МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ
УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ
КАФЕДРА БІОМЕДИЧНОЇ КІБЕРНЕТИКИ

Комп’ютерний практикум №4

з дисципліни «Обробка медичних зображень»
на тему: «Піксельні дані в медичних зображеннях»

Варіант №16

Виконав:
студент гр. БС-91мп
Шуляк Я.І.
Перевірив:
доцент каф. БМК
к.т.н. Алхімова С.М.

Зараховано від ___.___._____

_____
(підпис викладача)

Київ-2019

Завдання

1. Вивчити теоретичні відомості щодо специфіки роботи з пікселними даними медичних зображень.
2. Розробити програмний застосунок для завантаження медичного зображення (томографічного зрізу) в форматі DICOM виконання операцій з пікселними даними цього зображення.
3. Розміри частини вікна програмного застосунку для візуалізації графічних даних (без інтерфейсу користувача) мають відповідати розмірам завантаженого медичного зображення; завантажене медичне зображення має мати масштаб 100% (одному пікселу зображення відповідає один піксел екрана).
4. Створити події, при обробці яких можна окремо виконати задану в варіанті завдання операцію, що призведе до зміни представленого в пам’яті програми пікселних даних зображення та його відображення на екрані, та відновити оригінальне відображення завантаженого томографічного зрізу.
5. Визначити та вивести на екран інформацію стосовно пікселних даних мінімальне та максимальне значення, що зберігаються в піксельних даних відображуваного зображення, значення коефіцієнта масштабування, значення коефіцієнта зсуву, типу піксельних даних для відображуваного зображення.
6. Виконати збереження створеного відповідно до заданого в варіанті завдання похідного зображення до файлу в форматі DICOM. Надати можливість завантаження збереженого похідного зображення в розроблену програму із відповідним оновленням інформації стосовно піксельних даних, що має виводитися на екран.
7. Скласти і захистити звіт по роботі.

Номер варіанта	Коефіцієнт масштабування	Коефіцієнт зсуву	Верхня межа діапазону піксельних даних	Нижня межа діапазону піксельних даних
16	0.0101769	3.142	250	85

Лістинг програми:

