Experimentally Verifying Fresnel and Fraunhofer Models for Single Slit Diffraction

PHYS 365

Kurt Hahn

Rochester Institute of Technology, NY

1 May 2024

Single-slit diffraction is a common optics demonstration used in unviersity physics classes. The resulting diffraction pattrern illustrates how coherent light through a aperture, comprabable in size to the wavelength, can noticeably interfere with itself to create bright and dark spots on a screen a given distance away. This pattern has been well-modelled using approximations made by Augustin-Jean Fresnel and Joseph von Fraunhofer. We present our work in verifying the experimental validity of both theories through computational modelling.

1. Theory

Fresnel's approximation for modelling single-slit diffraction patterns was to only consider points on a screen a distance z away such that distance from said points R were approximately equal:

$$R \approx z \tag{1}$$

This yields the following integral[1] for the field E at some point $\vec{r} = (x, y, z)$ on the screen after going through an aperture A:

$$E(\vec{r}) \approx -\frac{ie^{ikz}e^{i\frac{k}{2z}|r|^2}}{\lambda z}\int_A E(\vec{r}')e^{-\frac{k}{2z}|r'|^2}e^{-i\frac{k}{z}\vec{r}\cdot\vec{r}'}dA \tag{2}$$

Where k is the wavenumber $k = \frac{2\pi}{\lambda}$, λ is the wavelength, and $\vec{r}'$ is some position $(x', y', z' = 0)$ at the aperture. Note that this equation satisfies the paraxial approximation wave equation[1]:

$$\left(\partial_x^2 + \partial_y^2 + 2ik\partial_z\right)\tilde{E}(\vec{r}) \approx 0 \tag{3}$$

The Fresnel approximation is valid for a regime defined by the Fresnel number[2], F:

$$\left(F = \frac{a^2}{\lambda z}\right) \approx 1 \tag{4}$$

Where a is the aperture radius.

Fraunhofer made an additional approximation[1] on Fresnel's small-angle restriction; Fraunhofer considered the case to where the screen was placed far from the aperture (a.k.a the "far-field):

$$E(\vec{r}) \approx -\frac{ie^{ikz}e^{i\frac{k}{2z}|r|^2}}{\lambda z}\int_A E(\vec{r}\,')e^{-i\frac{k}{z}\vec{r}\cdot\vec{r}\,'}dA \tag{5}$$

The condition for this approximation is given as[1]:

$$F \ll 1 \Rightarrow z \gg \frac{a^2}{\lambda} \tag{6}$$

Because neither integral theory can be solved analytically, computer simulation necessitated numerical evaluation. If we consider A to be the set of all points in the plane $(x', y', 0)$, we can consider an obstacle B to be $U - A$, where U is the set of all points in space. Modelling the diffraction pattern from such a obstruction will follow Babinet's Principle[1]:

$$E(\vec{r}) = \left(\int_U dU - \int_A dA\right)\tilde{E}(\vec{r}') \tag{7}$$

In our experiments, a Helium-Neon laser was used, which was asssumed to be a Gaussian beam. Such a beam can be modelled by[1]:

$$E(\vec{r}') = E_0 e^{-\frac{|r'|^2}{w^2}} \tag{8}$$

Where w is the beam "waist" radius at the aperture.

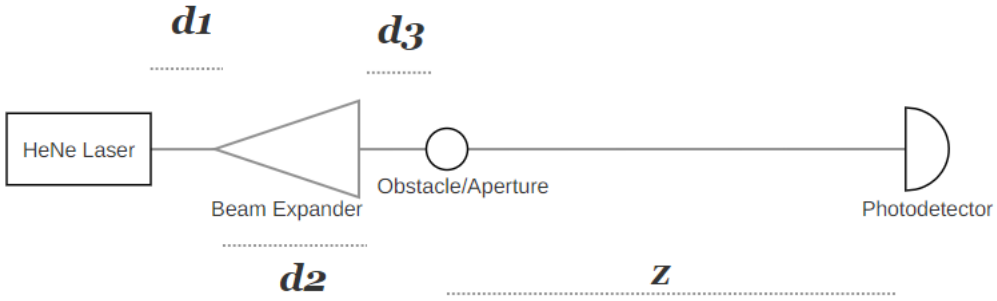


Fig. 1. General experimental setup with adjustable obstacle/aperture radius and adjustable distance z.

2. Simulation

Because the integrals from Eqs. 2 and 4 can only be solved numerically, a few additional approximations were necessary for the sake of practicality. As an approximation of the continuous integrands, our model would simulate a $n \times n$ grid of points at the aperture and screen, with n typically being 256 or 512. To evaluate the integral given in Eq. 6 for an obstacle, it was necessary to consider U to be the set of points in the space of significant radius from the aperture (≈10X the aperture radius) instead of to infinity. Further description of the code is shared in **Appendix A.**

To compare the theoretical model to our experimental data, we calculate the Mean-Squared-Error, or Deviation, (MSD) of our theoretical predictions and our data to measure the compatibility[3].

$$\mathbf{MSD} = \frac{1}{n}\sum_{i=0}^{n-1}(Y_i - Y_i')^2 \tag{9}$$

Before this, "normalize" our data with the theory and theoretical predictions (that is, we would set the maximums of both to 1.0 by dividing each dataset by its maximum value). This was so the MSD would describe disparities in the shape of the theoretical and experimental diffraction patterns.

