Informatyka w medycynie

Raport - Projekt 1: Symulator tomografu

1. Skład grupy: Michał Wiśniewski, nr 141335
2. Zastosowany model tomografu: stożkowy
3. Zastosowany język programowania oraz dodatkowe biblioteki: Python + numpy, tkinter, PIL, thread, time
4. Opis głównych funkcji programu
    i. Funkcja tworząca sinogram pobiera parametry podane przez użytkownika i inicjalizuje dwuwymiarową tabelę sinogramu oraz inne zmienne.
    ii. Pętla przechodzi przez kolejne kąty $\Delta\alpha$ i wyznacza współrzędne punktów odpowiadających emiterowi oraz dektorów z podanych wzorów, dopasowując tak by został zasymulowany obrót wokół środka obrazu.
    iii. Biorąc wyliczone współrzędne, dla każdego detektora wyznaczana jest wiązka - linia punktów algorytmu Bresenhama, na podstawie której generowana jest suma jasności piksela dla każdego detektora.
    iv. Piksele wygenerowane przy danym emiterze są zapisywane w tabeli sinogramu i na bieżąco pokazywane na ekranie.
    v. Funkcja generująca odwrotną tranformatę Radona przechodzi przez stworzoną tabelę sinogramu oraz dokonuje odwrotnych kalkulacji

```
def makeSinogram(self):
    pic = picture.input
    nDet = int(self.detectorsEntry.get())
    alpha = float(self.angleEntry.get())
    span = float(self.coneSpanEntry.get())
    spanRad = span * np.pi / 180;
    r = len(pic[0])
    lines = []
    i = 0
    sinogram=[[0 for x in range(nDet)] for y in range(int(360/alpha))]
    while i < 360:
        lines.append([])
        angleRad = i * np.pi / 180
        xe = r * np.cos(angleRad)
        ye = r * np.sin(angleRad)
        xe = int(xe) + np.floor(r / 2)
        ye = int(ye) + np.floor(r / 2)

        for d in range(0, nDet):
            xd = r * np.cos(angleRad + np.pi - spanRad / 2 + d * (spanRad / (nDet - 1)))
            yd = r * np.sin(angleRad + np.pi - spanRad / 2 + d * (spanRad / (nDet - 1)))
            xd = int(xd) + np.floor(r / 2)
            yd = int(yd) + np.floor(r / 2)

            line = bresenham(xe, ye, xd, yd)
            pixel = np.float(0)
            counter = int(0)
            for [x, y] in line:
                if x >= 0 and y >= 0 and x < r and y < r:
                    pixel += float(pic[x, y])
                    counter += 1
            if counter > 0:
                sinogram[int(i/alpha)][d]=(pixel / counter)
            else:
                sinogram[int(i/alpha)][d]=0
            lines[-1].append([xe, ye, xd, yd])
        i += alpha
        time.sleep((100-self.speedSlider.get())/1000)
        if self.stepsVar.get() == 1:
            self.setSinogramOutput(sinogram)
    self.setSinogramOutput(sinogram)
    start_new_thread(self.makePicture, (sinogram,lines,pic))
    return sinogram, lines
```

Figure 1 Transformata Radona

```
def makeOutputPicture(self, sinogram, lines, pic):
    self.outputCanvas.create_rectangle(0, 0, self.outputCanvas.winfo_width(), self.outputCanvas.winfo_height(), fill="black")
    OutPic = np.zeros([np.shape(pic)[0], np.shape(pic)[1]])
    OutPicsums = np.zeros([np.shape(pic)[0],np.shape(pic)[1]])
    sx = np.shape(sinogram)[0]
    sy = np.shape(sinogram)[1]
    counter = np.zeros([np.shape(pic)[0], np.shape(pic)[1]])
    for i in range(0, sx):
        for j in range(0, sy):
            x0, y0, x1, y1 = lines[i][j]
            line = bresenham(x0, y0, x1, y1)
            for [x, y] in line:
                if x >= 0 and y >= 0 and x < np.shape(pic)[0] and y < np.shape(pic)[1]:
                    OutPicsums[x][y]+=sinogram[i][j]
                    counter[x][y]+=1
                    OutPic[x][y]=OutPicsums[x][y]
                    OutPic[x][y]=OutPicsums[x][y]/counter[x][y]
        time.sleep((100-self.speedSlider.get())/1000)
        if self.stepsVar.get()==1:
            self.setOutputPicture(OutPic)
    self.setOutputPicture(OutPic)
    return OutPic
```

Figure 2 Odwrotna transformata Radona