Symulator tomografu komputerowego

1. Skład grupy:

Mateusz Poprawa 148073

2. Zastosowany model tomografu:

Stożkowy

3. Zastosowany język programowania oraz dodatkowe biblioteki:

język: python

biblioteki: streamlit, pydicom, numpy, PIL, math

4. Opis głównych funkcji programu:

a. pozyskiwanie odczytów dla poszczególnych detektorów

```python
def emiter_cord(r, alpha, height, width):
    xe = r * math.cos(math.radians(alpha)) + (width / 2)
    ye = (height / 2) - r * math.sin(math.radians(alpha))
    return xe, ye

def detector_cord(r, alpha,  height, width, i):
    xd = r * math.cos(math.radians(alpha) + math.pi - math.radians(settings.phi / 2) + math.radians(i * settings.phi / (settings.n - 1))) + (width / 2)
    yd = (height / 2) - r * math.sin(math.radians(alpha) + math.pi - math.radians(settings.phi / 2) + math.radians(i * settings.phi / (settings.n - 1)))
    return xd, yd
```

```python
def Bresenham_Algorithm_DA_X(x1, y1, x2, y2, image, view, illum = False)
    dx = x2 - x1
    dy = y2 - y1
    m = abs(dy / dx)
    if dx > 0:
        x_inc = 1
    else:
        x_inc = -1
    if dy > 0:
        y_inc = 1
    else:
        y_inc = -1
    i1 = math.floor(x1)
    i2 = math.floor(x2)
    j = math.floor(y1)
    e =  -(1 - (y1 - j) - (dy*(1 - (x1 - i1))) / dx)
    l = []
    for i in range(i1, i2, x_inc):
        if i >= 0 and j >= 0 and i < image.width and j < image.height:
            l.append((i, j))
        if (e >= 0):
            j += y_inc
            e -= 1.0
        e += m
    if(illum):
        illuminate(l, image, view)
    return l
```

```
for alpha in range(0, 360, settings.alpha_step):
    sinogram.append([])
    xe, ye = emiter_cord(r, alpha, height, width)
    for i in range(0, settings.n):
        xd, yd = detector_cord(r, alpha, height, width, i)
        points = calcualte_points(xe, ye, xd, yd, image, image_view)
        val = 0
        count = 0
        for (x, y) in points:
            val += im_matrix[y][x]
            count += 1
        if count > 0:
            sinogram[j].append(val / count)
        else:
            sinogram[j].append(0)
```

b .filtrowanie sinogramu, zastosowany rozmiar maski = 21

```
def create_kernel():
    kernel = []
    start = -(settings.kernel_size - 1) / 2
    end = (settings.kernel_size - 1) / 2 + 1
    for k in range(int(start), int(end)):
        if k == 0:
            kernel.append(1)
        elif k % 2 == 0:
            kernel.append(0)
        else:
            kernel.append((-4 / (np.pi * np.pi)) / (k * k))
    return kernel

def filter(sinogram):
    kernel = create_kernel()
    for i in range(len(sinogram)):
        sinogram[i] = np.convolve(sinogram[i], kernel, mode="same")
    return sinogram
```

c. ustalanie jasności poszczególnych punktów obrazu wynikowego oraz jego przetwarzanie końcowe

```
for i in range(0, settings.n):
    xd, yd = detector_cord(r, alpha, height, width, i)
    points = calcualte_points(xe, ye, xd, yd, image, view)
    for (x, y) in points:
        im_matrix[y][x] += sinogram[j][i]
```

```
normalized_matrix = 255*(matrix - np.min(matrix))/(np.max(matrix) - np.min(matrix))
```

d. wyznaczanie wartości miary RMSE na podstawie obrazu źródłowego oraz wynikowego

```
def RMSE(image, backprojection_image):
    width, height = image.width, image.height
    rmse = 0
    img1 = np.array(ImageOps.grayscale(image))
    img2 = np.array(ImageOps.grayscale(backprojection_image))
    for x in range(width):
        for y in range(height):
            rmse += (int(img1[y][x]) - int(img2[y][x]))**2
    rmse /= width * height
    rmse = math.sqrt(rmse)
    print(rmse)
```

