<u>COSC 4372 “Fundamentals of Medical Imaging”</u>

X-Ray Machine For Leg Phantom

CONTENT

Members of the Team:

Name	PeopleSoft ID	Department	Undergrad/Grad
Shahd Abu-Obeid	2173299	NSM	Undergrad
Duaa Aslam	1930214	NSM	Undergrad
Muskan Oad	2253232	NSM	Undergrad
Salma Abbady	2050843	NSM	Undergrad

INTRODUCTION

The goal of this project is to create a virtual X-ray system that mimics how a traditional X-ray machine would create plain film X-ray images. Through the use of a graphical user interface (GUI), this system allows users to examine reconstructed images for analysis, control data acquisition, and modify a number of parameters. This simulation aims to improve comprehension of X-ray imaging methods by offering an interactive platform for modifying and evaluating X-ray machine settings.

Developing a 3D human phantom to mimic X-ray imaging for mammography or related applications, as well as building a validation phantom for testing and calibration, are the two main aims of the project. Users may see how the system's parameters affect image quality and reconstruction by adjusting things like beam energy, X-ray angle, and distances. By helping users analyze and comprehend the consequences of various acquisition parameters, this virtual setup hopes to provide a hands-on learning experience in radiography imaging techniques.

Aim 1:

Create a virtual X-ray system that simulates a traditional X-ray machine.

Aim 2:

Develop a 3D human phantom and a validation phantom for testing and calibration.

Aim 3:

Allow users to adjust parameters like beam energy, X-ray angle, and distances.

Aim 4:

Provide an interactive platform to observe and analyze how these parameters impact image quality.

METHODS

1. **Phantoms Generation**

Since phantoms are frequently employed in imaging investigations to replicate human anatomy and evaluate system performance, 3D modeling for phantoms follows basic medical imaging concepts. Comprehending and developing these models facilitates the analysis of how well imaging parameters describe tissue.

2. **Parameter Adjustments and Controls**

Basic imaging concepts are covered by GUI for User Interaction since regulating elements such as beam energy, source angle, and distance is essential for maximizing image quality and comprehending how various parameters impact the final image. In medical imaging, this information is crucial for customizing imaging methods for certain diagnostic objectives.

3. **Image Simulation**

Since they mimic the creation and reconstruction of images in actual X-ray machines, X-ray projection and reconstruction algorithms are essential subjects in medical imaging fundamentals. Effective X-ray picture interpretation and analysis require an understanding of these algorithms.

4. **Contrast and Angle Analysis**

The principles of image contrast and resolution in medical imaging are closely related to contrast calculation algorithms and angle-based analysis. These analyses aid in illustrating how imaging methods distinguish between tissues according to their density and structure, which is crucial for identifying lesions or fractures.

5. **Testing And Validation:**

A key component of medical imaging is the evaluation of imaging systems, which includes parameter variation and testing. To ensure accurate diagnostic results and comprehend how various parameters affect image quality, it is essential to test the effects of variables such as distance, angle, and tissue density.

6. **Result Evaluation And Analysis:**

Using visual and statistical analysis to evaluate image quality is consistent with fundamental imaging concepts. In order to confirm the precision and efficacy of imaging methods and make sure they satisfy diagnostic standards, medical imaging frequently depends on quantitative measures (such as signal-to-noise ratio) and visual evaluations.

REVISED

Code to Create a 3D Leg Phantom with Visualization

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Define dimensions and properties of the phantom
leg_radius = 50     # Radius of the leg (soft tissue) in arbitrary units
bone_radius = 20    # Radius of the bone (inner cylinder) in arbitrary units
height = 100        # Height of the leg phantom in arbitrary units

# Create an empty 3D matrix (phantom) filled with zeros
phantom = np.zeros((height, 2 * leg_radius, 2 * leg_radius))

# Populate the phantom with soft tissue and bone based on distance from
center
for z in range(height):
    for x in range(2 * leg_radius):
        for y in range(2 * leg_radius):
            # Calculate the distance from the center of the cylinder
            distance = np.sqrt((x - leg_radius) ** 2 + (y - leg_radius) ** 2)
            if distance <= bone_radius:
```

```
            # Bone region
            phantom[z, x, y] = 2  # Higher attenuation value for bone
        elif distance <= leg_radius:
            # Soft tissue region
            phantom[z, x, y] = 1  # Lower attenuation value for soft tissue

# Function to simulate an orthogonal split
def add_orthogonal_split(phantom, split_z_start, split_z_end):
    """
    Simulate an orthogonal split in the phantom by setting attenuation values to
zero
    in the specified z-range.
    """
    phantom[split_z_start:split_z_end, :, :] = 0

# Function to simulate an angled split
def add_angled_split(phantom, m, n, x0, y0, z0):
    """
    Simulate an angled split in the phantom by setting attenuation values to
zero
    where z >= m*(x - x0) + n*(y - y0) + z0
    """
    height, width_x, width_y = phantom.shape

    # Generate coordinate grids
    z_indices = np.arange(height)[:, np.newaxis, np.newaxis]
    x_indices = np.arange(width_x)[np.newaxis, :, np.newaxis]
    y_indices = np.arange(width_y)[np.newaxis, np.newaxis, :]
    # Calculate the plane equation
    plane = m * (x_indices - x0) + n * (y_indices - y0) + z0

    # Create a mask where the attenuation values will be set to zero
    mask = z_indices >= plane

    # Apply the mask to the phantom
    phantom[mask] = 0

# Function to visualize a cross-section of the phantom
def visualize_phantom(phantom, slice_index):
    plt.figure(figsize=(8, 8))
    plt.imshow(phantom[slice_index], cmap="gray", origin='lower')
    plt.title(f"Cross-section of the Leg Phantom at Slice {slice_index}")
    plt.xlabel("X-axis")
    plt.ylabel("Y-axis")
    plt.colorbar(label="Attenuation Value")
    plt.show()

# Visualize the middle cross-section of the phantom before adding splits
visualize_phantom(phantom, height // 2)

# Save a copy of the original phantom for later comparison
phantom_original = phantom.copy()

# Add an orthogonal split
split_z_start = height // 2 + 5   # Adjusted to avoid zeroing out the middle slice
split_z_end = height // 2 + 15
add_orthogonal_split(phantom, split_z_start, split_z_end)
```

```
# Visualize a slice just before the split to observe the effect
visualize_phantom(phantom, height // 2 + 5)

# Reset the phantom to the original before adding the angled split
phantom = phantom_original.copy()

# Add an angled split
# Parameters for the angled plane
m = -0.5  # Negative slope to affect the upper part of the phantom
n = 0     # Slope in y-direction
x0 = leg_radius   # Center x-coordinate
y0 = leg_radius   # Center y-coordinate
z0 = height // 2  # The plane passes through the middle of the phantom

add_angled_split(phantom, m, n, x0, y0, z0)

# Visualize a slice affected by the angled split
visualize_phantom(phantom, height // 2 + 10)

# Function to plot attenuation profile along a line
def plot_attenuation_profile(phantom, z_slice, y_coord):
    attenuation_profile = phantom[z_slice, :, y_coord]
    plt.figure(figsize=(8, 4))
    plt.plot(attenuation_profile)
    plt.title(f"Attenuation Profile at Z={z_slice}, Y={y_coord}")
    plt.xlabel("X-axis")
    plt.ylabel("Attenuation Value")
    plt.show()
```

```python
# Plot attenuation profiles before and after splits
# Before splits
phantom = phantom_original.copy()
plot_attenuation_profile(phantom, height // 2, leg_radius)

# After orthogonal split
phantom_with_orthogonal_split = phantom_original.copy()
add_orthogonal_split(phantom_with_orthogonal_split, split_z_start,
split_z_end)
plot_attenuation_profile(phantom_with_orthogonal_split, height // 2 + 5,
leg_radius)

# After angled split
phantom_with_angled_split = phantom_original.copy()
add_angled_split(phantom_with_angled_split, m, n, x0, y0, z0)
plot_attenuation_profile(phantom_with_angled_split, height // 2 + 10,
leg_radius)

# 3D Visualization (optional)
def visualize_3d_phantom(phantom):
    fig = plt.figure(figsize=(10, 10))
    ax = fig.add_subplot(111, projection="3d")

    # Get coordinates where the phantom has bone (attenuation value 2) and
soft tissue (attenuation value 1)
    bone_coords = np.where(phantom == 2)
    soft_tissue_coords = np.where(phantom == 1)
    # Plot the bone in red
    ax.scatter(bone_coords[1], bone_coords[2], bone_coords[0], color="red", alpha=0.3, s=1, label="Bone")

    # Plot the soft tissue in blue
    ax.scatter(soft_tissue_coords[1], soft_tissue_coords[2], soft_tissue_coords[0], color="blue", alpha=0.1, s=1, label="Soft Tissue")

    ax.set_title("3D Visualization of the Leg Phantom with Splits")
    ax.set_xlabel("X-axis")
    ax.set_ylabel("Y-axis")
    ax.set_zlabel("Height (Z-axis)")
    ax.legend()
    plt.show()