main.cpp

```
#include <GL/glew.h>
#include <GL/freeglut.h>
```

```cpp
#include <dcmtk/dcmimgle/dcmimage.h>
#include <iostream>
#include "Image.h"
#include "TextRenderer.h"
#include "DicomFileWrapper.h"

int windowWidth = 520;
int windowHeight = 400;
int imageWidth;
int imageHeight;

DicomFileWrapper* dicomFileWrapper;

std::vector<Renderable*> renderableObjects;

const float givenRescaleSlope = 0.0101769;
const float givenRescaleIntercept = 3.142;
const int upperPixelBound = 250;
const int lowerPixelBound = 85;

struct DisplayMode {
    std::string helpMessage;
    std::string imageType;
    std::string pixelType;
    std::string pixelRange;
    std::string rescaleSlope;
    std::string rescaleIntercept;
    Image* image;
};

std::vector<DisplayMode> modes;
int currentDisplayMode = 0;
std::pair<float, float> rescaleMinMax;
const float* scaledPixels = nullptr;

void initDrawableObjects(const std::string &imagePath);
const float* applyRescale(float rescaleSlope, float rescaleIntercept, const unsigned char* pixels);
void updateTextLabels(const DisplayMode& displayMode);
void render();
void keyboardInput(unsigned char key, int x, int y);

int main(int argc, char *argv[]) {

    std::string imagePath;
    std::cout << "Enter path to image: ";
    std::cin >> imagePath;

    glutInit(&argc, argv);
    glutInitContextVersion(4, 0);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(windowWidth, windowHeight);
    glutCreateWindow("Lab4");
    glewInit();

    initDrawableObjects(imagePath);

    glutDisplayFunc(render);
    glutKeyboardFunc(keyboardInput);

    glutMainLoop();

    return 0;
}

void initDrawableObjects(const std::string &imagePath) {

    dicomFileWrapper = new DicomFileWrapper(imagePath);

    imageWidth = dicomFileWrapper->getUShort(DcmTagKey(0x0028, 0x0011));
    imageHeight = dicomFileWrapper->getUShort(DcmTagKey(0x0028, 0x0010));

    DcmCodeString imageTypeCodeString = dicomFileWrapper->getCodeString(DcmTagKey(0x0008, 0x0008));
    OFString pixelDataCharacteristic;
    imageTypeCodeString.getOFString(pixelDataCharacteristic, 0);
    auto imageType = std::string(pixelDataCharacteristic.c_str());

    auto rescaleIntercept = dicomFileWrapper->getDouble(DcmTagKey(0x0028, 0x1052));
    auto rescaleSlope = dicomFileWrapper->getDouble(DcmTagKey(0x0028, 0x1053));
```

```
    auto rescaleInterceptString = std::to_string(rescaleIntercept);
    auto rescaleSlopeString = std::to_string(rescaleSlope);

    auto defaultHelpMessage = "Controls: D - next, A - previous, S - save derived";
    DisplayMode originalDisplayMode;
    originalDisplayMode.helpMessage = defaultHelpMessage;
    originalDisplayMode.imageType = imageType;
    originalDisplayMode.rescaleSlope = rescaleSlopeString;
    originalDisplayMode.rescaleIntercept = rescaleInterceptString;

    if (imageType == "ORIGINAL" && rescaleIntercept == 0 && rescaleSlope == 1) {

        auto bitsAllocated = dicomFileWrapper->getUShort(DcmTagKey(0x0028, 0x0100));
        if (bitsAllocated != 8)
            throw std::runtime_error("Image not supported");

        auto originalPixelData = dicomFileWrapper->getUCharArray(DcmTagKey(0x7FE0, 0x0010));
        auto minMaxPixels = DicomFileWrapper::findMinMaxPixel(imageWidth, imageHeight,
originalPixelData);
        originalDisplayMode.pixelRange =
std::to_string(minMaxPixels.first).append("-").append(std::to_string(minMaxPixels.second));
        originalDisplayMode.pixelType = "GL_UNSIGNED_BYTE";
        originalDisplayMode.image = new Image(windowWidth, windowHeight, imageWidth, imageHeight,
GL_UNSIGNED_BYTE, originalPixelData);
        modes.push_back(originalDisplayMode);

        auto updatedRangePixelData = new unsigned char[imageWidth * imageHeight];
        for (int i = 0; i < imageWidth * imageHeight; i++) {
            auto color = *(originalPixelData + i);
            if (color > upperPixelBound) color = upperPixelBound;
            if (color < lowerPixelBound) color = lowerPixelBound;
            *(updatedRangePixelData + i) = color;
        }
        auto updatedRangeMinMax = DicomFileWrapper::findMinMaxPixel(imageWidth, imageHeight,
updatedRangePixelData);

        DisplayMode updatedRangeDisplayMode;
        updatedRangeDisplayMode.helpMessage = defaultHelpMessage;
        updatedRangeDisplayMode.imageType = "Temp. Applied pixel range";
        updatedRangeDisplayMode.pixelRange =
std::to_string(updatedRangeMinMax.first).append("-").append(std::to_string(updatedRangeMinMax.second
));
        updatedRangeDisplayMode.rescaleSlope = originalDisplayMode.rescaleSlope;
        updatedRangeDisplayMode.pixelType = "GL_UNSIGNED_BYTE";
        updatedRangeDisplayMode.rescaleIntercept = originalDisplayMode.rescaleIntercept;
