Kraken - User manual

Joel Herrera V., Carlos Guerrero P., Morgan Rhaí Najera Roa, Anais Sotelo B., Ilse Plauchu F.

Instituto de Astronomía. Universidad Nacional Autónoma de México.

Km. 103 Carretera Tijuana-Ensenada, Ensenada, B. C., México.

joel@astro.unam.mx

|  |
| --- |
|  |

SUMMARY:

This manual shows the use of the library of optical simulation and ray tracing "Kraken", which has been developed in the Python language and was designed to grow and adapt to different user requirements in different tasks of optics. The library was developed under the paradigm of object-oriented programming seeking mainly its simplicity and the ability to be incorporated into code where ray tracing is necessary. Being developed in Python there are a lot of tools with which it can be combined by increasing its capabilities. At the end of this manual is shown an appendix with a series of very useful examples where all the functions are used.

Index

[Kraken - User manual 1](#_Toc80736670)

[Introducción 4](#_Toc80736671)

[1. Prerequisites 4](#_Toc80736672)

[1. Classes and Attributes 4](#_Toc80736673)

[1. Working with the library 9](#_Toc80736674)

[2. PupilCalc tool 15](#_Toc80736675)

[Atmospheric Spare Parts in PupilCalc 18](#_Toc80736676)

[Parax Tool 20](#_Toc80736677)

[Handling the 3D viewfinder 20](#_Toc80736678)

[3. Appendix - Examples 20](#_Toc80736679)

[3.1 Example - Ray 21](#_Toc80736680)

[*3.2* Example - Perfect Lens 22](#_Toc80736681)

[3.3 Example - Doublet Lens 3D color 24](#_Toc80736682)

[3.4 Example - Doublet Lens Tilt 26](#_Toc80736683)

[3.5 Example - Doublet Lens (Cálculos paraxiales) 27](#_Toc80736684)

[3.6 Example - Doublet Lens Tilt Nulls 29](#_Toc80736685)

[*3.7* Example - Doublet Lens NonSec 31](#_Toc80736686)

[3.8 Example - Doublet Lens Zernike 33](#_Toc80736687)

[3.9 Example - Doublet Lens Tilt nonSec 34](#_Toc80736688)

[3.10 Example - Doublet Lens Pupil 36](#_Toc80736689)

[3.11 Example - Doublet Lens Commands System 37](#_Toc80736690)

[3.12 Example - Doublet Lens Pupil Seidel 39](#_Toc80736691)

[3.13 Example - Doublet Lens Cylinder 41](#_Toc80736692)

[3.14 Example - Axicon 43](#_Toc80736693)

[3.15 Example - Axicon and Cylinder 44](#_Toc80736694)

[3.16 Example - Flat Mirror 45 Deg 46](#_Toc80736695)

[*3.17* Example - Parabole Mirror Shift 48](#_Toc80736696)

[*3.18* Example - Diffraction Grating Transmission 49](#_Toc80736697)

[*3.19* Example - Diffraction Grating Reflection 50](#_Toc80736698)

[3.20 Example - Tel 2M Spyder Spot Diagram 52](#_Toc80736699)

[3.21 Example - Tel 2M Spyder Spot Tilt M2 54](#_Toc80736700)

[3.22 Example - Tel 2M Pupila 57](#_Toc80736701)

[*3.23* Example - Tel 2M Error Map 58](#_Toc80736702)

[3.24 Example - Tel 2M Wavefront Fitting 60](#_Toc80736703)

[3.25 Ejemplo – Tel 2M-STL\_ImageSlicer.py 62](#_Toc80736704)

[3.26 Ejemplo – Tel 2M\_Atmospheric\_Refraction\_Corrector.py 66](#_Toc80736705)

[*3.27* Example - Extra Shape Micro Lens Array 69](#_Toc80736706)

[*3.28* Example - Extra Shape Radial Sine 71](#_Toc80736707)

[3.29 Example - Extra Shape XY Cosines 72](#_Toc80736708)

[3.30 Example - Multicore 74](#_Toc80736709)

[3.31 Example - Solid Objects STL ARRAY 76](#_Toc80736710)

[3.32 Example - Sourse\_Distribution\_Function 79](#_Toc80736711)

# Introduction

The objective of this manual is to present the list with the commands and implementations within the library, in addition, it also shows a series of examples in which all the functions are used. Kraken is a tool for the simulation of optical systems, it consists of a library developed in the Python 3.0 programming language and the Numpy, PyVTK and PyVista libraries which allows it to provide three-dimensional visualization of optical elements. This tool has focused on the paradigm of object-oriented programming, this apparently is naturally the one that gives us greater simplicity in the implementation of a system as will be seen in this document.

# Prerequisites

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

1. Python '3.7.4'
2. numpy '1.18.5'
3. pyvista '0.25.3'
4. pyvtk '0.5.18'
5. vtk '8.2'
6. csv '1.0'
7. Place the "library", "Cat" directiony and the Kraken.py file in the same path where the code you want to run is.
8. All files included in the directiony and beginning their name with the word "***Example - -***" are examples in which the functions of the library are used.

# Classes and Attributes

The library has been simplified to the point of having only two classes of objects for the definition of a system, these are ***surf*** and ***system,*** whose application is described later in this section.

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

|  |  |
| --- | --- |
| ***Table 1. Parameters of a surface*** | |
| surf. Name="" | Element name, useful only for 2D diagram or ray identification |
| surf. NamePos=(0,0) | Position of the note with the Name "Name" in the 2D diagram |
| surf. Note="None" | Useful for user notes on that surface |
| surf. Rc = 9999999999.0 | Radius of paraxial curvature in millimeters, |
| surf. Cylinder\_Rxy\_Ratio=1 | Ratio between the radius of axial and sagittal curvature, useful for cylindrical and toroid lenses |
| surf. Axicon=0 | Simulation of axicons, for nonzero values an axicon is generated with the angle entered. |
| surf. Thickness=0.0 | Separation between this surface and the next surface. |
| surf. Diameter = 1.0 | The outer diameter of the surface. |
| surf. InDiameter=0.0 | Internal diameter of the surface, useful for elements such as a primary mirror with central aperture. |
| surf.k=0.0 | Conicity constant for classical conic surfaces, k=0 for spherical, k=-1 for parabola, etc. |
| surf. DespX=0.0 | Surface displacement on the X,Y and Z axis (in millimeters) |
| surf. DespY=0.0 |
| surf. DespZ=0.0 |
| surf. TiltX=0.0 | Rotation of the surface on the X,Y and Z axis (in degrees). |
| surf. TiltY=0.0 |
| surf. TiltZ=0.0 |
| surf. Order=0 | It defines the order of the transformations, if the value is 0 first the translations are performed and then the rotations, if the value is 1 then the rotations are performed first and then as translations. |
| surf. AxisMove=1 | Define what will happen to the optical axis after a coordinate transformation, if the value is zero the transformation is only performed 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 doubled, the latter is useful for flat mirrors, see example. |
| surf. Diff\_Ord=0.0 | Diffraction order, converts the element into a diffraction grid, the curvature radius value must be omitted to be a flat diffraction grid, it can be used in transmission or refraction. |
| surf. Grating\_D=0.0 | Separation (microns) between the lines of the diffraction grid. |
| surf. Grating\_Angle=0.0 | Angle of the grid lines in the plane of the surface i.e. around the optical axis. This is useful for simulating conical scattering. |
| surf. ZNK=np.zeros(36) | Numpy type arrangement of 36 elements corresponding to the coefficients of the Zernike polynomials in noll ref. xxxxx nomenclature. |
| surf. ShiftX=0 | Offset of the surface profile function on the x or y axis, this is useful for example for off-axis surfaces such as parabolas. |
| surf. ShiftY=0 |
| surf. Mask=0 | 0 non masked, 1 apperture, 2 Obstruction. |
| surf. Mask\_Shape=Objeto\_3D | Example: Objeto\_3D=pv. Disc(center=[0.0,0.0,0.0], inner=0, outer=0.001, normal=(0,0,1),r\_res=3.c\_res=3). |
| surf. AspherData =np.zeros(Fix) | Arrangement of coefficients for aspherical surface |
| Self. ExtraData=[f, coef] | User-created surface with a sagite function dependent on (x,y,V), where V can contain an arrangement of coefficients usable by the function:  ﻿  -The sagite function is defined  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  z = (c \* s \* s / (1.0 + np.sqrt(InRoot)))  return z  -Coefficients are defined if necessary  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] | The parameter receives an error map generated with a Numpy array for the coordinates in X, and Y and the height in Z with a **SPACE** value between X, Y generated as follows.  Example:﻿  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] |
| surf. Drawing=1 | 1 to have the element drawn in the 3D graph, 0 to omit the drawing of the element. |
| surf. Color=[0,0,0] | Set the color of the item in the format [1,1,1] |
| surf. Solid\_3d\_stl="None" | The 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 that the ray traverses, the public implementations of ***system*** are shown below.

***To understand how these elements are called see: Example - -Doublet\_Lens\_CommandsSystem***

|  |  |
| --- | --- |
| ***Table 2. Parameters for communicating with system*** | |
| system. Trace(pS, dC, wV) | Trace is the main implementation of the *system* object; it performs the tracing of a ray through all the surfaces it encounters in its path sequentially. The ray must be defined by a point of origin "pS", the direction cosines (dC) the wavelength "wV". See examples  pS = [1.0, 0.0, 0.0]  dC=[0.0,0.0,1.0]  wV=0.4 |
| system. NsTrace(pS, dC, wV) | Trace rays in the same way as Trace, however, it does so non-sequentially, the parameters of the ray are defined in the same way. |
| Prx=system. Parax(w) | Returns the following paraxial calculations also accessible from system:  Prx=SistemMatrix, S\_Matrix, N\_Matrix, a, b, c, d, EFFL, PPA, PPP, CC, N\_Prec, DD |
| system.disable\_inner | Enables and disables center apertures, this is very useful, for example, when you want to calculate a ray stroke without the vignetting of the aperture of a primary mirror. |
| system.enable\_inner | Rear main plane |
| system. SURFACE | Returns a list of the number of surfaces that the ray passes through.  Returns a list of the name of surfaces that the ray passed through, if a name was not added to the surfaces then the list will appear with empty fields. Naming surfaces is very useful, for example, to identify whether a ray has touched that surface. |
| system.NAME |
| system. GLASS | Returns a list of the materials that the ray has traversed. |
| system. XYZ | Returns the *[X,Y,Z]* coordinates of the ray from its origin to the image plane |
| system. S\_XYZ | Returns the *[X,Y,Z]* coordinates 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 it intersects to the image plane. |
| system. OST\_XYZ | Returns the *[X,Y,Z]* coordinates 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 of the distances traveled by the ray between intersection points. |
| system. OP | Returns a list of the optical paths traveled by the ray between intersection points, for this the dispersion of the glass and the distance between the intersection points are considered. |
| system. TOP | Returns the total optical path of the ray from the source to the last point of intersection. |
| system. TOP\_S | Returns a cumulative list of the optical path of the ray from the source to the last point of intersection. |
| 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 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 of the direction cosses [*L,M*,*N]* of the normals for the intersection points of a ray across all the interfaces it passes through. |
| system. LMN | Returns a list of the direction cosses [*L,M*,*N]* of an incident ray across all the interfaces it traverses. |
| system. R\_LMN | Returns a list of the resulting ray direction cosines [*L,M*,*N]* across all the interfaces it traverses. |
| system. N0 | Refractive indices before and after each interface through which the ray passes. This index is calculated with the dispersion of the material and the wavelength of the ray in question. |
| system. N1 | Returns a list of the resulting ray direction cosines [*L,M*,*N]* across all the interfaces it traverses. |
| system. WAV | Ray wavelength in question, although the command returns a list all values are equal because the wavelength is constant for a ray, the size of the list indicates only the number of iterations with system interfaces.  Returns a list of the terms, *L, M,*  or  *N* of the direction cosses that define the diffraction grid lines over the plane. |
| system. G\_LMN |
| system. ORDER | Returns a list of diffraction commands associated with the ray in question. |
| system. GRATING | The distance between lines of the diffraction grid. |
| system. RP | Returns a list with the Fresnel coefficients of reflection and transmission for polarization S and P |
| system. RS | The distance between lines of the diffraction grid. |
| system. TP | Returns a list with the Fresnel coefficients of reflection and transmission for polarization S and P  Total energy transmitted or reflected per element.  Total transmitted or total reflected energy. |
| system. TS |
| system. TTBE |
| system.TT |
| system.targ\_surf(int) | Limits ray tracing to the defined surface (int) |
| system.flat\_surf(int) | It is defined with the integer of the surface that needs to be made flat, -1 to restore |
| system.disable\_inner() | If any system surface has a central hole it is disabled. |
| system.enable\_inner() | If any surface () has a central hole disabled this command rehabilitates it. |

# Working with the library

**Example: Ray**

﻿

|  |
| --- |
| **import** numpy **as** np  **import** Kraken **as** kn |

Allsurfaces are declared separately, each with the possible parameters for a surface shown in Table 1.

|  |
| --- |
| **P\_Obj = kn.surf()**  **P\_Obj.Rc = 0.0**  **P\_Obj.Thickness = 0.1**  **P\_Obj.Glass = "AIR"**  **P\_Obj.Diameter = 30.0**    **P\_Obj2 = kn.surf()**  **P\_Obj2.Rc = 0.0**  **P\_Obj2.Thickness = 10**  **P\_Obj2.Glass = "AIR"**  **P\_Obj2.Diameter = 30.0**    **L1a = kn.surf()**  **L1a.Rc = 9.284706570002484E+001**  **L1a.Thickness = 6.0**  **L1a.Glass = "BK7"**  **L1a.Diameter = 30.0**  **L1a.Axicon = 0**    **L1b = kn.surf()**  **L1b.Rc = -3.071608670000159E+001**    **L1b.Thickness = 3.0**  **L1b.Glass = "F2"**  **L1b.Diameter = 30**    **L1c = kn.surf()**  **L1c.Rc = -7.819730726078505E+001**  **L1c.Thickness = 9.737604742910693E+001**  **L1c.Glass = "AIR"**  **L1c.Diameter = 30**    **P\_Ima = kn.surf()**  **P\_Ima.Rc = 0.0**  **P\_Ima.Thickness = 0.0**  **P\_Ima.Glass = "AIR"**  **P\_Ima.Diameter = 18.0**  **P\_Ima.Name = "Plano imagen"** |

The ***Kraken-setup*** class is loaded, within this the directories of the glass catalogs will be established among other things, in future versions with different configurations can be established for a system which can be useful. Now we will load the default configuration in the ***configuracion\_1***.

