```python
# -*- coding: utf-8 -*-
"""
Created on Tue Apr  2 16:44:23 2024

@author: Niloy Roy
"""
from openslide import open_slide
import glob
import os
import tifffile
from pystackreg import StackReg
from skimage.transform import resize
#from skimage.color import rgba2rgb, rgb2gray
#from skimage.io import imsave
import numpy as np
from PIL import Image


def save_image(image, filename, output_folder, file_format):
    # Convert the image to a PIL Image object
    image_pil = Image.fromarray(image).convert("L")

    # Specify the file path to save the image
    file_path = os.path.join(output_folder, f"{filename}.{file_format}")

    # Save the image
    image_pil.save(file_path)

    print(f"{file_format.upper()} image saved successfully at: {file_path}")

def rgba_to_gray(rgba_image):
    # Extract the RGB channels
    rgb_image = rgba_image[:, :, :3]
    # Convert to grayscale while considering alpha channel
    gray_image = np.dot(rgb_image[..., :3], [0.2989, 0.5870, 0.1140]) * rgba_image[..., 3] /
255.0
    return gray_image


def process_slide(slide_path):
    # Load the slide file into an object.
    slide = open_slide(slide_path)



   # Get slide dims at each level.
    dims = slide.level_dimensions
    #num_levels = len(dims)


 # Get the index of the highest level
    #highest_level_index = num_levels - 1

    # Get dimensions of the highest level
    #highest_level_dim = dims[highest_level_index]


    level_img = slide.read_region((0, 0), 5, dims[5])  # Pillow object, mode=RGBA



    # Convert the image into a numpy array for processing
    level_img_np = np.array(level_img, dtype='uint8')

    level_img_np_grayscale = rgba_to_gray(level_img_np)

    return level_img_np_grayscale
```

```
# Function to process and resize images
# order 4 = Bi-quartic interpolation
def process_and_resize_image(filename, target_height, target_width):
    img = process_slide(filename)
    # Assuming process_slide returns a numpy array
    img_resized = resize(img, (target_height, target_width), order=4, anti_aliasing=True,
preserve_range=True).astype(np.uint8)

    return img_resized

# Load the NDPI files
# Set the desired input folder path
input_folder = 'F:/Germany_2022/TU Illmenau/hiwi/DataSets/histology/NDp'

# Get the list of NDPI files in the input folder
ndpi_files = glob.glob(os.path.join(input_folder, '*.ndpi'))

if len(ndpi_files) % 2 == 0:
    middle_index = len(ndpi_files) // 2
else:
    middle_index = (len(ndpi_files) - 1) // 2

# Initialize StackReg
sr = StackReg(StackReg.RIGID_BODY)

# Process and resize images starting from the middle index
reference_image_path = ndpi_files[middle_index]
reference_image = process_slide(reference_image_path)
reference_height, reference_width = reference_image.shape[:2]

# Set the desired folder pathpair_by_pair_Previous
output_folder = 'F:/Germany_2022/TU
Illmenau/hiwi/DataSets/histology/Result/Middle_To_End/pairbypair_withPrevious'
output_folder_1 = 'F:/Germany_2022/TU
Illmenau/hiwi/DataSets/histology/Result/Middle_To_Start/pair_by_pair_Previous'

# Create the output folder if it doesn't exist
os.makedirs(output_folder, exist_ok=True)
os.makedirs(output_folder_1, exist_ok=True)


# Save the reference image
tifffile.imwrite(os.path.join(output_folder_1, f'reference_{middle_index}.tif'),
reference_image.astype(np.uint8))
save_image(reference_image, f'reference_{middle_index}', output_folder_1, 'png')


for i in range(middle_index - 1, -1, -1):
    # Resize the offset image to match the reference image dimensions
    offset_image_resized = process_and_resize_image(ndpi_files[i], reference_height,
reference_width)

    # Co-register the offset image to the reference image
    registered_image = sr.register_transform(reference_image, offset_image_resized)

    # Save the reference and registered image pair
    #registered_image_pil = Image.fromarray(registered_image).convert("L")
    #registered_image_pil.save(output_folder)


    tifffile.imwrite(os.path.join(output_folder_1, f'registered_{i}.tif'),
registered_image.astype(np.uint8))
    save_image(registered_image, f'registered_{i}', output_folder_1, 'png')

    # Update the reference image and its dimensions for the next iteration
    reference_image = registered_image
```

```
    reference_height, reference_width = reference_image.shape[:2]


for i in range(middle_index + 1, len(ndpi_files)):
    # Resize the offset image to match the reference image dimensions
    offset_image_resized = process_and_resize_image(ndpi_files[i], reference_height,
reference_width)

    # Co-register the offset image to the reference image
    registered_image = sr.register_transform(reference_image, offset_image_resized)

    # Save the reference and registered image pair
    #registered_image_pil = Image.fromarray(registered_image).convert("L")
    #registered_image_pil.save(output_folder)


    tifffile.imwrite(os.path.join(output_folder, f'registered_{i}.tif'),
registered_image.astype(np.uint8))
    #save_image(registered_image, f'registered_{i}', output_folder, 'png')

    # Update the reference image and its dimensions for the next iteration
    reference_image = registered_image
    reference_height, reference_width = reference_image.shape[:2]
```

```python
# -*- coding: utf-8 -*-
"""
Created on Wed Feb 28 22:55:40 2024

@author: Dell
"""

from pystackreg import StackReg
from skimage import io
from matplotlib import pyplot as plt




#load reference and "moved" image
ref_img = io.imread('F:/Germany_2022/TU Illmenau/hiwi/DataSets/histology/NDp/FLM-005_J-13-2235_1564_HE.ndpi')
offset_img = io.imread('F:/Germany_2022/TU Illmenau/hiwi/DataSets/histology/NDp/FLM-005_J-13-2235_1568_HE.ndpi')




#Rigid Body transformation
sr = StackReg(StackReg.RIGID_BODY)
out_rot = sr.register_transform(ref_img, offset_img)



fig = plt.figure(figsize=(10, 10))
ax1 = fig.add_subplot(2,2,1)
ax1.imshow(ref_img, cmap='gray')
ax1.title.set_text('Input Image')

ax3 = fig.add_subplot(2,2,3)
ax3.imshow(out_rot, cmap='gray')
ax3.title.set_text('Rigid Body')

plt.show()
```

```
# -*- coding: utf-8 -*-
"""
Created on Wed Apr  3 19:19:13 2024

@author: Dell
"""

# -*- coding: utf-8 -*-
"""
Created on Tue Apr  2 16:44:23 2024

@author: Niloy Roy
"""
from openslide import open_slide
import glob
import os
import tifffile
from pystackreg import StackReg
from skimage.transform import resize
#from skimage.color import rgba2rgb, rgb2gray
#from skimage.io import imsave
import numpy as np
from PIL import Image


def save_image(image, filename, output_folder, file_format):
    # Convert the image to a PIL Image object
    image_pil = Image.fromarray(image).convert("L")

    # Specify the file path to save the image
    file_path = os.path.join(output_folder, f"{filename}.{file_format}")

    # Save the image
    image_pil.save(file_path)

    print(f"{file_format.upper()} image saved successfully at: {file_path}")

def rgba_to_gray(rgba_image):
    # Extract the RGB channels
    rgb_image = rgba_image[:, :, :3]
    # Convert to grayscale while considering alpha channel
    gray_image = np.dot(rgb_image[..., :3], [0.2989, 0.5870, 0.1140]) * rgba_image[..., 3] /
255.0
    return gray_image


def process_slide(slide_path):
    # Load the slide file into an object.
    slide = open_slide(slide_path)



   # Get slide dims at each level.
    dims = slide.level_dimensions
    #num_levels = len(dims)


 # Get the index of the highest level
    #highest_level_index = num_levels - 1

    # Get dimensions of the highest level
    #highest_level_dim = dims[highest_level_index]


    level_img = slide.read_region((0, 0), 5, dims[5])  # Pillow object, mode=RGBA
```

```
    # Convert the image into a numpy array for processing
    level_img_np = np.array(level_img, dtype='uint8')

    level_img_np_grayscale = rgba_to_gray(level_img_np)

    return level_img_np_grayscale

# Function to process and resize images
# order 4 = Bi-quartic interpolation
def process_and_resize_image(filename, target_height, target_width):
    img = process_slide(filename)
    # Assuming process_slide returns a numpy array
    img_resized = resize(img, (target_height, target_width), order=4, anti_aliasing=True,
preserve_range=True).astype(np.uint8)

    return img_resized

# Load the NDPI files
# Set the desired input folder path
input_folder = 'F:/Germany_2022/TU Illmenau/hiwi/DataSets/histology/NDp'

# Get the list of NDPI files in the input folder
ndpi_files = glob.glob(os.path.join(input_folder, '*.ndpi'))

if len(ndpi_files) % 2 == 0:
    middle_index = len(ndpi_files) // 2
else:
    middle_index = (len(ndpi_files) - 1) // 2

# Initialize StackReg
sr = StackReg(StackReg.RIGID_BODY)

# Process and resize images starting from the middle index
reference_image_path = ndpi_files[middle_index]
reference_image = process_slide(reference_image_path)
reference_height, reference_width = reference_image.shape[:2]

# Set the desired folder path

output_folder = 'F:/Germany_2022/TU
Illmenau/hiwi/DataSets/histology/Result/Middle_To_End/oneReference'
output_folder_1= 'F:/Germany_2022/TU
Illmenau/hiwi/DataSets/histology/Result/Middle_To_Start/oneReference'

# Create the output folder if it doesn't exist
os.makedirs(output_folder, exist_ok=True)
os.makedirs(output_folder_1, exist_ok=True)

# Save the reference image
tifffile.imwrite(os.path.join(output_folder_1, f'reference_{middle_index}.tif'),
reference_image.astype(np.uint8))
save_image(reference_image, f'reference_{middle_index}', output_folder_1, 'png')

for i in range(middle_index - 1, -1, -1):
    # Resize the offset image to match the reference image dimensions
    offset_image_resized = process_and_resize_image(ndpi_files[i], reference_height,
reference_width)

    # Co-register the offset image to the reference image
    registered_image = sr.register_transform(reference_image, offset_image_resized)

    # Save the reference and registered image pair
    #registered_image_pil = Image.fromarray(registered_image).convert("L")
    #registered_image_pil.save(output_folder)
```

```
    tifffile.imwrite(os.path.join(output_folder_1, f'registered_{i}.tif'),
registered_image.astype(np.uint8))
    save_image(registered_image, f'registered_{i}', output_folder_1, 'png')


for i in range(middle_index + 1, len(ndpi_files)):
    # Resize the offset image to match the reference image dimensions
    offset_image_resized = process_and_resize_image(ndpi_files[i], reference_height,
reference_width)

    # Co-register the offset image to the reference image
    registered_image = sr.register_transform(reference_image, offset_image_resized)

    # Save the reference and registered image pair
    #registered_image_pil = Image.fromarray(registered_image).convert("L")
    #registered_image_pil.save(output_folder)


    tifffile.imwrite(os.path.join(output_folder, f'registered_{i}.tif'),
registered_image.astype(np.uint8))
    save_image(registered_image, f'registered_{i}', output_folder, 'png')
```

```python
# -*- coding: utf-8 -*-
"""
Created on Mon Apr  8 16:22:24 2024

@author: Dell
"""

import os
import glob
from pystackreg import StackReg
from skimage.io import imread
import numpy as np
from PIL import Image

def save_registered_images(registered_stack, output_folder, file_format):
    for i, img in enumerate(registered_stack):
        filename = f'registered_{i}.{file_format}'
        file_path = os.path.join(output_folder, filename)

        # Save as PNG
        Image.fromarray(img).save(file_path)
        print(f"PNG image saved successfully at: {file_path}")

        # Save as TIFF
        np.save(file_path[:-3] + 'tif', img)
        print(f"TIFF image saved successfully at: {file_path[:-3] + 'tif'}")

input_folder = 'F:/Germany_2022/TU Illmenau/hiwi/DataSets/histology/NDp'
ndpi_files = sorted(glob.glob(os.path.join(input_folder, '*.ndpi')))
output_folder_start = 'F:/Germany_2022/TU
Illmenau/hiwi/DataSets/histology/NDp_coregistered_middle_to_start/level5'
output_folder_end = 'F:/Germany_2022/TU
Illmenau/hiwi/DataSets/histology/NDp_coregistered_middle_to_end/level5'
os.makedirs(output_folder_start, exist_ok=True)
os.makedirs(output_folder_end, exist_ok=True)

# Determine the middle index
middle_index = len(ndpi_files) // 2

# Load the images into two separate stacks: one for middle to start, one for middle to end
img_stack_start = [imread(f, plugin='tifffile') for f in ndpi_files[:middle_index+1][::-1]]  #
Reverse to start from middle to first
img_stack_end = [imread(f, plugin='tifffile') for f in ndpi_files[middle_index:]]

# Convert lists to numpy arrays
img_stack_start = np.array(img_stack_start)
img_stack_end = np.array(img_stack_end)

# Initialize StackReg
sr = StackReg(StackReg.RIGID_BODY)

# Register the stacks
registered_start = sr.register_transform_stack(img_stack_start, reference='previous')[::-1]
registered_end = sr.register_transform_stack(img_stack_end, reference='previous')

# Save the registered images
save_registered_images(registered_start, output_folder_start, 'png')
save_registered_images(registered_start, output_folder_start, 'tif')
save_registered_images(registered_end, output_folder_end, 'png')
save_registered_images(registered_end, output_folder_end, 'tif')
```

```
"""
Created on Wed Feb 28 23:46:20 2024

@author: Niloy Roy
"""
from pystackreg import StackReg
from openslide import open_slide
import openslide
import numpy as np
from matplotlib import pyplot as plt
#from skimage.color import rgb2gray
from skimage.transform import resize

def process_slide(slide_path, level):
    # Load the slide file into an object.
    slide = open_slide(slide_path)

    # Get slide properties
    slide_props = slide.properties
    print("Properties:", slide_props)

    print("Vendor is:", slide_props['openslide.vendor'])
    print("Pixel size of X in um is:", slide_props['openslide.mpp-x'])
    print("Pixel size of Y in um is:", slide_props['openslide.mpp-y'])

    # Objective used to capture the image
    objective = float(slide.properties[openslide.PROPERTY_NAME_OBJECTIVE_POWER])
    print("The objective power is: ", objective)

    # Get slide dimensions for the level 0 - max resolution level
    slide_dims = slide.dimensions
    print("Slide dimensions:", slide_dims)

    # Get a thumbnail of the image and visualize
    slide_thumb_600 = slide.get_thumbnail(size=(600, 600))
    slide_thumb_600.show()

    # Convert thumbnail to numpy array
    slide_thumb_600_np = np.array(slide_thumb_600)
    plt.figure(figsize=(8, 8))
    plt.imshow(slide_thumb_600_np)

    # Get slide dims at each level.
    dims = slide.level_dimensions
    num_levels = len(dims)
    print("Number of levels in this image are:", num_levels)
    print("Dimensions of various levels in this image are:", dims)

    # By how much are levels downsampled from the original image?
    factors = slide.level_downsamples
    print("Each level is downsampled by an amount of: ", factors)

    # Copy an image from a level
    level_dim = dims[level]
    level_img = slide.read_region((0, 0), level, level_dim)  # Pillow object, mode=RGBA

    # Convert the image to RGB
    level_img_RGB = level_img.convert('RGB')
    level_img_RGB.show()

    # Convert the image into a numpy array for processing
    level_img_np = np.array(level_img, dtype='uint8')


    return level_img_np

def rgba_to_gray(rgba_image):
```

```
    # Extract the RGB channels
    rgb_image = rgba_image[:, :, :3]
    # Convert to grayscale while considering alpha channel
    gray_image = np.dot(rgb_image[..., :3], [0.2989, 0.5870, 0.1140]) * rgba_image[..., 3] /
255.0
    return gray_image

# Example usage:
slide_path = "F:/Germany_2022/TU Illmenau/hiwi/DataSets/NDp/FLM-005_J-13-2235_1564_HE.ndpi"
ref_img = process_slide(slide_path, level=9)

slide_path2 = "F:/Germany_2022/TU Illmenau/hiwi/DataSets/NDp/FLM-005_J-13-2235_1568_HE.ndpi"
offset_img = process_slide(slide_path2, level=8)



ref_gray = rgba_to_gray(ref_img)
offset_gray = rgba_to_gray(offset_img)
# Resize images to have the same dimensions
max_height = max(ref_gray.shape[0], offset_gray.shape[0])
max_width = max(ref_gray.shape[1], offset_gray.shape[1])
ref_resized = resize(ref_gray, (max_height, max_width))
offset_resized = resize(offset_gray, (max_height, max_width))






#Rigid Body transformation
sr = StackReg(StackReg.RIGID_BODY)
out_rot = sr.register_transform(ref_resized, offset_resized)

#Scaled Rotation transformation
#sr = StackReg(StackReg.SCALED_ROTATION)
#out_sca = sr.register_transform(ref_img, offset_img)


fig = plt.figure(figsize=(10, 10))
ax1 = fig.add_subplot(2,2,1)
ax1.imshow(ref_img, cmap='gray')
ax1.title.set_text('Input Image')

ax3 = fig.add_subplot(2,2,3)
ax3.imshow(out_rot, cmap='gray')
ax3.title.set_text('Rigid Body')

plt.show()


from PIL import Image

# Convert the reference and registered image arrays to PIL Image
ref_image_pil = Image.fromarray(ref_img).convert("L")
registered_image_pil = Image.fromarray(out_rot).convert("L")


# Specify the file paths to save the images
ref_image_path = "F:/Germany_2022/TU Illmenau/hiwi/DataSets/NDp/co-
registered/reference_image.png"
registered_image_path = "F:/Germany_2022/TU Illmenau/hiwi/DataSets/NDp/co-
registered/registered_image.png"

# Save the reference image
ref_image_pil.save(ref_image_path)
print("Reference image saved successfully at:", ref_image_path)
```

```
# Save the registered image
registered_image_pil.save(registered_image_path)
print("Registered image saved successfully at:", registered_image_path)
```

```
import os
import glob
from openslide import open_slide
from pystackreg import StackReg
import numpy as np
from PIL import Image

def save_image(image, filename, output_folder, file_format):
    image_pil = Image.fromarray(image).convert("L")
    file_path = os.path.join(output_folder, f"{filename}.{file_format}")
    image_pil.save(file_path)
    print(f"{file_format.upper()} image saved successfully at: {file_path}")



def rgba_to_gray(rgba_image):
    # Extract the RGB channels
    rgb_image = rgba_image[:, :, :3]
    # Convert to grayscale while considering alpha channel
    gray_image = np.dot(rgb_image[..., :3], [0.2989, 0.5870, 0.1140]) * rgba_image[..., 3] /
255.0
    return gray_image

def process_slide(slide_path, level):
    slide = open_slide(slide_path)
    dims = slide.level_dimensions
    level_img = slide.read_region((0, 0), level, dims[level])
    level_img_np = np.array(level_img, dtype='uint8')
    level_img_np_grayscale = rgba_to_gray(level_img_np)
    return level_img_np_grayscale

input_folder = 'F:/Germany_2022/TU Illmenau/hiwi/DataSets/histology/NDp'
ndpi_files = sorted(glob.glob(os.path.join(input_folder, '*.ndpi')))
output_folder_start = 'F:/Germany_2022/TU
Illmenau/hiwi/DataSets/histology/Stack_transform_middle_to_start/previous'
output_folder_end = 'F:/Germany_2022/TU
Illmenau/hiwi/DataSets/histology/Stack_transform_middle_to_end/previous'
output_folder_start_first = 'F:/Germany_2022/TU
Illmenau/hiwi/DataSets/histology/Stack_transform_middle_to_start/first'
output_folder_end_first = 'F:/Germany_2022/TU
Illmenau/hiwi/DataSets/histology/Stack_transform_middle_to_start/first'
os.makedirs(output_folder_start, exist_ok=True)
os.makedirs(output_folder_end, exist_ok=True)

# Determine the middle index
if len(ndpi_files) % 2 == 0:
    middle_index = len(ndpi_files) // 2
else:
    middle_index = (len(ndpi_files) - 1) // 2

# Load the images into two separate stacks: one for middle to start, one for middle to end
img_stack_start = [process_slide(f, level=5) for f in ndpi_files[:middle_index+1][::-1]]  #
Reverse to start from middle to first
img_stack_end = [process_slide(f, level=5) for f in ndpi_files[middle_index:]]

# Convert lists to numpy arrays
img_stack_start = np.array(img_stack_start)
img_stack_end = np.array(img_stack_end)

# Initialize StackReg
sr = StackReg(StackReg.RIGID_BODY)

# Register the stacks
registered_start = sr.register_transform_stack(img_stack_start, reference='previous')[::-1]
registered_end = sr.register_transform_stack(img_stack_end, reference='previous')

# Save the registered images
```

```
for i, registered_image in enumerate(registered_start):
    save_image(registered_image, f'registered_{i}', output_folder_start, 'png')
    save_image(registered_image, f'registered_{i}', output_folder_start, 'tif')

for i, registered_image in enumerate(registered_end):
    save_image(registered_image, f'registered_{i}', output_folder_end, 'png')
    save_image(registered_image, f'registered_{i}', output_folder_end, 'tif')


# Register the stacks to the first image in each stack
registered_start_first = sr.register_transform_stack(img_stack_start, reference='first')[::-1]
# Reverse back after registration
registered_end_first = sr.register_transform_stack(img_stack_end, reference='first')

# Save the registered images
for i, registered_image in enumerate(registered_start_first):
    save_image(registered_image, f'registered_{i}', output_folder_start_first, 'png')
    save_image(registered_image, f'registered_{i}', output_folder_start_first, 'tif')

for i, registered_image in enumerate(registered_end_first):
    save_image(registered_image, f'registered_{i}', output_folder_end_first, 'png')
    save_image(registered_image, f'registered_{i}', output_folder_end_first, 'tif')
```

```
import os
import glob
from openslide import open_slide
from pystackreg import StackReg
import numpy as np
from PIL import Image
from skimage.transform import resize

def save_image(image, filename, output_folder, file_format):
    image_pil = Image.fromarray(image).convert("L")
    file_path = os.path.join(output_folder, f"{filename}.{file_format}")
    image_pil.save(file_path)
    print(f"{file_format.upper()} image saved successfully at: {file_path}")

def rgba_to_gray(rgba_image):
    rgb_image = rgba_image[:, :, :3]
    gray_image = np.dot(rgb_image[..., :3], [0.2989, 0.5870, 0.1140]) * rgba_image[..., 3] /
255.0
    return gray_image

def process_slide(slide_path, level, target_size=None):
    slide = open_slide(slide_path)
    dims = slide.level_dimensions
    level_img = slide.read_region((0, 0), level, dims[level])
    level_img_np = np.array(level_img, dtype='uint8')
    level_img_np_grayscale = rgba_to_gray(level_img_np)
    if target_size is not None:
        level_img_np_grayscale_resized = resize(level_img_np_grayscale, target_size,
anti_aliasing=True)
        return level_img_np_grayscale_resized
    else:
        return level_img_np_grayscale

input_folder = 'F:/Germany_2022/TU Illmenau/hiwi/DataSets/histology/NDp'
ndpi_files = sorted(glob.glob(os.path.join(input_folder, '*.ndpi')))
output_folder_start = 'F:/Germany_2022/TU
Illmenau/hiwi/DataSets/histology/Scaled/Stack_transform_middle_to_start/previous'
output_folder_end = 'F:/Germany_2022/TU
Illmenau/hiwi/DataSets/histology/Scaled/Stack_transform_middle_to_end/previous'
output_folder_start_first = 'F:/Germany_2022/TU
Illmenau/hiwi/DataSets/histology/Scaled/Stack_transform_middle_to_start/first'
output_folder_end_first = 'F:/Germany_2022/TU
Illmenau/hiwi/DataSets/histology/Scaled/Stack_transform_middle_to_end/first'
os.makedirs(output_folder_start, exist_ok=True)
os.makedirs(output_folder_end, exist_ok=True)
os.makedirs(output_folder_start_first, exist_ok=True)
os.makedirs(output_folder_end_first, exist_ok=True)

# Determine the middle index
if len(ndpi_files) % 2 == 0:
    middle_index = len(ndpi_files) // 2
else:
    middle_index = (len(ndpi_files) - 1) // 2

# Load the middle image and get its dimensions
middle_image_path = ndpi_files[middle_index]
middle_image = process_slide(middle_image_path, level=5)
target_height, target_width = middle_image.shape[:2]

# Load the images into two separate stacks: one for middle to start, one for middle to end
img_stack_start = [process_slide(f, level=5, target_size=(target_height, target_width)) for f
in ndpi_files[:middle_index+1][::-1]]
img_stack_end = [process_slide(f, level=5, target_size=(target_height, target_width)) for f in
ndpi_files[middle_index:]]

# Convert lists to numpy arrays
img_stack_start = np.array(img_stack_start)
```

```
img_stack_end = np.array(img_stack_end)

# Initialize StackReg
sr = StackReg(StackReg.SCALED_ROTATION)

# Register the stacks
registered_start = sr.register_transform_stack(img_stack_start, reference='previous')[::-1]
registered_end = sr.register_transform_stack(img_stack_end, reference='previous')

# Save the registered images
for i, registered_image in enumerate(registered_start):
    save_image(registered_image, f'registered_{i}', output_folder_start, 'png')
    save_image(registered_image, f'registered_{i}', output_folder_start, 'tif')

for i, registered_image in enumerate(registered_end):
    save_image(registered_image, f'registered_{i}', output_folder_end, 'png')
    save_image(registered_image, f'registered_{i}', output_folder_end, 'tif')

# Register the stacks to the first image in each stack
registered_start_first = sr.register_transform_stack(img_stack_start, reference='first')[::-1]
registered_end_first = sr.register_transform_stack(img_stack_end, reference='first')

# Save the registered images
for i, registered_image in enumerate(registered_start_first):
    save_image(registered_image, f'registered_{i}', output_folder_start_first, 'png')
    save_image(registered_image, f'registered_{i}', output_folder_start_first, 'tif')

for i, registered_image in enumerate(registered_end_first):
    save_image(registered_image, f'registered_{i}', output_folder_end_first, 'png')
    save_image(registered_image, f'registered_{i}', output_folder_end_first, 'tif')
```

```python
# -*- coding: utf-8 -*-
"""
Created on Thu Apr 25 18:14:20 2024

@author: Dell
"""

import os
import glob
from openslide import open_slide
from pystackreg import StackReg
import numpy as np
from PIL import Image
from skimage.transform import resize

def save_image(image, filename, output_folder, file_format):
    image_pil = Image.fromarray(image).convert("L")
    file_path = os.path.join(output_folder, f"{filename}.{file_format}")
    image_pil.save(file_path)
    print(f"{file_format.upper()} image saved successfully at: {file_path}")

def rgba_to_gray(rgba_image):
    rgb_image = rgba_image[:, :, :3]
    gray_image = np.dot(rgb_image[..., :3], [0.2989, 0.5870, 0.1140]) * rgba_image[..., 3] /
255.0
    return gray_image

def process_slide(slide_path, level, target_size=None):
    slide = open_slide(slide_path)
    dims = slide.level_dimensions
    level_img = slide.read_region((0, 0), level, dims[level])
    level_img_np = np.array(level_img, dtype='uint8')
    level_img_np_grayscale = rgba_to_gray(level_img_np)
    if target_size is not None:
        level_img_np_grayscale_resized = resize(level_img_np_grayscale, target_size,
anti_aliasing=True)
        return level_img_np_grayscale_resized
    else:
        return level_img_np_grayscale

input_folder = 'F:/Germany_2022/TU Illmenau/hiwi/DataSets/histology/NDp'
ndpi_files = sorted(glob.glob(os.path.join(input_folder, '*.ndpi')))
output_folder_start = 'F:/Germany_2022/TU
Illmenau/hiwi/DataSets/histology/Scaled/Stack_transform_middle_to_start/previous'
output_folder_end = 'F:/Germany_2022/TU
Illmenau/hiwi/DataSets/histology/Scaled/Stack_transform_middle_to_end/previous'
output_folder_start_first = 'F:/Germany_2022/TU
Illmenau/hiwi/DataSets/histology/Scaled/Stack_transform_middle_to_start/first'
output_folder_end_first = 'F:/Germany_2022/TU
Illmenau/hiwi/DataSets/histology/Scaled/Stack_transform_middle_to_end/first'
os.makedirs(output_folder_start, exist_ok=True)
os.makedirs(output_folder_end, exist_ok=True)
os.makedirs(output_folder_start_first, exist_ok=True)
os.makedirs(output_folder_end_first, exist_ok=True)

# Determine the middle index
if len(ndpi_files) % 2 == 0:
    middle_index = len(ndpi_files) // 2
else:
    middle_index = (len(ndpi_files) - 1) // 2

# Load the middle image and get its dimensions
middle_image_path = ndpi_files[middle_index]
middle_image = process_slide(middle_image_path, level=5)
target_height, target_width = middle_image.shape[:2]

# Load the images into two separate stacks: one for middle to start, one for middle to end
```

```
img_stack_start = [process_slide(f, level=5, target_size=(target_height, target_width)) for f
in ndpi_files[:middle_index+1][::-1]]
img_stack_end = [process_slide(f, level=5, target_size=(target_height, target_width)) for f in
ndpi_files[middle_index:]]

# Convert lists to numpy arrays
img_stack_start = np.array(img_stack_start)
img_stack_end = np.array(img_stack_end)

# Initialize StackReg
sr = StackReg(StackReg.SCALED_ROTATION)

# Register the stacks
registered_start = sr.register_transform_stack(img_stack_start, reference='previous')[::-1]
registered_end = sr.register_transform_stack(img_stack_end, reference='previous')

# Save the registered images
for i, registered_image in enumerate(registered_start):
    save_image(registered_image, f'registered_{i}', output_folder_start, 'png')
    save_image(registered_image, f'registered_{i}', output_folder_start, 'tif')

for i, registered_image in enumerate(registered_end):
    save_image(registered_image, f'registered_{i}', output_folder_end, 'png')
    save_image(registered_image, f'registered_{i}', output_folder_end, 'tif')

# Register the stacks to the first image in each stack
registered_start_first = sr.register_transform_stack(img_stack_start, reference='first')[::-1]
registered_end_first = sr.register_transform_stack(img_stack_end, reference='first')

# Save the registered images
for i, registered_image in enumerate(registered_start_first):
    save_image(registered_image, f'registered_{i}', output_folder_start_first, 'png')
    save_image(registered_image, f'registered_{i}', output_folder_start_first, 'tif')

for i, registered_image in enumerate(registered_end_first):
    save_image(registered_image, f'registered_{i}', output_folder_end_first, 'png')
    save_image(registered_image, f'registered_{i}', output_folder_end_first, 'tif')
```

```python
# -*- coding: utf-8 -*-
"""
Created on Tue Apr  2 16:44:23 2024

@author: Niloy Roy
"""
from openslide import open_slide
import glob
import os
from pystackreg import StackReg
from skimage.transform import resize
import numpy as np
from PIL import Image

def save_image(image, filename, output_folder, file_format):
    # Convert the image to a PIL Image object
    image_pil = Image.fromarray(image).convert("L")

    # Specify the file path to save the image
    file_path = os.path.join(output_folder, f"{filename}.{file_format}")

    # Save the image
    image_pil.save(file_path)

    print(f"{file_format.upper()} image saved successfully at: {file_path}")

def rgba_to_gray(rgba_image):
    # Extract the RGB channels
    rgb_image = rgba_image[:, :, :3]
    # Convert to grayscale while considering alpha channel
    gray_image = np.dot(rgb_image[..., :3], [0.2989, 0.5870, 0.1140]) * rgba_image[..., 3] /
255.0
    return gray_image

def process_slide(slide_path):
    # Load the slide file into an object.
    slide = open_slide(slide_path)

    # Get slide dims at each level.
    dims = slide.level_dimensions
    num_levels = len(dims)

    # Get the index of the highest level
    highest_level_index = num_levels - 1

    # Get dimensions of the highest level
    highest_level_dim = dims[highest_level_index]

    level_img = slide.read_region((0, 0), highest_level_index, highest_level_dim)  # Pillow
object, mode=RGBA

    # Convert the image into a numpy array for processing
    level_img_np = np.array(level_img, dtype='uint8')

    level_img_np_grayscale = rgba_to_gray(level_img_np)

    return level_img_np_grayscale

# Function to process and resize images
def resize_image(img, max_height, max_width):
    #img = process_slide(filename)
    # Assuming process_slide returns a numpy array
    img_resized = resize(img, (max_height, max_width), order=4, anti_aliasing=True,
preserve_range=True).astype(np.uint8)
    return img_resized

# Load the NDPI files
```

```
# Set the desired input folder path
input_folder = 'F:/Germany_2022/TU Illmenau/hiwi/DataSets/histology/NDp'

#'F:/Germany_2022/TU Illmenau/hiwi/DataSets/histology/NDp'
#'F:/Germany_2022/TU Illmenau/hiwi/DataSets/histology/test_input'

# Get the list of NDPI files in the input folder
ndpi_files = glob.glob(os.path.join(input_folder, '*.ndpi'))

if len(ndpi_files) % 2 == 0:
    middle_index = len(ndpi_files) // 2
else:
    middle_index = (len(ndpi_files) - 1) // 2

# Initialize StackReg
sr = StackReg(StackReg.RIGID_BODY)

# Set the desired folder path
output_folder = 'F:/Germany_2022/TU Illmenau/hiwi/DataSets/histology/NDp_coregistered'
#'F:/Germany_2022/TU Illmenau/hiwi/DataSets/histology/test_output'
#'F:/Germany_2022/TU Illmenau/hiwi/DataSets/histology/NDp_coregistered'

# Create the output folder if it doesn't exist
os.makedirs(output_folder, exist_ok=True)

# Process and resize images starting from the middle index
reference_image_path = ndpi_files[middle_index]
reference_image = process_slide(reference_image_path)
reference_height, reference_width = reference_image.shape[:2]

# Save the reference image
save_image(reference_image, f'reference_{middle_index}', output_folder, 'png')

for i in range(middle_index + 1, len(ndpi_files)):
    # Process the offset image
    offset_image_path = ndpi_files[i]
    offset_image = process_slide(offset_image_path)
    offset_height, offset_width = offset_image.shape[:2]

    # Find the maximum height and width of the reference and offset images
    max_height = max(reference_height, offset_height)
    max_width = max(reference_width, offset_width)

    # Resize the reference and offset images to the maximum dimensions
    reference_image_resized = resize_image(reference_image, max_height, max_width)
    offset_image_resized = resize_image(offset_image, max_height, max_width)

    # Co-register the offset image to the reference image
    registered_image = sr.register_transform(reference_image_resized, offset_image_resized)

    # Save the reference and registered images
    save_image(reference_image_resized, f'reference_{middle_index}', output_folder, 'png')
    save_image(registered_image, f'registered_{i}', output_folder, 'png')

    # Update the reference image and its dimensions for the next iteration
    reference_image = registered_image
    reference_height, reference_width = reference_image.shape[:2]
```

