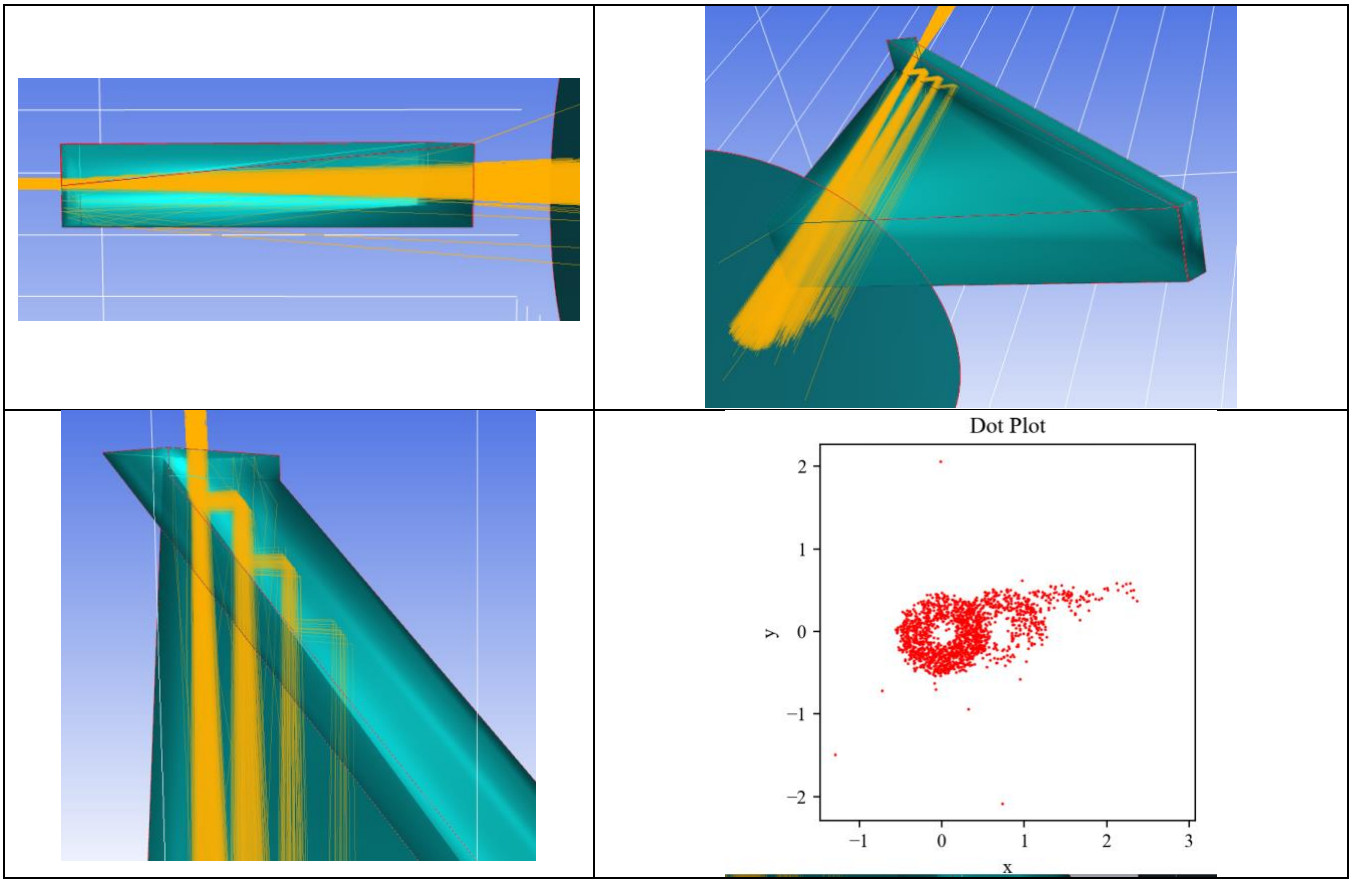


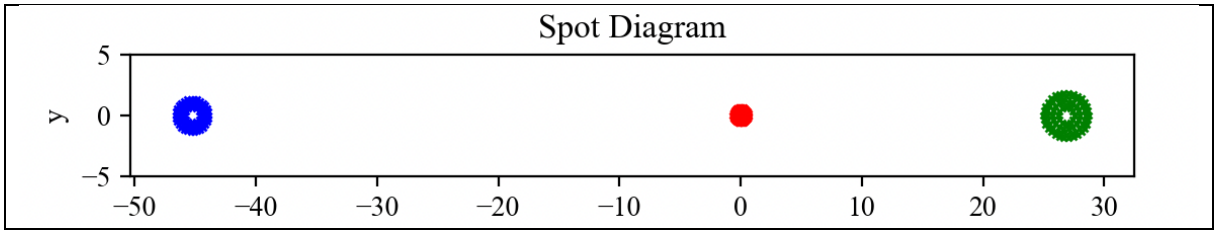
Figura 31. Example de Image Slicer Bowel Walraven a partir de un modelo STL generado en OpenScad.

7.26 Example – 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 KrakenOS as Kos
import numpy as np
import matplotlib.pyplot as plt
import time


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

configuracion_1=Kos.Setup()
Telescopio=Kos.system(A,configuracion_1)
Rayos1=Kos.raykeeper(Telescopio)
Rayos2=Kos.raykeeper(Telescopio)
Rayos3=Kos.raykeeper(Telescopio)

W = 0.4
Surf= 1
AperVal = 2010
AperType = "EPD" # "STOP"
Pup = Kos.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)
    Rayos1.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)
