Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Самарский национальный исследовательский университет
имени академика С.П.Королёва»

Институт информатики, математики и электроники
Факультет информатики
Кафедра технической кибернетики

**Отчет по лабораторной работе No1**
**по курсу «Оптоинформационные технологии и системы»**

Выполнил студент: Белоусов А. А.
Группа: 6409
Преподаватель: Кириленко М.С.

Самара 2019

# ЗАДАНИЕ

1. Требуется создать программу, выполняющую расчет пересечения луча с заданными поверхностями. Результатом работы программы должны быть координаты точки пересечения луча с поверхностью. В качестве поверхности выбрать плоскость, сферу и эллипсоид.

2. Дополнить программу, чтобы среди результатов были параметры луча, отраженного данной поверхностью.

3. Дополнить программу, чтобы среди результатов были параметры луча, преломленного данной поверхностью.

4. Создать механизм отображения хода лучей и поверхности в двумерном случае.

## 1 Описание луча

Луч задан параметрически в виде:

$$\vec{r} = \vec{p_0} + \vec{e}t, \tag{1}$$

где $\vec{p_0}$ – радиус-вектор точки начала луча, $\vec{e}$ – вектор направления луча, $t$ – длина луча.

## 2 Пересечение с плоскостью

Уравнение плоскости записывается в виде:

$$(\vec{n},\, \vec{r} - \vec{r_0}) = 0, \tag{2}$$

где $\vec{n}$ – вектор нормали плоскости, $vecr_0$ – радиус-вектор точки, через которую проходит плоскость.

Формулу для длины луча получим подстановкой (1) в (2):

$$t = \frac{(\vec{n},\, \vec{r} - \vec{p_0})}{(\vec{n},\, \vec{e})}. \tag{3}$$

Используем для расчета следующие параметры луча и плоскости:

1. Параметры луча: $\vec{p_0} = (0, 0)$, $\vec{e} = (1, 2)$;

2. Параметры плоскости: $\vec{n} = (1, 0)$, $\vec{r_0} = (3, 2)$.

Получим точку пересечения луча с плоскостью: $(3, 6)$.

Длина луча до точки пересечения: $t = 1.044$.

На рисунках 1-3 представлены результаты пересечения луча и плоскости при различных значениях коэффициентов преломления $n_1$, $n_2$.

