**Завдання**

**Завдання**:

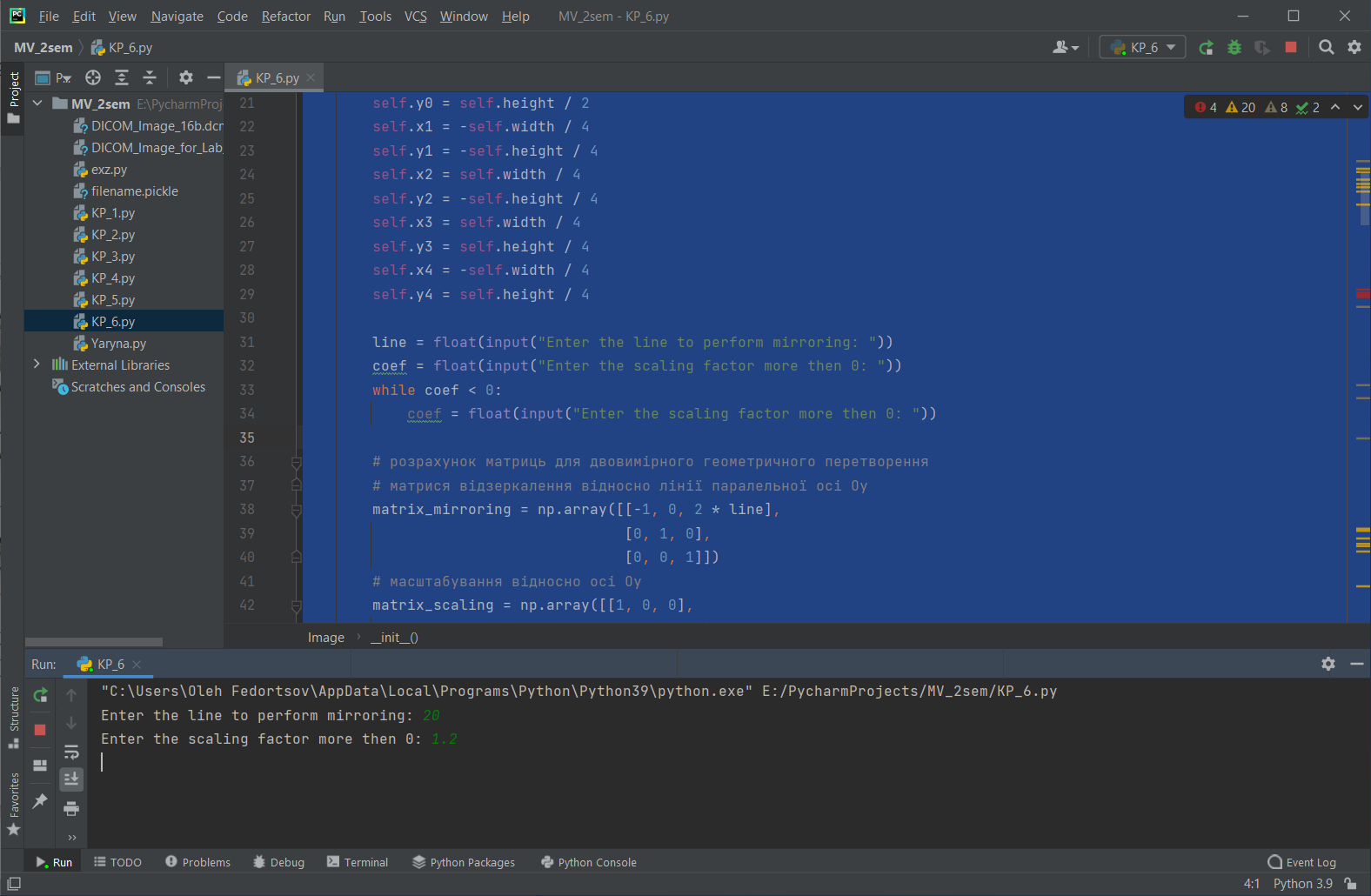
1. Ознайомитися з теоретичними відомостями реалізації алгоритмів сегментації медичних зображень, що засновані на пошуку границь.
2. Розробити програмний додаток для завантаження медичного зображення в форматі DICOM та виконання складних двовимірних геометричних перетворень.
3. Розміри частини вінка програмного додатка для візуалізації графічних даних (без інтерфейсу користувача) мають бути вдвічі більше відповідно розмірів завантаженого медичного зображення; відображання даних пікселів зображення має бути виконанане симетрично відносно центра (початок координат має бути розташований в центрі частини вікна програмного додатка, що відведена для візуалізації графічних даних); завантажене медичне зображення має бути розташоване в центрі вінка і мати масштаб 100%(одному пікселю зображення відповідає один піксель екрана).
4. Створити дві події від клавіатури, при обробці яких можна окремо виконати відображення результату виконання складного двовимірного геометричного перетворення і повернення до відображення оригінального зображення після виконання зворотного перетворення.
5. Виконати складне геометричне перетворення, визначивши матрицю складного перетворення через множення матриць заданих у варіанті завдання перетворень, при цьому параметри перетворення (вектор трансляції, кут повороту та зсуву, коефіцієнти масштабування та рівняння ліній для віддзеркалення) задавати інтерактивно.
6. Скласти і захистити звіт по роботі.