    Rayos2.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)
    Rayos3.push()


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

# Kos.display3d(Telescopio,Rayos,2)
X,Y,Z,L,M,N=Rayos1.pick(-1)
plt.plot(X*1000.0,Y*1000.0, 'x', c="b")
X,Y,Z,L,M,N=Rayos2.pick(-1)
plt.plot(X*1000.0,Y*1000.0, 'x', c="r")
X,Y,Z,L,M,N=Rayos3.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()
```



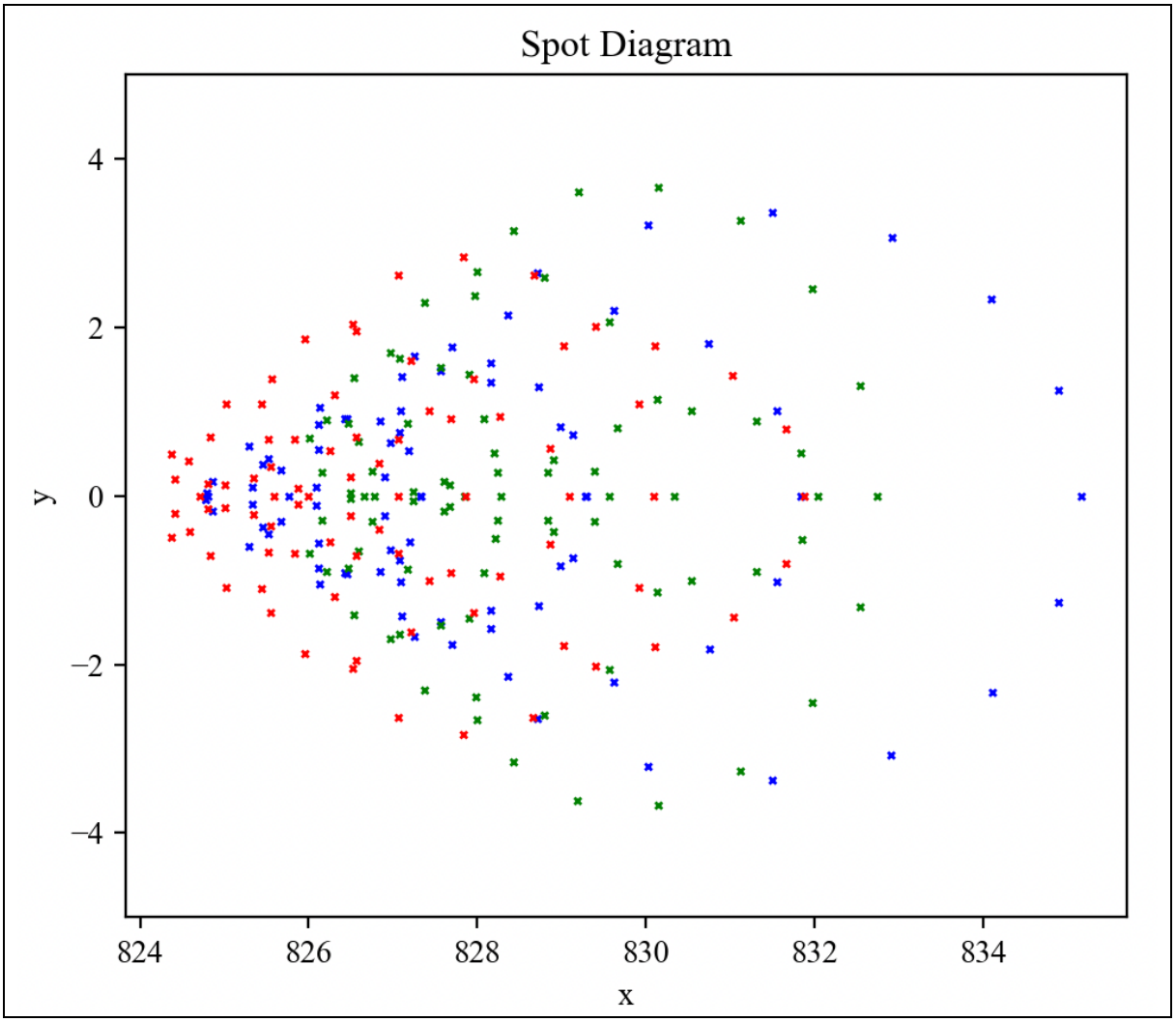


Figura 32. (Arriba) Example de el efecto de la atmosfera en tres longitudes de onda distintas y la implementación de un corrector de dispersión atmosférica (Abajo).

7.27 Example - Extra Shape Micro Lens Array

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

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

