Author: **Daniel José Moreno Pérez**

Table of Contents

Undertand what we are doing

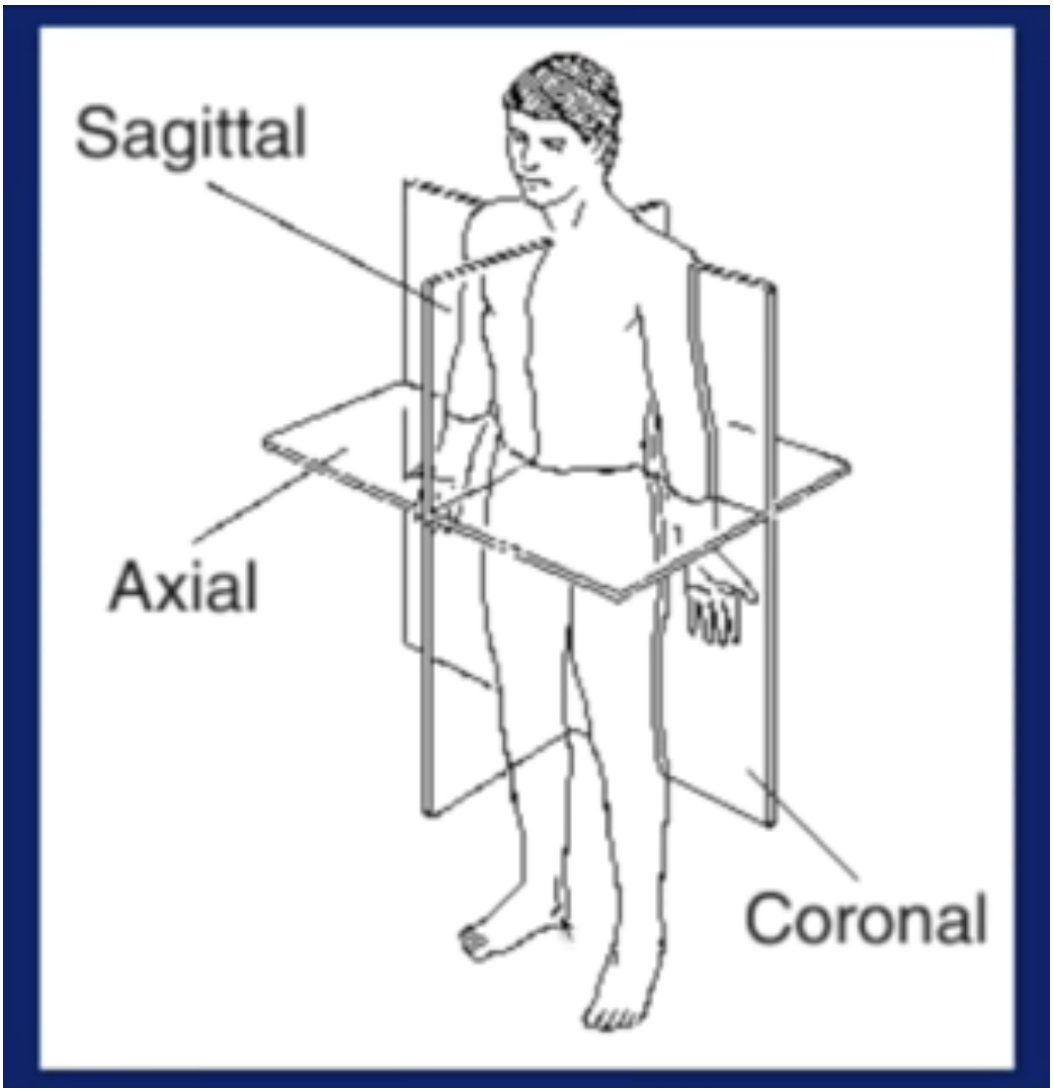
What is a CT image, Hounsfield untis, windowing and Anatomical planes

- A **CT (Computed Tomography)** image is a medical imaging method that employs computer-processed combinations of multiple X-ray measurements taken from different angles to produce cross-sectional (tomographic) images of specific areas of a scanned object. The brightness represents the density. The brighter the more density is represented. This is where Hounsfield untis come to play.

- **Hounsfield units (HU)** are a quantitative scale for describing radiodensity. A value of 0 HU is assigned to the density of water, and the density of air is set at -1000 HU. Each tissue type within the body has a distinctive range on the Hounsfield scale, which allows for identification and characterization. Bone typically has very high HU values, for example, while fat tissue is at the lower end of the scale.

- Normally, most of the tissues are in the range from 0 to 250 (grey scale), so it's difficult to differ them. **Windowing** consists of rescale the range so we get more information within 0 to 250 while losing it outside of the range.