# Uncomment the line below to visualize the 3D structure (optional, may take time for large arrays)
# visualize_3d_phantom(phantom_with_angled_split)
```

Phantom Creation:

- We define the dimensions and properties of the phantom with an outer cylinder (soft tissue) and an inner cylinder (bone).
- For each slice (z-axis), we check the distance of each point in the cross-sectional plane from the center to determine whether it's in the bone or soft tissue region.

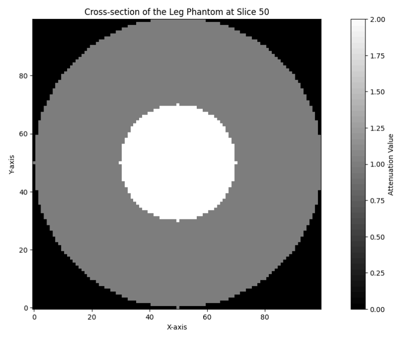
Cross-Section Visualization:

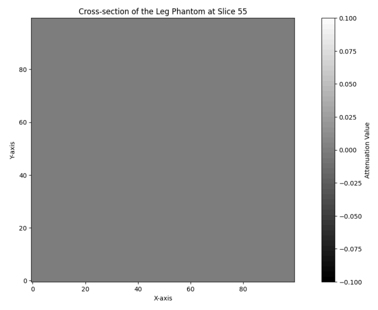
- The visualize_phantom function displays a 2D cross-section of the phantom at a specified slice index (in this case, the middle slice) using a grayscale colormap.

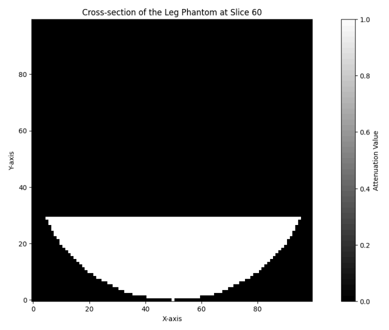
3D Visualization (Optional):

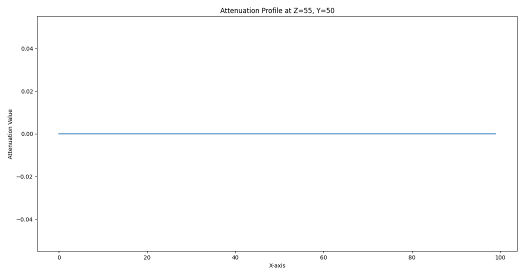
- The visualize_3d_phantom function provides a 3D scatter plot where bone and soft tissue are shown in different colors.
- This step is optional, as 3D visualization can be computationally intensive.

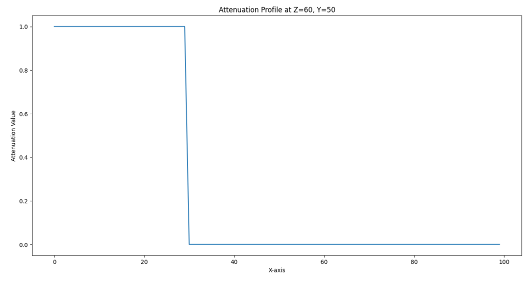
Output

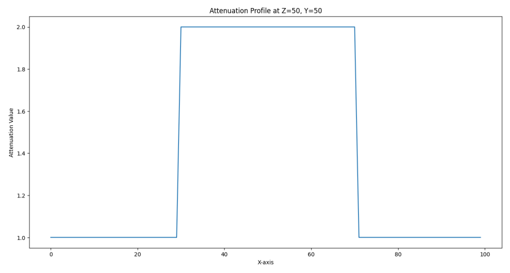












1. **Outer Circle (Soft Tissue):**

   The large, outer gray region represents the soft tissue, which should have a uniform attenuation value, as shown.

2. **Inner Circle (Bone):**

The smaller, inner white circle represents the bone with a higher attenuation value, which is typical in medical imaging since bones block more X-rays and appear lighter.

3. **Background:**

   The black area outside the phantom represents empty space or air, which should have an attenuation value of zero.

The gradient on the right clearly shows the attenuation values, with the outer layer (soft tissue) having a lower value than the inner core (bone). This matches the expected structure of a leg cross-section, with distinct bone and soft tissue regions.

REVISED

Parameter Adjustment GUI

```
import tkinter as tk
from tkinter import ttk
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import numpy as np
from scipy.ndimage import rotate

# Phantom generation function
def generate_leg_phantom(leg_radius=50, bone_radius=20, height=100):
    phantom = np.zeros((height, 2 * leg_radius, 2 * leg_radius))
    for z in range(height):
        for x in range(2 * leg_radius):
            for y in range(2 * leg_radius):
                distance = np.sqrt((x - leg_radius) ** 2 + (y - leg_radius) ** 2)
                if distance <= bone_radius:
                    phantom[z, x, y] = 2  # Bone region
            elif distance <= leg_radius:
                phantom[z, x, y] = 1  # Soft tissue region
    return phantom

# Apply transformations for angle, beam energy, and distance
def adjust_phantom_slice(slice_image, angle, beam_energy, source_distance):
    # Determine the padding size to avoid cropping during rotation
    pad_x = slice_image.shape[0] // 2
    pad_y = slice_image.shape[1] // 2

    # Pad the slice symmetrically
    padded_slice = np.pad(slice_image, ((pad_x, pad_x), (pad_y, pad_y)),
mode='constant', constant_values=0)

    # Rotate the padded slice
    rotated_slice = rotate(padded_slice, angle, reshape=False, mode='nearest')

    # Crop back to the original size
    start_x = (rotated_slice.shape[0] - slice_image.shape[0]) // 2
    start_y = (rotated_slice.shape[1] - slice_image.shape[1]) // 2
    cropped_slice = rotated_slice[start_x:start_x + slice_image.shape[0],
start_y:start_y + slice_image.shape[1]]

    # Adjust intensity based on beam energy and source distance
    attenuation_factor = (beam_energy / 100) * (200 / source_distance)
    adjusted_slice = cropped_slice * attenuation_factor

    # Normalize the values for display (to avoid clipping or too dark visuals)
    adjusted_slice = np.clip(adjusted_slice / adjusted_slice.max(), 0, 1)
    return adjusted_slice



# GUI for parameter adjustment
class XRaySimulatorApp:
    def __init__(self, root):
        self.root = root
        self.root.title("X-Ray Simulator - Leg Phantom")

        self.beam_energy = tk.DoubleVar(value=50.0)
        self.xray_angle = tk.DoubleVar(value=0.0)
        self.source_distance = tk.DoubleVar(value=100.0)

        self.phantom = generate_leg_phantom()
        self.setup_gui()

    def setup_gui(self):
        ttk.Label(self.root, text="Beam Energy (keV)").grid(row=0, column=0,
padx=10, pady=5)
        self.beam_energy_slider = ttk.Scale(
            self.root, from_=30, to=100, variable=self.beam_energy,
command=self.update_preview)
        self.beam_energy_slider.grid(row=0, column=1, padx=10, pady=5)

        ttk.Label(self.root, text="X-Ray Angle (degrees)").grid(row=1,
column=0, padx=10, pady=5)
        self.xray_angle_slider = ttk.Scale(
            self.root, from_=0, to=180, variable=self.xray_angle,
command=self.update_preview)
        self.xray_angle_slider.grid(row=1, column=1, padx=10, pady=5)

        ttk.Label(self.root, text="Source Distance (cm)").grid(row=2, column=0,
padx=10, pady=5)
        self.source_distance_slider = ttk.Scale(
            self.root, from_=50, to=200, variable=self.source_distance,
command=self.update_preview)
        self.source_distance_slider.grid(row=2, column=1, padx=10, pady=5)

        self.fig, self.ax = plt.subplots(figsize=(4, 4))
        self.ax.axis('off')
        self.image_canvas = FigureCanvasTkAgg(self.fig, master=self.root)
        self.image_canvas.get_tk_widget().grid(row=3, column=0, columnspan=3,
padx=10, pady=10)

        self.visualize_button = ttk.Button(
            self.root, text="Visualize Full Phantom Slice",
command=self.open_visualization_window)
        self.visualize_button.grid(row=4, column=0, columnspan=3, pady=10)

        self.update_preview()

    def update_preview(self, event=None):
        slice_index = self.phantom.shape[0] // 2
        slice_image = self.phantom[slice_index]

        adjusted_image = adjust_phantom_slice(
            slice_image,
            angle=self.xray_angle.get(),
            beam_energy=self.beam_energy.get(),
            source_distance=self.source_distance.get()
        )

        self.ax.clear()
        self.ax.imshow(adjusted_image, cmap="gray", origin="lower")
        self.ax.set_title(
            f"Beam Energy={self.beam_energy.get():.1f} keV, "
            f"Angle={self.xray_angle.get():.1f}°,
Distance={self.source_distance.get():.1f} cm"
        )
        self.ax.axis('off')
        self.image_canvas.draw()

    def open_visualization_window(self):
        """Open a new window to display the full visualization with details."""
        slice_index = self.phantom.shape[0] // 2
        slice_image = self.phantom[slice_index]

        # Apply transformations based on user inputs
        adjusted_image = adjust_phantom_slice(
            slice_image,
            angle=self.xray_angle.get(),
            beam_energy=self.beam_energy.get(),
            source_distance=self.source_distance.get()
        )
```

```
# Create a new window
new_window = tk.Toplevel(self.root)
new_window.title("Phantom Slice Visualization")

# Create the figure
fig, ax = plt.subplots(figsize=(6, 6))
ax.imshow(adjusted_image, cmap="gray", origin="lower")
ax.set_title("Full Phantom Slice Visualization")
ax.axis("off")

# Display beam energy, intensity, and angle
beam_energy = self.beam_energy.get()
xray_angle = self.xray_angle.get()
source_distance = self.source_distance.get()

# Add annotations to the plot
ax.text(
    0.05, 0.95,
    f"Beam Energy: {beam_energy:.1f} keV",
    transform=ax.transAxes,
    fontsize=12,
    color="white",
    verticalalignment="top"
)
ax.text(
    0.05, 0.90,
    f"Angle: {xray_angle:.1f}°",
    transform=ax.transAxes,
    fontsize=12,
            color="white",
            verticalalignment="top"
        )
        ax.text(
            0.05, 0.85,
            f"Source Distance: {source_distance:.1f} cm",
            transform=ax.transAxes,
            fontsize=12,
            color="white",
            verticalalignment="top"
        )

