README.md

Optical Ray Tracing (ORT) Module

Description

The ORT module provides a system of functions for geometrical optics simulations in 3D. It is capable of modelling compound optical systems comprised of optical elements with flat and spherical surfaces. It has been made as part of a 2nd year BSc physics computing project at Imperial College London.

Getting Started

Installation

To install the module, copy the ORT.py file into the working directory of your project. Import the module by including the following command at the beginning of the program which is to use the module.

```
import ORT as ort
```

Dependencies

The module requires Python 3 to be installed on the system. Furthermore it requires the NumPy and Matplotlib libraries to be installed in the working environment. These can be installed using the pip python package manager with the following commands

```
pip install numpy
pip install matplotlib
```

Additionally, the lens optimisation task which is included in the Tasks folder requires the SciPy library which can again be installed using pip with the command

```
pip install scipy
```

Alternatively, the Anaconda Python distribution comes preinstalled with all of the required libraries and can be installed here.

Getting Started

Initialising light and surfaces

The basics of what the module does and how to use it can be understood through the documentation included in the module.

To initialise a ray of light use the command

```
ray = ort.Ray(p, k, color)
```

A collection of rays can be initialised as a beam with a given profile using the command

```
beam = ort.Ray(p, k, r, n, profile, color)
```

A flat propagating surface can be initialised with the command

```
surface = ort.FlatSurf(p, n, n1, n2, apt, color)
```

and alternatively, a surface with a spherical profile can be initialised with

```
surface = ort.SpherSurf(p, curv, n1, n2, apt, color)
```

A spherical lens comprised of two propagating surfaces can be initialised using the following command

```
lens = ort.SpherLens(p, z, curv1, curv2, n1, n2, apt, color)
```

Finally, a screen to image the light can be initialised as

```
screen = ort.SpherLens(p, n, virtual, color)
```

Propagating and Intercepting Light

To propagate a ray or a beam of light through an optical element which can be any surface or a lens use the general method shown below.

```
element.propagate(light)
```

An intercept method can be used to intercept a beam of light with a screen using the command shown below.

```
screen.intercept(light)
```

Additionally, to intercept light giving rise to a virtual image set virtual to True when initialising the screen.

The intercept method can also be used to intercept light with a propagating surface, this however only returns the length along the ray or beam at which it is intercepted by the surface.

Visual Output

The show method can be used to construct a ray diagram looking along the x-z plane. To do this simply use the commands

```
ray.show()
beam.show()
surface.show()
screen.show()
```

The spot diagram of the initial profile of a beam can be shown using the profile method of the beam as shown below

```
beam.profile()
```

Similarly, a spot diagram showing the image formed when a ray or a beam of light is intercepted by a screen can be constructed using the image method

```
screen.image(light)
```

Note to show a diagram the matplotlib.pyplot show method must be called after the code which constructs the diagrams. For example as shown below

```
ray.show()
beam.show()
surface.show()
screen.show()
matplotlib.pyplot.show()
```

Numerical Methods

The RMS spot radius of a beam, after being intercepted by a surface, can be calculated by calling the get_RMS method

```
RMS = beam.get_RMS()
```

The RMS spot radius can also be minimised along the z-axis giving a numerical estimate of the position of the focal point as well as the minimum RMS spot radius. To do this use the command

```
p_focus, RMS_focus  = beam.get_fnum(dz, show)
```

And by setting show to True the determined focal plane will be shown on a ray diagram.

A theoretical value for the paraxial focal length of a spherical surface or a thick lens can be calculated using the get_fparax method as shown below

```
f_parax = surface.get_fparax(show)
f_parax = lens.get_fparax(show)
```

Finally, the Coddington shape factor q of a spherical lens can be calculated by calling the get_q method of a spherical lens

```
q = lens.get_q()
```

Useful Functions

The module also includes a variety of useful module level functions.

To calculate the magnitude of any vector simply use the function

```
magnitude = ort.get_mag(vector)
```

A vector can also be normalised with the use of the function

```
vector = ort.get_norm(vector)
```

For more complicated ventures into geometrical optics an implementation of the Snell's law in vector form can be used to obtain the direction of travel of a refracted ray as

```
refracted = ort.snell(incident, normal, n1, n2)
```

Other functions for example to generate common ray profiles can be found in the module documentation using the command

```
help(ort)
```

And with this I would like to wish you plenty of luck and joy in your optical ray tracing endeavours.

Examples

Examples of simple simulations such as a glass prism can be found in the Examples folder distributed with the module.

Tasks and Investigations

Tasks and investigations carried out using the module can be found in the Tasks and Investigations folders which are also distributed with the module. Finally there is a technical report on characterisation of spherical lenses using the ORT module.

Version History

The initial publicly released version of the module is v4.1 which only supports refraction and absorption by surfaces. In the near future, v5.0 is to be released and it will also support reflection as a method for surfaces to propagate light.

Author

- Lukas Kostal

Please feel free to contact me via email.

License