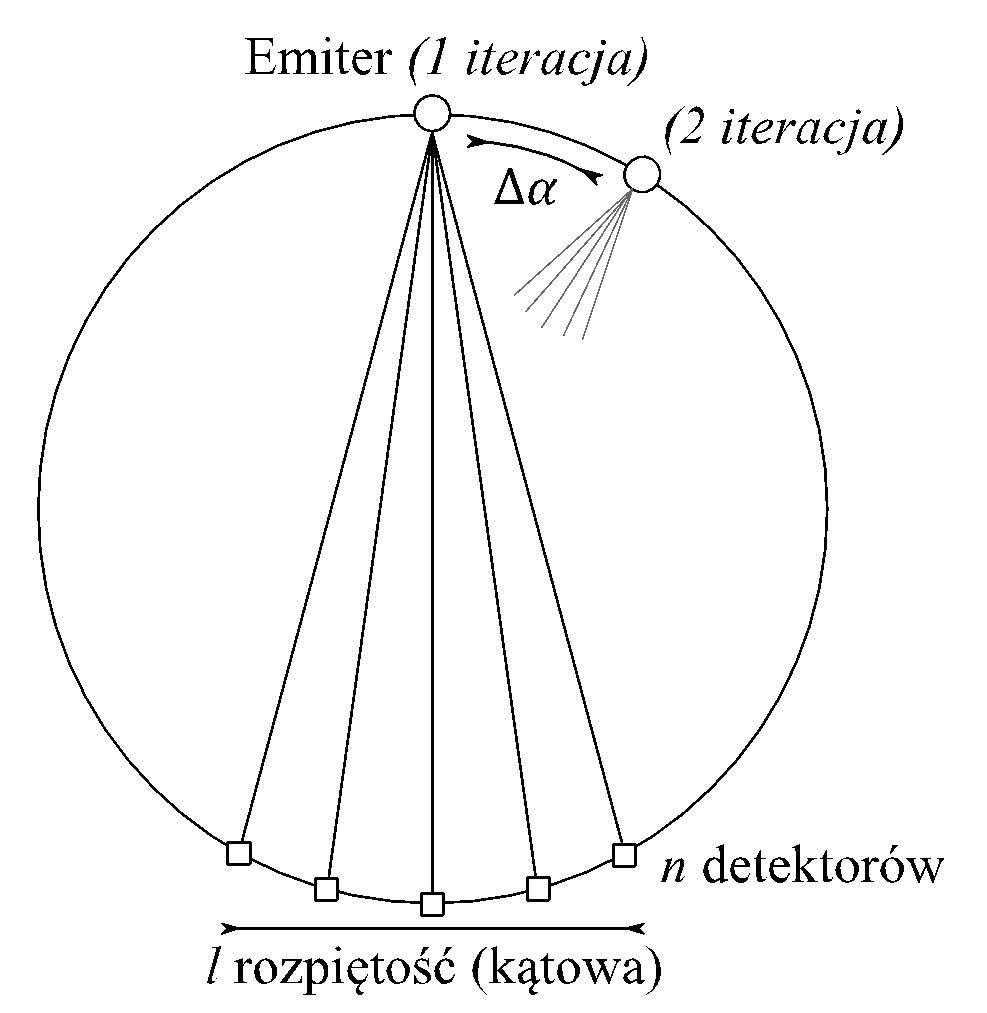
Informatyka w medycynie - Tomograf - Raport

|  |  |  |
| --- | --- | --- |
| **Grupa laboratoryjna:** L9 | Skład grupy: | **Prowadzący zajęcia:** |
| Wiktor Jordeczka (151 785) |
| Konrad Kaczmarek (151 741) | **dr inż. Iwo Błądek** |

1. Zastosowany model tomografu:

Wykorzystano model stożkowy, 1 emiter do n detektorów.