| **Номер варіанту** | **Перше геометричне перетворення** | **Друге геометричне перетворення** |
| --- | --- | --- |
| 8 | Віддзеркалення відносно лінії, що паралельна осі Оy | Масштабування вздовж осі Оу відносно початку координат із заданим коефіцієнтом масштабування |

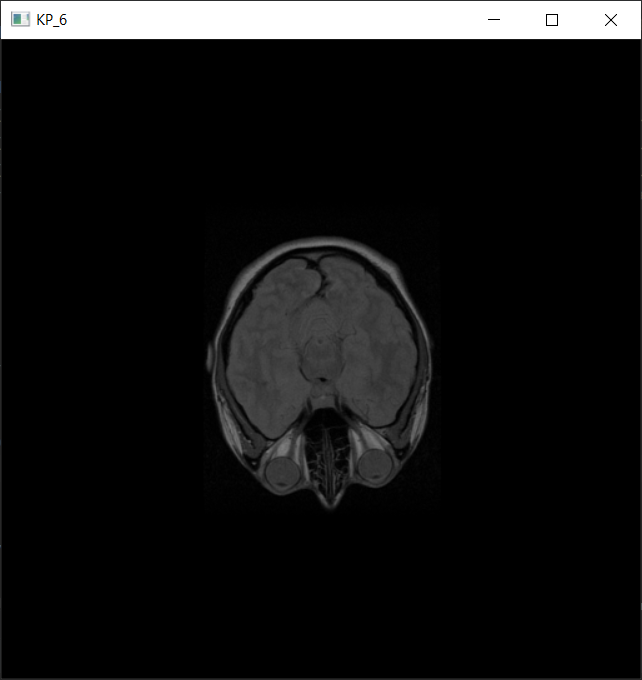
**Лістинг программи**

from OpenGL.GL import \*  
from OpenGL.GLU import \*  
from OpenGL.GLUT import \*  
import pydicom  
import numpy as np  
from OpenGL.arrays.numpymodule import ARRAY\_TO\_GL\_TYPE\_MAPPING  
  
path\_file = "DICOM\_Image\_16b.dcm"  
  
  
class Image:  
 def \_\_init\_\_(self, path):  
 self.ds = pydicom.read\_file(path)  
 self.image\_pixels = np.array(self.ds.pixel\_array)  
 self.width, self.height = self.ds[0x280010].value, self.ds[0x280011].value  
  
 self.rows = 0; self.cols = 0  
  
 self.isTransform = False  
 self.isReverse = False  
 self.isReturn = False  
  
 line = float(input("Enter the line to perform mirroring: "))  
 coef = float(input("Enter the scaling factor more then 0: "))  
 while coef < 0:  
 сoef = float(input("Enter the scaling factor more then 0: "))  
  
 # розрахунок матриць для двовимірного геометричного перетворення  
 # матрися відзеркалення відносно лінії паралельної до осі Оу  
 matrix\_mirroring = np.array([[-1, 0, 0, 0],  
 [0, 1, 0, 0],  
 [0, 0, 1, 0],  
 [2 \* line, 0, 0, 1]])  
  