3. Experiments

A schematic of our setup is shown in **Fig. 1.** The obstacle/aperture can be switched out for a pinhead of any radius and an adjustable window slit. The laser is a ThorLabs Helium-Neon (HeNe) red laser of wavelength of wavelength 632.8 nm and power output of 0.8 mW to 4 mW. The photodetector is a ThorLabs Switchable Gain Detector with an adjustable signal gain setting. A 10X beam expander was also used.

A 0.005 inch (0.127 mm) circular aperture was fixed to the photodetector, as well. This second aperture was placed close enough to the photodetector to ensure that any diffraction effects due to this second aperture were negligible.

The photodetector was fixed to a motorized stage that moved in both directions along the axis perpendicular to the beam and parallel to the table. The speed of this stage was measured to be 2 mm/s.

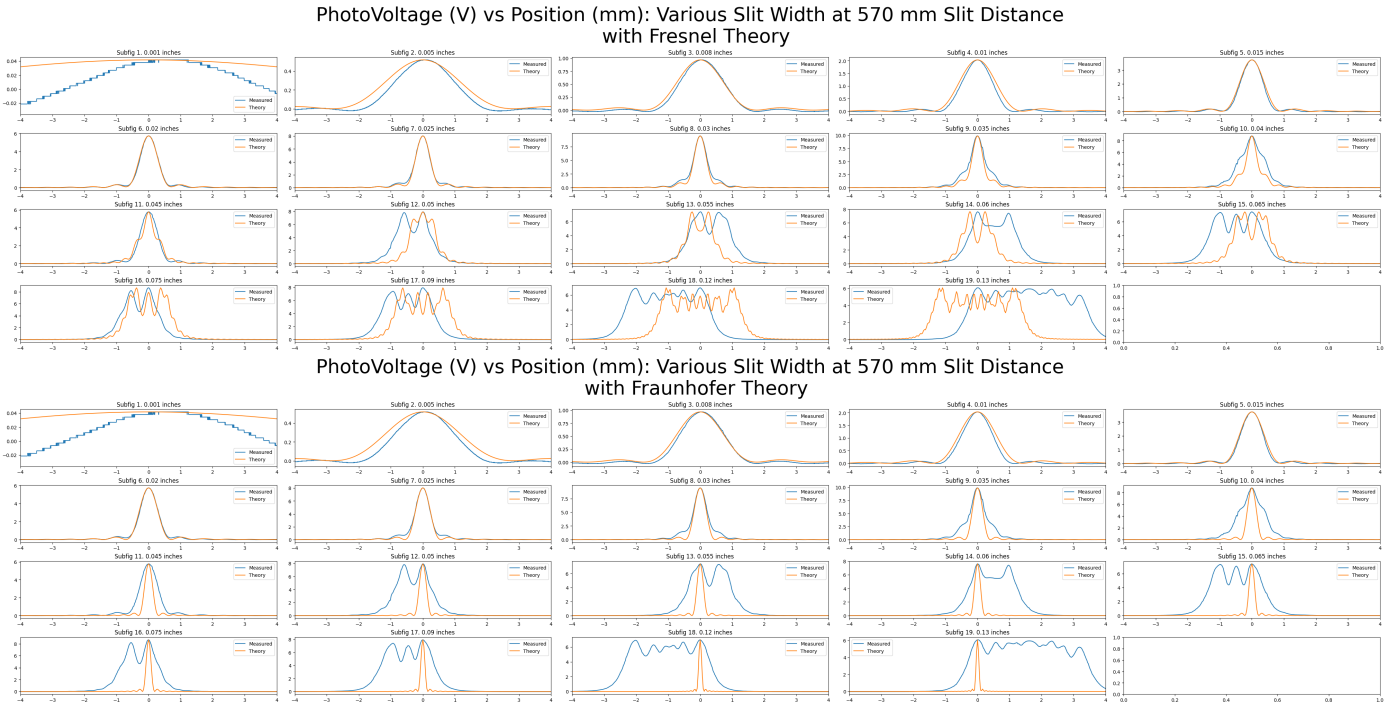
Great care was taken to ensure the beam did not stray from alignment with the beam expander, obstacle/aperture, and photodetector. Mirrors were not used because they were not large enough to reflect the entire diffraction pattern.

a. Varying Slit Width

The slit distance z was fixed to 570 mm, while the slit width a was adjusted from values of 0.001 inches (0.025 mm) to 0.13 inches (3.3 mm). Eq. 5 gives us the Fraunhofer limit aperture radius for this distance as $a \ll 0.6$ mm or 0.024 inches. (Likewise, the Fresnel condition would be $a \approx 0.024$ inches.) d_1, d_2, and d_3 were fixed at 42, 73, and 84 mm, respectively.