1. Zastosowany język programowania oraz dodatkowe biblioteki:
   1. Język: Python
   2. Dodatkowe niestandardowe biblioteki:
      1. numpy – operacje na wektorach i macierzach
      2. skimage (scikit-image w pip) – wczytywanie obrazka
      3. pydicom – obsługa plików dicom
      4. tkinter - GUI
      5. ttkthemes - GUI
      6. tkcalendar - GUI
2. Opis głównych części programu
   1. Pozyskiwanie odczytów dla poszczególnych detektorów

1. def radonTransform(img,t,img\_label, emitterRange = 180, numOfDetectors = 180, numOfScans = 180, alphaShift = 2, animating = False, animationInterval = 20):

2.   t.set\_label\_progress("Transformata Radona")

3.   alphaShift = math.radians(alphaShift)

4.   center = (len(img)//2, len(img[0])//2)

5.   R = max(center) \* math.sqrt(2)

6.   alpha = (2\*math.pi\*90)/360 # alpha emitera, zaczynamy od 90 stopni

7.   phi = (2\*math.pi\*emitterRange)/360 # kąt rozwarcia dla detektorów

8.   sinogram = np.zeros(shape=(numOfScans,numOfDetectors))

9.   for scan in range(numOfScans): # dla każdego skanu (pozycji emitera)

10.     t.progress((scan/numOfScans)\*100)

11.     # tutaj dla wzyznaczanych współrzędnych odnoszę się do środka, aby nie robić potem przesunięć

12.     xe = center[0] + round(R \* math.cos(alpha))

13.     ye = center[1] - round(R \* math.sin(alpha))

14.     for det in range(numOfDetectors): # dla każdego detektora wyznaczamy jego współrzędne

15.       xd = center[0] + round(R \* math.cos(alpha+math.pi-phi/2 + det\*phi/(numOfDetectors-1)))

16.       yd = center[1] - round(R \* math.sin(alpha+math.pi-phi/2 + det\*phi/(numOfDetectors-1)))

17.       sinogram[scan][det] = bresenham(img,xe,ye,xd,yd) # rysujemy linię od emitera do detektora, zapisujemy średnią jasność pikseli

18.     alpha += alphaShift # wykonujemy przesunięcie

19.     if animating and scan % animationInterval == 0: # sekcja animacji

20.       t.showImage(normalize(sinogram),img\_label)

21.   return sinogram

W linijce 17. Korzystamy z funkcji rysującej linię algorytmem Bresenhama, zwracającej średnią jasność pikseli   
w narysowanej linii:

1. def bresenham(img,x1,y1,x2,y2): # algorytm Bresenhama, zwraca średnią jasność

2.   max\_x = len(img)

3.   max\_y = len(img[0])

4.   sum = 0

5.   n = 0

6.   x = x1

7.   x\_inc = 1 # inkrement

8.   dx = x2 - x1 # przesunięcie

9.   y = y1

10.   y\_inc = 1

11.   dy = y2 - y1

12.   if x1>=x2: # rysujemy w lewo, zamiast w prawo

13.     x\_inc = -1

14.     dx = x1 - x2

15.   if y1>=y2: # rysujemy w dół, zamiast w górę

16.     y\_inc = -1

17.     dy = y1 - y2

18.   if dx > dy: # oś wiodąca OX

19.     ep\_inc\_a = 2\*(dy-dx)

20.     ep\_inc\_b = 2\*dy

21.     ep = ep\_inc\_b - dx # błąd (odchylenie od współrzędnych całkowitych)

22.     while x!=x2:

23.       if not(x>=max\_x or y>=max\_y or x<0 or y<0): # sprawdzamy czy nie jesteśmy poza obrazkiem

24.         sum += img[x-1][y-1]

25.         n += 1

26.       if ep>=0:

27.         y += y\_inc

28.         ep += ep\_inc\_a

29.       else:

30.         ep += ep\_inc\_b

31.       x += x\_inc

32.   else: # oś wiodąca OY

33.     ep\_inc\_a = 2\*(dx-dy)

34.     ep\_inc\_b = 2\*dx

35.     ep = ep\_inc\_b - dy # błąd (odchylenie od współrzędnych całkowitych)

36.     x = x1

37.     while y!=y2:

38.       if not(x>=max\_x or y>=max\_y or x<0 or y<0):

39.         sum += img[x-1][y-1]

40.         n += 1

41.       if ep>=0:

42.         x += x\_inc

43.         ep += ep\_inc\_a

44.       else:

45.         ep += ep\_inc\_b

46.       y += y\_inc

47.   try:

48.     a = sum/n # średnia jasność pikseli w rysowanej linii

49.   except:

50.     a = 0 # Uroki okręgu opisanego ¯\\_(ツ)\_/¯

51.   return a

* 1. Filtrowanie sinogramu, zastosowany rozmiar maski

1. def filtr(sinogram,t,img\_label, animating = False): # filtr splotowy

2.   t.set\_label\_progress("Filtrowanie")

3.   result = []

4.   for i in range(1,11): # tworzymy połowę maski

5.     if i%2==0:

6.       result.append(0)

7.     else:

8.       result.append(-4/pow(math.pi,2)/pow(i,2))

9.   res2 = result.copy()

10.   result.reverse() # wykorzystujemy odbicie lustrzane

11.   kernel = result + [1] + res2 # tworzymy maskę o rozmiarze 21

12.   for i in range(sinogram.shape[0]):

13.     t.progress((i/sinogram.shape[0])\*100)

14.     sinogram[i, :] = np.convolve(sinogram[i, :], kernel, mode="same") # wykonujemy splot

15.     if animating:

16.       t.showImage(normalize(sinogram),img\_label,flag=True)

* 1. Ustalanie jakości poszczególnych punktów obrazu wynikowego oraz jego przetwarzanie końcowe (uśrednianie, normalizacja)

Uśrednianie jest wykonywane w trakcie rysowania linii algorytmem Bresenhama, zaprezentowanego [powyżej.](#Bresenham)

Jasność pikseli obrazu końcowego jest ustalana podczas wykonywania odwrotnej transformaty Radona:

1. def inverseRadonTransform(sinogram, img,t,img\_label, emitterRange = 180, numOfDetectors = 180, numOfScans = 180, alphaShift = 2, animating = False, animationInterval=20):

2.   t.set\_label\_progress("Odwrotna Transformata Radona")

3.   alphaShift = math.radians(alphaShift)

4.   center = (img.shape[0]//2, img.shape[1]//2)

5.   R = max(center) \* math.sqrt(2)

6.   alpha = (2\*math.pi\*90)/360 # alpha emitera, zaczynamy od 90 stopni

7.   phi = (2\*math.pi\*emitterRange)/360 # kąt rozwarcia dla detektorów

8.   reconstructed = np.zeros(shape=img.shape)

9.   for scan in range(numOfScans): # dla każdego skanu (wzmocnienia)

10.     t.progress((scan/numOfScans)\*100)

11.     # tutaj dla wzyznaczanych współrzędnych odnoszę się do środka, aby nie robić potem przesunięć

12.     xe = center[0] + round(R \* math.cos(alpha))

13.     ye = center[1] - round(R \* math.sin(alpha))

14.     for det in range(numOfDetectors): # dla każdego detektora

15.       xd = center[0] + round(R \* math.cos(alpha+math.pi-phi/2 + det\*phi/(numOfDetectors-1)))

16.       yd = center[1] - round(R \* math.sin(alpha+math.pi-phi/2 + det\*phi/(numOfDetectors-1)))

17.       coords = inverseBresenham(img.shape[0], img.shape[1],xe,ye,xd,yd) # wyznaczamy linię między emiterem a detektorem

18.       reconstructed[tuple(np.transpose(coords))] += sinogram[scan][det] # wzmacniamy piksele wyznaczonej linii o zapisaną wcześniej średnią

19.     alpha += alphaShift # wykonujemy przesunięcie

20.     if animating and scan % animationInterval == 0: # sekcja animacji

21.       t.showImage(normalize(reconstructed),img\_label)

22.       mse, rmse = calcRMSE(img,normalize(reconstructed))

23.       t.set\_label\_progress(f"Odwrotna Transformata Radona\nRMSE: {rmse:.4f}")

24.   mse, rmse = calcRMSE(img,normalize(reconstructed))

25.   t.set\_label\_progress(f"Odwrotna Transformata Radona\nRMSE: {rmse:.4f}")

