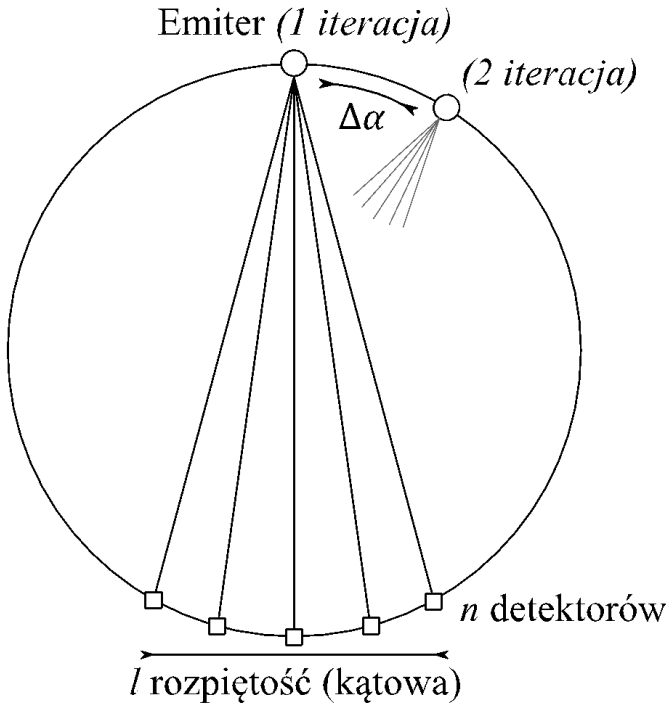
Informatyka w medycynie - Tomograf - Raport

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.



2) Zastosowany język programowania oraz dodatkowe biblioteki:

a) Język: Python
b) Dodatkowe niestandardowe biblioteki:

i) numpy – operacje na wektorach i macierzach
ii) skimage (scikit-image w pip) – wczytywanie obrazka
iii) pydicom – obsługa plików dicom
iv) tkinter - GUI
v) ttkthemes - GUI
vi) tkcalendar - GUI

3) Opis głównych części programu

a) 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
```

```
      else:
        ep += ep_inc_b
      y += y_inc
  try:
    a = sum/n # średnia jasność pikseli w rysowanej linii
  except:
    a = 0 # Uroki okręgu opisanego ¯\_(ツ)_/¯
  return a
```

b) Filtrowanie sinogramu, zastosowany rozmiar maski

```
def filtr(sinogram,t,img_label, animating = False): # filtr splotowy
  t.set_label_progress("Filtrowanie")
  result = []
  for i in range(1,11): # tworzymy połowę maski
    if i%2==0:
      result.append(0)
    else:
      result.append(-4/pow(math.pi,2)/pow(i,2))
  res2 = result.copy()
  result.reverse() # wykorzystujemy odbicie lustrzane
  kernel = result + [1] + res2 # tworzymy maskę o rozmiarze 21
  for i in range(sinogram.shape[0]):
    t.progress((i/sinogram.shape[0])*100)
    sinogram[i, :] = np.convolve(sinogram[i, :], kernel, mode="same") # wykonujemy splot
    if animating:
      t.showImage(normalize(sinogram),img_label,flag=True)
```

c) Ustalanie jakości poszczególnych punktów obrazu wynikowego oraz jego przetwarzanie końcowe (uśrednianie, normalizacja)

Uśrednianie jest wykonywane w trakcie rysowania linii algorytmem Bresenhama, zaprezentowanego powyżej.