MedVis 23 - Exercise Sheet 1 - Additional Material

Preparation

Here the necessary libraries are included.

```
!pip install scipy
!pip install pydicom

import os # for file-handling
import numpy as np # maths
import matplotlib.pyplot as plt # plotting
from scipy import ndimage # image processing
from pydicom import dcmread # reading DICOM files
```

```
Requirement already satisfied: scipy in
/usr/local/lib/python3.10/dist-packages (1.11.3)
Requirement already satisfied: numpy<1.28.0,>=1.21.6 in
/usr/local/lib/python3.10/dist-packages (from scipy) (1.23.5)
Collecting pydicom
  Downloading pydicom-2.4.3-py3-none-any.whl (1.8 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 8.7 MB/s eta
0:00:00

Successfully installed pydicom-2.4.3
```

Example of how to read and display a DICOM dataset

Here is some sample code how medical image data can be read and displayed using Python. The dataset is from Jeff Mather (DICOM Example Files, MATLAB Central File Exchange. Retrieved April 8, 2022.).

As you may know, medical imaging like CT and MRI often produce volume data by generating cross-sectional slices through the body. One way to view this data is by visualising each slice. In our DICOM data set each slice is saved as a separate file. Therefore, you first have to traverse the slice files to read and then save them in an 3D array. This is done by the function read_dcm_volume. You can then choose one slice of this volume by entering its index, e.g. `dcm_volume[:,:,8]`.

Alternatively, you can read one specific slice file right away using its file name, e.g. `dcmread("dataset1/brain_011.dcm")`.

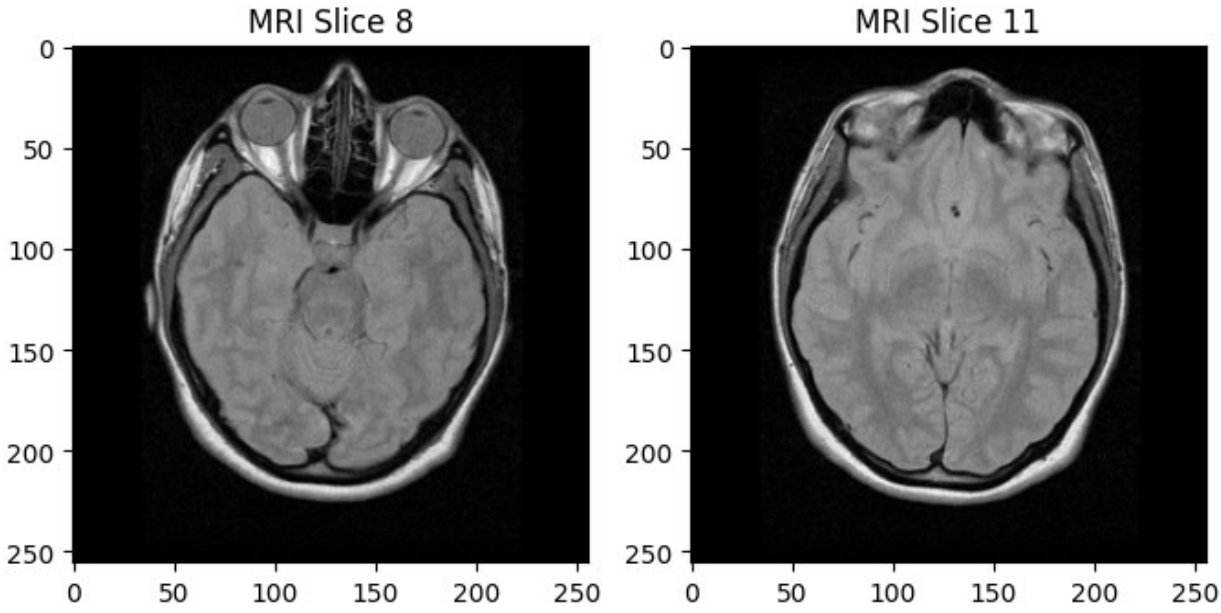
To display a slice using a python figure, we have to convert it to an pixel array, e.g. `np.array(dcm_slice.pixel_array)`.