- **Anatomical planes**
  - **Sagittal Plane**: This is a vertical plane that divides the body into left and right parts
  - **Axial Plane**: Also known as the transverse plane, this horizontal plane divides the body into upper (superior) and lower (inferior) parts.
  - **Coronal Plane**: This is another type of vertical plane that divides the body into front (anterior) and back (posterior) sections



How we divide space into voxels

We make a grid of the space of our CT image, each part of the grid is called voxel (volume pixel). This is the same approach as in 2D imaging, only that the least units are squares (pixels) and in 3D is a cube (voxel). These cubes have specific dimensions height, width and depth. Later in the programming section we'll see that our image is divided into voxels with dimensions $0.9433594 \times 0.9433594 \times 2.5$ (in milimiters).

Reescaling the voxels

We need to calculate the scale factor for each coordinate.

$$\text{scale factor}_{\text{axis}} = \frac{\text{original dimension axis}}{\text{desired dimension axis}}$$

In our example:

$$\text{scale factor}_{\text{height}} = \frac{0.9433594}{3} = 0.3145$$
$$\text{scale factor}_{\text{width}} = \frac{0.9433594}{3} = 0.3145$$
$$\text{scale factor}_{\text{depth}} = \frac{2.5}{3} = 0.8333$$

Affine transformation and coordinates

In simple words, affine spaces are a more general abstraction of vector spaces. We can apply affine spaces to position voxels relatively to a reference space. This relation is store in the affine matrix. Thanks to this, two images, although the might have not been taken in the exact same postion, can be compared because the relative position to the reference space is known in both pictures.

In other words, the affine matrix allows as to compare the position of two different images.

Key points:

- Cordinate (0,0,0) is the magnet isocenter
- Units for all three axes are milimeters
- If we have two affine transformations with A and B affine matrix. Applying, transformation A and then B is equivalent to apply $C = B \times A$.

3D Affine Matrix

In 3D, an affine matrix usually takes the form of a 4x4 matrix. This allows for representing transformations in three-dimensional space, including translation, which wouldn't be possible with a 3x3 matrix used for 2D transformations.

Here's a general form of a 3D affine matrix: $A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Where $a11$ through $a33$ are responsible for rotation, scaling, and shearing, and tx, ty, and tz are the translation components for the x, y, and z axes, respectively. The bottom row is used to maintain the homogeneity of the matrix, which allows for affine transformations to be applied to points in 3D space.

The dimension of the matrix is $(\text{output dimension} + 1, \text{input dimension} + 1)$

Example of a 3D Affine Transformation

Consider that you need to compare two images A, B where B needs to be moved by 5 units along the x-axis, 3 units along the y-axis, and 2 units along the z-axis, and also rotates the object 30 degrees around the z-axis in order to have the "same coordinates" as A. The corresponding affine matrix would look like this:

$$M = \begin{bmatrix} \cos(30^\circ) & -\sin(30^\circ) & 0 & 5 \\ \sin(30^\circ) & \cos(30^\circ) & 0 & 3 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Let, v_A be coordinates of the image A, and w_B be coordinates of the image B. Then:
then $v_B = M \times v_A$ and v_B can be compared with w_B
and $w_A = M^{-1} \times w_B$ and w_A can be compared with v_A

Why is it necessary to change affine matrix when reescaling the voxels?

Reescaling voxels can be interprated as affine transformation. Scaling matrices can be written as follows:

$$R = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where s_x, s_y, and s_z are the scaling factors along the x, y, and z axes respectively. This matrix, when applied to a vector in homogeneous coordinates, scales the vector's components by the corresponding factors without rotating or shearing it.

So the new affine transformation associated to the new affine matrix M_{new} would be calculated as follows:
$M_{\text{new}} = R \times M$ where R is the reescaling matrix and M is the previous affine matrix (voxels not changed)

MIP and Average operation

A medical image is tipically 3 dimensional $M \in \mathbb{R}^{h \times w \times d}$. So in order to plot a 2d image we need to fix one of the three axis. This implies that we lose information. That is why there are some operations to convert 3D images into 2D images in a certain way to minimize the lost information. In this examples we asume we apply it to the width axis, although it can be applied to any axis.

MIP operation

Mathematically, $MIP : \mathbb{R}^{h \times w \times d} \longrightarrow \mathbb{R}^{h \times w}$.
This is achived by applying the max function to the depth axis. Let $i \in 1, \ldots, h$ represent a layer height, $j \in 1, \ldots, w$ layer in width and $k \in 1, \ldots, d$ layer in depth.

Then if we fix k we get a matrix:

$$M_k = \begin{bmatrix} m_{11k} & m_{12k} & \cdots & m_{1wk} \\ m_{21} & m_{22} & \cdots & m_{2wk} \\ \vdots & \vdots & \ddots & \vdots \\ m_{h1} & m_{h2} & \cdots & m_{hwk} \end{bmatrix}$$

So if we apply the max function per row we get:

$$MIP_k(M_k) = \begin{bmatrix} \max\limits_{j\in 1,\dots,w} m_{1jk} \\ \max\limits_{j\in 1,\dots,w} m_{2jk} \\ \vdots \\ \max\limits_{j\in 1,\dots,w} m_{hjk} \end{bmatrix} \in \mathbb{R}^{h\times 1\times 1}$$