```
P_Obj.Diameter = 30.0

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

L1c = Kos.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 = Kos.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 = Kos.Setup()

Lens = Kos.system(A, Config_1)
Rays = Kos.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()


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

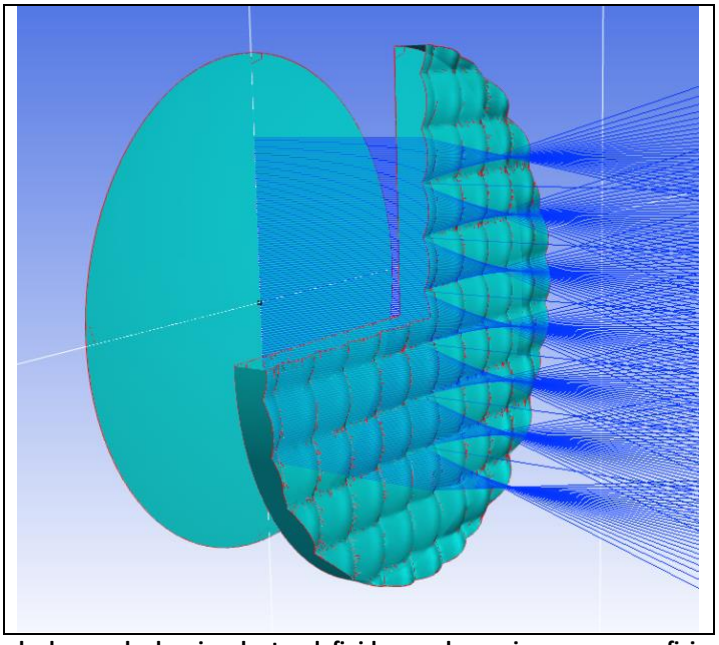


Figura 33. Example de arreglo de micro lentes definida por el usuario en una superficie del tipo ExtraShape.