You can see the example code below. It might be useful for the following tasks.

```
# Read a volume and convert to numpy array
def read_dcm_volume(name):
    volume_img = []
    for file in sorted(os.listdir(name)):
        dcm_slice = dcmread(os.path.join(name, file))
        volume_img.append(dcm_slice.pixel_array)
    return np.transpose(np.array(volume_img),(1,2,0))


# load the volume
dcm_volume = read_dcm_volume('/opt/google/drive/dicom_data') # make
sure you have the correct path
# extract slice 8 in z-direction
img1 = dcm_volume[:,:,8]


# load one slice
dcm_slice = dcmread("/opt/google/drive/dicom_data/brain_011.dcm") #
make sure you have the correct path
# convert it to numpy array for plotting the image
img2 = np.array(dcm_slice.pixel_array)


# Create a figure (window)
fig = plt.figure(figsize=(8,4))

# display using gray levels
ax1 = fig.add_subplot(1, 2, 1)
ax1.set_title('MRI Slice 8')
ax1.imshow(img1, cmap='gray')

ax2 = fig.add_subplot(1, 2, 2)
ax2.set_title('MRI Slice 11')
ax2.imshow(img2, cmap='gray')

# Extract and print the required parameters
print("Imaging Modality:", dcm_slice.Modality)
print("Slice Thickness in z-direction:", dcm_slice.SliceThickness)
print("Pixel Spacing:", dcm_slice.PixelSpacing)
print("Grid Size (Rows, Columns):", dcm_slice.Rows, ",",
dcm_slice.Columns)
print("Smallest Image Pixel Value:",
dcm_slice.SmallestImagePixelValue)
```

```
print("Largest Image Pixel Value:", dcm_slice.LargestImagePixelValue)
```

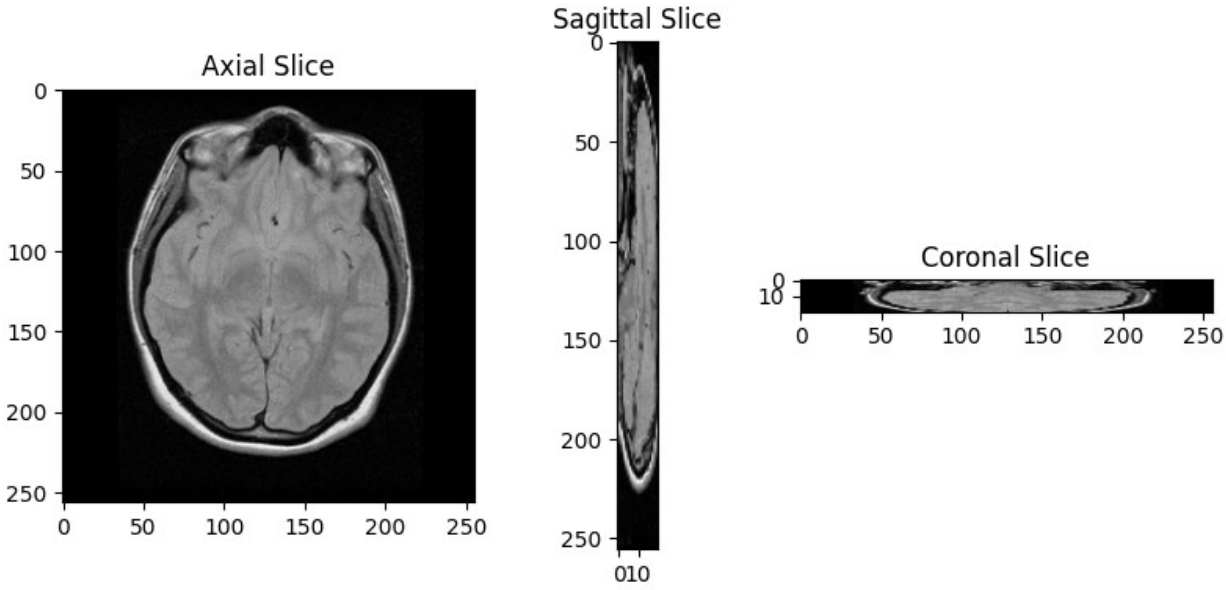
```
Imaging Modality: MR
Slice Thickness in z-direction: 5.00000
Pixel Spacing: [0.859375, 0.859375]
Grid Size (Rows, Columns): 256 , 256
Smallest Image Pixel Value: 0
Largest Image Pixel Value: 901
```



The problem of anisotrophy

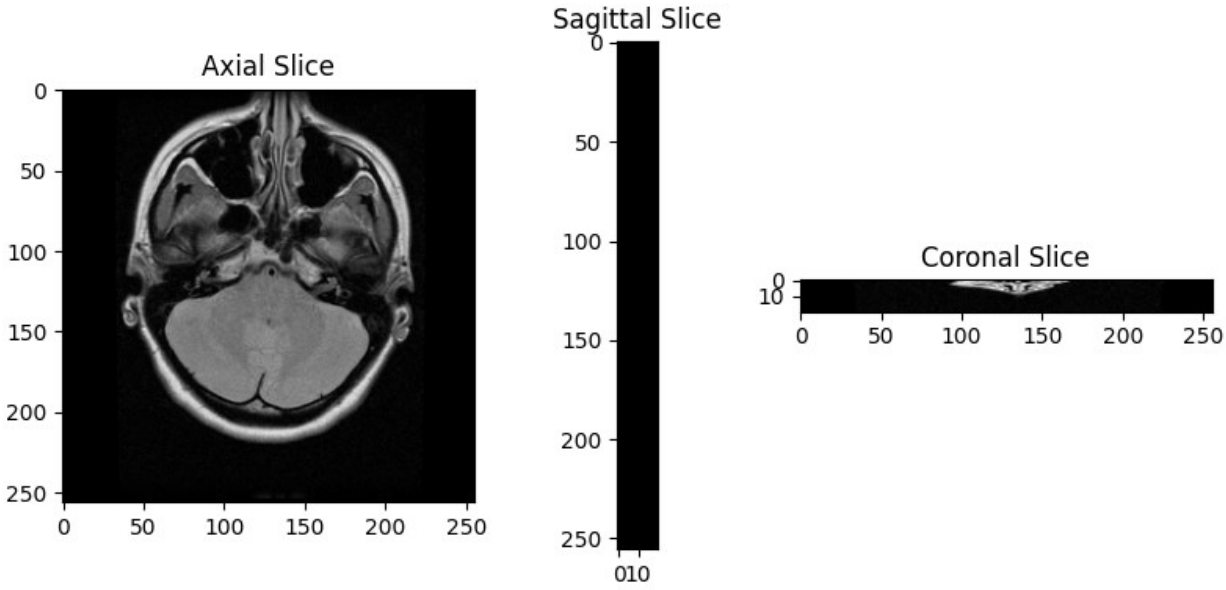
```
# Given dcm_volume is your 3D image data with size [256, 256, 20]


axial_slice_index = 10  # Choose a middle slice index for axial view
sagittal_slice_index = 128  # Choose a middle slice index for sagittal
view
coronal_slice_index = 128  # Choose a middle slice index for coronal
view
```

```
# Create a figure (window)
fig = plt.figure(figsize=(8, 4))

# Axial Slice
axial = plt.subplot(1, 3, 1)
axial.set_title('Axial Slice')
axial.imshow(dcm_volume[:, :, axial_slice_index], cmap='gray')

# Sagittal Slice
sagittal = plt.subplot(1, 3, 2)
sagittal.set_title('Sagittal Slice')
sagittal.imshow(dcm_volume[:, sagittal_slice_index, :], cmap='gray')

# Coronal Slice
coronal = plt.subplot(1, 3, 3)
coronal.set_title('Coronal Slice')
coronal.imshow(dcm_volume[coronal_slice_index, :, :].T, cmap='gray')
# Transpose for correct orientation

plt.tight_layout()
plt.show()
```



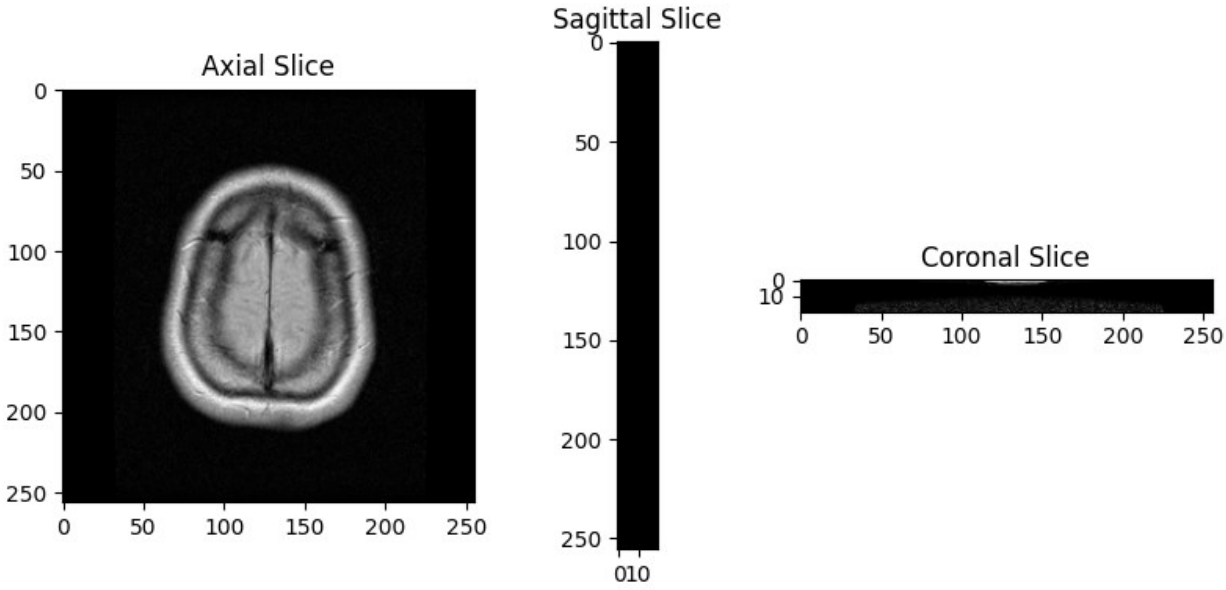

```
# Given dcm_volume is your 3D image data with size [256, 256, 20]


axial_slice_index = 5  # Choose an early slice index for axial view
sagittal_slice_index = 10  # Choose an initial slice index for the
sagittal view
coronal_slice_index = 10 # Choose an initial slice index for coronal
view
```

```
# Create a figure (window)
fig = plt.figure(figsize=(8, 4))

# Axial Slice
axial = plt.subplot(1, 3, 1)
axial.set_title('Axial Slice')
axial.imshow(dcm_volume[:, :, axial_slice_index], cmap='gray')

# Sagittal Slice
sagittal = plt.subplot(1, 3, 2)
sagittal.set_title('Sagittal Slice')
sagittal.imshow(dcm_volume[:, sagittal_slice_index, :], cmap='gray')

# Coronal Slice
coronal = plt.subplot(1, 3, 3)
coronal.set_title('Coronal Slice')
coronal.imshow(dcm_volume[coronal_slice_index, :, :].T, cmap='gray')
# Transpose for correct orientation

plt.tight_layout()
plt.show()
```




```
# Given dcm_volume is your 3D image data with size [256, 256, 20]


axial_slice_index = 19  # Choose a later slice index for axial view
sagittal_slice_index = 250  # Choose a later initial slice index for
the sagittal view
coronal_slice_index = 254 # Choose an later slice index for coronal
view
```

```
# Create a figure (window)
fig = plt.figure(figsize=(8, 4))

# Axial Slice
axial = plt.subplot(1, 3, 1)
axial.set_title('Axial Slice')
axial.imshow(dcm_volume[:, :, axial_slice_index], cmap='gray')

# Sagittal Slice
sagittal = plt.subplot(1, 3, 2)
sagittal.set_title('Sagittal Slice')
sagittal.imshow(dcm_volume[:, sagittal_slice_index, :], cmap='gray')

# Coronal Slice
coronal = plt.subplot(1, 3, 3)
coronal.set_title('Coronal Slice')
coronal.imshow(dcm_volume[coronal_slice_index, :, :].T, cmap='gray')
# Transpose for correct orientation

plt.tight_layout()
plt.show()
```



MedVis 23 - Exercise Sheet 3 - Additional Material

Preparation

Here the necessary libraries are included.

```
!pip install scipy
!pip install pydicom
!pip install PyMCubes


import os # for file-handling
import numpy as np # maths
import matplotlib.pyplot as plt # plotting
from scipy import ndimage # image processing
from pydicom import dcmread # reading DICOM files

import plotly.graph_objects as go # 3D plot
import mcubes
```

```
Requirement already satisfied: scipy in
/usr/local/lib/python3.10/dist-packages (1.11.3)
Requirement already satisfied: numpy<1.28.0,>=1.21.6 in
/usr/local/lib/python3.10/dist-packages (from scipy) (1.23.5)
Requirement already satisfied: pydicom in
/usr/local/lib/python3.10/dist-packages (2.4.3)
Collecting PyMCubes
  Downloading PyMCubes-0.1.4-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (274 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 274.3/274.3 kB 4.8 MB/s eta
0:00:00
ent already satisfied: numpy in /usr/local/lib/python3.10/dist-
packages (from PyMCubes) (1.23.5)
Requirement already satisfied: scipy>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from PyMCubes) (1.11.3)
Installing collected packages: PyMCubes
Successfully installed PyMCubes-0.1.4
```

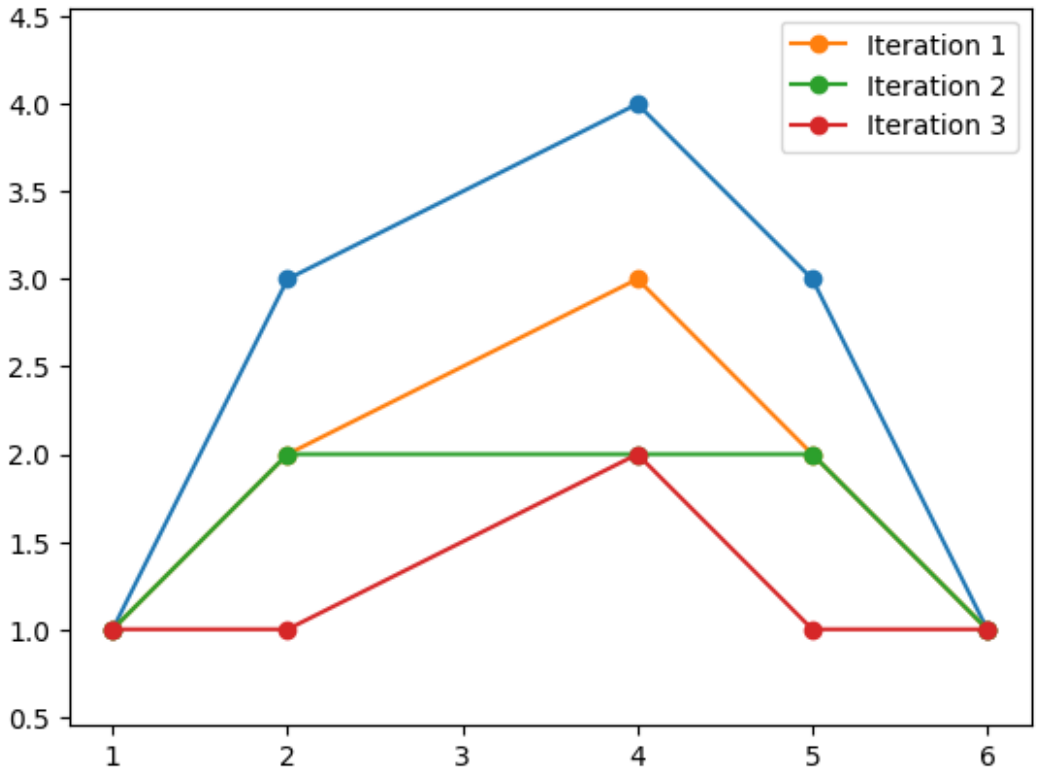
Function for reading the MRI volume

```
# Read a volume and convert to numpy array
def read_dcm_volume(name):
```

```
    volume_img = []
    for file in sorted(os.listdir(name)):
        dcm_slice = dcmread(os.path.join(name, file))
        volume_img.append(dcm_slice.pixel_array)
    return np.transpose(np.array(volume_img),(1,2,0))
```

Task 4

```
x = np.array([1,2,4,5,6])
y = np.array([1,3,4,3,1])

plt.plot(x,y)
plt.scatter(x,y)
plt.gca().axis('equal')


def laplacian_smoothing(x, y, iterations, lam):
    for _ in range(iterations):
        smoothed_y = y.copy()
        for i in range(1, len(y) - 1):
            smoothed_y[i] = (1 - lam) * y[i] + lam * (y[i - 1] + y[i +
1]) / 2
        y = smoothed_y

    return y

# Perform Laplacian smoothing for three iterations with lambda (lam)
value
lam = 0.2  # You can adjust this value based on your preference
iterations = 3

for iteration in range(1, iterations + 1):
    y_smoothed = laplacian_smoothing(x, y, iteration, lam)
    plt.plot(x, y_smoothed, label=f'Iteration {iteration}',
marker='o')

plt.legend()
plt.gca().axis('equal')
plt.show()
```



Task 5

```
# create a 3D data set of the slice images, like in the previous
exercise sheets
# this can be used as input for the marching cubes algorithm
dcm_volume = read_dcm_volume('/opt/google/drive/dicom_data')

############################
# Use Marching Cubes here #
############################
# set the appropriate isovalue based on your data
isovalue = 1000

# Generate surface mesh using Marching Cubes
vertices, faces = mcubes.marching_cubes(dcm_volume, isovalue)

# after generating vertices and faces using marching cubes
# adjust the position of the data (remember that the data set is
anisotropic)
vertices[:,0] *= 0.859375
vertices[:,1] *= 0.859375
vertices[:,2] *= 7
```

Below is some example code for plotting the data. Remember to rename the variables so that it will fit to your previous code. If you do not manage to plot the data, you can also export it to an .OBJ file using PyMCubes.

```
# transpose matrices
x, y, z = vertices.T
i,j,k = faces.T

#
fig = go.Figure(data=[go.Mesh3d(x=x, y=y, z=z, i=i,j=k,k=j,
flatshading=True,color='lightpink', opacity=1.00,
              lighting=dict(ambient=0.18,
                            diffuse=1,
                            fresnel=0.1,
                            specular=1,
                            roughness=0.05,
                            facenormalsepsilon=1e-15,
                            vertexnormalsepsilon=1e-15),
               lightposition=dict(x=0,
                                  y=100,
                                  z=100
                                 ))])

fig.show()
```

MedVis 23 - Exercise Sheet 1 - Additional Material

Preparation

Here the necessary libraries are included.

```
!pip install scipy
!pip install pydicom

import os # for file-handling
import numpy as np # maths
import matplotlib.pyplot as plt # plotting
from scipy import ndimage # image processing
from pydicom import dcmread # reading DICOM files
```

```
Requirement already satisfied: scipy in
/usr/local/lib/python3.10/dist-packages (1.11.3)
Requirement already satisfied: numpy<1.28.0,>=1.21.6 in
/usr/local/lib/python3.10/dist-packages (from scipy) (1.23.5)
Collecting pydicom
  Downloading pydicom-2.4.3-py3-none-any.whl (1.8 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 18.5 MB/s eta
0:00:00

Successfully installed pydicom-2.4.3
```

Example of how to read and display a DICOM dataset

Here is some sample code how medical image data can be read and displayed using Python. The dataset is from Jeff Mather (DICOM Example Files, MATLAB Central File Exchange. Retrieved April 8, 2022.).

As you may know, medical imaging like CT and MRI often produce volume data by generating cross-sectional slices through the body. One way to view this data is by visualising each slice. In our DICOM data set each slice is saved as a separate file. Therefore, you first have to traverse the slice files to read and then save them in an 3D array. This is done by the function read_dcm_volume. You can then choose one slice of this volume by entering its index, e.g. `dcm_volume[:,:,8]`.

Alternatively, you can read one specific slice file right away using its file name, e.g. `dcmread("dataset1/brain_011.dcm")`.

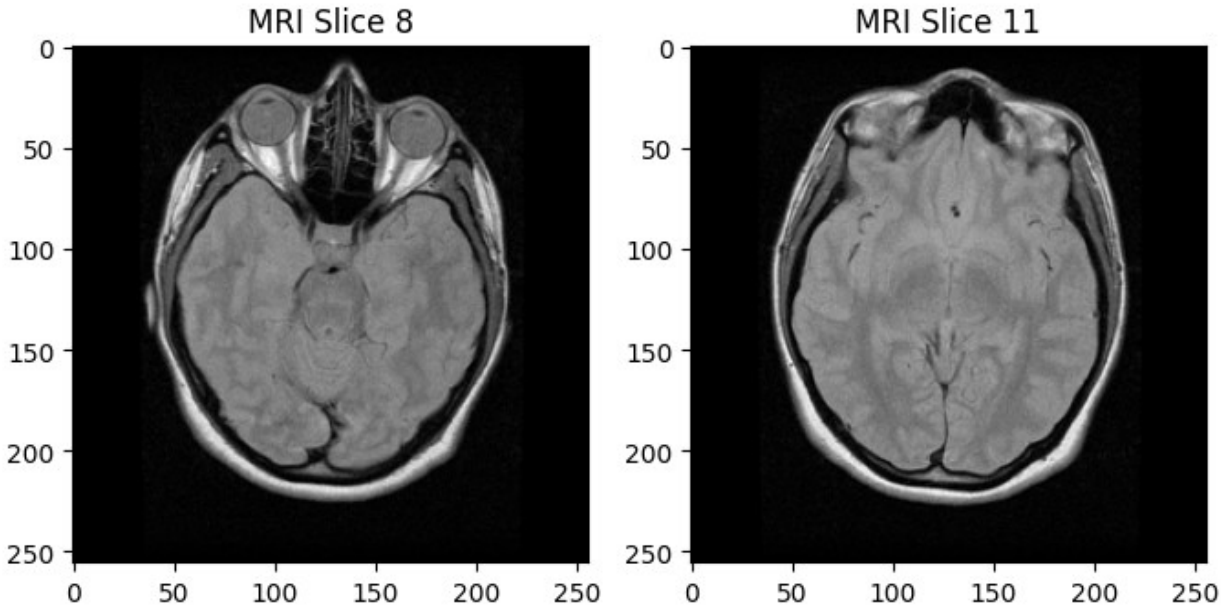
To display a slice using a python figure, we have to convert it to an pixel array, e.g. `np.array(dcm_slice.pixel_array)`.