26.   return normalize(reconstructed) # normalizujemy obraz wynikowy

W linijce 26. wykorzystana jest poniższa funkcja normalizująca:

1. def normalize(im): # normalizacja obrazu

2.   norm = np.zeros(shape=(im.shape))

3.   im\_max = im.max()

4.   im\_min = 0

5.   for x in range(im.shape[0]):

6.     norm[x] = np.interp(im[x], (im\_min, im\_max), (0, 255))

7.   return norm

* 1. Wyznaczenie wartości miary RMSE na podstawie obrazu źródłowego i wynikowego

1. def calcRMSE(im1, im2): # obliczanie RMSE

2.   mse = np.square(np.subtract(im1, im2)).mean()

3.   return (mse, math.sqrt(mse)) # (mse, rmse)

* 1. Odczyt plików DICOM

1. def choose\_file(self):

2-8.       ...

9.         if self.file\_path[-4:] == ".dcm": # dla plików dicom

10.             dcm = dic.dcmread(self.file\_path) # wczytanie pliku

11.             self.im = tm.normalize(dcm.pixel\_array) # wydobycie danych pikseli z pliku dicom + normalizacja

12.             print(dcm)

13.             # odczytujemy wybrane informacje

14.             self.name\_entry.insert(0, dcm.PatientName.family\_comma\_given())

15.             self.id\_entry.insert(0, dcm.get('PatientID', 'BRAK'))

16.             self.date\_entry.set\_date(datetime.strptime(dcm.get('StudyDate', ''), '%m/%d/%y').date())

17.             if (0x0010, 0x4000) in dcm:

18.                 self.comment\_entry.insert(0,dcm[0x00104000].value)

* 1. Zapis plików DICOM

1. def jpg\_to\_dcm(reconstructed,name="BRAK",patient\_id="0",date="",comment="BRAK"): # zapisywanie obrazu do dicom

2.   # Meta tagi

3.   file\_meta = dic.dataset.FileMetaDataset()

4.   file\_meta.MediaStorageSOPClassUID = dic.uid.UID('1.2.840.10008.5.1.4.1.1.2')

5.   file\_meta.MediaStorageSOPInstanceUID = dic.uid.generate\_uid()

6.   file\_meta.ImplementationClassUID = dic.uid.UID("1.2.826.0.1.3680043.8.498.1")

7.   dcm = dic.dataset.FileDataset("output.dcm", {}, file\_meta=file\_meta, preamble=b"\0" \* 128)

8.   dcm.file\_meta.TransferSyntaxUID = dic.uid.ImplicitVRLittleEndian

9.   dcm.is\_little\_endian = True

10.   dcm.is\_implicit\_VR = True

11.

12.   # Zapisujemy wybrane dane o pacjencie i badaniu

13.   dt = datetime.datetime.now()

14.   if date == "":

15.     dcm.StudyDate = dt.strftime('%Y%m%d')

16.     dcm.ContentDate = dt.strftime('%Y%m%d')

17.   else:

18.     dcm.StudyDate = date

19.     dcm.ContentDate = date

20.   dcm.StudyTime = dt.strftime('%H%M%S')

21.   timeStr = dt.strftime('%H%M%S.%f')

22.   dcm.ContentTime = timeStr

23.   dcm.PatientName = name

24.   dcm.PatientID = patient\_id

25.   dcm.StudyID = "1234"

26.   dcm.SeriesNumber = "1"

27.   dcm.PatientComments = comment

28.

29.   # Generujemy unikatowe ID instancji

30.   dcm.SOPInstanceUID = dic.uid.generate\_uid()

31.   dcm.SeriesInstanceUID = dic.uid.generate\_uid()

32.   dcm.StudyInstanceUID = dic.uid.generate\_uid()

33.   dcm.FrameOfReferenceUID = dic.uid.generate\_uid()

34.

