Моделирование дифракции на отверстии произвольной формы

Исмагилов Амир, Куцубин Савва, Мельников Михаил
Группа Б02-201

4 декабря 2022 г.

Краткое описасние к описанию

Целью нашего проекта является моделирование дифракции Фраунгофера на отверстии произвольной формы. В предоставленном вашему вниманию тексте будут описанны используемые нами методы для вычисления интенсивности электромагнитного излучения в каждой точке рассматриваемой части экрана где наблюдается диффракция, так же будет показано как осуществляется загрузка файла содержащего необходимую информацию об отверстии на котором наблюдается дифракция.

1 Инструкция по использованию

- Создайте 24-разрядную bmp-картинку изображающую отверстие на котором Вы собираетесь наблюдать дифракцию
- Учтите, что точки чёрного цвета являются точками отверстия
- Переместите картинку в директорию с файлом
- Запустите программу

2 class Diffraction

2.1 center_of_mass

Сначала предлагается рассмотреть функцию center_of_mass. Которая определяет геометрический центр отверстия, считающийся точкой с нулевой фазой. Фактически центр масс вычисляется по формуле

$$\vec{r_c} = \frac{\sum\limits_i m_i \vec{r_i}}{\sum\limits_i m_i}$$

если наделить каждый пиксель принадлежащий отверстию единичной массой. Ниже приведён текст соответствующей функции.

```
    def center_of_mass(self):
        x_rel = 0
        y_rel = 0
        num_of_cells = 0
        for i in range(self.grid_size):
```

```
            for j in range(self.grid_size):
                if self.matrix[i][j] == 0:
                    x_rel += i
                    y_rel += j
                    num_of_cells += 1
        x_rel /= num_of_cells
        y_rel /= num_of_cells
        return (x_rel, y_rel)
```

Прежде чем приступить к изложению сущности двух следующих функций следует отметить, что для расчета интенсивности удобно разбить отверстие на множество квадратов и считать создаваемую ими напряженность в некоторой точке, а уже потом посредством суммирования получить искомую напряженность в рассматриваемой точке экрана.

2.2 calc_intensity

Функция calc_intensity делает то, что написано в её описании, используя формулу для расчета напряженности создаваемой единицей площади отверстия (прямоугольником) содержащей геометрический центр отверстия.

$$E = \int_{-a/2}^{a/2}\int_{-b/2}^{b/2} e^{ik(s_x x + s_y y)}dxdy = ab\frac{\sin\alpha}{\alpha}\frac{\sin\beta}{\beta},\ \alpha = \frac{1}{2}kas_x,\ \beta = \frac{1}{2}kbs_y.$$

s_x - Проекция единичного вектора направления дифрагированного пучка на ось OX
s_y - Проекция единичного вектора направления дифрагированного пучка на ось OY
x_{rel} - координата геометрического центра отверстия (определяется как точка с нулевой начальной фазой).
y_{rel} - координата геометрического центра отверстия (определяется как точка с нулевой начальной фазой).
a,b - длины сторон прямоугольника

```
    def calc_intensity(self):
        '''
        Подсчет интенсивности результирующего излучения в наблюдаемой точке.
        Заполняет матрицу color_grid_matrix значениями, соответствующими
        значениям интенсивности в соответствующей точке экрана.
        Returns
        -------
        None.
        '''
        # Подсчет положения центра масс отверстия
        x_rel, y_rel = self.center_of_mass()

        for i in tqdm(range(self.color_grid_size)):
            for j in range(self.color_grid_size):
                # Нахождение проекций вектора направления распространения
                # дифрагированного пучка
                s_x = (i - self.color_grid_size / 2) *\
                    self.pixel_len * self.scale
                s_y = (j - self.color_grid_size / 2) *\
                    self.pixel_len * self.scale

                # Модуль вектора направления распространения
                ro = np.sqrt(s_x**2 + s_y**2 + self.L**2)
```

```
            # Конечные значения проекций единичного вектора
            s_x /= ro
            s_y /= ro

            # alpha и beta в формулах
            alpha = np.pi * self.pixel_len * s_x / self.Lambda
            beta = np.pi * self.pixel_len * s_y / self.Lambda
            if alpha == 0:
                a_s = 1
            else:
                a_s = np.sin(alpha) / alpha

            if beta == 0:
                b_s = 1
            else:
                b_s = np.sin(beta) / beta

            d = self.pixel_len
            default_e = d**2 * a_s * b_s

            # Суммарная интенсивность в каждой точке экрана
            self.color_matrix[self.color_grid_size - j - 1][i] = \
                self.summing_tension(s_x, s_y, default_e, x_rel, y_rel)
```

