Diffraction Guru

Class 12 Computer Science Project

Anand Balivada 12B 1
Dhruv Poduval 12B 9

Acknowledgements

We would like to thank our Computer Science teacher Mrs. Suguna Ganesh for her support and valuable feedback during the course of the project, and National Public School Indiranagar and Microsoft for facilitating communication channels, allowing school to be run virtually.

Introduction

Diffraction is a fascinating and ubiquitous phenomenon observed in various kinds of waves; electromagnetic, acoustic, fluid surface waves, etc.

Our project aims to simulate the diffraction of light, microwave radiation, and radio waves through a rectangular aperture.

The physical setting of the simulation is light with an electric field of unit amplitude ($E(x) = e^{i\kappa x}$, $\kappa = \frac{2\pi}{\lambda}$) and wavelength '$\lambda$' (adjustable by the user; ranging from 0 to $6000 * 10^{-7}$ m), passing through a rectangular aperture whose length and breadth (each ranging from 0 to 3000 micrometers, i.e., 3mm) can be varied by the user.

A screen is placed at a variable length (from 0 to 10m), parallel to the aperture, and the pattern that would form on the screen is the object of the simulation.

The pattern is displayed as a contour plot of the intensity of light ($|E|^2$) on the screen, and is accompanied with graphs of the intensity along the x and y axes, passing through the centre of the screen.

We hope that this can be used by students to gain an intuitive understanding of the phenomenon of diffraction. This is

especially true for understanding how the diffraction pattern is influenced by the wavelength of light, as conducting diffraction experiments at home is extremely difficult and usually only possible for wavelengths of light in the visible region.

Our project is capable of simulating both Fresnel (near-field) and Fraunhofer (far-field) diffraction. However, the limits set in the GUI sliders are primarily for diffraction in the Fraunhofer Region, as the patterns are more distinctive and well-known, aiding students' initial intuitions about the subject.

Project Architecture

The project consists of 2 python files; GUI3.py and Monte_Carlo_Fresnel.py, which handle the front-end and back-end respectively.

i) GUI3.py -
GUI3.py is the single file that must be executed to actually run the project.
The modules imported are:

- Tkinter; a user-interface library
- Matplotlib; a mathematical plotting library
- Numpy; a numerical computation library
- Monte_Carlo_Fresnel.py; which consists of the algorithm to calculate the diffraction pattern.

ii) Monte_Carlo_Fresnel.py -
This is capable of running independently of GUI3.py, but only returns to the shell. For those interested in the exact numerical values of the intensity on the screen, it can be used.
For the purposes of this project, we only import Monte_Carlo_Fresnel.py into GUI3.py.
The modules imported are:

- Numpy
- Matplotlib
- Math
- Random.

Mathematical Basis

Monte_Carlo_Fresnel.py performs the calculation of the electric field on the plane of the screen, with the Fresnel-Kirchhoff Diffraction Integral, based on its values on the plane of the aperture (See [2] for further information).