You can see the example code below. It might be useful for the following tasks.

```
# Read a volume and convert to numpy array
def read_dcm_volume(name):
    volume_img = []
    for file in sorted(os.listdir(name)):
        dcm_slice = dcmread(os.path.join(name, file))
        volume_img.append(dcm_slice.pixel_array)
    return np.transpose(np.array(volume_img),(1,2,0))


# load the volume
dcm_volume = read_dcm_volume('/opt/google/drive/dicom_data') # make
sure you have the correct path
# extract slice 8 in z-direction
img1 = dcm_volume[:,:,8]


# load one slice
dcm_slice = dcmread("/opt/google/drive/dicom_data/brain_011.dcm") #
make sure you have the correct path
# convert it to numpy array for plotting the image
img2 = np.array(dcm_slice.pixel_array)


# Create a figure (window)
fig = plt.figure(figsize=(8,4))

# display using gray levels
ax1 = fig.add_subplot(1, 2, 1)
ax1.set_title('MRI Slice 8')
ax1.imshow(img1, cmap='gray')

ax2 = fig.add_subplot(1, 2, 2)
ax2.set_title('MRI Slice 11')
ax2.imshow(img2, cmap='gray')

# Extract and print the required parameters
print("Imaging Modality:", dcm_slice.Modality)
print("Slice Thickness in z-direction:", dcm_slice.SliceThickness)
print("Pixel Spacing:", dcm_slice.PixelSpacing)
print("Grid Size (Rows, Columns):", dcm_slice.Rows, ",",
dcm_slice.Columns)
print("Smallest Image Pixel Value:",
dcm_slice.SmallestImagePixelValue)
```

```
print("Largest Image Pixel Value:", dcm_slice.LargestImagePixelValue)
```

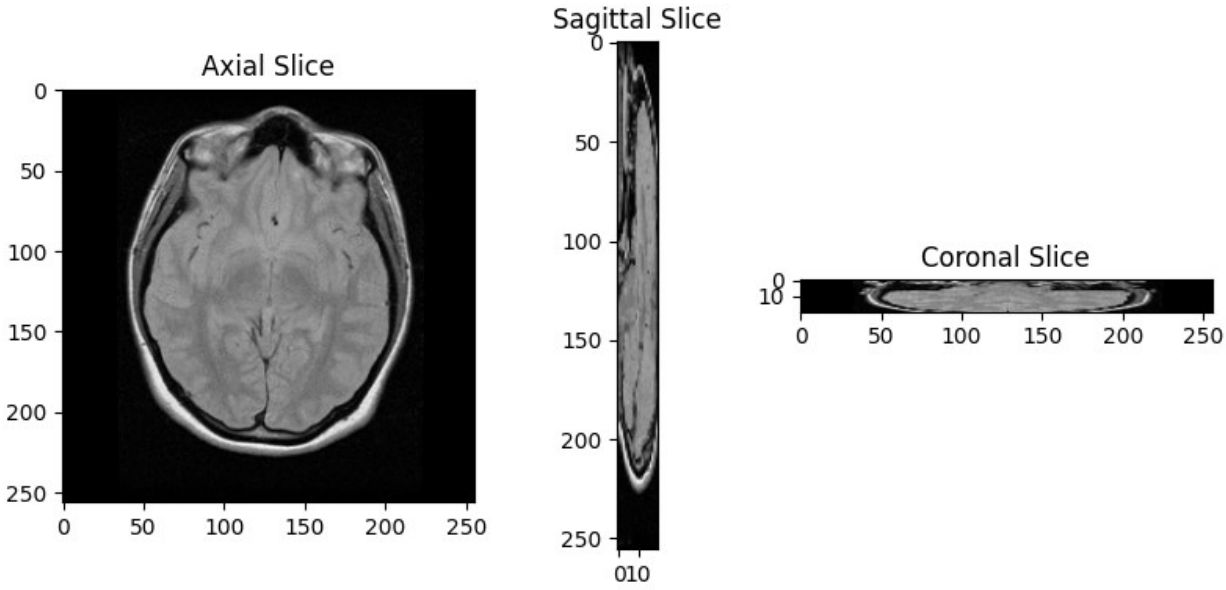
```
Imaging Modality: MR
Slice Thickness in z-direction: 5.00000
Pixel Spacing: [0.859375, 0.859375]
Grid Size (Rows, Columns): 256 , 256
Smallest Image Pixel Value: 0
Largest Image Pixel Value: 901
```



The problem of anisotrophy

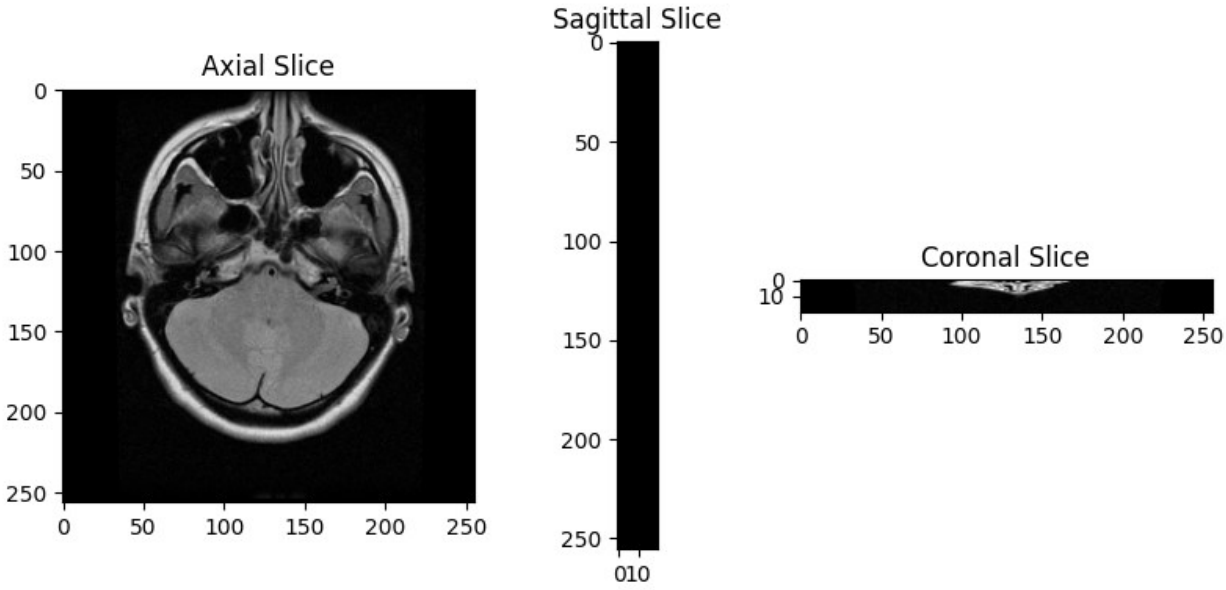
```
# Given dcm_volume is your 3D image data with size [256, 256, 20]


axial_slice_index = 10  # Choose a middle slice index for axial view
sagittal_slice_index = 128  # Choose a middle slice index for sagittal
view
coronal_slice_index = 128  # Choose a middle slice index for coronal
view
```

```
# Create a figure (window)
fig = plt.figure(figsize=(8, 4))

# Axial Slice
axial = plt.subplot(1, 3, 1)
axial.set_title('Axial Slice')
axial.imshow(dcm_volume[:, :, axial_slice_index], cmap='gray')

# Sagittal Slice
sagittal = plt.subplot(1, 3, 2)
sagittal.set_title('Sagittal Slice')
sagittal.imshow(dcm_volume[:, sagittal_slice_index, :], cmap='gray')

# Coronal Slice
coronal = plt.subplot(1, 3, 3)
coronal.set_title('Coronal Slice')
coronal.imshow(dcm_volume[coronal_slice_index, :, :].T, cmap='gray')
# Transpose for correct orientation

plt.tight_layout()
plt.show()
```



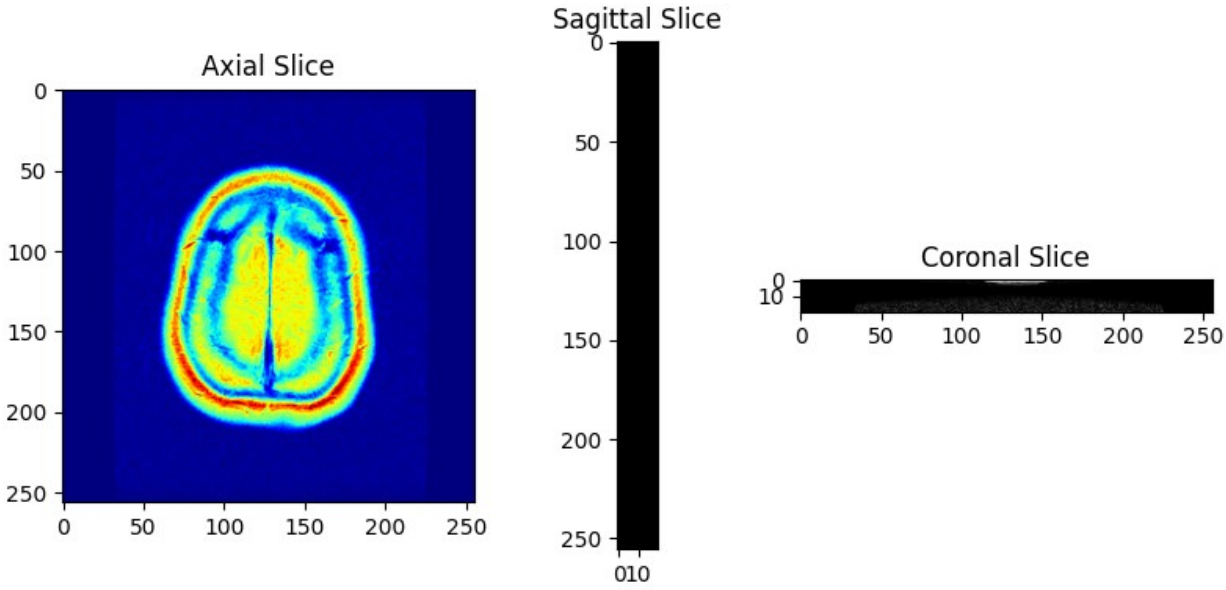

```
# Given dcm_volume is your 3D image data with size [256, 256, 20]


axial_slice_index = 5  # Choose an early slice index for axial view
sagittal_slice_index = 10  # Choose an initial slice index for the
sagittal view
coronal_slice_index = 10 # Choose an initial slice index for coronal
view
```

```
# Create a figure (window)
fig = plt.figure(figsize=(8, 4))

# Axial Slice
axial = plt.subplot(1, 3, 1)
axial.set_title('Axial Slice')
axial.imshow(dcm_volume[:, :, axial_slice_index], cmap='gray')

# Sagittal Slice
sagittal = plt.subplot(1, 3, 2)
sagittal.set_title('Sagittal Slice')
sagittal.imshow(dcm_volume[:, sagittal_slice_index, :], cmap='gray')

# Coronal Slice
coronal = plt.subplot(1, 3, 3)
coronal.set_title('Coronal Slice')
coronal.imshow(dcm_volume[coronal_slice_index, :, :].T, cmap='gray')
# Transpose for correct orientation

plt.tight_layout()
plt.show()
```




```
# Given dcm_volume is your 3D image data with size [256, 256, 20]


axial_slice_index = 19  # Choose a later slice index for axial view
sagittal_slice_index = 250  # Choose a later initial slice index for
the sagittal view
coronal_slice_index = 254 # Choose an later slice index for coronal
view
```

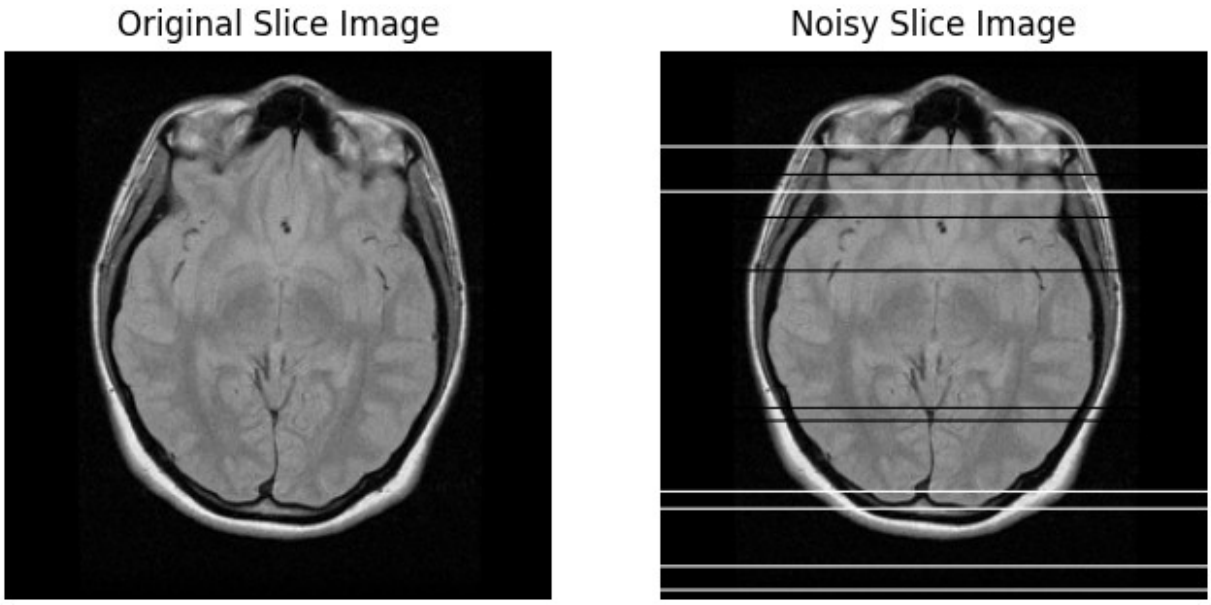
```
# Create a figure (window)
fig = plt.figure(figsize=(8, 4))

# Axial Slice
axial = plt.subplot(1, 3, 1)
axial.set_title('Axial Slice')
axial.imshow(dcm_volume[:, :, axial_slice_index], cmap='jet')

# Sagittal Slice
sagittal = plt.subplot(1, 3, 2)
sagittal.set_title('Sagittal Slice')
sagittal.imshow(dcm_volume[:, sagittal_slice_index, :], cmap='gray')

# Coronal Slice
coronal = plt.subplot(1, 3, 3)
coronal.set_title('Coronal Slice')
coronal.imshow(dcm_volume[coronal_slice_index, :, :].T, cmap='gray')
# Transpose for correct orientation

plt.tight_layout()
plt.show()
```



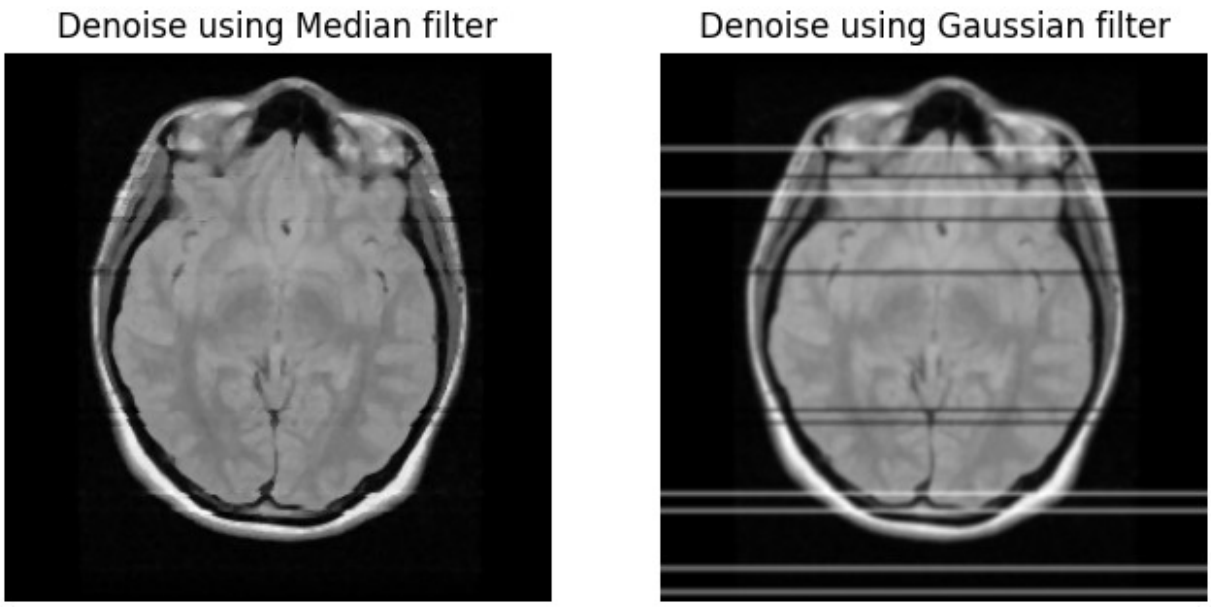
TASK 3: Image Preprocessing - Smoothing

```
import numpy as np
from scipy import ndimage

def add_salt_pepper_noise(img):
    # Need to produce a copy as to not modify the original image
    img_copy = img.copy()
    row, col = img_copy.shape
```

```
    salt_vs_pepper = 0.5
    amount = 0.00009
    num_salt = np.ceil(amount * img_copy.size * salt_vs_pepper)
    num_pepper = np.ceil(amount * img_copy.size * (1.0 -
salt_vs_pepper))

    # Add Salt noise
    coords = [np.random.randint(0, i - 1, int(num_salt)) for i in
img_copy.shape]
    img_copy[coords] = 901

    # Add Pepper noise
    coords = [np.random.randint(0, i - 1, int(num_pepper)) for i in
img_copy.shape]
    img_copy[coords] = 0

    return img_copy

noisy_slice_image = add_salt_pepper_noise(img2)
# Plot the original and noisy images
plt.figure(figsize=(8, 4))

plt.subplot(1, 2, 1)
plt.imshow(img2, cmap='gray')
plt.title('Original Slice Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(noisy_slice_image, cmap='gray')
plt.title('Noisy Slice Image')
plt.axis('off')

plt.show()
```




```
# Denoise using Median filter
img_copy_median = ndimage.median_filter(noisy_slice_image, size=3)

# Denoise using Gaussian filter
img_copy_gaussian = ndimage.gaussian_filter(noisy_slice_image,
sigma=1)

# Plot the original and noisy images
plt.figure(figsize=(8, 4))

plt.subplot(1, 2, 1)
plt.imshow(img_copy_median, cmap='gray')
plt.title('Denoise using Median filter')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(img_copy_gaussian, cmap='gray')
plt.title('Denoise using Gaussian filter')
plt.axis('off')

plt.show()
```



In the context of medical visualization, the choice between a Median filter and a Gaussian filter depends on the specific characteristics of the noise and the desired outcome. Here's how each filter works and their suitability in different scenarios:

Median Filter:

- **How it Works:** A Median filter replaces each pixel's value with the median value in its neighborhood. It is effective in removing salt-and-pepper noise (random, isolated pixels with extreme values).
- **Suitability:**
  - **Works Well for Salt-and-Pepper Noise:** Median filters are excellent for reducing salt-and-pepper noise, a common type of noise in medical images caused by sensor errors or transmission issues.
  - **Preserves Edges:** Median filters are good at preserving edges and fine details in the image, making them suitable for images with thin structures like vessels.
  - **Not Suited for Gaussian Noise:** Median filters are less effective for reducing Gaussian noise, which is a continuous and smooth type of noise. Applying a Median filter to Gaussian noise might result in a loss of image details.