        updatedRangeDisplayMode.image = new Image(windowWidth, windowHeight, imageWidth,
imageHeight, GL_UNSIGNED_BYTE, updatedRangePixelData);
        modes.push_back(updatedRangeDisplayMode);

        scaledPixels = applyRescale(givenRescaleSlope, givenRescaleIntercept,
updatedRangePixelData);

        DisplayMode appliedRescaleDisplayMode;
        appliedRescaleDisplayMode.helpMessage = defaultHelpMessage;
        appliedRescaleDisplayMode.imageType = "Temp. Applied rescale";
        appliedRescaleDisplayMode.pixelRange =
std::to_string(rescaleMinMax.first).append("-").append(std::to_string(rescaleMinMax.second));
        appliedRescaleDisplayMode.rescaleSlope = std::to_string(givenRescaleSlope);
        appliedRescaleDisplayMode.rescaleIntercept = std::to_string(givenRescaleIntercept);
        appliedRescaleDisplayMode.pixelType = "GL_FLOAT";
        appliedRescaleDisplayMode.image = new Image(windowWidth, windowHeight, imageWidth,
imageHeight, GL_FLOAT, scaledPixels);
        modes.push_back(appliedRescaleDisplayMode);

    } else {
        originalDisplayMode.pixelType = "GL_FLOAT";
        auto pixels = dicomFileWrapper->getFloatArray(DcmTagKey(0x7FE0,0x0008));
        float min = 10000.0f, max = 0.0f;
        for (int i = 0; i < imageWidth * imageHeight; i++) {
            float color = *(pixels + i);
            if (color < min) min = color;
            if (color > max) max = color;
        }
        originalDisplayMode.pixelRange =
std::to_string(min).append("-").append(std::to_string(max));
        originalDisplayMode.image = new Image(windowWidth, windowHeight, imageWidth, imageHeight,
GL_FLOAT, pixels);
        originalDisplayMode.helpMessage = "Displaying derived image";
```

```
        modes.push_back(originalDisplayMode);
    }

    updateTextLabels(modes.at(currentDisplayMode));

}

const float* applyRescale(float rescaleSlope, float rescaleIntercept, const unsigned char* pixels) {
    auto tempFloatArray = new float[imageWidth * imageHeight];
    float min = 10000.0f, max = 0.0f;
    for (int i = 0; i < imageWidth * imageHeight; i++) {
        float color = *(pixels + i);
        color = rescaleSlope * color + rescaleIntercept;
        *(tempFloatArray + i) = color;
        if (color < min) min = color;
        if (color > max) max = color;
    }
    rescaleMinMax = std::pair(min, max);
    for (int i = 0; i < imageWidth * imageHeight; i++) {
        auto color = *(tempFloatArray + i);
        color = (color - min) / (max - min);
        *(tempFloatArray + i) = color;
    }
    return tempFloatArray;
}

void updateTextLabels(const DisplayMode& displayMode) {
    renderableObjects.clear();
    auto builder = TextRendererBuilder(windowWidth, windowHeight);
    renderableObjects.push_back(builder.setPosition(0, windowHeight -
16).setText(displayMode.helpMessage).build());
    renderableObjects.push_back(builder.setPosition(0, windowHeight - 32).setText(std::string("Image
Type: ").append(displayMode.imageType)).build());
    renderableObjects.push_back(builder.setPosition(0, windowHeight - 48).setText(std::string("Pixel
Type:").append(displayMode.pixelType)).build());
    renderableObjects.push_back(builder.setPosition(0, windowHeight - 64).setText(std::string("Pixel
Range(min-max): ").append(displayMode.pixelRange)).build());
    renderableObjects.push_back(builder.setPosition(0, windowHeight -
96).setText(std::string("Rescale Intercept: ").append(displayMode.rescaleIntercept)).build());
    renderableObjects.push_back(builder.setPosition(0, windowHeight -
112).setText(std::string("Rescale Slope: ").append(displayMode.rescaleSlope)).build());
}


void render() {

    glClearColor(1, 1, 1, 1);
    glClear(GL_COLOR_BUFFER_BIT);

    auto displayMode = modes.at(currentDisplayMode);
    displayMode.image->render();

    for (auto renderable : renderableObjects)
        renderable->render();

    glutSwapBuffers();

}

void keyboardInput(unsigned char key, int x, int y) {
    if (modes.size() == 1)
        return;

    switch (key) {
        case 'd':
            currentDisplayMode = (currentDisplayMode + 1) % (int) modes.size();
            updateTextLabels(modes.at(currentDisplayMode));
            break;
        case 'a': {
            int tempDisplayModeIndex = currentDisplayMode - 1;
            currentDisplayMode = tempDisplayModeIndex < 0 ? (int) modes.size() - 1 :
tempDisplayModeIndex;
            updateTextLabels(modes.at(currentDisplayMode));
            break;
        }
        case 's':
            dicomFileWrapper->saveTransformedImage(imageWidth, imageHeight, scaledPixels,
givenRescaleSlope, givenRescaleIntercept);
```

```
            std::cout << "Transformed image was saved to file: derived.dcm" << std::endl;
            break;
    }

    glutPostRedisplay();
}
```

