Зміст

[Розділ 1. Постановка задачі 3](#_heading=h.gjdgxs)

[Розділ 2. Теоретичні відомості 4](#_heading=h.30j0zll)

[Розділ 3. Розрахунки за варіантом 5](#_heading=h.1fob9te)

[Розділ 4. Лістинг програми 6](#_heading=h.2et92p0)

[Розділ 5. Результати роботи програмного застосунку 9](#_heading=h.tyjcwt)

# Розділ 1. Постановка задачі

1. Відповідно до свого варіанту розрахувати тривимірні координати пікселя зображення, який розташований в заданому рядку та стовпчику.
2. Розробити програмний застосунок для визначення просторового положення піксельних даних томографічних зображень та визначення тривимірних координат пікселя зображення, який розташований в заданому рядку та стовпчику; розміри рограмного вікна застосунку (графічна частина) мають бути достатніми для візуалізації всіх даних, що передбачену в завданні до РГР.
3. Реалізувати вивід текстового повідомлення із тривимірними координатами пікселя зображення, який розташований в заданому відповідно до варіанта рядку та стовпчика.
4. Реалізувати у вікні програмного застосунку графічне відображення координатних осей із початком координат в ізлцентрі сканера; границь об’єму, який визначає отримана під час дослідження серії томографічних зображень; площини томографічного зображення, що визначене в просторі відовідно до варіанту, та положення пікселя, який розташований в заданому рядку та стовпчику цього зображення.
5. Скласти і захистити звіт по робот.

Дані для варінату

| **Номер варінта** | **0028,0010 Rows** | **0028,0011 Columns** | **0028,0030 Pixel Spacing** | **0018,0050 Slice Thickness** | **0018,0088 Spacing Between Slices** | **Координати пікселя в площині зображення** |
| --- | --- | --- | --- | --- | --- | --- |
| 19 | 512 | 512 | 0.744000\0.744000 | 3.00 | 3.00 | (462;320) |

| **Номер варінту** | **0020,0032 Image Position, шуканий зріз (25й)** | **0020,0037 Image Orientation, шуканий зріз** |
| --- | --- | --- |
| 19 | 103.467\-190.625\1449.592 | 0.0\1.0\0.0\0.0\0.0\-1.0 |

| **Номер варінта** | **0020,0032 Image Position, 1й зріз** | **0020,003\ Image Orientation, 1й зріз** | **Кількість зображень в серії** |
| --- | --- | --- | --- |
| 19 | 163.467\-190.625\1449.592 | 0.0\1.0\0.0\0.0\0.0\-1.0 | 112 |

# Розділ 2. Теоретичні відомості

Просторове положення зображення можна визначити за двома елементами:

* *0020,0032 “Image Position”* – зберігає інформацію щодо координат лівого верхнього кута (першого пікселя) зображення в міліматрах, що допомагає визначити положення у тривимірному просторі.
* *0020,0037 “Image Orientation”* – зберігає напрямні косинуси тривіальних векторів напрямку рядків (перші три значення елемента) та стовпців (наступні три).

Вектори та , що починаються з координат пікселя, заданих в Image Position елементі, однозначно визначають площину зображення в тривимірному просторі. Наприклад, значення вектора , яке відповідає , означає, що напрямок рядків зображення точно співпадає з напрямком осі , а значення вектора , яке відповідає , означає, що напрямок стовпців зображення точно співпадає з напрямкком осі .

Для розрахунку тривимірної координати пікселя зображення *P*, який розташований в **r-м**у рядку та в c-му стовпчику, можна використати формулу:

Інформація стосовно фізичних розмірів зображення в міліметрах зберігаєтсья у елементі *0028,0030 “Pixel Spacing”*. Якщо у елемнті знаходяться значення 0.5мм для та 0.5мм для , то для довжини в 10 пікселів відповідна фізична довжина рівна 5мм. Знаючи роздільність зображення в пікселях, можна визначити його фізичні розміри (область перегляду, *FoV*).

В елементі *0018,0050 "Slice Thickness”* зберігається інформація щодо товщини томографічного зрізу в міліметрах. Проте на реальних данних відстань між зрізами може не дорівнювати товщині самих зрізів, вона може бути меньша або більша. Через це використовувати його для визначення розміру відсканованого об’єкту недоцільно.

