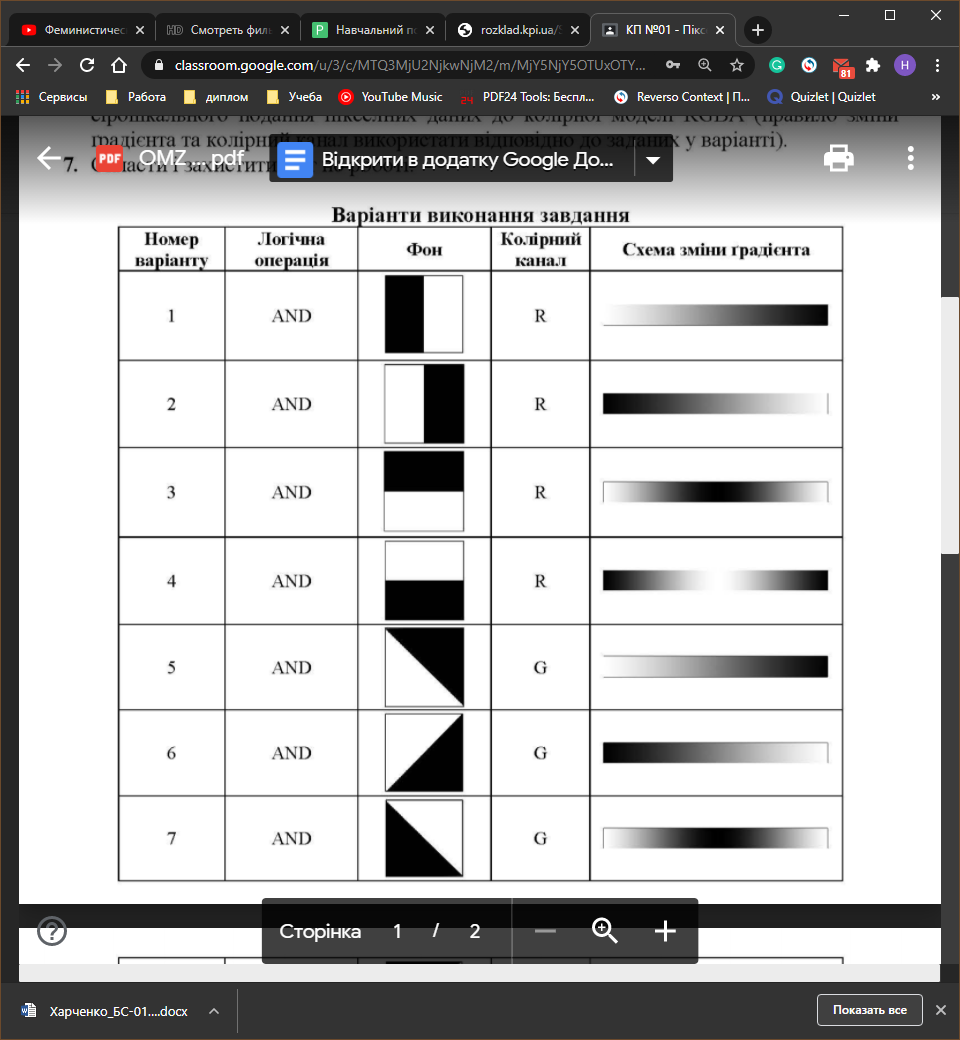