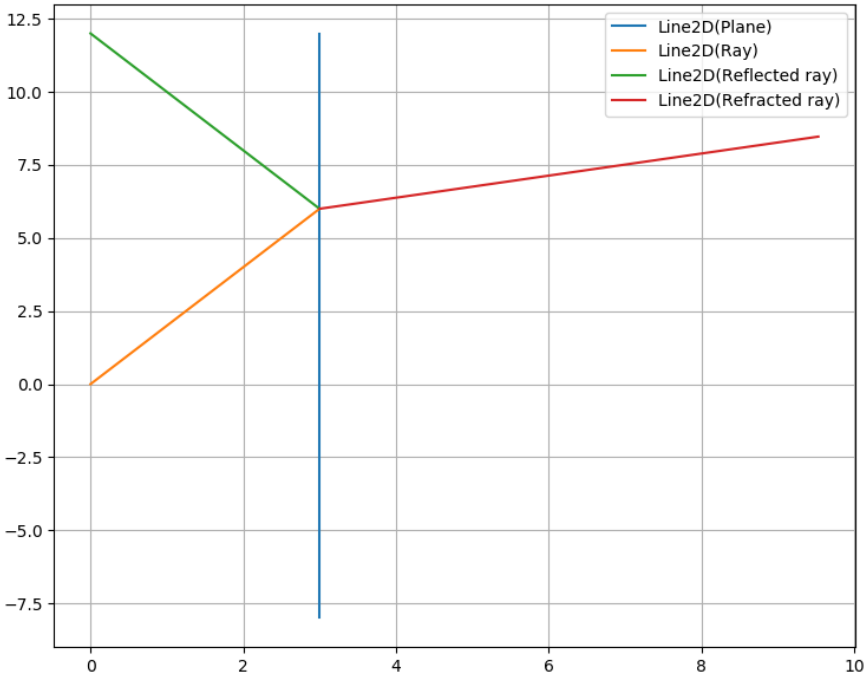


*Рисунок 1 – Пересечение с плоскостью при* $n_1 = 1$, $n_2 = 2$

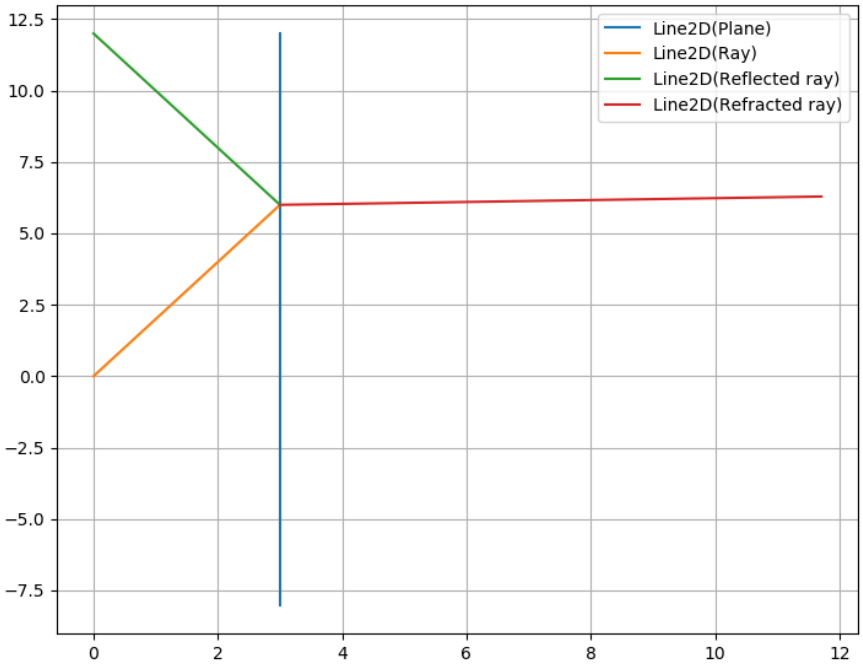


*Рисунок 2 – Пересечение с плоскостью при* $n_1 = 0.1$, $n_2 = 2$

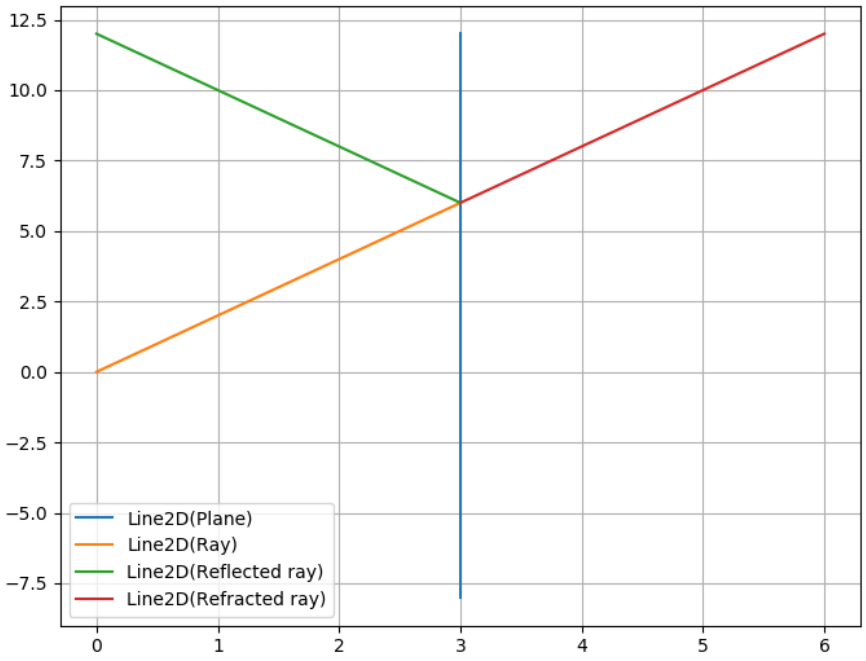


*Рисунок 3 – Пересечение с плоскостью при* $n_1 = 0.5$, $n_2 = 0.5$

## 3 Пересечение луча со сферой

Сфера задаётся следующим уравнением:

$$(\vec{p} - \vec{p_0}, \vec{p} - \vec{p_0}) = R^2, \tag{4}$$

где $\vec{p_0}$ – радиус-вектор точки цетра сферы, $R$ – радиус сферы.

Формулу для длины луча получим подстановкой (1) в (4):

$$t_{1,2} = (\vec{r_0} - \vec{p_0}, \vec{e}) \pm \sqrt{(\vec{r_0} - \vec{p_0}, \vec{e})^2 - (\vec{r_0} - \vec{p_0}, \vec{r_0} - \vec{p_0}) - R^2}. \tag{5}$$

Луч пересекается со сферой только в следующем случае:

$$(\vec{r_0} - \vec{p_0}, \vec{e})^2 - (\vec{r_0} - \vec{p_0}, \vec{r_0} - \vec{p_0}) - R^2 \geq 0. \tag{6}$$

Если луч пересекает сферу, то для нормали в точках пересечения имеем два варианта:

$$\vec{n} = \frac{r(\vec{t_{1,2}}) - \vec{p_0}}{||r(\vec{t_{1,2}}) - \vec{p_0}||} \tag{7}$$

Если поверхность выпуклая, то $(\vec{n}, \vec{e}) > 0$, если вогнутая, то $(\vec{n}, \vec{e}) < 0$.

Используем для расчета следующие параметры луча и сферы:

1. Параметры луча: $\vec{p_0} = (-1, -1)$, $\vec{e} = (1, 2)$;

2. Параметры сферы: $\vec{p_0} = (1, 1)$, $R = 2$.

Получим точки пересечения луча со сферой: $(-0.6, -0.2)$, $(1, 3)$.

Расстояния до пересечения с ближней и дальней сторонами сферы: $t_0 = 1.2, t_1 = 5.99$.

На рисунках 4-5 представлены результаты пересечения луча и сферы при различных значениях коэффициентов преломления $n_1$, $n_2$.