﻿ **configuracion\_1 = kn.Kraken\_setup()**

An arrangement ***A*** is made with the elements declared for all surfaces, then an optical system is created with all the surfaces contained in ***A*** and the ***configuracion\_1,*** and finally a container is created for the rays with ***Raykeeper****.*

|  |
| --- |
| **A = [P\_Obj, P\_Obj2, L1a, L1b, L1c, P\_Ima]**  **configuracion\_1 = kn.Kraken\_setup()**  **Doblete = kn.system(A, configuracion\_1)** |

Every ray has 3 parameters, a source coordinate called ***XYZ*** for this example, a direction defined by direction cosines, defined below as LMN, these two parameters are arrays with three values [x,y,z] and [l, m, n]. the third parameter to be defined is the wavelength, here expressed as ***w*** with a value of 0.4μm.

|  |
| --- |
| **XYZ = [0, 14, 0]**  **Theta = 0.1**  **LMN = [0.0, np.sin(np.deg2rad(Theta)), -np.cos(np.deg2rad(Theta))]**  **W = 0.4** |

The ray is traced through the ***Doublet*** system as follows

**Doblete.Trace(XYZ, LMN, W)**

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

**print(Doblete.GLASS)**

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

**print(Doblete.XYZ)**

**print(Doblete.LMN)**

for which we obtain the following outputs respectively:

|  |
| --- |
| Doblete.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, 0.05825657548273888, 116.37604742910693]] |
| Doblete.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 type arrays, therefore we can perform the operations we want with them directly, for example, below we see the shape of the arrays

**print(np.shape(Doblete.XYZ))**

**print(np.shape(Doblete.LMN))**

what returns us:

**np.shape(Doblete.XYZ) = (5, 3)**

**np.shape(Doblete.LMN) = (4, 3)**

Note that Doblete.XYZ contains 5 arrays of three elements (x,y,z coordinates in three-dimensional space), while Doublet.LMN only contains 4 elements, this is because it only shows the direction cosines between surfaces, that is, if a system has 5 elements from the object plane to the image plane, then there are only 4 ray segments between surfaces and therefore only 4 sets of direction cosines, exemplified below with -> between surfaces.

A=[P\_Obj -> L1a -> L1b -> L1c -> P\_Ima]

After making the stroke of a ray with the ***Trace*** option we can request information to Doblete in the way mentioned above, with this information you can perform graphs or different analyses, usually in necessary to make a trace with many rays, to preserve the results for one or more rays we have the ray container in the ***Raykeeper***class, with which for the example of this document we create the Ray object.

**Rayos = kn.raykeeper(Doblete)**

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

Each individual ray is saved by passing to the system in question as a parameter to the container as follows, this is repeated each time a ray is drawn.

**Rayos.push()**

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

For two-dimensional ***display2D graphs(System, Raykeeper, parameter)*** where the parameter can be integers 0 or 1 indicating whether the graph will be in the xz or yz plane.

For two-dimensional ***display3D graphs(System, Raykeeper, parameter***) where the parameter can be integers 0 through 2.

|  |  |
| --- | --- |
| 0 | for a deployment of the complete elements |
| 1 | for deployment with surfaces with a cut of 1/4 |
| 2 | for the deployment of cross-sections. |

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

**Display2D(Doblete,Rayos,0)**

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

|  |
| --- |
| Gráfico  Descripción generada automáticamente |
| Figure 1. 2D visualization of a ray traced through a doublet. |

To perform the three-dimensional deployment, we will use:

**Display3D(Doblete,Rayos,2)**

this opens the next viewer window

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

We can store several rays, for example, then nest the creation of the ray, its stroke and it’s stored inside a ***for*** loop.

|  |
| --- |
| **for x in range(-10, 10):**  ***# Creamos un rayo paralelo al eje óptico y cambiamos la altura con "x" en el for***  **XYZ = [0.0, x, 0.0]**  **LMN =[0.0,0.0,1.0]**  **W=0.4**  ***# Trazamos el rayo***  **Doblete.Trace(XYZ, LMN,W)**  ***#Almacenamos el rayo***  **Rayos.push(Doblete)**    ***# Despliega sistema con todos los rayos***  **Display3D(Doblete,Rayos,2)** |

Where so we get all the rays

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

The ray container contains a list of preserved almost all the parameters that are accessible in the ***System***class, however, now, by containing multiple rays these parameters are arrays of arrays so certain considerations must be considered.

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

**print(Rayos.nrays) Resultado: 100**

Therefore, we have in this example 100 positions with rays on which we can request information from the container through the parameters shown in Table 3 (Middle Column), these have the same information described in Table 2 of the ***System***class. These calls to the ***Raykeeper*** container depend on the ray number defined by an integer value ***[#].*** If the lightning count is not indicated, the total data set in Numpy arrays is obtained. As already mentioned there are rays that did not touch any surface, which if they touched any surface we call ***valid*** rays and you can ask the container for a list of the address of these in the list of rays, this is shown below.

**print(Rayos.valid())**

so, we get the following list.

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

We can now call the ray information using these values and the keywords in Table 3 (Center Column), however, when executing the ***valid()*** statement on line 78 of the example, a new set of calls is created, they are the ones shown in Table 3 (right column) these are identical to those in the middle column, but with the prefix ***valid\_,***it is important to clarify that if the ***valid()***statement is not executed. The calls shown in the right column will be empty so if you do not take this care, you will get errors.

The arrangements obtained with this new call set contain only valid rays and can now be used directly as the empty rays have been removed in them, therefore, the numbering will be shifted according to the number of empty spaces removed. Depending on the task you want to perform, it could have application in one way or the other.

In the same way a call set is created for invalid rays, as expected, only the initial ray information can be obtained, these calls appear in Table 3 left column and is limited to coordinates of origin, direction and wavelength. The prefix for these 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 erase the rays inside a ***Raykeeper*** you can redefine the container, repeating its creation of the container, another way is using the internal ***clean*** implementation.

Rayos.clean()

# PupilCalc tool

The aperture of the system or as it is commonly known "Aperture Stop" is the element that defines the amount of light that passes through the entire optical system, the image of the aperture of the system produced by the optical elements prior to it is known as the input pupil, the image of the input pupil produced by the entire system is known as the output pupil. Contrary to what might first be assumed, the input pupil is not necessarily defined by the first element sometimes the element that defines the amount of light is not an optical surface but a mechanical element.

The following figure shows an example of a "Stop Aperture" on an element after a lens, specifically on surface 4. In addition, in the figure two different fields are shown, both also cross this opening in an identical way, here a problem arises because to generate rays that uniformly cover the opening of the system it would be necessary to make a ray tracing towards the object plane and with this define the properties of the ray so that it crosses said area of the pupil, note in the figure that the fields have different origins, so generating one by one the rays can be a complicated task to perform by calculating them manually.

|  |
| --- |
|  |
| Figure 4. Display of two beams that match the position defined as pupil. |

To facilitate the generation of rays that pass through the pupil you have the ***pupilecalc*** tool, this class generates an iterative object as follows.

﻿

|  |
| --- |
| **W = 0.4**  **Surf= 4**  **AperVal = 10**  **AperType = "EPD"**  **Pup = kn.PupilCalc(Doblete, sup, W, AperType, AperVal)** |

Where ***Doblete*** is the system that we generate with the ***system*** class, ***Surf***is the surface number that represents the aperture of the system, in the example of the image this is the surface 4, and ***W*** is the wavelength for which the output and input pupil will be calculated.

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

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

|  |  |
| --- | --- |
| **Object attributes corresponding to pupil parameters** | |
| Input pupil radius | Pup.RadPupInp |
| Input pupil position | Pup.PosPupInp |
| Output pupil radius | Pup.RadPupOut |
| Exit pupil position | Pup.PosPupOut |
| Output pupil position with respect to the focal plane | Pup.PosPupOutFoc |
| Exit pupil orientation | Pup.DirPupSal |

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

In addition to obtaining these parameters, you can also generate ray patterns in the pupil, calculating the direction cosines and the origin coordinates by defining the parameters.

|  |  |
| --- | --- |
| Generation of automatic rays based on the pupil | |
| Pup.Samp= # | Integer for ray sampling in the pupil, default value 5 |
| Pup.Ptype=**"Array type"** | **"rteta"** Generates a ray at an angle of the unit pupil to a radial position, the radius and angle must be defined in the form:  **Pup.rad=n**  **Pup.teta=m**  Where n is a floating number of 0-1 and tete is the angle from 0 to 360 |
| **"chief"** Main ray passing through the center of the pupil |
| **"hexapolar",** Hexapolar ray arrangement |
| **"square"**, rectangular ray arrangement |
| **"fanx",**Line array only on the x-axis |
| **"fany",**Line array only on the y-axis |
| **"fan"**, Line array on the x-y-axis |
| **"rand"**, Random arrangement |
| Pup.FieldType=**"height" or "angle"** | Defines the field type, in terms of the height of the object at the distance of the object plane with the **"height"**parameter, for parallel rays reaching the pupil 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 chosen in FieldType |

The attributes shown in the previous table define the type of rays we want, to obtain the arrangement of rays all we must do is transfer them from the unitary pupil to the real pupil obtaining the direction cosines and the coordinates of origin in the form of arrangements as follows:

﻿

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

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

﻿

|  |
| --- |
| **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()** |

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

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

We can then generate the spot diagram

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

It is important to note that, if you want to analyze different fields, you must decide for each field and save the ray tracing in different containers if you do not want to combine the results.

## Atmospheric Spare Parts in PupilCalc

If ***FieldType*** is defined as "angle", then as rays come from infinity, they are very suitable for use in telescopes. That is why an open source ***AstroAtmosphere*** library has also been included to complement the article: "Quantification of the expected residual dispersion of the MICADO Near-IR imaging instrument, vanden Born & Jellema, 2020, MNRAS, DOI:  [10.1093/mnras/staa1870."](https://doi.org/10.1093/mnras/staa1870)

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

﻿

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

Below is an example using a telescope where 3 groups of rays are generated, only changing the wavelength between them

﻿﻿

|  |
| --- |
| **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()** |

Ray tracing is then performed for the three groups and stored in different containers.

﻿

|  |
| --- |
| **for i in range(0,len(xa)):**  **pSource\_0 = [xa[i], ya[i], za[i]]**  **dCos=[La[i], Ma[i], Na[i]]**  **Telescopio.Trace(pSource\_0, dCos, W1)**  **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()** |

As a result of making the spot diagrams we get the following image where you can see a separation of approximately 72μm between the extreme wavelengths.

﻿

|  |
| --- |
| **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")** |

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

## Parax Tool

It performs a series of paraxial calculations which can later be called from system.

**Prx = Doblete.Parax(0.4)**

**SistemMatrix, S\_Matrix, N\_Matrix, a, b, c, d, EFFL, PPA, PPP, CC, N\_Prec, DD = Prx**

## Handling the 3D viewfinder

* Rotate simulation in any direction, LEFT MOUSE BUTTON
* Zoom, move the mouse up and down by pressing the right mouse button
* Drag the simulation, drag by pressing LEFT MOUSE BUTTON and pressing shift key on your keyboard.