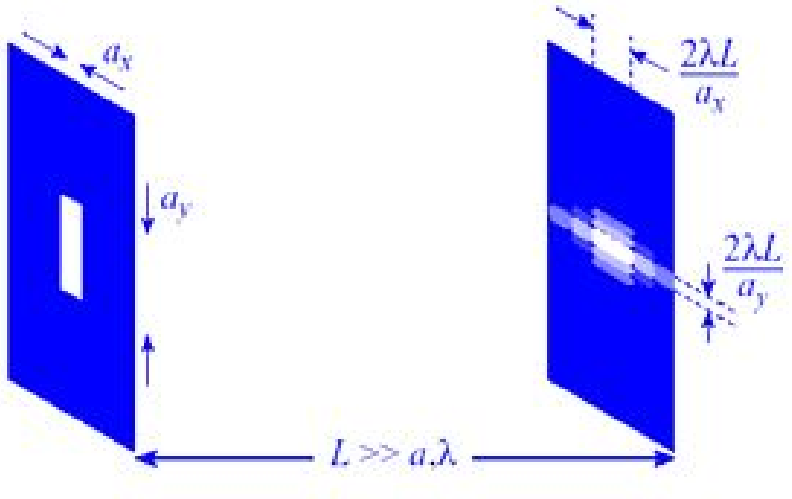


Figure 1: Diffraction through Rectangular Aperture

Source: http://labman.phys.utk.edu/phys222core/modules/m9/resolving_power.htm

The coordinate system on the plane of the screen is (x, y, z), where z is the distance of the screen from the aperture.
The aperture is of dimensions: length *w*, breadth *b,* with coordinate system (x', y', 0).
E(x',y',0) = 1 for (x', y') $\in [-\frac{b}{2}, \frac{b}{2}] \times [-\frac{w}{2}, \frac{w}{2}]$, 0 otherwise.

$$E(x,y,z) = \frac{1}{i\lambda} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} E(x',\, y',0) \frac{e^{ikr}}{r} cos(\theta) dx'dy' \quad \text{, where}$$

$$r = \sqrt{(x-x')^2 + (y-y')^2 + (z-z')^2} \quad \text{and} \quad cos(\theta) = \frac{z}{r}.$$

Thus, to find the electric field at a single point on the screen, an integral is performed over the aperture. Then $|E(x,y,z)|^2$ is evaluated to give the intensity at the point (x, y, z). We only calculate the integral for $(x,\, y,\, z) \in [\frac{-35(w+b)}{4}, \frac{35(w+b)}{4}] \times [\frac{-35(w+b)}{4}, \frac{35(w+b)}{4}] \times \{z\}$
The choice of the above values is a balance between accuracy, computation time and effective visibility of the diffraction pattern (if the region is too large, or too small, no real information is conveyed).

A monte carlo integration method is used to evaluate the integral approximately, to a relatively high degree of accuracy.

In monte carlo integration, 'n' points are sampled at random from the region. The integral of the function 'f' over the region

is given by V<f> + O($\frac{1}{\sqrt{n}}$), where V is the volume (In our case, area) of the region, and <f> is the average value of the function on the 'n' points. The error O($\frac{1}{\sqrt{n}}$) doesn't change with the dimension of the domain of integration, and thus this method was chosen for our integration over a 2-dimensional domain; minimising the runtime and memory. The approximation error in many other popular numerical integration algorithms scales as $O(\frac{1}{\sqrt[d]{N}})$ for dimension d; if n points are used in 1 dimension, n^2 points need to be used for d = 2 to maintain the same error, etc. See [1] for more details and additional references.