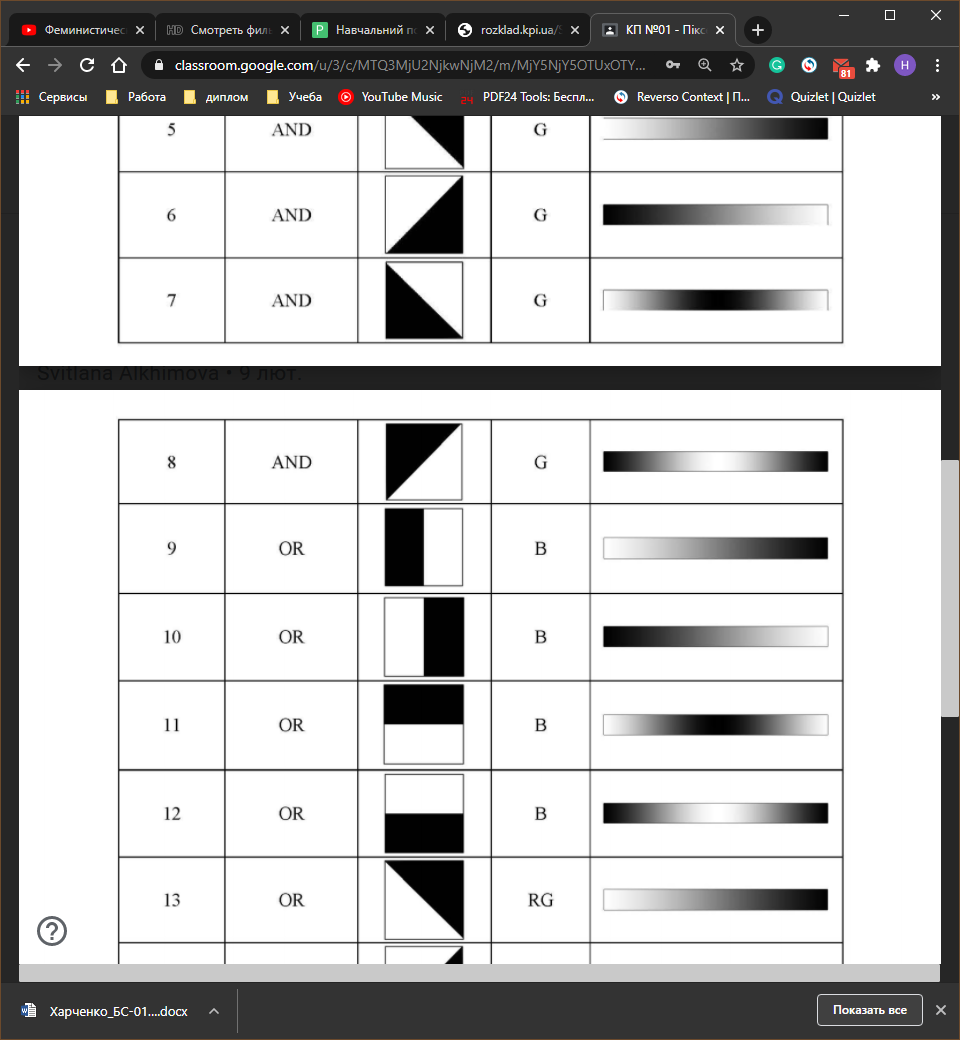
**Завдання**