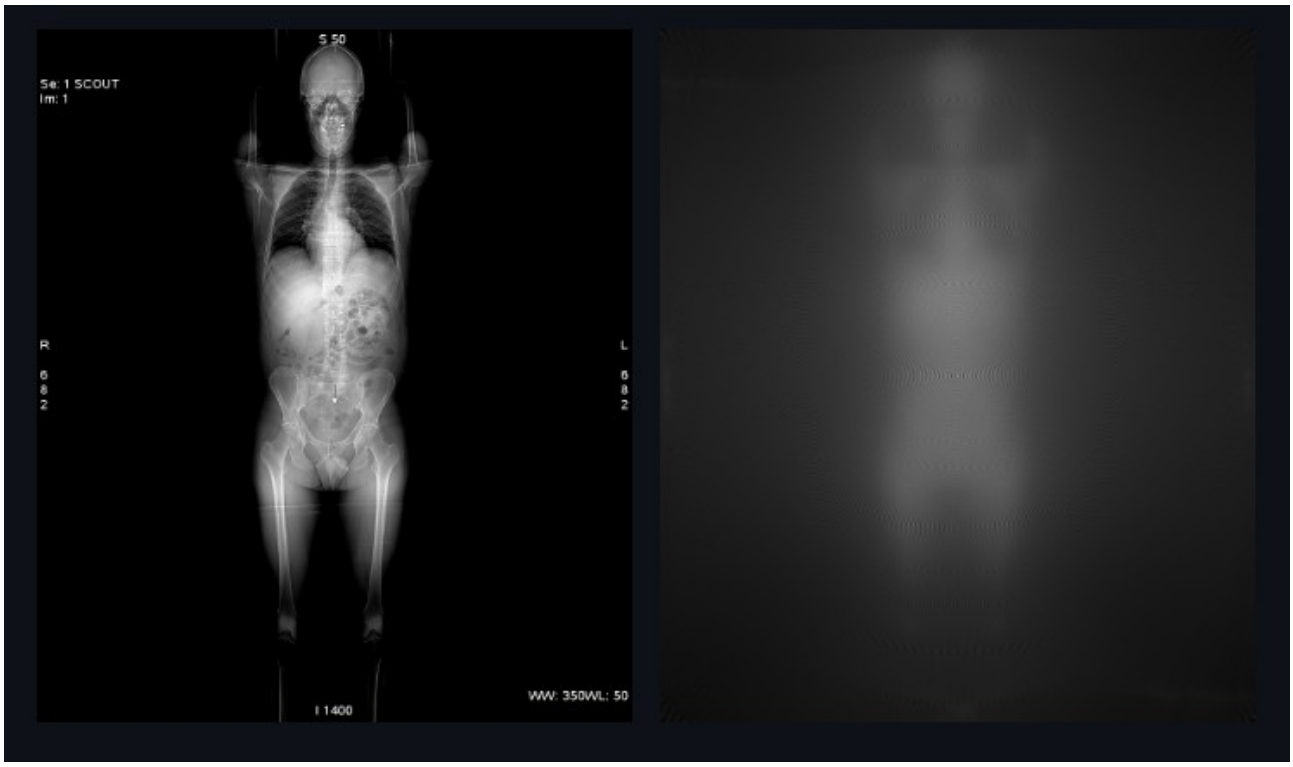
e. odczyt i zapis plików DICOM

```python
def save_as_dicom(file_name, img, patient_data):
    img_converted = convert_image_to_ubyte(img)

    # Populate required values for file meta information
    meta = Dataset()
    meta.MediaStorageSOPClassUID = pydicom._storage_sopclass_uids.CTImageStorage
    meta.MediaStorageSOPInstanceUID = pydicom.uid.generate_uid()
    meta.TransferSyntaxUID = pydicom.uid.ExplicitVRLittleEndian

    ds = FileDataset(None, {}, preamble=b"\0" * 128)
    ds.file_meta = meta

    ds.is_little_endian = True
    ds.is_implicit_VR = False

    ds.SOPClassUID = pydicom._storage_sopclass_uids.CTImageStorage
    ds.SOPInstanceUID = meta.MediaStorageSOPInstanceUID

    ds.PatientName = patient_data["PatientName"]
    ds.PatientID = patient_data["PatientID"]
    ds.ImageComments = patient_data["ImageComments"]
    ds.Date = patient_data["Date"]


    ds.Modality = "CT"
    ds.SeriesInstanceUID = pydicom.uid.generate_uid()
    ds.StudyInstanceUID = pydicom.uid.generate_uid()
    ds.FrameOfReferenceUID = pydicom.uid.generate_uid()

    ds.BitsStored = 8
    ds.BitsAllocated = 8
    ds.SamplesPerPixel = 1
    ds.HighBit = 7

    ds.ImagesInAcquisition = 1
    ds.InstanceNumber = 1

    ds.Rows, ds.Columns = img_converted.shape

    ds.ImageType = r"ORIGINAL\PRIMARY\AXIAL"

    ds.PhotometricInterpretation = "MONOCHROME2"
    ds.PixelRepresentation = 0

    pydicom.dataset.validate_file_meta(ds.file_meta, enforce_standard=True)

    ds.PixelData = img_converted.tobytes()

    ds.save_as(file_name, write_like_original=False)
```

```python
st.header("Odczyt pliku DICOM")
dcm_file = st.file_uploader("Wybierz plik", type=['dcm'])
image_tab, data_tab = st.tabs(["Obraz", "Dane"])
if dcm_file is not None:
    dcm = dcmread(dcm_file)
    with image_tab:
        st.image(dcm.pixel_array)
    with data_tab:
        st.text(dcm)
```