        # Embed the figure into the Tkinter window
        canvas = FigureCanvasTkAgg(fig, master=new_window)
        canvas.get_tk_widget().pack()
        canvas.draw()


# Run the application
root = tk.Tk()
app = XRaySimulatorApp(root)
root.mainloop()
```

This GUI will allow you to modify:

1. **Beam Energy:**

   Simulated as an adjustable parameter.

2. **X-ray Angle:**

   Angle of the X-ray beam.

3. **Distance between Source and Phantom:**

   Adjusts the distance to simulate its effect on the image.

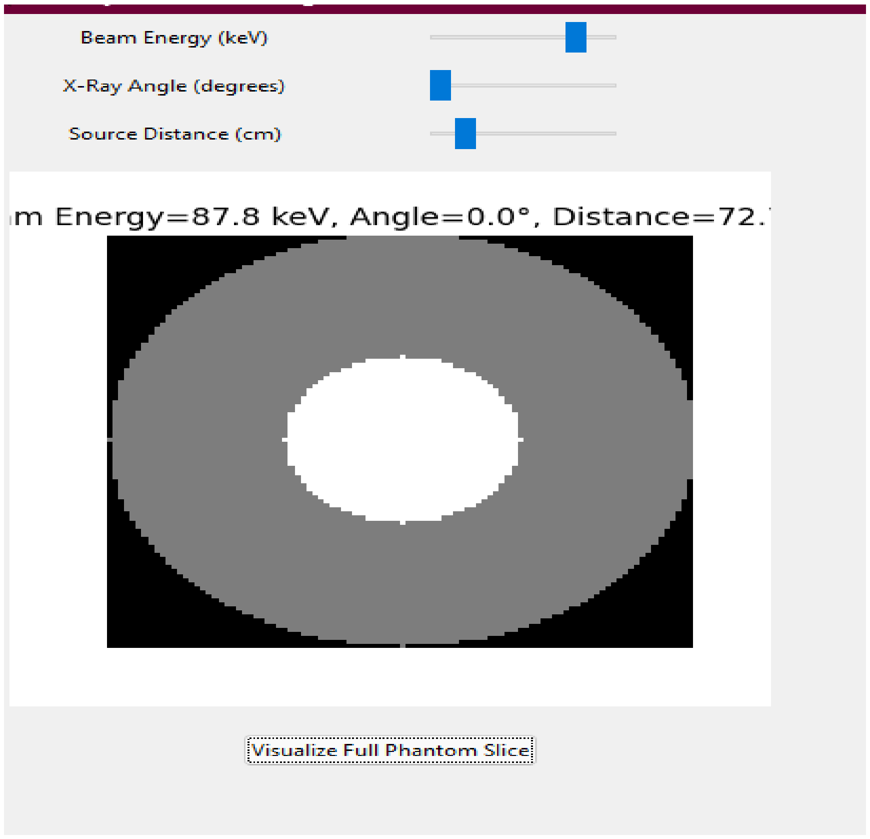
This code uses tkinter for the GUI to let users interactively modify these parameters. The GUI will not perform the full X-ray simulation but will display the adjusted values, which you could later integrate with the image simulation function.

Each slider triggers the update_simulation function, which currently prints parameter values. In a full simulation, this function would be expanded to adjust the actual image simulation.

Visualizing Phantom Slice:

- A button labeled "Visualize Phantom Slice" opens a visualization of the middle slice of the phantom to verify its structure.
- The visualize_phantom_slice function displays the specified slice in grayscale, useful for confirming the phantom setup.

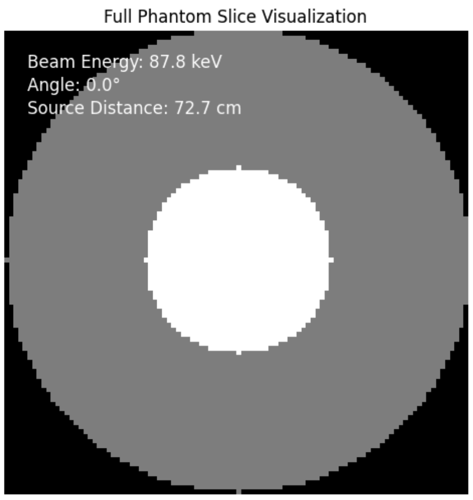
Output


Beam Energy (keV)

X-Ray Angle (degrees)

Source Distance (cm)

m Energy=87.8 keV, Angle=0.0°, Distance=72.


Full Phantom Slice Visualization

Beam Energy: 87.8 keV
Angle: 0.0°
Source Distance: 72.7 cm

REVISED

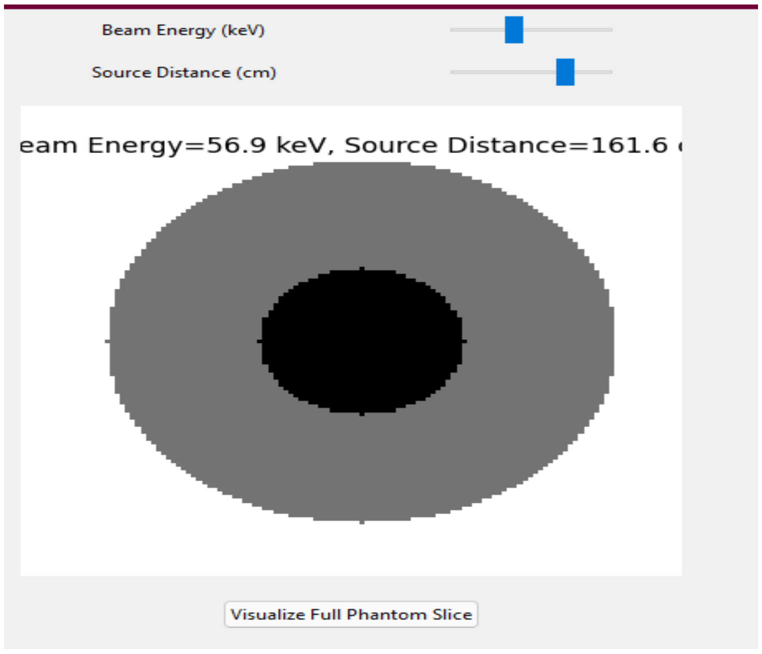
X-Ray Stimulation and Image Reconstruction

```python
import tkinter as tk

from tkinter import Tk, Toplevel, ttk

import matplotlib.pyplot as plt

from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

import numpy as np

from scipy.ndimage import rotate  # For smooth angle rotation




# Function from X-ray Simulation File

def generate_leg_phantom(leg_radius=50, bone_radius=20, height=100):

    phantom = np.zeros((height, 2 * leg_radius, 2 * leg_radius))

    for z in range(height):

        for x in range(2 * leg_radius):

            for y in range(2 * leg_radius):

                distance = np.sqrt((x - leg_radius) ** 2 + (y - leg_radius) ** 2)

                if distance <= bone_radius:

                    phantom[z, x, y] = 2  # Bone region

                elif distance <= leg_radius:

                    phantom[z, x, y] = 1  # Soft tissue region
    return phantom



def simulate_xray_image(phantom, beam_energy, source_distance):
    height, width, depth = phantom.shape
    reconstructed_image = np.zeros((width, depth))

    # Attenuation factors based on beam energy and tissue type
    bone_attenuation = 0.5 + (beam_energy / 100) * 0.5  # Adjustable factor for
bone
    tissue_attenuation = 0.2 + (beam_energy / 100) * 0.3  # Adjustable factor for
tissue

    # Calculate the X-ray intensity at each point by summing attenuations
through the depth
    for x in range(width):
        for y in range(depth):
            attenuation_sum = 0
            for z in range(height):
                if phantom[z, x, y] == 2:  # Bone region
                attenuation_sum += bone_attenuation
            elif phantom[z, x, y] == 1:  # Soft tissue region
                attenuation_sum += tissue_attenuation
        # Calculate intensity based on attenuation and distance
        reconstructed_image[x, y] = np.exp(-attenuation_sum /
source_distance)

    return reconstructed_image


# GUI for parameter adjustment
class XRaySimulatorApp:
    def __init__(self, root):
        self.root = root
        self.root.title("X-Ray Simulator - Leg Phantom")

        self.beam_energy = tk.DoubleVar(value=50.0)
        self.source_distance = tk.DoubleVar(value=100.0)
        self.phantom = generate_leg_phantom()
        self.setup_gui()
    def setup_gui(self):
        ttk.Label(self.root, text="Beam Energy (keV)").grid(row=0, column=0, padx=10, pady=5)
        self.beam_energy_slider = ttk.Scale(
            self.root, from_=30, to=100, variable=self.beam_energy, command=self.update_preview)
        self.beam_energy_slider.grid(row=0, column=1, padx=10, pady=5)

        ttk.Label(self.root, text="Source Distance (cm)").grid(row=1, column=0, padx=10, pady=5)
        self.source_distance_slider = ttk.Scale(
            self.root, from_=50, to=200, variable=self.source_distance, command=self.update_preview)
        self.source_distance_slider.grid(row=1, column=1, padx=10, pady=5)

        self.fig, self.ax = plt.subplots(figsize=(4, 4))
        self.ax.axis('off')
        self.image_canvas = FigureCanvasTkAgg(self.fig, master=self.root)
        self.image_canvas.get_tk_widget().grid(row=2, column=0, columnspan=2, padx=10, pady=10)
        self.visualize_button = ttk.Button(
            self.root, text="Visualize Full Phantom Slice",
command=self.open_visualization_window)
        self.visualize_button.grid(row=3, column=0, columnspan=2, pady=10)


        self.update_preview()


    def update_preview(self, event=None):
        reconstructed_image = simulate_xray_image(
            self.phantom,
            self.beam_energy.get(),
            self.source_distance.get()
        )


        self.ax.clear()
        self.ax.imshow(reconstructed_image, cmap="gray", origin="lower")
        self.ax.set_title(
            f"Beam Energy={self.beam_energy.get():.1f} keV, "
            f"Source Distance={self.source_distance.get():.1f} cm"
        )
        self.ax.axis('off')
        self.image_canvas.draw()

    def open_visualization_window(self):
        """Open a new window to display the full visualization with details."""
        slice_index = self.phantom.shape[0] // 2
        slice_image = self.phantom[slice_index]