1. Вивчити теоретичні основи відображення даних графічних форматів та методи піксельних перетворень зображень.
2. Розробити програмний додаток для завантаження медичного зображення в форматі DICOM та виконання піксельних перетворень.
3. Розміри частини вінка програмного додатка для візуалізації графічних даних (без інтерфейсу користувача) мають відповідати розмірам завантаженого медичного зображення; завантажене медичне зображення має мати масштаб 100% (одному пікселю зображення відповідає один піксель екрана).
4. Створити події від клавіатури, при обробці яких можна окремо виконати задані у варіанті пікселні операції над активним зображенням та відновити оригінальне відображення завантаженого томографічного зрізу.
5. Виконати логічне перетворення медичного зображення, застосувавши задану у варіанті логічну операцію до даних оригінального зображення та фонової текстури, яка створена як бітова мапа у програмі відповідно до схеми завдання у варіанті.
6. Виконати кольорове моделювання медичного зображення для переходу від сірошкального подання піксельних даних до колірної моделі RGBA (правило зміни градієнта та колірний канал використати відповідно до заданих у варінаті).
7. Скласти і захистити звіт по роботі.

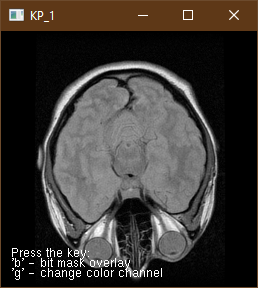
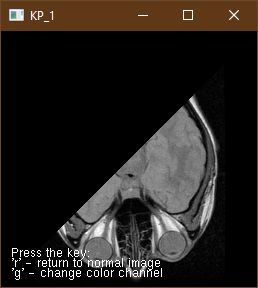


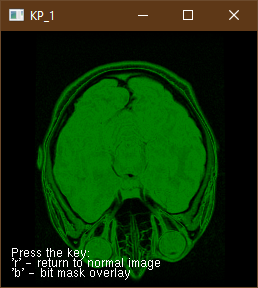
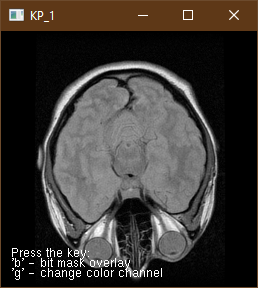


**Лістинг программи**

from OpenGL.GL import \*  
from OpenGL.GLU import \*  
from OpenGL.GLUT import \*  
import pydicom  
import numpy as np  
  
path\_file = "DICOM\_Image\_for\_Lab\_2.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  
  
 def init(self):  
 glClearColor(0, 0, 0, 0.0)  
 glMatrixMode(GL\_PROJECTION)  
 glLoadIdentity()  
 gluOrtho2D(0, self.width, 0, self.height)  
  
 def display(self):  
 glClear(GL\_COLOR\_BUFFER\_BIT)  
 glColor3f(1, 1, 1)  
  
 self.draw\_texture(self.image\_pixels, GL\_LUMINANCE)  
  
 self.print\_text(10, 10, GLUT\_BITMAP\_HELVETICA\_12, "'g' - change color channel")  
 self.print\_text(10, 20, GLUT\_BITMAP\_HELVETICA\_12, "'b' - bit mask overlay")  
 self.print\_text(10, 30, GLUT\_BITMAP\_HELVETICA\_12, "Press the key:")  
  
 glutSwapBuffers()  
  
 def print\_text(self, x, y, font, line):  
 glColor3f(1, 1, 1)  
 glPushAttrib(GL\_DEPTH\_TEST)  
 glRasterPos2d(x, y)  
 for i in line:  
 glutBitmapCharacter(font, ord(i))  
 glPopAttrib()  
  
 def draw\_texture(self, data, internal\_format):  
 bits = self.ds[0x280100].value  
 r\_intercept = self.ds[0x281053].value  
 r\_slope = self.ds[0x281052].value  
  
 if (r\_slope != 0) & (r\_intercept != 1):  
 data\_type = GL\_FLOAT  
 elif bits == 8:  
 data\_type = GL\_UNSIGNED\_BYTE  
  
 glTexImage2D(GL\_TEXTURE\_2D, 0, internal\_format, self.width, self.height, 0, internal\_format, data\_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)  
  
 glBegin(GL\_QUADS)  
 glTexCoord2d(0.0, 0.0)  
 glVertex2d(0.0, 0.0)  
 glTexCoord2d(1.0, 0.0)  
 glVertex2d(self.width, 0.0)  
 glTexCoord2d(1.0, 1.0)  
 glVertex2d(self.width, self.height)  
 glTexCoord2d(0.0, 1.0)  
 glVertex2d(0.0, self.height)  
 glEnd()  
  
 glDisable(GL\_TEXTURE\_2D)  
  
 def create\_bit\_mask(self):  
 bit\_mask = []  
  
 for i in range(self.width):  
 bit\_mask.append([])  
 for j in range(self.height):  
 if i >= j:  
 bit\_mask[i].append(0)  
 else:  
 bit\_mask[i].append(255)  
  
 return bit\_mask  
  
 def boolean\_transformations(self, data):  
 transform\_image = []  
 for i in range(self.width):  
 transform\_image.append([])  
 for j in range(self.height):  
 transform\_image[i].append(data[i][j] and self.image\_pixels[i][j])  
  
 return transform\_image  
  
 def create\_gradient(self):  
 gradient = {}  
 color = 0  
 for key in range(np.amin(self.image\_pixels), np.amax(self.image\_pixels) // 2):  
 gradient[key] = color  
 color += 1  
 if color > 255:  
 color = 255  
 for key in range(np.amax(self.image\_pixels) // 2, np.amax(self.image\_pixels) + 1):  
 gradient[key] = color  
 color -= 1  
 if color < 0:  
 color = 0  
  