Renderable.h

```
class Renderable {
public:
    virtual void render() = 0;
};
```

Image.h

```
#include <GL/glew.h>
#include <string>
#include <vector>
#include "Shader.h"
#include "Renderable.h"

class Image : public Renderable {
private:
    Shader *shader;
    GLuint vao;
    GLuint vbo;
    GLuint ebo;
    GLuint texture;
public:
    Image(int windowWidth, int windowHeight, int imageWidth, int imageHeight, GLenum pixelType,
const void* pixelData);
    ~Image();
    void render() override;
};
```

Image.cpp

```
#include "Image.h"

Image::Image(int windowWidth, int windowHeight, int imageWidth, int imageHeight, GLenum pixelType,
const void *pixelData) {

    float vertexes[] = {
            //Vertices      //Texture Coords
            0, 0,           0, 1,
            1, 0,           1, 1,
            1, 1,           1, 0,
            0, 1,           0, 0
    };
    glGenVertexArrays(1, &vao);
    glBindVertexArray(vao);

    glGenBuffers(1, &vbo);
    glBindBuffer(GL_ARRAY_BUFFER, vbo);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertexes), &vertexes, GL_STATIC_DRAW);

    GLuint elements[] = {
            0, 1, 2, //First Triangle
            2, 3, 0  // Second Triangle
    };

    glGenBuffers(1, &ebo);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, ebo);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(elements), &elements, GL_STATIC_DRAW);

    auto vertexShader =
            "#version 400\n"
            "uniform mat4 projection;"
            "uniform mat4 model;"
            "uniform mat4 view;"
            "in vec2 position; "
            "in vec2 texture_coord_in; "
            "out vec2 texture_coord_out; "
            "void main()"
            "{"
```

```
            "    gl_Position = projection * view * model * vec4(position, 0.0, 1.0);"
            "    texture_coord_out = texture_coord_in;"
            "}";

    auto fragmentShader =
            "#version 400\n"
            "uniform sampler2D texture1;"
            "in vec2 texture_coord_out;"
            "out vec4 FragColor;"
            "void main()"
            "{"
            "    vec4 tmpColor = texture(texture1, texture_coord_out);"
            "    FragColor = vec4(tmpColor.r, tmpColor.r, tmpColor.r, 1);"
            "}";

    shader = new Shader(vertexShader, fragmentShader);
    glUseProgram(shader->getReference());

    glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 4 * sizeof(float), (void *) 0);
    glEnableVertexAttribArray(0);

    glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 4 * sizeof(float), (void *) (2 *
sizeof(float)));
    glEnableVertexAttribArray(1);

    glm::mat4 projection = glm::ortho<float>(0, windowWidth, 0, windowHeight, -1, 1);
    glm::mat4 view = glm::lookAt(
            glm::vec3(0, 0, 1.0f),
            glm::vec3(0, 0, 0.0f),
            glm::vec3(0.0f, 1.0f, 0.0f)
    );
    glm::mat4 model = glm::scale(glm::mat4(1.0f), glm::vec3(imageWidth, imageHeight, 1.0f));

    shader->setMatrix4("projection", projection);
    shader->setMatrix4("view", view);
    shader->setMatrix4("model", model);

    glActiveTexture(GL_TEXTURE0);
    glGenTextures(1, &texture);
    glBindTexture(GL_TEXTURE_2D, texture);

    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);

    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8, imageWidth, imageHeight, 0, GL_RED, pixelType,
pixelData);

    glUseProgram(shader->getReference());
    glUniform1i(glGetUniformLocation(shader->getReference(), "texture1"), 0);

}

Image::~Image() {
    delete shader;
}

void Image::render() {

    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture);
    glUseProgram(shader->getReference());
    glBindVertexArray(vao);
    glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, nullptr);

}
```

Shader.h

```
#include <GL/glew.h>
#include <string>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>

class Shader {
```

```
private:
    GLuint shaderProgram;
    GLuint compileShader(const char *shaderText, GLenum shaderType);

public:
    Shader(std::string vertexShaderScript, std::string fragmentShaderScript);
    void setMatrix4(const std::string &name, glm::mat4 matrix);
    GLuint getReference() { return shaderProgram; }

};
```

Shader.cpp

```
#include "Shader.h"
#include <iostream>

Shader::Shader(std::string vertexShaderScript, std::string fragmentShaderScript) {

    GLuint vertexShader = compileShader(vertexShaderScript.c_str(), GL_VERTEX_SHADER);
    GLuint fragmentShader = compileShader(fragmentShaderScript.c_str(), GL_FRAGMENT_SHADER);

    shaderProgram = glCreateProgram();
    glAttachShader(shaderProgram, vertexShader);
    glAttachShader(shaderProgram, fragmentShader);
    glLinkProgram(shaderProgram);
    glUseProgram(shaderProgram);

}

GLuint Shader::compileShader(const char *shaderText, GLenum shaderType) {

    //Load and Compile Shader

    GLuint shader = glCreateShader(shaderType);
    glShaderSource(shader, 1, &shaderText, nullptr);
    glCompileShader(shader);

    //Check compile status

    GLint status;
    glGetShaderiv(shader, GL_COMPILE_STATUS, &status);
    if (status != GL_TRUE) {
        char buffer[512];
        glGetShaderInfoLog(shader, 512, nullptr, buffer);
        throw std::runtime_error(buffer);
    }

    return shader;
}

void Shader::setMatrix4(const std::string &name, glm::mat4 matrix) {
    glUniformMatrix4fv(glGetUniformLocation(shaderProgram, name.c_str()), 1, GL_FALSE, &matrix[0]
[0]);
}
```