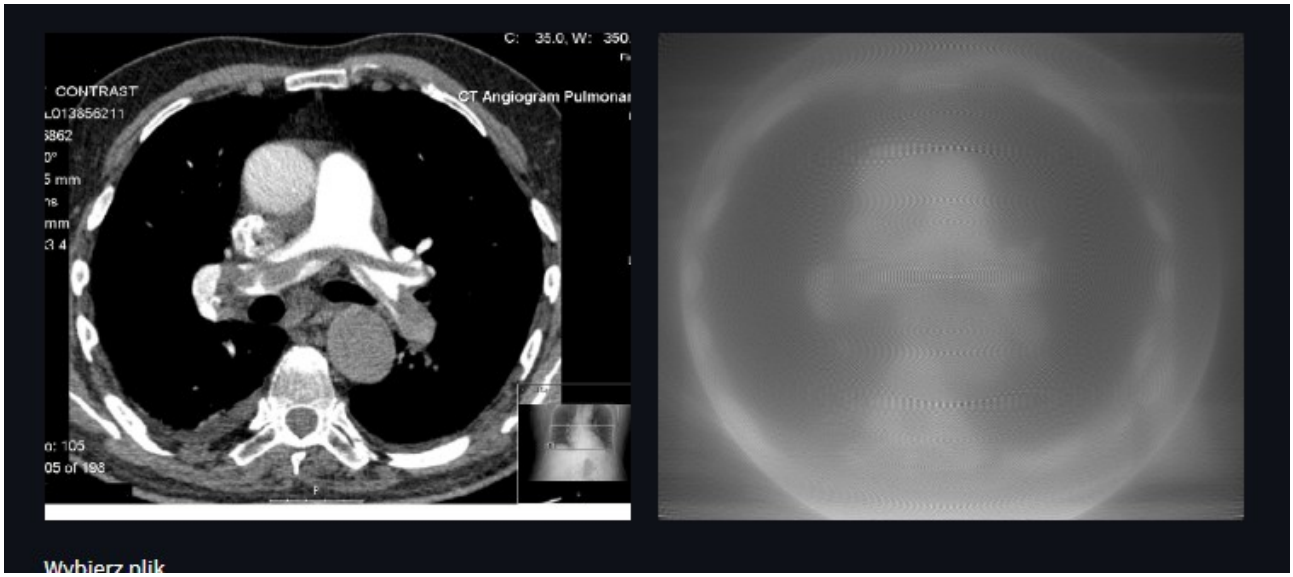
5. Przykład działania programu dla dwóch obrazków wejściowych