7.28 Example - Extra Shape Radial Sine

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

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

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

L1c = Kos.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 = Kos.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 = Kos.Setup()

Lens = Kos.system(A, Config_1)
Rays = Kos.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()


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

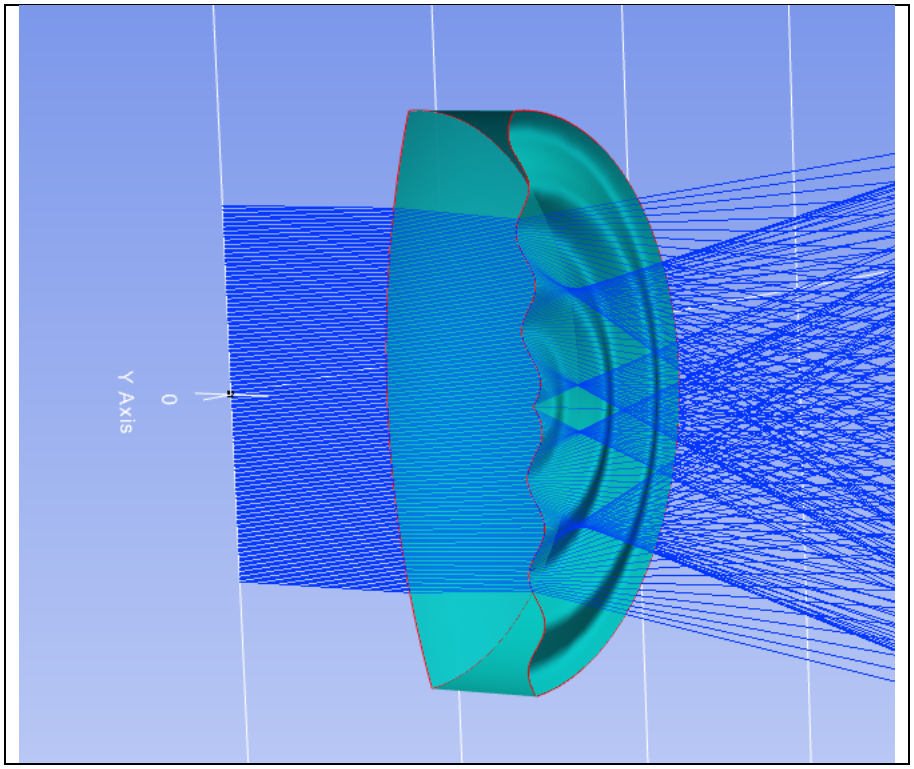


Figura 34. Vista de una lente con una superficie definida por el usuario por medio de una función radial.

7.29 Example - Extra Shape XY Cosines

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

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

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

L1c = Kos.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 = Kos.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 = Kos.Setup()

Lens = Kos.system(A, Config_1)
Rays = Kos.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()