For each trial, the photodetector on the motorized stage was started around 50 mm from one side of the center bright spot and allowed to scan across the diffraction pattern to 50 mm to the other side.

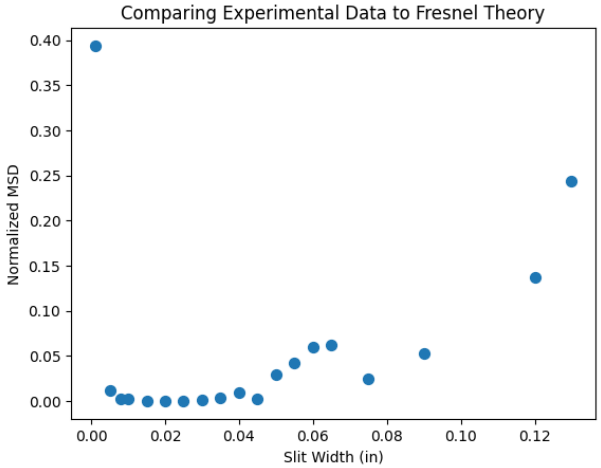
Figs. 2. and **3.** Detected photovoltage (in Volts) scanned across the beam (positions in m), centered at the maximum. The gain setting on the detector[4] was set to 70 dB (4.75×10^6 $\frac{\text{V}}{\text{A}}$). Each subfigure was trimmed to show positions ranging from -4 to 4 mm with our simulation of the Fresnel and Fraunhofer models are overlayed.

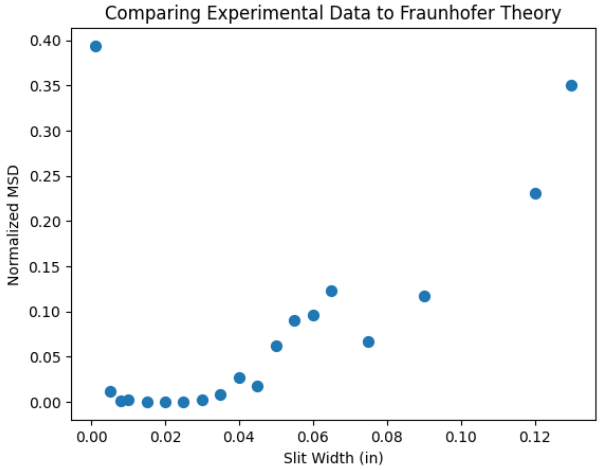


In our theoretical calculations, the slit window aperture was modelled as a slit with a height signicantly greater than its width. These simualtions were then compared with our collected data via Eq. 8 and plotted as shown in **Figs. 4** and **5** shown to the right.

However, it is apparent that our measurements for some subfigures (particularly **Subfigs. 12-19** in both Figures,) are not well-aligned with our models. This suggests that centering our data at the maximum is insufficent to gauge its consistency with the theory. More work is to be done to develop an algorithm to properly align the points on the graphs.

It is evident that the experimental data is most consistent with the theoretical models for smaller aperture radii (save for the outlier at a slit width of $a = 0.001$ inches). Our data is also consistent with the Fraunhofer limit of $a \ll 0.024$ inches (0.6 mm) and Fresnel condition of $a \approx 0.024$ inches; the Fraunhofer model excels within this regime. The Fresnel model expectedly, has a broader regime. Slit widths of 0.04 inches and greater appear to move into the ray-optics regime. From this, we can conclude that both the Fresnel and Fraunhofer are accurate models for the diffraction patterns within certain paraxial restrictions.

