Dawid Królak 145383
Grzegorz Piątek 145370

28.03.2022

Symulacja Tomografu Komputerowego

SPECYFIKACJA

1. Równoległy model tomografu.
2. Język programowania: python.
3. Użyte biblioteki:
    - ➢ Streamlit
    - ➢ os
    - ➢ PIL
    - ➢ numpy
    - ➢ math
    - ➢ Algorithms
    - ➢ matplotlib.pyplot
    - ➢ copy

GŁÓWNE FUNKCJE PROGRAMU

1. Pozyskiwanie odczytów dla poszczególnych detektorów

Sinogram jest generowany poprzez wykonanie transformaty radona na obrazie dla każdego kąta pomiarowego tomografu. Dla jednego kąta pomiarowego pozyskiwane są pozycje emiterów i detektorów, a następnie, za pomocą algorytmu Bresenhama wyznaczane są pozycje pikseli leżących na liniach łączących odpowiednie pary tych urządzeń. Dla każdej tak wyznaczonej linii, sumowane są wartości ze skali szarości pikseli na nich leżących. Ostatnim krokiem jest normalizacja wynikowego wektora, który tworzy już kolumnę w gotowym obrazie sinogramu:

```python
def calcSinogramData(self):
    results = np.zeros((self.scanCount, self.devicesCount))

    for i, angle in enumerate(self.measurementAngles):
        results[i] = self.radonTransform(angle)

    self.transposedSinogram = np.transpose(results)
```

```python
def radonTransform(self, angle):
    emittersCoords = self.getEmittersCoords(angle)      You, 2 weeks ago • Write Bresenham al
    detectorsCoords = self.getDetectorsCoords(angle)
    lines = self.getLinesBetweenDevices(emittersCoords, detectorsCoords)

    result = normalizeArray(np.array([np.sum((self.image[tuple(line)])) for line in lines]))

    return result
```

```python
def getEmittersCoords(self, angle):
    return self.getDevicesCoords(angle)

def getDetectorsCoords(self, angle):
    return self.getDevicesCoords(angle+180)[::-1]

def getDevicesCoords(self, angle):
    devicesAngles = np.linspace(0, self.rangeAngle, self.devicesCount) + radians(angle)
    centerX, centerY = self.center
    devicesX = (self.radius * np.cos(devicesAngles) - centerX).astype(int)
    devicesY = (self.radius * np.sin(devicesAngles) - centerY).astype(int)
    devicesCoords = list(zip(devicesX, devicesY))
```

```python
def getLinesBetweenDevices(self, emittersCoords, detectorsCoords):
    lines = [list(bresenham(emitterCoords, detectorCoords))
                for emitterCoords, detectorCoords
                in zip(emittersCoords, detectorsCoords)]      You
```

2. Filtrowanie sinogramu, zastosowany rozmiar maski (wymaganie na 5.0),

3. Ustalanie jasności poszczególnych punktów obrazu wynikowego oraz jego przetwarzanie końcowe (np. uśrednianie, normalizacja),

   Jasność poszczególnych punktów na obrazie wynikowym jest brana na podstawie danych z sinogramu. Do wartości każdego punktu na linii między daną parą emiterów i detektorów jest dodawana wartość z odpowiedniej kolumny i wiersza sinogramu. Tak zzsumowany obraz musi być następnie znormalizowany, więc jednocześnie wypełniana jest tablica, która zawiera dane o tym ile razy dany piksel był przecięty linią, co później jest wykorzystywane przy skalowaniu wartości pikseli.

```
def calcResultData(self):      You, 3 days ago • Inverse radon transform works …
    for i, angle in enumerate(self.measurementAngles):
        self.inverseRadonTransform(angle)

    self.normalizationMatrix[self.normalizationMatrix == 0] = 1
    self.resultImage = normalizeArray(self.resultImage / self.normalizationMatrix)
```

```
def inverseRadonTransform(self, angle):
    print(angle)
    emittersCoords = self.getEmittersCoords(angle)
    detectorsCoords = self.getDetectorsCoords(angle)
    lines = self.getLinesBetweenDevices(emittersCoords, detectorsCoords)

    for i, line in enumerate(lines):
        for point in np.transpose(line):
            if(int(angle/(180/self.scanCount)) < self.scanCount):
                self.resultImage[point[0], point[1]] += np.transpose(self.transposedSinogram)[int(angle/(180/self.scanCount)),i]
                self.normalizationMatrix[point[0], point[1]] += 1
```

4. Wyznaczanie wartości miary RMSE na podstawie obrazu źródłowego oraz wynikowego (wymaganie na 5.0),

5. Odczyt i zapis plików DICOM (wymagania na 4.0).

def write(self, patient_data, img):

```
img_converted = img_as_ubyte(rescale_intensity( img, out_range=(0.0, 1.0)))

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.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(self.filename, write_like_original=False)

def read(self, filename):
    if path.exists(filename):
        self.filename = filename
        self.patientData = dcmread(filename)
    else:
        raise IOError(f"Nie ma takiego pliku: {filename}!")
```