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

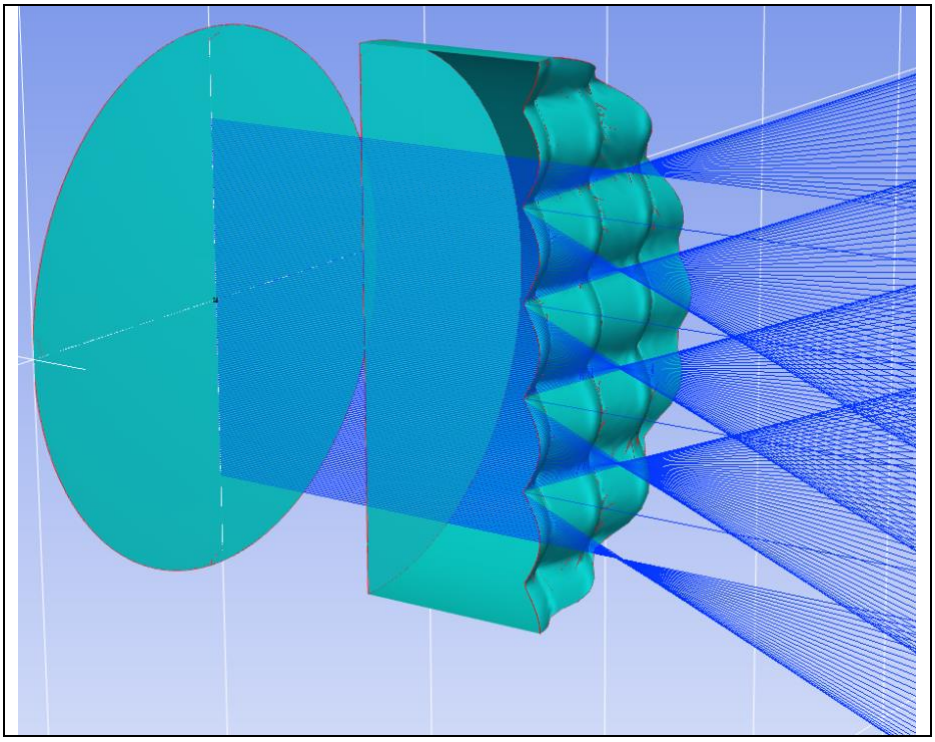


Figura 35. Vista del corte de una lente definida por el usuario con una función de sagita con componentes en X y Y.

7.30 Example - Multicore

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

start_time = time.time()

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

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

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

L1c = Kos.surf()
```

```
L1c.Rc = -7.819730726078505E+001
L1c.Thickness = 9.737604742910693E+001
L1c.Glass = "AIR"
L1c.Diameter = 30

P_Ima = Kos.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 = Kos.Setup()

Doblete1 = Kos.system(A, config_1)

def trax1(xyz, lmn, w, q):
    Rayos = Kos.raykeeper(Doblete1)
    start_time = time.time()
    for i in range(0, 700):
        Doblete1.Trace(xyz, lmn, w)
        Rayos.push()
    A = Rayos.pick(-1)
    Rayos.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))
```

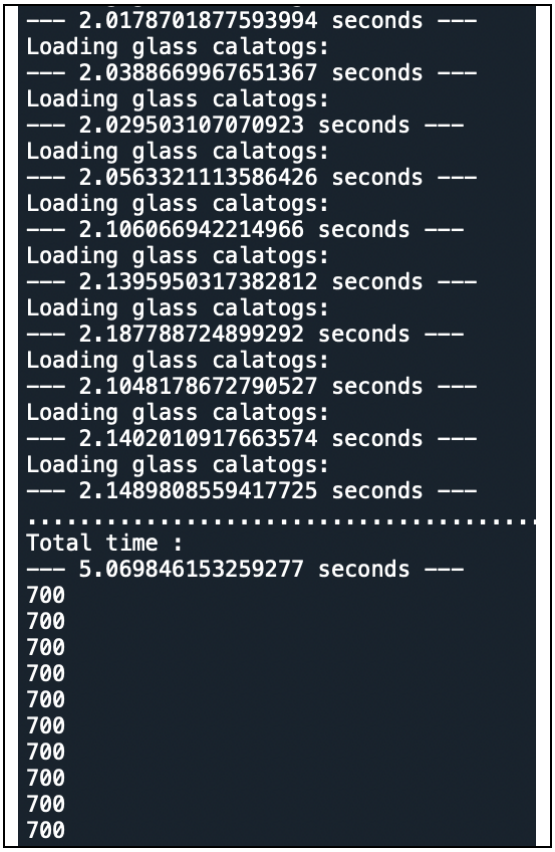


Figura 36. Salida de consola al ejecutar el mismo trazado de rayos con 8 núcleos del procesador simultáneamente.

7.31 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 KrakenOS as Kos

P_Obj = Kos.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 = Kos.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 = Kos.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 = Kos.Setup()

    Telescope = Kos.system(A, configur)
    Rays = Kos.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()
Kos.display3d(Telescope, Rays, 0)
```

