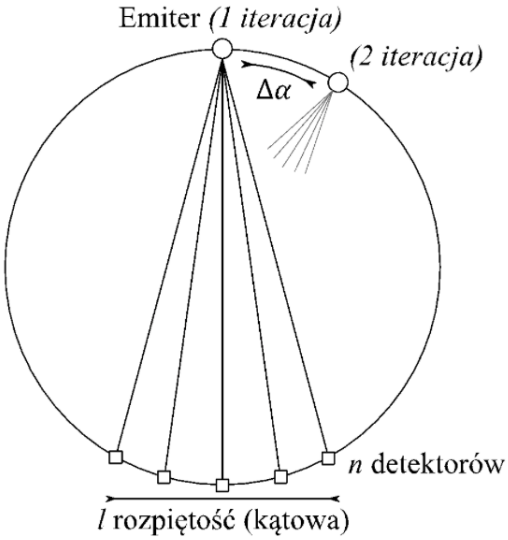
Symulator tomografu komputerowego

Przemysław Ambroży, Błażej Celmer

25 marca 2021

1 Model



rys. 1. model stożkowy

Zastosowany został model stożkowy (rys. 1.). Zakłada on wykorzystanie jednego emitera, który współpracuje ze wszystkimi detektorami, tworząc wspomniany stożek (W przeciwieństwie do modelu równoległego, gdzie każdy detektor posiada własny emiter). coś bym tu jeszcze dopisał

2 Program

2.1 Środowisko

Do zasymulowania tomografu skorzystaliśmy z języka Python w środowisku Jupyter Notebook, co pozwoliło niskim kosztem uzyskać interfejs użytkownika. Skorzystaliśmy również z bibliotek, które oferowały dodatkowe funkcjonalności, m.in.:

numpy funkcje i stałe matematyczne

matplotlib wyświetlanie grafik

ipywidgets interfejs użytkownika

pydicom odczyt i zapis plików DICOM

2.2 Opis działania

2.2.1 Sinogram

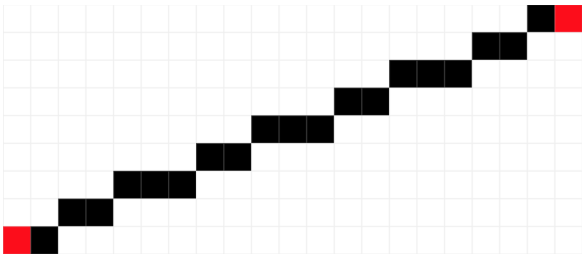
Sinogram, jest to tablica danych, gdzie każdy wiersz to jedna iteracja (pozycja emitera), a każda kolumna oznacza jeden detektor. Aby otrzymać sinogram, należy wykonać n iteracji, w których emiter wraz z detektorami będzie się stopniowo przesuwał po okręgu, aż obrócimy cały układ o 180 stopni. Podczas każdej iteracji wyznaczamy współrzędne punktów, w których znajduje się emiter oraz detektory.

Opis wyznaczana współrzędnych

Aby pobrać dane z obiektu znajdującego się między emiterem a detektorami (w naszym przypadku z obrazka), wykorzystujemy algorytm **Bresenhama**. Służy on do wyznaczenia pikseli znajdujących się między dwoma punktami. Przechodzi iteracyjnie przez wszystkie wartości na jednej osi (np. OX, jeśli obrazek jest szerszy niż wyższy). W każdym kroku decyduje, czy wykonać tylko ruch wzdłuż jednej osi (poziomo/pionowo), czy obu (na ukos).

```
for i in range(x1, x2):
    points.append((i, j))
    if e > 0:
        j += m
        e += 2 * (dy - dx)
    else:
        e += 2 * dy
```

Rysunek 1: Fragment kodu algorytmu



Rysunek 2: Przykład działania – znajdowanie pikseli między dwoma czerwonymi

Znając wszystkie piksele leżące między emiterem, a detektorem, sumujemy ich jasność i zapisujemy w odpowiednim miejscu sinogramu. Na koniec normalizujemy wszystkie dane, tak aby wartości były z przedziału od 0 do 1. W tym celu znajdujemy największą wartość (najjaśniejszy piksel) i dzielimy przez nią pozostałe.