TextRenderer.h

```
#include <vector>
#include <string>
#include "Shader.h"
#include <ft2build.h>
#include FT_FREETYPE_H
#include "Renderable.h"

struct Character {
    GLuint     TextureID;  // ID handle of the glyph texture
    glm::ivec2 Size;       // Size of glyph
    glm::ivec2 Bearing;    // Offset from baseline to left/top of glyph
    GLuint     Advance;    // Offset to advance to next glyph
};

class TextRenderer : public Renderable {

private:

    std::vector<Character> characters;
```

```
    Shader* shader;
    GLuint vao;
    GLuint vbo;

    FT_Library ftLibrary;
    FT_Face ftFace;

    bool isVisible = true;

    std::string text;

    int textX = 0;
    int textY = 0;

public:
    TextRenderer(std::string font, int fontSize, int width, int height);
    ~TextRenderer();
    void setText(std::string text);
    void setTextPosition(int x, int y);
    void toggleVisibility();
    void render() override;
};


class TextRendererBuilder {
private:
    std::string fontName = "arial.ttf";
    std::string text = "";
    int windowX = 0, windowY = 0, posX = 0, posY = 0, fontSize = 14;
public:

    TextRendererBuilder(int x, int y) {
        windowX = x;
        windowY = y;
    }

    TextRendererBuilder& setFont(const std::string& fontName) {
        this->fontName = fontName;
        return *this;
    }

    TextRendererBuilder& setFontSize(int fontSize) {
        this->fontSize = fontSize;
        return *this;
    }

    TextRendererBuilder& setPosition(int x, int y) {
        this->posX = x;
        this->posY = y;
        return *this;
    }

    TextRendererBuilder& setText(std::string text) {
        this->text = text;
        return *this;
    }

    TextRenderer* build() {
        auto renderer = new TextRenderer(fontName, fontSize, windowX, windowY);
        renderer->setText(text);
        renderer->setTextPosition(posX, posY);
        return renderer;
    }

};
```

TextRenderer.cpp

```
#include "TextRenderer.h"

TextRenderer::TextRenderer(std::string font, int fontSize, int width, int height) {

    FT_Init_FreeType(&ftLibrary);
    FT_New_Face(ftLibrary, font.c_str(), 0,  &ftFace);

    FT_Set_Pixel_Sizes(ftFace, 0, fontSize);
```

```
auto vertexShader =
        "#version 400\n"
        "in vec4 vertex;"
        "out vec2 TexCoords;"
        "uniform mat4 projection;"
        "void main()"
        "{"
        "    gl_Position = projection * vec4(vertex.xy, 0.0, 1.0);"
        "    TexCoords = vertex.zw;"
        "}  ";

auto fragmentShader =
        "#version 400\n"
        "in vec2 TexCoords;"
        "out vec4 color;"
        "uniform sampler2D text;"
        "void main()"
        "{"
        "    vec4 sampled = vec4(1.0, 1.0, 1.0, texture(text, TexCoords).r);"
        "    color = vec4(0.0, 0.0, 0.0, 1.0) * sampled;"
        "} ";

shader = new Shader(vertexShader, fragmentShader);

glUseProgram(shader->getReference());

glm::mat4 projection = glm::ortho(0.0f, (float) width, 0.0f, (float) height);
glUniformMatrix4fv(glGetUniformLocation(shader->getReference(), "projection"), 1, GL_FALSE,
        glm::value_ptr(projection));

glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

for (GLubyte character = 0; character <= 127; character++) {

    FT_Load_Char(ftFace, character, FT_LOAD_RENDER);

    GLuint texture;
    glGenTextures(1, &texture);
    glBindTexture(GL_TEXTURE_2D, texture);
    glTexImage2D(
            GL_TEXTURE_2D,
            0,
            GL_RED,
            ftFace->glyph->bitmap.width,
            ftFace->glyph->bitmap.rows,
            0,
            GL_RED,
            GL_UNSIGNED_BYTE,
            ftFace->glyph->bitmap.buffer
    );

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

    Character characterObj = {
            texture,
            glm::ivec2(ftFace->glyph->bitmap.width, ftFace->glyph->bitmap.rows),
            glm::ivec2(ftFace->glyph->bitmap_left, ftFace->glyph->bitmap_top),
            (GLuint) ftFace->glyph->advance.x
    };

    characters.push_back(characterObj);

}

FT_Done_Face(ftFace);
FT_Done_FreeType(ftLibrary);

glGenVertexArrays(1, &vao);
glGenBuffers(1, &vbo);
glBindVertexArray(vao);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER, sizeof(GLfloat) * 6 * 4, NULL, GL_DYNAMIC_DRAW);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 4 * sizeof(GLfloat), 0);
glBindBuffer(GL_ARRAY_BUFFER, 0);
```

```
    glBindVertexArray(0);

}

TextRenderer::~TextRenderer() {
    delete shader;
}

void TextRenderer::setText(std::string text) {
    this->text = text;
}

void TextRenderer::setTextPosition(int x, int y) {
    textX = x;
    textY = y;
}

void TextRenderer::toggleVisibility() {
    isVisible = !isVisible;
}

void TextRenderer::render() {

    if (isVisible) {

        int x = textX, y = textY, scale = 1;

        glEnable(GL_BLEND);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

        glUseProgram(shader->getReference());

        glActiveTexture(GL_TEXTURE0);
        glBindVertexArray(vao);

        for(char character : text) {

            Character ch = characters[character];

            GLfloat xpos = x + ch.Bearing.x * scale;
            GLfloat ypos = y - (ch.Size.y - ch.Bearing.y) * scale;

            GLfloat w = ch.Size.x * scale;
            GLfloat h = ch.Size.y * scale;
            // Update vbo for each character
            GLfloat vertices[6][4] = {
                    { xpos,     ypos + h,   0.0, 0.0 },
                    { xpos,     ypos,       0.0, 1.0 },
                    { xpos + w, ypos,       1.0, 1.0 },

                    { xpos,     ypos + h,   0.0, 0.0 },
                    { xpos + w, ypos,       1.0, 1.0 },
                    { xpos + w, ypos + h,   1.0, 0.0 }
            };
            // Render glyph texture over quad
            glBindTexture(GL_TEXTURE_2D, ch.TextureID);
            // Update content of vbo memory
            glBindBuffer(GL_ARRAY_BUFFER, vbo);
            glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(vertices), vertices);
            glBindBuffer(GL_ARRAY_BUFFER, 0);
            // Render quad
            glDrawArrays(GL_TRIANGLES, 0, 6);
            // Now advance cursors for next glyph (note that advance is number of 1/64 pixels)
            x += (ch.Advance >> 6) * scale; // Bitshift by 6 to get value in pixels (2^6 = 64)

        }

    }

}
```