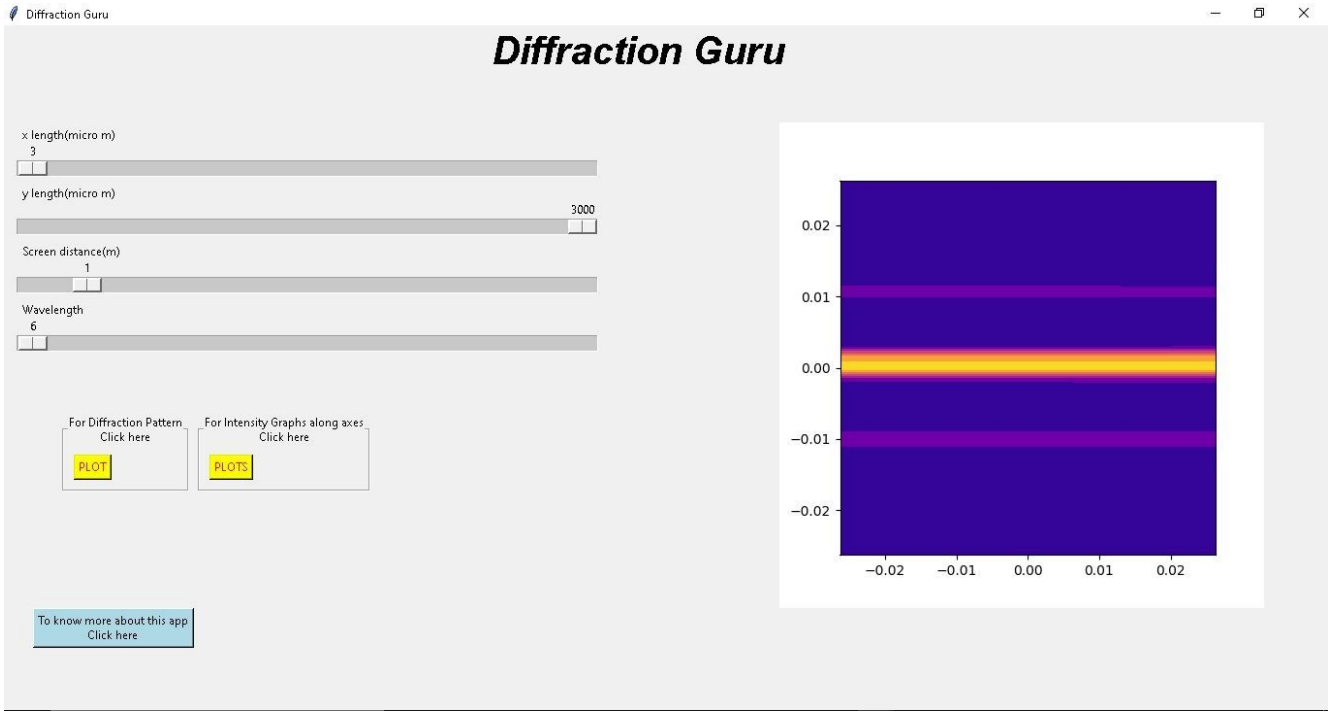
Algorithm

1. The parameters xsize (corresponding to '*b*' above), ysize (corresponding to *'w'* above), z, and l (corresponding to λ) are the arguments of function 'pattern' which returns a 2D numpy array phi((k,k)), where k = 35. phi, when plotted gives $|E(x,y,z)|^2$ at 1225 (k^2) points in the region $(x,\ y,\ z) \in [\frac{-35(w+b)}{4}, \frac{35(w+b)}{4}] \times [\frac{-35(w+b)}{4}, \frac{35(w+b)}{4}] \times \{z\}$
2. n = 35 points ((x', y')) are randomly sampled in the rectangle $[-\frac{xsize}{2}, \frac{xsize}{2}] \times [-\frac{ysize}{2}, \frac{ysize}{2}]$, and stored in numpy array montepoints((n,2)).
3. The real and imaginary parts of the electric field, phi_real and phi_im are calculated using the monte-carlo formula

applied to $f = E(x', y', 0)\frac{e^{ikr}}{r}cos(\theta)$ (f is split into real and imaginary parts using Euler's formula), at all the screen points (x,y, z) = (-k*kstep/2 + itx*kstep, -k*kstep/2 + ity*kstep, z), where itx and ity range from 0 to k (i.e., 0 to 35), and kstep = $\frac{xsize + ysize}{2}$.

4. Then phi is evaluated as (phi_real^2 + phi_im^2)/(l^2).
5. This algorithm runs in $O(nk^2)$ time, and $O(n + k^2)$ memory.
6. The function 'pattern' is called from GUI3.py, when the 'PLOT' button is clicked. The values on the sliders are the arguments for 'pattern'.
7. Finally, the filled contour plot of array phi is plotted on the same tkinter window as GUI3.py, making use of the 'canvas' feature of tkinter and matplotlib.