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

```
def inverseRadonTransform(sinogram, img,t,img_label, emitterRange = 180, numOfDetectors = 180, numOfScans = 180, alphaShift = 2, animating = False, animationInterval=20):
  t.set_label_progress("Odwrotna Transformata Radona")
  alphaShift = math.radians(alphaShift)
  center = (img.shape[0]//2, img.shape[1]//2)
  R = max(center) * math.sqrt(2)
  alpha = (2*math.pi*90)/360 # alpha emitera, zaczynamy od 90 stopni
  phi = (2*math.pi*emitterRange)/360 # kąt rozwarcia dla detektorów
  reconstructed = np.zeros(shape=img.shape)
  for scan in range(numOfScans): # dla każdego skanu (wzmocnienia)
    t.progress((scan/numOfScans)*100)
    # tutaj dla wzyznaczanych współrzędnych odnoszę się do środka, aby nie robić potem przesunięć
    xe = center[0] + round(R * math.cos(alpha))
    ye = center[1] - round(R * math.sin(alpha))
    for det in range(numOfDetectors): # dla każdego detektora
      xd = center[0] + round(R * math.cos(alpha+math.pi-phi/2 + det*phi/(numOfDetectors-1)))
      yd = center[1] - round(R * math.sin(alpha+math.pi-phi/2 + det*phi/(numOfDetectors-1)))
      coords = inverseBresenham(img.shape[0], img.shape[1],xe,ye,xd,yd) # wyznaczamy linię między emiterem a detektorem
      reconstructed[tuple(np.transpose(coords))] += sinogram[scan][det] # wzmacniamy piksele wyznaczonej linii o zapisaną wcześniej średnią
    alpha += alphaShift # wykonujemy przesunięcie
    if animating and scan % animationInterval == 0: # sekcja animacji
      t.showImage(normalize(reconstructed),img_label)
      mse, rmse = calcRMSE(img,normalize(reconstructed))