Gaussian Filter:

- **How it Works:** A Gaussian filter calculates the weighted average of pixel values in the neighborhood, giving more weight to closer pixels. It is effective for reducing Gaussian noise (random noise with a Gaussian distribution).
- **Suitability:**
  - **Works Well for Gaussian Noise:** Gaussian filters are suitable for reducing Gaussian noise, which is prevalent in medical images due to various factors like signal interference.
  - **Smoothens Image:** Gaussian filters smooth out the image and reduce high-frequency noise, making them suitable for images where a more uniform appearance is desired.

- **May Blur Edges:** Gaussian filters can blur sharp edges and fine details in the image. While they reduce noise, they might not be the best choice for preserving intricate structures like blood vessels.

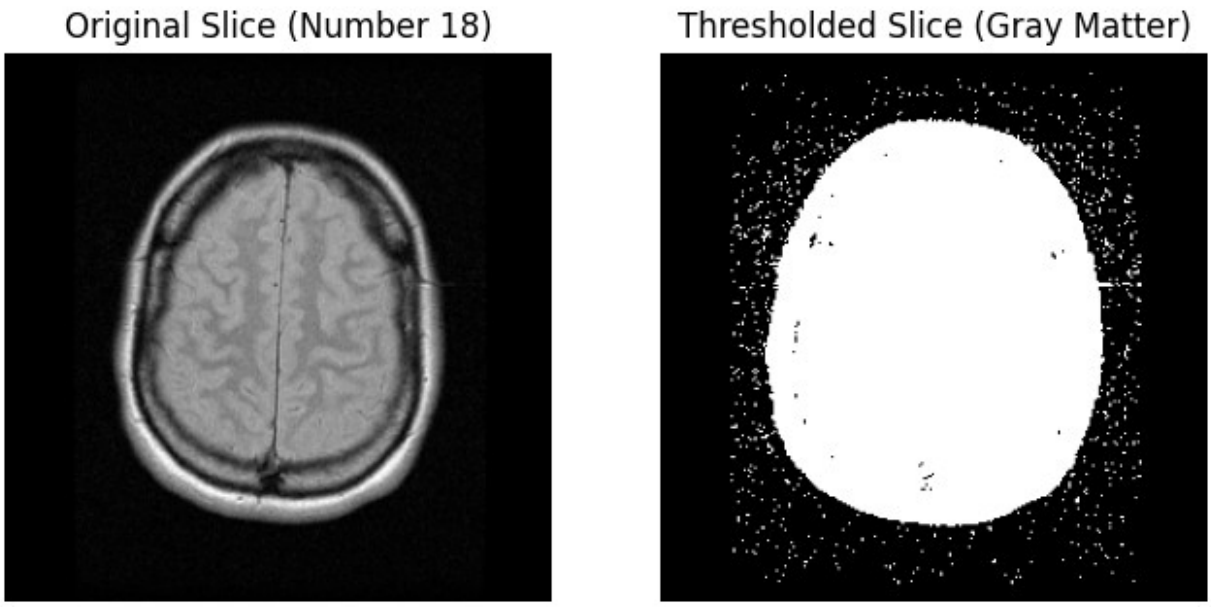
Typical Artifacts and Considerations:

- **Thin Structures (Vessels):** Images with thin structures like blood vessels require filters that preserve edges and details. Median filters are often preferred in such cases as they preserve edges well.
- **Typical Artifacts:** Medical images can suffer from artifacts such as **salt-and-pepper noise**, **motion artifacts**, and **aliasing artifacts**. Median filters are effective against salt-and-pepper noise, while Gaussian filters can help in reducing motion artifacts.

In summary, **if the primary concern is reducing salt-and-pepper noise and preserving fine details (such as blood vessels), a Median filter might be a better choice. If the noise is Gaussian in nature and a smoother image is desired, a Gaussian filter might be more appropriate. Often, a combination of filtering techniques or more advanced denoising methods may be used in practice to achieve the desired results**. The choice depends on the specific characteristics of the noise and the details of the anatomical structures in the medical images.

TASK 4: Image Segmentation

```
# Assuming dcm_volume is loaded from the DICOM files as mentioned in
the previous code snippet
# Slice number 18 (assuming 0-based index)
slice_number = 17  # Note: Python uses 0-based indexing

# Threshold value (adjust this based on the intensity characteristics
of gray matter in your images)
threshold_value = 35  # Example threshold value, adjust as needed

# Perform simple thresholding on slice number 18
thresholded_slice = dcm_volume[:, :, slice_number] > threshold_value

# Plot the original and thresholded images for visualization
plt.figure(figsize=(8, 4))

plt.subplot(1, 2, 1)
plt.imshow(dcm_volume[:, :, slice_number], cmap='gray')
plt.title('Original Slice (Number 18)')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(thresholded_slice, cmap='gray')
plt.title('Thresholded Slice (Gray Matter)')
plt.axis('off')

plt.show()
```



In this code, threshold_value **represents the intensity threshold above which pixels are considered part of the gray matter**. You may need to adjust this threshold value based on the intensity characteristics of the gray matter in your specific images. **The thresholding operation creates a binary mask (True for gray matter, False for non-gray matter) based on the threshold condition**.

Follow up Question

Then think about the following questions from the perspective of medical visualization: • Which filter works better? Why? • In which cases is a median filter suited / not suited? • In which cases is a gaussian filter suited / not suited? Try to think about different characteristics of anatomical structures, e.g., how about images with thin structures such as vessels? Can you think of typical artifacts in medical imaging that might be reduced by these filters?

Answer:

Thresholding is a simple and commonly used technique for image segmentation. It works by classifying pixels in an image based on their intensity values. However, in medical imaging, especially when dealing with complex structures like the brain, simple thresholding may not always yield perfect results. Here are a few reasons why thresholding might not produce an ideal segmentation:

1. **Intensity Variations:** Brain tissues can have varying intensity levels, and these variations might not be captured well by a single threshold value. Gray matter, white matter, and other structures can have **overlapping intensity ranges**, making it challenging to separate them accurately.

2. **Noise:** Medical images often contain noise due to various factors such as acquisition artifacts or sensor imperfections. Noise can cause fluctuations in pixel intensities, making it difficult to set a single threshold that effectively distinguishes between different tissues.

3. **Intensity Inhomogeneity:** Sometimes, the intensity of an object in an image can vary due to non-uniform lighting or other imaging artifacts. This intensity inhomogeneity can lead to inconsistent pixel intensities within the same tissue, making it hard to apply a global threshold.

4. **Complex Structures:** The brain contains intricate structures, and different regions might have similar intensity values. For instance, certain pathological structures or adjacent tissues can have similar intensity characteristics, making it challenging to differentiate them accurately.

5. **Partial Volume Effect:** Pixels on the boundary between different tissues (partial volume pixels) can have intensity values representing a mix of both tissues. Thresholding might misclassify these pixels as belonging to either tissue, leading to segmentation errors.

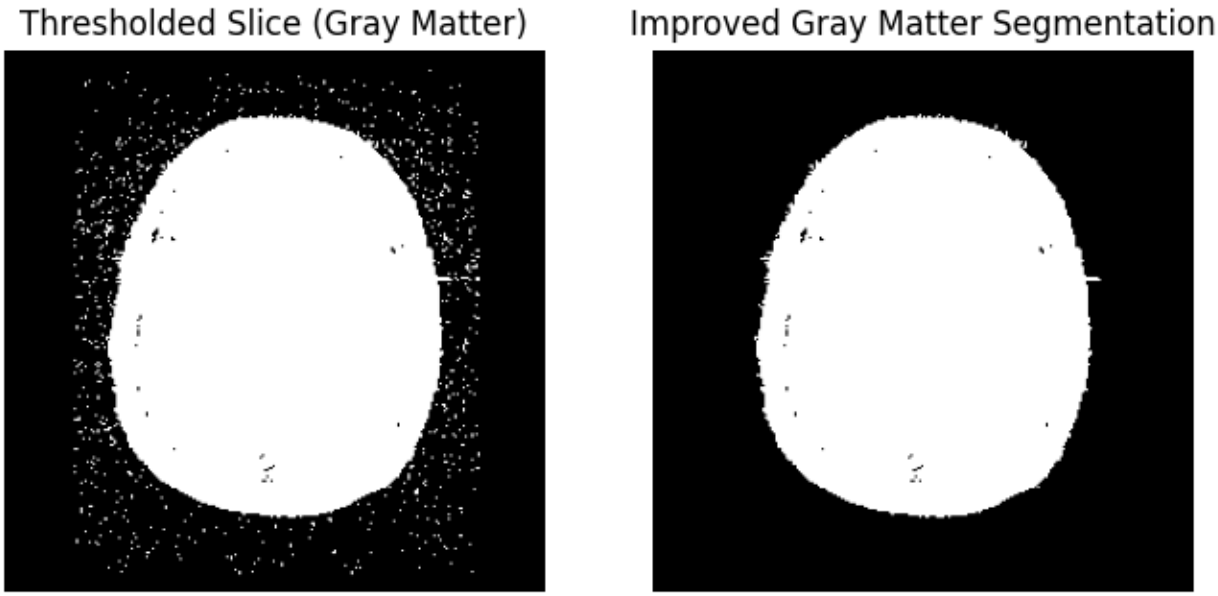
6. **Resolution Limitations:** The spatial resolution of the imaging modality might not be high enough to clearly distinguish between adjacent structures. In such cases, even a perfect segmentation algorithm would struggle to accurately delineate boundaries.

To address these challenges, more advanced segmentation techniques, such as region-based methods, edge-based methods, or machine learning approaches like deep learning, can be employed. These methods take into account spatial information, pixel connectivity, and contextual features to achieve more accurate and robust segmentations, especially in the presence of noise, intensity variations, and complex structures.

TASK 4: Optional (connected component analysis)

```
from skimage.measure import label, regionprops

# Assuming thresholded_slice is your previously thresholded binary
image
# Perform connected component analysis
label_im = label(thresholded_slice)  # Label connected components

# Get region properties
regions = regionprops(label_im)

# Sort the regions in ascending order of size
lst_sorted = sorted(regions, key=lambda x: x.area)
```

```
# Take the data from the last index from the list (largest connected component)
gray_matter_mask = label_im == lst_sorted[-1].label

# Plot the original and improved segmentations for visualization
plt.figure(figsize=(8, 4))

plt.subplot(1, 2, 1)
plt.imshow(thresholded_slice, cmap='gray')
plt.title('Thresholded Slice (Gray Matter)')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(gray_matter_mask, cmap='gray')
plt.title('Improved Gray Matter Segmentation')
plt.axis('off')

plt.show()
```



TASK 5: Morphological Operations

```
import os
import cv2 as cv
import matplotlib.pyplot as plt

# Specify the folder path containing PNG images
folder_path = '/opt/google/drive/png'  # Replace 'path_to_your_folder' with the actual folder path
```

```
# Get a list of all PNG files in the folder
png_files = [file for file in os.listdir(folder_path) if
file.endswith('.png')]

# Define structuring elements (kernels)
rectangular_kernel = cv.getStructuringElement(cv.MORPH_RECT, (5, 5))
elliptical_kernel = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5, 5))
cross_kernel = cv.getStructuringElement(cv.MORPH_CROSS, (5, 5))

# Perform morphological operations on each PNG image in the folder
for png_file in png_files:
    # Load PNG image
    img = cv.imread(os.path.join(folder_path, png_file),
cv.IMREAD_GRAYSCALE)

    # Apply morphological operations with rectangular kernel
    erosion_rectangular = cv.erode(img, rectangular_kernel,
iterations=1)
    dilation_rectangular = cv.dilate(img, rectangular_kernel,
iterations=1)
    opening_rectangular = cv.morphologyEx(img, cv.MORPH_OPEN,
rectangular_kernel)
    closing_rectangular = cv.morphologyEx(img, cv.MORPH_CLOSE,
rectangular_kernel)

    # Apply morphological operations with elliptical kernel
    erosion_elliptical = cv.erode(img, elliptical_kernel,
iterations=1)
    dilation_elliptical = cv.dilate(img, elliptical_kernel,
iterations=1)
    opening_elliptical = cv.morphologyEx(img, cv.MORPH_OPEN,
elliptical_kernel)
    closing_elliptical = cv.morphologyEx(img, cv.MORPH_CLOSE,
elliptical_kernel)

    # Apply morphological operations with cross-shaped kernel
    erosion_cross = cv.erode(img, cross_kernel, iterations=1)
    dilation_cross = cv.dilate(img, cross_kernel, iterations=1)
    opening_cross = cv.morphologyEx(img, cv.MORPH_OPEN, cross_kernel)
    closing_cross = cv.morphologyEx(img, cv.MORPH_CLOSE, cross_kernel)

    # Plot original and processed images for comparison
    plt.figure(figsize=(18, 12))

    plt.subplot(3, 5, 1), plt.imshow(img, cmap='gray')
    plt.title('Original'), plt.axis('off')

    plt.subplot(3, 5, 2), plt.imshow(erosion_rectangular, cmap='gray')
    plt.title('Erosion (Rectangular)'), plt.axis('off')
    plt.subplot(3, 5, 3), plt.imshow(dilation_rectangular,
```

```
cmap='gray')
    plt.title('Dilation (Rectangular)'), plt.axis('off')
    plt.subplot(3, 5, 4), plt.imshow(opening_rectangular, cmap='gray')
    plt.title('Opening (Rectangular)'), plt.axis('off')
    plt.subplot(3, 5, 5), plt.imshow(closing_rectangular, cmap='gray')
    plt.title('Closing (Rectangular)'), plt.axis('off')

    plt.subplot(3, 5, 7), plt.imshow(erosion_elliptical, cmap='gray')
    plt.title('Erosion (Elliptical)'), plt.axis('off')
    plt.subplot(3, 5, 8), plt.imshow(dilation_elliptical, cmap='gray')
    plt.title('Dilation (Elliptical)'), plt.axis('off')
    plt.subplot(3, 5, 9), plt.imshow(opening_elliptical, cmap='gray')
    plt.title('Opening (Elliptical)'), plt.axis('off')
    plt.subplot(3, 5, 10), plt.imshow(closing_elliptical, cmap='gray')
    plt.title('Closing (Elliptical)'), plt.axis('off')

    plt.subplot(3, 5, 12), plt.imshow(erosion_cross, cmap='gray')
    plt.title('Erosion (Cross)'), plt.axis('off')
    plt.subplot(3, 5, 13), plt.imshow(dilation_cross, cmap='gray')
    plt.title('Dilation (Cross)'), plt.axis('off')
    plt.subplot(3, 5, 14), plt.imshow(opening_cross, cmap='gray')
    plt.title('Opening (Cross)'), plt.axis('off')
    plt.subplot(3, 5, 15), plt.imshow(closing_cross, cmap='gray')
    plt.title('Closing (Cross)'), plt.axis('off')

    plt.suptitle(f'Image: {png_file}')
    plt.show()
```