Samples:



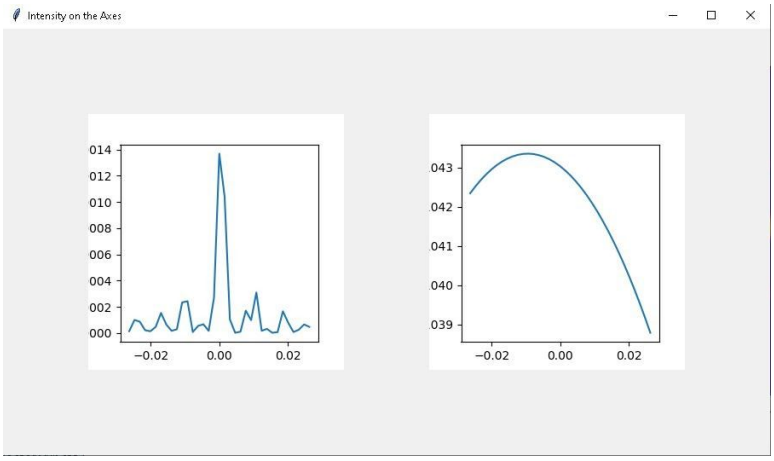


Figure 2: Rectangular aperture with length along y as 3mm, and length along x as 3 microns. The wavelength of light is $6 * 10^{-7}$ m, and the screen is placed 1m away. The intensity graph on LHS is along the y axis, and the RHS is along the x-axis. This corresponds to the well-known case of single slit diffraction, with the slit along the y axis. It is depicted in Figure 1 above.