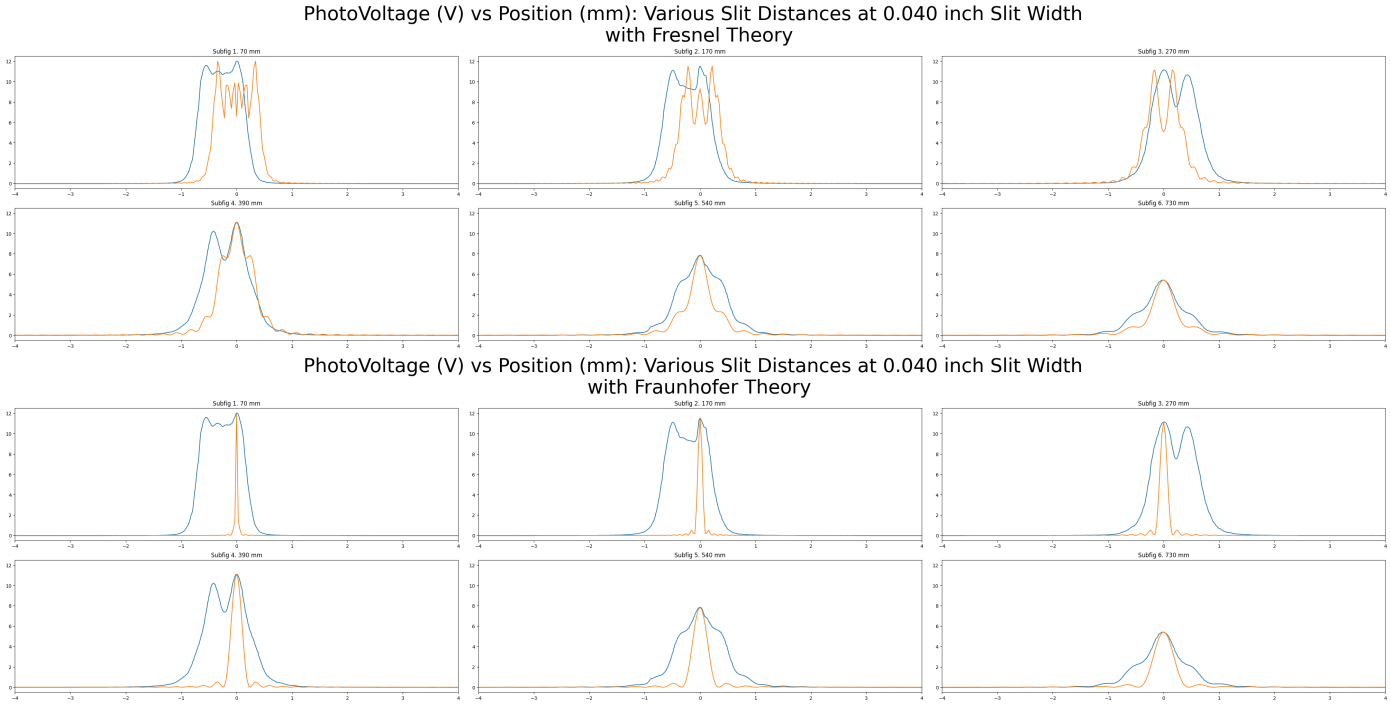




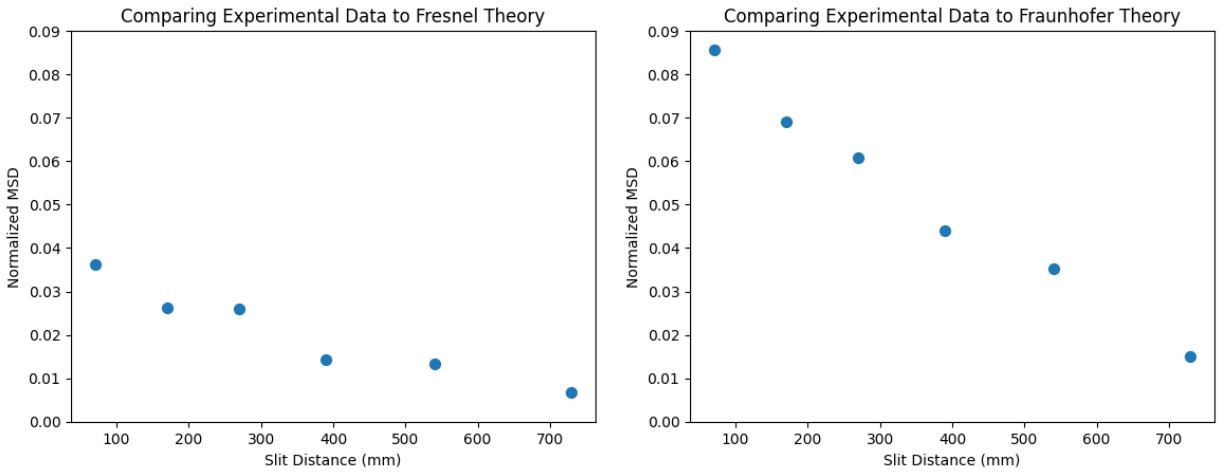
Figs. 4 and **5** show the MSD for both theories.

b. Varying Slit Distance

Another way we sought to confirm the two theories was to vary the disance from the slit to the detector z while fixing the slit width a to a constant 0.040 inches (1.016 mm). Again, much attention was paid to properly aligning the beam and the equipment. The distance z was varied from 70 mm to 730 mm through six trials. All other distances were kept the same as in **3.a**. For each trial, we again used the motorized stage to scan across the diffraction pattern with the photodetector.



Figs 6. and **7.** Detected voltage scanned across beam with gain setting[4] of 60 dB (1.50×10^6 $\frac{\text{V}}{\text{A}}$) for subfigures (1-4) and 70 dB for subfigures (5-6); voltage measurements at 60 dB were scaled to equivalent values to if they were taken at 70 dB. Profiles are centered at maximum. Each subfigure trimmed for positions ranging from -4 to 4 mm and overlayed with Fresnel and Fraunhofer theory.



Our data was outside of the Fresnel regime where $z \approx 1631$ mm but nicely fit with the theory nonetheless. Unfortunately, our optical bench was restricted in useable surface area, and the Fraunhofer limit of $z \gg 1631$ mm was not reached in our experiments. The use of mirrors had already been considered to be problematic. In future experiments, we hope to repeat our procedures with larger distances.

Still, from **Figs. 6** and **7** we can see that the Fraunhofer -and Fresnel- models were successful at modelling the diffraction pattern as z was increased.