Image: A_One.png

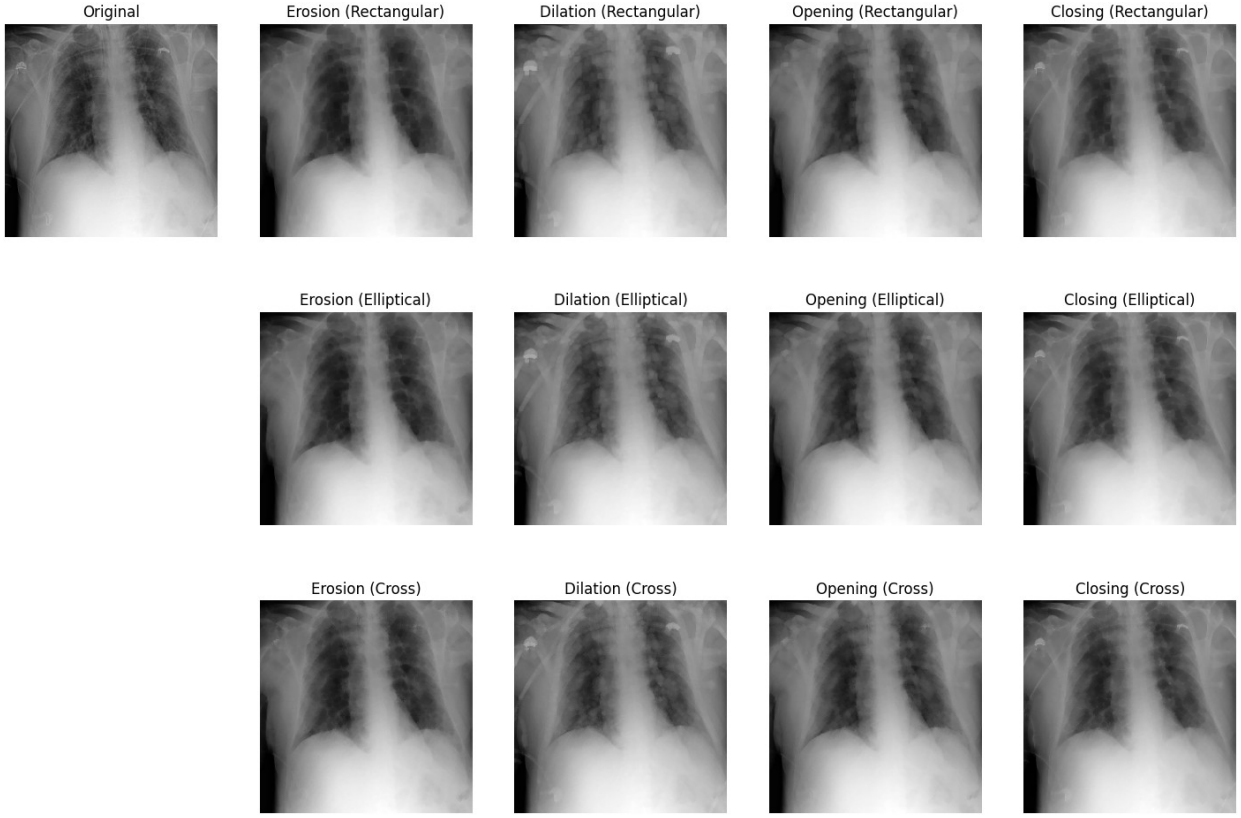


Image: B_Two.png

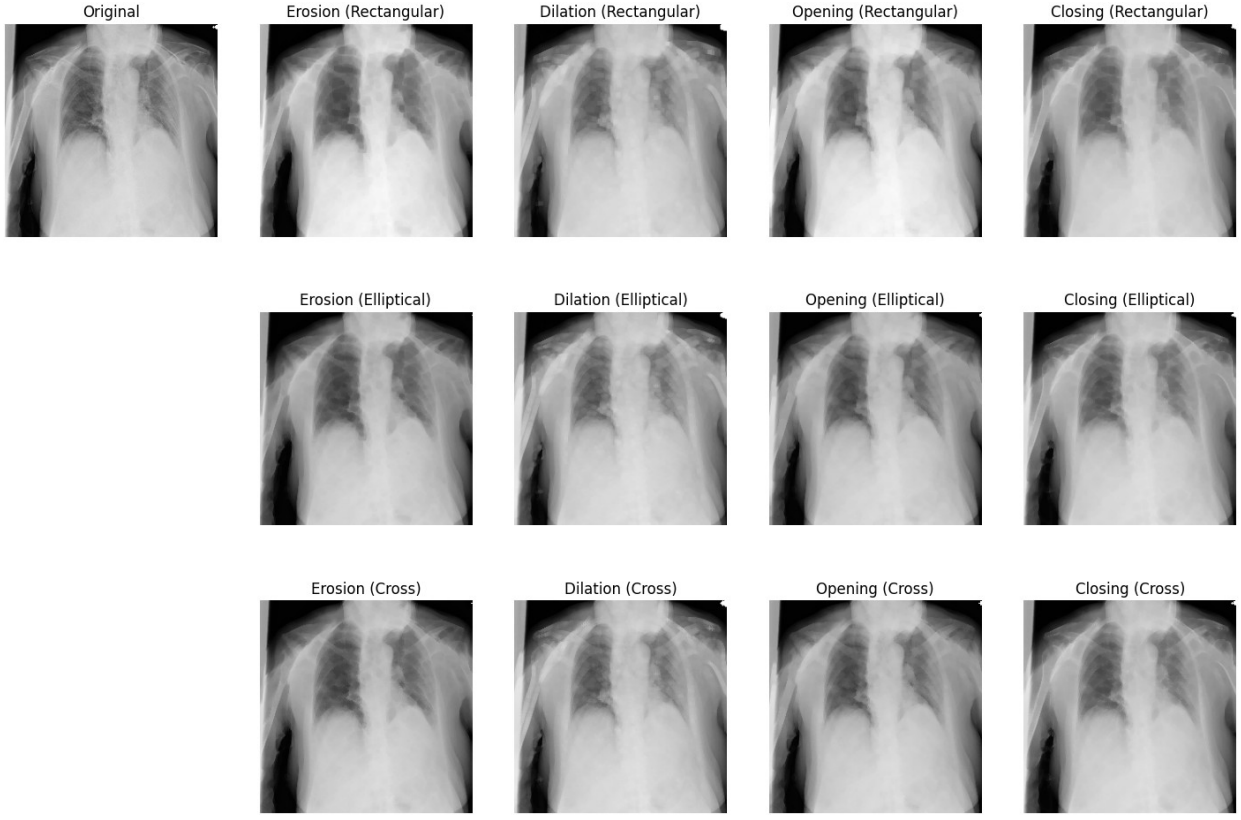


Image: E_Five.png

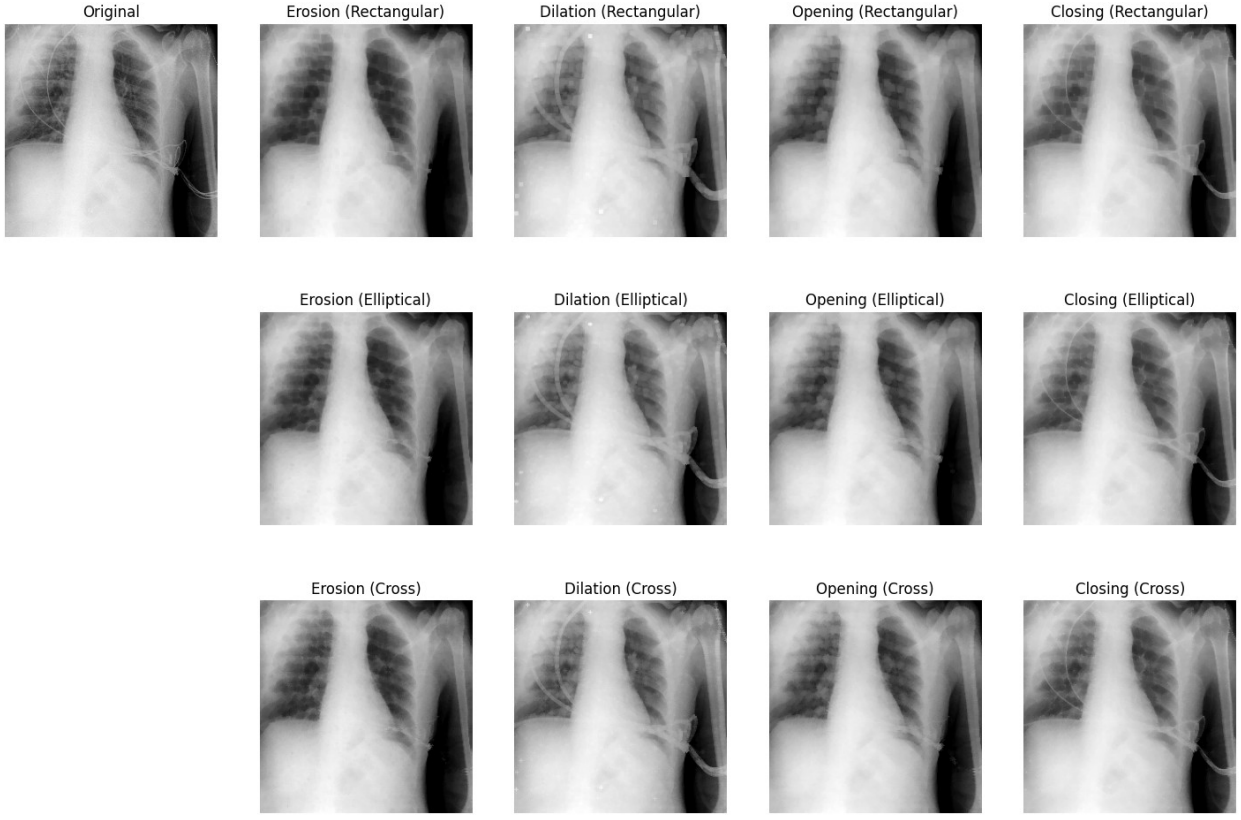


Image: D_Four.png

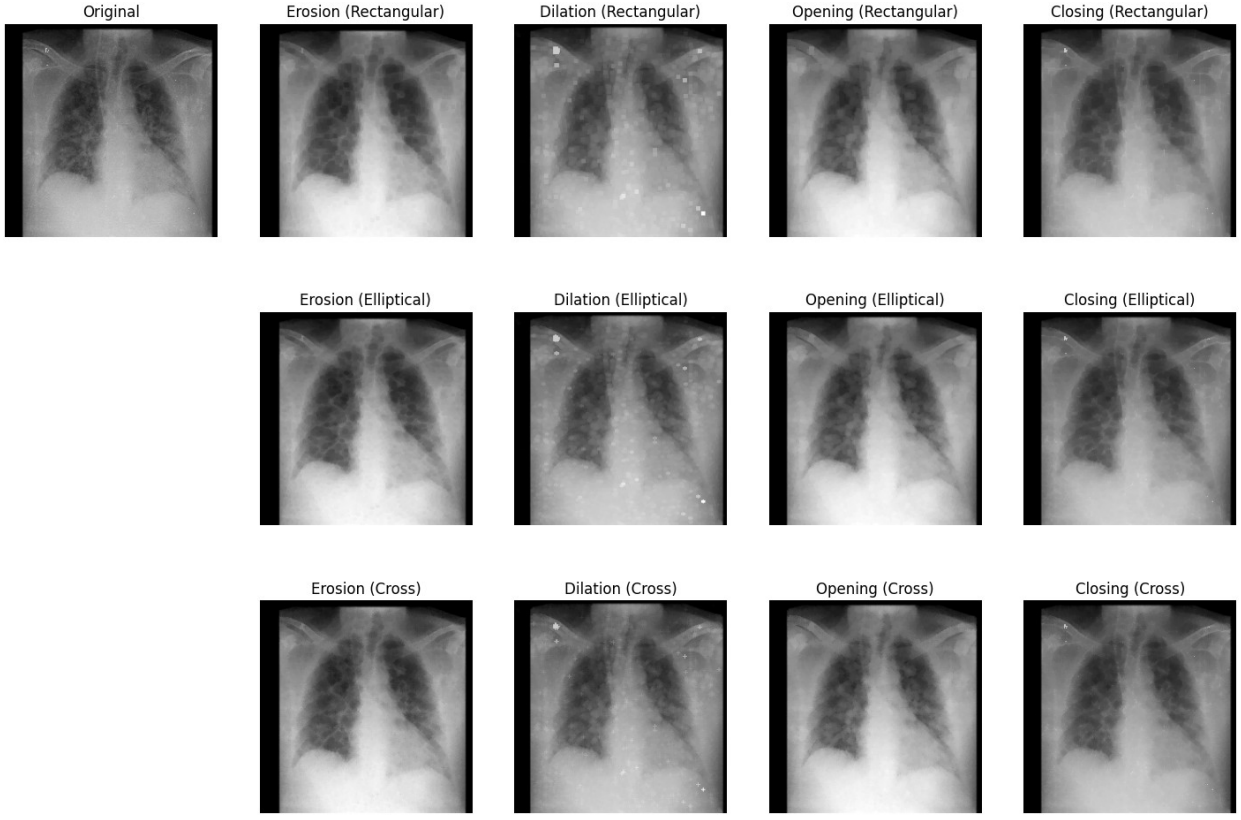
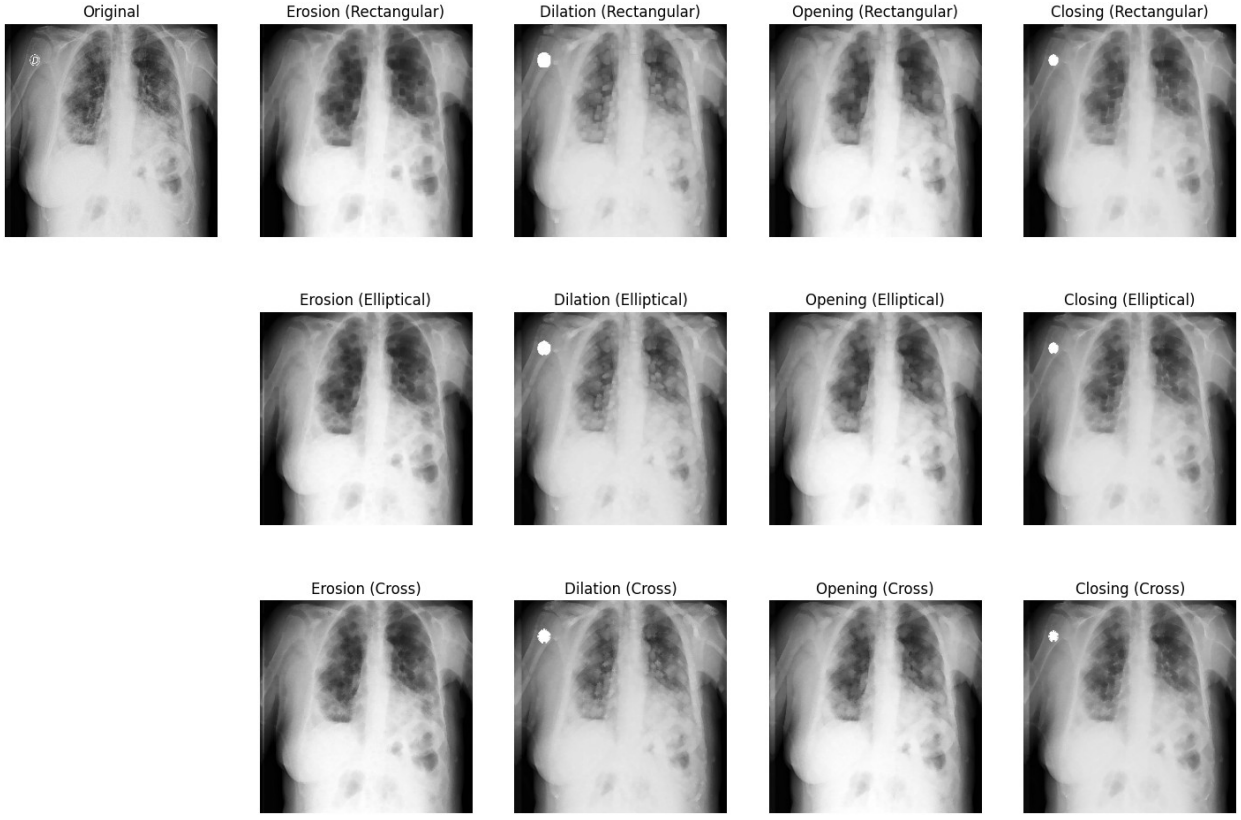


Image: C_Three.png



is23-exercise4-additional-material

December 1, 2023

1 MedVis 23 - Exercise Sheet 4 - Additional Material

---

2 Preparation

Here the necessary libraries are included.

[1]:
```
!pip install scipy
!pip install pydicom
!pip install seaborn

import seaborn as sns # for teh TF plot
from mpl_toolkits.axes_grid1 import make_axes_locatable # for teh TF plot
from matplotlib.cm import ScalarMappable # for teh TF plot
from matplotlib.colors import Normalize # for teh TF plot
import os # for file-handling
import numpy as np # maths
import matplotlib.pyplot as plt # plotting
from scipy import ndimage # image processing
from pydicom import dcmread # reading DICOM files
```

```
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages
(1.11.3)
Requirement already satisfied: numpy<1.28.0,>=1.21.6 in
/usr/local/lib/python3.10/dist-packages (from scipy) (1.23.5)
Collecting pydicom
  Downloading pydicom-2.4.3-py3-none-any.whl (1.8 MB)
                             1.8/1.8 MB
23.7 MB/s eta 0:00:00
Installing collected packages: pydicom
Successfully installed pydicom-2.4.3
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-
packages (0.12.2)
Requirement already satisfied: numpy!=1.24.0,>=1.17 in
/usr/local/lib/python3.10/dist-packages (from seaborn) (1.23.5)
Requirement already satisfied: pandas>=0.25 in /usr/local/lib/python3.10/dist-
packages (from seaborn) (1.5.3)
```

```
Requirement already satisfied: matplotlib!=3.6.1,>=3.1 in
/usr/local/lib/python3.10/dist-packages (from seaborn) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn)
(1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
packages (from matplotlib!=3.6.1,>=3.1->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn)
(4.44.3)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn)
(1.4.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn)
(23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-
packages (from matplotlib!=3.6.1,>=3.1->seaborn) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn)
(3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas>=0.25->seaborn) (2023.3.post1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.1->seaborn) (1.16.0)
```

Function for reading the MRI volume

[2]:
```python
# Read a volume and convert to numpy array
def read_dcm_volume(name):
    volume_img = []
    for file in sorted(os.listdir(name)):
        dcm_slice = dcmread(os.path.join(name, file))
        volume_img.append(dcm_slice.pixel_array)
    return np.transpose(np.array(volume_img),(1,2,0))
```

3 Task 4: Transfer Function Example

To better understand what TFs are doing, we want to look at a practical example here of a window function. The calculation of the window function and the creation of the plots are done in the two code blocks below. Don't forget to run both of them after changing the code! Also, remember to upload a dataset (e.g. dataset1) and update the file path in the code before running it.

As you already know, window functions have two main parameters: * Window Level (also called Window Center) * Window Width (also called Window Range)

Change the values for window level and window width. Which one steers brightness and which one contrast?

Instead of just mapping to gray values, transfer functions can also map the intensity values of the original image to color. Try it out by choosing another colormap instead of `gray`. You can find a list of premade colormaps here.

[4]:
```
# window function
def create_window(img, WL, WW):
    upper, lower = WL+WW/2, WL-WW/2
    X = np.clip(img.copy(), lower, upper)
    X = X - np.min(X)
    X = X / np.max(X)
    return X

# plot window function
def plot_window(x, level, width, heigth=1):
  leftBorder = level-width/2
  rightBorder = level+width/2

  y = ((x-leftBorder)/width) *heigth
  y[x<leftBorder] = 0
  y[x>rightBorder] = heigth
  print(rightBorder)
  return y


# load one slice
dcm_slice = dcmread("/opt/google/drive/dataset1/brain_001.dcm")
# convert it to numpy array for plotting the image
img = np.array(dcm_slice.pixel_array)
# normalize
img = img/np.max(img)


# window of the original image
x_original = np.arange(np.min(img), np.max(img), 0.01)
y_original = plot_window(x_original, np.max(img)/2, np.max(img))

#########################################################################
# Change the window level and window width to find out which one
# steeres brightness and which one steeres contrast.
#########################################################################
# user-steered window
window_level = 0.5
window_width = 0.5

#########################################################
```

```
# Use the transfer function to map the intensity
# values to color by changing the color map.
# These are possible color maps:
# https://matplotlib.org/stable/tutorials/colors/colormaps.html
#############################################################
colormap = 'gray'

windowed_img = create_window(img, window_level, window_width)

x = np.arange(np.min(img), np.max(img), 0.01)
ylin = plot_window(x, window_level, window_width)
```

```
1.0
0.75
```

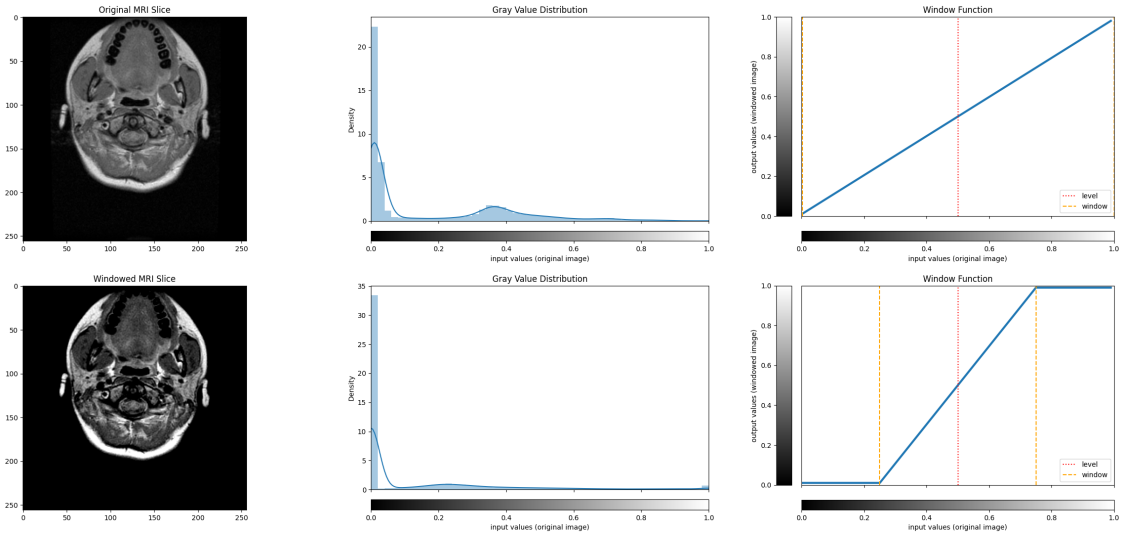
[5]:
```
#@title Run this cell to create the plots

# Create a figure (window)
fig = plt.figure(figsize=(30,20))

ax1 = fig.add_subplot(3, 3, 1)
ax1.set_title('Original MRI Slice')
ax1.imshow(img, cmap='gray')

ax2 = fig.add_subplot(3, 3, 2)
ax2.set_title('Gray Value Distribution')
sns.distplot(img.flatten(), ax=ax2);
ax2.set_xlim((np.min(img),np.max(img)))
# plot bottom colorbar
div = make_axes_locatable(ax2)
cax = div.append_axes("bottom", size="5%", pad=0.2)
clb2 = plt.colorbar(ScalarMappable(Normalize(0,1), cmap=plt.get_cmap("gray")),␣
 ↪cax=cax, use_gridspec=True, orientation="horizontal")
clb2.set_label('input values (original image)')
cax.yaxis.set_ticks_position('right')
cax.yaxis.set_label_position('right')
plt.setp(ax2.get_xticklabels(), visible=False)

ax3 = fig.add_subplot(3, 3, 3)
ax3.plot(x_original,y_original, linewidth=3)
plt.yticks([])
#plt.xticks([])
ax3.set_title('Window Function')
ax3.set_ylim((-0.01,1.01))
# plot left colorbar
div = make_axes_locatable(ax3)
cax = div.append_axes("left", size="5%", pad=0.2)
```

```
clb2 = plt.colorbar(ScalarMappable(Normalize(0,1), cmap=plt.get_cmap("gray")),␣
 ↪cax=cax, use_gridspec=True)
clb2.set_label('output values (windowed image)')
cax.yaxis.set_ticks_position('left')
cax.yaxis.set_label_position('left')
# plot bottom colorbar
cax = div.append_axes("bottom", size="5%", pad=0.3, sharex=ax3)
clb = fig.colorbar(ScalarMappable(Normalize(0,np.max(img)), cmap=plt.
 ↪get_cmap("gray")), cax=cax , orientation="horizontal")
clb.set_label('input values (original image)')
plt.setp(ax3.get_xticklabels(), visible=False)
# plot vertical lines and legend
ax3.axvline(x=np.max(img)/2, color='red', linestyle=':', label="level")
ax3.axvline(x=np.min(img)+0.003, color='orange', linestyle='--', label="window")
ax3.axvline(x=np.max(img), color='orange', linestyle='--')
ax3.legend(loc="lower right")

ax4 = fig.add_subplot(3, 3, 4)
ax4.set_title('Windowed MRI Slice')
ax4.imshow(windowed_img, cmap=colormap)

ax5 = fig.add_subplot(3, 3, 5)
ax5.set_title('Gray Value Distribution')
sns.distplot(windowed_img.flatten(), ax=ax5)
ax5.set_xlim((0,1))
# plot bottom colorbar
div = make_axes_locatable(ax5)
cax = div.append_axes("bottom", size="5%", pad=0.2)
clb2 = plt.colorbar(ScalarMappable(Normalize(0,1), cmap=plt.
 ↪get_cmap(colormap)), cax=cax, use_gridspec=True, orientation="horizontal")
clb2.set_label('input values (original image)')
cax.yaxis.set_ticks_position('right')
cax.yaxis.set_label_position('right')
plt.setp(ax5.get_xticklabels(), visible=False)

ax6 = fig.add_subplot(3, 3, 6)
ax6.plot(x,ylin, linewidth=3)
ax6.set_title('Window Function')
ax6.set_ylim((-0.01,1.01))
ax6.set_yticks([])
#ax6.set_xticks([])
# plot left colorbar
div = make_axes_locatable(ax6)
cax = div.append_axes("left", size="5%", pad=0.2)
clb2 = plt.colorbar(ScalarMappable(Normalize(0,1), cmap=plt.
 ↪get_cmap(colormap)), cax=cax, use_gridspec=True)
clb2.set_label('output values (windowed image)')
```

```
cax.yaxis.set_ticks_position('left')
cax.yaxis.set_label_position('left')
# plot bottom colorbar
cax = div.append_axes("bottom", size="5%", pad=0.3, sharex=ax6)
clb = fig.colorbar(ScalarMappable(Normalize(0,np.max(img)), cmap=plt.
 ↪get_cmap("gray")), cax=cax , orientation="horizontal")
clb.set_label('input values (original image)')
plt.setp(ax6.get_xticklabels(), visible=False)
# plot vertical lines and legend
ax6.axvline(x=window_level, color='red', linestyle=':', label="level")
ax6.axvline(x=window_level-window_width/2, color='orange', linestyle='--',␣
 ↪label="window")
ax6.axvline(x=window_level+window_width/2, color='orange', linestyle='--')
ax6.legend(loc="lower right")
```

```
<ipython-input-5-721b8110f270>:12: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(img.flatten(), ax=ax2);
<ipython-input-5-721b8110f270>:53: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(windowed_img.flatten(), ax=ax5)
```