      t.set_label_progress(f"Odwrotna Transformata Radona\nRMSE: {rmse:.4f}")
  mse, rmse = calcRMSE(img,normalize(reconstructed))
  t.set_label_progress(f"Odwrotna Transformata Radona\nRMSE: {rmse:.4f}")
```

```
  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
```

d) 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)
```

e) 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)
```

f) Zapis plików DICOM

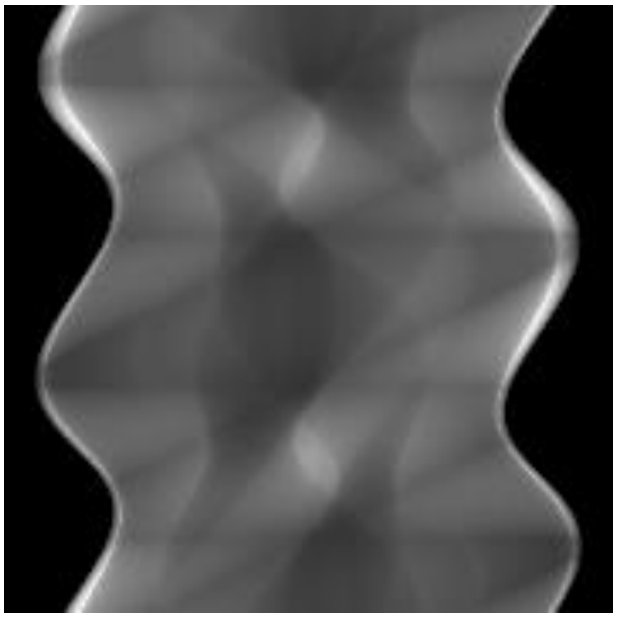
```
 1. def jpg_to_dcm(reconstructed,name="BRAK",patient_id="0",date="",comment="BRAK"): # zapisywanie obrazu do
dicom
 2.   # Meta tagi
 3.   file_meta = dic.dataset.FileMetaDataset()
 4.   file_meta.MediaStorageSOPClassUID = dic.uid.UID('1.2.840.10008.5.1.4.1.1.2')
 5.   file_meta.MediaStorageSOPInstanceUID = 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)