Conclusions

This study confirms the accuracy of the Fresnel and Fraunhofer models for single-slit diffraction. We also demonstrate a simpler approach to transitioning between the two regimes by varying the aperture width rather than the aperture-screen distance.

It is possible that the beam expander produced significant enough diffraction effects that account for the experimental deviation from the theory. Additionally, later usage of the ThorLabs photodetector showed slight fluctuations in recorded photovoltage.

Discussion

For inspiring physicists and students, the single-slit diffraction experiment is virtually impossible to avoid throughout one's career. Despite its prevalence at the collegiate level, diffraction is an important phenomenon and tool that continues to be studied.

Experiments involving organic molecules through a diffraction grating have been conducted in order to analyze their intramolecular Van der Waal forces[5]. Matter waves that propagate and interfere from an aperture are characterized by their DeBroglie wavelength, and the resulting diffraction patterns can be modelled, similarly to light, within the Fraunhofer far-field regime. One important difference is that matter waves have greater variability in propagation speed than light waves.

Within the medical field, X-ray diffraction scattering has been shown to be effective in tissue-selective imaging. In one study[6], X-ray scattering was able to distinguish between layers of bone, muscle, and fat of a raw lamb-chop. This is possible because different materials may interact differently with x-rays in significant ways. According to the researchers at the University of Konstanz, x-ray diffraction imaging may be particularly effective in studying bone material of live patients.

It is admirable, and inspiring, that the theories developed centuries ago continue to be a reliable model for this phenomenon. Clearly, diffraction has many interesting and beneficial applications, which is why it is crucial it continues to be studied by the next generations of physicists.

Acknowledgements

Thank you to Dr. Charles Lusignan (Rochester Institute of Technology) for your patience and advice in the lab, as well as for reviewing this article prior to submission.

References

(1) Peatross, J.; Ware, M., 2023. *Physics of Light and Optics*, Bringham Young University, Utah, pp.265-266, 290

(2) Glückstad, J.; Madsen, A. E. G., 2023. New Analytical Diffraction Expressions for the Fresnel-Fraunhofer Transition Regime, *Optik* (285); doi: https://doi.org/10.1016/j.ijleo.2023.170950.

(3) Pishro-Nik, H. *Introduction to Probability, Statistics, and Random Process*, Kappa Research, LLC., https://www.probabilitycourse.com.

(4) ThorLabs. (2017). *PDA36A(-EC) Si Switchable Gain Detector - User Guide.* https://www.thorlabs.com/_sd.cfm?fileName=13053-D04.pdf&partNumber=PDA36A

(5) Brand, C. et al., 2021. Single-, Double-, and Triple-Slit Diffraction of Molecular Matter Waves, *American Journal of Physics* (89), pp. 1132-1138; doi: 10.1119/5.0058805

(6) Kleuker, U. et al., 1998. Feasibility Study of X-Ray Diffraction Computed Tomography for Medical Imaging, *Physics in Medicine & Biology* (43), pp. 2911-2923

Appendix A - diffractionsim

__init__.py

```
import GaussianBeam
import RegionFuncs
import Simulation
import Wavelet
```

GaussianBeam.py

```
import numpy as np

def return_incident_field(xPos: float, yPos: float, waistRadius: float) -> float:
    """
    :param xPos: x position (meters)
    :param yPos: y position (meters)
    :param waistRadius: beam radius at incident location (meters)
    :returns: field at incident location
    """
    field = np.exp(-(xPos**2 + yPos**2) / waistRadius**2)
    return field
```

RegionFuncs.py

```
import numpy as np

class Circle():
    def __init__(self, radius: float):
        """
        :param radius: radius of circular region (meters)
        """
        self.radius = radius

    def is_in_region(self, xPos, yPos):
        return np.sqrt(xPos**2 + yPos**2) < self.radius

class Square():
    def __init__(self, xRange: float, yRange: float):
        """"
        :param xRange: range from center x position (meters)
        :param yRange: range from center y position (meters)
        """
        self.xRange = xRange
        self.yRange = yRange

    def is_in_region(self, xPos, yPos):

        return np.abs(xPos) < self.xRange and np.abs(yPos) < self.yRange
```

```
class Slit():
    def __init__(self, xRange: float):
        """
        :param xRange: range from center x position (meters)
        """
        self.xRange = xRange

    def is_in_region(self, xPos, yPos):
        return np.abs(xPos) < self.xRange

class Lollipop():
    def __init__(self, radius: float, xRange: float):
        """
        :param radius: radius of circular part of region (meters)
        :param xRange: range from center of x position of stick part of region (meters)
        """
        self.radius = radius
        self.xRange = xRange

    def is_in_region(self, xPos, yPos):
        return np.logical_or(np.sqrt(xPos**2 + yPos**2) <
                self.radius, np.logical_and(yPos <= 0, np.abs(xPos) < self.xRange))
```