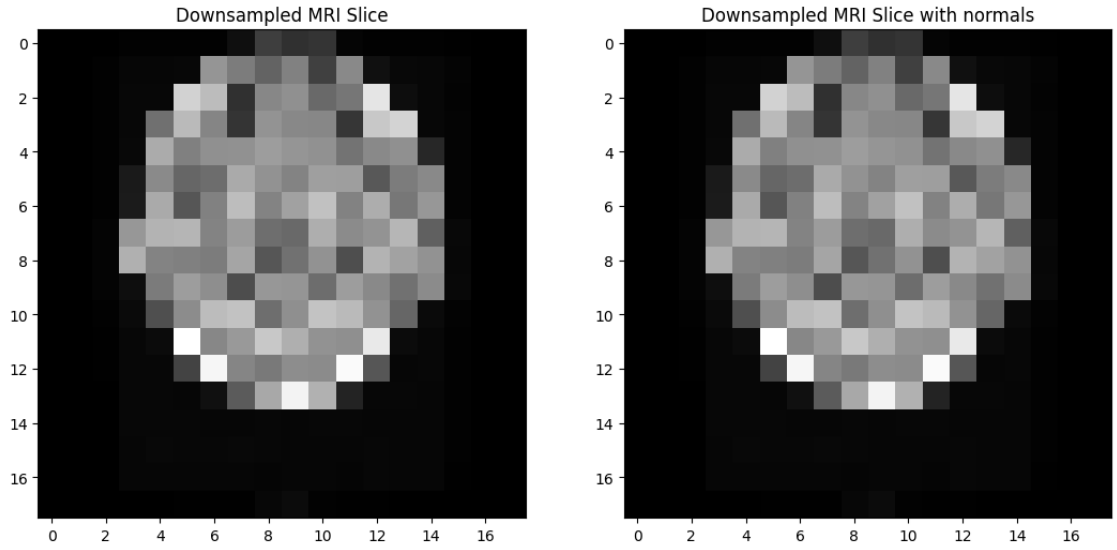
[5]: <matplotlib.legend.Legend at 0x795b753489a0>



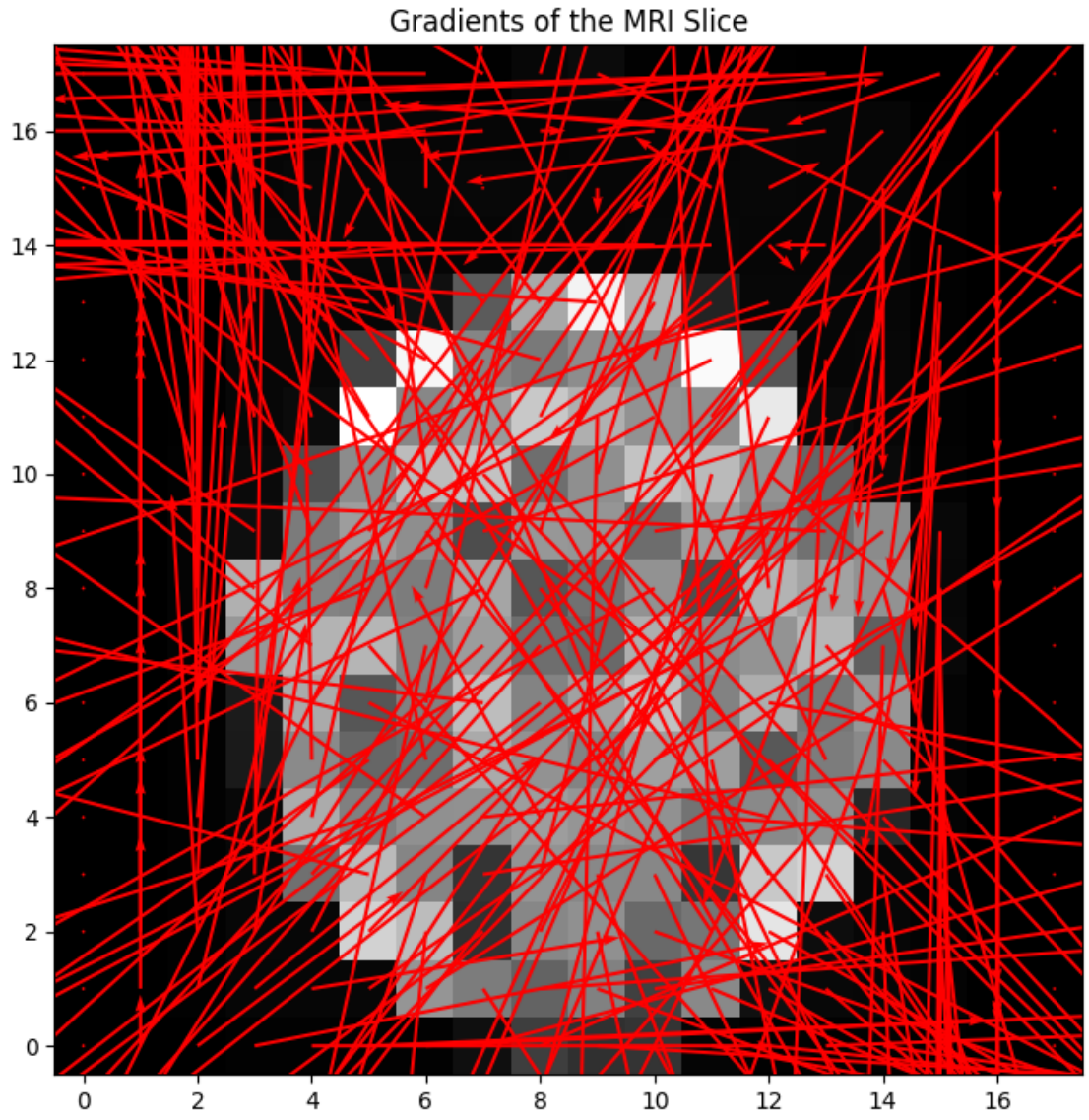
4 Task 5

[7]:
```
dcm_slice = dcmread("/opt/google/drive/dataset1/brain_001.dcm")
sampling_freq = 15

img = np.array(dcm_slice.pixel_array)
smoothed_img = ndimage.gaussian_filter(img, sigma=3)
downsampled_img = smoothed_img[::sampling_freq,::sampling_freq]


##################################################
# Calculate the normals of the down-samled image
##################################################


# Create a figure (window)
fig = plt.figure(figsize=(20,10))

ax1 = fig.add_subplot(1, 3, 1)
ax1.set_title('Downsampled MRI Slice')
ax1.imshow(downsampled_img, cmap='gray')

ax2 = fig.add_subplot(1, 3, 2)
ax2.set_title('Downsampled MRI Slice with normals')
ax2.imshow(downsampled_img, cmap='gray')
# add normals to the plot here, e.g. by using plt.quiver()
```

[7]:
```
<matplotlib.image.AxesImage at 0x795b73a4a7a0>
```



[8]:
```python
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt

# Simplified image from the Google Colab Notebook
# Use the downsampled_img as the input image
img = downsampled_img

# Compute gradients using np.gradient
gradient_x, gradient_y = np.gradient(img)

# Create a meshgrid for quiver plot
x, y = np.meshgrid(np.arange(img.shape[1]), np.arange(img.shape[0]))

# Plot the image
plt.figure(figsize=(10, 8))
plt.imshow(img, cmap='gray', origin='lower')

# Plot quiver plot for gradients
plt.quiver(x, y, gradient_x, gradient_y, color='r', scale=20)

plt.title('Gradients of the MRI Slice')
plt.show()
```


Gradients of the MRI Slice
0
2
4
6
8
10
12
14
16

Open3D Visualize in Google Colab

This notebook will think about how to visualize Open3D in Google Colab.

Install Open3D

```
!pip install open3d
```

```
Collecting open3d
  Downloading open3d-0.17.0-cp310-cp310-manylinux_2_27_x86_64.whl
(420.5 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 420.5/420.5 MB 1.9 MB/s eta
0:00:00
ent already satisfied: numpy>=1.18.0 in
/usr/local/lib/python3.10/dist-packages (from open3d) (1.23.5)
Collecting dash>=2.6.0 (from open3d)
  Downloading dash-2.14.1-py3-none-any.whl (10.4 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.4/10.4 MB 77.3 MB/s eta
0:00:00
ent already satisfied: werkzeug>=2.2.3 in
/usr/local/lib/python3.10/dist-packages (from open3d) (3.0.1)
Collecting nbformat==5.7.0 (from open3d)
  Downloading nbformat-5.7.0-py3-none-any.whl (77 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 77.1/77.1 kB 6.9 MB/s eta
0:00:00
 open3d)
  Downloading ConfigArgParse-1.7-py3-none-any.whl (25 kB)
Collecting ipywidgets>=8.0.4 (from open3d)
  Downloading ipywidgets-8.1.1-py3-none-any.whl (139 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 139.4/139.4 kB 16.2 MB/s eta
0:00:00
 open3d)
  Downloading addict-2.4.0-py3-none-any.whl (3.8 kB)
Requirement already satisfied: pillow>=9.3.0 in
/usr/local/lib/python3.10/dist-packages (from open3d) (9.4.0)
Requirement already satisfied: matplotlib>=3 in
/usr/local/lib/python3.10/dist-packages (from open3d) (3.7.1)
Requirement already satisfied: pandas>=1.0 in
/usr/local/lib/python3.10/dist-packages (from open3d) (1.5.3)
Requirement already satisfied: pyyaml>=5.4.1 in
/usr/local/lib/python3.10/dist-packages (from open3d) (6.0.1)
Requirement already satisfied: scikit-learn>=0.21 in
/usr/local/lib/python3.10/dist-packages (from open3d) (1.2.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-
packages (from open3d) (4.66.1)
Collecting pyquaternion (from open3d)
  Downloading pyquaternion-0.9.9-py3-none-any.whl (14 kB)
```

```
Requirement already satisfied: fastjsonschema in
/usr/local/lib/python3.10/dist-packages (from nbformat==5.7.0->open3d)
(2.19.0)
Requirement already satisfied: jsonschema>=2.6 in
/usr/local/lib/python3.10/dist-packages (from nbformat==5.7.0->open3d)
(4.19.2)
Requirement already satisfied: jupyter-core in
/usr/local/lib/python3.10/dist-packages (from nbformat==5.7.0->open3d)
(5.5.0)
Requirement already satisfied: traitlets>=5.1 in
/usr/local/lib/python3.10/dist-packages (from nbformat==5.7.0->open3d)
(5.7.1)
Requirement already satisfied: Flask<3.1,>=1.0.4 in
/usr/local/lib/python3.10/dist-packages (from dash>=2.6.0->open3d)
(2.2.5)
Requirement already satisfied: plotly>=5.0.0 in
/usr/local/lib/python3.10/dist-packages (from dash>=2.6.0->open3d)
(5.15.0)
Collecting dash-html-components==2.0.0 (from dash>=2.6.0->open3d)
  Downloading dash_html_components-2.0.0-py3-none-any.whl (4.1 kB)
Collecting dash-core-components==2.0.0 (from dash>=2.6.0->open3d)
  Downloading dash_core_components-2.0.0-py3-none-any.whl (3.8 kB)
Collecting dash-table==5.0.0 (from dash>=2.6.0->open3d)
  Downloading dash_table-5.0.0-py3-none-any.whl (3.9 kB)
Requirement already satisfied: typing-extensions>=4.1.1 in
/usr/local/lib/python3.10/dist-packages (from dash>=2.6.0->open3d)
(4.5.0)
Requirement already satisfied: requests in
/usr/local/lib/python3.10/dist-packages (from dash>=2.6.0->open3d)
(2.31.0)
Collecting retrying (from dash>=2.6.0->open3d)
  Downloading retrying-1.3.4-py3-none-any.whl (11 kB)
Collecting ansi2html (from dash>=2.6.0->open3d)
  Downloading ansi2html-1.8.0-py3-none-any.whl (16 kB)
Requirement already satisfied: nest-asyncio in
/usr/local/lib/python3.10/dist-packages (from dash>=2.6.0->open3d)
(1.5.8)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.10/dist-packages (from dash>=2.6.0->open3d)
(67.7.2)
Requirement already satisfied: importlib-metadata in
/usr/local/lib/python3.10/dist-packages (from dash>=2.6.0->open3d)
(6.8.0)
Collecting comm>=0.1.3 (from ipywidgets>=8.0.4->open3d)
  Downloading comm-0.2.0-py3-none-any.whl (7.0 kB)
Requirement already satisfied: ipython>=6.1.0 in
/usr/local/lib/python3.10/dist-packages (from ipywidgets>=8.0.4-
>open3d) (7.34.0)
Collecting widgetsnbextension~=4.0.9 (from ipywidgets>=8.0.4->open3d)
```

```
  Downloading widgetsnbextension-4.0.9-py3-none-any.whl (2.3 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.3/2.3 MB 73.5 MB/s eta
0:00:00
ent already satisfied: jupyterlab-widgets~=3.0.9 in
/usr/local/lib/python3.10/dist-packages (from ipywidgets>=8.0.4-
>open3d) (3.0.9)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3->open3d)
(1.2.0)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3->open3d)
(0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3->open3d)
(4.44.3)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3->open3d)
(1.4.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3->open3d)
(23.2)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3->open3d)
(3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3->open3d)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.0->open3d)
(2023.3.post1)
Requirement already satisfied: scipy>=1.3.2 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.21-
>open3d) (1.11.3)
Requirement already satisfied: joblib>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.21-
>open3d) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.21-
>open3d) (3.2.0)
Requirement already satisfied: MarkupSafe>=2.1.1 in
/usr/local/lib/python3.10/dist-packages (from werkzeug>=2.2.3->open3d)
(2.1.3)
Requirement already satisfied: Jinja2>=3.0 in
/usr/local/lib/python3.10/dist-packages (from Flask<3.1,>=1.0.4-
>dash>=2.6.0->open3d) (3.1.2)
Requirement already satisfied: itsdangerous>=2.0 in
/usr/local/lib/python3.10/dist-packages (from Flask<3.1,>=1.0.4-
>dash>=2.6.0->open3d) (2.1.2)
Requirement already satisfied: click>=8.0 in
```

```
/usr/local/lib/python3.10/dist-packages (from Flask<3.1,>=1.0.4-
>dash>=2.6.0->open3d) (8.1.7)
Collecting jedi>=0.16 (from ipython>=6.1.0->ipywidgets>=8.0.4->open3d)
  Downloading jedi-0.19.1-py2.py3-none-any.whl (1.6 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 72.0 MB/s eta
0:00:00
ent already satisfied: decorator in /usr/local/lib/python3.10/dist-
packages (from ipython>=6.1.0->ipywidgets>=8.0.4->open3d) (4.4.2)
Requirement already satisfied: pickleshare in
/usr/local/lib/python3.10/dist-packages (from ipython>=6.1.0-
>ipywidgets>=8.0.4->open3d) (0.7.5)
Requirement already satisfied: prompt-toolkit!=3.0.0,!
=3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from
ipython>=6.1.0->ipywidgets>=8.0.4->open3d) (3.0.41)
Requirement already satisfied: pygments in
/usr/local/lib/python3.10/dist-packages (from ipython>=6.1.0-
>ipywidgets>=8.0.4->open3d) (2.16.1)
Requirement already satisfied: backcall in
/usr/local/lib/python3.10/dist-packages (from ipython>=6.1.0-
>ipywidgets>=8.0.4->open3d) (0.2.0)
Requirement already satisfied: matplotlib-inline in
/usr/local/lib/python3.10/dist-packages (from ipython>=6.1.0-
>ipywidgets>=8.0.4->open3d) (0.1.6)
Requirement already satisfied: pexpect>4.3 in
/usr/local/lib/python3.10/dist-packages (from ipython>=6.1.0-
>ipywidgets>=8.0.4->open3d) (4.8.0)
Requirement already satisfied: attrs>=22.2.0 in
/usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6-
>nbformat==5.7.0->open3d) (23.1.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in
/usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6-
>nbformat==5.7.0->open3d) (2023.11.1)
Requirement already satisfied: referencing>=0.28.4 in
/usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6-
>nbformat==5.7.0->open3d) (0.31.0)
Requirement already satisfied: rpds-py>=0.7.1 in
/usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6-
>nbformat==5.7.0->open3d) (0.13.0)
Requirement already satisfied: tenacity>=6.2.0 in
/usr/local/lib/python3.10/dist-packages (from plotly>=5.0.0-
>dash>=2.6.0->open3d) (8.2.3)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7-
>matplotlib>=3->open3d) (1.16.0)
Requirement already satisfied: zipp>=0.5 in
/usr/local/lib/python3.10/dist-packages (from importlib-metadata-
>dash>=2.6.0->open3d) (3.17.0)
Requirement already satisfied: platformdirs>=2.5 in
/usr/local/lib/python3.10/dist-packages (from jupyter-core-
```

```
>nbformat==5.7.0->open3d) (4.0.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->dash>=2.6.0-
>open3d) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests->dash>=2.6.0-
>open3d) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->dash>=2.6.0-
>open3d) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->dash>=2.6.0-
>open3d) (2023.7.22)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in
/usr/local/lib/python3.10/dist-packages (from jedi>=0.16-
>ipython>=6.1.0->ipywidgets>=8.0.4->open3d) (0.8.3)
Requirement already satisfied: ptyprocess>=0.5 in
/usr/local/lib/python3.10/dist-packages (from pexpect>4.3-
>ipython>=6.1.0->ipywidgets>=8.0.4->open3d) (0.7.0)
Requirement already satisfied: wcwidth in
/usr/local/lib/python3.10/dist-packages (from prompt-toolkit!=3.0.0,!
=3.0.1,<3.1.0,>=2.0.0->ipython>=6.1.0->ipywidgets>=8.0.4->open3d)
(0.2.10)
Installing collected packages: dash-table, dash-html-components, dash-
core-components, addict, widgetsnbextension, retrying, pyquaternion,
jedi, configargparse, comm, ansi2html, ipywidgets, dash, nbformat,
open3d
  Attempting uninstall: widgetsnbextension
    Found existing installation: widgetsnbextension 3.6.6
    Uninstalling widgetsnbextension-3.6.6:
      Successfully uninstalled widgetsnbextension-3.6.6
  Attempting uninstall: ipywidgets
    Found existing installation: ipywidgets 7.7.1
    Uninstalling ipywidgets-7.7.1:
      Successfully uninstalled ipywidgets-7.7.1
  Attempting uninstall: nbformat
    Found existing installation: nbformat 5.9.2
    Uninstalling nbformat-5.9.2:
      Successfully uninstalled nbformat-5.9.2
Successfully installed addict-2.4.0 ansi2html-1.8.0 comm-0.2.0
configargparse-1.7 dash-2.14.1 dash-core-components-2.0.0 dash-html-
components-2.0.0 dash-table-5.0.0 ipywidgets-8.1.1 jedi-0.19.1
nbformat-5.7.0 open3d-0.17.0 pyquaternion-0.9.9 retrying-1.3.4
widgetsnbextension-4.0.9
```