 # масштабування відносно осі Оу  
 matrix\_scaling = np.array([[1, 0, 0, 0],  
 [0, coef, 0, 0],  
 [0, 0, 1, 0],  
 [0, 0, 0, 1]])  
  
 self.default\_matrix = None  
 # множення матриць для виконання геометричного перетворення  
 self.matrix\_transform = matrix\_mirroring.dot(matrix\_scaling)  
 # зворотня матриця для оберненного геометричного перетворення  
 self.reverse\_matrix\_transform = np.linalg.inv(self.matrix\_transform)  
  
 def init(self):  
 glClearColor(0, 0, 0, 0.0)  
 glMatrixMode(GL\_PROJECTION)  
 glLoadIdentity()  
 gluOrtho2D(-self.width / 2, self.width / 2, -self.height / 2, self.height / 2)  
 glMatrixMode(GL\_MODELVIEW)  
 self.default\_matrix = glGetFloatv(GL\_MODELVIEW\_MATRIX)  
  
 def display(self):  
 glClear(GL\_COLOR\_BUFFER\_BIT)  
 glColor3f(1, 1, 1)  
  
 draw\_pixel = np.copy(self.image\_pixels)  
 self.rows = len(draw\_pixel)  
 self.cols = len(draw\_pixel[0])  
  
 draw\_pixel = self.normalization\_func(draw\_pixel, 0, 1)  
  
 if self.isTransform:  
 self.draw\_texture(np.array(draw\_pixel, np.uint8), GL\_LUMINANCE, 'c')  
 elif self.isReverse:  
 self.draw\_texture(np.array(draw\_pixel, np.uint8), GL\_LUMINANCE, 'r')  
 elif self.isReturn:  
 self.draw\_texture(np.array(draw\_pixel, np.uint8), GL\_LUMINANCE, 'b')  
 else:  
 self.draw\_texture(np.array(draw\_pixel, np.uint8), GL\_LUMINANCE, '')  
 # self.print\_text(- self.width / 2, self.height / 2 - 40, GLUT\_BITMAP\_HELVETICA\_12, "'b' - back to start position ")  
 # self.print\_text(- self.width / 2, self.height / 2 - 30, GLUT\_BITMAP\_HELVETICA\_12, "'r' - reverse transformation")  
 # self.print\_text(- self.width / 2, self.height / 2 - 20, GLUT\_BITMAP\_HELVETICA\_12, "'c' - complex transformation")  
 # self.print\_text(- self.width / 2, self.height / 2 - 10, GLUT\_BITMAP\_HELVETICA\_12, "Press the key:")  
  
 glutSwapBuffers()  
  
 # def print\_text(self, x, y, font, line):  
 # glColor3f(0, 0, 1)  
 # glPushAttrib(GL\_DEPTH\_TEST)  
 # glRasterPos2d(x, y)  
 # for i in line:  
 # glutBitmapCharacter(font, ord(i))  
 # glPopAttrib()  
  
 def draw\_texture(self, data, internal\_format, type\_transform):  
 gl\_type = ARRAY\_TO\_GL\_TYPE\_MAPPING.get(data.dtype)  
  
 glTexImage2D(GL\_TEXTURE\_2D, 0, internal\_format, self.width, self.height, 0, internal\_format, gl\_type, data)  
 glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST)  
 glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST)  
 glEnable(GL\_TEXTURE\_2D)  
  
 glMatrixMode(GL\_MODELVIEW)  
 glLoadMatrixf(glGetFloatv(GL\_MODELVIEW\_MATRIX))  
 if type\_transform == 'c':  
 glMultMatrixf(self.matrix\_transform)  
 self.isTransform = False  
 elif type\_transform == 'r':  
 glMultMatrixf(self.reverse\_matrix\_transform)  
 self.isReverse = False  
 elif type\_transform == 'b':  
 glLoadMatrixf(self.default\_matrix)  
  
 self.default()  
  
 glDisable(GL\_TEXTURE\_2D)  
 glFlush()  
  
 def default(self):  
 glBegin(GL\_QUADS)  
 glTexCoord2d(0.0, 0.0)  
 glVertex2d(-self.width / 4, -self.height / 4)  
 glTexCoord2d(0.0, 1.0)  
 glVertex2d(-self.width / 4, self.height / 4)  
 glTexCoord2d(1.0, 1.0)  
 glVertex2d(self.width / 4, self.height / 4)  
 glTexCoord2d(1.0, 0.0)  
 glVertex2d(self.width / 4, -self.height / 4)  
 glEnd()  
  