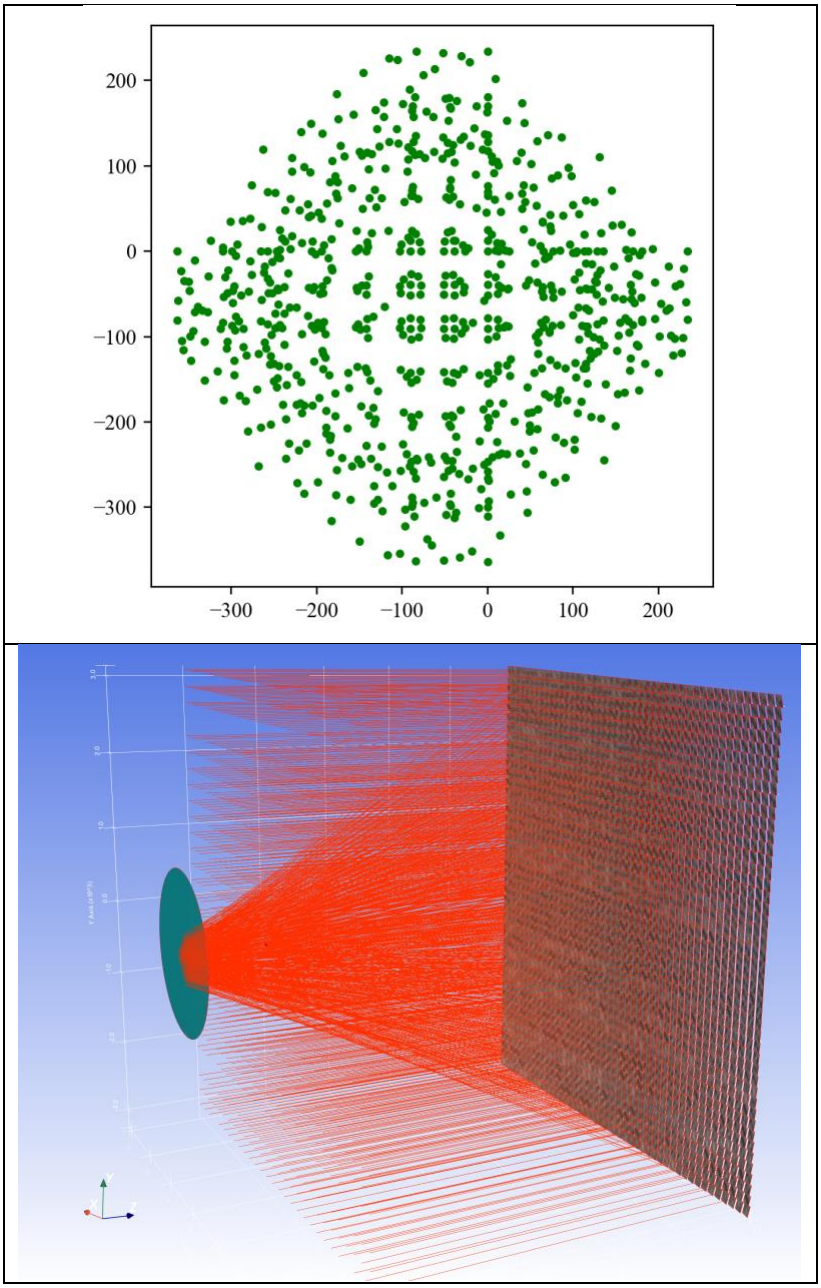


Figura 37. (Superior) Diagrama de manchas generado por un arreglo de espejos planos mostrados en la figura inferior.