        # Apply transformations based on user inputs
        adjusted_image = simulate_xray_image(
            self.phantom,
            self.beam_energy.get(),
            self.source_distance.get()
        )

        # Create a new window
        new_window = tk.Toplevel(self.root)
        new_window.title("Full X-Ray Image Visualization")
```

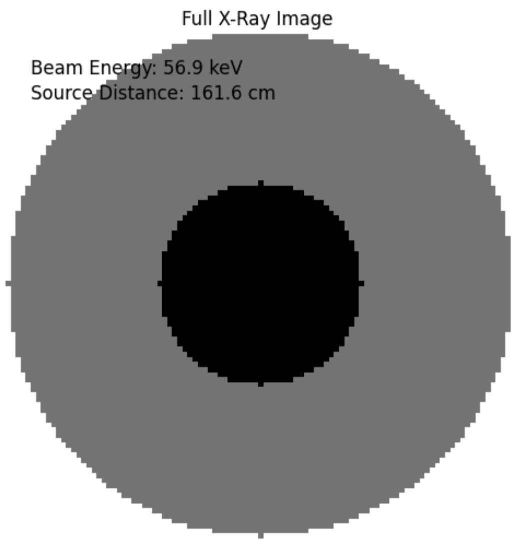
```
# Create the figure

fig, ax = plt.subplots(figsize=(6, 6))

ax.imshow(adjusted_image, cmap="gray", origin="lower")

ax.set_title("Full X-Ray Image")

ax.axis("off")


# Display beam energy, intensity, and angle

beam_energy = self.beam_energy.get()

source_distance = self.source_distance.get()


# Add annotations to the plot (with black text color)

ax.text(

    0.05, 0.95,

    f"Beam Energy: {beam_energy:.1f} keV",

    transform=ax.transAxes,

    fontsize=12,

    color="black",

    verticalalignment="top"
    )

    ax.text(

        0.05, 0.90,

        f"Source Distance: {source_distance:.1f} cm",

        transform=ax.transAxes,

        fontsize=12,

        color="black",

        verticalalignment="top"

    )


    # Embed the figure into the Tkinter window

    canvas = FigureCanvasTkAgg(fig, master=new_window)

    canvas.get_tk_widget().pack()

    canvas.draw()




# Run the application

root = Tk()

app = XRaySimulatorApp(root)
```

```
root.mainloop()
```

Output



eam Energy=56.9 keV, Source Distance=161.6



Explanation

Phantom Generation (generate_leg_phantom):

- This function creates a 3D array representing a leg phantom, with two regions: one for the bone and another for soft tissue. Bone has a higher attenuation value than soft tissue.

X-ray Simulation (simulate_xray_image):

- This function calculates the X-ray attenuation based on beam energy and source distance. It uses different attenuation coefficients for bone and soft tissue.
- It loops through the phantom depth to sum the attenuation values at each pixel, which determines the intensity of the X-ray at each position in the final image.

Image Visualization (visualize_reconstructed_image):

- This function uses Matplotlib to display the reconstructed X-ray image in grayscale. The intensity levels correspond to the attenuation values calculated in the simulation.

GUI Setup (XRaySimulatorApp class):

- This class sets up a GUI using tkinter, with sliders for beam energy and source distance.
- A button labeled "Simulate X-Ray Image" initiates the simulation and displays the reconstructed X-ray image.
- The update_simulation method prints the current parameter values for debugging purposes.

Running the Application:

- The root.mainloop() command starts the tkinter GUI. You can adjust the parameters using the sliders and click the "Simulate X-Ray Image" button to see the impact of changes.

KEY POINTS

Beam Energy: Adjusts the attenuation values, simulating how different energies impact the X-ray image.

Source Distance: Modifies the intensity calculation to simulate the effect of the source's distance on the image clarity.

Visualization: Displays the reconstructed image based on the input parameters

Validation of Algorithms (testing.py):

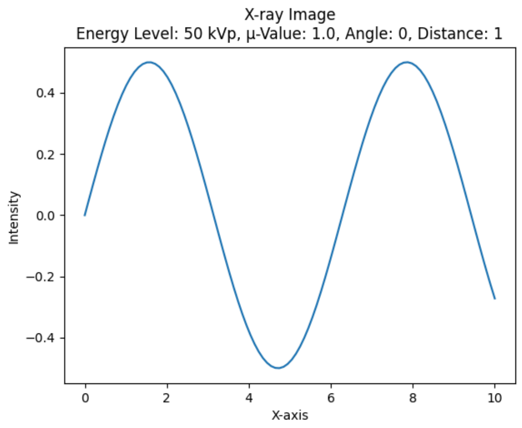
Code to perform validations on different acquisition parameters and leg fractures.

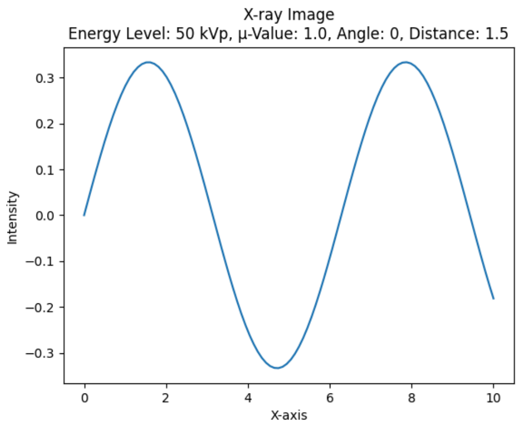
```
import numpy as np

import matplotlib.pyplot as plt



def generate_xray_image(energy_level, mu_value, angle=0, distance=1):

    x = np.linspace(0, 10, 100)

    y = np.sin(x + np.radians(angle)) * energy_level / 100 * mu_value / distance

    plt.plot(x, y)

    plt.title(f'X-ray Image\nEnergy Level: {energy_level} kVp, µ-Value:
{mu_value}, Angle: {angle}, Distance: {distance}')

    plt.xlabel('X-axis')

    plt.ylabel('Intensity')

    plt.show()
```

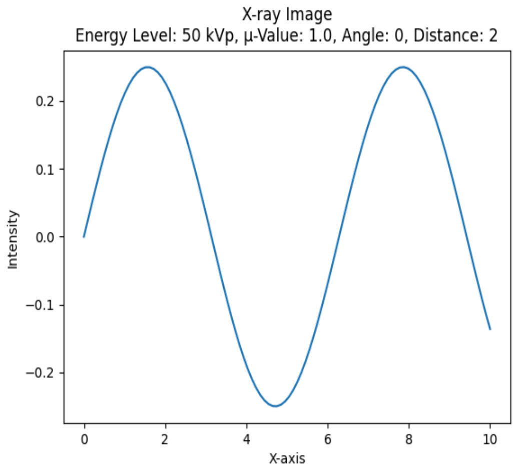
```python
def simulate_leg_fracture(angle):

    x = np.linspace(0, 10, 100)

    y = np.piecewise(x, [x < 5, x >= 5], [lambda x: np.sin(x), lambda x: np.sin(x + 
np.radians(angle))])

    plt.plot(x, y)

    plt.title(f'Leg Phantom with Fracture at Angle: {angle}°')

    plt.xlabel('X-axis')

    plt.ylabel('Intensity')

    plt.show()



def validate_acquisition_parameters(energy_levels, angles, distances):

    for energy in energy_levels:

        for angle in angles:

            for distance in distances:

                generate_xray_image(energy, mu_value=1.0, angle=angle, 
distance=distance)



def validate_leg_fractures(angles):

    for angle in angles:

        simulate_leg_fracture(angle)
```

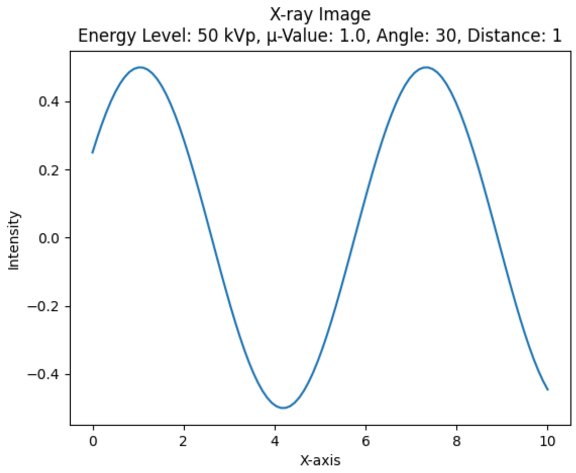
```
# Example values for validation

energy_levels = [50, 100, 150]

angles = [0, 30, 60]

distances = [1, 1.5, 2]

fracture_angles = [15, 30, 45]


# Validate acquisition parameters

print("Validating acquisition parameters...")

validate_acquisition_parameters(energy_levels, angles, distances)


# Validate leg fractures

print("Validating leg fractures...")