Simulation.py

```
import numpy as np
import matplotlib.figure
import matplotlib.pyplot as plt
import GaussianBeam
import Wavelet

class Obstacle():
    def __init__(self, waistRadius: float, wavelength: float, xRange: float,
                       yRange: float, obstacleFunction):
        """
        :param waistRadius: beam radius at obstacle location (meters)
        :param wavelength: wavelength of light (meters)
        :param xRange: range of x positions to consider at obstacle location (meters)
        :param yRange: range of y positions to consider at obstacle location (meters)
        :param obstacleFunction: Python function determining whether points (x,y) are
                                 in the obstacle region
        ------------------------------------
        Simulates diffraction pattern for obstacle
        Assume the use of Gaussian beam
        """
        self.waistRadius = waistRadius
        self.wavelength = wavelength
        self.xRange = xRange
        self.yRange = yRange
        self.obstacleFunction = obstacleFunction

    def return_figure_incident_intensity(self,
                                         numberPoints: int,
                                         yObstacle=0.0) -> matplotlib.figure.Figure:
        """
        :param numberPoints: number of points to plot
        :param yObstacle: y position of obstacle (meters) (default: 0.0)
        :returns: figure of incident gaussian beam intensity along x
        """
        xAperture = np.linspace(-3*self.waistRadius, 3*self.waistRadius, numberPoints)
        intensities = np.abs(
            GaussianBeam.return_incident_field(xAperture, yObstacle,
                                               self.waistRadius) )**2
        intensitiesPlaneWave = (
            GaussianBeam.return_incident_field(xAperture, yObstacle,
                                               10*self.waistRadius) )**2
        figure = plt.figure(0, figsize=(6,4))
        ax = figure.add_axes([0.1, 0.3, 0.8, 0.6])
        xAperture *= 1000 # mm / m
        ax.plot(xAperture, intensities, color="blue", label=f"Gaussian Beam,
                w = {self.waistRadius*1000} mm")
        ax.plot(xAperture, intensitiesPlaneWave, color="orange",
                label=f"Approximated Plane Wave, w = {10*self.waistRadius*1000} mm")
```

```
    ax.set_xlabel("Position Along Aperture (mm)")
    ax.set_ylabel(r"$I\propto\left|E\right|^2$")
    ax.set_xlim(xAperture[0], xAperture[-1])
    ax.set_ylim(0, 1.1)
    ax.legend(loc="lower center", bbox_to_anchor=(0.5, -0.5))
    ax.grid()
    ax.set_title(r"Gaussian Beam Intensity at Aperture ($z=0$)")
    return figure


def return_total_field(self, xPos: float, yPos: float, zPos: float,
                       wavletType: str, numberPoints: int) -> float:
    """
    :param xPos: x position (meters)
    :param yPos: y position (meters)
    :param zPos: z position, i.e. distance away from obstacle (meters)
    :param waveletType: diffraction wavelet theory, either 'Huygens',
                        'Fresnel', or 'Fraunhofer'
    :param numberPoints: number of points of integration
    :returns: total field at point
    """
    xAperture = np.linspace(-self.xRange, self.xRange, numberPoints)
    yAperture = np.linspace(-self.yRange, self.yRange, numberPoints)
    dx = np.abs(xAperture[0] - xAperture[1])
    dy = np.abs(yAperture[0] - yAperture[1])
    xAperture, yAperture = np.meshgrid(xAperture, yAperture)
    positionsAperture = self.obstacleFunction(xAperture, yAperture)
    incidentField = GaussianBeam.return_incident_field(xAperture, yAperture,
                                                       self.waistRadius)
    wavenumber = 2*np.pi / self.wavelength
    waveletField = None
    match wavletType:
        case "Huygens":
            wavelet = Wavelet.Huygens(wavenumber)
            waveletField =
                wavelet.return_field(xPos, yPos, zPos, xAperture, yAperture)
        case "Fresnel":
            wavelet = Wavelet.Fresnel(wavenumber)
            waveletField =
                wavelet.return_field(xPos, yPos, zPos, xAperture, yAperture)
        case "Fraunhofer":
            wavelet = Wavelet.Fraunhofer(wavenumber)
            waveletField =
                wavelet.return_field(xPos, yPos, zPos, xAperture, yAperture)
        case _:
            raise Exception("Invalid Wavelet Type")
    fields = (1 - positionsAperture) * incidentField * waveletField * dx * dy
    totalField = np.sum(fields)
    return totalField
```

```
    def return_figure_total_intensity(self, xPositions: np.ndarray, yPos: float,
                                            zPos: float, waveletType: str,
                                            numberPoints: int) ->
                                                matplotlib.figure.Figure:
        """
        :param xPositions: numpy array of x positions along screen to calculate
                           intensity at (meters)
        :param yPos: y position on screen (meters)
        :param zPos: z position of screen, i.e. distance from obstacle (meters)
        :param waveletType:  diffraction wavelet theory, either 'Huygens',
                             'Fresnel', or 'Fraunhofer'
        :param numberPoints: number of points of integration
        :returns: figure of normalized intensity along x (diffraction pattern)
        """
        intensities = []
        for xPos in xPositions:
            intensity = np.abs(
                self.return_total_field(xPos, yPos, zPos, waveletType,
                                        numberPoints) )**2
            intensities.append(intensity)
        intensities = np.array(intensities)
        intensities /= np.max(intensities)
        figure = plt.figure(0, figsize=(6,4))
        ax = figure.add_axes([0.1, 0.3, 0.8, 0.6])
        xPositions *= 1000 # mm / m
        ax.plot(xPositions, intensities, color="blue")
        ax.set_xlabel(r"Screen Position Along  $x$ (mm)")
        ax.set_ylabel(r"Normalized Intensity $\frac{I}{I_0}$")
        ax.set_xlim(xPositions[0], xPositions[-1])
        ax.set_ylim(0, 1.1)
        ax.grid()
        ax.set_title(f"Normalized Diffraction Pattern from Obstacle, y={yPos} m,
                       z={zPos} m")
        return figure

class Aperture():