So we've transformed 2D images to 1D. Applying this MIP function to every M_k we obtained a 2D image from a 3D image.

$$MIP(M) = \begin{bmatrix} \max\limits_{j\in 1,\dots,w} m_{1j1} & \cdots & \max\limits_{j\in 1,\dots,w} m_{1jk} \\ \max\limits_{j\in 1,\dots,w} m_{2j1} & \cdots & \max\limits_{j\in 1,\dots,w} m_{2jk} \\ \vdots & \vdots & \ddots & \vdots \\ \max\limits_{j\in 1,\dots,w} m_{hj1} & \cdots & \max\limits_{j\in 1,\dots,w} m_{hjk} \end{bmatrix} \text{ where } M \in \mathbb{R}^{h\times w\times d}$$

Average operation

$AVG : \mathbb{R}^{h\times w\times d} \longrightarrow \mathbb{R}^{h\times w}$

We proceed similarly as with MIP, the difference is that when we fix k the operation is: $AVG_k(M_k) = \begin{bmatrix} \sum_{j=1}^{w} m_{1jk} \\ \sum_{j=1}^{w} m_{2jk} \\ \vdots \\ \sum_{j=1}^{w} m_{hjk} \end{bmatrix} \in \mathbb{R}^{h\times 1\times 1}$

Then, $AVG(M) = \begin{bmatrix} \sum_{j=1}^{w} m_{1j1} & \cdots & \sum_{j=1}^{w} m_{1jk} \\ \sum_{j=1}^{w} m_{2j1} & \cdots & \sum_{j=1}^{w} m_{2jk} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{j=1}^{w} m_{hj1} & \cdots & \sum_{j=1}^{w} m_{hjk} \end{bmatrix}$ where $M \in \mathbb{R}^{h\times w\times d}$

Note:

In the context of medical imaging the **width** would be the **axial** plane (x-axis), **depth** the **sagittal** plane (y-axis) and **height** the **coronal** plane (z-axis).

NIfTI format and NiBabel library

NIfTI format used for storing MRI (Magnetic Resonance Imaging) data. It evolved from the ANALYZE format, addressing its limitations by supporting both 3D and 4D

(time series) imaging data, incorporating spatial and temporal units, and allowing for the storage of metadata within the file. In NIfTI files, the structure typically includes a header, an affine matrix, and the data itself.

Load image:

```
img = nib.load('Path')
```

Header

Contains metadata such as image dimensions, datatype, and voxel sizes.
Access the header:

```
header = img.header
```

`header['dim']`, `header['datatype']`, etc., to interact with specific components.

Affine Matrix

```
affine_matrix = img.affine
```

Example of use Affine Matrix

```
import nibabel as nib

# Load the NIfTI image
nifti_img = nib.load('path_to_your_nifti_file.nii')

# Get the affine matrix
affine_matrix = nifti_img.affine

# Example voxel coordinates
input_coords = [10, 20, 30, 1]  # Note: 1 is added for
homogeneous coordinates

# Convert voxel coordinates to world coordinates
output_coords = affine_matrix.dot(voxel_coords)[:3]  # [:3] to
drop the homogeneous coordinate

print("World coordinates:", output_coords)
```

Data

Numpy array of the image data for manipulation or analysis.

```
img.get_fdata()
```

Solution to the exercise

Reescaling and MIP

First, import the libraries and functions

In [ ]:
```
import nibabel as nib
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import zoom
from nibabel.processing import resample_from_to
```

Load `CT.nii` and print the key components.

In [ ]:
```
# Load the image
img = nib.load('CT.nii')

# print("Header: ")
# print(img.header)
print("\n Affine: ")
print(img.affine)
print("\n Data: ")
print(img.get_fdata())
# Shape of the image
print("\n Shape: ")
print(img.shape)

# Dimensions of the voxels
print("\n Voxel dimensions: ")
print(img.header.get_zooms())

# A voxel
print("\n A voxel: ")
print(img.get_fdata()[0, 0, 0])
print("That's the value of the first voxel. It's a Hounsfield unit (HU) v
```

```
Affine:
[[ -0.94335938   0.           0.         249.02832031]
 [ -0.           0.94335938   0.         -30.52832031]
 [  0.          -0.           2.5        -14.        ]
 [  0.           0.           0.           1.        ]]

Data:
[[[-1024. -1024. -1024. ... -1024. -1024. -1024.]
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]
  ...
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]]

 [[-1024. -1024. -1024. ... -1024. -1024. -1024.]
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]
  ...
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]]

 [[-1024. -1024. -1024. ... -1024. -1024. -1024.]
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]
  ...
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]]

 ...

 [[-1024. -1024. -1024. ... -1024. -1024. -1024.]
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]
  ...
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]]

 [[-1024. -1024. -1024. ... -1024. -1024. -1024.]
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]
  ...
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]]

 [[-1024. -1024. -1024. ... -1024. -1024. -1024.]
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]
  ...
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]
  [-1024. -1024. -1024. ... -1024. -1024. -1024.]]]

Shape:
(512, 512, 541)
```

```
 Voxel dimensions:
(0.9433594, 0.9433594, 2.5)

 A voxel:
-1024.0
That's the value of the first voxel. It's a Hounsfield unit (HU) value. In
this case, it's -1024.0 HU, which is the value for air.
```