7.32 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 KrakenOS as Kos

P_Obj = Kos.surf()
P_Obj.Thickness = 5000.0
P_Obj.Glass = "AIR"
P_Obj.Diameter = 6.796727741707513E+002 * 2.0
P_Obj.Drawing = 0
##############################################################################################
objeto = Kos.surf()
objeto.Rc=-12000
objeto.k=-1
objeto.Diameter=2500
objeto.Thickness = -6000
objeto.Glass = "MIRROR"
P_Ima = Kos.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 = Kos.Setup()
Telescope = Kos.system(A, configur)
Rays = Kos.raykeeper(Telescope)
W = 0.633
Sun = Kos.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()
```

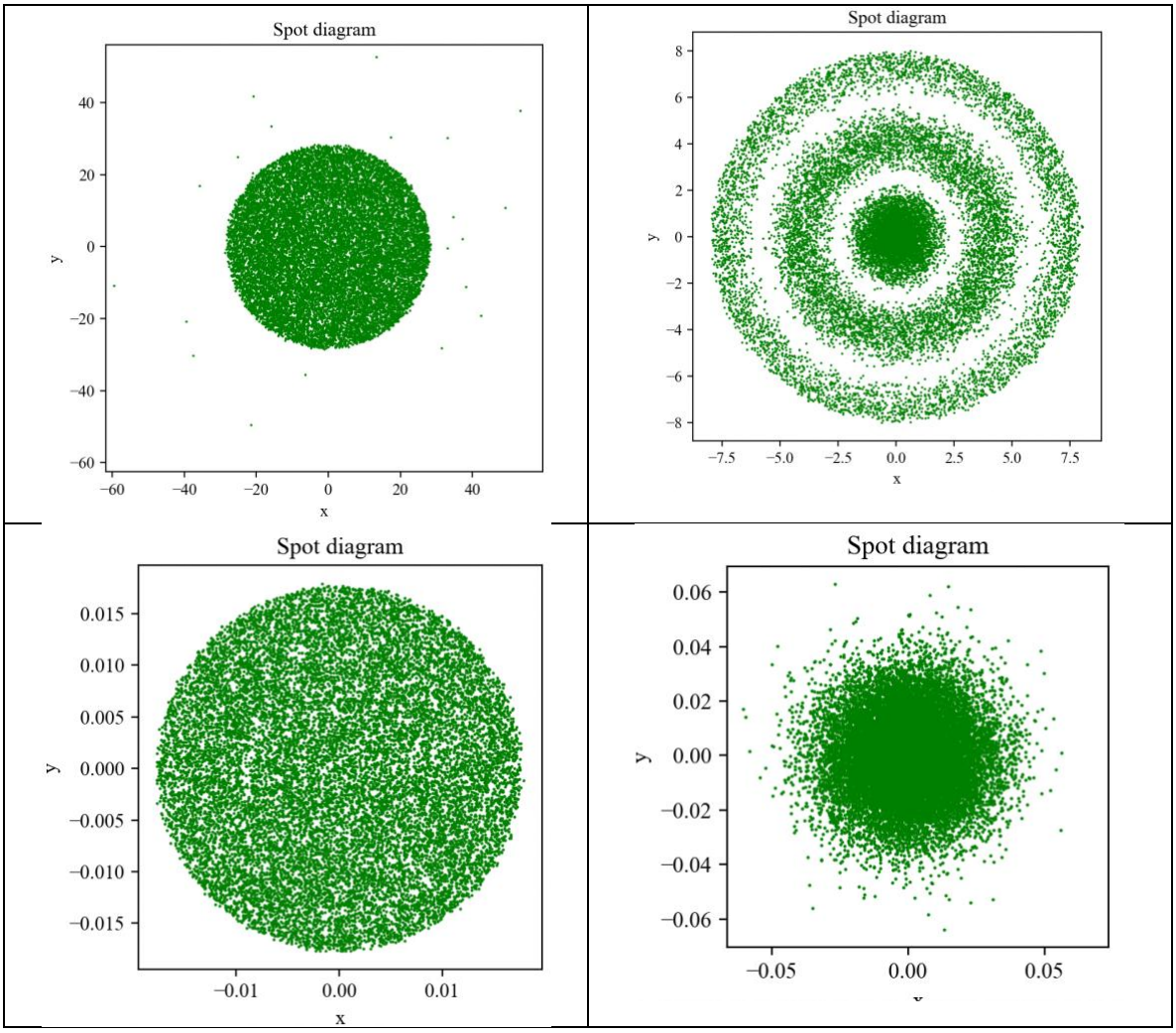


Figura 38. Examples de imágenes generadas con los rayos que provienen de 4 fuentes distintas cuya función de distribución es una función matemática. (Arriba izquierda) Distribución definida por el sol. (Arriba derecha) Función Sinc. (Abajo izquierda) constante. (Derecha abajo) Distribución Gaussiana.