# Appendix - Examples

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

Examp-Axicon.py

Examp-Axicon\_And\_Cylinder.py

Examp-Diffraction\_Grating\_Reflection.py

Examp-Diffraction\_Grating\_Transmission.py

Examp-Doublet\_Lens-ParaxMatrix.py

Examp-Doublet\_Lens.py

Examp-Doublet\_Lens\_3Dcolor.py

Examp-Doublet\_Lens\_CommandsSystem.py

Examp-Doublet\_Lens\_Cylinder.py

Examp-Doublet\_Lens\_NonSec.py

Examp-Doublet\_Lens\_Pupil.py

Examp-Doublet\_Lens\_Pupil\_Seidel.py

Examp-Doublet\_Lens\_Tilt-Nulls.py

Examp-Doublet\_Lens\_Tilt.py

Examp-Doublet\_Lens\_Tilt\_non\_sec.py

Examp-Doublet\_Lens\_Zernike.py

Examp-ExtraShape\_Micro\_Lens\_Array.py

Examp-ExtraShape\_Radial\_Sine.py

Examp-ExtraShape\_XY\_Cosines.py

Examp-Flat\_Mirror\_45Deg.py

Examp-MultiCore.py

Examp-ParaboleMirror\_Shift.py

Examp-Perfect\_lens.py

Examp-Ray.py

Examp-Solid\_Object\_STL.py

Examp-Solid\_Object\_STL\_ARRAY.py

Examp-Sourse\_Distribution\_Function.py

Examp-Tel\_2M.py

Examp-Tel\_2M\_Error\_Map.py

Examp-Tel\_2M\_Pupila.py

Examp-Tel\_2M\_Spyder\_Spot\_Diagram.py

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

# Example - Ray

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**Examp Ray"""**

**import numpy as np**

**import Kraken as kn**

**P\_Obj = kn.surf()**

**P\_Obj.Rc = 0.0**

**P\_Obj.Thickness = 0.1**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 30.0**

**P\_Obj2 = kn.surf()**

**P\_Obj2.Rc = 0.0**

**P\_Obj2.Thickness = 10**

**P\_Obj2.Glass = "AIR"**

**P\_Obj2.Diameter = 30.0**

**L1a = kn.surf()**

**L1a.Rc = 9.284706570002484E+001**

**L1a.Thickness = 6.0**

**L1a.Glass = "BK7"**

**L1a.Diameter = 30.0**

**L1a.Axicon = 0**

**L1b = kn.surf()**

**L1b.Rc = -3.071608670000159E+001**

**L1b.Thickness = 3.0**

**L1b.Glass = "F2"**

**L1b.Diameter = 30**

**L1c = kn.surf()**

**L1c.Rc = -7.819730726078505E+001**

**L1c.Thickness = 9.737604742910693E+001**

**L1c.Glass = "AIR"**

**L1c.Diameter = 30**

**P\_Ima = kn.surf()**

**P\_Ima.Rc = 0.0**

**P\_Ima.Thickness = 0.0**

**P\_Ima.Glass = "AIR"**

**P\_Ima.Diameter = 18.0**

**P\_Ima.Name = "Plano imagen"**

**A = [P\_Obj, P\_Obj2, L1a, L1b, L1c, P\_Ima]**

**configuracion\_1 = kn.Kraken\_setup()**

**Doblete = kn.system(A, configuracion\_1)**

**Rayos = kn.raykeeper(Doblete)**

**pSource\_0 = [0, 14, 0]**

**tet = 0.1**

**dCos = [0.0, np.sin(np.deg2rad(tet)), -np.cos(np.deg2rad(tet))]**

**W = 0.4**

**Doblete.Trace(pSource\_0, dCos, W)**

**Rayos.push()**

**kn.display3d(Doblete, Rayos, 2)**

|  |
| --- |
|  |

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

# Example - Perfect Lens

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**Examp Perfect Lens"""**

**import time**

**import matplotlib.pyplot as plt**

**import numpy as np**

**import Kraken as kn**

**start\_time = time.time()**

**P\_Obj = kn.surf()**

**P\_Obj.Rc = 0.0**

**P\_Obj.Thickness = 50**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 30.0**

**L1a = kn.surf()**

**L1a.Thin\_Lens = 100.**

**L1a.Thickness = (100 + 50)**

**L1a.Rc = 0.0**

**L1a.Glass = "AIR"**

**L1a.Diameter = 30.0**

**L1b = kn.surf()**

**L1b.Thin\_Lens = 50.**

**L1b.Thickness = 100.**

**L1b.Rc = 0.0**

**L1b.Glass = "AIR"**

**L1b.Diameter = 30.0**

**P\_Ima = kn.surf()**

**P\_Ima.Rc = 0.0**

**P\_Ima.Thickness = 0.0**

**P\_Ima.Glass = "AIR"**

**P\_Ima.Diameter = 100.0**

**P\_Ima.Name = "Plano imagen"**

**A = [P\_Obj, L1a, L1b, P\_Ima]**

**config\_1 = kn.Kraken\_setup()**

**Doblete = kn.system(A, config\_1)**

**Rayos1 = kn.raykeeper(Doblete)**

**Rayos2 = kn.raykeeper(Doblete)**

**Rayos3 = kn.raykeeper(Doblete)**

**RayosT = kn.raykeeper(Doblete)**

**tam = 10**

**rad = 10.0**

**tsis = len(A) - 1**

**for j in range(-tam, tam + 1):**

**for i in range(-tam, tam + 1):**

**x\_0 = (i / tam) \* rad**

**y\_0 = (j / tam) \* rad**

**r = np.sqrt((x\_0 \* x\_0) + (y\_0 \* y\_0))**

**if r < rad:**

**tet = 0.0**

**pSource\_0 = [x\_0, y\_0, 0.0]**

**dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]**

**W = 0.4**

**Doblete.Trace(pSource\_0, dCos, W)**

**Rayos1.push()**

**RayosT.push()**

**kn.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))**

|  |
| --- |
|  |
|  |

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

# Example - Doublet Lens 3D color

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**Examp Doublet Lens 3D color"""**

**import numpy as np**

**import Kraken as kn**

**P\_Obj = kn.surf()**

**P\_Obj.Rc = 0.0**

**P\_Obj.Thickness = 10**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 30.0**

**L1a = kn.surf()**

**L1a.Rc = 9.284706570002484E+001**

**L1a.Thickness = 6.0**

**L1a.Glass = "BK7"**

**L1a.Diameter = 30.0**

**L1a.Axicon = 0**

**L1a.Color = [.8, .7, .4]**

**L1b = kn.surf()**

**L1b.Rc = -3.071608670000159E+001**

**L1b.Thickness = 3.0**

**L1b.Glass = "F2"**

**L1b.Diameter = 30**

**L1b.Color = [.7, .4, .4]**

**L1c = kn.surf()**

**L1c.Rc = -7.819730726078505E+001**

**L1c.Thickness = 9.737604742910693E+001**

**L1c.Glass = "AIR"**

**L1c.Diameter = 30**

**P\_Ima = kn.surf()**

**P\_Ima.Rc = 0.0**

**P\_Ima.Thickness = 0.0**

**P\_Ima.Glass = "AIR"**

**P\_Ima.Diameter = 100.0**

**P\_Ima.Name = "Plano imagen"**

**A = [P\_Obj, L1a, L1b, L1c, P\_Ima]**

**configuracion\_1 = kn.Kraken\_setup()**

**Doblete = kn.system(A, configuracion\_1)**

**Rayos = kn.raykeeper(Doblete)**

**tam = 5**

**rad = 10.0**

**tsis = len(A) - 1**

**for i in range(-tam, tam + 1):**

**for j in range(-tam, tam + 1):**

**x\_0 = (i / tam) \* rad**

**y\_0 = (j / tam) \* rad**

**r = np.sqrt((x\_0 \* x\_0) + (y\_0 \* y\_0))**

**if r < rad:**

**tet = 0.0**

**pSource\_0 = [x\_0, y\_0, 0.0]**

**dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]**

**W = 0.4**

**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()**

**kn.display3d(Doblete, Rayos, 1)**

|  |
| --- |
|  |

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

# Example - Doublet Lens Tilt

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**Examp Doublet Lens Tilt"""**

**import numpy as np**

**import Kraken as kn**

**P\_Obj = kn.surf()**

**P\_Obj.Rc = 0.0**

**P\_Obj.Thickness = 10**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 30.0**

**L1a = kn.surf()**

**L1a.Rc = 9.284706570002484E+001**

**L1a.Thickness = 6.0**

**L1a.Glass = "BK7"**

**L1a.Diameter = 30.0**

**L1a.AxisMove = 1**

**L1a.TiltX = 1**

**L1a.DespY = 10.**

**L1b = kn.surf()**

**L1b.Rc = (-3.071608670000159E+001)**

**L1b.Thickness = 3.0**

**L1b.Glass = "F2"**

**L1b.Diameter = 30**

**L1c = kn.surf()**

**L1c.Rc = (-7.819730726078505E+001)**

**L1c.Thickness = 9.737604742910693E+001**

**L1c.Glass = "AIR"**

**L1c.Diameter = 30**

**P\_Ima = kn.surf()**

**P\_Ima.Rc = 0.0**

**P\_Ima.Thickness = 0.0**

**P\_Ima.Glass = "AIR"**

**P\_Ima.Diameter = 100.0**

**P\_Ima.Name = "Plano imagen"**

**A = [P\_Obj, L1a, L1b, L1c, P\_Obj, L1a, L1b, L1c, P\_Ima]**

**configuracion\_1 = kn.Kraken\_setup()**

**Doblete = kn.system(A, configuracion\_1)**

**Rayos = kn.raykeeper(Doblete)**

**tam = 5**

**rad = 10.0**

**tsis = len(A) - 1**

**for i in range(-tam, tam + 1):**

**for j in range(-tam, tam + 1):**

**x\_0 = (i / tam) \* rad**

**y\_0 = (j / tam) \* rad**

**r = np.sqrt((x\_0 \* x\_0) + (y\_0 \* y\_0))**

**if r < rad:**

**tet = 0.0**

**pSource\_0 = [x\_0, y\_0, 0.0]**

**dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]**

**W = 0.4**

**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()**

**kn.display3d(Doblete, Rayos, 2)**

**kn.display2d(Doblete, Rayos, 0)*# !/usr/bin/env python3***

|  |
| --- |
|  |

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

# Example - Doublet Lens (Cálculos paraxiales)

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**Examp Doublet Lens Para Matrix"""**

**import time**

**import Kraken as kn**

**start\_time = time.time()**

**P\_Obj = kn.surf()**

**P\_Obj.Rc = 0.0**

**P\_Obj.Thickness = 10**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 30.0**

**L1a = kn.surf()**

**L1a.Rc = 9.284706570002484E+001**

**L1a.Thickness = 6.0**

**L1a.Glass = "BK7"**

**L1a.Diameter = 30.0**

**L1a.Axicon = 0**

**L1b = kn.surf()**

**L1b.Rc = -3.071608670000159E+001**

**L1b.Thickness = 3.0**

**L1b.Glass = "F2"**

**L1b.Diameter = 30**

**L1c = kn.surf()**

**L1c.Rc = -7.819730726078505E+001**

**L1c.Thickness = 9.737604742910693E+001**

**L1c.Glass = "AIR"**

**L1c.Diameter = 30**

**P\_Ima = kn.surf()**

**P\_Ima.Rc = 0.0**

**P\_Ima.Thickness = 0.0**

**P\_Ima.Glass = "AIR"**

**P\_Ima.Diameter = 3.0**

**P\_Ima.Name = "Plano imagen"**

**A = [P\_Obj, L1a, L1b, L1c, P\_Ima]**

**config\_1 = kn.Kraken\_setup()**

**Doblete = kn.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)**

|  |
| --- |
|  |

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

# Example - Doublet Lens Tilt Nulls

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**Examp Doublet Lens Tilt Nulls"""**

**import numpy as np**

**import Kraken as kn**

**P\_Obj = kn.surf()**

**P\_Obj.Rc = 0.0**

**P\_Obj.Thickness = 1**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 30.0**

**L1a = kn.surf()**

**L1a.Rc = 5.513435044607768E+001**

**L1a.Thickness = 6.0**

**L1a.Glass = "BK7"**

**L1a.Diameter = 30.0**

**L1a.TiltX = 4**

**Null1\_L1a = kn.surf()**

**Null1\_L1a.Thickness = -L1a.Thickness**

**Null1\_L1a.Glass = "NULL"**

**Null1\_L1a.Diameter = L1a.Diameter**

**Null1\_L1a.TiltX = -L1a.TiltX**

**Null1\_L1a.Order = 1**

**Null2\_L1a = kn.surf()**

**Null2\_L1a.Thickness = L1a.Thickness**

**Null2\_L1a.Glass = "NULL"**

**Null2\_L1a.Diameter = L1a.Diameter**

**L1b = kn.surf()**

**L1b.Rc = -4.408716526030626E+001**

**L1b.Thickness = 3.0**

**L1b.Glass = "F2"**

**L1b.Diameter = 30**

**L1c = kn.surf()**

**L1c.Rc = -2.246906271406796E+002**

**L1c.Thickness = 9.737871661422000E+001**

**L1c.Glass = "AIR"**

**L1c.Diameter = 30**

**P\_Ima = kn.surf()**

**P\_Ima.Rc = 0.0**

**P\_Ima.Thickness = 0.0**

**P\_Ima.Glass = "AIR"**

**P\_Ima.Diameter = 100.0**

**P\_Ima.Name = "Plano imagen"**

**A = [P\_Obj, L1a, Null1\_L1a, Null2\_L1a, L1b, L1c, P\_Ima]**

**configuracion\_1 = kn.Kraken\_setup()**

**Doblete = kn.system(A, configuracion\_1)**

**Rayos = kn.raykeeper(Doblete)**

**tam = 5**

**rad = 10.0**

**tsis = len(A) - 1**

**for i in range(-tam, tam + 1):**

**for j in range(-tam, tam + 1):**

**x\_0 = (i / tam) \* rad**

**y\_0 = (j / tam) \* rad**

**r = np.sqrt((x\_0 \* x\_0) + (y\_0 \* y\_0))**

**if r < rad:**

**tet = 0.0**

**pSource\_0 = [x\_0, y\_0, 0.0]**

**dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]**

**W = 0.4**

**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()**

**kn.display2d(Doblete, Rayos, 0)**

|  |
| --- |
|  |
|  |

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

# Example - Doublet Lens NonSec

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**Examp Doublet Lens NonSec"""**

**import numpy as np**

**import Kraken as kn**

**P\_Obj = kn.surf()**

**P\_Obj.Rc = 0.0**

**P\_Obj.Thickness = 0**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 30.0**

**P\_Obj2 = kn.surf()**

**P\_Obj2.Rc = 0.0**

**P\_Obj2.Thickness = 10**

**P\_Obj2.Glass = "AIR"**

**P\_Obj2.Diameter = 100.0**

**L1a = kn.surf()**

**L1a.Rc = 9.284706570002484E+001**

**L1a.Thickness = 6.0**

**L1a.Glass = "BK7"**

**L1a.Diameter = 30.0**

**L1a.Axicon = 0**

**L1b = kn.surf()**

**L1b.Rc = -3.071608670000159E+001**

**L1b.Thickness = 3.0**

**L1b.Glass = "F2"**

**L1b.Diameter = 30**

**L1c = kn.surf()**

**L1c.Rc = -7.819730726078505E+001**

**L1c.Thickness = 9.737604742910693E+001**

**L1c.Glass = "AIR"**

**L1c.Diameter = 30**

**P\_Ima = kn.surf()**

**P\_Ima.Rc = 0.0**

**P\_Ima.Thickness = 0.0**

**P\_Ima.Glass = "MIRROR"**

**P\_Ima.Diameter = 30.0**

**P\_Ima.Name = "Plano imagen"**

**P\_Ima.DespZ = 10**

**P\_Ima.TiltX = 6.**

**A = [P\_Obj, P\_Obj2, L1a, L1b, L1c, P\_Ima]**

**configuracion\_1 = kn.Kraken\_setup()**

**Doblete = kn.system(A, configuracion\_1)**

**Rayos = kn.raykeeper(Doblete)**

**tam = 10**

**rad = 14.0**

**tsis = len(A) - 1**

**for nsc in range(0, 100):**

**for j in range(-tam, tam + 1):**

**x\_0 = (0 / tam) \* rad**

**y\_0 = (j / tam) \* rad**

**r = np.sqrt((x\_0 \* x\_0) + (y\_0 \* y\_0))**

**if r < rad:**

**tet = 0.0**

**pSource\_0 = [x\_0, y\_0, 0.0]**

**dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]**

**W = 0.4**

**Doblete.NsTrace(pSource\_0, dCos, W)**

**Rayos.push()**

**kn.display2d(Doblete, Rayos, 0)**

|  |
| --- |
|  |
|  |

Figure 14. 2D and 3D visualization of a non-sequential ray tracing, some rays are reflected back depending on the value of the Fresnel coefficients, these depend on the wavelength, the material and the angle of the ray with respect to the normal of the surface at the point of intersection.