validate_leg_fractures(fracture_angles)
```

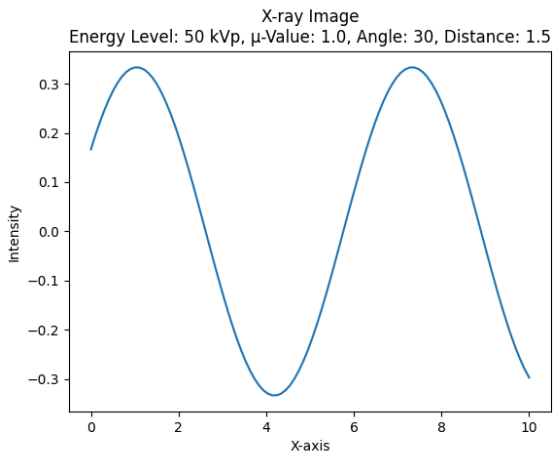
OUTPUT

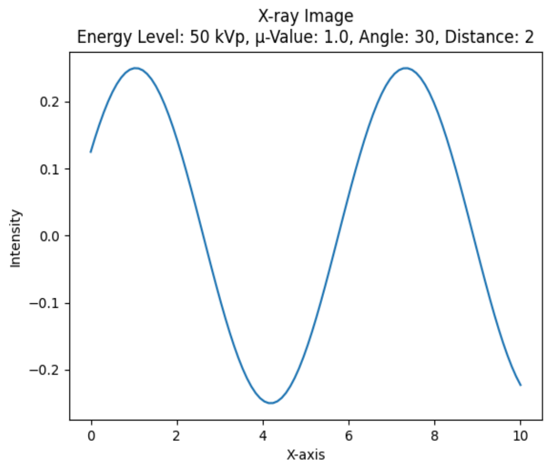

X-ray Image
Energy Level: 50 kVp, μ-Value: 1.0, Angle: 0, Distance: 1
Intensity
X-axis
0.4
0.2
0.0
−0.2
−0.4
0
2
4
6
8
10

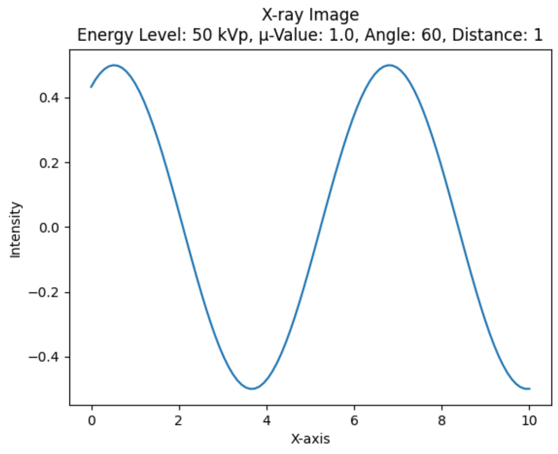

X-ray Image
Energy Level: 50 kVp, μ-Value: 1.0, Angle: 0, Distance: 1.5
Intensity
X-axis
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
2
4
6
8
10

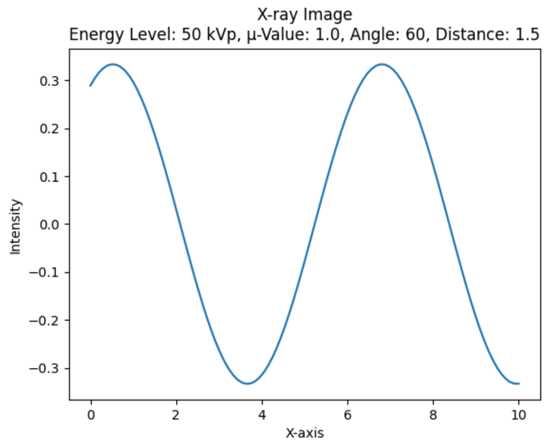

X-ray Image
Energy Level: 50 kVp, μ-Value: 1.0, Angle: 0, Distance: 2
Intensity
X-axis
0.2
0.1
0.0
−0.1
−0.2
0
2
4
6
8
10

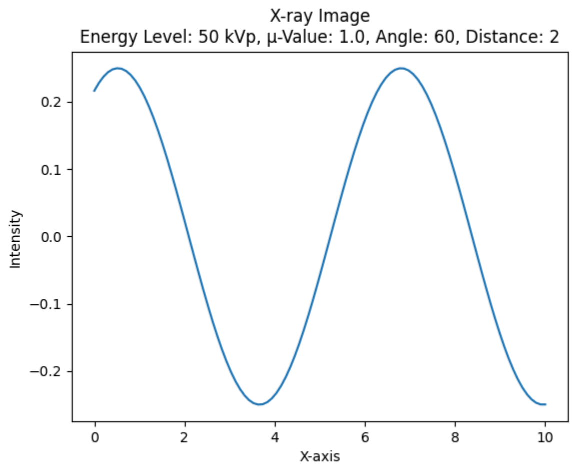

X-ray Image
Energy Level: 50 kVp, μ-Value: 1.0, Angle: 30, Distance: 1
Intensity
X-axis
0.4
0.2
0.0
−0.2
−0.4
0
2
4
6
8
10

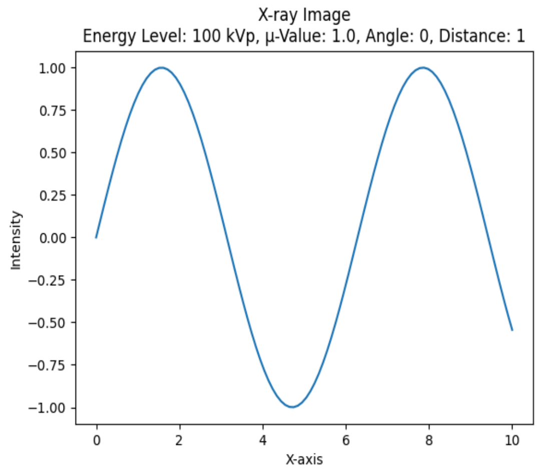

X-ray Image
Energy Level: 50 kVp, µ-Value: 1.0, Angle: 30, Distance: 1.5
Intensity
X-axis
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
2
4
6
8
10


X-ray Image
Energy Level: 50 kVp, µ-Value: 1.0, Angle: 30, Distance: 2
Intensity
X-axis
0.2
0.1
0.0
−0.1
−0.2
0
2
4
6
8
10

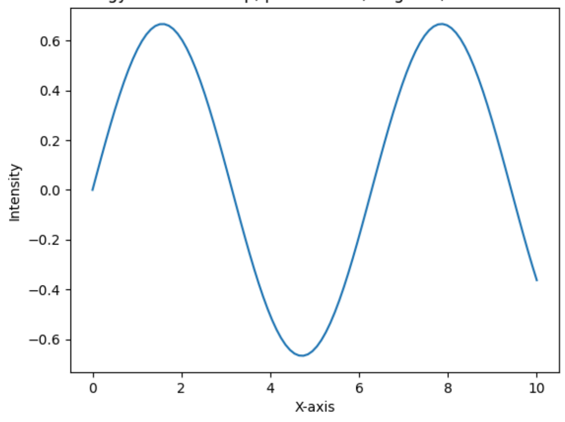

X-ray Image
Energy Level: 50 kVp, µ-Value: 1.0, Angle: 60, Distance: 1
Intensity
X-axis
0.4
0.2
0.0
−0.2
−0.4
0
2
4
6
8
10


X-ray Image
Energy Level: 50 kVp, µ-Value: 1.0, Angle: 60, Distance: 1.5
Intensity
X-axis
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
2
4
6
8
10

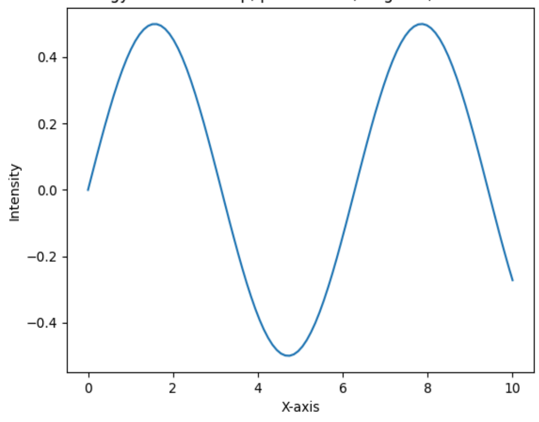

X-ray Image
Energy Level: 50 kVp, µ-Value: 1.0, Angle: 60, Distance: 2
Intensity
X-axis
0.2
0.1
0.0
−0.1
−0.2
0
2
4
6
8
10


X-ray Image
Energy Level: 100 kVp, µ-Value: 1.0, Angle: 0, Distance: 1
Intensity
X-axis
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2
4
6
8
10

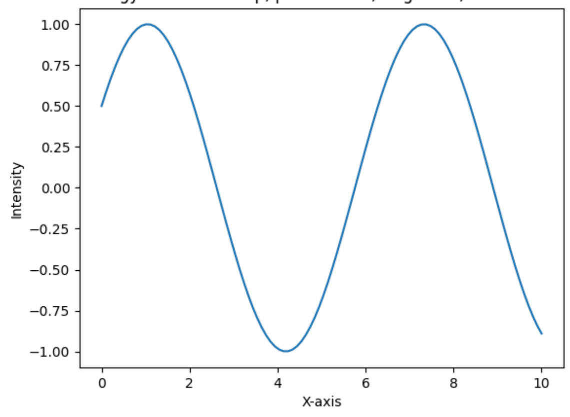
X-ray Image
Energy Level: 100 kVp, μ-Value: 1.0, Angle: 0, Distance: 1.5



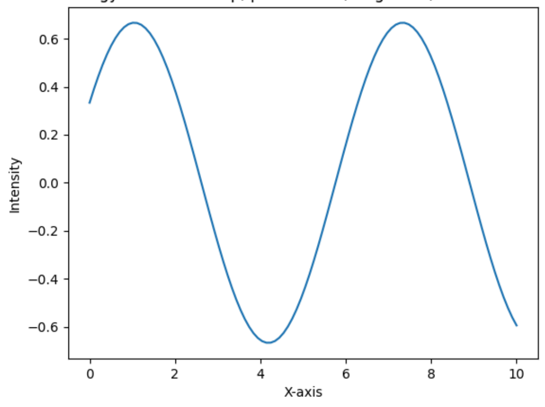
X-ray Image
Energy Level: 100 kVp, μ-Value: 1.0, Angle: 0, Distance: 2