    def __init__(self, waistRadius: float, wavelength: float, xRange: float,
                       yRange: float, apertureFunction):
        """
        :param waistRadius: beam radius at aperture location (meters)
        :param wavelength: wavelength of light (meters)
        :param xRange: range of x positions to consider at aperture location (meters)
        :param yRange: range of y positions to consider at aperture location (meters)
        :param apertureFunction: Python function determining whether points (x,y) are
                                 in the aperture region
        ----------------------------------
        Simulates diffraction pattern for aperture
        Assume the use of Gaussian beam
        """
        self.waistRadius = waistRadius
```

```
        self.wavelength = wavelength
        self.xRange = xRange
        self.yRange = yRange
        self.apertureFunction = apertureFunction

    def return_figure_incident_intensity(self, numberPoints: int, yAperture=0.0) ->
                                                                matplotlib.figure.Figure:
        """
        :param numberPoints: number of points to plot
        :param yAperture: y position of aperture (meters) (default: 0.0)
        :returns: figure of incident gaussian beam intensity along x
        """
        xAperture = np.linspace(-3*self.waistRadius, 3*self.waistRadius, numberPoints)
        intensities = np.abs(
            GaussianBeam.return_incident_field(xAperture, yAperture,
                                               self.waistRadius) )**2
        intensitiesPlaneWave = (
            GaussianBeam.return_incident_field(xAperture, yAperture,
                                               10*self.waistRadius) )**2
        figure = plt.figure(0, figsize=(6,4))
        ax = figure.add_axes([0.1, 0.3, 0.8, 0.6])
        xAperture *= 1000 # mm / m
        ax.plot(xAperture, intensities, color="blue",
                label=f"Gaussian Beam, w = {self.waistRadius*1000} mm")
        ax.plot(xAperture, intensitiesPlaneWave, color="orange",
                label=f"Approximated Plane Wave, w = {10*self.waistRadius*1000} mm")
        ax.set_xlabel("Position Along Aperture (mm)")
        ax.set_ylabel(r"$I\propto\left|E\right|^2$")
        ax.set_xlim(xAperture[0], xAperture[-1])
        ax.set_ylim(0, 1.1)
        ax.legend(loc="lower center", bbox_to_anchor=(0.5, -0.5))
        ax.grid()
        ax.set_title(r"Gaussian Beam Intensity at Aperture ($z=0$)")
        return figure

    def return_total_field(self, xPos: float, yPos: float, zPos: float,
                           wavletType: str, numberPoints: int) -> float:
        """
        :param xPos: x position (meters)
        :param yPos: y position (meters)
        :param zPos: z position, i.e. distance away from aperture (meters)
        :param waveletType: diffraction wavelet theory, either 'Huygens',
                            'Fresnel', or 'Fraunhofer'
        :param numberPoints: number of points of integration
        :returns: total field at point
        """
        xAperture = np.linspace(-self.xRange, self.xRange, numberPoints)
        yAperture = np.linspace(-self.yRange, self.yRange, numberPoints)
        dx = np.abs(xAperture[0] - xAperture[1])
        dy = np.abs(yAperture[0] - yAperture[1])
```

```
        xAperture, yAperture = np.meshgrid(xAperture, yAperture)
        positionsAperture = self.apertureFunction(xAperture, yAperture)
        incidentField = GaussianBeam.return_incident_field(xAperture,
                                                           yAperture, self.waistRadius)
        wavenumber = 2*np.pi / self.wavelength
        waveletField = None
        match wavletType:
            case "Huygens":
                wavelet = Wavelet.Huygens(wavenumber)
                waveletField =
                    wavelet.return_field(xPos, yPos, zPos, xAperture, yAperture)
            case "Fresnel":
                wavelet = Wavelet.Fresnel(wavenumber)
                waveletField =
                    wavelet.return_field(xPos, yPos, zPos, xAperture, yAperture)
            case "Fraunhofer":
                wavelet = Wavelet.Fraunhofer(wavenumber)
                waveletField =
                    wavelet.return_field(xPos, yPos, zPos, xAperture, yAperture)
            case _:
                raise Exception("Invalid Wavelet Type")
        fields = positionsAperture * incidentField * waveletField * dx * dy
        totalField = np.sum(fields)
        return totalField

    def return_figure_total_intensity(self, xPositions: np.ndarray,
                                      yPos: float, zPos: float,
                                      waveletType: str, numberPoints: int) ->
                                                    matplotlib.figure.Figure:
        """
        :param xPositions: numpy array of x positions along screen to calculate
                           intensity at (meters)
        :param yPos: y position on screen (meters)
        :param zPos: z position of screen, i.e. distance from aperture (meters)
        :param waveletType:  diffraction wavelet theory, either 'Huygens',
                             'Fresnel', or 'Fraunhofer'
        :param numberPoints: number of points of integration
        :returns: figure of normalized intensity along x (diffraction pattern)
        """
        intensities = []
        for xPos in xPositions:
            intensity = np.abs(
                self.return_total_field(xPos, yPos, zPos, waveletType,
                                        numberPoints) )**2
            intensities.append(intensity)
        intensities = np.array(intensities)
        intensities /= np.max(intensities)
        figure = plt.figure(0, figsize=(6,4))
        ax = figure.add_axes([0.1, 0.3, 0.8, 0.6])
        xPositions *= 1000 # mm / m
```

```
        ax.plot(xPositions, intensities, color="blue")
        ax.set_xlabel(r"Screen Position Along  $x$ (mm)")
        ax.set_ylabel(r"Normalized Intensity $\frac{I}{I_0}$")
        ax.set_xlim(xPositions[0], xPositions[-1])
        ax.set_ylim(0, 1.1)
        ax.grid()
        ax.set_title(f"Normalized Diffraction Pattern from Aperture, y={yPos} m,
                       z={zPos} m")
        return figure
```