Bez filtrowania

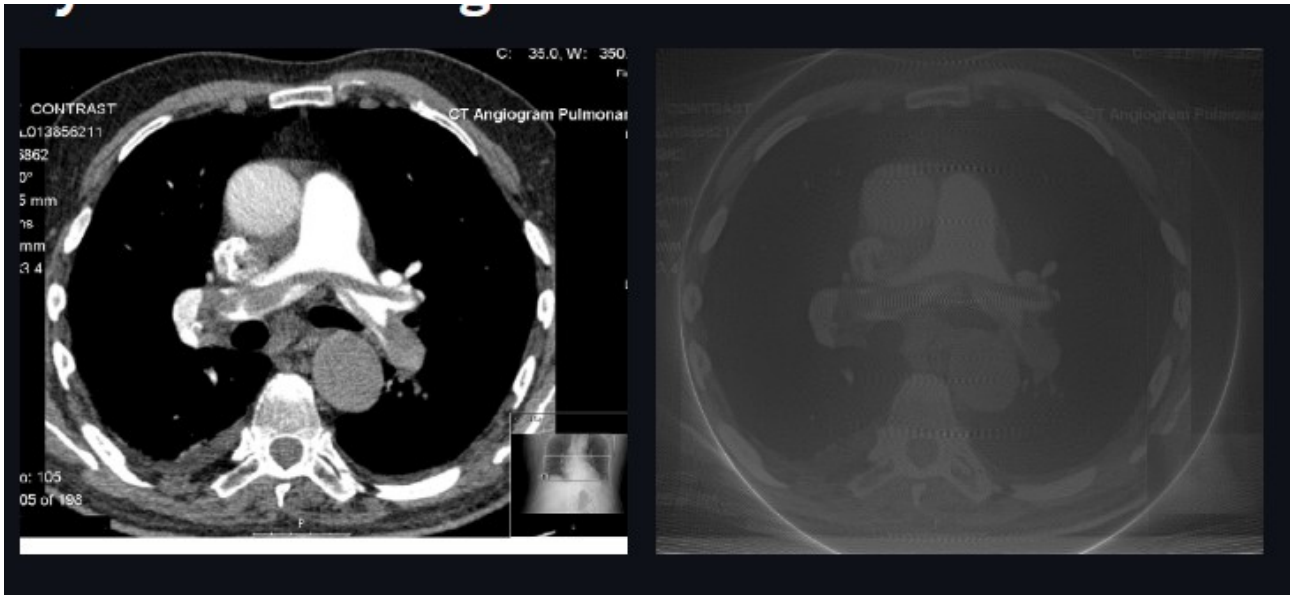


Z filtrowaniem

Bez filtrowania



Z filtrowaniem



6. Wynik eksperymentu sprawdzającego wpływ poszczególnych parametrów (liczba detektorów, liczba skanów, rozpiętość stożka/wachlarza z detektorami) na jakość obrazu wynikowego wyrażoną za pomocą miary RMSE.