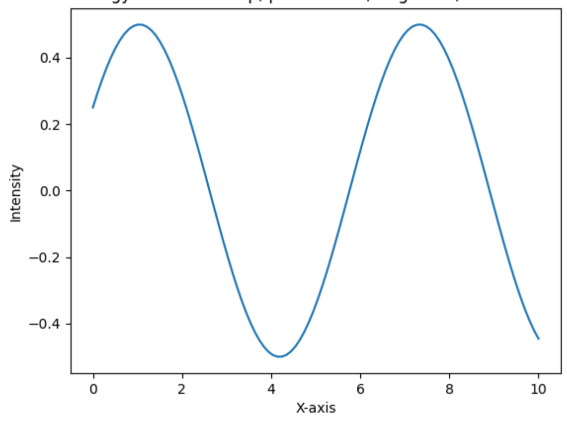
X-ray Image
Energy Level: 100 kVp, μ-Value: 1.0, Angle: 30, Distance: 1



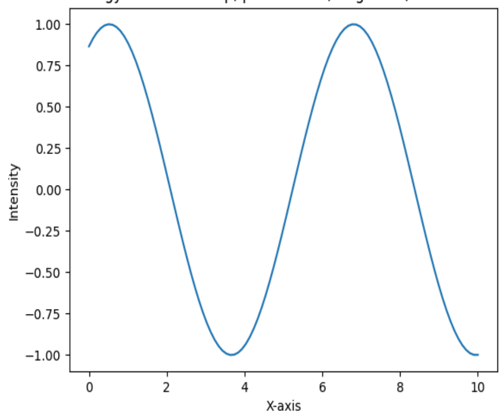
X-ray Image
Energy Level: 100 kVp, μ-Value: 1.0, Angle: 30, Distance: 1.5

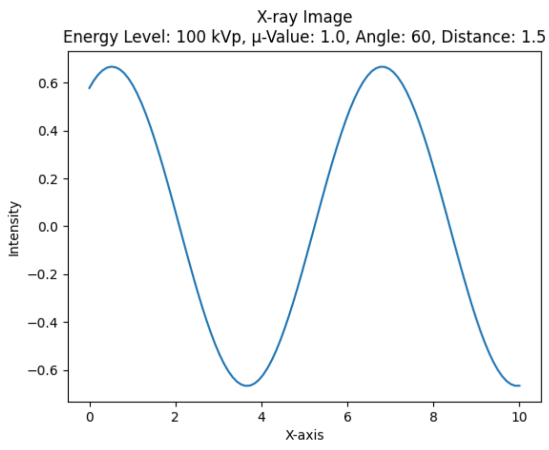
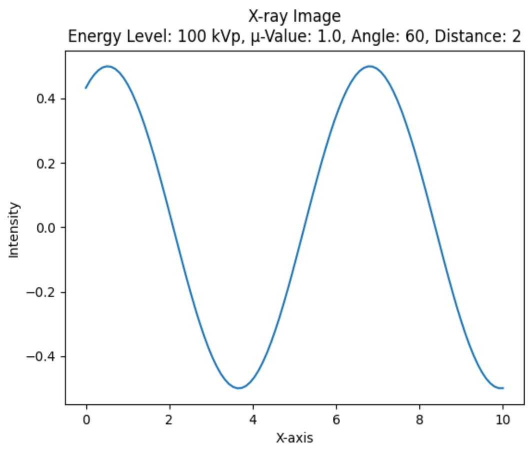
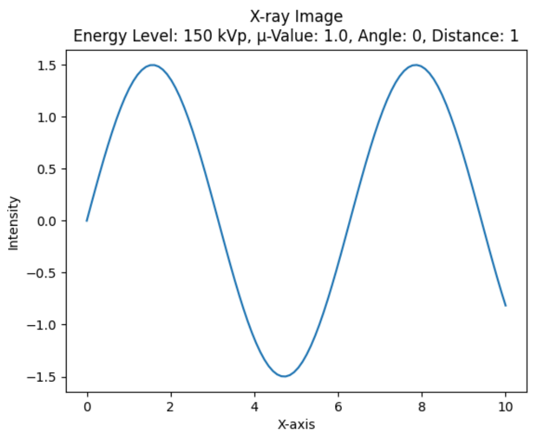
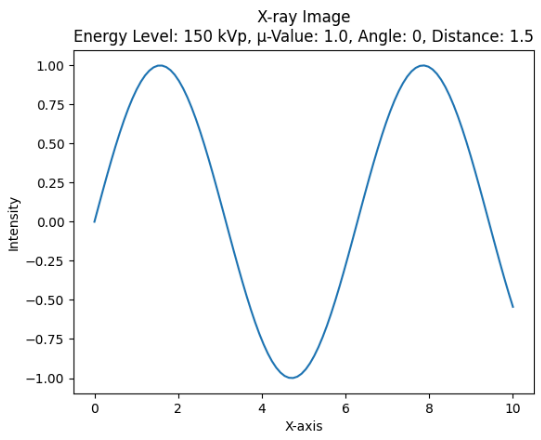
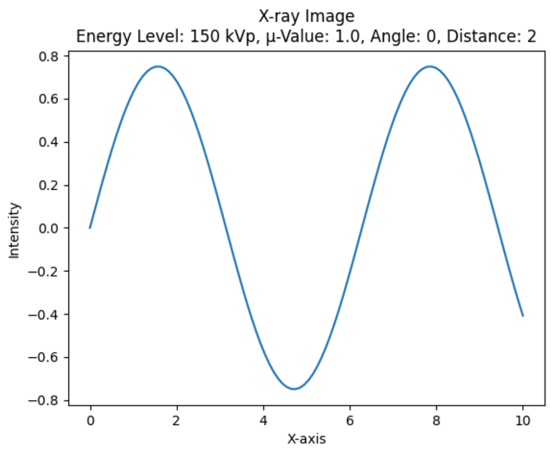
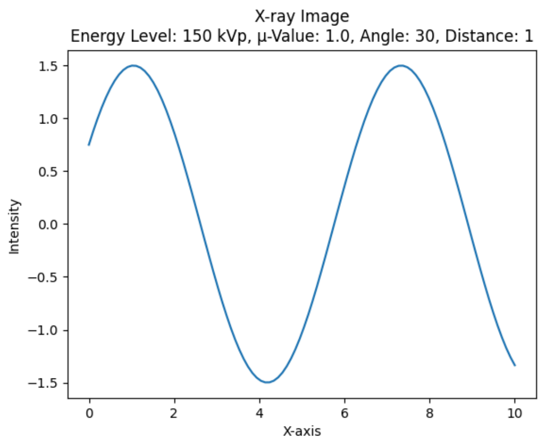


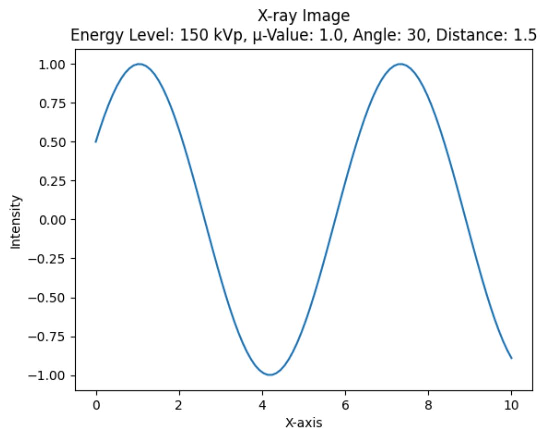
X-ray Image
Energy Level: 100 kVp, μ-Value: 1.0, Angle: 30, Distance: 2

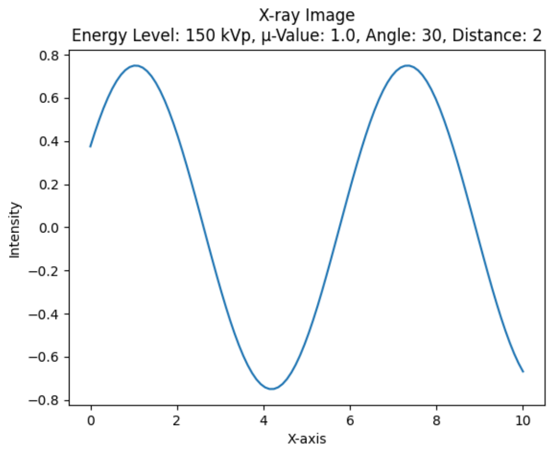


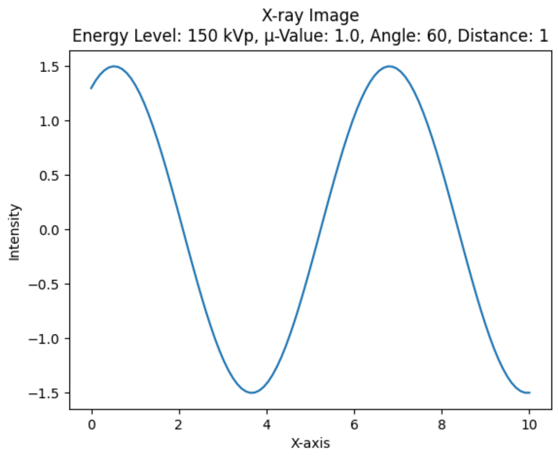
X-ray Image
Energy Level: 100 kVp, μ-Value: 1.0, Angle: 60, Distance: 1

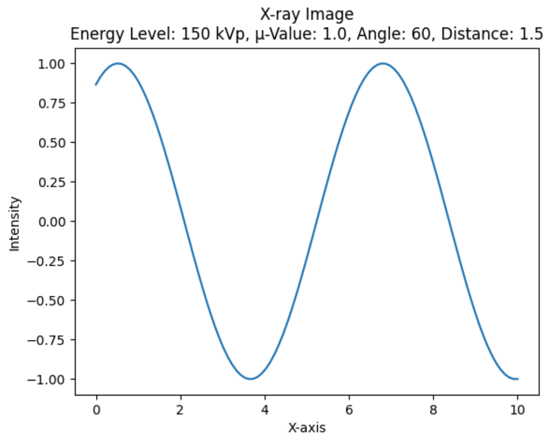


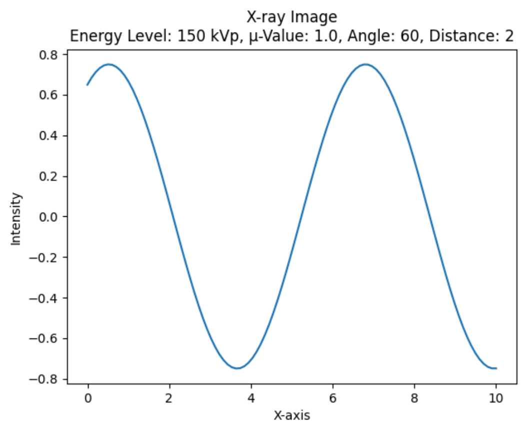

X-ray Image
Energy Level: 100 kVp, µ-Value: 1.0, Angle: 60, Distance: 1.5
Intensity
X-axis

X-ray Image
Energy Level: 100 kVp, µ-Value: 1.0, Angle: 60, Distance: 2
Intensity
X-axis

X-ray Image
Energy Level: 150 kVp, µ-Value: 1.0, Angle: 0, Distance: 1
Intensity
X-axis

X-ray Image
Energy Level: 150 kVp, µ-Value: 1.0, Angle: 0, Distance: 1.5
Intensity
X-axis

X-ray Image
Energy Level: 150 kVp, µ-Value: 1.0, Angle: 0, Distance: 2
Intensity
X-axis

X-ray Image
Energy Level: 150 kVp, µ-Value: 1.0, Angle: 30, Distance: 1
Intensity
X-axis

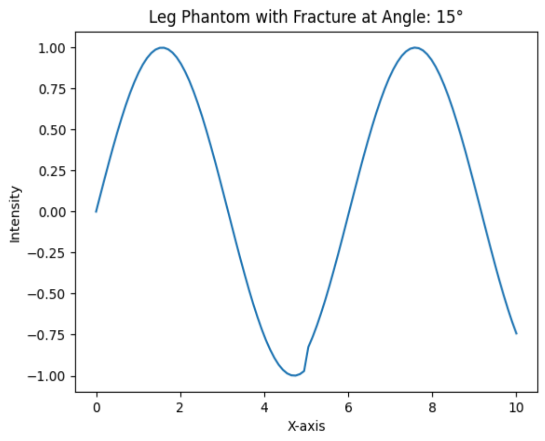

X-ray Image
Energy Level: 150 kVp, μ-Value: 1.0, Angle: 30, Distance: 1.5
Intensity
X-axis

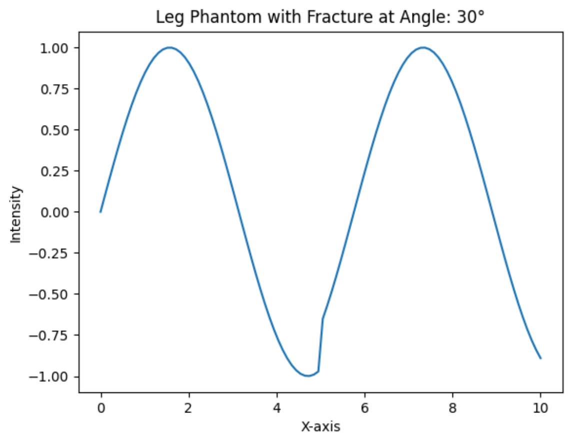

X-ray Image
Energy Level: 150 kVp, μ-Value: 1.0, Angle: 30, Distance: 2
Intensity
X-axis

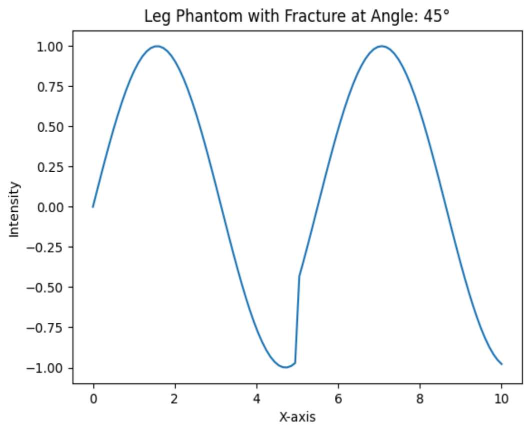

X-ray Image
Energy Level: 150 kVp, μ-Value: 1.0, Angle: 60, Distance: 1
Intensity
X-axis


X-ray Image
Energy Level: 150 kVp, μ-Value: 1.0, Angle: 60, Distance: 1.5
Intensity
X-axis









Interactive Testing

Code:

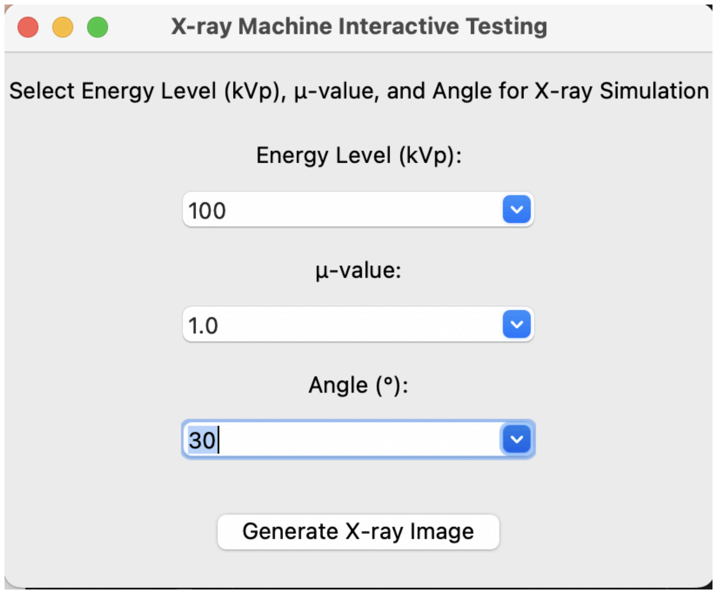
```
import tkinter as tk