2.3 summing_tension

Функция summing_tension находит интенсивность поля в рассматриваемой точке, используя для этого следующую формулу:

$$E(\mathbf{s}) = \int_{\Omega} e^{ik(\mathbf{sr})} dF$$

```
    def summing_tension(self, s_x, s_y, default_e, x_rel, y_rel):
        '''
        Находит интенсивность поля в рассматриваемой точке
        Parameters
        ----------
        s_x : float
            Проекция единичного вектора направления дифрагированного пучка
            на ось OX.
        s_y : float
            Проекция единичного вектора направления дифрагированного пучка
            на ось OY..
        default_e : float
            Коэффициент
        x_rel : float
            x - координата геометрического центра отверстия
            определяется( как точка с нулевой начальной фазой).
        y_rel : float
            y - координата геометрического центра отверстия
            определяется( как точка с нулевой начальной фазой).
        Returns
        -------
        e : float
```

```
            Интенсивность в точке наблюдения P.
        '''
        e = 0
        for i in range(self.grid_size):
            for j in range(self.grid_size):
                if self.matrix[i][j] == 0:
                    x_c = (i - x_rel) * self.pixel_len
                    y_c = (j - y_rel) * self.pixel_len
                    e += default_e * \
                        np.cos((x_c * s_x + y_c * s_y)
                               * 2 * np.pi/self.Lambda)
        return abs(e)
```

3 Часть программы отвечающая за загрузку файла

Импортируем библиотеки и проверяем файл на корректность.

```
import matplotlib.pyplot as plt
import numpy as np


def error_incorrect_file():
    print('Файл BMP некорректный')
    return 0


def error_bigsize():
    print('Файл слишком большой')
    return 0
```

Основная часть:

```
def LoadBMP(filename):
    '''
    == ОПИСАНИЕ ФУНКЦИИ ==
    Функция получает имя файла формата BMP: filename.
    Возвращает (w, h, pixels), где w - ширина, h - высота массива,
        pixels - целочисленный массив numpy, содержащий только 0 или 1.
    Файл BMP должен быть несжатым.
    Каждый пиксель должен принимать значение (0, 0, 0) или (255, 255, 255).
    (0, 0, 0) переводится в 0 в массив numpy, (255, 255, 255) - в 1.
    В случае, если размеры файла превышают HMAX и WMAX, вызывается функция
    error_bigsize(). Если она возвращает 0, то LoadBMP() завершает работу и
    возвращает (0, 0, 0), в противном случае инчего не происходит.
    В случае, если файл не соответствует указанным выше требованиям
    или он повреждён, вызывается функция error_incorrect_file()
    и функция LoadBMP возвращает (0, 0, 0).
    '''

    HMAX = 1000
    WMAX = 1000

    header = np.zeros(27, 'int')
```

```
file = open(filename, "rb")

% Читаем заголовок файла из( полезной информации там w и h)
for i in range(27):
    a = int.from_bytes(file.read(1), 'little')
    b = int.from_bytes(file.read(1), 'little')
    header[i] = b * 256 + a

# Проверка корректности файланесжатый(, RGB, 8 бит на цвет и мб чтото- ещё)
if (header[0] != 19778
        or header[5] != 54
        or header[7] != 40
        or header[13] != 1
        or header[14] != 24):
    file.close()
    error_incorrect_file()
    return (0, 0, 0)

w = header[9]  # ширина
h = header[11]  # высота
d = w % 4  # количество пустых байт, дополняющих каждую строку в конце

# Проверка размера файла
if (w >= WMAX or h >= HMAX):
    if (error_bigsize() == 0):
        file.close()
        return (0, 0, 0)

# Массив, где будет храниться картинка
pixels = np.zeros((h, w), 'int')

# Массив под 3 байта R, G, B
pixel = [30] * 3

for i in range(h):
    for j in range(w):
        for c in range(3):
            # Считываем один пиксель
            pixel[c] = int.from_bytes(file.read(1), 'little')

# Записысаем подходящее значение в файл, проверяя соответствие формату
        if (pixel[0] == pixel[1] and pixel[1] == pixel[2]):
            if (pixel[0] == 0):
                pixels[h - i - 1][j] = 0
            elif (pixel[0] == 255):
                pixels[h - i - 1][j] = 1
            else:
                file.close()
                error_incorrect_file()
                return (0, 0, 0)
        else:
            file.close()
            error_incorrect_file()
            return (0, 0, 0)
```

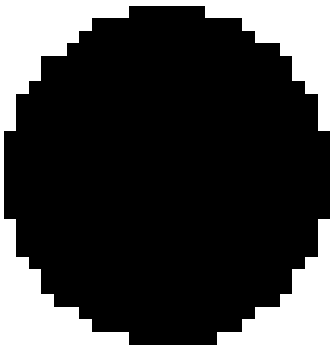
```
        # Пропускаем пустые байты
        if d != 0:
            for i in range(d):
                file.read(1)

    file.close()
    return(w, h, pixels)
    # Конец функции LoadBMP()
```