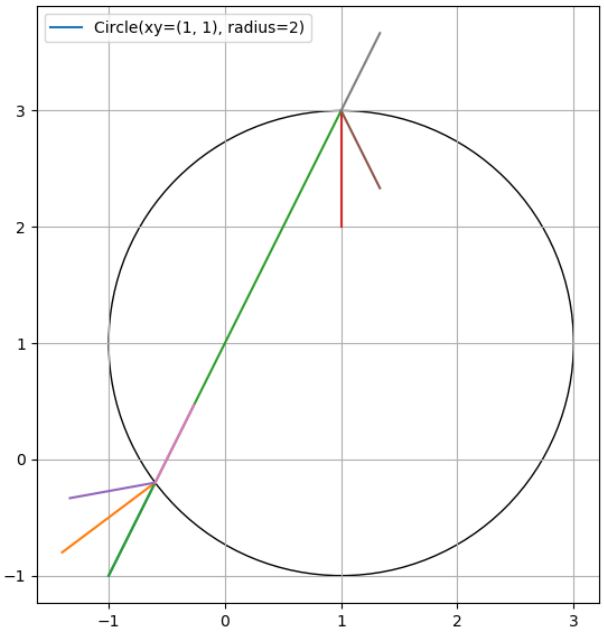


*Рисунок 4 – Пересечение со сферой при* $n_1 = 0.1$, $n_2 = 0.1$

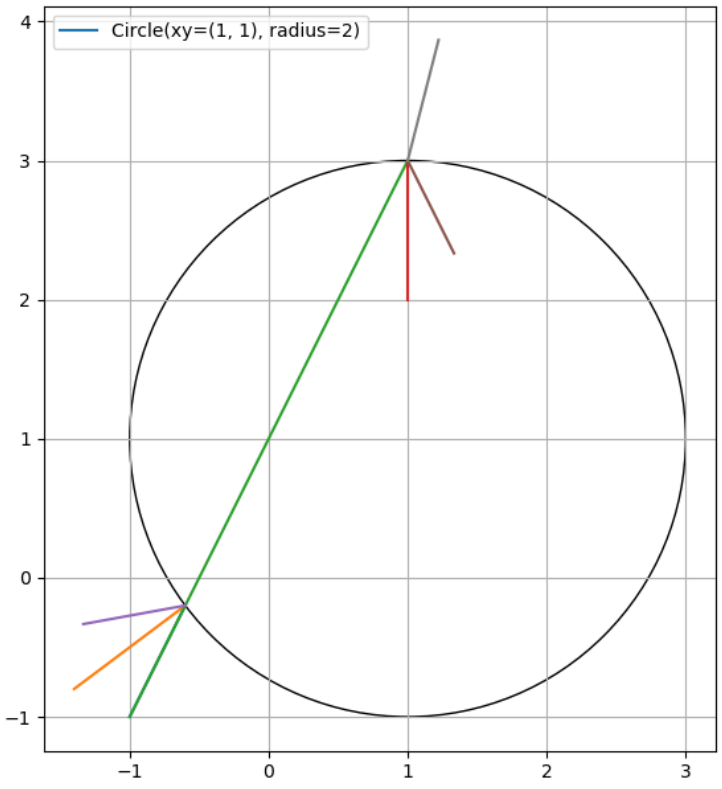


*Рисунок 5 – Пересечение со сферой при* $n_1 = 1.5$, $n_2 = 1$

## 4 Пересечение луча с эллипсоидом

Эллипс задаётся следующим уравнением:

$$\frac{(x-p_x)^2}{a^2}+\frac{(y-p_y)^2}{b^2}=1. \tag{8}$$

где $(p_x, p_y)$ – радиус-вектор точки центра эллипса; $a$, $b$ – длины полуосей эллипса.

Подставив (1) в (8), получаем квадратное уравнение относительно длины луча.

Используем для расчета следующие параметры луча и сферы:

1. Параметры луча: $\vec{p_0} = (-1, -1)$, $\vec{e} = (1, 2)$;
2. Параметры эллипса: $\vec{p_0} = (2, 2)$, $a = 4$, $b = 2$.

Получим точки пересечения луча с эллипсом: $(-0.31, 0.36)$, $(1.49, 3.98)$.

Расстояния до пересечения с ближней и дальней сторонами эллипса: $t_0 = 2.053, t_1 = 7.476$.

Результаты представлены на рисунках 6-7.