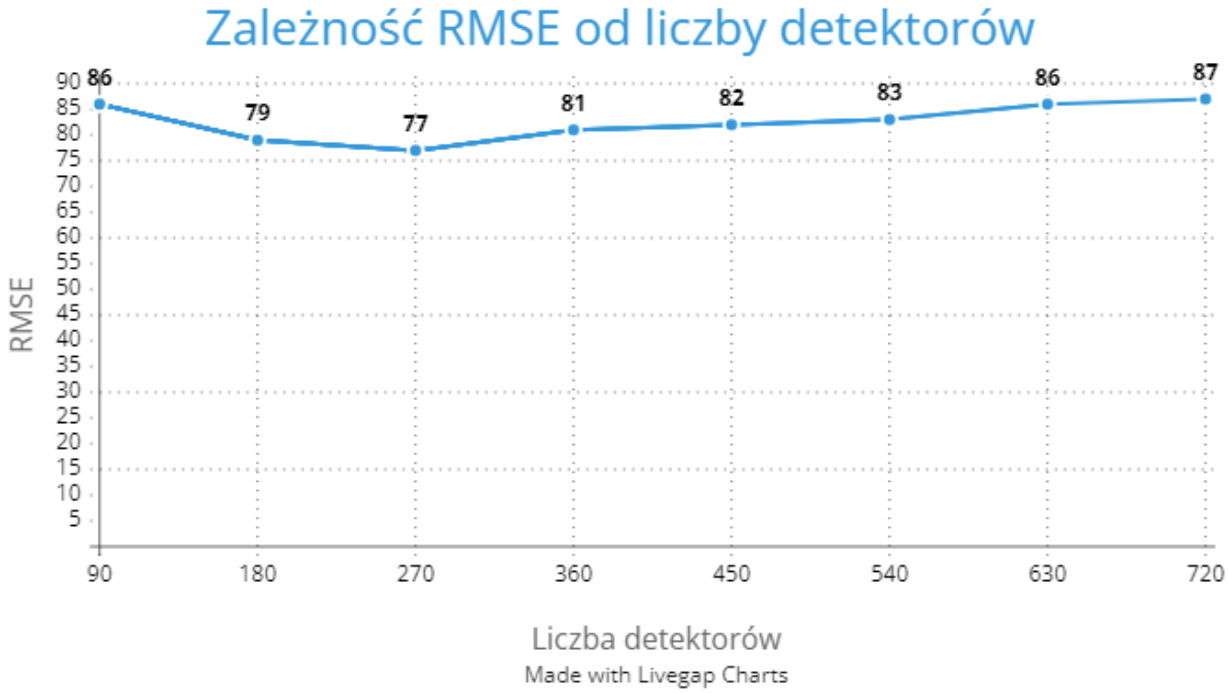
a.

detektory	90	180	270	360	450	540	630	720
RMSE	86	79	77	81	82	83	86	87

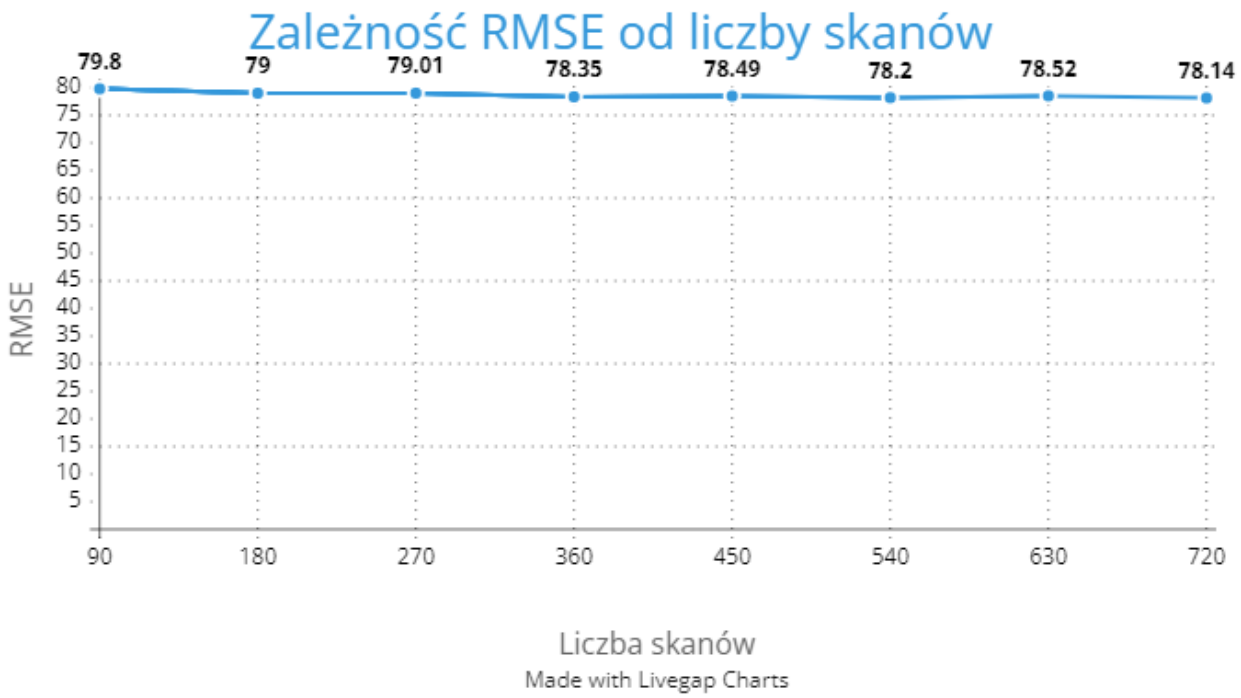
Liczba skanów	90	180	270	360	450	540	630	720
RMSE	79.80	79.00	79.01	78.35	78.49	78.20	78.52	78.14