Import Open3D and Numpy

```
import numpy as np
import open3d as o3d
```

Read Point Cloud

```
!wget
https://raw.githubusercontent.com/PointCloudLibrary/pcl/master/test/
bunny.pcd
```

```
--2023-11-23 14:13:00--
https://raw.githubusercontent.com/PointCloudLibrary/pcl/master/test/
bunny.pcd
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|
185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10878 (11K) [text/plain]
Saving to: ‘bunny.pcd’

 bunny.pcd             0%[                    ]       0  --.-KB/s
bunny.pcd           100%[===================>]  10.62K  --.-KB/s    in
0s

2023-11-23 14:13:00 (78.9 MB/s) - ‘bunny.pcd’ saved [10878/10878]
```

```
cloud = o3d.io.read_point_cloud("bunny.pcd")
if cloud.is_empty(): exit()
```

Estimate Normals

```
cloud.estimate_normals(search_param=o3d.geometry.KDTreeSearchParamHybr
id(radius=0.1, max_nn=30))
```

Visualize using o3d.visualization.draw_geometries()

o3d.visualization.draw_geometries() is work fine outside Jupyter.
But, it does not work inside Jupyter and Goolge Colab.

```
#o3d.visualization.draw_geometries([cloud])
```

Visualize using o3d.visualization.Visualizer()

o3d.visualization.Visualizer() is work fine outside Jupyter.
But, it does not work inside Jupyter and Goolge Colab.

```
#visualizer = o3d.visualization.Visualizer()
#visualizer.create_window(window_name="point cloud", width=1280,
height=720, visible=False)
#visualizer.add_geometry(cloud)
#visualizer.run()
```

Visualize using JVisualizer()

Opne3D added experimental support for Jupyter visualization with WebGL. JVisualizer() is work fine inside Jupyter at local.
It is required with Open3D with feature enabled. It does not work in Goolge Colab.

```
# from open3d import JVisualizer

#visualizer = JVisualizer()
#visualizer.add_geometry(cloud)
#visualizer.show()
```

Visualize using o3d.visualization.Visualizer() with IPython.display

o3d.visualization.Visualizer() with IPython.display is work fine inside Jupyter. It displays image that captured viewer launched outside.
But, it does not work in Goolge Colab.

```
#import PIL.Image
#from IPython.display import display

#visualizer = o3d.visualization.Visualizer()
#visualizer.create_window(window_name="point cloud", width=1280,
height=720, visible=False)
#visualizer.add_geometry(cloud)
#visualizer.update_geometry(cloud)
#visualizer.poll_events()
#image = visualizer.capture_screen_float_buffer()
#visualizer.destroy_window()
#image = (255.0 * np.asarray(image)).astype(np.uint8)
#display(PIL.Image.fromarray(image, "RGB"))
```

Visualize using Matplotlib

In this section, it will visualize point cloud as a scatter plot using matplotlib.
It works fine in inside Jupyter and Goolge Colab. But, it is image.

```
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d

points = np.asarray(cloud.points)

colors = None
if cloud.has_colors():
    colors = np.asarray(cloud.colors)
elif cloud.has_normals():
    colors = (0.5, 0.5, 0.5) + np.asarray(cloud.normals) * 0.5
```

```
else:
    geometry.paint_uniform_color((1.0, 0.0, 0.0))
    colors = np.asarray(geometry.colors)

# Single View
ax = plt.axes(projection='3d')
ax.view_init(90, -90)
ax.axis("off")
ax.scatter(points[:,0], points[:,1], points[:,2], s=1, c=colors)
plt.show()
```

```
# Multi View
figsize = plt.rcParams.get('figure.figsize')
fig = plt.figure(figsize=(figsize[0] * 2, figsize[1]))
ax1 = fig.add_subplot(1, 2, 1, projection = '3d')
ax2 = fig.add_subplot(1, 2, 2, projection = '3d')
ax1.axis("off")
ax1.view_init(90, -90) # front view
ax1.scatter(points[:,0], points[:,1], points[:,2], s=1, c=colors)
ax2.axis("off")
ax2.view_init(90 + 90, -90) # top view
ax2.scatter(points[:,0], points[:,1], points[:,2], s=1, c=colors)
plt.show()
```

Visualize with Plotly

In this section, it will visualize point cloud as a scatter plot using plotly. It is able to move viewpoint using mouse.
It works fine in inside Jupyter and Goolge Colab. :thumbsup:

```
import plotly.graph_objects as go

fig = go.Figure(
    data=[
        go.Scatter3d(
            x=points[:,0], y=points[:,1], z=points[:,2],
            mode='markers',
            marker=dict(size=1, color=colors)
        )
    ],
    layout=dict(
        scene=dict(
            xaxis=dict(visible=False),
            yaxis=dict(visible=False),
            zaxis=dict(visible=False)
        )
    )
)
fig.show()
```

As well as, You can visualize mesh using plotly in Google Colab.

```
!wget https://graphics.stanford.edu/~mdfisher/Data/Meshes/bunny.obj
```

```
--2023-11-23 14:14:04--
https://graphics.stanford.edu/~mdfisher/Data/Meshes/bunny.obj
Resolving graphics.stanford.edu (graphics.stanford.edu)...
```

```
171.67.77.70
Connecting to graphics.stanford.edu (graphics.stanford.edu)|
171.67.77.70|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 205917 (201K) [text/plain]
Saving to: ‘bunny.obj’

bunny.obj           100%[===================>] 201.09K   361KB/s    in
0.6s

2023-11-23 14:14:05 (361 KB/s) - ‘bunny.obj’ saved [205917/205917]
```

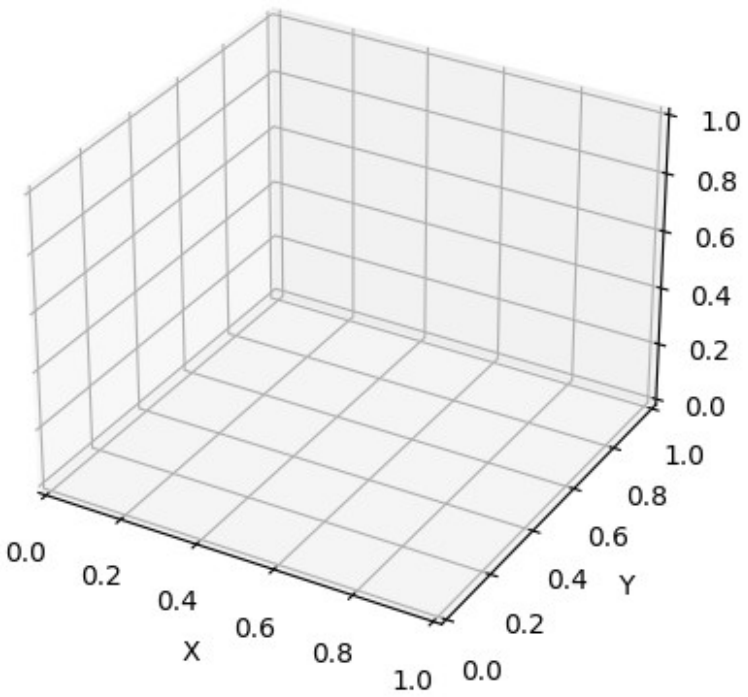
```
mesh = o3d.io.read_triangle_mesh("bunny.obj")
if mesh.is_empty(): exit()

if not mesh.has_vertex_normals(): mesh.compute_vertex_normals()
if not mesh.has_triangle_normals(): mesh.compute_triangle_normals()

triangles = np.asarray(mesh.triangles)
vertices = np.asarray(mesh.vertices)
colors = None
if mesh.has_triangle_normals():
    colors = (0.5, 0.5, 0.5) + np.asarray(mesh.triangle_normals) * 0.5
    colors = tuple(map(tuple, colors))
else:
    colors = (1.0, 0.0, 0.0)

fig = go.Figure(
    data=[
        go.Mesh3d(
            x=vertices[:,0],
            y=vertices[:,1],
            z=vertices[:,2],
            i=triangles[:,0],
            j=triangles[:,1],
            k=triangles[:,2],
            facecolor=colors,
            opacity=0.50)
    ],
    layout=dict(
        scene=dict(
            xaxis=dict(visible=False),
            yaxis=dict(visible=False),
            zaxis=dict(visible=False)
        )
    )
)
fig.show()
```

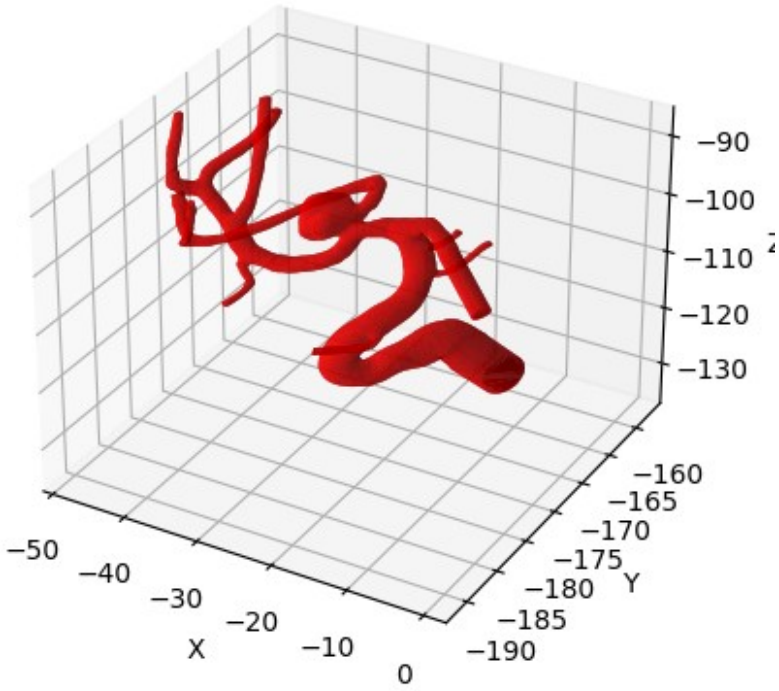
\\You can use Plotly like o3d.visualization.draw_geometries() for visualize Open3D geometries.

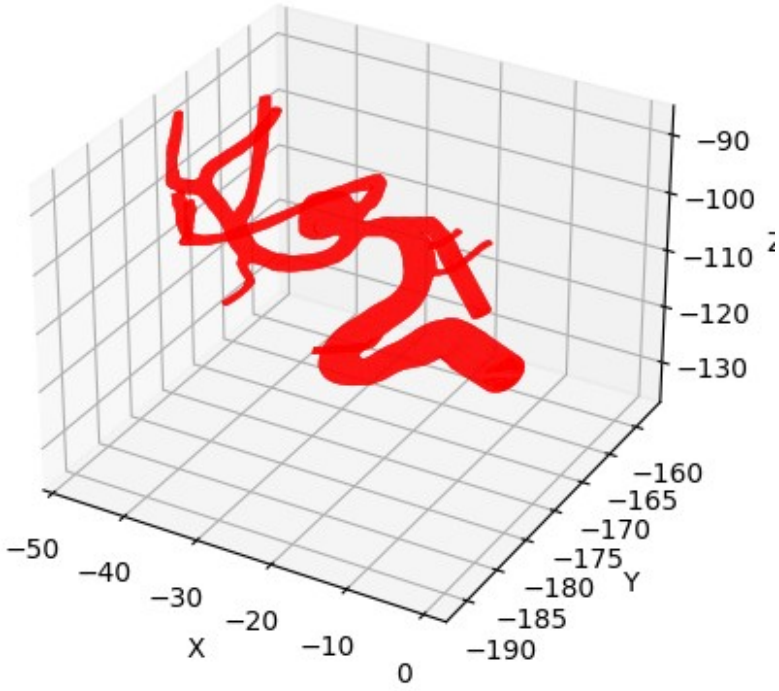
```
def draw_geometries(geometries):
    graph_objects = []

    for geometry in geometries:
        geometry_type = geometry.get_geometry_type()

        if geometry_type == o3d.geometry.Geometry.Type.PointCloud:
            points = np.asarray(geometry.points)
            colors = None
            if geometry.has_colors():
                colors = np.asarray(geometry.colors)
            elif geometry.has_normals():
                colors = (0.5, 0.5, 0.5) +
np.asarray(geometry.normals) * 0.5
            else:
                geometry.paint_uniform_color((1.0, 0.0, 0.0))
                colors = np.asarray(geometry.colors)

            scatter_3d = go.Scatter3d(x=points[:,0], y=points[:,1],
z=points[:,2], mode='markers', marker=dict(size=1, color=colors))
            graph_objects.append(scatter_3d)

        if geometry_type == o3d.geometry.Geometry.Type.TriangleMesh:
            triangles = np.asarray(geometry.triangles)
            vertices = np.asarray(geometry.vertices)
            colors = None
            if geometry.has_triangle_normals():
                colors = (0.5, 0.5, 0.5) +
np.asarray(geometry.triangle_normals) * 0.5
                colors = tuple(map(tuple, colors))
            else:
                colors = (1.0, 0.0, 0.0)

            mesh_3d = go.Mesh3d(x=vertices[:,0], y=vertices[:,1],
z=vertices[:,2], i=triangles[:,0], j=triangles[:,1], k=triangles[:,2],
facecolor=colors, opacity=0.50)
            graph_objects.append(mesh_3d)

    fig = go.Figure(
        data=graph_objects,
        layout=dict(
            scene=dict(
                xaxis=dict(visible=False),
                yaxis=dict(visible=False),
                zaxis=dict(visible=False)
            )
        )
    )
    fig.show()
```

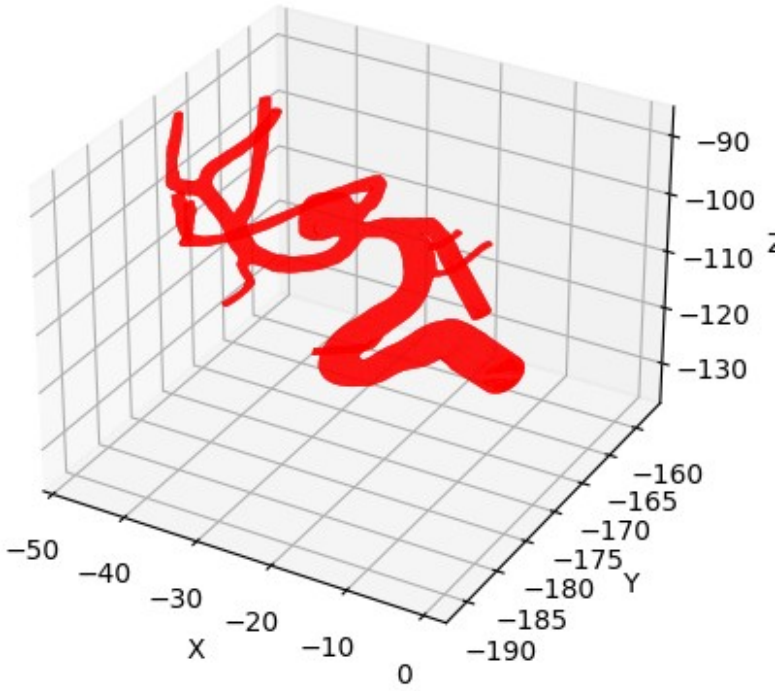
```
o3d.visualization.draw_geometries = draw_geometries # replace function
o3d.visualization.draw_geometries([cloud])
o3d.visualization.draw_geometries([mesh])

from mpl_toolkits.mplot3d import Axes3D

def plot_trisurf_with_shading(vertices, triangles, colors,
shading_method=None):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    if shading_method == 'flat':
        ax.plot_trisurf(vertices[:, 0], vertices[:, 1], vertices[:,
2], triangles=triangles, color=colors, shade=True)
    elif shading_method == 'gouraud':
        ax.plot_trisurf(vertices[:, 0], vertices[:, 1], vertices[:,
2], triangles=triangles, color=colors, shade=False)
    elif shading_method == 'phong':
        ax.plot_trisurf(vertices[:, 0], vertices[:, 1], vertices[:,
2], triangles=triangles, color=colors, shade=False)

    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    plt.show()

# Load the Aneurism object file in Google Colab
# Note: Make sure to upload the "Aneurism.obj" file to your Colab
environment first
file_path = '/opt/google/drive/Aneurism.obj'  # Adjust the path
accordingly

# Read the triangle mesh from the object file
aneurism_mesh = o3d.io.read_triangle_mesh(file_path)
if aneurism_mesh.is_empty():
    exit()

# Extract triangles, vertices, and colors from the mesh
triangles = np.asarray(aneurism_mesh.triangles)
vertices = np.asarray(aneurism_mesh.vertices)

# Check if triangle normals are available
if aneurism_mesh.has_triangle_normals():
    # Use triangle normals for coloring
    colors = (0.5, 0.5, 0.5) +
np.asarray(aneurism_mesh.triangle_normals) * 0.5
else:
    # If triangle normals are not available, use a default color
    colors = (1.0, 0.0, 0.0)

# Visualize the Aneurism mesh with different shading methods using
```

```
Matplotlib
shading_methods = [None, 'flat', 'gouraud', 'phong']

for shading_method in shading_methods:
    plot_trisurf_with_shading(vertices, triangles, colors,
shading_method)
```




−90
−100
−110
−120
−130
−160
−165
−170
−175
−180
−185
−190
Y
−50
−40
−30
−20
−10
0
X


−90
−100
−110
−120
−130
−160
−165
−170
−175
−180
−185
−190
Y
−50
−40
−30
−20
−10
0
X




```
def visualize_mesh_with_shading(mesh, shading=None):
    if mesh.is_empty():
        print("Mesh is empty.")
        return

    # Ensure vertex normals are computed
    if not mesh.has_vertex_normals():
        mesh.compute_vertex_normals()

    # Ensure triangle normals are computed
    if not mesh.has_triangle_normals():
        mesh.compute_triangle_normals()

    # Extract triangles, vertices, and colors from the mesh
    triangles = np.asarray(mesh.triangles)
    vertices = np.asarray(mesh.vertices)

    # Check if triangle normals are available
    if mesh.has_triangle_normals():
        # Use triangle normals for coloring
        colors = (0.5, 0.5, 0.5) + np.asarray(mesh.triangle_normals) *
0.5
    else:
        # If triangle normals are not available, use a default color
        colors = (1.0, 0.0, 0.0)
```

```
    # Apply shading based on the specified shading method
    if shading == 'flat':
        mesh.compute_triangle_normals()
        colors = (0.5, 0.5, 0.5) + np.asarray(mesh.triangle_normals) *
0.5
    elif shading == 'gouraud' or shading == 'phong':
        mesh.compute_vertex_normals()
        colors = (0.5, 0.5, 0.5) + np.asarray(mesh.vertex_normals) *
0.5

    # Create a Plotly Figure with a Mesh3d trace
    fig = go.Figure(
        data=[
            go.Mesh3d(
                x=vertices[:, 0],
                y=vertices[:, 1],
                z=vertices[:, 2],
                i=triangles[:, 0],
                j=triangles[:, 1],
                k=triangles[:, 2],
                facecolor=tuple(map(tuple, colors)),
                opacity=0.50)
        ],
        layout=dict(
            scene=dict(
                xaxis=dict(visible=False),
                yaxis=dict(visible=False),
                zaxis=dict(visible=False)
            )
        )
    )

    # Show the Plotly figure
    fig.show()

# Load the Aneurism object file in Google Colab
# Note: Make sure to upload the "Aneurism.obj" file to your Colab
environment first
file_path = '/opt/google/drive/Aneurism.obj'  # Adjust the path
accordingly

# Read the triangle mesh from the object file
aneurism_mesh = o3d.io.read_triangle_mesh(file_path)
if aneurism_mesh.is_empty():
    exit()

# Visualize the Aneurism mesh with different shading methods
shading_methods = [None, 'flat', 'gouraud', 'phong']
```

```
for shading_method in shading_methods:
    visualize_mesh_with_shading(aneurism_mesh, shading=shading_method)
```