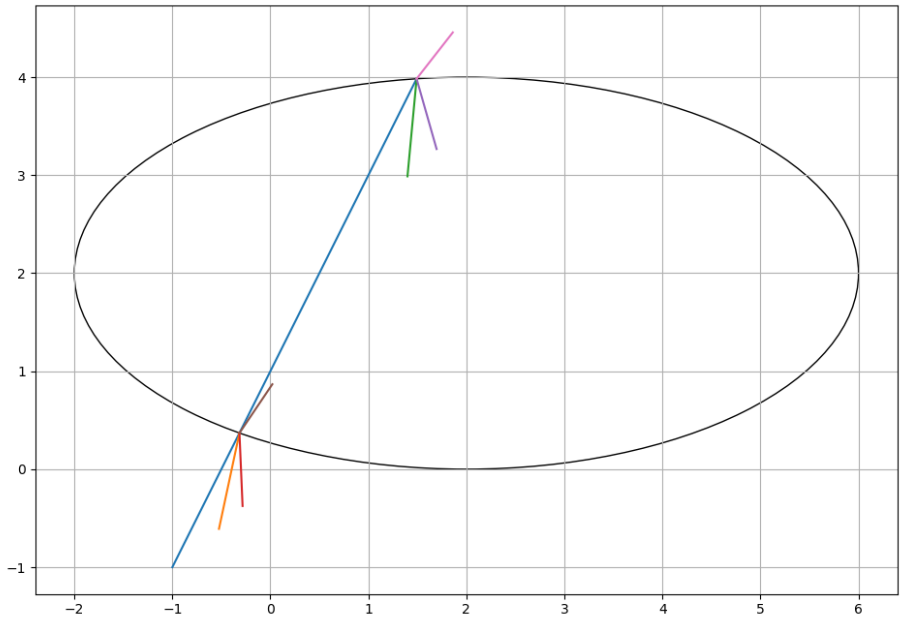


*Рисунок 6 – Пересечение с эллипсом при* $n_1 = 1$, $n_2 = 1.5$

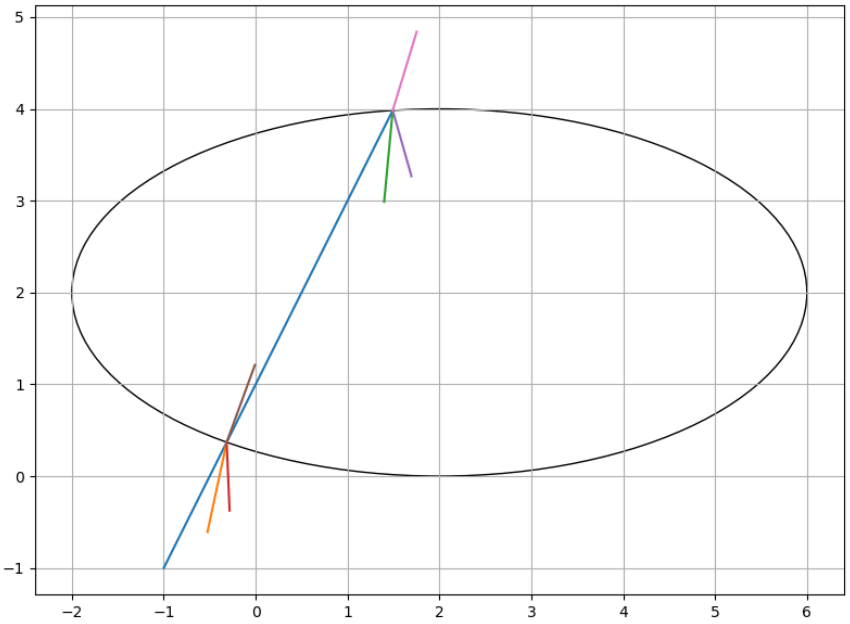


*Рисунок 7 – Пересечение с эллипсом при* $n_1 = 1$, $n_2 = 1.5$

# ЗАКЛЮЧЕНИЕ

В данной лабораторной работе была создана программа, которая запрашивает параметры луча и параметры поверхности. Результатом работы программы являлись координаты точки пересечения луча с поверхностью. В качестве поверхности были выбраны плоскость, сфера и эллипсоид. Также был реализован механизм отображения хода лучей и поверхности в двумерном случае. Программа может отображать падающий, отраженный и преломленный лучи. Были построенные соответствующие графики хода лучей из сред с разными коэффициентами преломления. С помощью данной программы легко увидеть, как луч проходит через разные поверхности, что угол падения меньше угла преломления при переходе луча из оптически более плотной среды в оптически менее плотную. Также можно увидеть, что в зависимости от поверхности луч отражается по разному.