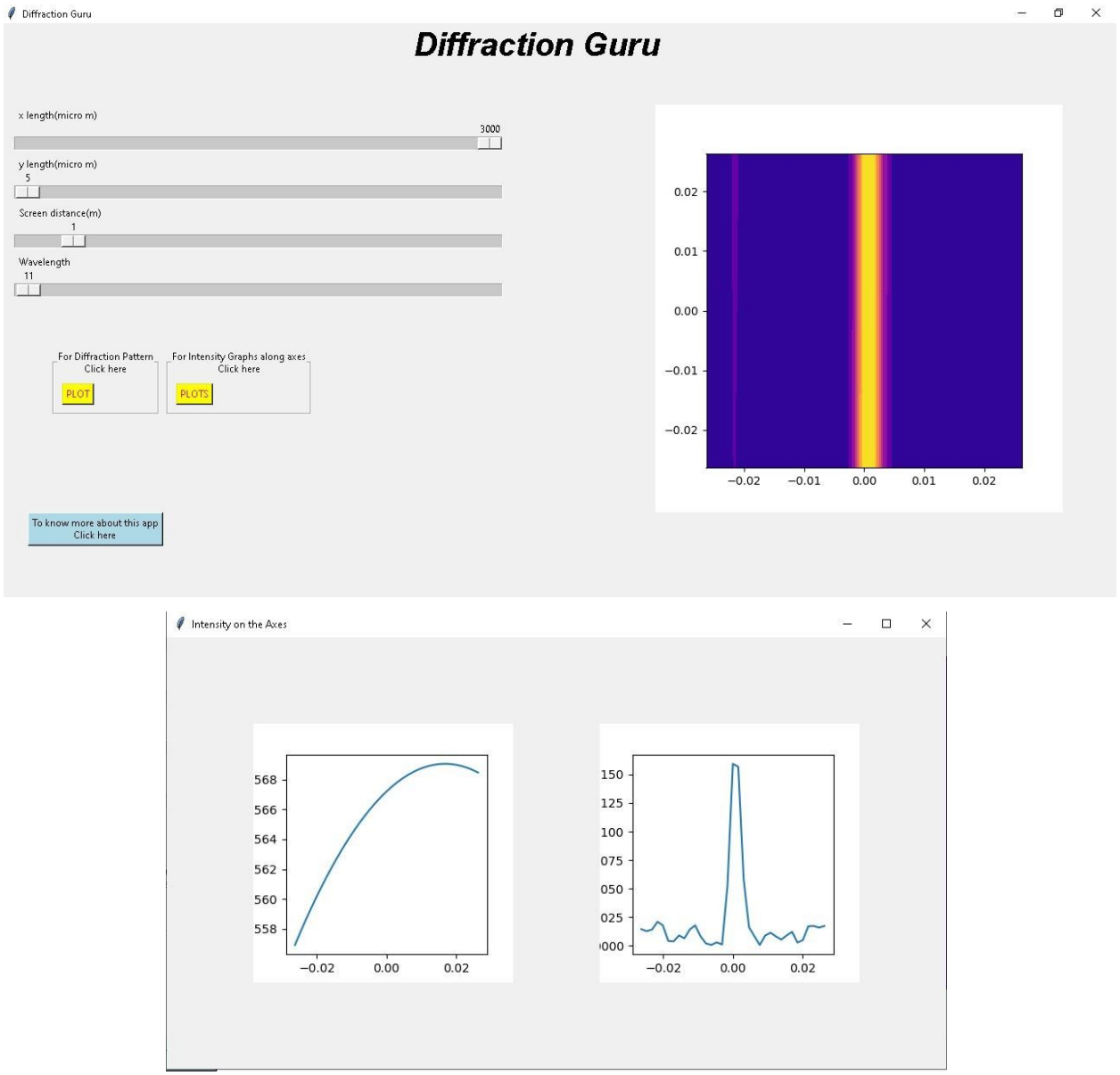


Figure 3: Rectangular aperture with length along x as 3mm and length along y as 3 microns. The wavelength of light is $15 * 10^{-7}$ m and the screen is placed 1m away. The intensity graph on the LHS is along the y axis and the RHS is along the x-axis. This corresponds to the well-known case of single slit diffraction with the slit along the x-axis.