Resample image

In [ ]:
```
# Define the new zooms (target resolution)
new_zooms = (3, 3, 3)

# Calculate scale factors for resampling

## Get the original zooms
original_zooms = img.header.get_zooms()

## Calculate the scale factors
scale_factor = [oz/nz for oz, nz in zip(original_zooms, new_zooms)]
print(scale_factor)
```

```
[0.314453125, 0.314453125, 0.8333333333333334]
```

We need to change the affine matrix due to the reescaling

In [ ]:
```
# Compute the new affine transformation matrix
new_affine = np.diag(scale_factor + [1]) @ img.affine
print(new_affine)

# Update the header with new zooms and create a new Nifti1Image
new_header = img.header.copy()
new_header.set_zooms(new_zooms)
resampled_img = nib.Nifti1Image(img.get_fdata(), new_affine, header=new_h

# Resample the image to the new resolution
resampled_img = resample_from_to(img, (resampled_img.shape, resampled_img
```

```
[[ -0.2966423    0.           0.          78.30773354]
 [  0.           0.2966423    0.          -9.59972572]
 [  0.           0.           2.08333333 -11.66666667]
 [  0.           0.           0.           1.        ]]
```

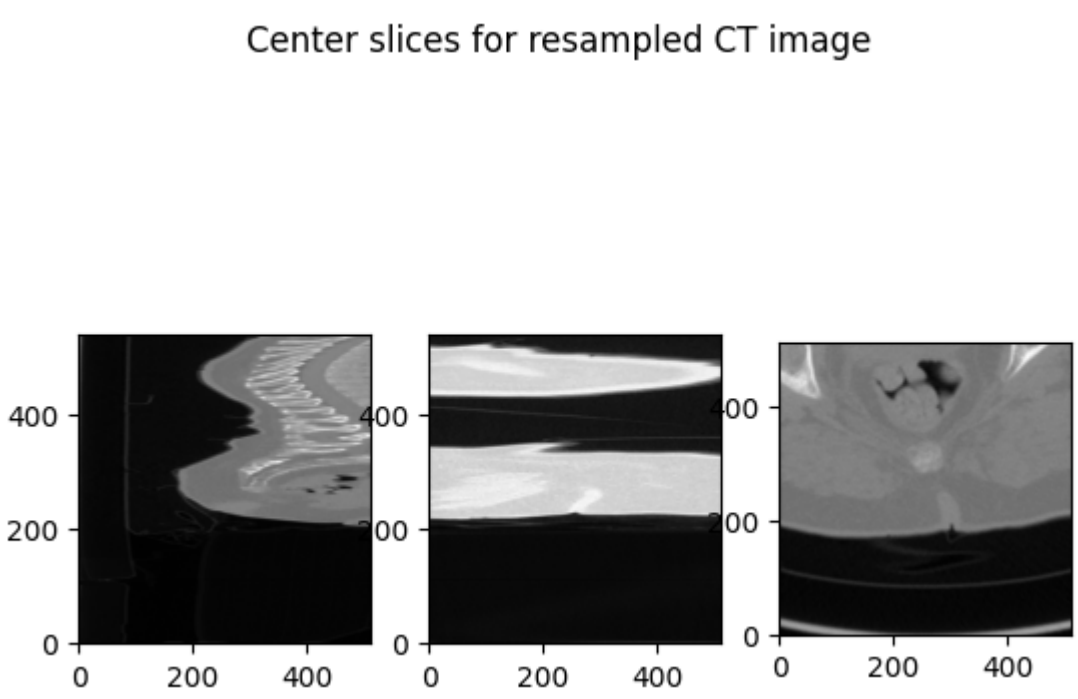
The rescaling matrix is:

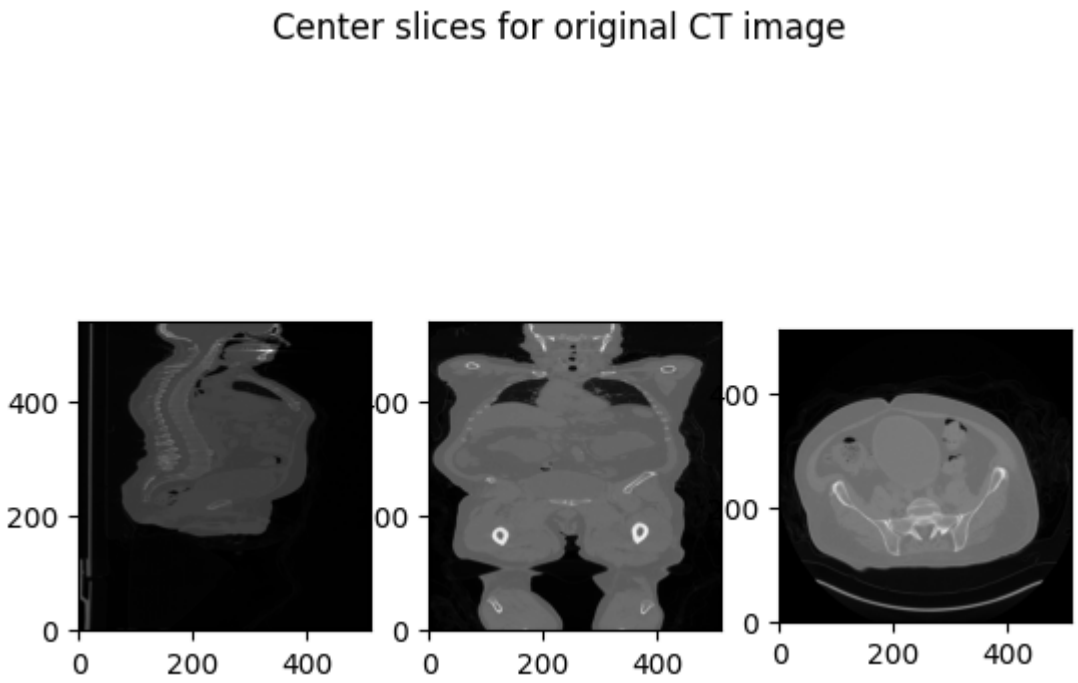
In [ ]:
```
print(np.diag(scale_factor + [1]))
```

```
[[0.31445312 0.         0.         0.        ]
 [0.         0.31445312 0.         0.        ]
 [0.         0.         0.83333333 0.        ]
 [0.         0.         0.         1.        ]]
```

In [ ]:
```
# Convert the image to a NumPy array for visualization
data_resampled = resampled_img.get_fdata()
data = img.get_fdata()

# Function to visualize the slices
def show_slices(slices):
    """ Function to display row of image slices """
    fig, axes = plt.subplots(1, len(slices))
```

```
    for i, slice in enumerate(slices):
        axes[i].imshow(slice.T, cmap="gray", origin="lower")

# Select the middle slice from each dimension to display
slice_0 = data_resampled[data_resampled.shape[0]//2, :, :] ## // for inte
slice_1 = data_resampled[:, data_resampled.shape[1]//2, :]
slice_2 = data_resampled[:, :, data_resampled.shape[2]//2]

# Select the middle slice from each dimension to display
slice_0_original = data[data.shape[0]//2, :, :]
slice_1_original = data[:, data.shape[1]//2, :]
slice_2_original = data[:, :, data.shape[2]//2]

# Call the function to display the slices for the resampled image
show_slices([slice_0, slice_1, slice_2])
plt.suptitle("Center slices for resampled CT image")
plt.show()

# Call the function to display the slices for the original image
show_slices([slice_0_original, slice_1_original, slice_2_original])
plt.suptitle("Center slices for original CT image")
plt.show()
```