# ПРИЛОЖЕНИЕ А

## Код программы

```
import matplotlib.pyplot as plt

from ray import Ray
from traceObjects.circle import Circle
from traceObjects.ellipse import Ellipse
from traceObjects.plane import Plane


def plane_example(ray, n1, n2):
    plane = Plane(
        normal=[1.0, 0.0, 0.0],
        radius_vec=[5.0, 3.0, 0.0],
    )
    plane_lgnd = plane.draw("plane")
    ray_lgnd, reflection_lgnd, refraction_lgnd = plane.draw_intersection(ray, n1, n2, "plane")
    plt.legend([plane_lgnd, ray_lgnd, reflection_lgnd, refraction_lgnd])
    plt.grid(True)


def circle_example(ray, n1, n2):
    circle = Circle(
        origin=[1, 1, 0],
        radius=2
    )
    circle_lgnd = circle.draw("circle")
    ray_res, norm_1, norm_2 = circle.draw_intersection(ray, n1, n2, "circle")
    plt.legend([circle_lgnd])
    plt.grid(True)


def ellipse_example(ray, n1, n2):
    ellipse = Ellipse(
        origin=[1, 2],
        a=3,
        b=2
    )
    ray.origin = [ray.origin[0], ray.origin[1]]
```

```
    ray.direction = [ray.direction[0], ray.direction[1]]
    ellipse_lgnd = ellipse.draw("ellipse")
    pussy = ellipse.draw_intersection(ray, n1, n2, "ellipse")
    # plt.legend([ellipse_lgnd, pussy, destroyer, xXX1337XXx])
    plt.grid(True)


if __name__ == "__main__":
    ray = Ray(
        origin=[-1, 0, 0],
        direction=[1, 2, 0]
    )
    # plane_example(ray, 2.5, 4)
    circle_example(ray, 2, 1.5)
    # ellipse_example(ray, 1.2, 1)
    plt.show()




class Ray(object):
    def __init__(self, origin: np.array, direction: np.array):
        self.origin = np.array(origin)
        self.direction = normalize(np.array(direction))




def normalize(v):
    norm = np.linalg.norm(v, ord=1)
    if norm == 0:
        norm = np.finfo(v.dtype).eps
    return v / norm


def reflection(e, n):
    reflection_point = e - 2 * np.dot(e, n) * n
    print(reflection_point)
    return reflection_point
```

```
def refraction(e, n, n1, n2):
    under_root = 1 - ((n1 ** 2) / (n2 ** 2)) * (1 - np.dot(e, n) ** 2)
    e = np.array(e)
    refraction_point = (e * n1 - n * (
           n1 * np.dot(e, n) - n2 * np.sqrt(under_root))) / n2
    return refraction_point




class Plane(object):
    def __init__(self, normal, radius_vec):
        self.normal = normalize(np.array(normal))
        self.radius_vec = np.array(radius_vec)

    def draw(self, figure="Fig"):
        plt.figure(figure)
        n_point = np.array([self.normal[1], -self.normal[0], 0.0])
        first = self.radius_vec - n_point * 10
        second = self.radius_vec + n_point * 10
        xs = np.array([first[0], second[0]])
        ys = np.array([first[1], second[1]])
        plane_lgnd, = plt.plot(xs, ys, label="Plane")
        return plane_lgnd

    def intersect_ray(self, ray: Ray, n1, n2):
        if abs(self.normal.dot(ray.direction)) <= 0:
            return None
        t = self.normal.dot(self.radius_vec - ray.origin) / self.normal.dot(ray.direction)
        intersection_point = ray.origin + ray.direction * t
        reflected_direction = ray.direction - 2 * ray.direction.dot(self.normal) * self.normal
        reflected_point = intersection_point + t * reflected_direction
        refracted_direction = n1 * ray.direction - self.normal * (
               n1 * (ray.direction.dot(self.normal)) - n2 *
               np.sqrt(1 - ((n1 ** 2) / (n2 ** 2)) * (
                      1 - (ray.direction.dot(self.normal) ** 2))))
        refracted_direction = normalize(refracted_direction)
        refracted_point = intersection_point + refracted_direction * t
        return intersection_point, reflected_point, refracted_point
```

```