# Розділ 3. Розрахунки за варіантом

Для розрахунку тривимірної координати пікселя зображення *P*, який розташований в r-му рядку та в c-му стовпчику, можна використати формулу:

Тривимірні координати для 1-го зрізу:

;

;

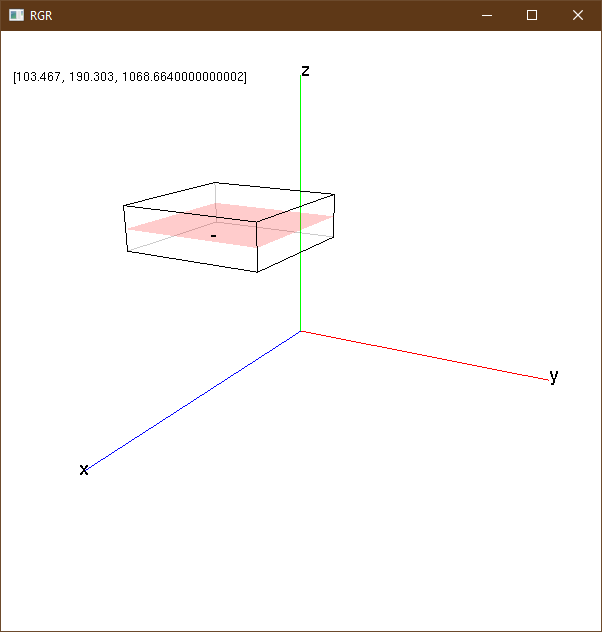
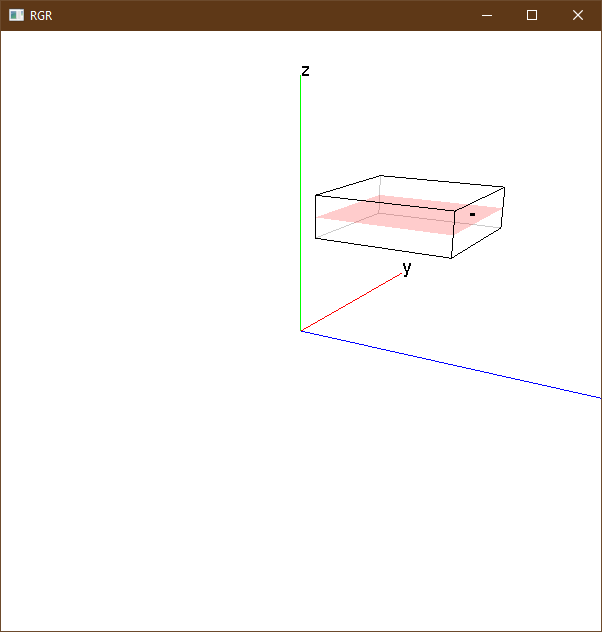
За допомогою формул для множення вектора на число та додавання векторів, розрахуємо:

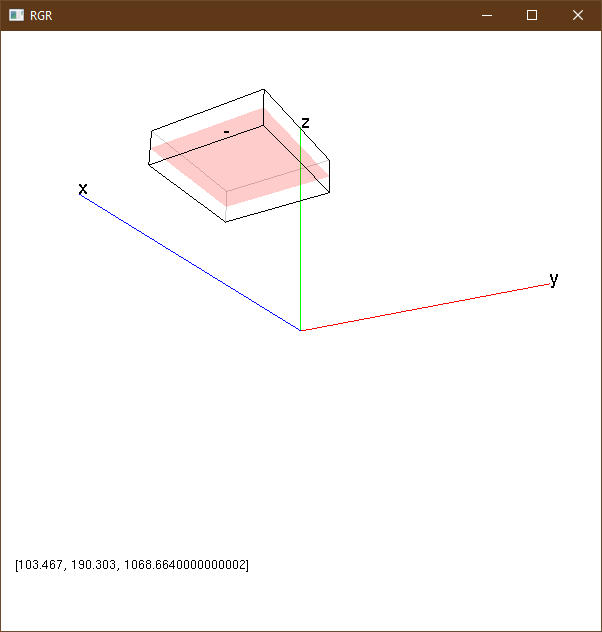
Для побудови зображення розрахуємо:

# Розділ 4. Лістинг програми

from OpenGL.GL import \*  
from OpenGL.GLU import \*  
from OpenGL.GLUT import \*  
import numpy as np  
  
  
class Image:  
 def \_\_init\_\_(self):  
 self.number\_of\_images = 112  
 self.rows = 512  
 self.columns = 512  
 self.pixel\_spacing = 0.744000  
 self.slice\_thickness = 3.0  
 self.spacing\_between\_slices = 3.0  
 self.scale = 512  
  
 self.X\_orientation = np.array([0.0, 1.0, 0.0])  
 self.Y\_orientation = np.array([0.0, 0.0, -1.0])  
 self.x\_coord, self.y\_coord = 462, 320  
  
 self.image\_position\_25 = np.array([103.467, -190.625, 1449.592])  
 self.image\_position\_1 = np.array([163.467, -190.625, 1449.592])  
 self.position\_25 = self.get\_position\_25()  
  
 self.width, self.height = 600, 600  
 self.isTextVisible = False  
 self.mouse\_on\_image = False  
  
 def init(self):  
 glClearColor(0.0, 0.0, 0.0, 0.0)  
 glClearDepth(1.0)  
 glDepthFunc(GL\_LESS)  
 glEnable(GL\_DEPTH\_TEST)  
 glShadeModel(GL\_SMOOTH)  
 glMatrixMode(GL\_PROJECTION)  
 glLoadIdentity()  
 gluPerspective(45.0, self.width / self.height, 0.1, 100.0)  
 glMatrixMode(GL\_MODELVIEW)  
  
 def get\_position\_25(self):  
 position\_25 = [  
 self.image\_position\_25[0] + self.rows \* self.X\_orientation[0] \* self.pixel\_spacing + self.columns \* self.Y\_orientation[0] \* self.pixel\_spacing,  
 self.image\_position\_25[1] + self.rows \* self.X\_orientation[1] \* self.pixel\_spacing + self.columns \* self.Y\_orientation[1] \* self.pixel\_spacing,  
 self.image\_position\_25[2] + self.rows \* self.X\_orientation[2] \* self.pixel\_spacing + self.columns \* self.Y\_orientation[2] \* self.pixel\_spacing]  
 print(position\_25)  
 return position\_25  
  
 def display(self):  
 glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT)  
 glClearColor(1.0, 1.0, 1.0, 0.0)  
  
 self.draw\_texture()  
  
 self.print\_text(0.0, 0.0, 2.0, GLUT\_BITMAP\_HELVETICA\_18, 'z')  
 self.print\_text(0.0, 2.0, 0.0, GLUT\_BITMAP\_HELVETICA\_18, 'y')  
 self.print\_text(2.5, 0.0, 0.0, GLUT\_BITMAP\_HELVETICA\_18, 'x')  
  
 self.print\_text\_2D(-40, -50, GLUT\_BITMAP\_HELVETICA\_12, str(self.position\_25))  
 glutSwapBuffers()  
  
 def print\_text\_2D(self, x, y, font, line):  
 glColor3f(0, 0, 0)  
 glPushAttrib(GL\_DEPTH\_TEST)  
 glRasterPos2d(x, y)  
 for i in line:  
 glutBitmapCharacter(font, ord(i))  
 glPopAttrib()  
  
 def print\_text(self, x, y, z, font, line):  
 glColor3f(0, 0, 0)  
 glPushAttrib(GL\_DEPTH\_TEST)  
 glRasterPos3d(x, y, z)  
 for i in line:  
 glutBitmapCharacter(font, ord(i))  
 glPopAttrib()  
  
 def draw\_texture(self):  
 glClear(GL\_COLOR\_BUFFER\_BIT)  
 glLoadIdentity()  
 glTranslatef(0.0, 0.0, -6.0)  
  
 glRotatef(-69.0, 1.0, 0.0, 0.0)  
 glRotatef(0.0, 0.0, 1.0, 0.0)  
 glRotatef(-119.0, 0.0, 0.0, 1.0)  
  
 # Оси координат  
 # ------------------------  
 glBegin(GL\_LINES)  
 # x  
 glColor3f(0.0, 0.0, 1.0)  
 glVertex3f(0.0, 0.0, 0.0)  
 glVertex3f(2.5, 0.0, 0.0)  
 # y  
 glColor3f(1.0, 0.0, 0.0)  
 glVertex3f(0.0, 0.0, 0.0)  
 glVertex3f(0.0, 2.0, 0.0)  
 # z  
 glColor3f(0.0, 1.0, 0.0)  
 glVertex3f(0.0, 0.0, 0.0)  
 glVertex3f(0.0, 0.0, 2.0)  
 glEnd()  
 # ------------------------  
  