Rozpiętość wachlarza	45	90	135	180	225	270
RMSE	83.4	83.54	81.36	79.00	81.53	83.73

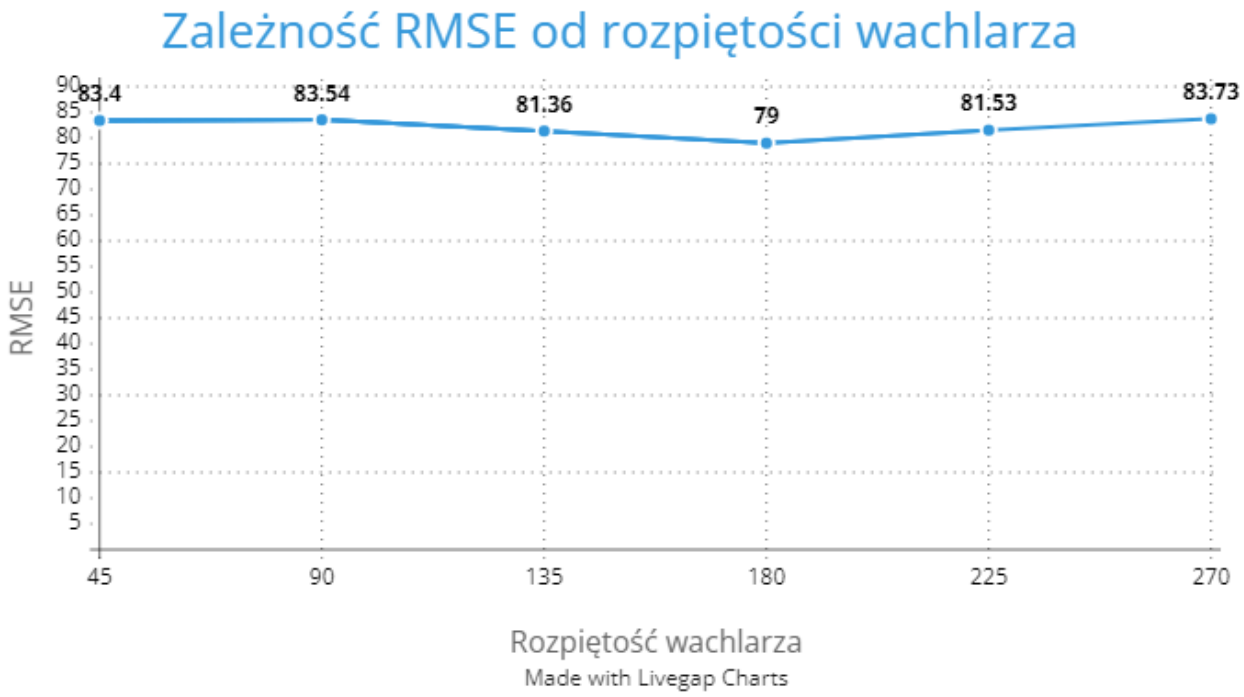
b.



Wraz ze zwiększaniem się liczby detektorów więcej pikseli przyjmuje kolor szary w wyniku czego zwiększa się błąd jednak obraz staje się bardziej wyraźny.



Wraz ze wzrostem liczby skanów obraz staje się trochę bardziej wyraźny, błąd utrzymuje się na podobnym poziome lekko spadając.