    def draw_intersection(self, ray, n1, n2, figure="Fig"):
        intersect = self.intersect_ray(ray, n1, n2)
        if intersect is not None:
            (intersection, reflection, refraction) = intersect
            plt.figure(figure)
            ray_fig, = plt.plot([ray.origin[0], intersection[0]], [ray.origin[1], intersection[1]],
            reflection_fig, = plt.plot([intersection[0], reflection[0]], [intersection[1], reflectio
                                       label="Reflected ray")
            refraction_fig, = plt.plot([intersection[0], refraction[0]], [intersection[1], refractio
                                       label="Refracted ray")
            print("Plane int. point: {}".format(intersection))
            return ray_fig, reflection_fig, refraction_fig
        else:
            print("No intersection found")




class Circle(object):
    def __init__(self, origin, radius):
        self.origin = np.array(origin)
        self.radius = radius

    def draw(self, figure="Fig"):
        plt.figure(figure)
        ax = plt.gca()

        circle = plt.Circle(xy=(self.origin[0], self.origin[1]), radius=self.radius, fill=False, la
        ax.add_patch(circle)
        plt.gca().set_aspect("equal")
        return circle

    def sphere_intersection(self, ray: Ray):
        t1 = (np.dot(ray.origin - self.origin, ray.direction) + np.sqrt(
            (np.dot(ray.origin - self.origin, ray.direction)) ** 2 - np.dot(ray.direction, ray.direc
                  np.dot(ray.origin - self.origin, ray.origin - self.origin) - self.radius ** 2)))
            ray.direction,
            ray.direction)
```

```
        t2 = (np.dot(ray.origin - self.origin, ray.direction) - np.sqrt(
            (np.dot(ray.origin - self.origin, ray.direction)) ** 2 - np.dot(ray.direction, ray.direc
                  np.dot(ray.origin - self.origin, ray.origin - self.origin) - self.radius ** 2)))
            ray.direction,
            ray.direction)
        return t1, t2

    def intersect_ray(self, ray: Ray, en1, en2):
        t1, t2 = self.sphere_intersection(ray)
        r1 = ray.origin + ray.direction * -t1
        r2 = ray.origin + ray.direction * -t2
        n1 = -(r1 - self.origin) / np.sqrt(np.dot(r1 - self.origin, r1 - self.origin))
        n2 = (r2 - self.origin) / np.sqrt(np.dot(r2 - self.origin, r2 - self.origin))
        rfl1 = -reflection(ray.direction, n1)
        rfl2 = -reflection(ray.direction, n2)
        rfr1 = -refraction(ray.direction, n1, en1, en2)
        rfr2 = -refraction(ray.direction, n2, en2, en1)
        return r1, r2, rfl1, rfl2, rfr1, rfr2, n1, n2

    def draw_intersection(self, ray, on1, on2, figure="Fig"):
        plt.figure(figure)
        r1, r2, rfl1, rfl2, rfr1, rfr2, n1, n2 = self.intersect_ray(ray, on1, on2)
        ray_plt = plt.plot([ray.origin[0], r1[0]], [ray.origin[1], r1[1]], label="Ray")
        plt.plot([r1[0], r1[0] - n1[0]],
                 [r1[1], r1[1] - n1[1]], label="Ray")
        plt.plot([ray.origin[0], r2[0]],
                 [ray.origin[1], r2[1]], label="Ray")
        plt.plot([r2[0], r2[0] - n2[0]],
                 [r2[1], r2[1] - n2[1]], label="Ray")
        n1_plt, = plt.plot([r1[0], r1[0] - rfl1[0]],
                           [r1[1], r1[1] - rfl1[1]], label="Normal")
        plt.plot([r2[0], r2[0] - rfl2[0]],
                 [r2[1], r2[1] - rfl2[1]], label="Ray")
        n2_plt, = plt.plot([r1[0], r1[0] - rfr1[0]], [r1[1], r1[1] - rfr1[1]], label="Normal")
        plt.plot([r2[0], r2[0] - rfr2[0]],
                 [r2[1], r2[1] - rfr2[1]], label="Ray")
        plt.plot()
        print("Sphere int. point: {}".format(r1))
        return ray_plt, n1_plt, n2_plt
```