 # функція виконує нормальзацію  
 def normalization\_func(self, pixels, p\_min, p\_max):  
 pixel\_max = int(float(p\_max \* pixels.max()))  
 pixel\_min = int(float(p\_min \* pixels.max()))  
  
 new\_min = 0  
 new\_max = np.iinfo(np.int8).max#255  
 normalization = []  
  
 for row in pixels:  
 new\_row = []  
 for pixel in row:  
 new\_pixel = ((pixel - pixel\_min) / (pixel\_max - pixel\_min)) \* (new\_max - new\_min)  
 if new\_pixel <= 0:  
 new\_pixel = 0  
 if new\_pixel > new\_max:  
 new\_pixel = new\_max  
 new\_row.append(new\_pixel)  
 normalization.append(new\_row)  
  
 return normalization  
  
 def keyboard\_func(self, my\_key, x, y):  
 key = unicode(my\_key, errors='ignore')  
 if key == 'c':  
 self.isTransform = True  
 self.isReverse = False  
 self.isReturn = False  
 if key == 'r':  
 self.isTransform = False  
 self.isReverse = True  
 self.isReturn = False  
 if key == 'b':  
 self.isTransform = False  
 self.isReverse = False  
 self.isReturn = True  
 self.display()  
  
def init\_window(width, height):  
 glutInitWindowSize(width, height)  
 glutInitWindowPosition((glutGet(GLUT\_SCREEN\_WIDTH) - width) // 2, (glutGet(GLUT\_SCREEN\_HEIGHT) - height) // 2)  
 glutCreateWindow('KP\_6')  
  
  
def main():  
 glutInit()  
 glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB)  
 file = Image(path\_file)  
 init\_window(file.width \* 2, file.height \* 2)  
 file.init()  
 glutDisplayFunc(file.display)  
 glutKeyboardFunc(file.keyboard\_func)  
 glutMainLoop()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

