Symulator tomografu komputerowego

Maciej Handke 146549
Patryk Samelak 146535

1 Zastosowany model tomografu

W aplikacji zastosowano równoległy model tomografu

2 Zastosowany język programowania oraz dodatkowe biblioteki

Aplikacja została napisana w języku Python, wykorzystane zostały następujące biblioteki:

- streamlit
- numpy
- pydicom
- skimage

3 Opis głównych funkcji

3.1 Pozyskiwanie odczytów dla poszczególnych detektorów

```
def detector_points(self, angles, spread_size, detectors_number,
                    circle_radius, circle_center):
    self.angles = angles
    self.spread_size = spread_size
    self.detectors_number = detectors_number
    self.circle_radius = circle_radius
    self.circle_center = circle_center
    self.angle_shift = math.radians(self.angles - self.spread_size / 2)
    self.angle_range = math.radians(self.spread_size)
    self.all_angles = np.linspace(0, self.angle_range,
                      self.detectors_number) + self.angle_shift
    self.points = self.angle_points(self.all_angles, self.circle_center,
                  self.circle_radius)

    return self.points

def emitter_points(self, angles, spread_size, detectors_number,
                   circle_radius, circle_center):
    self.angles = angles
    self.spread_size = spread_size
    self.detectors_number = detectors_number
    self.circle_radius = circle_radius
    self.circle_center = circle_center
    self.angle_shift = math.radians(self.angles - self.spread_size / 2 +
                                    180)
    self.angle_range = math.radians(self.spread_size)
    self.all_angles = np.linspace(0, self.angle_range,
                      self.detectors_number) + self.angle_shift
    self.points = self.angle_points(self.all_angles, self.circle_center,
                  self.circle_radius)

    return self.points
```

```
def angle_points(self, angles, circle_center, circle_radius):
    self.angles = angles
    self.circle_center = circle_center
    self.circle_radius = circle_radius
    self.circle_x, self.circle_y = self.circle_center
    self.x = self.circle_radius * np.cos(self.angles) - self.circle_x
    self.y = self.circle_radius * np.sin(self.angles) - self.circle_y
    self.points = np.array(list(zip(self.x, self.y)))

    return self.points
```

3.2 Ustalanie jasności poszczególnych punktów obrazu wynikowego oraz jego przetwarzanie końcowe

```
def tomograph_reconstruction(self, detectors, emitters, file,
                             output_image):
    self.detectors = detectors
    self.emitters = emitters
    self.file = file
    self.output_image = output_image
    self.sinogram_row = []

    for i in range(len(self.detectors)):
        self.line = draw.line_nd(self.emitters[len(self.emitters) - 1 - i],
                    self.detectors[i])
        self.sinogram = np.average(self.file[self.line])
        self.sinogram_row.append(self.sinogram)
        self.output_image[self.line] += self.sinogram

    return self.sinogram_row
```

```
def normalization(data):
    return (data - np.min(data)) / (np.max(data) - np.min(data))
```