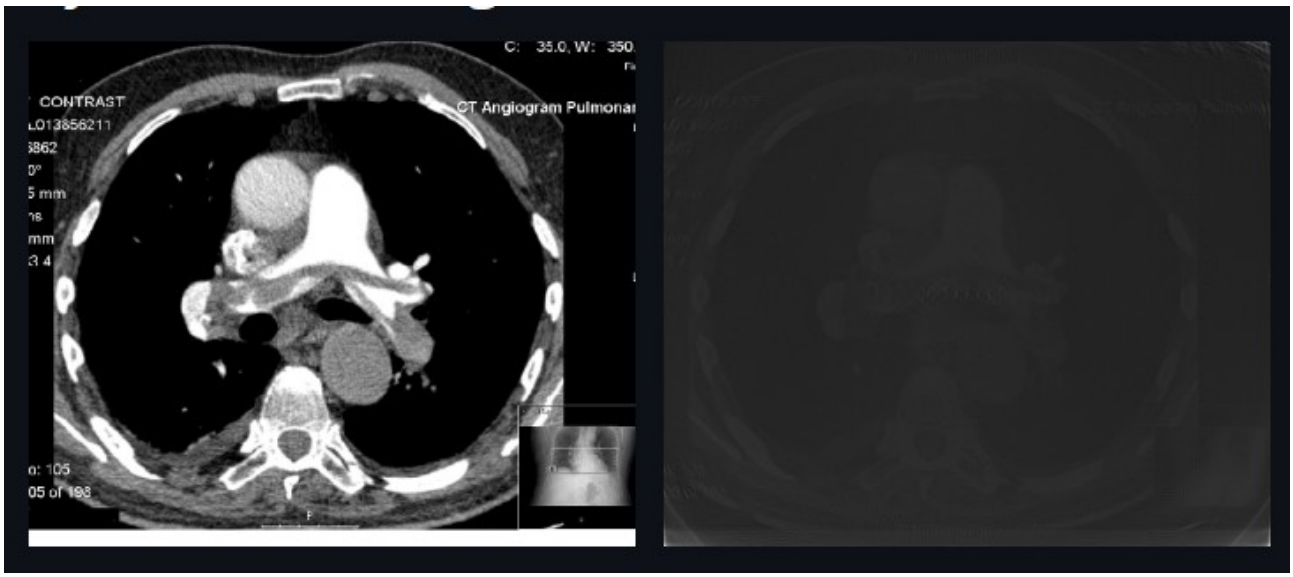


Dla skrajny wartości obraz jest słabej jakości, a błąd jest wartości. Dla średnich wartości obraz jest najlepszy, a błąd najmniejszy.

c.