from tkinter import ttk

import numpy as np

import matplotlib.pyplot as plt
```

```
# Function to simulate X-ray image
def generate_xray_image(energy_level, mu_value, angle):
    x = np.linspace(0, 10, 100)
    y = np.sin(x + np.radians(angle)) * energy_level / 100 * mu_value
    plt.figure(figsize=(8, 6))
    plt.plot(x, y, label=f'Energy: {energy_level} kVp, μ-value: {mu_value}, Angle: {angle}°')
    plt.title(f'X-ray Simulation\nEnergy: {energy_level} kVp, Angle: {angle}°')
    plt.xlabel('Position')
    plt.ylabel('X-ray Intensity')
    plt.legend()
    plt.grid(True)
    plt.show()

# Create a window using Tkinter
window = tk.Tk()
window.title("X-ray Machine Interactive Testing")


# Add a label
```

```
label = tk.Label(window, text="Select Energy Level (kVp), μ-value, and Angle for X-ray Simulation")

label.pack(pady=10)



# Energy Level Dropdown

energy_label = tk.Label(window, text="Energy Level (kVp):")

energy_label.pack(pady=5)



energy_levels = [50, 75, 100, 125, 150]

energy_dropdown = ttk.Combobox(window, values=energy_levels)

energy_dropdown.set(100)  # Default value

energy_dropdown.pack(pady=5)



# Mu Value Dropdown

mu_label = tk.Label(window, text="μ-value:")

mu_label.pack(pady=5)



mu_values = [0.1, 0.5, 1.0, 1.5, 2.0]

mu_dropdown = ttk.Combobox(window, values=mu_values)

mu_dropdown.set(1.0)  # Default value
mu_dropdown.pack(pady=5)


# Angle Dropdown

angle_label = tk.Label(window, text="Angle (°):")

angle_label.pack(pady=5)


angles = [0, 15, 30, 45, 60]

angle_dropdown = ttk.Combobox(window, values=angles)

angle_dropdown.set(0)  # Default value

angle_dropdown.pack(pady=5)


# Function to update plot based on selected values

def on_button_click():

    energy_level = int(energy_dropdown.get())

    mu_value = float(mu_dropdown.get())

    angle = int(angle_dropdown.get())

    generate_xray_image(energy_level, mu_value, angle)


# Add a button to generate the X-ray image
```

```
button = tk.Button(window, text="Generate X-ray Image",
command=on_button_click)

button.pack(pady=20)