 gradient\_image = []  
 for pixel\_line in self.image\_pixels:  
 line = []  
 for pixel in pixel\_line:  
 line.append(gradient[pixel])  
 gradient\_image.append(line)  
  
 return np.array(gradient\_image)  
  
 def channel\_G(self, data):  
 rgb = np.zeros((self.height, self.width, 3), 'uint8')  
  
 rgb[..., 1] = data  
 rgb[..., 0] = 0  
 rgb[..., 2] = 0  
  
 return rgb  
  
 def keyboard\_func(self, my\_key, x, y):  
 key = unicode(my\_key, errors='ignore')  
 if key == 'g':  
 gradient = self.create\_gradient()  
 self.draw\_texture(self.channel\_G(gradient), GL\_RGB)  
  
 self.print\_text(10, 10, GLUT\_BITMAP\_HELVETICA\_12, "'b' - bit mask overlay")  
 self.print\_text(10, 20, GLUT\_BITMAP\_HELVETICA\_12, "'r' - return to normal image")  
 self.print\_text(10, 30, GLUT\_BITMAP\_HELVETICA\_12, "Press the key:")  
  
 glutSwapBuffers()  
 if key == 'b':  
 bit\_mask = self.create\_bit\_mask()  
 self.draw\_texture(self.boolean\_transformations(bit\_mask), GL\_LUMINANCE)  
  
 self.print\_text(10, 10, GLUT\_BITMAP\_HELVETICA\_12, "'g' - change color channel")  
 self.print\_text(10, 20, GLUT\_BITMAP\_HELVETICA\_12, "'r' - return to normal image")  
 self.print\_text(10, 30, GLUT\_BITMAP\_HELVETICA\_12, "Press the key:")  
  
 glutSwapBuffers()  
 if key == 'r':  
 self.draw\_texture(self.image\_pixels, GL\_LUMINANCE)  
  
 self.print\_text(10, 10, GLUT\_BITMAP\_HELVETICA\_12, "'g' - change color channel")  
 self.print\_text(10, 20, GLUT\_BITMAP\_HELVETICA\_12, "'b' - bit mask overlay")  
 self.print\_text(10, 30, GLUT\_BITMAP\_HELVETICA\_12, "Press the key:")  
  
 glutSwapBuffers()  
  
  
def init\_window(width, height):  
 glutInitWindowSize(width, height)  
 glutInitWindowPosition((glutGet(GLUT\_SCREEN\_WIDTH) - width) // 2, (glutGet(GLUT\_SCREEN\_HEIGHT) - height) // 2)  
 glutCreateWindow('KP\_1')  
  
  
def main():  
 glutInit()  
 glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB)  
 file = Image(path\_file)  
 init\_window(file.width, file.height)  
 file.init()  
 glutDisplayFunc(file.display)  
 glutKeyboardFunc(file.keyboard\_func)  
 glutMainLoop()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

Результат

**Контрольні запитання**

1. Що таке альфа-канал, принцип його використання.

Альфа-канал – це характеристика пікселів зображення, яка відповідає за повну або часткову прозорість пікселів. 8-бітний альфа-канал може мати 256 ступенів прозорості від 0 (повністю прозорий) до 255 (повністю не прозорий). При накладанні декількох пікселів один на одного їх результуюча яскравість може розраховуватися за певними формулами: α-змішування від Сміта та Кетмала, α-змішування від Волеса та Левоя, α-змішування від Портера та Дафа.

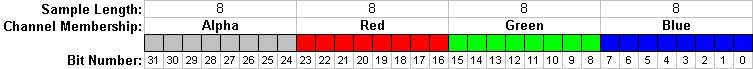
1. Як реалізувати запис та зчитування даних значення кожного з колірних каналів, якщо зображення зберігаються в 32-х бітному цілому?

У такому зображені на кожен піксель вивожиться чотири байти значення, що містять синій, зелений, червоний кольори і один байт для альфа-компоненти (прозорість)), отже необхідно считати ці значення та вивести їх на екран (передаючи значення на відео-буфер).