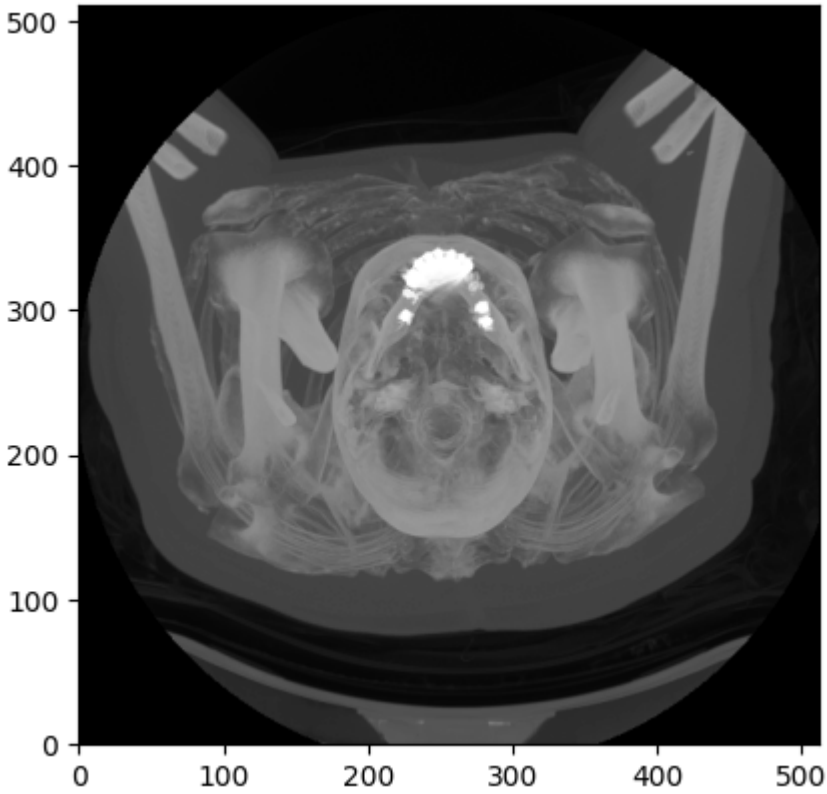


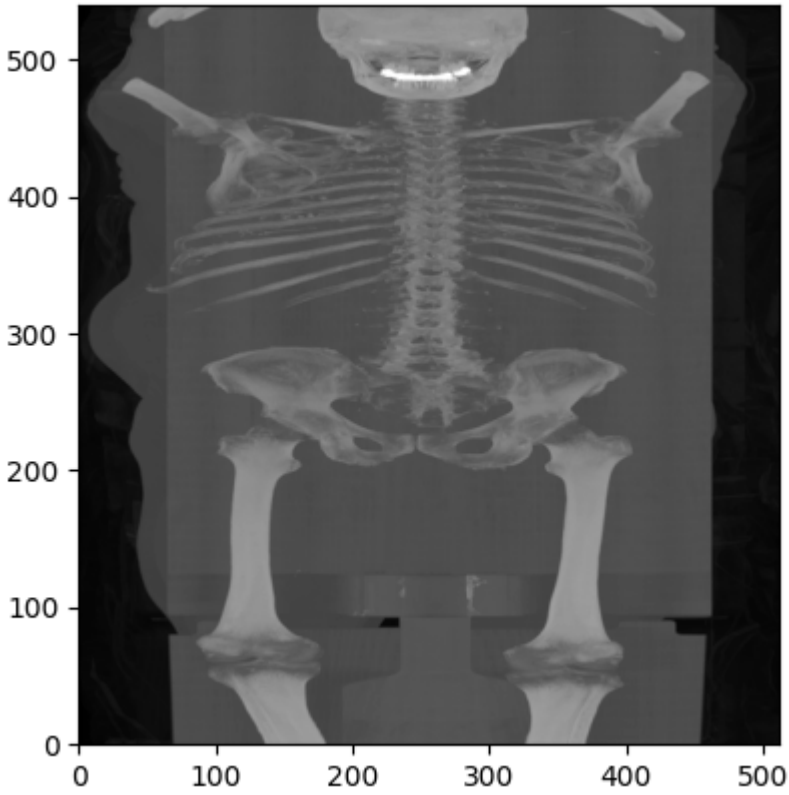
MIP operation

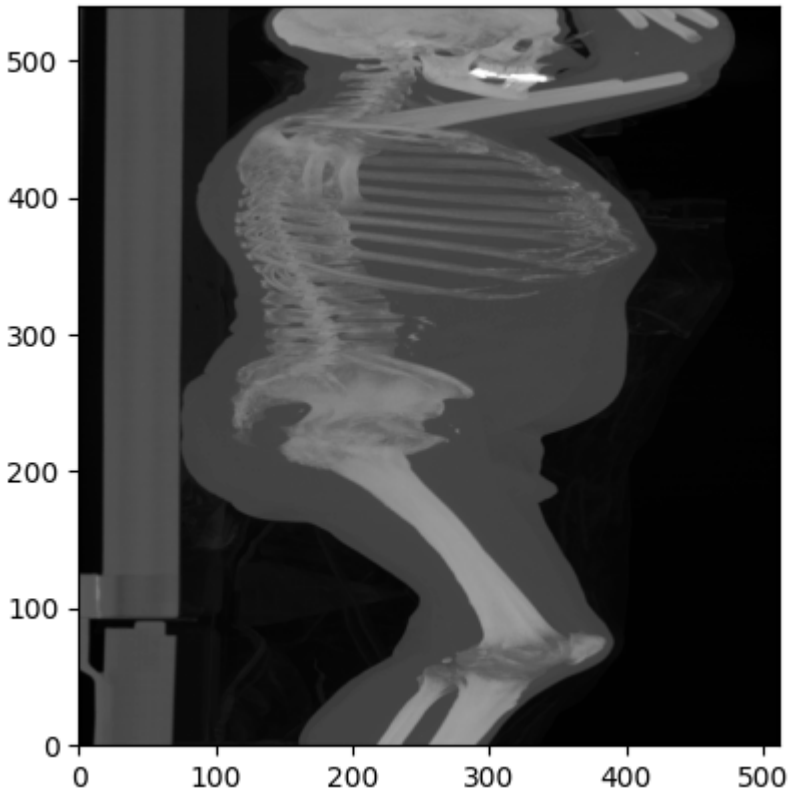
MIP consists on applying max function, so instead of having (.,.,.) array now we have (.,.). It highlights brightness.