35.   # Parametry obrazu

36.   dcm.ImageType = ["ORIGINAL", "PRIMARY", "AXIAL"]

37.   dcm.Modality = "CT"

38.   dcm.Rows = reconstructed.shape[0]

39.   dcm.Columns = reconstructed.shape[1]

40.   dcm.BitsAllocated = 8

41.   dcm.BitsStored = 8

42.   dcm.HighBit = dcm.BitsStored - 1

43.   dcm.SamplesPerPixel = 1

44.   dcm.PhotometricInterpretation = 'MONOCHROME2'

45.   dcm.PixelRepresentation = 0

46.   dcm.PixelData = reconstructed.astype(np.uint8).tobytes() # piksele obrazka

47.

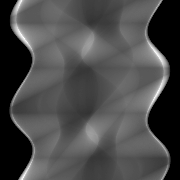
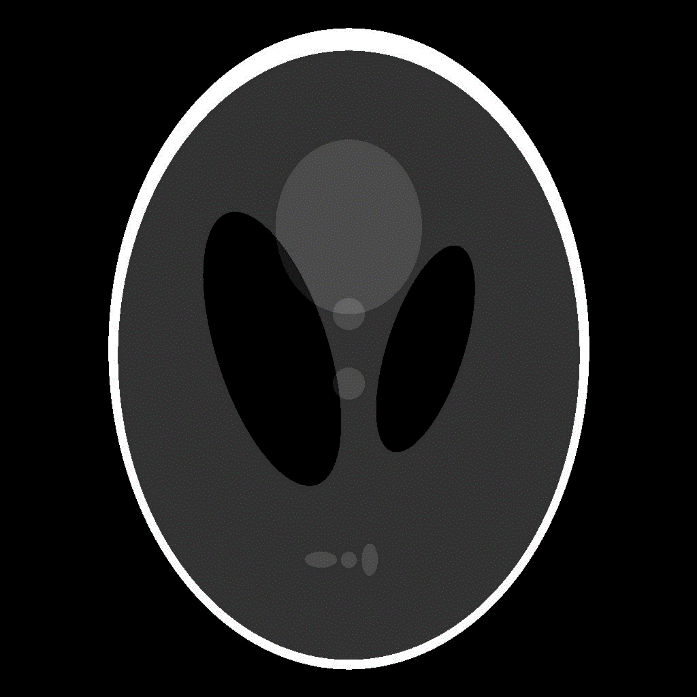
48.   # Prywatne bloki do zapisania własnych danych, np.: komentarza

49.   block = dcm.private\_block(0x000b, "PUT 151785 151741", create=True)

50.   block.add\_new(0x01, "SH", comment)

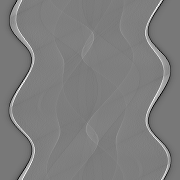
51.   dcm.save\_as("output.dcm", write\_like\_original=False)