2.2.2 Obraz wynikowy

Sinogram nie jest zrozumiały dla człowieka, dlatego należy go przekształcić. Zaczynamy od czarnego obrazu i znów przejdziemy przez n iteracji. Każdej iteracji przypisana będzie pozycja emitera wraz z detektorami (będą to te same pozycje, co podczas generowania sinogramu). Wyznaczamy linie przechodzące przez obraz między emiterem i detektorami (algorytm Bresenhama). Każdy piksel, przez który przechodzi linia będzie rozjaśniony o wartość odczytaną z sinogramu (dla danego detektora w danej iteracji). Nałożenie wszystkich linii na obrazie stworzy obraz wynikowy. to samo co wyżej, no i myślę jeszcze o jakiś obrazkach

2.3 Standard DICOM

Wyniki zapisujemy w standardzie DICOM. Czy mam opisywać co to jest za standard? Do obsługi standardu używamy biblioteki `pydicom`. W pliku, oprócz samego skanu, zapisujemy też dane pacjenta (imię i nazwisko, płeć, datę urodzenia), komentarze, datę i czas badania. Wykorzystujemy w tym celu następujące pola:

Identyfikator pola	Nazwa pola	Opis
(0008, 0023)	Content Date	data badania
(0008, 0033)	Content Time	czas badania
(0010, 0010)	Patient's Name	imię i nazwisko pacjenta
(0010, 0030)	Patient's Birth Date	data urodzenia pacjenta
(0010, 0040)	Patient's Sex	płeć pacjenta
(0010, 4000)	Patient Comments	komentarze
(0028, 0010)	Rows	liczba wierszy obrazka (wysokość)
(0028, 0011)	Columns	liczba kolumn obrazka (szerokość)
(7fe0, 0010)	Pixel Data	obrazek (piksele)

Rysunek 3: Niektore z wykorzystanych pól

Fragment kodu odpowiedzialnego za zapis pokazuje rysunek 4. Do odczytu pliku skorzystaliśmy z funkcji `pydicom.dcmread(filename)`, która pozwala następnie na dostęp do wszystkich pól z pliku. Czy dodać kod odczytu pliku?

```
ds = FileDataset(filename, {}, file_meta=file_meta, preamble=b"\0" * 128)
ds.PatientName = name
ds.PatientSex = sex
ds.PatientBirthDate = birth.strftime('%Y%m%d')
ds.PatientComments = comment

ds.ContentDate = date.strftime('%Y%m%d')
ds.ContentTime = date.strftime('%H%M%S.%f')

# 1 lub 3 - ile kolorow na obrazie
ds.SamplesPerPixel = 1
# interpretacja (MONOCHROME, RGB, HSV, ...)
ds.PhotometricInterpretation = "MONOCHROME2"
# 0 - unsigned int, 1 - U2
ds.PixelRepresentation = 0
# najwyzszy bit
ds.HighBit = 15
# ilosc bitow na piksel
ds.BitsStored = 16
# ilosc bitow zaalokowanych na piksel
ds.BitsAllocated = 16
# minimalna wartosc piksela - 0
ds.SmallestImagePixelValue = str.encode('\x00\x00')
# maksymalna wartosc piksela - ffff?
ds.LargestImagePixelValue = str.encode('\xff\xff')
# kolumny, wiersze i dane
ds.Columns = image.shape[1]
ds.Rows = image.shape[0]
ds.PixelData = (image * 65535).astype('uint16').tobytes()

ds = correct_ambiguous_vr(ds, True)
ds.save_as(filename, write_like_original=False)
```

Rysunek 4: Fragment kodu – zapis do pliku DICOM