In [ ]:
```
# MIP in the axial plane
mip_axial = np.max(data, axis=2)
print(mip_axial.shape)
plt.imshow(mip_axial.T, cmap='gray', origin='lower')
plt.show()
axial = data.shape[2] // 2


# MIP in the coronal plane
mip_coronal = np.max(data, axis=1)
plt.imshow(mip_coronal.T, cmap='gray', origin='lower')
plt.show()

# MIP in the sagittal plane
mip_sagittal = np.max(data, axis=0)
plt.imshow(mip_sagittal.T, cmap='gray', origin='lower')
plt.show()
```

```
(512, 512)
```

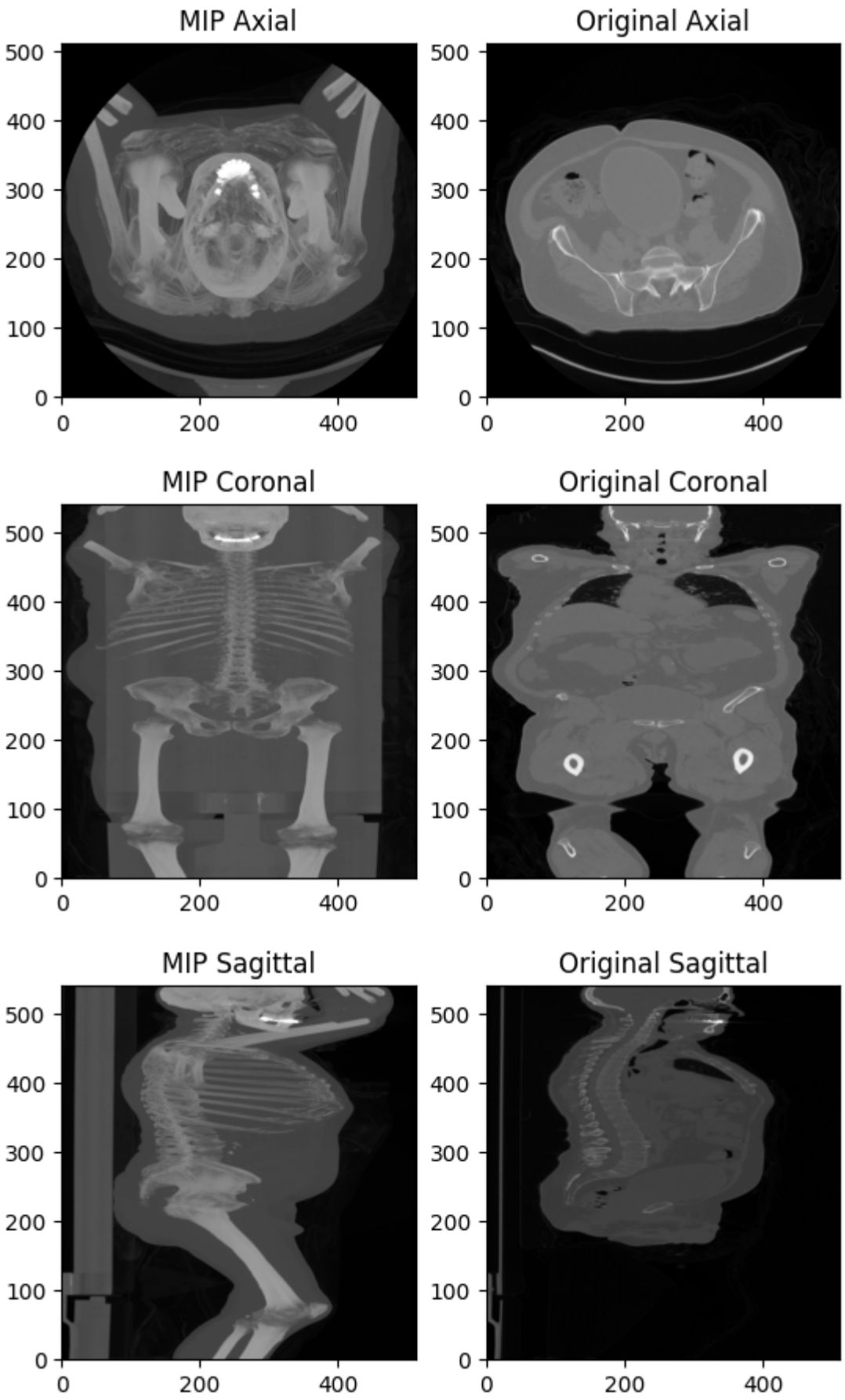

500
400
300
200
100
0
0
100
200
300
400
500


500
400
300
200
100
0
0
100
200
300
400
500



In [ ]:
```
# Plot mip_axial and original
fig, axes = plt.subplots(1, 2)
axes[0].imshow(mip_axial.T, cmap='gray', origin='lower')
axes[0].set_title('MIP Axial')
axes[1].imshow(data[:, :, data.shape[2]//2].T, cmap='gray', origin='lower
print(data.shape[2]//2)
axes[1].set_title('Original Axial')
plt.show()

# Plot mip_coronal and original
fig, axes = plt.subplots(1, 2)
axes[0].imshow(mip_coronal.T, cmap='gray', origin='lower')
axes[0].set_title('MIP Coronal')
axes[1].imshow(data[:, data.shape[1]//2, :].T, cmap='gray', origin='lower
axes[1].set_title('Original Coronal')
plt.show()

# Plot mip_sagittal and original
fig, axes = plt.subplots(1, 2)
axes[0].imshow(mip_sagittal.T, cmap='gray', origin='lower')
axes[0].set_title('MIP Sagittal')
axes[1].imshow(data[data.shape[0]//2, :, :].T, cmap='gray', origin='lower
axes[1].set_title('Original Sagittal')
plt.show()
```

```
270
```



When comparing both pictures we can find out the usefulness of MIP. Bones are represented by high HU values, as a result, they can be easily seen in the output image after applying MIP.