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.patches import Ellipse as Elp

import utils
from ray import Ray


class Ellipse(object):
    def __init__(self, origin, a, b):
        self.origin = np.array(origin)
        self.a = a
        self.b = b

    def norm(self, x0, y0):
        return -(y0 * (self.a ** 2)) / (x0 * (self.b ** 2))


    def intersection_ellipse_points(self, ray: Ray):
        m = np.array([[self.b, 0], [0, self.a]])
        # mr = np.dot(m, ray.origin - self.origin)
        # print(mr)
        # me = np.dot(m, ray.direction)
        a = np.dot(np.dot(m, ray.direction), np.dot(m, ray.direction))
        b = 2 * np.dot(np.dot(m, ray.direction), np.dot(m, ray.origin - self.origin))
        c = np.dot(np.dot(m, ray.origin - self.origin), np.dot(m, ray.origin - self.origin)) - (m[0

        # print(a, b, c)
        return np.roots([a, b, c])


    def draw(self, figure="Fig"):
        plt.figure(figure)
        ax = plt.gca()
        ellipse = Elp((self.origin[0], self.origin[1]), 2 * self.a, 2 * self.b,
                      fill=False,
                      label="Ellipse")
        ax.add_patch(ellipse)
```

```
    # plt.axis('scaled')
    plt.gca().set_aspect("equal")
    plt.grid(True)

def intersect_ray(self, ray: Ray, nn1, nn2):
    t = self.intersection_ellipse_points(ray)
    r = ray.origin + np.array(ray.direction) * t[1]
    r2 = ray.origin + np.array(ray.direction) * t[0]
    a = self.norm(r[0], r[1])
    a = np.arctan(a)
    a = np.array([np.cos(a), np.sin(a)])
    a2 = - self.norm(r2[0], r2[1])
    a2 = np.arctan(a2)
    a2 = np.array([np.cos(a2), np.sin(a2)])
    refl = - utils.reflection(ray.direction, a)
    refl2 = - utils.reflection(ray.direction, a2)
    refr = - utils.refraction(ray.direction, a, nn1, nn2)
    refr2 = - utils.refraction(ray.direction, a2, nn1, nn2)
    ray_lgnd, = plt.plot([ray.origin[0], r[0], r2[0]], [ray.origin[1], r[1], r2[1]])
    normal1, = plt.plot([r[0], r[0] - a[0]], [r[1], r[1] - a[1]])
    normal2, = plt.plot([r2[0], r2[0] - a2[0]], [r2[1], r2[1] - a2[1]])
    reflection, = plt.plot([r[0], r[0] - refl[0]], [r[1], r[1] - refl[1]])
    reflection2, = plt.plot([r2[0], r2[0] - refl2[0]], [r2[1], r2[1] - refl2[1]])
    refraction, = plt.plot([r[0], r[0] - refr[0]], [r[1], r[1] - refr[1]])
    refraction2, = plt.plot([r2[0], r2[0] - refr2[0]], [r2[1], r2[1] - refr2[1]])
    print(f"Length: {t[0]}")
    #plt.legend([ray_lgnd, normal1, normal2],
     # ["Ray", "Normal1", "Normal2"])
    # a = np.array([np.cos(a), np.sin(a)])

def draw_intersection(self, ray: Ray, nn1, nn2, figure="Fig"):
    plt.figure(figure)
    self.intersect_ray(ray, nn1, nn2)
```