# Run the Tkinter event loop

window.mainloop()
```

OUTPUT



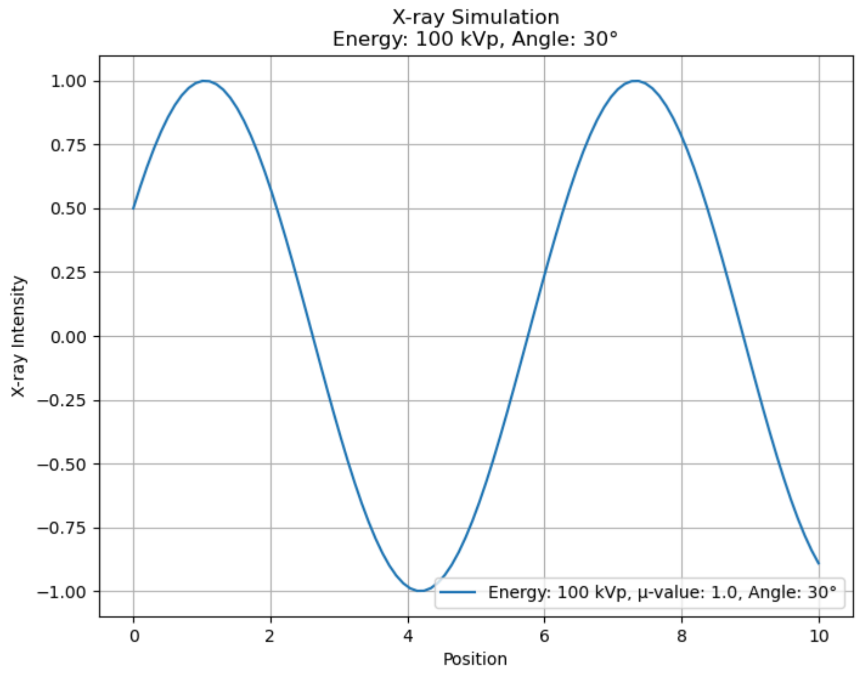
X-ray Machine Interactive Testing

Select Energy Level (kVp), µ-value, and Angle for X-ray Simulation

Energy Level (kVp): 100

µ-value: 1.0

Angle (°): 30



For the interactive testing component of the virtual X-ray machine simulation, a graphical user interface (GUI) was created using Tkinter. The interface allows users to select various parameters, such as energy levels (kVp), μ-values, and angles, via dropdown menus. Based on the selected values, the system generates and displays an X-ray simulation plot, enabling real-time testing and validation of how different parameters affect the X-ray image. This interactive tool facilitates easy manipulation and visualization, aiding in the assessment of the X-ray machine's performance under different configurations.

Contrast And Angle Analysis

```
import numpy as np

import matplotlib.pyplot as plt

from scipy.ndimage import rotate



# Define the function to generate the phantom
```

```
def generate_leg_phantom(leg_radius=50, bone_radius=20, height=100):
    phantom = np.zeros((height, 2 * leg_radius, 2 * leg_radius))
    for z in range(height):
        for x in range(2 * leg_radius):
            for y in range(2 * leg_radius):
                distance = np.sqrt((x - leg_radius) ** 2 + (y - leg_radius) ** 2)
                if distance <= bone_radius:
                    phantom[z, x, y] = 2  # Bone region
                elif distance <= leg_radius:
                    phantom[z, x, y] = 1  # Soft tissue region
    return phantom


# Generate the phantom
phantom = generate_leg_phantom()


# Add the functions for contrast and angle analysis (as previously provided)
def calculate_contrast(phantom_slice):
    bone_pixels = phantom_slice[phantom_slice == 2]
    soft_tissue_pixels = phantom_slice[phantom_slice == 1]
    if bone_pixels.size > 0 and soft_tissue_pixels.size > 0:

        contrast = np.abs(np.mean(bone_pixels) - np.mean(soft_tissue_pixels))

    else:

        contrast = 0

    return contrast



def generate_angle_projection(phantom, angle):

    rotated_phantom = rotate(phantom, angle, axes=(1, 2), reshape=False,
mode='constant', cval=0)

    projection = np.sum(rotated_phantom, axis=0)

    return projection



def analyze_contrast_and_angle(phantom, slice_index, angles):

    phantom_slice = phantom[slice_index]

    contrast = calculate_contrast(phantom_slice)

    print(f"Contrast for slice {slice_index}: {contrast}")

    for angle in angles:

        projection = generate_angle_projection(phantom, angle)

        plt.figure(figsize=(8, 8))

        plt.imshow(projection, cmap="gray")
    plt.title(f"X-Ray Projection at {angle}°")

    plt.xlabel("X-axis")

    plt.ylabel("Y-axis")

    plt.colorbar(label="Intensity Sum")

    plt.show()


# Example usage

slice_index = phantom.shape[0] // 2

angles = [0, 45, 90, 135]

analyze_contrast_and_angle(phantom, slice_index, angles)
```

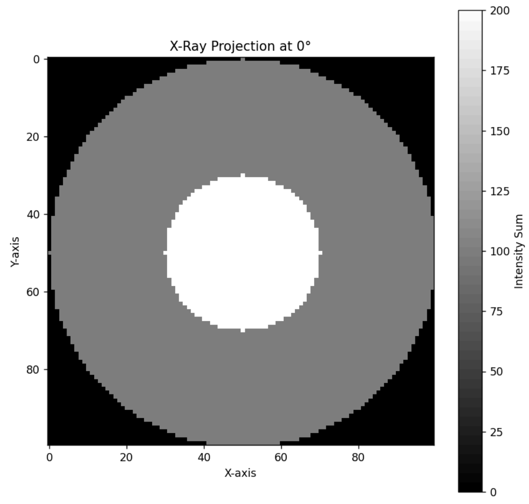
Explanation:

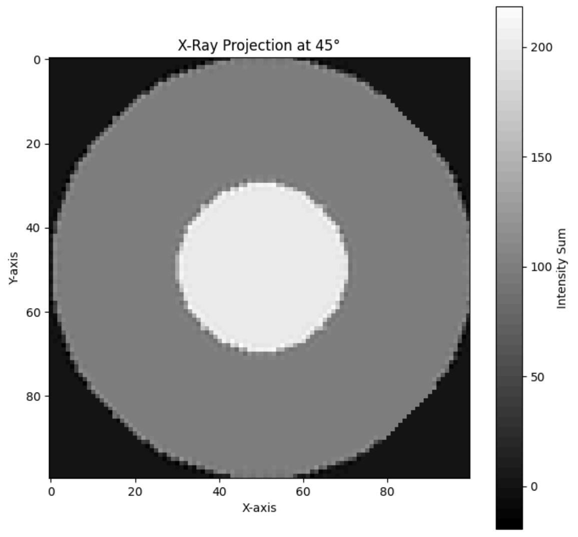
For Calculating Contrast:
Determines the average intensity difference in a given phantom slice between soft tissue (value 1) and bone (value 2). It gives an indication of the degree of differentiation between soft tissue and bone.

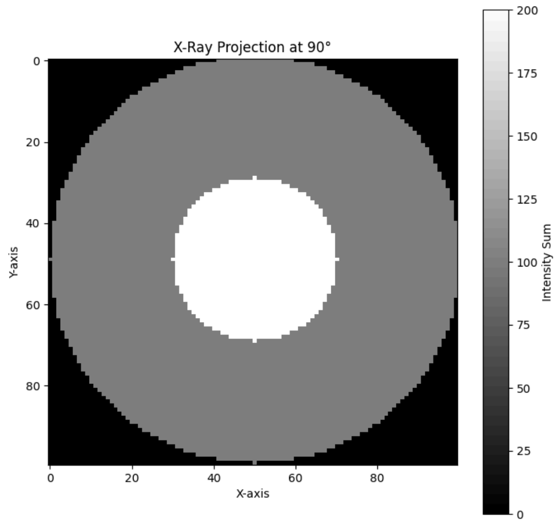
For Projection Based on Angle:
rotates the phantom and adds up intensities along the height axis to simulate X-ray projections. It creates two-dimensional pictures of the phantom at predetermined angles, such as 0°, 45°, 90°, and 135°.

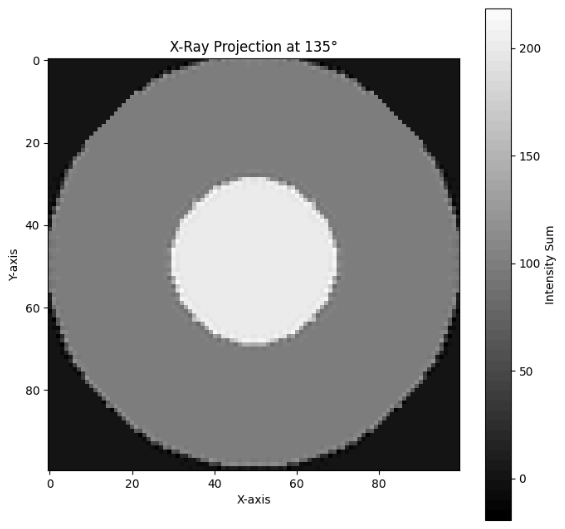
For Visualization:
It displays the center slice's contrast value. It displays X-ray projections in grayscale with an intensity color bar.

Output









Explanation:

Contrast:
Because of their different attenuation levels, there is a strong contrast between soft tissue and bone.

Projected:
0° Projection: Bone and soft tissue are represented by transparent concentric circles.
135° and 45° Projections: Slightly rotated but with similar concentric patterns.
90° Projection: Symmetric view because of the phantom's cylindrical shape.

Observations:
Thick tissue sections result in higher intensity readings.
The phantom's circular shape ensures symmetry across projections.