Wavelet.py

```
import numpy as np

class Huygens():
    def __init__(self, wavenumber: float):
        """
        :param wavenumber: wave number (inverse meters)
        --------------------------------
        Simulates single Huygens wavelet at aperture (or obstacle)
        """
        self.wavenumber = wavenumber

    def return_field(self, xPos: float, yPos: float, zPos: float,
                           xAperture: float, yAperture: float) -> float:
        """
        :param xPos: x position (meters)
        :param yPos: y position (meters)
        :param zPos: z position, i.e. distance from aperture (or obstacle) (meters)
        :param xAperture: wavelet x location at aperture (or obstacle) (meters)
        :param yAperture: wavelet y location at aperture (or obstacle) (meters)
        :returns: field at some point from wavelet located at aperture (or obstacle)
        """
        distance = np.sqrt((xPos-xAperture)**2 + (yPos-yAperture)**2 + zPos**2)
        field = -1j * self.wavenumber * np.exp(1j * self.wavenumber * distance) /
                (2 * np.pi * distance)
        return field

class Fresnel():
    def __init__(self, wavenumber):
        """
        :param wavenumber: wave number (inverse meters)
        --------------------------------
        Simulates single Fresnel wavelet at aperture (or obstacle)
        """
        self.wavenumber = wavenumber

    def return_field(self, xPos, yPos, zPos, xAperture, yAperture):
        """
        :param xPos: x position (meters)
```

```
        :param yPos: y position (meters)
        :param zPos: z position, i.e. distance from aperture (or obstacle) (meters)
        :param xAperture: wavelet x location at aperture (or obstacle) (meters)
        :param yAperture: wavelet y location at aperture (or obstacle) (meters)
        :returns: field at some point from wavelet located at aperture (or obstacle)
        """
        coefficient = (-1j * self.wavenumber) / (2 * np.pi * zPos)
        exponent1 = 1j * self.wavenumber * zPos
        exponent2 = (1j * self.wavenumber * (xPos**2+yPos**2)) / (2 * zPos)
        exponent3 = (-1j * self.wavenumber * (xPos*xAperture+yPos*yAperture)) / zPos
        exponent4 = (1j * self.wavenumber * (xAperture**2+yAperture**2)) / (2 * zPos)
        field = coefficient * np.exp(exponent1+exponent2+exponent3+exponent4)
        return field

class Fraunhofer():
    def __init__(self, wavenumber):
        """
        :param wavenumber: wave number (inverse meters)
        ------------------------------
        Simulates single Fraunhofer wavelet at aperture (or obstacle)
        """
        self.wavenumber = wavenumber

    def return_field(self, xPos, yPos, zPos, xAperture, yAperture):
        """
        :param xPos: x position (meters)
        :param yPos: y position (meters)
        :param zPos: z position, i.e. distance from aperture (or obstacle) (meters)
        :param xAperture: wavelet x location at aperture (or obstacle) (meters)
        :param yAperture: wavelet y location at aperture (or obstacle) (meters)
        :returns: field at some point from wavelet located at aperture (or obstacle)
        """
        coefficient = (-1j * self.wavenumber) / (2 * np.pi * zPos)
        exponent1 = 1j * self.wavenumber * zPos
        exponent2 = (1j * self.wavenumber * (xPos**2+yPos**2)) / (2 * zPos)
        exponent3 = (-1j * self.wavenumber * (xPos*xAperture+yPos*yAperture)) / zPos
        field = coefficient * np.exp(exponent1+exponent2+exponent3)
        return field
```