# Example - Doublet Lens Zernike

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**"""Doublet Lens Zernike"""**

**import Kraken as kn**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import time**

**P\_Obj = kn.surf()**

**P\_Obj.Rc = 0.0**

**P\_Obj.Thickness = 1**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 30.0**

**L1a = kn.surf()**

**L1a.Rc = 5.513435044607768E+001**

**L1a.Thickness = 6.0**

**L1a.Glass = "BK7"**

**L1a.Diameter = 30.0**

**L1b = kn.surf()**

**L1b.Rc = -4.408716526030626E+001**

**L1b.Thickness = 3.0**

**L1b.Glass = "F2"**

**L1b.Diameter = 30**

**L1c = kn.surf()**

**L1c.Rc = -2.246906271406796E+002**

**L1c.Thickness = 9.737871661422000E+001**

**L1c.Glass = "AIR"**

**L1c.Diameter = 30**

**Z = np.zeros(36)**

**Z[8] = 0.5**

**Z[9] = 0.5**

**Z[10] = 0.5**

**Z[11] = 0.5**

**Z[12] = 0.5**

**Z[13] = 0.5**

**Z[14] = 0.5**

**Z[15] = 0.5**

**L1c.ZNK = Z**

**P\_Ima = kn.surf()**

**P\_Ima.Rc = 0.0**

**P\_Ima.Thickness = 0.0**

**P\_Ima.Glass = "AIR"**

**P\_Ima.Diameter = 100.0**

**P\_Ima.Name = "Plano imagen"**

**A = [P\_Obj, L1a, L1b, L1c, P\_Ima]**

**configuracion\_1 = kn.Kraken\_setup()**

**Doblete = kn.system(A, configuracion\_1)**

**Rayos = kn.raykeeper(Doblete)**

**tam = 5**

**rad = 12.0**

**tsis = len(A) - 1**

**for i in range(-tam, tam + 1):**

**for j in range(-tam, tam + 1):**

**x\_0 = (i / tam) \* rad**

**y\_0 = (j / tam) \* rad**

**r = np.sqrt((x\_0 \* x\_0) + (y\_0 \* y\_0))**

**if r < rad:**

**tet = 0.0**

**pSource\_0 = [x\_0, y\_0, 0.0]**

**dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]**

**W = 0.4**

**Doblete.Trace(pSource\_0, dCos, W)**

**Rayos.push()**

**kn.display3d(Doblete, Rayos, 2)**

**kn.display2d(Doblete, Rayos, 0)**

|  |
| --- |
|  |

Figure 15. Example of a lens with a surface defined by coefficients of the Zernike polynomials.

# Example - Doublet Lens Tilt nonSec

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**Examp Doublet Lens Tilt nonSec"""**

**import numpy as np**

**import Kraken as kn**

**P\_Obj = kn.surf()**

**P\_Obj.Rc = 0.0**

**P\_Obj.Thickness = 10**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 30.0**

**L1a = kn.surf()**

**L1a.Rc = 9.284706570002484E+001**

**L1a.Thickness = 6.0**

**L1a.Glass = "BK7"**

**L1a.Diameter = 30.0**

**L1a.AxisMove = 1**

**L1a.TiltX = 13.0**

**L1a.DespZ = 5.0**

**L1b = kn.surf()**

**L1b.Rc = (-3.071608670000159E+001)**

**L1b.Thickness = 3.0**

**L1b.Glass = "F2"**

**L1b.Diameter = 30**

**L1c = kn.surf()**

**L1c.Rc = (-7.819730726078505E+001)**

**L1c.Thickness = 9.737604742910693E+001**

**L1c.Glass = "AIR"**

**L1c.Diameter = 30**

**P\_Ima = kn.surf()**

**P\_Ima.Rc = 0.0**

**P\_Ima.Thickness = 0.0**

**P\_Ima.Glass = "AIR"**

**P\_Ima.Diameter = 1000.0**

**P\_Ima.Name = "Plano imagen"**

**A = [P\_Obj, L1a, L1b, L1c, P\_Obj, L1a, L1b, L1c, P\_Ima]**

**configuracion\_1 = kn.Kraken\_setup()**

**Doblete = kn.system(A, configuracion\_1)**

**Rayos = kn.raykeeper(Doblete)**

**tam = 30**

**rad = 18.0**

**tsis = len(A) - 1**

**for j in range(-tam, tam + 1):**

**i = 0**

**x\_0 = (i / tam) \* rad**

**y\_0 = (j / tam) \* rad**

**r = np.sqrt((x\_0 \* x\_0) + (y\_0 \* y\_0))**

**if r < rad:**

**tet = 0.0**

**pSource\_0 = [x\_0, y\_0, 0.0]**

**dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]**

**W = 0.4**

**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()**

**kn.display3d(Doblete, Rayos, 2)**

|  |
| --- |
|  |

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

# Example - Doublet Lens Pupil

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**Examp Doublet Lens Pupil"""**

**import Kraken as kn**

**P\_Obj = kn.surf()**

**P\_Obj.Rc = 0.0**

**P\_Obj.Thickness = 100**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 30.0**

**P\_Obj.Name = "P Obj"**

**L1a = kn.surf()**

**L1a.Rc = 9.284706570002484E+001**

**L1a.Thickness = 6.0**

**L1a.Glass = "BK7"**

**L1a.Diameter = 30.0**

**L1a.Axicon = 0**

**L1b = kn.surf()**

**L1b.Rc = -3.071608670000159E+001**

**L1b.Thickness = 3.0**

**L1b.Glass = "F2"**

**L1b.Diameter = 30**

**L1c = kn.surf()**

**L1c.Rc = -7.819730726078505E+001**

**L1c.Thickness = 9.737604742910693E+001 - 40**

**L1c.Glass = "AIR"**

**L1c.Diameter = 30**

**pupila = kn.surf()**

**pupila.Rc = 30**

**pupila.Thickness = 40.**

**pupila.Glass = "AIR"**

**pupila.Diameter = 5**

**pupila.Name = "Ap Stop"**

**pupila.DespY = 0.**

**P\_Ima = kn.surf()**

**P\_Ima.Rc = 0.0**

**P\_Ima.Thickness = 0.0**

**P\_Ima.Glass = "AIR"**

**P\_Ima.Diameter = 20.0**

**P\_Ima.Name = "Plano imagen"**

**A = [P\_Obj, L1a, L1b, L1c, pupila, P\_Ima]**

**config\_1 = kn.Kraken\_setup()**

**Doblete = kn.system(A, config\_1)**

**Rayos = kn.raykeeper(Doblete)**

**W = 0.4**

**Surf= 4**

**AperVal = 10**

**AperType = "EPD"**

**Pup = kn.PupilCalc(Doblete, sup, W, AperType, AperVal)**

**print("Radio pupila de entrada: ")**

**print(Pup.RadPupInp)**

**print("Posición pupila de entrada: ")**

**print(Pup.PosPupInp)**

**print("Rádio pupila de salida: ")**

**print(Pup.RadPupOut)**

**print("Posicion pupila de salida: ")**

**print(Pup.PosPupOut)**

**print("Posicion pupila de salida respecto al plano focal: ")**

**print(Pup.PosPupOutFoc)**

**print("Orientación pupila de salida")**

**print(Pup.DirPupSal)**

**[L, M, N] = Pup.DirPupSal**

**print(L, M, N)**

**Pup.Samp = 5**

**Pup.Ptype = "fan"**

**Pup.FieldType = "angle"**

**Pup.FieldY = 2.0**

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

**for i in range(0, len(x)):**

**pSource\_0 = [x[i], y[i], z[i]]**

**dCos = [L[i], M[i], N[i]]**

**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()**

**kn.display2d(Doblete, Rayos, 0)**

|  |
| --- |
|  |

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

# Example - Doublet Lens Commands System

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**Examp Doublet Lens Commands System"""**

**import numpy as np**

**import Kraken as kn**

**P\_Obj = kn.surf()**

**P\_Obj.Rc = 0.0**

**P\_Obj.Thickness = 0.1**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 30.**

**P\_Obj2 = kn.surf()**

**P\_Obj2.Rc = 0.0**

**P\_Obj2.Thickness = 10**

**P\_Obj2.Glass = "AIR"**

**P\_Obj2.Diameter = 30.0**

**L1a = kn.surf()**

**L1a.Rc = 9.284706570002484E+001**

**L1a.Thickness = 6.0**

**L1a.Glass = "BK7"**

**L1a.Diameter = 30.0**

**L1a.Axicon = 0**

**L1b = kn.surf()**

**L1b.Rc = -3.071608670000159E+001**

**L1b.Thickness = 3.0**

**L1b.Glass = "F2"**

**L1b.Diameter = 30**

**L1c = kn.surf()**

**L1c.Rc = -7.819730726078505E+001**

**L1c.Thickness = 9.737604742910693E+001**

**L1c.Glass = "AIR"**

**L1c.Diameter = 30**

**P\_Ima = kn.surf()**

**P\_Ima.Rc = 0.0**

**P\_Ima.Thickness = 0.0**

**P\_Ima.Glass = "AIR"**

**P\_Ima.Diameter = 18.0**

**P\_Ima.Name = "Plano imagen"**

**A = [P\_Obj, P\_Obj2, L1a, L1b, L1c, P\_Ima]**

**configuracion\_1 = kn.Kraken\_setup()**

**Doblete = kn.system(A, configuracion\_1)**

**Rayos = kn.raykeeper(Doblete)**

**pSource\_0 = [0, 14, 0]**

**tet = 0.1**

**dCos = [0.0, np.sin(np.deg2rad(tet)), -np.cos(np.deg2rad(tet))]**

**W = 0.4**

**Doblete.Trace(pSource\_0, dCos, W)**

**Rayos.push()**

**kn.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)**

# Example - Doublet Lens Pupil Seidel

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**Examp Doublet Lens Pupil Seidel"""**

**import Kraken as kn**

**import numpy as np**

**P\_Obj = kn.surf()**

**P\_Obj.Rc = 0.0**

**P\_Obj.Thickness = 100**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 30.0**

**P\_Obj.Name = "P Obj"**

**L1a = kn.surf()**

**L1a.Rc = 9.284706570002484E+001**

**L1a.Thickness = 6.0**

**L1a.Glass = "N-BK7"**

**L1a.Diameter = 30.0**

**L1a.Axicon = 0**

**L1b = kn.surf()**

**L1b.Rc = -3.071608670000159E+001**

**L1b.Thickness = 3.0**

**L1b.Glass = "F2"**

**L1b.Diameter = 30**

**L1c = kn.surf()**

**L1c.Rc = -7.819730726078505E+001**

**L1c.Thickness = 9.737604742910693E+001 - 40**

**L1c.Glass = "AIR"**

**L1c.Diameter = 30**

**pupila = kn.surf()**

**pupila.Rc = 0**

**pupila.Thickness = 40.**

**pupila.Glass = "AIR"**

**pupila.Diameter = 15.0**

**pupila.Name = "Ap Stop"**

**P\_Ima = kn.surf()**

**P\_Ima.Rc = 0.0**

**P\_Ima.Thickness = 0.0**

**P\_Ima.Glass = "AIR"**

**P\_Ima.Diameter = 20.0**

**P\_Ima.Name = "Plano imagen"**

**A = [P\_Obj, L1a, L1b, L1c, pupila, P\_Ima]**

**config\_1 = kn.Kraken\_setup()**

**Doblete = kn.system(A, config\_1)**

**W = 0.6**

**Surf= 4**

**AperVal = 3**

**AperType = "EPD"**

**field = 3.25**

**fieldType = "angle"**

**AB = kn.Seidel(Doblete, sup, W, AperType, AperVal, field, fieldType)**

**print( AB[0][0])**

**print(np.sum(AB[1][0]), np.sum(AB[1][1]), np.sum(AB[1][2]), np.sum(AB[1][3]), np.sum(AB[1][4]))**

**j=1**

**print( AB[0][0+j])**

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

**j=2**

**print( AB[0][0+j])**

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

**j=3**

**print( AB[0][0+j])**

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

**Pup = kn.PupilCalc(Doblete, sup, W, AperType, AperVal)**

**Pup.Samp = 25**

**Pup.Ptype = "fan"**

**Pup.FieldY = field**

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

**Rayos = kn.raykeeper(Doblete)**

**for i in range(0, len(x)):**

**pSource\_0 = [x[i], y[i], z[i]]**

**dCos = [L[i], M[i], N[i]]**

**Doblete.Trace(pSource\_0, dCos, W)**

**Rayos.push()**

**kn.display2d(Doblete, Rayos, 0)**

|  |
| --- |
|  |
|  |

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

# Example - Doublet Lens Cylinder

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**Examp Doublet Lens Cylinder"""**