DicomFileWrapper.h

```
#include <iostream>
#include <dcmtk/dcmdata/dctk.h>
#include <dcmtk/dcmimgle/dcmimage.h>

class DicomFileWrapper {
```

```
private:
    DcmFileFormat *dcmFileFormat;
public:
    DicomFileWrapper(const std::string &imagePath);
    ~DicomFileWrapper();

    unsigned short getUShort(const DcmTagKey &dcmTagKey);
    DcmCodeString getCodeString(const DcmTagKey& dcmTagKey);
    double getDouble(const DcmTagKey& dcmTagKey);
    std::string getString(const DcmTagKey &dcmTagKey);
    const unsigned char* getUCharArray(const DcmTagKey &dcmTagKey);
    const float* getFloatArray(const DcmTagKey &dcmTagKey);
    void saveTransformedImage(int width, int height, const float* pixels, double rescaleSlope,
double rescaleIntercept);

    static std::pair<double, double> findMinMaxPixel(int width, int height, const unsigned char*
pixels);

};
```

DicomFileWrapper.cpp

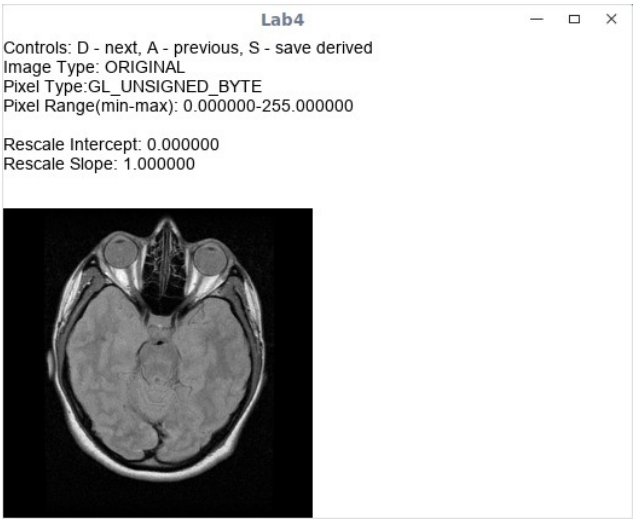
```
DicomFileWrapper::DicomFileWrapper(const std::string &imagePath) {
    dcmFileFormat = new DcmFileFormat();
    dcmFileFormat->loadFile(imagePath.c_str());
}

DicomFileWrapper::~DicomFileWrapper() {
    delete dcmFileFormat;
}

unsigned short DicomFileWrapper::getUShort(const DcmTagKey &dcmTagKey) {
    unsigned short value = 0;
    dcmFileFormat->getDataset()->findAndGetUint16(dcmTagKey, value);
    return value;
}

const unsigned char *DicomFileWrapper::getUCharArray(const DcmTagKey &dcmTagKey) {
    const unsigned char *value;
    dcmFileFormat->getDataset()->findAndGetUint8Array(dcmTagKey, value);
    return value;
}

const float *DicomFileWrapper::getFloatArray(const DcmTagKey &dcmTagKey) {
    const float *value;
    dcmFileFormat->getDataset()->findAndGetFloat32Array(dcmTagKey, value);
    return value;
}

DcmCodeString DicomFileWrapper::getCodeString(const DcmTagKey &dcmTagKey) {
    OFString value;
    dcmFileFormat->getDataset()->findAndGetOFStringArray(dcmTagKey, value);
    DcmCodeString dcmCodeString(DcmTag(dcmTagKey), value.size());
    dcmCodeString.putString(value.c_str());