1. Przykład działania programu dla dwóch obrazków
   1. Shepp\_Logan



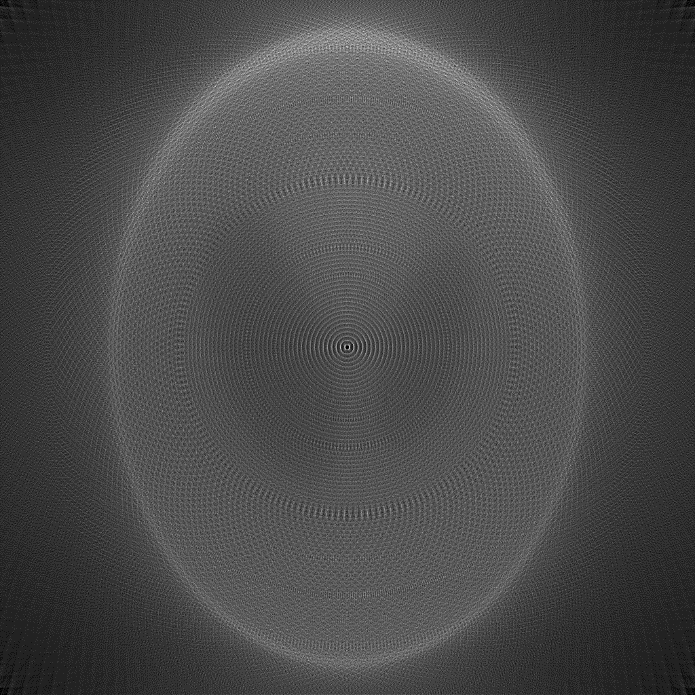
1.1. Obraz oryginalny

1.2. Sinogram



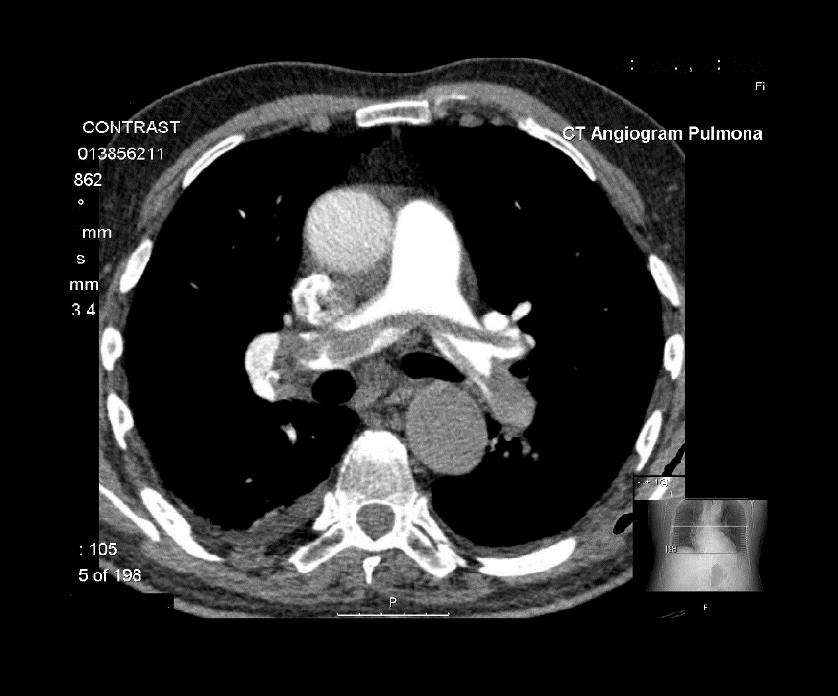
1.3. Filtrowany sinogram

1.4. Obraz zrekonstruowany



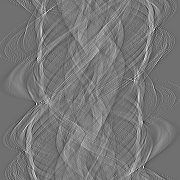
1.5. Obraz zrekonstruowany bez filtrowania