**import numpy as np**

**import Kraken as kn**

**P\_Obj = kn.surf()**

**P\_Obj.Rc = 0.0**

**P\_Obj.Thickness = 10**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 30.0**

**L1a = kn.surf()**

**L1a.Rc = 9.284706570002484E+001**

**L1a.Thickness = 6.0**

**L1a.Glass = "BK7"**

**L1a.Diameter = 30.0**

**L1b = kn.surf()**

**L1b.Rc = -3.071608670000159E+001**

**L1b.Thickness = 3.0**

**L1b.Glass = "F2"**

**L1b.Diameter = 30**

**L1b.TiltZ = 30**

**L1b.AxisMove = 0**

**L1c = kn.surf()**

**L1c.Rc = -7.819730726078505E+001**

**L1c.Thickness = 9.737604742910693E+001**

**L1c.Glass = "AIR"**

**L1c.Diameter = 30**

**L1c.Cylinder\_Rxy\_Ratio = 0**

**L1c.TiltZ = 45**

**L1c.AxisMove = 0**

**P\_Ima = kn.surf()**

**P\_Ima.Rc = 0.0**

**P\_Ima.Thickness = 0.0**

**P\_Ima.Glass = "AIR"**

**P\_Ima.Diameter = 100.0**

**P\_Ima.Name = "Plano imagen"**

**A = [P\_Obj, L1a, L1b, L1c, P\_Ima]**

**configuracion\_1 = kn.Kraken\_setup()**

**Doblete = kn.system(A, configuracion\_1)**

**Rayos = kn.raykeeper(Doblete)**

**tam = 5**

**rad = 10.0**

**tsis = len(A) - 1**

**for i in range(-tam, tam + 1):**

**for j in range(-tam, tam + 1):**

**x\_0 = (i / tam) \* rad**

**y\_0 = (j / tam) \* rad**

**r = np.sqrt((x\_0 \* x\_0) + (y\_0 \* y\_0))**

**if r < rad:**

**tet = 0.0**

**pSource\_0 = [x\_0, y\_0, 0.0]**

**dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]**

**W = 0.4**

**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()**

**kn.display3d(Doblete, Rayos, 1)**

|  |
| --- |
|  |
|  |
|  |
|  |

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

# Example - Axicon

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**Examp Axicon"""**

**import numpy as np**

**import Kraken as kn**

**P\_Obj = kn.surf()**

**P\_Obj.Rc = 0.0**

**P\_Obj.Thickness = 10**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 30.0**

**L1a = kn.surf()**

**L1a.Rc = 0**

**L1a.Thickness = 26.0**

**L1a.Glass = "BK7"**

**L1a.Diameter = 30.0**

**L1c = kn.surf()**

**L1c.Rc = 0**

**L1c.Thickness = 9.737604742910693E+001**

**L1c.Axicon = -35.0**

**L1c.ShiftY = 0**

**L1c.Glass = "AIR"**

**L1c.Diameter = 30**

**P\_Ima = kn.surf()**

**P\_Ima.Rc = 0.0**

**P\_Ima.Thickness = 0.0**

**P\_Ima.Glass = "AIR"**

**P\_Ima.Diameter = 100.0**

**P\_Ima.Name = "Plano imagen"**

**configuracion\_1 = kn.Kraken\_setup()**

**A = [P\_Obj, L1a, L1c, P\_Ima]**

**Doblete = kn.system(A, configuracion\_1)**

**Rayos = kn.raykeeper(Doblete)**

**tam = 5**

**rad = 10.0**

**tsis = len(A) - 1**

**for i in range(-tam, tam + 1):**

**for j in range(-tam, tam + 1):**

**x\_0 = (i / tam) \* rad**

**y\_0 = (j / tam) \* rad**

**r = np.sqrt((x\_0 \* x\_0) + (y\_0 \* y\_0))**

**if r < rad:**

**tet = 0.0**

**pSource\_0 = [x\_0, y\_0, 0.0]**

**dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]**

**W = 0.4**

**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()**

**kn.display3d(Doblete, Rayos, 2)**

|  |  |
| --- | --- |
|  |  |

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

# Example - Axicon and Cylinder

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**Examp Axicon and Cylinder"""**

**import numpy as np**

**import Kraken as kn**

**configuracion\_1 = kn.Kraken\_setup()**

**P\_Obj = kn.surf()**

**P\_Obj.Rc = 0.0**

**P\_Obj.Thickness = 10**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 30.0**

**L1a = kn.surf()**

**L1a.Rc = 0**

**L1a.Thickness = 26.0**

**L1a.Glass = "BK7"**

**L1a.Diameter = 30.0**

**L1c = kn.surf()**

**L1c.Rc = 0.**

**L1c.K = -1**

**L1c.Thickness = 9.737604742910693E+001**

**L1c.Axicon = (-35.0)**

**L1c.ShiftY = 0**

**L1c.Cylinder\_Rxy\_Ratio = 0**

**L1c.Glass = "AIR"**

**L1c.Diameter = 30**

**P\_Ima = kn.surf()**

**P\_Ima.Rc = 0.0**

**P\_Ima.Thickness = 0.0**

**P\_Ima.Glass = "AIR"**

**P\_Ima.Diameter = 100.0**

**P\_Ima.Name = "Plano imagen"**

**A = [P\_Obj, L1a, L1c, P\_Ima]**

**Doblete = kn.system(A, configuracion\_1)**

**Rayos = kn.raykeeper(Doblete)**

**tam = 5**

**rad = 10.0**

**tsis = len(A) - 1**

**for i in range(-tam, tam + 1):**

**for j in range(-tam, tam + 1):**

**x\_0 = (i / tam) \* rad**

**y\_0 = (j / tam) \* rad**

**r = np.sqrt((x\_0 \* x\_0) + (y\_0 \* y\_0))**

**if r < rad:**

**tet = 0.0**

**pSource\_0 = [x\_0, y\_0, 0.0]**

**dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]**

**W = 0.4**

**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()**

**kn.display3d(Doblete, Rayos, 0)**

|  |
| --- |
|  |
|  |

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

# Example - Flat Mirror 45 Deg

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**Examp Flat Mirror 45 Deg"""**

**import time**

**import matplotlib.pyplot as plt**

**import numpy as np**

**import Kraken as kn**

**start\_time = time.time()**

**P\_Obj = kn.surf()**

**P\_Obj.Rc = 0.0**

**P\_Obj.Thickness = 10**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 30.0**

**L1a = kn.surf()**

**L1a.Rc = 9.284706570002484E+001**

**L1a.Thickness = 6.0**

**L1a.Glass = "BK7"**

**L1a.Diameter = 30.0**

**L1a.Axicon = 0**

**L1b = kn.surf()**

**L1b.Rc = -3.071608670000159E+001**

**L1b.Thickness = 3.0**

**L1b.Glass = "F2"**

**L1b.Diameter = 30**

**POS\_ESP = -40**

**L1c = kn.surf()**

**L1c.Rc = -7.819730726078505E+001**

**L1c.Thickness = 9.737604742910693E+001 + POS\_ESP**

**L1c.Glass = "AIR"**

**L1c.Diameter = 30**

**Esp90 = kn.surf()**

**Esp90.Thickness = POS\_ESP**

**Esp90.Glass = "MIRROR"**

**Esp90.Diameter = 30.0**

**Esp90.Name = "Espejo a 90 grados"**

**Esp90.TiltX = 45.**

**Esp90.AxisMove = 2.**

**P\_Ima = kn.surf()**

**P\_Ima.Rc = 0.0**

**P\_Ima.Thickness = 0.0**

**P\_Ima.Glass = "AIR"**

**P\_Ima.Diameter = 3.0**

**P\_Ima.Name = "Plano imagen"**

**A = [P\_Obj, L1a, L1b, L1c, Esp90, P\_Ima]**

**config\_1 = kn.Kraken\_setup()**

**Doblete = kn.system(A, config\_1)**

**Rayos1 = kn.raykeeper(Doblete)**

**Rayos2 = kn.raykeeper(Doblete)**

**Rayos3 = kn.raykeeper(Doblete)**

**RayosT = kn.raykeeper(Doblete)**

**tam = 10**

**rad = 10.0**

**tsis = len(A) - 1**

**for j in range(-tam, tam + 1):**

**for i in range(-tam, tam + 1):**

**x\_0 = (i / tam) \* rad**

**y\_0 = (j / tam) \* rad**

**r = np.sqrt((x\_0 \* x\_0) + (y\_0 \* y\_0))**

**if r < rad:**

**tet = 0.0**

**pSource\_0 = [x\_0, y\_0, 0.0]**

**dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]**

**W = 0.4**

**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()**

**kn.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()**

|  |  |
| --- | --- |
|  |  |

Figure 22. (Izquierda) Example of a diagonal mirror with a 45 ° rotation, (Right) Blob diagram for three different wavelengths, 0.4 µm, 0.5 µm and 0.6 µm.

# Example - Parabole Mirror Shift

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**Examp Parabole Mirror Shift"""**

**import numpy as np**

**import Kraken as kn**

**P\_Obj = kn.surf()**

**P\_Obj.Thickness = 1000.0**

**P\_Obj.Diameter = 300**

**P\_Obj.Drawing = 0**

**M1 = kn.surf()**

**M1.Rc = -2000.0**

**M1.Thickness = M1.Rc / 2**

**M1.k = -1.0**

**M1.Glass = "MIRROR"**

**M1.Diameter = 300**

**M1.ShiftY = 200**

**P\_Ima = kn.surf()**

**P\_Ima.Glass = "AIR"**

**P\_Ima.Diameter = 1600.0**

**P\_Ima.Drawing = 0**

**P\_Ima.Name = "Plano imagen"**

**A = [P\_Obj, M1, P\_Ima]**

**configuracion\_1 = kn.Kraken\_setup()**

**Espejo = kn.system(A, configuracion\_1)**

**Rayos = kn.raykeeper(Espejo)**

**tam = 5**

**rad = 150.0**

**tsis = len(A) - 1**

**for i in range(-tam, tam + 1):**

**for j in range(-tam, tam + 1):**

**x\_0 = (i / tam) \* rad**

**y\_0 = (j / tam) \* rad**

**r = np.sqrt((x\_0 \* x\_0) + (y\_0 \* y\_0))**

**if r < rad:**

**tet = 0.0**

**pSource\_0 = [x\_0, y\_0, 0.0]**

**dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]**

**W = 0.4**

**Espejo.Trace(pSource\_0, dCos, W)**

**Rayos.push()**

**kn.display3d(Espejo, Rayos, 0)**

|  |  |
| --- | --- |
|  |  |

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

# Example - Diffraction Grating Transmission

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**Examp Diffraction Grating Transmission"""**

**import numpy as np**

**import Kraken as kn**

**P\_Obj = kn.surf()**

**P\_Obj.Rc = 0.0**

**P\_Obj.Thickness = 10**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 30.0**

**Dif\_Obj\_c1 = kn.surf()**

**Dif\_Obj\_c1.Rc = 0.0**

**Dif\_Obj\_c1.Thickness = 1**

**Dif\_Obj\_c1.Glass = "BK7"**

**Dif\_Obj\_c1.Diameter = 30.0**

**Dif\_Obj\_c1.Grating\_D = 1.0**

**Dif\_Obj\_c1.Diff\_Ord = 1.**

**Dif\_Obj\_c1.Grating\_Angle = 45.**

**Dif\_Obj\_c2 = kn.surf()**

**Dif\_Obj\_c2.Rc = 0.0**

**Dif\_Obj\_c2.Thickness = 10**

**Dif\_Obj\_c2.Glass = "AIR"**

**Dif\_Obj\_c2.Diameter = 30.0**

**L1a = kn.surf()**

**L1a.Rc = 5.513435044607768E+001**

**L1a.Thickness = 6.0**

**L1a.Glass = "BK7"**

**L1a.Diameter = 30.0**

**L1b = kn.surf()**

**L1b.Rc = -4.408716526030626E+001**

**L1b.Thickness = 3.0**

**L1b.Glass = "F2"**

**L1b.Diameter = 30**

**L1c = kn.surf()**

**L1c.Rc = -2.246906271406796E+002**

**L1c.Thickness = 9.737871661422000E+001**

**L1c.Glass = "AIR"**

**L1c.Diameter = 30**

**P\_Ima = kn.surf()**

**P\_Ima.Name = "Plano imagen"**

**P\_Ima.Rc = 0.0**

**P\_Ima.Thickness = 0.0**

**P\_Ima.Glass = "AIR"**

**P\_Ima.Diameter = 300.0**

**A = [P\_Obj, Dif\_Obj\_c1, Dif\_Obj\_c2, L1a, L1b, L1c, P\_Ima]**

**configuracion\_1 = kn.Kraken\_setup()**

**Doblete = kn.system(A, configuracion\_1)**

**Rayos = kn.raykeeper(Doblete)**

**tam = 5**

**rad = 10.0**

**tsis = len(A) - 1**

**for i in range(-tam, tam + 1):**

**for j in range(-tam, tam + 1):**

**x\_0 = (i / tam) \* rad**

**y\_0 = (j / tam) \* rad**

**r = np.sqrt((x\_0 \* x\_0) + (y\_0 \* y\_0))**

**if r < rad:**

**tet = 0.0**

**pSource\_0 = [x\_0, y\_0, 0.0]**

**dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]**

**W = 0.4**

**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()**

**kn.display2d(Doblete, Rayos, 0)**

|  |  |
| --- | --- |
|  |  |

Figure 24. 3D and 2D visualization of a system that has a transmission difaction grating.

# Example - Diffraction Grating Reflection

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**Examp Diffraction Grating Reflection"""**

**import numpy as np**

**import Kraken as kn**

**P\_Obj = kn.surf()**

**P\_Obj.Rc = 0.0**

**P\_Obj.Thickness = 10**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 30.0**

**L1a = kn.surf()**

**L1a.Rc = 5.513435044607768E+001**

**L1a.Thickness = 6.0**

**L1a.Glass = "BK7"**

**L1a.Diameter = 30.0**

**L1b = kn.surf()**

**L1b.Rc = -4.408716526030626E+001**

**L1b.Thickness = 3.0**

**L1b.Glass = "F2"**

**L1b.Diameter = 30**

**L1c = kn.surf()**

**L1c.Rc = -2.246906271406796E+002**

**L1c.Thickness = 9.737871661422000E+001 - 50.0**

**L1c.Glass = "AIR"**

**L1c.Diameter = 30**

**Dif\_Obj = kn.surf()**

**Dif\_Obj.Rc = 0.0**

**Dif\_Obj.Thickness = -50**

**Dif\_Obj.Glass = "MIRROR"**

**Dif\_Obj.Diameter = 30.0**

**Dif\_Obj.Grating\_D = 1.0**

**Dif\_Obj.Diff\_Ord = 1**

**Dif\_Obj.Grating\_Angle = 45.0**

**Dif\_Obj.Surface\_type = 1**

**P\_Ima = kn.surf()**

**P\_Ima.Rc = 0.0**

**P\_Ima.Name = "Plano imagen"**

**P\_Ima.Thickness = 0.0**

**P\_Ima.Glass = "AIR"**

**P\_Ima.Diameter = 300.0**

**P\_Ima.Drawing = 0**

**A = [P\_Obj, L1a, L1b, L1c, Dif\_Obj, P\_Ima]**

**configuracion\_1 = kn.Kraken\_setup()**

**Doblete = kn.system(A, configuracion\_1)**

**Rayos = kn.raykeeper(Doblete)**

**tam = 5**

**rad = 10.0**

**tsis = len(A) - 1**

**for i in range(-tam, tam + 1):**

**for j in range(-tam, tam + 1):**

**x\_0 = (i / tam) \* rad**

**y\_0 = (j / tam) \* rad**

**r = np.sqrt((x\_0 \* x\_0) + (y\_0 \* y\_0))**

**if r < rad:**

**tet = 0.0**

**pSource\_0 = [x\_0, y\_0, 0.0]**

**dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]**

**W = 0.4**

**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()**

**kn.display2d(Doblete, Rayos, 1)**

|  |
| --- |
|  |

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

# Example - Tel 2M Spyder Spot Diagram

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**Examp Tel 2M Spyder Spot Diagram"""**