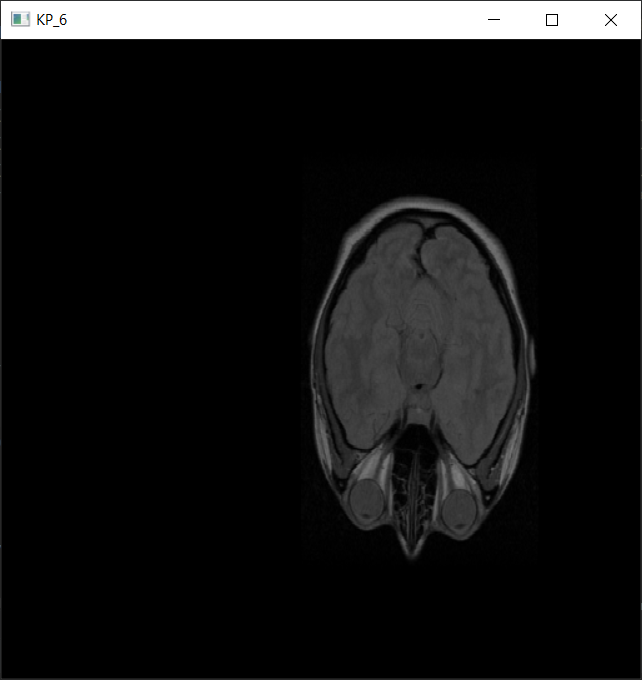
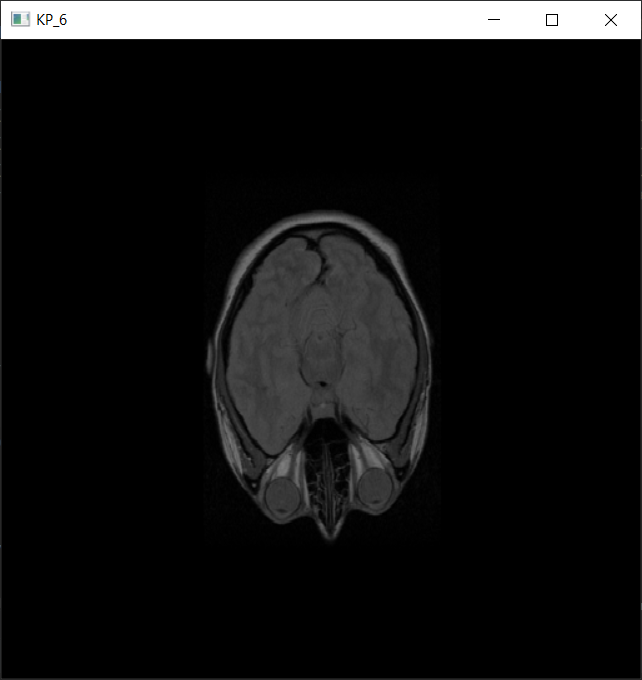
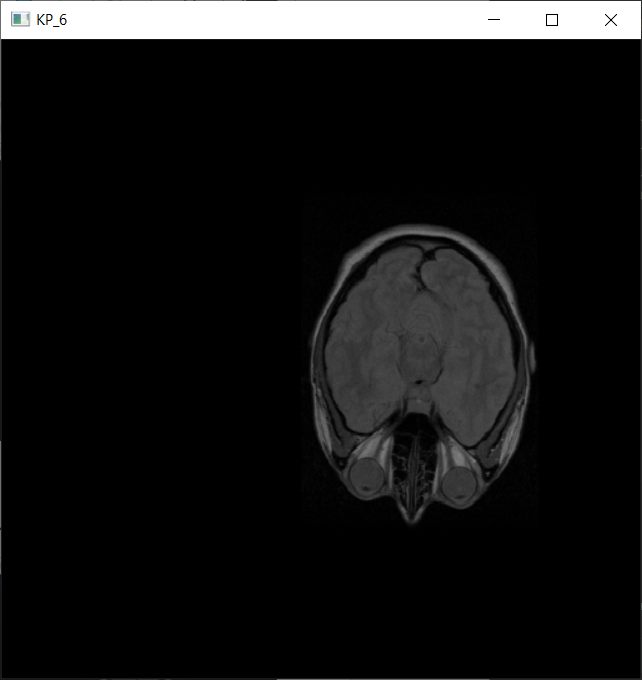
Результат