```

4) Przykład działania programu dla dwóch obrazków

a) Shepp_Logan

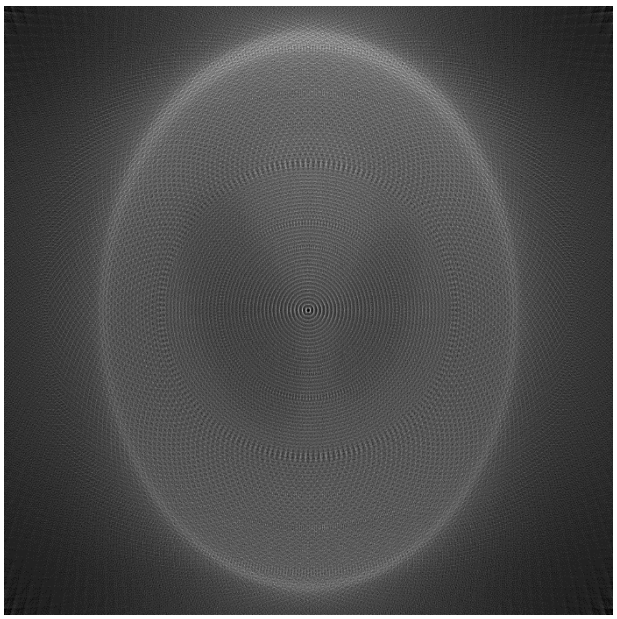
1.1. Obraz oryginalny



1.2. Sinogram

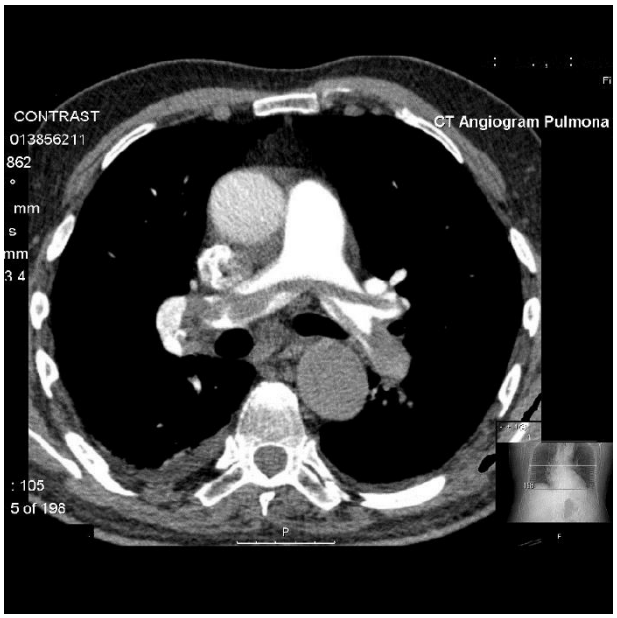
1.3. Filtrowany sinogram

1.4. Obraz zrekonstruowany

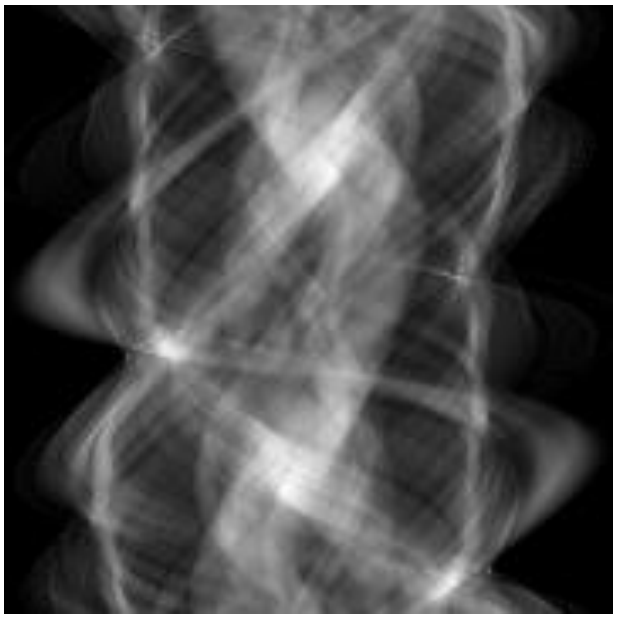


1.5. Obraz zrekonstruowany bez filtrowania

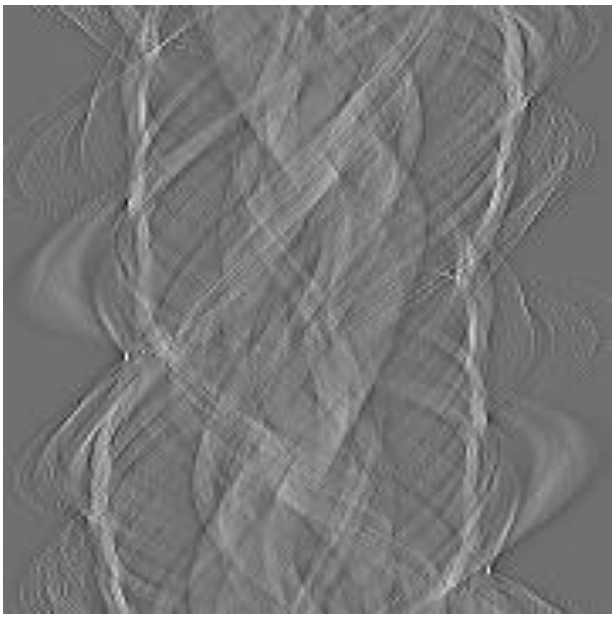
b) SADDLE_PE-large



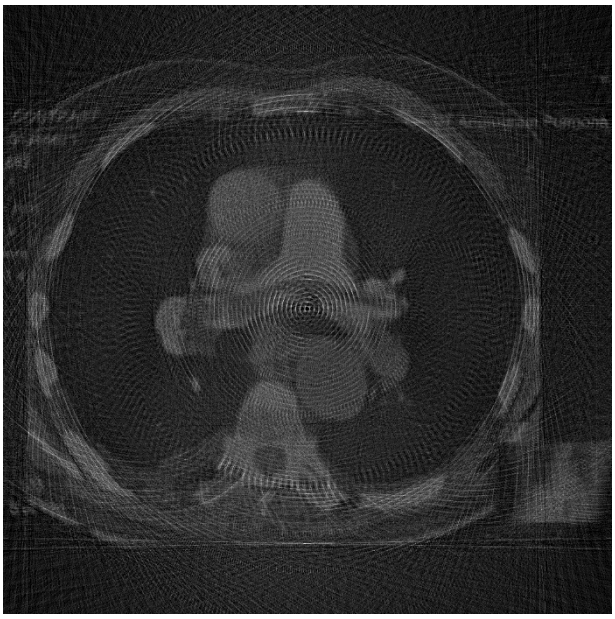
2.2. Obraz oryginalny



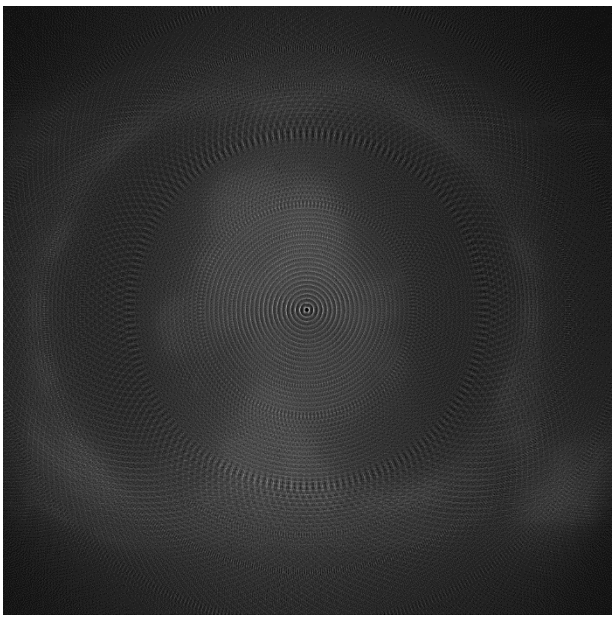
2.2. Sinogram



2.3. Filtrowany sinogram



2.4. Zrekonstruowany obraz



2.5. Zrekonstruowany obraz bez filtrowania

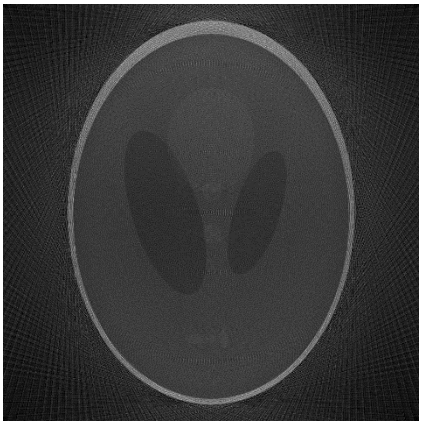
5) Wyniki eksperymentów sprawdzających wpływ poszczególnych parametrów na jakość obrazu wynikowego