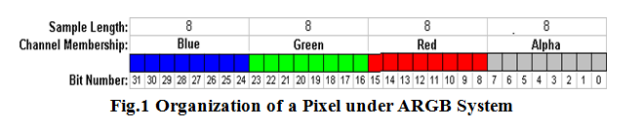
1. Як архітектура little-endian та big-endian впливає на розрашування в пам’яті байтів каналів ***ARGB-***зображення?

Для ARGB-зображення, А є найяважливішим значенням байтів 32-бітного значення, R друге по значенню, далі G та B має найменш значимі байти.

Отже для little-endian розположення байтів виглядає таким чином:



Отже для big-endian розположення байтів виглядає таким чином:



1. Як зберігаються кольорові зображення відповідно до стандарту ***DICOM***?

Для збереження інформації про колірний простір зображення можна передати за допомогою RGB-моделі або HSV-моделі. У перехрестну таблицю просторових посилань (LUT LookUp Table) записуються номера індексів для виведення значенб. LUT виглядає як мапа кольорів. Три значення Palette Color Lookup Table Descriptor (0028,1101-1104) зберігають інформацію про LUT

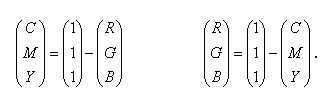
Red Palette Color Lookup Table Descriptor Attribute (0028, 1101)

Green Palette Color Lookup Table Descriptor Attribute (0028, 1102)

Blue Palette Color Lookup Table Descriptor Attribute (0028, 1103)

1. Яка формула переходу від колірної моделі ***RGB*** до ***CMYK***?

Спочатку необідно перетворити RGB в CMY:



Після чого вже CMY в CMYK:

1. Навіщо в колірній моделі ***CMYK*** використовіється додатковий четвертий параметр?

**CMYK** (Cyan, Magenta, Yellow, Key или Black).  
У палітрі **CMY** при змішуванні усіх кольорів в однаковій кількості виходить не чорний колір, а грязно-коричневий, що пов’язано з тим, що у реальних красках існуються домішки. Голубий колір має збиток синього, а пурпурна та жовта – збиток червоного. Для виправлення цього було вирішено додати чорний колір, який являється ключовим кольором для отримання більш чітких, глубоких чорних тонів та віддітнків.

1. Визначити призначення кожної з арифметичних операцій (додавання, віднімання, множення, ділення) при обробці медичних зображень.

Додавання використовується частіше за все на проміжних кроках обробки зображення для більш складних алгоритмів. Також вони використовуються для збільшення або зменшення яскравості зображення.

Віднімання використовуються для того щоб відняти певну маску від зображення, наприклад при введенні контрастної речовини, таким чином ми зможемо побачити саму рух цієї речовини і більш нічого іншого.

Ділення використовуєтьяс для виділення окремих частин зображення та переміщення об’єктів.

Множення використовується для виділення певних об’єктів (плям) шляхом накладання певної маски. Чіткість зображення залежить від розміру маски.

1. Визначити призначення кожної з логічних операцій (***AND***, ***OR***, ***NOT***, ***XOR***) при обробці медичних зображень.

Операції ***AND*** та ***OR*** можна використовувати для виділення певних частин зображення за допомогою двійковими масками (бітовими мапами), корисно для обробки ізольованих ділянок.

Операція ***NOT*** дозволяє інвертувати зображення.

Опаріція ***OR*** може використовувати для виділення на зображенні пікселів, які існують на одному з вихідних зображень, а ***XOR*** (виключаюче або) виділяються пікселі колір яких не співпадає з вихідним зображенням.

1. Що таке бітова мапа, навіщо вона потрібна?

Bitmap (бітова карта) – матриця даних з 1 і 0, що зберігає інформація про значення кожного пікселя зображення.

1. Як зберігати бітові мапи з оптимальним використанням ресурсів пам’яті?

Для оптимізації використання ресурсів пам’яті можна використовувати різноманітні способи стиснення інформації. Одним із поширених є алгоритм реєстрового ущільнення, який має коефіцієнт стиснення до 40:1.