3.3 Odczyt i zapis plików DICOM

```
def dicom_open(self, file_path):
    self.file_path = file_path
    self.file = dcmread(self.file_path)
    self.keys = {x for x in dir(self.file) if x[0].isupper()} -
{'PixelData'}
    self.meta = {x: getattr(self.file, x) for x in self.keys}
    self.image = self.file.pixel_array

    return self.image, self.meta
```

```
def dicom_save(self, file_name, image, patient_info):
    self.file_name = file_name
    self.image = image
    self.patient_info = patient_info

    self.converted_image = self.image_convert(self.image)

    self.meta = Dataset()
    self.meta.MediaStorageSOPClassUID =
                              pydicom._storage_sopclass_uids.CTImageStorage
    self.meta.MediaStorageSOPInstanceUID = pydicom.uid.generate_uid()
    self.meta.TransferSyntaxUID = pydicom.uid.ExplicitVRLittleEndian
```

```
    self.file = FileDataset(None, {}, preamble=b"\0" * 128)
    self.file.file_meta = self.meta
    self.file.is_little_endian = True
    self.file.is_implicit_VR = False

    self.file.SOPClassUID = pydicom._storage_sopclass_uids.CTImageStorage
    self.file.SOPInstanceUID = self.meta.MediaStorageSOPInstanceUID

    self.file.PatientName = self.patient_info["PatientName"]
    self.file.PatientID = self.patient_info["PatientID"]
    self.file.PatientBirthDate = self.patient_info["PatientBirthDate"]
    self.file.ImageComments = self.patient_info["ImageComments"]

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

    self.file.SamplesPerPixel = 1
    self.file.HighBit = 7
    self.file.BitsStored = 8
    self.file.BitsAllocated = 8

    self.file.ImagesInAcquisition = 1
    self.file.InstanceNumber = 1

    self.file.Rows, self.file.Columns = self.converted_image.shape
    self.file.ImageType = r"ORIGINAL\PRIMARY\AXIAL"
    self.file.PhotometricInterpretation = "MONOCHROME2"
    self.file.PixelRepresentation = 0

    pydicom.dataset.validate_file_meta(self.file.file_meta,
                                       enforce_standard=True)

    self.file.PixelData = self.converted_image.tobytes()
    self.file.save_as(self.file_name, write_like_original=False)

    return self.file_name
```

```
with st.expander("Zapisz jako plik DICOM"):
    patient_name = st.text_input("Imię i nazwisko")
    patient_id = st.text_input("ID")
    patient_birthdate = st.date_input("Data badania")
    comment = st.text_area("Komentarz")
    patient_info = {
        "PatientName": patient_name,
        "PatientID": patient_id,
        "PatientBirthDate": str(patient_birthdate),
        "ImageComments": comment
    }
    if len(patient_name) and len(patient_id):
        file_name = f'{patient_name} {patient_id}.dcm'
    else:
        file_name = "unnamed.dcm"
    save_image = st.radio("Zapisz jako DICOM ", ["Obraz wejściowy", "Obraz
                          wyjściowy"])
    if save_image == "Obraz wyjściowy":
        if st.button("Zapisz"):
```

```
            dicom.dicom_save(file_name,
np.array(normalization(output_image[len(output_image) - 1])), patient_info)
    else:
        if st.button("Zapisz"):
            dicom.dicom_save(file_name, image, patient_info)
```

4 Sprawdzenie poprawności pliku DICOM

Poprawność utworzonych plików DICOM została zweryfikowana za pomocą **https://www.ofoct.com/viewer/dicom-viewer-online.html**

Name	Value
FileMetaInformationGroupLength	206
FileMetaInformationVersion	0, 1
MediaStorageSOPClassUID	1.2.840.10008.5.1.4.1.1.2□
MediaStorageSOPInstanceUID	1.2.826.0.1.3680043.8.498.681750642349292155523574518523268151 33
TransferSyntaxUID	1.2.840.10008.1.2.1□
ImplementationClassUID	1.2.826.0.1.3680043.8.498.1□
ImplementationVersionName	PYDICOM 2.3.0
ImageType	ORIGINAL, PRIMARY, AXIAL
SOPClassUID	1.2.840.10008.5.1.4.1.1.2□
SOPInstanceUID	1.2.826.0.1.3680043.8.498.681750642349292155523574518523268151 33
Modality	CT
PatientName	Jan Kowalski
PatientID	5d3Z9s0
PatientBirthDate	2022-02-24
StudyInstanceUID	1.2.826.0.1.3680043.8.498.114163985652710462171612958926475342 65
SeriesInstanceUID	1.2.826.0.1.3680043.8.498.153843033117788255449977926381467798 90
InstanceNumber	1
FrameOfReferenceUID	1.2.826.0.1.3680043.8.498.852468218245447579898496151913594752 91
ImagesInAcquisition	1
ImageComments	Uraz
SamplesPerPixel	1
PhotometricInterpretation	MONOCHROME2
Rows	1024
Columns	1024
BitsAllocated	8
BitsStored	8
HighBit	7
PixelRepresentation	0
PixelData	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...