a) Eksperymenty zostały przeprowadzone na obrazie Shepp_Logan, domyślne parametry ustawione na: 180 detektorów, 180 skanów, rozpiętość wachlarza 180°.
b) Zmienna liczba detektorów od 90 do 720 z krokiem 90

i) Obrazy

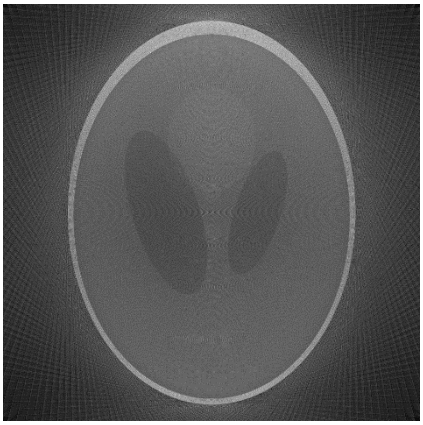
Liczba detektorów = 90

Liczba detektorów = 180

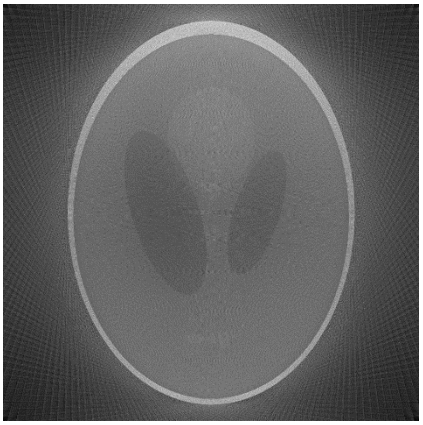


Liczba detektorów = 270

Liczba detektorów = 360



Liczba detektorów = 450

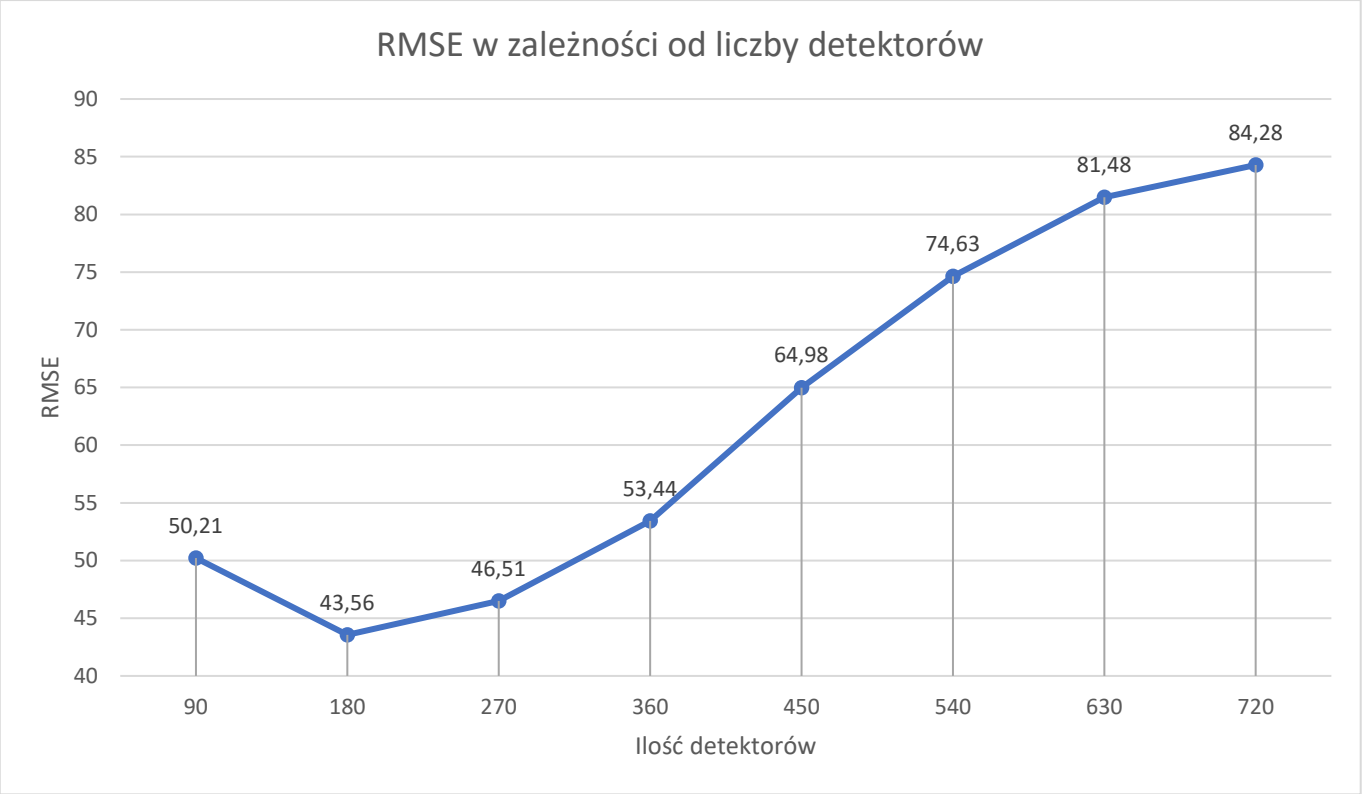


Liczba detektorów = 540

Liczba detektorów = 630

Liczba detektorów = 720

ii) Wykres RMSE



iii) 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.

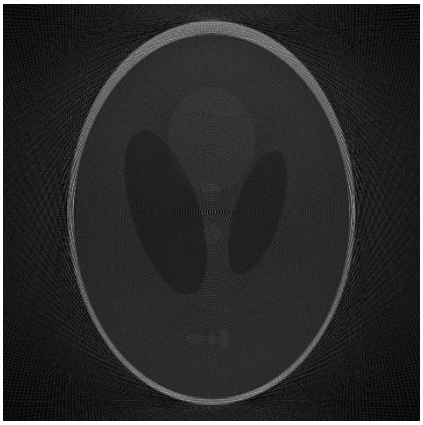
c) Zmienna liczba skanów od 90 do 720 z krokiem 90

i) Obrazy

Liczba skanów = 90

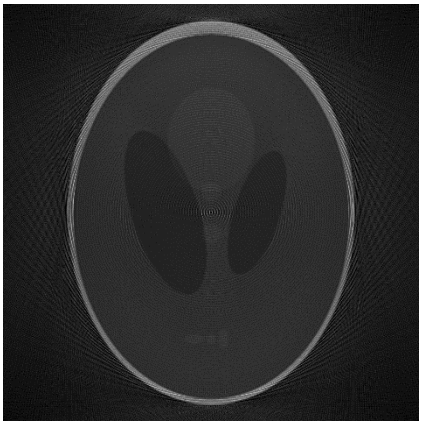
Liczba skanów = 180

Liczba skanów = 270

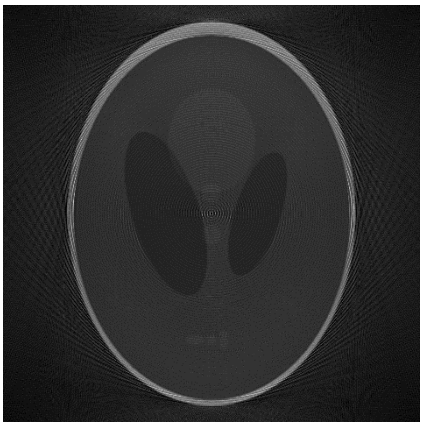