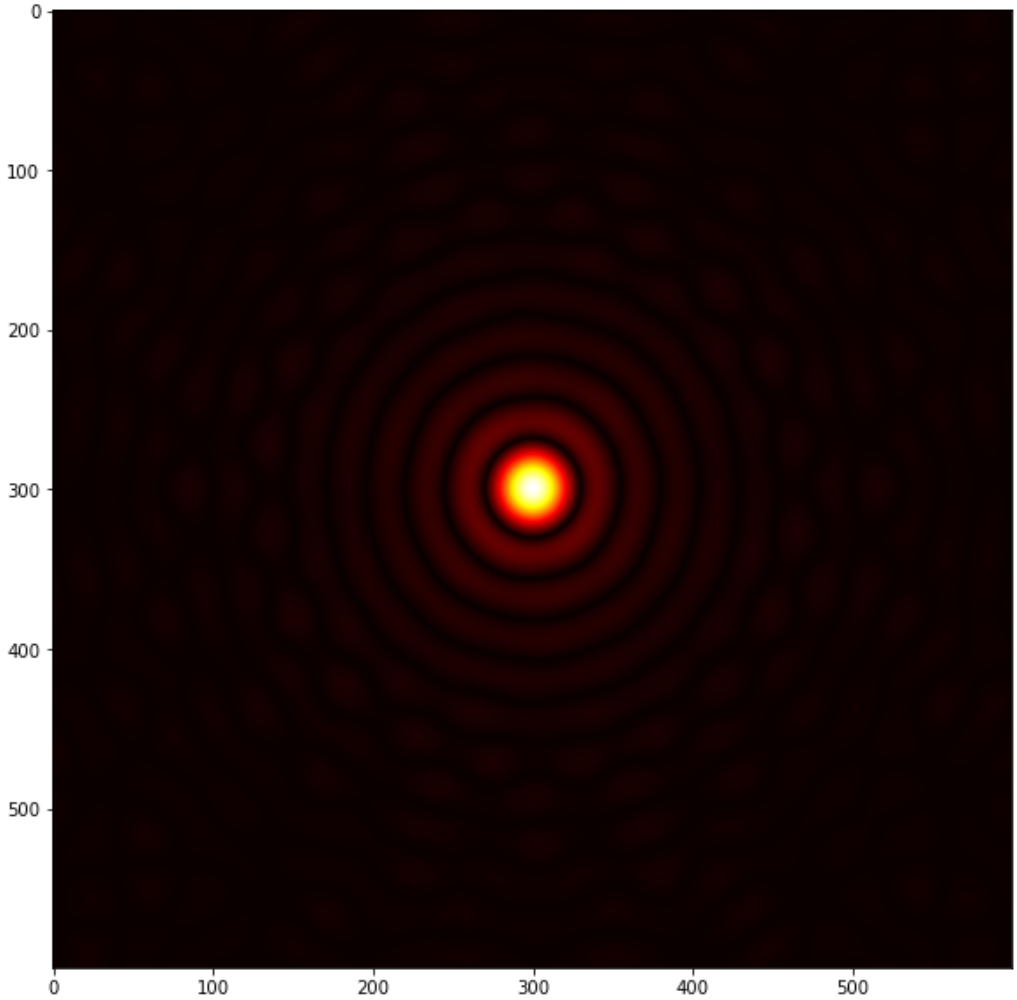
4 Загружаем конкретный файл и получаем дифракционную картину

```
# Обработка файла
h, w, points = LoadBMP('СПроект://img.bmp')

# Параметры
DifractionPicture = Difraction()
DifractionPicture.grid_size = h

# Четкость выходной картинкирекомендуется( ставить значения 150, 300 или 600)
DifractionPicture.color_grid_size = 150
# Масштаб дифракционной картины рекомендуемые( значения: от 1 до 60)
DifractionPicture.scale = 60

DifractionPicture.matrix = points
DifractionPicture.color_matrix = np.zeros(
    (DifractionPicture.color_grid_size, DifractionPicture.color_grid_size))
DifractionPicture.L = 2 * 10 ** 5
DifractionPicture.Lambda = 5
DifractionPicture.pixel_len = 10

# Подсчет интенсивности
DifractionPicture.calc_intensity()

# График matplotlib
fig, ax = plt.subplots(figsize=(10, 10))
ax.imshow(DifractionPicture.color_matrix, 'hot')
```

5 Примеры

5.1





5.2