Calculate volume

aorta is the largest artery in the body that carries blood from the heart to the rest of the body.

Segmentation can be achieved through various techniques, ranging from manual delineation by clinical experts to automated algorithms that utilize image processing and machine learning techniques. The purpose of aorta segmentation is to aid in the diagnosis, analysis, and treatment planning for cardiovascular diseases. By accurately segmenting the aorta, clinicians and researchers can measure important features such as the diameter, length, and volume of the aorta, as well as detect abnormalities like aneurysms, dissections, or other pathologies.

Automated segmentation methods are particularly valuable for their potential to provide quick, reproducible, and objective assessments, which are especially important in high-volume clinical settings or in longitudinal studies where consistency in measurement is crucial.

In [ ]:
```
# Load the segmentation image
segmentation_img = nib.load('segmentation.nii.gz')

# Get the data array from the image
segmentation_data = segmentation_img.get_fdata()

print(segmentation_img.header)
print("\n the shape is: " + str(segmentation_img.shape))
```

```
<class 'nibabel.nifti1.Nifti1Header'> object, endian='<'
sizeof_hdr      : 348
data_type       : b''
db_name         : b''
extents         : 0
session_error   : 0
regular         : b''
dim_info        : 0
dim             : [  3 512 512 541   1   1   1   1]
intent_p1       : 0.0
intent_p2       : 0.0
intent_p3       : 0.0
intent_code     : none
datatype        : uint8
bitpix          : 8
slice_start     : 0
pixdim          : [-1.         0.9433594  0.9433594  2.5        1.
1.
  1.         1.       ]
vox_offset      : 0.0
scl_slope       : nan
scl_inter       : nan
slice_end       : 0
slice_code      : unknown
xyzt_units      : 2
cal_max         : 0.0
cal_min         : 0.0
slice_duration  : 0.0
toffset         : 0.0
glmax           : 0
glmin           : 0
descrip         : b''
aux_file        : b''
qform_code      : unknown
sform_code      : aligned
quatern_b       : 0.0
quatern_c       : 1.0
quatern_d       : 0.0
qoffset_x       : 249.02832
qoffset_y       : -30.52832
qoffset_z       : -14.0
srow_x          : [ -0.9433594   0.          0.        249.02832  ]
srow_y          : [ -0.          0.9433594   0.        -30.52832  ]
srow_z          : [  0.   -0.    2.5 -14. ]
intent_name     : b''
magic           : b'n+1'
```

```
the shape is: (512, 512, 541)
```

We need to find how are the voxels that represent the aorta labeled.

In [ ]:
```
print(segmentation_data)
```

```
[[[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

 ...

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]]
```

Let's make a function to see the values that can be found in the arrays. I suppose that there are only 0 and 1. 0 means the voxel is not part of the aorta, 1 means that the voxel is part of the aorta. However, let's be sure.

In [ ]:

```
def extract_unique_values(segmentation_data):
    unique_values = set() # Use a set to avoid duplicates
    for layer in segmentation_data:
        for row in layer:
            for value in row:
```

```
                unique_values.add(value)
    return list(unique_values)

unique_values = extract_unique_values(segmentation_data)
print(unique_values)
```

```
[0.0, 1.0]
```

The only two values that can be found in the arrays are 0 and 1. And the interpretion of these values is the one I already meantioned.

In [ ]:
```
# Define the value that represents the aorta segmentation
aorta_value = 1 # We checked is 1 in the previous step (extract_unique_va

# Count the number of voxels corresponding to the aorta
aorta_voxels = (segmentation_data == aorta_value).sum()

# Calculate the volume of a single voxel
voxel_dims = segmentation_img.header.get_zooms()  # This gives you the vo
voxel_volume_mm3 = np.prod(voxel_dims)  # Multiply the dimensions togethe

# Calculate the total aorta volume in cubic millimeters
aorta_volume_mm3 = aorta_voxels * voxel_volume_mm3

# Convert the volume to milliliters
aorta_volume_ml = aorta_volume_mm3 / 1000

print(f'The volume of the segmented aorta is: {aorta_volume_ml} mL')
```

```
The volume of the segmented aorta is: 323.7954567146301 mL
```

Sources of information

- CT image:
  - https://en.wikipedia.org/wiki/CT_scan
  - https://www.youtube.com/watch?v=VnpqylFYtqI&t=1s
- Affine:
  - https://nipy.org/nibabel/coordinate_systems.html
  - https://www.youtube.com/watch?v=PVhgAbh01ks
- NIfTI images:
  - https://neuraldatascience.io/8-mri/nifti.html
- ChatGPT