    return dcmCodeString;
}

double DicomFileWrapper::getDouble(const DcmTagKey &dcmTagKey) {
    Float64 value = 0;
    dcmFileFormat->getDataset()->findAndGetFloat64(dcmTagKey, value);
    return value;
}

void DicomFileWrapper::saveTransformedImage(int width, int height, const float* pixels, double
rescaleSlope, double rescaleIntercept) {
    DcmFileFormat newImage(*dcmFileFormat);
    newImage.getDataset()->putAndInsertString(DcmTag(DcmTagKey(0x0008, 0x0008)), "DERIVED\\
SECONDARY\\MPR");
    newImage.getDataset()->putAndInsertString(DcmTag(DcmTagKey(0x0028, 0x1052)),
std::to_string(rescaleIntercept).c_str());
    newImage.getDataset()->putAndInsertString(DcmTag(DcmTagKey(0x0028, 0x1053)),
std::to_string(rescaleSlope).c_str());
    newImage.getDataset()->putAndInsertFloat32Array(DcmTagKey(0x7FE0,0x0008), pixels, width *
height);
    newImage.getDataset()->putAndInsertUint16(DcmTag(DcmTagKey(0x0028, 0x0100)), 32);

    char uid[64];
    newImage.getDataset()->putAndInsertString(DcmTagKey(0x0020,0x000E),
```

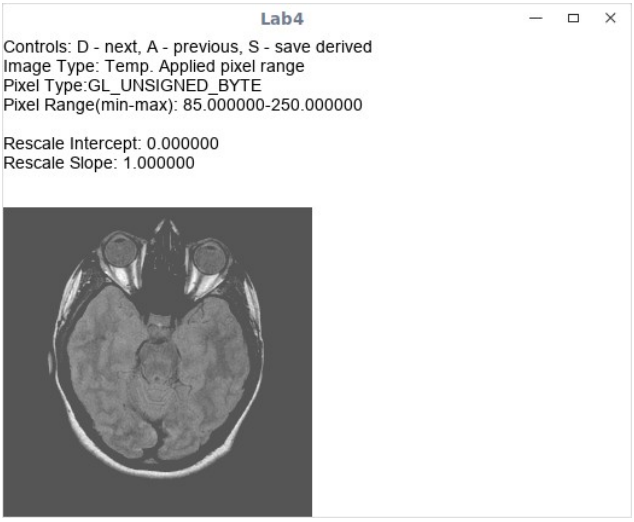
```
dcmGenerateUniqueIdentifier(uid)); //Series instance UID
    newImage.getDataset()->putAndInsertUint16(DcmTagKey(0x0200,0x0011), 1); //Series Number
    newImage.getDataset()->putAndInsertUint16(DcmTagKey(0x0200,0x0013), 1); //Instance Number
    newImage.getDataset()->putAndInsertString(DcmTagKey(0x0020,0x0016),
dcmGenerateUniqueIdentifier(uid)); //SOP instance UID
    newImage.getDataset()->putAndInsertString(DcmTagKey(0x0002,0x0003), uid); //Media Storage SOP
instance UID

    newImage.saveFile("derived.dcm");
}

std::pair<double, double> DicomFileWrapper::findMinMaxPixel(int width, int height, const unsigned
char* pixels) {
    double min = 255;
    double max = 0;
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            unsigned char pixel = *(pixels + width * y + x);
            if (pixel < min) min = pixel;
            if (pixel > max) max = pixel;
        }
    }
    return std::pair(min, max);
}
```