**import os**

**import numpy as np**

**import Kraken as kn**

**P\_Obj = kn.surf()**

**P\_Obj.Thickness = 2.000000000000000E+003**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 6.796727741707513E+002 \* 2.0**

**P\_Obj.Drawing = 0**

**M1 = kn.surf()**

**M1.Rc = -6.06044E+003**

**M1.Thickness = -1.774190000000000E+003 + 1.853722901194000E+000**

**M1.k = -1.637E+000**

**M1.Glass = "MIRROR"**

**M1.Diameter = 6.63448E+002 \* 2.0**

**M1.InDiameter = 228.6 \* 2.0**

**M1.DespY = 0.0**

**M1.TiltX = 0.0000**

**M1.AxisMove = 1**

**M2 = kn.surf()**

**M2.Rc = -6.06044E+003**

**M2.Thickness = -M1.Thickness**

**M2.k = -3.5782E+001**

**M2.Glass = "MIRROR"**

**M2.Diameter = 2.995730651164167E+002 \* 2.0**

**ED0 = np.zeros(20)**

**ED0[2] = 4.458178314555000E-018**

**M2.AspherData = ED0**

**Vertex = kn.surf()**

**Vertex.Thickness = 130.0**

**Vertex.Glass = "AIR"**

**Vertex.Diameter = 600.0**

**Vertex.Drawing = 0**

**currentDirectiony = os.getcwd()**

**direc = r"Prisma.stl"**

**objeto = kn.surf()**

**objeto.Diameter = 118.0 \* 2.0**

**objeto.Solid\_3d\_stl = direc**

**objeto.Thickness = 600**

**objeto.Glass = "BK7"**

**objeto.TiltX = 55**

**objeto.TiltY = 0**

**objeto.TiltZ = 45**

**objeto.DespX = 0**

**objeto.DespY = 0**

**objeto.AxisMove = 0**

**P\_Ima = kn.surf()**

**P\_Ima.Rc = 0**

**P\_Ima.Thickness = 100.0**

**P\_Ima.Glass = "BK7"**

**P\_Ima.Diameter = 500.0**

**P\_Ima.Drawing = 1**

**A = [P\_Obj, M1, M2, Vertex, objeto, P\_Ima]**

**configuracion\_1 = kn.Kraken\_setup()**

**Telescope = kn.system(A, configuracion\_1)**

**Rays = kn.raykeeper(Telescope)**

**W = 0.633**

**tam = 5**

**rad = 6.56727741707513E+002**

**tsis = len(A) + 2**

**for gg in range(0, 10):**

**for j in range(-tam, tam + 1):**

***# j=0***

**for i in range(-tam, tam + 1):**

**x\_0 = (i / tam) \* rad**

**y\_0 = (j / tam) \* rad**

**r = np.sqrt((x\_0 \* x\_0) + (y\_0 \* y\_0))**

**if r < rad:**

**tet = 0.0**

**pSource\_0 = [x\_0, y\_0, 0.0]**

***# print("-...............")***

**dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]**

**W = 0.633**

**Telescope.NsTrace(pSource\_0, dCos, W)**

**Rays.push()**

**kn.display3d(Telescope, Rays, 0)**

**print(Telescope.EFFL)**

|  |
| --- |
|  |
|  |

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

# Example - Tel 2M Spyder Spot Tilt M2

***# Rays.push()***

***#kn.display3d(Telescope, Rays, 0)***

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**Examp Tel 2M Spyder Spot Tilt M2"""**

**import matplotlib.pyplot as plt**

**import numpy as np**

**import pyvista as pv**

**import Kraken as kn**

**P\_Obj = kn.surf()**

**P\_Obj.Rc = 0**

**P\_Obj.Thickness = 1000**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 1.059E+003 \* 2.0**

**Spider = kn.surf()**

**Spider.Rc = 999999999999.0**

**Spider.Thickness = 3.452229924716749E+003 + 100.0**

**Spider.Glass = "AIR"**

**Spider.Diameter = 1.059E+003 \* 2.0**

**plane1 = pv.Plane(center=[0, 0, 0], direction=[0, 0, 1], i\_size=30, j\_size=2100, i\_resolution=10, j\_resolution=10)**

**plane2 = pv.Plane(center=[0, 0, 0], direction=[0, 0, 1], i\_size=2100, j\_size=30, i\_resolution=10, j\_resolution=10)**

**Baffle1 = pv.Disc(center=[0.0, 0.0, 0.0], inner=0, outer=875 / 2.0, normal=[0, 0, 1], r\_res=1, c\_res=100)**

**Baffle2 = Baffle1.boolean\_add(plane1)**

**Baffle3 = Baffle2.boolean\_add(plane2)**

**AAA = pv.MultiBlock()**

**AAA.append(plane1)**

**AAA.append(plane2)**

**AAA.append(Baffle1)**

**Spider.Mask\_Shape = AAA**

**Spider.Mask\_Type = 2**

**Spider.TiltZ = 0**

**Thickness = 3.452200000000000E+003**

**M1 = kn.surf()**

**M1.Rc = -9.638000000004009E+003**

**M1.Thickness = -Thickness**

**M1.k = -1.077310000000000E+000**

**M1.Glass = "MIRROR"**

**M1.Diameter = 1.059E+003 \* 2.0**

**M1.InDiameter = 250 \* 2.0**

**M2 = kn.surf()**

**M2.Rc = -3.93E+003**

**M2.Thickness = Thickness + 1.037535322418897E+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 = kn.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 = kn.Kraken\_setup()**

**Telescopio = kn.system(A, configuracion\_1)**

**Rayos = kn.raykeeper(Telescopio)**

**tam = 7**

**rad = 2200 / 2**

**tsis = len(A) - 1**

**for i in range(-tam, tam + 1):**

**for j in range(-tam, tam + 1):**

**x\_0 = (i / tam) \* rad**

**y\_0 = (j / tam) \* rad**

**r = np.sqrt((x\_0 \* x\_0) + (y\_0 \* y\_0))**

**if r < rad:**

**tet = 0.0**

**pSource\_0 = [x\_0, y\_0, 0.0]**

**dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]**

**W = 0.4**

**Telescopio.Trace(pSource\_0, dCos, W)**

**Rayos.push()**

**kn.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()**

|  |
| --- |
|  |
|  |

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

# Example - Tel 2M Pupila

***# !/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**Examp TEl 2M Pupila"""**

**import matplotlib.pyplot as plt**

**import numpy as np**

**import Kraken as kn**

**P\_Obj = kn.surf()**

**P\_Obj.Rc = 0**

**P\_Obj.Thickness = 1000 + 3.452200000000000E+003**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 1.059E+003 \* 2.0**

**Thickness = 3.452200000000000E+003**

**M1 = kn.surf()**

**M1.Rc = -9.638000000004009E+003**

**M1.Thickness = -Thickness**

**M1.k = -1.077310000000000E+000**

**M1.Glass = "MIRROR"**

**M1.Diameter = 1.059E+003 \* 2.0**

**M1.InDiameter = 250 \* 2.0**

**M2 = kn.surf()**

**M2.Rc = -3.93E+003**

**M2.Thickness = Thickness + 1.037525880125084E+003**

**M2.k = -4.328100000000000E+000**

**M2.Glass = "MIRROR"**

**M2.Diameter = 3.365E+002 \* 2.0**

**M2.TiltY = 0.1**

**M2.TiltX = 0.1**

**M2.AxisMove = 0**

**P\_Ima = kn.surf()**

**P\_Ima.Diameter = 300.0**

**P\_Ima.Glass = "AIR"**

**P\_Ima.Name = "Plano imagen"**

**A = [P\_Obj, M1, M2, P\_Ima]**

**configuracion\_1 = kn.Kraken\_setup()**

**Telescopio = kn.system(A, configuracion\_1)**

**W = 0.4**

**Surf= 1**

**AperVal = 2010**

**AperType = "EPD" *# "STOP"***

**Pup = kn.PupilCalc(Telescopio, sup, W, AperType, AperVal)**

**print("Radio pupila de entrada: ")**

**print(Pup.RadPupInp)**

**print("Posicion pupila de entrada: ")**

**print(Pup.PosPupInp)**

**print("Radio pupila de salida: ")**

**print(Pup.RadPupOut)**

**print("Posicion pupila de salida: ")**

**print(Pup.PosPupOut)**

**print("Posicion pupila de salida respecto al plano focal: ")**

**print(Pup.PosPupOutFoc)**

**print("Orientación pupila de salida")**

**print(Pup.DirPupSal)**

**[L, M, N] = Pup.DirPupSal**

**print(L, M, N)**

**TetX = np.rad2deg(np.arcsin(-M))**

**TetY = np.rad2deg(np.arcsin(L / np.cos(np.arcsin(-M))))**

**print(TetX, TetY)**

**print("---------------------------------------------------------------")**

**Pup.Samp = 10**

**Pup.Ptype = "hexapolar"**

**Pup.FieldY = 0.0**

**Pup.FieldType = "angle"**

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

**Rayos = kn.raykeeper(Telescopio)**

**for i in range(0, len(x)):**

**pSource\_0 = [x[i], y[i], z[i]]**

**dCos = [L[i], M[i], N[i]]**

**W = 0.4**

**Telescopio.Trace(pSource\_0, dCos, W)**

**Rayos.push()**

**kn.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)**

|  |  |
| --- | --- |
|  |  |

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

# Example - Tel 2M Error Map

***# -\*- coding: utf-8 -\*-***

**Examp Tel 2M Error Map"""**

**import matplotlib.pyplot as plt**

**import numpy as np**

**import Kraken as kn**

**import time**

**def ErrorGen():**

**L = 1000.**

**N = 20.**

**hight = 0.001**

**SPACE = 2 \* L / N**

**x = np.arange(-L, L + SPACE, SPACE)**

**y = np.arange(-L, L + SPACE, SPACE)**

**gx, gy = np.meshgrid(x, y)**

**R = np.sqrt((gx \* gx) + (gy \* gy))**

**arg = np.argwhere(R < L)**

**Npoints = np.shape(arg)[0]**

**X = np.zeros(Npoints)**

**Y = np.zeros(Npoints)**

**i = 0**

**for [a, b] in arg:**

**X[i] = gx[a, b]**

**Y[i] = gy[a, b]**

**i = i + 1**

**spa = 10000000**

**Z = hight \* (np.random.randint(-spa, spa, Npoints)) / (spa \* 2.0)**

**return [X, Y, Z, SPACE]**

**P\_Obj = kn.surf()**

**P\_Obj.Rc = 0**

**P\_Obj.Thickness = 3500**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 1.059E+003 \* 2.0**

**Thickness = 3.452200000000000E+003**

**M1 = kn.surf()**

**M1.Rc = -9.638000000004009E+003**

**M1.Thickness = -Thickness**

**M1.k = -1.077310000000000E+000**

**M1.Glass = "MIRROR"**

**M1.Diameter = 1.059E+003 \* 2.0**

**M1.InDiameter = 250 \* 2.0**

**M1.Error\_map = ErrorGen()**

**M2 = kn.surf()**

**M2.Rc = -3.93E+003**

**M2.Thickness = Thickness + 1.037525880125084E+003**

**M2.k = -4.328100000000000E+000**

**M2.Glass = "MIRROR"**

**M2.Diameter = 3.365E+002 \* 2.0**

**M2.AxisMove = 0**

**P\_Ima = kn.surf()**

**P\_Ima.Diameter = 1000.0**

**P\_Ima.Glass = "AIR"**

**P\_Ima.Name = "Plano imagen"**

**A = [P\_Obj, M1, M2, P\_Ima]**

**configuracion\_1 = kn.Kraken\_setup()**

**Telescopio = kn.system(A, configuracion\_1)**

**Rayos = kn.raykeeper(Telescopio)**

**tam = 9**

**rad = 2100 / 2**

**tsis = len(A) - 1**

**start\_time = time.time()**

**for i in range(-tam, tam + 1):**

**for j in range(-tam, tam + 1):**

**x\_0 = (i / tam) \* rad**

**y\_0 = (j / tam) \* rad**

**r = np.sqrt((x\_0 \* x\_0) + (y\_0 \* y\_0))**

**if r < rad:**

**print(i)**

**tet = 0.0**

**pSource\_0 = [x\_0, y\_0, 0.0]**

**dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]**

**W = 0.4**

**Telescopio.Trace(pSource\_0, dCos, W)**

**Rayos.push()**

**print("--- %s seconds ---" % (time.time() - start\_time))**

**kn.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()**

|  |  |
| --- | --- |
|  |  |

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

# Example - Tel 2M Wavefront Fitting

***# !/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**Examp Tel 2M Wavefront Fitting"""**

**import os**

**import sys**

**import matplotlib.pyplot as plt**

**import numpy as np**

**import Kraken as kn**

**from PhaseCalc import Phase**

**currentDirectiony = os.getcwd()**

**sys.path.insert(1, currentDirectiony + '/library')**

**P\_Obj = kn.surf()**

**P\_Obj.Rc = 0**

**P\_Obj.Thickness = 1000 + 3.452200000000000E+003**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 1.059E+003 \* 2.0**