* 1. SADDLE\_PE-large



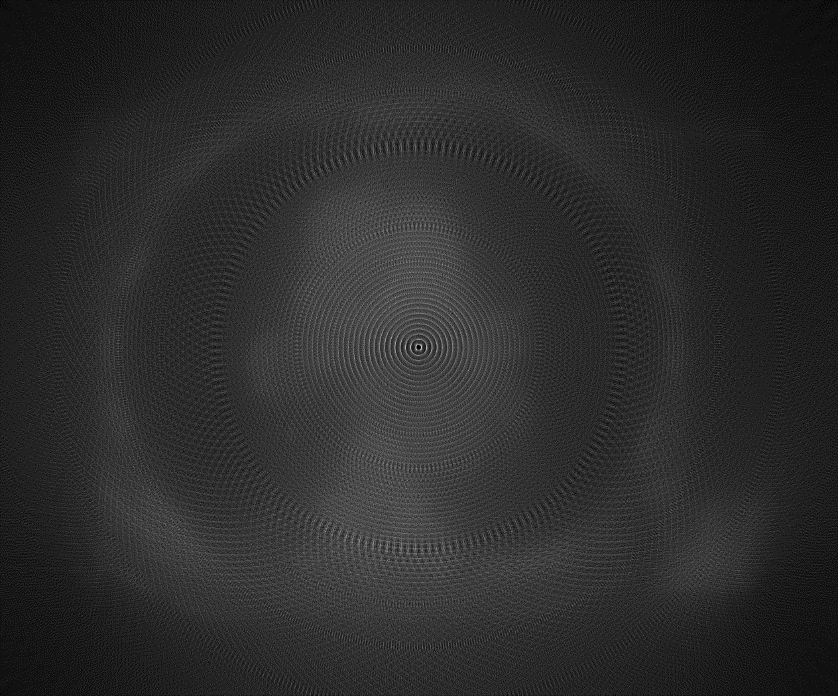
2.2. Obraz oryginalny

2.2. Sinogram



2.3. Filtrowany sinogram

2.4. Zrekonstruowany obraz



2.5. Zrekonstruowany obraz bez filtrowania

1. Wyniki eksperymentów sprawdzających wpływ poszczególnych parametrów na jakość obrazu wynikowego
   1. Eksperymenty zostały przeprowadzone na obrazie Shepp\_Logan, domyślne parametry ustawione na: 180 detektorów, 180 skanów, rozpiętość wachlarza 180°.
   2. Zmienna liczba detektorów od 90 do 720 z krokiem 90
      1. Obrazy

|  |  |  |  |
| --- | --- | --- | --- |
| Liczba detektorów = 90 | Liczba detektorów = 180 | | Liczba detektorów = 270 |
| Liczba detektorów = 360 | Liczba detektorów = 450 | | Liczba detektorów = 540 |
| Liczba detektorów = 630 | | Liczba detektorów = 720 | |

* + 1. Wykres RMSE
    2. Wnioski

Początkowo jakość rekonstrukcji zwiększa się, lecz przy następnych konfiguracjach błąd powiększa się ze względu na przejaśnienie obrazu. Wartości te są zgodne z wyglądem obrazu.

* 1. Zmienna liczba skanów od 90 do 720 z krokiem 90
     1. Obrazy

|  |  |  |  |
| --- | --- | --- | --- |
| Liczba skanów = 90 | Liczba skanów = 180 | | Liczba skanów = 270 |
| Liczba skanów = 360 | Liczba skanów = 450 | | Liczba skanów = 540 |
| Liczba skanów = 630 | | Liczba skanów = 720 | |

* + 1. Wykres RMSE
    2. Wnioski

Zwiększenie liczby skanów powoduje polepszenie jakości obrazu, lecz konfiguracje powyżej 270 nie powodują zauważalnej poprawy jakości. Zgadza się to z wartościami RMSE, które maleją nieliniowo.

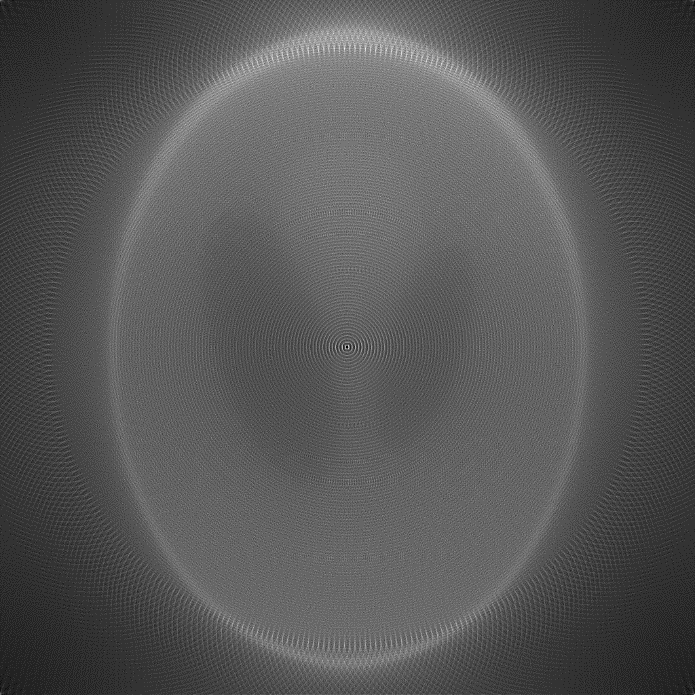
* 1. Zmienna rozpiętość wachlarza od 45° do 270° z krokiem 45°
     1. Obrazy

|  |  |  |
| --- | --- | --- |
| Rozpiętość detektorów = 45 | Rozpiętość detektorów = 90 | Rozpiętość detektorów = 135 |
| Rozpiętość detektorów = 180 | Rozpiętość detektorów = 225 | Rozpiętość detektorów = 270 |