 # Рисуем куб  
 # ------------------------  
 glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE)  
  
 # координаты точек куба  
 position\_0 = [self.image\_position\_1[0] / self.pixel\_spacing / self.scale,  
 self.image\_position\_1[1] / self.pixel\_spacing / self.scale,  
 self.image\_position\_1[2] / self.spacing\_between\_slices / self.scale]  
 position\_1 = [self.rows / self.scale + position\_0[0],  
 self.columns / self.scale + position\_0[1],  
 ((self.number\_of\_images - 1) + self.slice\_thickness) / self.rows / self.pixel\_spacing + position\_0[2]]  
  
 x0, y0, z0 = position\_0[0], position\_0[1], position\_0[2]  
 x1, y1, z1 = position\_1[0], position\_1[1], position\_1[2]  
  
 glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA)  
 glEnable(GL\_BLEND)  
 glBegin(GL\_QUADS)  
 glColor3f(0, 0, 0)  
 glVertex3f(x1, y1, z0), glVertex3f(x1, y1, z1), glVertex3f(x1, y0, z1), glVertex3f(x1, y0, z0)  
 glVertex3f(x1, y1, z0), glVertex3f(x0, y1, z0), glVertex3f(x0, y1, z1), glVertex3f(x1, y1, z1)  
 glVertex3f(x1, y0, z0), glVertex3f(x0, y0, z0), glVertex3f(x0, y1, z0), glVertex3f(x1, y1, z0)  
  
 glColor4f(0, 0, 0, 0.2)  
 glVertex3f(x0, y1, z1), glVertex3f(x0, y1, z0), glVertex3f(x0, y0, z0), glVertex3f(x0, y0, z1)  
 glVertex3f(x1, y1, z1), glVertex3f(x0, y1, z1), glVertex3f(x0, y0, z1), glVertex3f(x1, y0, z1)  
 glVertex3f(x1, y0, z1), glVertex3f(x0, y0, z1), glVertex3f(x0, y0, z0), glVertex3f(x1, y0, z0)  
  
 glEnd()  
 glDisable(GL\_BLEND)  
 # ------------------------  
  
 # Рисуем срез  
 # ------------------------  
 glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA)  
 glEnable(GL\_BLEND)  
 glColor4f(1, 0, 0, 0.2)  
 glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL)  
 slope\_25 = 25 \* self.spacing\_between\_slices / self.scale + z0  
 glColor4f(1, 0, 0, 0.2)  
 glBegin(GL\_QUADS)  
 glVertex3f(x0, y1, slope\_25), glVertex3f(x1, y1, slope\_25), glVertex3f(x1, y0, slope\_25), glVertex3f(x0, y0, slope\_25)  
 glEnd()  
 glDisable(GL\_BLEND)  
 # ------------------------  
  
 # Устанавливаем точку  
 # ------------------------  
 point = [self.x\_coord / self.scale + x0,  
 self.y\_coord / self.scale + y0,  
 slope\_25]  
 glPointSize(5)  
 glColor3f(0, 0, 0)  
 glBegin(GL\_POINTS)  
 glVertex3f(point[0], point[1], point[2])  
 glEnd()  
 # ------------------------  
  
  
def init\_window(width, height):  
 glutInitWindowSize(width, height)  
 glutInitWindowPosition((glutGet(GLUT\_SCREEN\_WIDTH) - width) // 2, (glutGet(GLUT\_SCREEN\_HEIGHT) - height) // 2)  
 glutCreateWindow('RGR')  
  
  
def main():  
 glutInit()  
 glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB)  
 file = Image()  
 init\_window(file.width, file.height)  
 file.init()  
 glutDisplayFunc(file.display)  
 glutIdleFunc(file.display)  
 glutMainLoop()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

# Розділ 5. Результати роботи програмного застосунку



**Висновки**

Результатом данної розрахунково-графічної роботи є розроблений програмний додаток для визначення просторового положення піксельних томографічних зображень та вивід тривимірних координат піксела зображення, що розташований у заданому рядку та стовпчику.

У результаті роботи програми на єкрані відображаетья координатні осі з початком координат в ізоцентрі сканера, границі об’єму зображення, який визначає отримана під час дослідження серія томографічних зображень; зріз серії зображень, що відповідає томографічному зображенні, яке визначається відповідно до отриманих даних, а також положення пікселя, який розташований у відповідному рядку та стовпчику томографічного зображення.

Програмний застосунок був розроблений за допомогою мови програмування Python та середовища розробки PyCharm, розробка графычного ынтерфейсу выдбувалася за допомогою быблыотеки OpenGL.