Symulator tomografu komputerowego

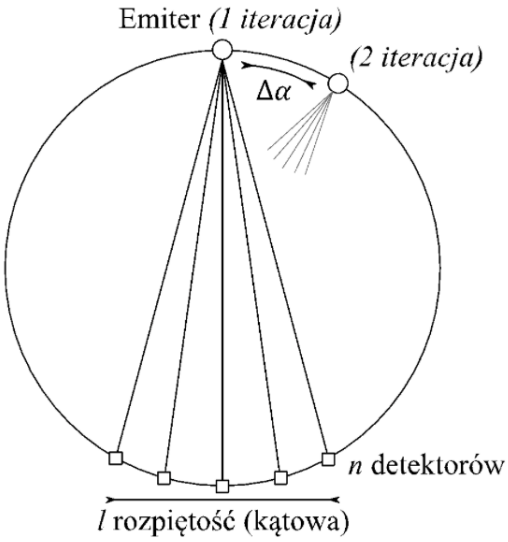
Przemysław Ambroży, Błażej Celmer

25 marca 2021

1 Wstęp

Ćwiczenie zakłada zasymulowanie tomografu komputerowego. Korzystając z obrazów i zdjęć postaramy się wygenerować ich sinogram, który reprezentuje dane odczytane z detektorów tomografu. Następnie, przekształcimy go do postaci wynikowej – czytelnej dla człowieka.

2 Model



Rysunek 1: model stożkowy

Zastosowany został model stożkowy (rysunek 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).

3 Program

3.1 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

3.2 Opis działania

3.2.1 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 E oraz detektory D_i.

$$E = [x_E, y_E] \qquad D_i = [x_{D_i}, y_{D_i}]$$
$$x_E = r \times \cos\alpha \qquad x_{D_i} = r \times \cos\left(\alpha + \pi - \frac{\phi}{2} + i \times \frac{\phi}{n-1}\right)$$
$$y_E = r \times \sin\alpha \qquad y_{D_i} = r \times \sin\left(\alpha + \pi - \frac{\phi}{2} + i \times \frac{\phi}{n-1}\right)$$

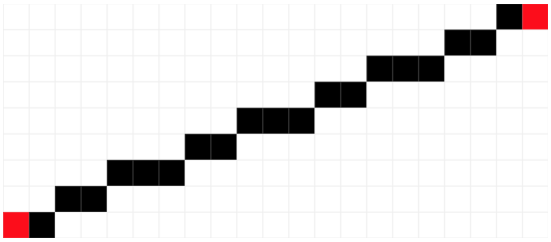
gdzie:
r – połowa przekątnej obrazu,
α – kąt przesunięcia systemu względem obrazu,
ϕ – rozpiętość kątowa układu detektorów.

Wyznaczone współrzędne zakładają punkt $(0, 0)$, jako środek systemu/obrazu. Należy przekształcić je do układu współrzędnych obrazu, gdzie punkt $(0, 0)$ znajduje się w jednym z narożników.

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 2: Fragment kodu algorytmu



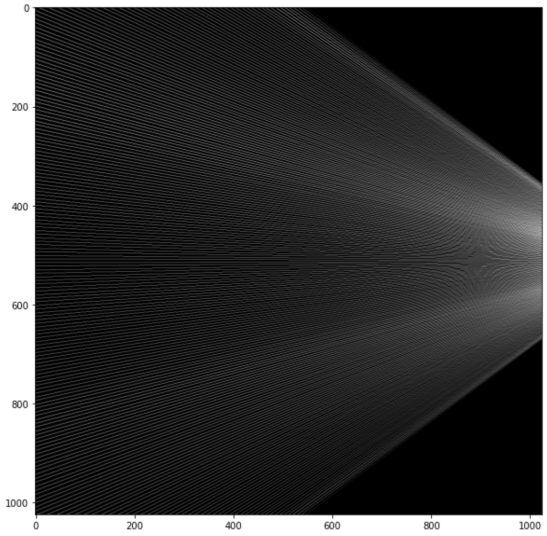
Rysunek 3: Przykład działania – znajdowanie pikseli między dwoma czerwonymi

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.

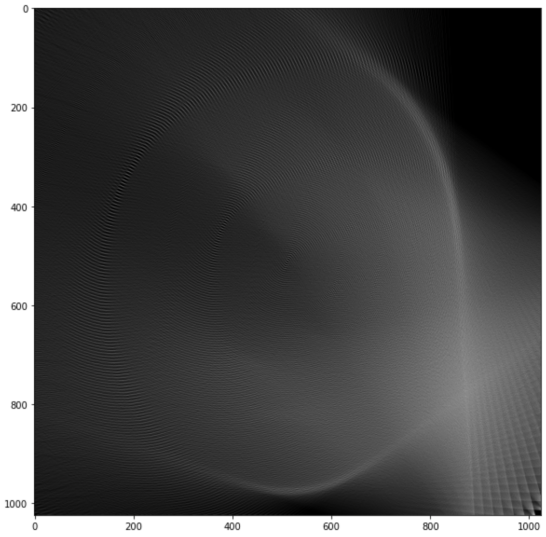
3.2.2 Obraz wynikowy

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.

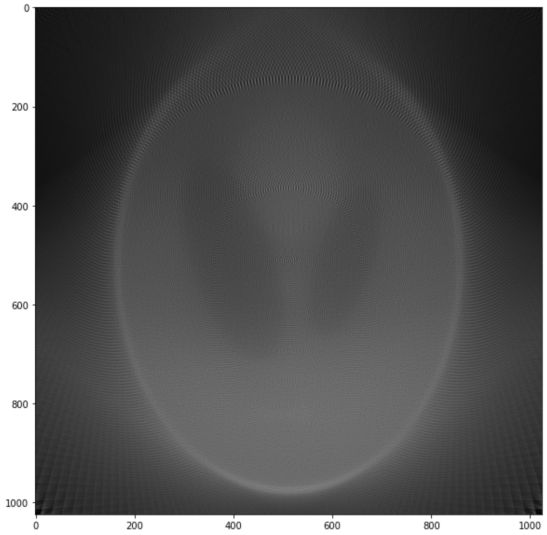
Rysunek 4: obraz wejściowy



Rysunek 5: obraz wyjściowy – 1. iteracja



Rysunek 6: obraz wyjściowy – 30. iteracja



Rysunek 7: obraz wyjściowy – 90. (ostatnia) iteracja

3.3 Standard DICOM