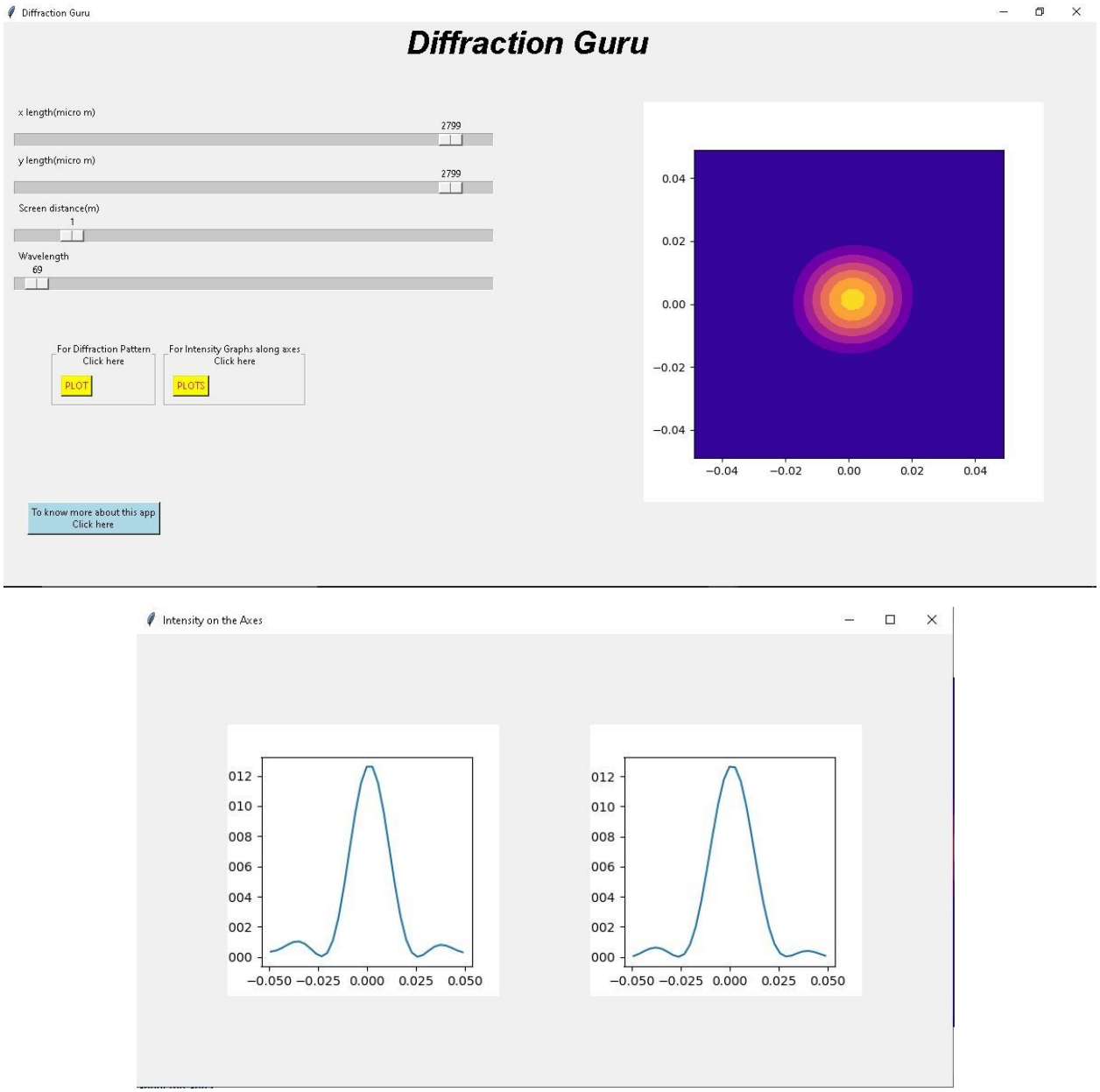


Figure 4: Diffraction through an approximately square aperture of side ~ 2.79mm. The wavelength of light is $69 * 10^{-7}$ m and the screen is 1m away from the aperture.

Source Code

Monte_Carlo_Fresnel.py

```
import numpy as np
import math
```

```
import random
import matplotlib.pyplot as plt

pi = 3.1415926535
def E(x,y,z = 2,l = 0.000006):
    ''' integrand of fresnel-kirchoff diffraction integral'''
    return complex(z*(x**2 + y**2 + z**2)**(-2)*np.cos(2*pi*(x**2 +
y**2 + z**2)**(0.5)/l), z*(x**2 + y**2 +
z**2)**(-2)*np.sin(2*pi*(x**2 + y**2 + z**2)**(0.5)/l))

def pattern(xsize = 0.003, ysize = 0.00003, z = 1, l = 0.000006):
    V = xsize*ysize   #area
    n = 35              #number of points to sample at
    k = 35              #O(k^2) is number of points at which field
components (and hence intensity) are calculated
    kstep = (xsize + ysize)/2         #Step size for above side
lengths

    #Random Sampling
    montepoints = np.random.rand(n, 2)
    for i in range(n):
        montepoints[i][0] =
montepoints[i][0]*xsize*0.5*(-1)**(random.randint(1, 4))
        montepoints[i][1] =
montepoints[i][1]*ysize*0.5*(-1)**(random.randint(1,4))
    #print(montepoints)

    #Plot the randomly sampled points
##    plt.close()
##    plt.scatter(np.transpose(montepoints)[0],
np.transpose(montepoints)[1])
##    plt.show()

    #Integration
    phi_real = np.zeros((k,k))
    phi_im = np.zeros((k,k))
    phi = np.zeros((k,k))
    for ity in range(k):
        for itx in range(k):
            f_real = 0
            f2_real = 0
            f_im = 0
            f2_im = 0
            r = [-k*kstep/2 + itx*kstep, -k*kstep/2 + ity*kstep]
            for monti in range(n):
```

```
                f_real += (E(r[0] - montepoints[monti][0], r[1] -
montepoints[monti][1], z, l).real)/n
                f2_real += (E(r[0] - montepoints[monti][0], r[1] -
montepoints[monti][1], z, l).real)**2 /n       #Not needed
                f_im += (E(r[0] - montepoints[monti][0], r[1] -
montepoints[monti][1], z, l).imag)/ n
                f2_im += (E(r[0] - montepoints[monti][0], r[1] -
montepoints[monti][1], z, l).imag)**2 /n         #Not needed
            phi_real[ity][itx] = V*f_real
            phi_im[ity][itx] = V*f_im
            phi[ity][itx] = (phi_real[ity][itx]**2 +
phi_im[ity][itx]**2)/(l**2)
    return phi
```