Liczba skanów = 360

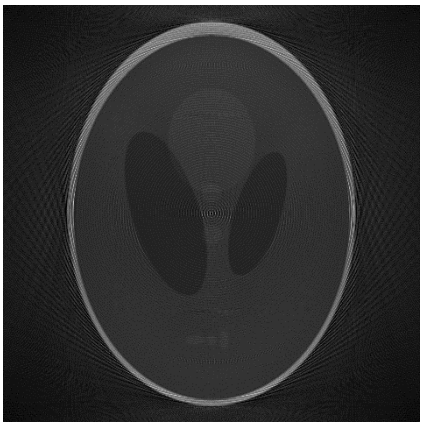
Liczba skanów = 450



Liczba skanów = 540

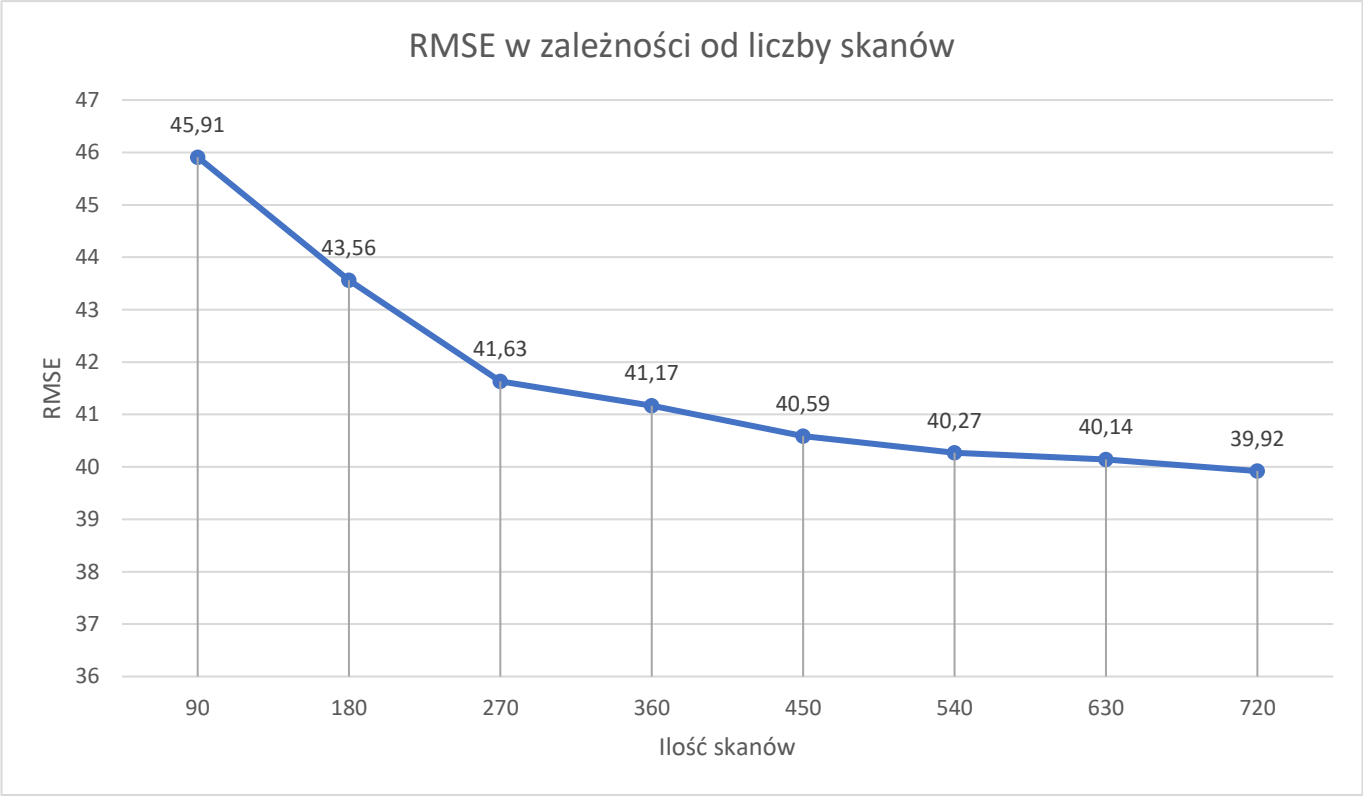


Liczba skanów = 630



Liczba skanów = 720

ii) Wykres RMSE

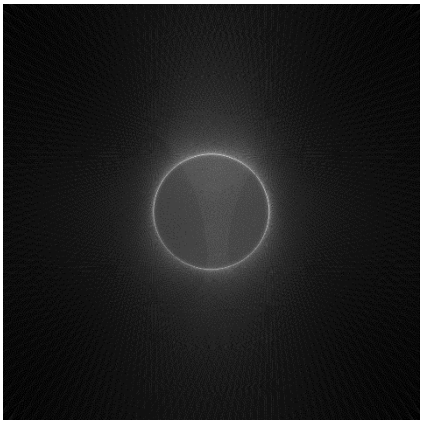


iii) Wnioski

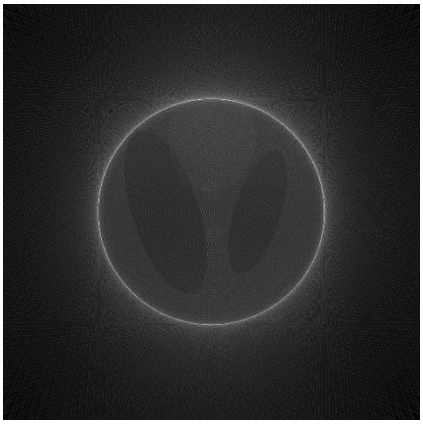
Zwiększenie liczby skanów powoduje polepszenie jakości obrazu, lecz konfiguracje powyżej 270 nie powodują zauważalnej poprawy jakości. Zgadza się to z wartościami RMSE, które maleją nieliniowo.

d) Zmienna rozpiętość wachlarza od 45° do 270° z krokiem 45°

i) Obrazy



Rozpiętość detektorów = 45



Rozpiętość detektorów = 90

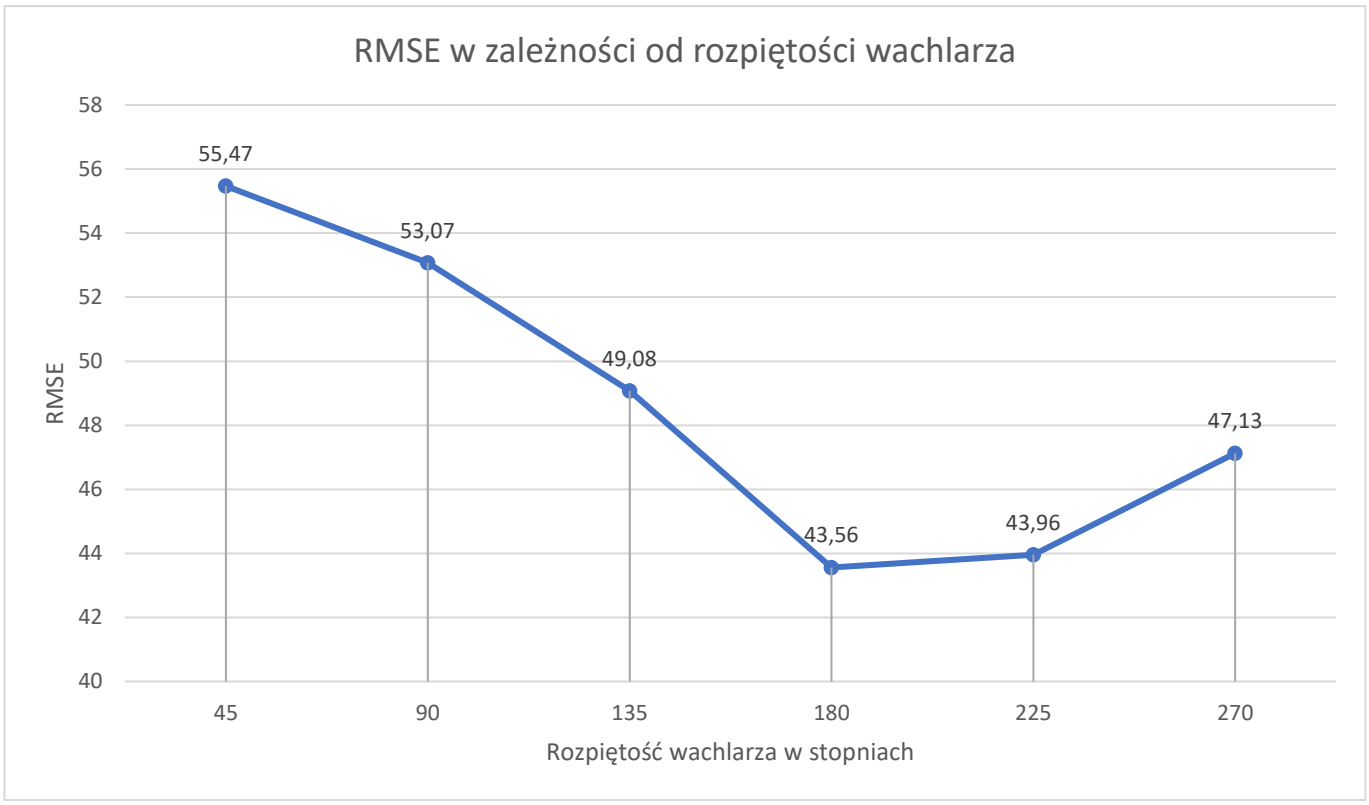
Rozpiętość detektorów = 135

Rozpiętość detektorów = 180

Rozpiętość detektorów = 225

Rozpiętość detektorów = 270