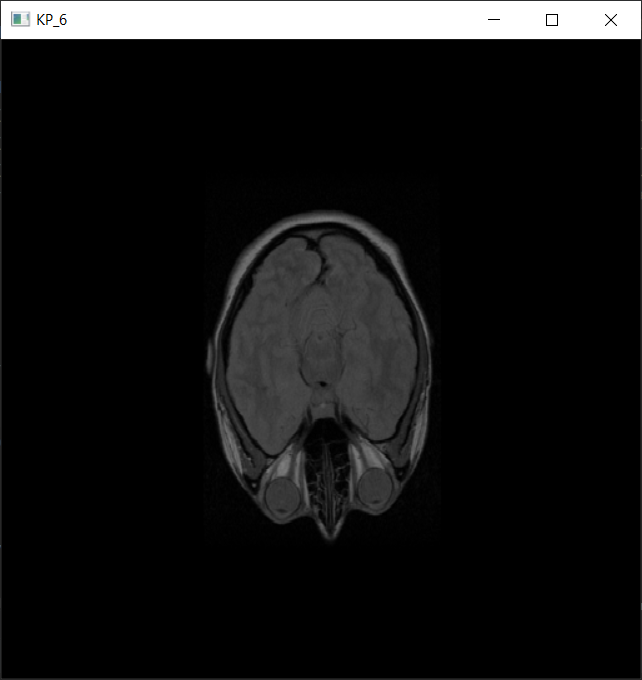
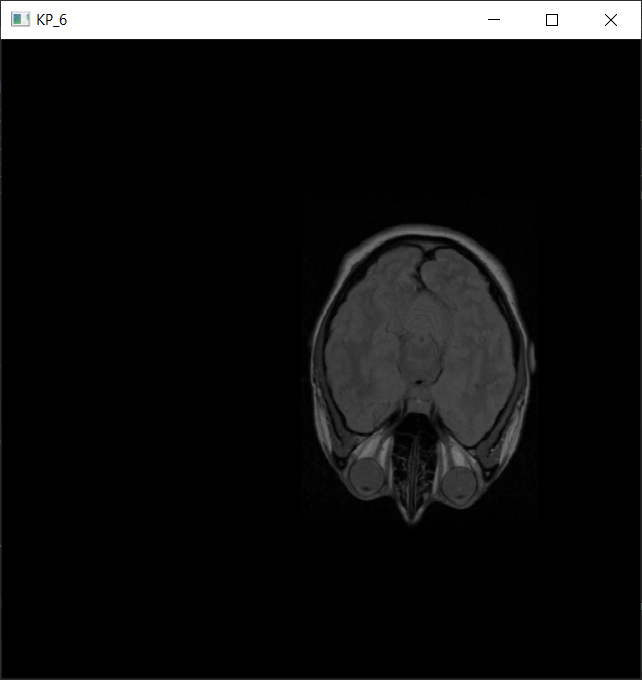
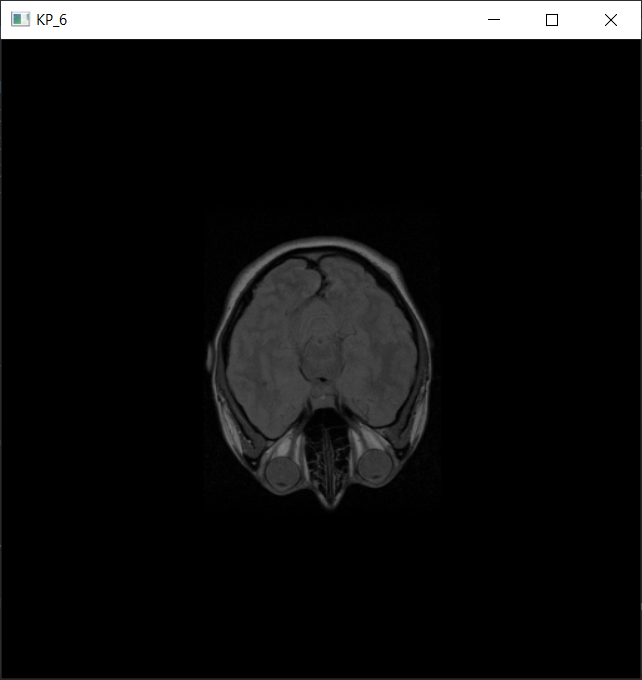
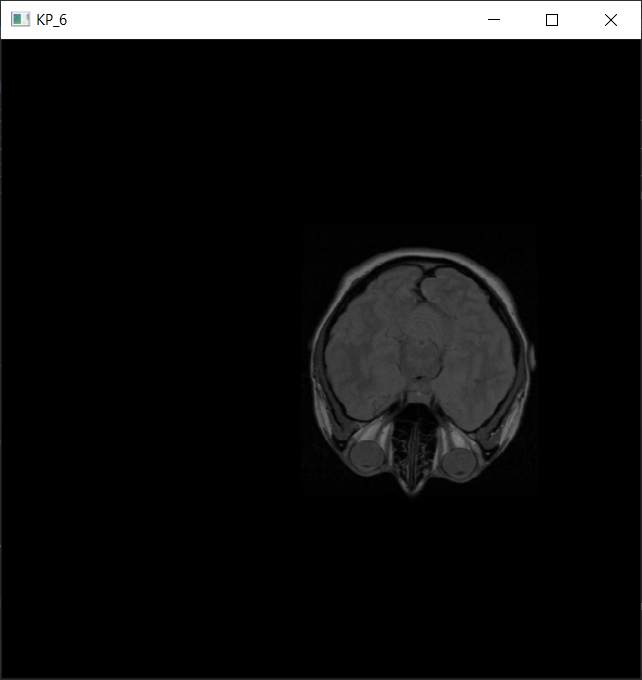
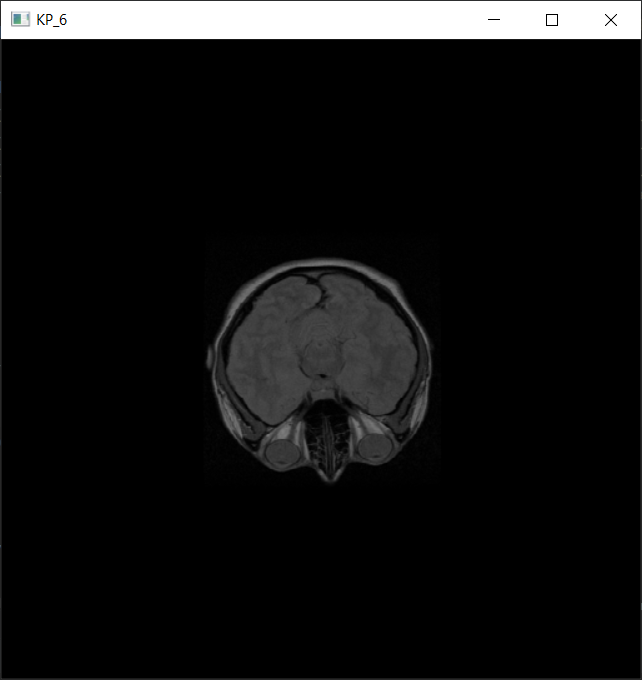
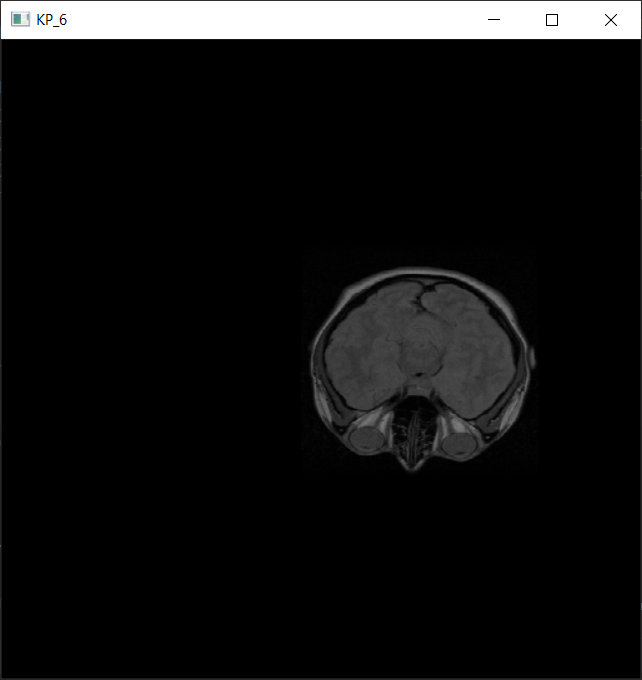
*Початкове зображення*