**Thickness = 3.452200000000000E+003**

**M1 = kn.surf()**

**M1.Rc = -9.638000000004009E+003**

**M1.Thickness = -Thickness**

**M1.k = -1.077310000000000E+000**

**M1.Glass = "MIRROR"**

**M1.Diameter = 1.059E+003 \* 2.0**

**M1.InDiameter = 250 \* 2.0**

**M1.TiltY = 0.0**

**M1.TiltX = 0.0**

**M1.AxisMove = 0**

**M2 = kn.surf()**

**M2.Rc = -3.93E+003**

**M2.Thickness = Thickness + 1037.525880**

**M2.k = -4.328100000000000E+000**

**M2.Glass = "MIRROR"**

**M2.Diameter = 3.365E+002 \* 2.0**

**M2.TiltY = 0.0**

**M2.TiltX = 0.0**

**M2.DespY = 0.0**

**M2.DespX = 0.0**

**M2.AxisMove = 0**

**P\_Ima = kn.surf()**

**P\_Ima.Diameter = 300.0**

**P\_Ima.Glass = "AIR"**

**P\_Ima.Name = "Plano imagen"**

**A = [P\_Obj, M1, M2, P\_Ima]**

**configuracion\_1 = kn.Kraken\_setup()**

**Telescopio = kn.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 = kn.Zernike\_Fitting(X, Y, Z, A)**

**A = np.abs(z\_coeff)**

**Zeros = np.argwhere(A > 0.0001)**

**AA = np.zeros\_like(A)**

**AA[Zeros] = 1**

**A = AA**

**z\_coeff, MatNotation, w\_rms, fitt\_error = kn.Zernike\_Fitting(X, Y, Z, A)**

**print("Peak to valley: ", P2V)**

**for i in range(0, NC):**

**print("z ", i + 1, " ", "{0:.6f}".format(float(z\_coeff[i])), " : ", MatNotation[i])**

**print("RMS: ", "{:.4f}".format(float(w\_rms)), " Error del ajuste: ", fitt\_error)**

**z\_coeff[0] = 0**

**print("RMS to chief: ", np.sqrt(np.sum(z\_coeff \* z\_coeff)))**

**z\_coeff[1] = 0**

**z\_coeff[2] = 0**

**print("RMS to centroid: ", np.sqrt(np.sum(z\_coeff \* z\_coeff)))**

**RR = kn.raykeeper(Telescopio)**

**Pup = kn.PupilCalc(Telescopio, sup, W, AperType, AperVal)**

**Pup.FieldX = FieldX**

**Pup.FieldY = FieldY**

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

**for i in range(0, len(x)):**

**pSource\_0 = [x[i], y[i], z[i]]**

**dCos = [L[i], M[i], N[i]]**

**Telescopio.Trace(pSource\_0, dCos, W)**

**RR.push()**

**kn.display3d(Telescopio, RR, 2)**

**X, Y, Z, L, M, N = RR.pick(-1)**

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

**plt.xlabel('numbers')**

**plt.ylabel('values')**

**plt.title('spot Diagram')**

**plt.axis('square')**

**plt.show()**

|  |
| --- |
|  |

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

# Ejemplo – Tel 2M-STL\_ImageSlicer.py

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**"""**

**Example - -2M-STL\_ImageSlicer.py**

**"""**

**import Kraken as kn**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import scipy**

**import os**

**A1=1**

**if A1==0:**

**P\_Obj=kn.surf()**

**P\_Obj.Rc=0**

**P\_Obj.Thickness=1000+3.452200000000000E+003**

**P\_Obj.Glass="AIR"**

**P\_Obj.Diameter=1.059E+003\*2.0**

**Thickness=3.452200000000000E+003**

**M1=kn.surf()**

**M1.Rc=-9.638000000004009E+003**

**M1.Thickness=-Thickness**

**M1.k=-1.077310000000000E+000**

**M1.Glass="MIRROR"**

**M1.Diameter=1.059E+003\*2.0**

**M1.InDiameter=250\*2.0**

**M2=kn.surf()**

**M2.Rc=-3.93E+003**

**M2.Thickness=Thickness+1.037525880125084E+003**

**M2.k=-4.328100000000000E+000**

**M2.Glass="MIRROR"**

**M2.Diameter=3.365E+002\*2.0**

**M2.AxisMove=0**

**P\_Image\_A=kn.surf()**

**P\_Image\_A.Diameter=10.0**

**P\_Image\_A.Glass="AIR"**

**P\_Image\_A.Thickness=10**

**P\_Image\_A.Name="Image plane Tel"**

**P\_Image\_A.DespZ=-100.0**

**A=[P\_Obj,M1,M2,P\_Image\_A]**

**configuracion\_1=kn.Kraken\_setup()**

**Telescopio=kn.system(A,configuracion\_1)**

**Rayos=kn.raykeeper(Telescopio)**

***# Gaussian***

**def f(x):**

**x=np.rad2deg(x)**

**seing=1.2/3600.0**

**sigma=seing/2.3548**

**mean = 0**

**standard\_deviation = sigma**

**y=scipy.stats.norm(mean, standard\_deviation)**

**res=y.pdf(x)**

**return res**

**Sun = kn.SourceRnd()**

**Sun.field=4\*1.2/(2.0\*3600.0)**

**Sun.fun = f**

**Sun.dim = 2100**

**Sun.num = 100000**

**L, M, N, X, Y, Z = Sun.rays()**

**Xr=np.zeros\_like(L)**

**Yr=np.zeros\_like(L)**

**Zr=np.zeros\_like(L)**

**Lr=np.zeros\_like(L)**

**Mr=np.zeros\_like(L)**

**Nr=np.zeros\_like(L)**

**NM=np.zeros\_like(L)**

**con=0**

**con2=0**

**W = 0.6**

**for i in range(0,Sun.num):**

**if con2==10:**

**print(100.\*i/Sun.num)**

**con2=0**

**pSource\_0 = [X[i], Y[i], Z[i]]**

**dCos = [L[i], M[i], N[i]]**

**Telescopio.Trace(pSource\_0, dCos, W)**

**x,y,z=Telescopio.XYZ[-1]**

**l,m,n=Telescopio.LMN[-1]**

**Xr[con]=x**

**Yr[con]=y**

**Zr[con]=z**

**Lr[con]=l**

**Mr[con]=m**

**Nr[con]=n**

**if Telescopio.NAME[-1]=="Image plane Tel":**

**NM[con]=i**

**else:**

**NM[con]=-1**

**con=con+1**

**con2=con2+1**

***# 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=kn.surf()**

**P\_Obj.Rc=0**

**P\_Obj.Thickness=100.+0.5**

**P\_Obj.Glass="AIR"**

**P\_Obj.Diameter=10**

**currentDirectiony = os.getcwd()**

**direc = r"Jherrera-ImageSlicerBW-00.stl"**

**P\_ImageSlicer=kn.surf()**

**P\_ImageSlicer.Diameter=10.0**

**P\_ImageSlicer.Glass="BK7"**

**P\_ImageSlicer.Name="Image slicer"**

**P\_ImageSlicer.Solid\_3d\_stl = direc**

**P\_ImageSlicer.Thickness = 13**

**P\_ImageSlicer.TiltX=180.0**

**P\_ImageSlicer.DespX=-0.55**

**P\_ImageSlicer.DespY=-0.03**

**P\_ImageSlicer.AxisMove=0**

**P\_Ima=kn.surf()**

**P\_Ima.Diameter=10.0**

**P\_Ima.Glass="AIR"**

**P\_Ima.Name="Plano imagen"**

**A=[P\_Obj, P\_ImageSlicer, P\_Ima]**

**configuracion\_1 = kn.Kraken\_setup()**

**ImageSlicer = kn.system(A,configuracion\_1)**

**Rayos=kn.raykeeper(ImageSlicer)**

**outfile="savedRays.npy"**

**R=np.load(outfile)**

**print(np.shape(R))**

**X,Y,Z,L,M,N,W=R[:,0],R[:,1],R[:,2],R[:,3],R[:,4],R[:,5],R[:,6]**

**nrays=2000**

**Xr=np.zeros(nrays)**

**Yr=np.zeros(nrays)**

**Zr=np.zeros(nrays)**

**Lr=np.zeros(nrays)**

**Mr=np.zeros(nrays)**

**Nr=np.zeros(nrays)**

**NM=np.zeros(nrays)**

**con=0**

**con2=0**

**for i in range(0,nrays):**

**if con2==10:**

**print(100.\*i/nrays)**

**con2=0**

**pSource\_0 = [X[i], Y[i], Z[i]\*0]**

**dCos = [L[i], M[i], N[i]]**

**ImageSlicer.NsTrace(pSource\_0, dCos, W[i])**

**x,y,z=ImageSlicer.XYZ[-1]**

**l,m,n=ImageSlicer.LMN[-1]**

**Xr[con]=x**

**Yr[con]=y**

**Zr[con]=z**

**Lr[con]=l**

**Mr[con]=m**

**Nr[con]=n**

**AA=ImageSlicer.SURFACE**

**AA=np.asarray(AA)**

**AW=np.argwhere(AA==1)**

**if ImageSlicer.NAME[-1]=="Plano imagen" and len(AW)<10 and ImageSlicer.TT<0.9:**

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

**NM[con]=i**

**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()***

**kn.display3d(ImageSlicer, Rayos, 0)**

|  |  |
| --- | --- |
|  |  |
|  |  |

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

# Ejemplo – Tel 2M\_Atmospheric\_Refraction\_Corrector.py

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**"""**

**Created on Sun Aug 2 12:04:14 2020**

**Example - -Tel\_2M\_Atmospheric\_Refraction\_Corrector.py**

**@author: joelherreravazquez**

**"""**

**import Kraken as kn**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import time**

**P\_Obj=kn.surf()**

**P\_Obj.Rc=0**

**P\_Obj.Thickness=1000+3.452200000000000E+003**

**P\_Obj.Glass="AIR"**

**P\_Obj.Diameter=1.059E+003\*2.0**

**Thickness=3.452200000000000E+003**

**M1=kn.surf()**

**M1.Rc=-9.638000000004009E+003**

**M1.Thickness=-Thickness**

**M1.k=-1.077310000000000E+000**

**M1.Glass="MIRROR"**

**M1.Diameter=1.059E+003\*2.0**

**M1.InDiameter=250\*2.0**

**M2=kn.surf()**

**M2.Rc=-3.93E+003**

**M2.Thickness=Thickness+1.037525880125084E+003**

**M2.k=-4.328100000000000E+000**

**M2.Glass="MIRROR"**

**M2.Diameter=3.365E+002\*2.0**

**M2.AxisMove=0**

**P\_Ima=kn.surf()**

**P\_Ima.Diameter=1000.0**

**P\_Ima.Glass="AIR"**

**P\_Ima.Name="Plano imagen"**

**A=[P\_Obj,M1,M2,P\_Ima]**

**configuracion\_1=kn.Kraken\_setup()**

**Telescopio=kn.system(A,configuracion\_1)**

**Rayos1=kn.raykeeper(Telescopio)**

**Rayos2=kn.raykeeper(Telescopio)**

**Rayos3=kn.raykeeper(Telescopio)**

**W = 0.4**

**Surf= 1**

**AperVal = 2010**

**AperType = "EPD" *# "STOP"***

**Pup = kn.PupilCalc(Telescopio, sup, W, AperType, AperVal)**

**Pup.Samp=11**

**Pup.FieldType = "angle"**

**Pup.AtmosRef = 1**

**Pup.T = 283.15 *# k***

**Pup.P = 101300 *# Pa***

**Pup.H = 0.5 *# Humidity ratio 1 to 0***

**Pup.xc = 400 *# ppm***

**Pup.lat = 31 *# degrees***

**Pup.h = 2800 *# meters***

**Pup.l1 = 0.60169 *# micron***

**Pup.l2 = 0.50169 *# micron***

**Pup.z0 = 55.0 *# degrees***

**Pup.Ptype = "hexapolar"**

**Pup.FieldX = 0.0**

**W1 = 0.50169**

**Pup.l2 = W1**

**xa,ya,za,La,Ma,Na=Pup.Pattern2Field()**

**W2 = 0.60169**

**Pup.l2 = W2**

**xb,yb,zb,Lb,Mb,Nb=Pup.Pattern2Field()**

**W3 = 0.70169**

**Pup.l2 = W3**

**xc,yc,zc,Lc,Mc,Nc=Pup.Pattern2Field()**

***############################################***

**for i in range(0,len(xa)):**

**pSource\_0 = [xa[i], ya[i], za[i]]**

**dCos=[La[i], Ma[i], Na[i]]**

**Telescopio.Trace(pSource\_0, dCos, W1)**

**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()**

***############################################***

***# kn.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()**

|  |
| --- |
|  |
|  |

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

# Example - Extra Shape Micro Lens Array

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**Examp Extra Shape Micro Lens Array"""**

**import Kraken as kn**

**import numpy as np**

**import matplotlib.pyplot as plt**

**P\_Obj = kn.surf()**

**P\_Obj.Rc = 0.0**

**P\_Obj.Thickness = 10**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 30.0**

**L1a = kn.surf()**

**L1a.Rc = 55.134\*0**

**L1a.Thickness = 2.0**

**L1a.Glass = "BK7"**

**L1a.Diameter = 30.0**

**L1c = kn.surf()**

**L1c.Thickness = 40**

**L1c.Glass = "AIR"**

**L1c.Diameter = 30**

**def f(x,y,E):**

**DeltaX=E[0]\*np.rint(x/E[0])**

**DeltaY=E[0]\*np.rint(y/E[0])**

**x=x-DeltaX**

**y=y-DeltaY**

**s = np.sqrt((x \* x) + (y \* y))**

**c = 1.0 / E[1]**

**InRoot = 1 - (E[2] + 1.0) \* c \* c \* s \* s**

**z = (c \* s \* s / (1.0 + np.sqrt(InRoot)))**

**return z**

**coef=[3.0, -3, 0]**

**L1c.ExtraData=[f, coef]**

**L1c.Res=2**

**P\_Ima = kn.surf()**

**P\_Ima.Rc = 0.0**

**P\_Ima.Thickness = 0.0**

**P\_Ima.Glass = "AIR"**

**P\_Ima.Diameter = 300.0**

**P\_Ima.Name = "Image plane"**

**A = [P\_Obj, L1a, L1c, P\_Ima]**

**Config\_1 = kn.Kraken\_setup()**

**Lens = kn.system(A, Config\_1)**

**Rays = kn.raykeeper(Lens)**

**Wav = 0.45**

**for i in range(-100, 100+1):**

**pSource = [0.0, i/10., 0.0]**

**dCos = [0.0, 0.0, 1.0]**

**Lens.Trace(pSource, dCos, Wav)**

**Rays.push()**

**kn.display3d(Lens, Rays, 1)**

**kn.display2d(Lens, Rays, 0)**

|  |
| --- |
|  |

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

# Example - Extra Shape Radial Sine

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**Examp Extra Shape Radial Sine"""**