ii) Wykres RMSE



iii) Wnioski

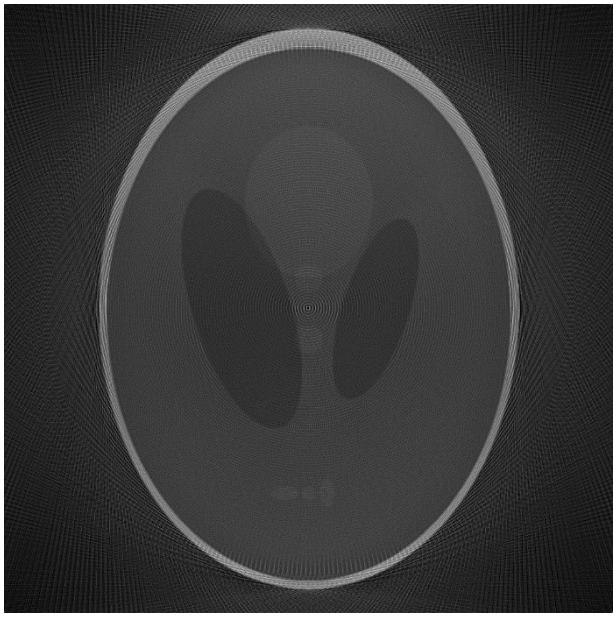
Początkowo zwiększanie rozpiętości wachlarza detektorów powoduje zdecydowane polepszenie jakości obrazu, jednakże po przekroczeniu 180° jakość zaczyna się pogarszać ze względu na zbyt duże przerwy pomiędzy liniami rysowanymi przez detektory. Ogólny trend RMSE zgadza się z subiektywną oceną obrazów, lecz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

e) Wpływ filtrowania na jakość dwóch wybranych obrazów

i) Shepp_Logan

1.*1. Obraz niefiltrowany*

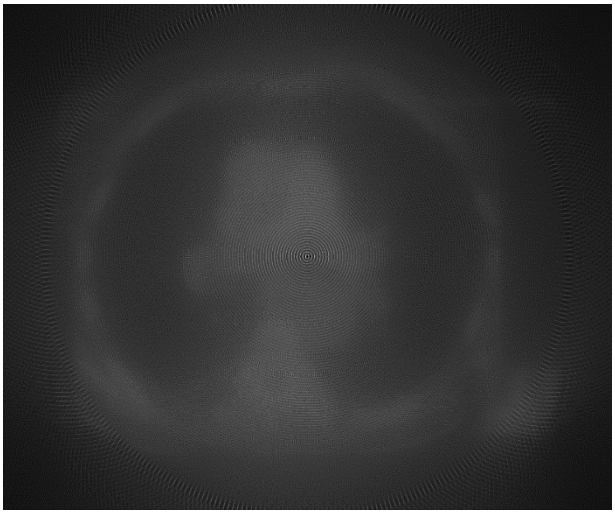
RMSE: 71,54



1.2. Obraz filtrowany

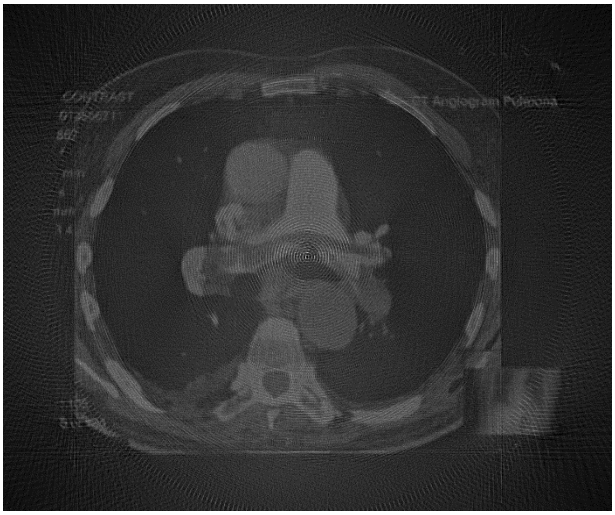
RMSE: 45,35

ii) SADDLE_PE-large



2.1. Obraz filtrowany

RMSE: 62,78



2.2. Obraz filtrowany

RMSE: 57,18

iii) 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.