GUI3.py

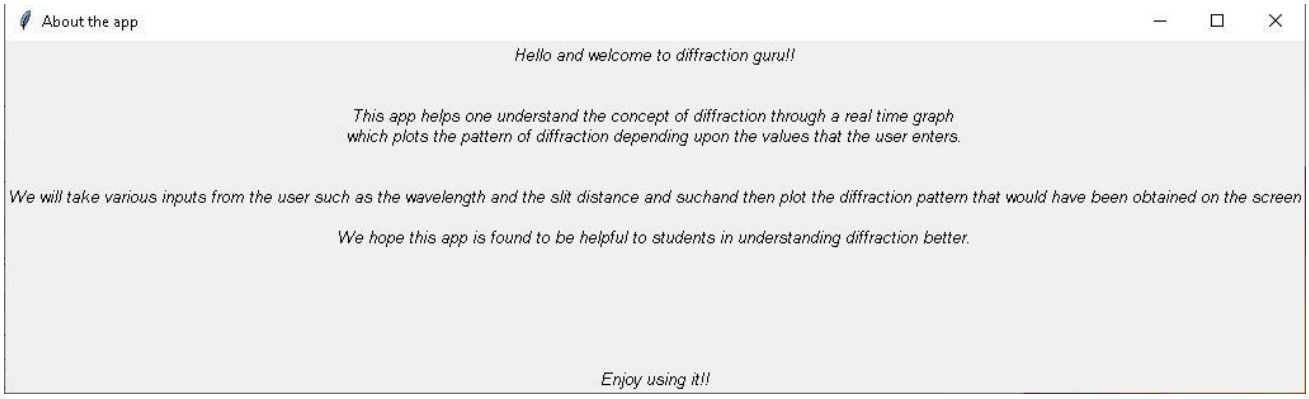
```
from tkinter import *
from Monte_Carlo_Fresnel import *
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.figure import Figure
from matplotlib import cm
from matplotlib.backends.backend_tkagg import (FigureCanvasTkAgg,
NavigationToolbar2Tk)

pat = []

def pl():
    fig = Figure(figsize=(5, 5), dpi=100)
    #fig, ax = plt.subplots(1,1)
    #ax.contourf(pattern(sxsize.get()*10**-6, sysize.get()*10**-6,
sz.get(), sl.get()*10**-6))
    global pat
    pat = pattern(sxsize.get()*10**-6, sysize.get()*10**-6, sz.get(),
sl.get()*10**-6)
    kstep = (sxsize.get()*10**-6 + sysize.get()*10**-6)/2
    k = 35 #Same k as used in random sampling part of
Monte_Carlo_Frensel.pattern()
    fig.add_subplot(111).contourf(np.linspace(-k*0.5*kstep,
k*0.5*kstep, k), np.linspace(-k*0.5*kstep, k*0.5*kstep, k), pat, cmap
= cm.plasma)
    canvas = FigureCanvasTkAgg(fig, master = master)
    canvas.draw()
    canvas.get_tk_widget().place(x=800, y=100)
```

```
def open():
    top=Tk()
    top.title("About the app")
    label=Label(top, text="Hello and welcome to diffraction
guru!!\n\n\nThis app helps one understand the concept of diffraction
through a real time graph \nwhich plots the pattern of diffraction
depending upon the values that the user enters.\n\n\nWe will take
various inputs from the user such as the wavelength and the slit
distance and suchand then plot the diffraction pattern that would
have been obtained on the screen\n\nWe hope this app is found to be
helpful to students in understanding diffraction
better.\n\n\n\n\n\n\nEnjoy using it!!", font=('arial', 10, 'italic'))
    label.grid(row=0, column=0)

def open2():
    top = Tk()
    top.geometry("900x500")
    top.title("Intensity on the Axes")
    fig2 = Figure(figsize = (3,3), dpi=100)
    fig3 = Figure(figsize = (3,3), dpi=100)
    xint = pat[18][0:len(pat[18])]
    yint = []
    k = 35
    kstep = (sxsize.get()*10**-6 + sysize.get()*10**-6)/2
    for i in range(k):
        yint += [pat[i][18]]
    maxy = 0
    for j in yint:
        if j > maxy:
            maxy = j
    fig2.add_subplot(111).plot(np.linspace(-k*0.5*kstep, k*0.5*kstep,
k), yint)