**import Kraken as kn**

**import numpy as np**

**import matplotlib.pyplot as plt**

**P\_Obj = kn.surf()**

**P\_Obj.Rc = 0.0**

**P\_Obj.Thickness = 10**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 30.0**

**P\_Obj.Drawing = 0**

**L1a = kn.surf()**

**L1a.Rc = 55.134**

**L1a.Thickness = 9.0**

**L1a.Glass = "BK7"**

**L1a.Diameter = 30.0**

**L1c = kn.surf()**

**L1c.Rc = -224.69**

**L1c.Thickness = 40**

**L1c.Glass = "AIR"**

**L1c.Diameter = 30**

**def f(x,y,E):**

**r=np.sqrt(x\*x+y\*y)**

**r=np.asarray(r)**

**H=2.0\*np.pi\*r/E[0]**

**z=np.sin(H) \* E[1]**

**return z**

**coef = np.zeros(36)**

**coef[0]=5**

**coef[1]=.5**

**ES = [f, coef]**

**L1c.ExtraData=ES**

**P\_Ima = kn.surf()**

**P\_Ima.Rc = 0.0**

**P\_Ima.Thickness = 0.0**

**P\_Ima.Glass = "AIR"**

**P\_Ima.Diameter = 200.0**

**P\_Ima.Name = "Image plane"**

**A = [P\_Obj, L1a, L1c, P\_Ima]**

**Config\_1 = kn.Kraken\_setup()**

**Lens = kn.system(A, Config\_1)**

**Rays = kn.raykeeper(Lens)**

**Wav = 0.45**

**for i in range(-100, 100+1):**

**pSource = [0.0, i/10., 0.0]**

**dCos = [0.0, 0.0, 1.0]**

**Lens.Trace(pSource, dCos, Wav)**

**Rays.push()**

**kn.display3d(Lens, Rays, 2)**

**kn.display2d(Lens, Rays, 0)**

|  |
| --- |
|  |

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

# Example - Extra Shape XY Cosines

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**Examp Extra Shape XY Cosines"""**

**import Kraken as kn**

**import numpy as np**

**import matplotlib.pyplot as plt**

**P\_Obj = kn.surf()**

**P\_Obj.Rc = 0.0**

**P\_Obj.Thickness = 10**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 30.0**

**L1a = kn.surf()**

**L1a.Rc = 55.134\*0**

**L1a.Thickness = 9.0**

**L1a.Glass = "BK7"**

**L1a.Diameter = 30.0**

**L1c = kn.surf()**

**L1c.Thickness = 40**

**L1c.Glass = "AIR"**

**L1c.Diameter = 30**

**def f(x,y,E):**

**r = np.sqrt((x \* x) + (y \* y \* 0))**

**H=2.0\*np.pi\*r/E[0]**

**zx=np.abs(np.cos(H) \* E[1])**

**r = np.sqrt((x \* x \* 0) + (y \* y))**

**H=2.0\*np.pi\*r/E[0]**

**zy=np.abs(np.cos(H) \* E[1])**

**return zx+zy**

**coef=[10.0,1.]**

**L1c.ExtraData=[f, coef]**

**L1c.Res=1**

**P\_Ima = kn.surf()**

**P\_Ima.Rc = 0.0**

**P\_Ima.Thickness = 0.0**

**P\_Ima.Glass = "AIR"**

**P\_Ima.Diameter = 300.0**

**P\_Ima.Name = "Image plane"**

**A = [P\_Obj, L1a, L1c, P\_Ima]**

**Config\_1 = kn.Kraken\_setup()**

**Lens = kn.system(A, Config\_1)**

**Rays = kn.raykeeper(Lens)**

**Wav = 0.45**

**for i in range(-100, 100+1):**

**pSource = [0.0, i/10., 0.0]**

**dCos = [0.0, 0.0, 1.0]**

**Lens.Trace(pSource, dCos, Wav)**

**Rays.push()**

**kn.display3d(Lens, Rays, 2)**

**kn.display2d(Lens, Rays, 0)**

|  |
| --- |
|  |

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

# Example - Multicore

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**Examp Multicore"""**

**import multiprocessing**

**import time**

**import numpy as np**

**import Kraken as kn**

**start\_time = time.time()**

**P\_Obj = kn.surf()**

**P\_Obj.Rc = 0.0**

**P\_Obj.Thickness = 10**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 30.0**

**L1a = kn.surf()**

**L1a.Rc = 9.284706570002484E+001**

**L1a.Thickness = 6.0**

**L1a.Glass = "BK7"**

**L1a.Diameter = 30.0**

**L1a.Axicon = 0**

**L1b = kn.surf()**

**L1b.Rc = -3.071608670000159E+001**

**L1b.Thickness = 3.0**

**L1b.Glass = "F2"**

**L1b.Diameter = 30**

**L1c = kn.surf()**

**L1c.Rc = -7.819730726078505E+001**

**L1c.Thickness = 9.737604742910693E+001**

**L1c.Glass = "AIR"**

**L1c.Diameter = 30**

**P\_Ima = kn.surf()**

**P\_Ima.Rc = 0.0**

**P\_Ima.Thickness = 0.0**

**P\_Ima.Glass = "AIR"**

**P\_Ima.Diameter = 3.0**

**P\_Ima.Name = "Plano imagen"**

**A = [P\_Obj, L1a, L1b, L1c, P\_Ima]**

**config\_1 = kn.Kraken\_setup()**

**Doblete1 = kn.system(A, config\_1)**

**def trax1(xyz, lmn, w, q):**

**Rayos = kn.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))**

|  |
| --- |
|  |

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

# Example - Solid Objects STL ARRAY

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**Examp Solid Objects STL ARRAY"""**

**import matplotlib.pyplot as plt**

**import numpy as np**

**import pyvista as pv**

**import Kraken as kn**

**P\_Obj = kn.surf()**

**P\_Obj.Thickness = 5000.0**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 6.796727741707513E+002 \* 2.0**

**P\_Obj.Drawing = 0**

**FOV = 0.5**

**Ref\_esp = 0.8**

**Tf = 200 *# 40***

**A1 = Tf \* Tf**

**Conc = 800**

**Conc = Conc / 0.8**

**fpl = int(np.round(np.sqrt(Conc) / 2.0))**

**print("Numero de facetas por lado (Calculo 1)", (fpl \* 2.0) + 1.0)**

**print("Tamanio por lado(mm) ", Tf \* ((fpl \* 2.0) + 1.0))**

**n = fpl**

**FN = 1**

**focal = Tf \* (fpl \* 2.0 + 1.0) \* FN**

**print("Distancia focal(mm) ", focal)**

**sobredim = 2.0 \* focal \* np.tan(np.deg2rad(FOV / 2.0))**

**print("Sobredim (mm): ", sobredim)**

**Tf = Tf - sobredim**

**A2 = Tf \* Tf**

**RA = A1 / A2**

**Conc = Conc \* RA**

**print("Nuevo tamaño de las facetas: ", Tf)**

**fpl = int(np.round(np.sqrt(Conc) / 2.0))**

**print("Nuevo numero de facetas por lado (Calculo 2)", (fpl \* 2.0) + 1.0)**

**print("Tamanio por lado 2a (mm) ", Tf \* ((fpl \* 2.0) + 1.0))**

**n = fpl**

**Cx = Tf**

**Cy = Tf**

**Cz = 0**

**Lx = Tf**

**Ly = Tf**

**Lz = 1.0**

**element0 = pv.Cube(center=(0.0, 0.0, 0.0), x\_length=0.1, y\_length=0.1, z\_length=0.1, bounds=None)**

**for A in range(-n, n + 1):**

**for B in range(-n, n + 1):**

**Ty = 0.5 \* np.rad2deg(np.arctan2(Cx \* A, focal))**

**Tx = -0.5 \* np.rad2deg(np.arctan2(Cy \* B, focal))**

**element1 = pv.Cube(center=(0.0, 0.0, 0.0), x\_length=Lx, y\_length=Ly, z\_length=Lz, bounds=None)**

**element1.rotate\_x(Tx)**

**element1.rotate\_y(Ty)**

**v = [-Cx / 2.0, -Cy / 2.0, 0]**

**element1.translate(v)**

**v = [Cx \* A, Cy \* B, Cz]**

**element1.translate(v)**

**element0 = element0 + element1**

**element0.save("salida.stl")**

**direc = r"salida.stl"**

**objeto = kn.surf()**

**objeto.Diameter = 118.0 \* 2.0**

**objeto.Solid\_3d\_stl = direc**

**objeto.Thickness = -6000**

**objeto.Glass = "MIRROR"**

**objeto.TiltX = 0**

**objeto.TiltY = 0**

**objeto.DespX = 0**

**objeto.DespY = 0**

**objeto.AxisMove = 0**

**P\_Ima = kn.surf()**

**P\_Ima.Rc = 0**

**P\_Ima.Thickness = -1.0**

**P\_Ima.Glass = "AIR"**

**P\_Ima.Diameter = 2000.0**

**P\_Ima.Drawing = 1**

**P\_Ima.Name = "Plano imagen"**

**A = [P\_Obj, objeto, P\_Ima]**

**configur = kn.Kraken\_setup()**

**Telescope = kn.system(A, configur)**

**Rays = kn.raykeeper(Telescope)**

**W = 0.633**

**tam = 25**

**rad = 5500.0**

**tsis = len(A) + 2**

**for j in range(-tam, tam + 1):**

**for i in range(-tam, tam + 1):**

**x\_0 = (i / tam) \* rad**

**y\_0 = (j / tam) \* rad**

**r = np.sqrt((x\_0 \* x\_0) + (y\_0 \* y\_0))**

**if r < rad:**

**tet = 0.0**

**pSource\_0 = [x\_0, y\_0, 0.0]**

**dCos = [0.0, np.sin(np.deg2rad(tet)), np.cos(np.deg2rad(tet))]**

**Telescope.NsTrace(pSource\_0, dCos, W)**

**if np.shape(Telescope.NAME)[0] != 0:**

**if Telescope.NAME[-1] == "Plano imagen":**

**plt.plot(Telescope.Hit\_x[-1], Telescope.Hit\_y[-1], '.', c="g")**

**Rays.push()**

**plt.axis('square')**

**plt.show()**

**kn.display3d(Telescope, Rays, 0)**

|  |
| --- |
|  |
|  |

Figure 37. (Top) Blob diagram generated by an array of planar mirrors shown in Figure below.

# Example - Sourse\_Distribution\_Function

***#!/usr/bin/env python3***

***# -\*- coding: utf-8 -\*-***

**"""**

**Created on Sun Aug 2 12:04:14 2020**

**Example - -Sourse\_Distribution\_Function.py**

**"""**

**import matplotlib.pyplot as plt**

**import numpy as np**

**import pyvista as pv**

**import scipy**

**import Kraken as kn**

**P\_Obj = kn.surf()**

**P\_Obj.Thickness = 5000.0**

**P\_Obj.Glass = "AIR"**

**P\_Obj.Diameter = 6.796727741707513E+002 \* 2.0**

**P\_Obj.Drawing = 0**

***######################################################################################***

**objeto = kn.surf()**

**objeto.Rc=-12000**

**objeto.k=-1**

**objeto.Diameter=2500**

**objeto.Thickness = -6000**

**objeto.Glass = "MIRROR"**

**P\_Ima = kn.surf()**

**P\_Ima.Rc = 0**

**P\_Ima.Thickness = -1.0**

**P\_Ima.Glass = "AIR"**

**P\_Ima.Diameter = 6000.0**

**P\_Ima.Drawing = 1**

**P\_Ima.Name = "Plano imagen"**

**A = [P\_Obj, objeto, P\_Ima]**

**configur = kn.Kraken\_setup()**

**Telescope = kn.system(A, configur)**

**Rays = kn.raykeeper(Telescope)**

**W = 0.633**

**Sun = kn.SourceRnd()**

**Example - =4**

**if Example - == 0:**

***# Sun distribution***

**def f(x):**

**teta=x**

**FI=np.zeros\_like(teta)**

**Arg2=np.argwhere(teta>(4.65/1000.0))**

**FI=np.cos(0.326 \* teta)/np.cos(0.308\*teta)**

**Chi2=.03**

**k=0.9\* np.log(13.5\*Chi2)\*np.power(Chi2,-0.3)**

**r=(2.2\* np.log(0.52\*Chi2)\*np.power(Chi2,0.43))-1.0**

**FI[Arg2]= np.exp(k)\*np.power(teta[Arg2] \* 1.0e3 , r)**

**return FI**

**Sun.field =20\*np.rad2deg((4.65/1000.0))**

**if Example - == 1:**

***# Sinc cunction***

**def f(x):**

**Wh=0.025**

**r=(x\*90.0/0.025)\*np.pi**

**res=np.sin(r)/r**

**return res**

**Sun.field =0.025\*3**

**if Example - == 2:**

***#Flat***

**def f(x):**

**res=1**

**return res**

**Sun.field =1.2/(2.\*3600.)**

**if Example - == 3:**

***# Parabolic***

**def f(x):**

**r=(x\*90.0/0.025)**

**res=r\*\*2**

**return res**

**Sun.field =1.2/(2.\*3600.)**

**if Example - == 4:**

***# Gaussian (Seeing)***

**def f(x):**

**x=np.rad2deg(x)**

**seing=1.2/3600.0**

**sigma=seing/2.3548**

**mean = 0**

**standard\_deviation = sigma**

**y=scipy.stats.norm(mean, standard\_deviation)**

**res=y.pdf(x)**

**return res**

**Sun.field=4\*1.2/(2.0\*3600.0)**

**Sun.fun = f**

**Sun.dim = 3000**

**Sun.num = 100000**

**L, M, N, X, Y, Z = Sun.rays()**

**Xr=np.zeros\_like(L)**

**Yr=np.zeros\_like(L)**

**Nr=np.zeros\_like(L)**

**con=0**

**con2=0**

**for i in range(0,Sun.num):**

**if con2==10:**

**print(100.\*i/Sun.num)**

**con2=0**

**pSource\_0 = [X[i], Y[i], Z[i]]**

**dCos = [L[i], M[i], N[i]]**

**Telescope.Trace(pSource\_0, dCos, W)**

**Xr[con]=Telescope.Hit\_x[-1]**

**Yr[con]=Telescope.Hit\_y[-1]**

**Nr[con]=Telescope.SURFACE[-1]**

**con=con+1**

**con2=con2+1**

***#Rays.push()***

**args=np.argwhere(Nr==2)**

**plt.plot(Xr[args], Yr[args], '.', c="g", markersize=1)**

***# axis labeling***

**plt.xlabel('x')**

**plt.ylabel('y')**

***# figure name***

**plt.title('Dot Plot')**

**plt.axis('square')**

**plt.show()**

|  |  |
| --- | --- |
|  |  |
|  |  |

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