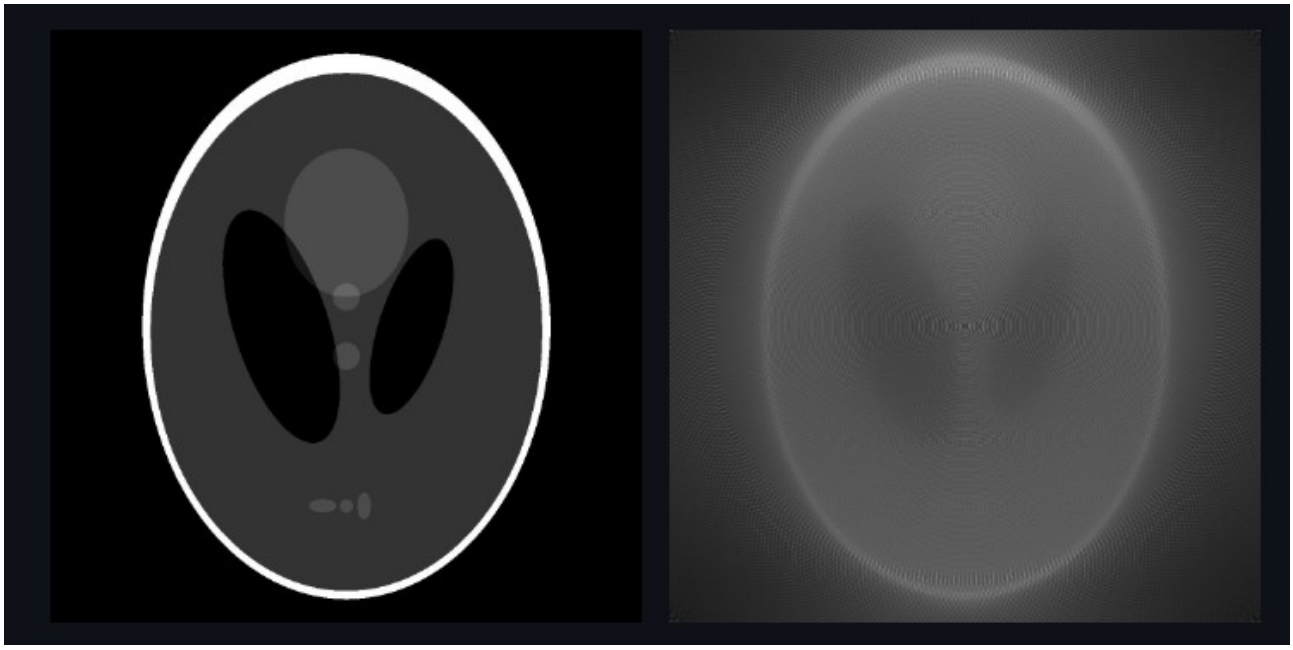
RMSE = 78.03 bez filtrowania

RMSE = 88.09 z filtrowaniem

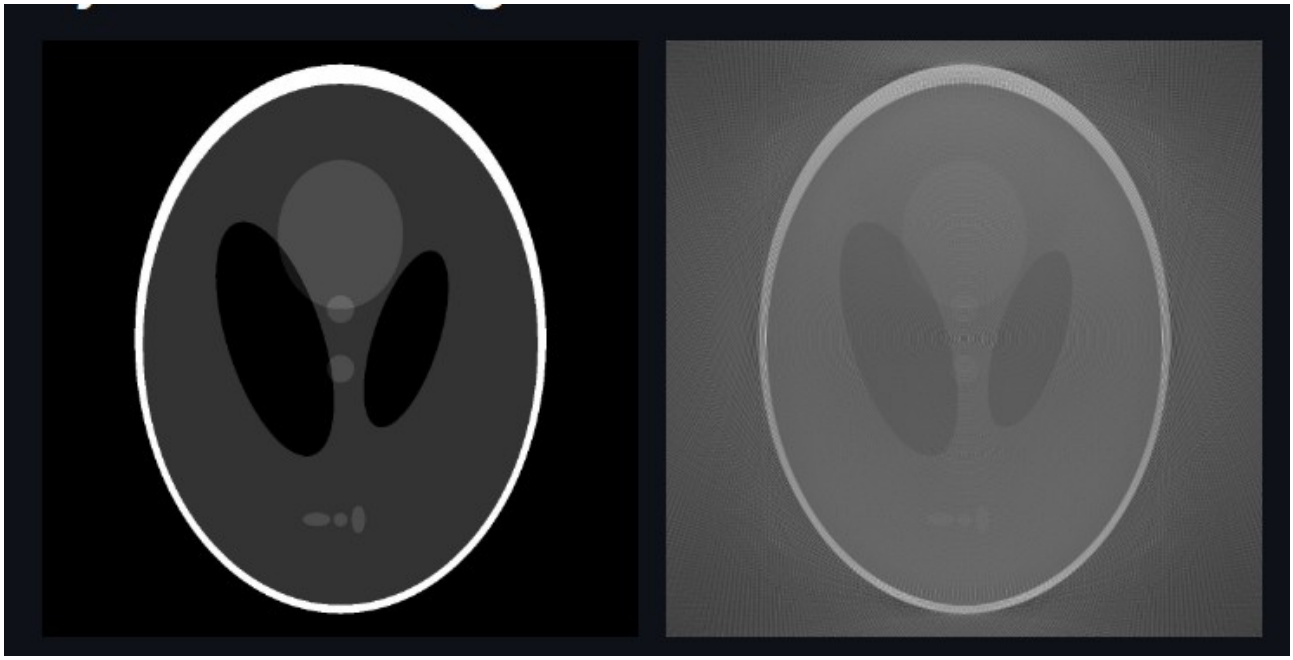


Obraz z filtrowaniem ma wyższy RMSE oraz jest ciemniejszy, jest też bardziej ostry.

RMSE = 62.53 bez filtrowania



RMSE = 75.77 z filtrowaniem



Obraz z filtrowaniem jest jaśniejszy oraz bardziej ostry, ma też większy RMSE.