* + 1. Wykres RMSE
    2. Wnioski

Początkowo zwiększanie rozpiętości wachlarza detektorów powoduje zdecydowane polepszenie jakości obrazu, jednakże po przekroczeniu 180° jakość zaczyna się pogarszać ze względu na zbyt duże przerwy pomiędzy liniami rysowanymi przez detektory. Ogólny trend RMSE zgadza się z subiektywną oceną obrazów, lecz wartości RMSE są względnie niskie nawet przy bardzo źle odtworzonych obrazach, co z kolei nie zgadza się z subiektywnymi oczekiwaniami.

* 1. Wpływ filtrowania na jakość dwóch wybranych obrazów
     1. Shepp\_Logan



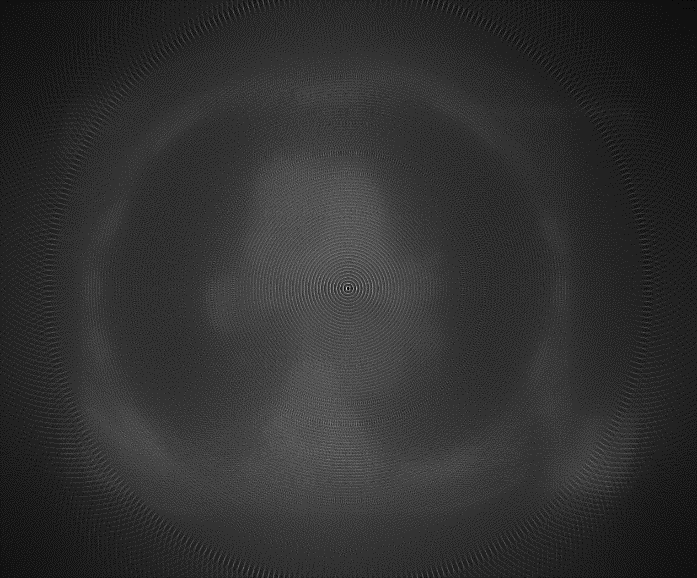
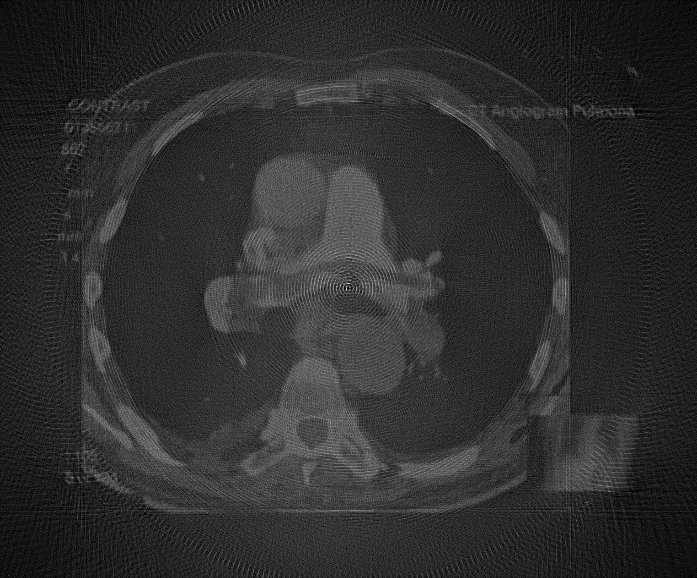
1.1. Obraz niefiltrowany

RMSE: 71,54

1.2. Obraz filtrowany

RMSE: 45,35

* + 1. SADDLE\_PE-large



2.2. Obraz filtrowany

RMSE: 57,18

2.1. Obraz filtrowany

RMSE: 62,78

* + 1. Wnioski

Różnica w jakości między obrazami filtrowanymi a niefiltrowanymi jest łatwo zauważalna. Dzięki filtrowaniu dochodzi do znacznej poprawy ostrości obrazu. Jest to również odzwierciedlone poprzez RMSE, jednak różnica wartości w przypadku obrazu Shepp\_logan jest większa ze względu na dodatkowe poprawienie jasności obrazu.