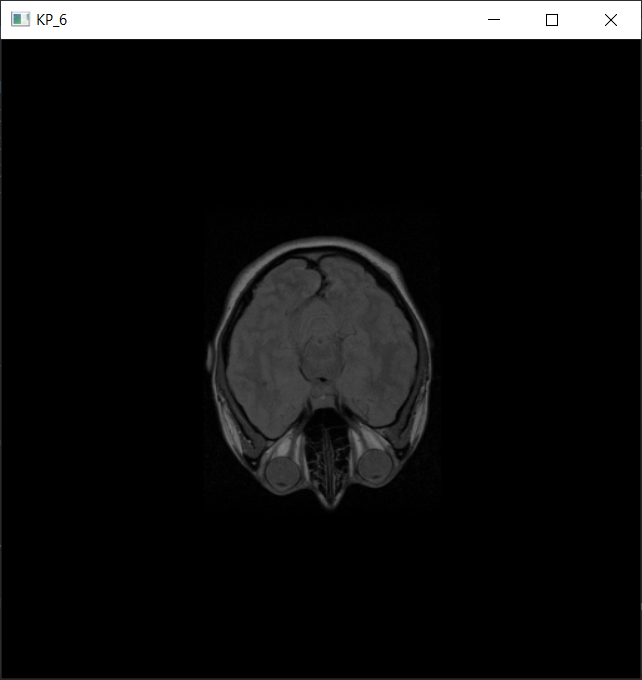
*Складне перетворення*



*Складне обернене перетворення*

*Повернення початкового вигляду*



**Контрольні запитання**

1. *Що таке однорідні координати? Навіщо їх використовувати?*

Однорідні координати це такі координати, які мають додатковий параметр *w*, що дозволяє формалізувати об’єкт.