    fig3.add_subplot(111).plot(np.linspace(-k*0.5*kstep, k*0.5*kstep,
k), xint)
    canvas2 = FigureCanvasTkAgg(fig2, master = top)
    canvas2.draw()
    canvas2.get_tk_widget().place(x=100, y=100)
    canvas3 = FigureCanvasTkAgg(fig3, master = top)
    canvas3.draw()
    canvas3.get_tk_widget().place(x=500, y=100)



master = Tk()
```

```
master.title('Diffraction Guru')
master.geometry("2000x2000")

frame=LabelFrame(master, text="For Diffraction Pattern\nClick here",
padx=10, pady=10)
frame.place(x=60, y=400)

label1=Label(master, text="Diffraction Guru",
font=('arial',30,'bold','italic'))
label1.place(x=500, y=0)

sxsize = Scale(master, label="x length(micro m)", from_=0, to=3000,
orient='horizontal', length = 600)
sxsize.set(3)
sxsize.place(x = 10, y = 100)

sysize = Scale(master, label="y length(micro m)", from_=0, to=3000,
orient='horizontal', length = 600)
sysize.set(3000)
sysize.place(x = 10, y = 160)

sz = Scale(master, label="Screen distance(m)", from_=0, to=10,
orient='horizontal', length = 600)
sz.set(1)
sz.place(x = 10, y = 220)

sl = Scale(master, label="Wavelength", from_=0, to=3000,
orient='horizontal', length = 600)
sl.set(6)
sl.place(x = 10, y = 280)

b = Button(frame, text = "PLOT", command=pl, fg="red", bg="yellow")
b.grid(row=300, column = 100)

btn = Button(master, text = "To know more about this app\nClick
here", bg="lightblue",command=open)
btn.place(x=30, y=600)

inteframe = LabelFrame(master, text = "For Intensity Graphs along
axes\nClick here", padx=10, pady=10)
inteframe.place(x = 200, y= 400)

intebtn = Button(inteframe, text = "PLOTS", command = open2 , fg =
"red", bg = "yellow")
intebtn.grid(row = 300, column = 100)
```

```
master.mainloop()
```

About



Contributions

Front-End (UI): Dhruv Poduval
Back-End (Algorithm and Mathematics): Anand Balivada
The net amount of work done by each team member was roughly the same.

References

[1] Numerical Recipes in C: The Art of Scientific Computing, Teukolsky, S.A., Press, W. H., Vetterling W.T, Flannery, B.P. (2nd ed.) (Section 7.6) (For Monte Carlo Integration)
[2] Principles of Optics, Born, M., Wolf, E. (Chapter 8 on Diffraction)
[3] matplotlib.org (Official documentation of matplotlib)
[4] tutorialspoint.com (For tkinter)