Розширення декартових точок об’єкта до трьохелементного координатного зображення дозволяє формалізувати опис об’єкта в однорідних координатах, де – така ненульова величина при якій . Отже, довільне подання в двовимірних однорідних координатах можна представити як . Те саме для тривимірних координат.

При координати описують точку з кінцевими координатами , а при нескінченно віддалену в напрямку .

Формалізація подання точок в однорідних координатах дозволяє подати всі рівняння геометричних або лінійних перетворень через множення(суперпозиції) матриць, що дозволяє використовувати уніфікований механізм для роботи з матрицями при перетвореннях.

1. *Чому однорідні координати вимагають три компоненти для координатного подання точки на площині та чотири компоненти для точки в просторі?*

Тому що для формалізації точок в однорідні координати в нас додається новий елемент , так как на площини точка має тільки дві координати після формалізаціє їх стає три , на площині же точка має три координати після перетворення .

1. *Чому перетворення з евклідових координат в однорідні є неоднозначним, тоді як перехід з однорідних координат в евклідові є однозначним перетворенням?*

Так як перетворення з однорідних координат в евклідові фактично відповідає проекції точки на площину , таке перетворення є однозначним. А зворотне перетворення може дорівнювати будь-якому ненульовому значенню параметра , що спричиняю появу нескінченної кількості еквівалентних значень.

1. *Які геометричні перетворення є базисними?*

Базисними являються афінні перетворення.

1. *Чому в комп’ютерній графіці використовують матричні форми геометричних перетворень зображень?*

Тому що в комп’ютерній графіці зображення зазвичай представляє з себе матрицю значень.

1. *Що визначає кожен елемент загальної матриці жорсткого перетворення?*

Загальний вигляд двовимірної матриці жорстких перетворень в однорідних координатах:

Де чотири елементи – це мультиплікативні члени повороту, а елементи та – це трансляційні члени.

1. *Вивести матричну форму для проведення двовимірного повороту навколо осі обертання.*

У ряді випадків рухома система координат може здійснювати поворот на кут *φ* відносно довільної осі *r*, що представляє собою одиничний вектор з компонентами , і , що виходить з початку координат *O*. Перевага такого повороту полягає в тому, що для деяких кутових рухів послідовність поворотів щодо основних осей систем координат *OXYZ* можна замінити одним поворотом системи навколо осі *r*. Щоб отримати матрицю повороту , можна спочатку зробити ряд поворотів щодо осей системи *OXYZ*, щоб поєднати вісь *r* з віссю *OZ*. Після цього зробимо необхідний поворот навколо осі *r* на кут *φ* і знову ряд поворотів щодо осей системи *OXYZ*, які повертають вісь *OZ* в вихідне положення. Поєднання осей *OZ* і *r* може бути реалізовано за допомогою повороту на кут *α* відносно осі *ОХ* (вісь *r* внаслідок виявиться в площині *XZ*), а потім на кут - *β* навколо осі *ОY* (внаслідок осі *OZ* і *r* співпадають). Після повороту на кут *φ* відносно *OZ* або *r* зробимо вказану вище послідовність поворотів, але в зворотному порядку і змінимо при цьому знаки кутів повороту на протилежні.

Результуюча матриця рівна:

Визначаємо, що:

Підставляємо наші значення у попередній вираз для :

де

1. *Які складні геометричні перетворення є комутативними, а які ні?*

Комутативними є лише перенесення-перенесення і масштабування-масштабування.

1. *Яким чином знаходять зворотне перетворення після проведення нелінійного перетворення?*

Для будь-якого перетворення *f*, яке однозначно визначає точкове перетворення в просторі, існує зворотне перетворення , яке можна знайти використовуючи алгоритм чисельних методів (найпоширенішим є метод Ньютона).  
Для обрахунку початкових координат для точки за координатами цієї точки після проведення нелінійного перетворення за формулою:

1. *З якою метою використовують нелінійні перетворення під час обробки медичних зображень?*

Нелінійні перетворення – це бідь-які перетворення зображення, що включають всі можливі перетворення об’єкта на зображенні. Вони дозволяють нелінійні деформації між зображеннями, надає значно більше число ступенів свободи, ніж лінійні.