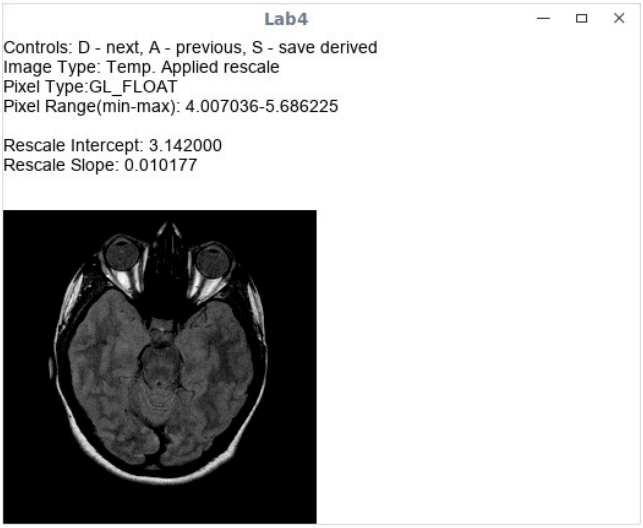
Результат роботи:

Консоль:

Enter path to image: original.dcm

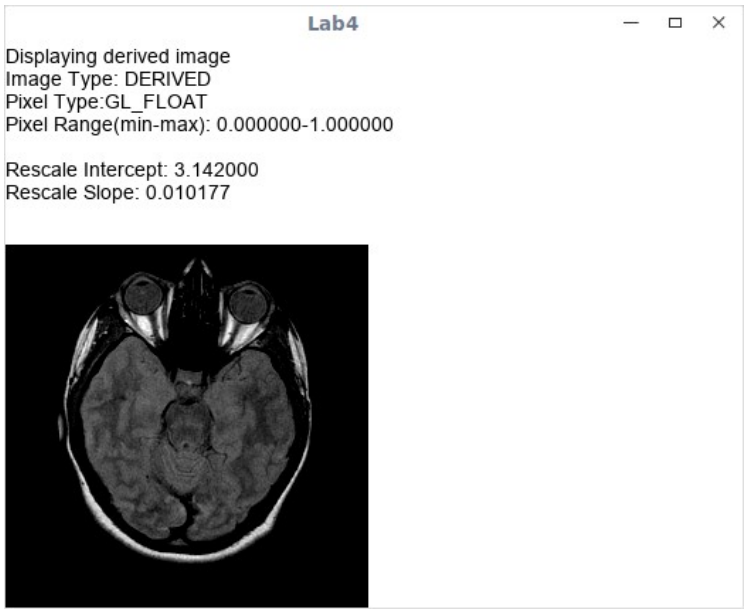






При натисненні на клавішу s виведеться повідомлення:
Transformed image was saved to file: derived.dcm

Далі перезапускаємо програму і вказуємо при запуску новий файл
Enter path to image: derived.dcm



Контрольні запитання

1. Який порядок кодування даних пікселів в площині зображення?

Першим по порядку є верхній лівий піксель, далі йдуть наступні пікселі по рядку, зверху вниз.

2. Пояснити поняття Pixel Cell?

Pixel Cell – це контейнер для зберігання значення піксела (Pixel Sample Value) та можливих додаткових бітів. Ці додаткові біти можуть, наприклад, використовуватися для зберігання інформації щодо оверлейних даних або для вирівнювання пікселів в певних межах (наприклад, byte, word).

3. Що таке оверлейні дані, як відповідно до стандарту DICOM зберігається інформації щодо оверлейних даних зображення?

Оверлейні дані представляють собою додаткові зображення, що можна малювати над основним, і є однобітними чорно-білими зображеннями. Оверлейні дані можуть представлятися як багатовимірними масивами, так і бути порожніми в разі їх відсутності.

4. Перелічити елементи DICOM, які визначають структуру пікселів зображення; їх призначення.

Тег (0028, 0100) - Bits Allocated – визначає об’єм пам’яті, що займає кожний окремий елемент Pixel Cell; це значення завжди більше або дорівнювати значенню, що зберігається в елементі Bits Stored.

Тег (0028, 0101) - Bits Stored – зберігає значення загальної кількості бітів, що були дійсно задіяні для зберігання даних кожного окремого піксела.
Тег (0028, 0102) - High Bits – визначає порядок розміщення даних окремого піксела всередині елементу Pixel Cell, який вказує, де слід розміщувати старший біт під час зберігання даних.

5. Призначення елементів Rescale Intercept і Resalale Slope.

Елементи DICOM Rescale Intercept (0028, 1052) і Rescale Slope (0028, 1053) визначають лінійне перетворення для значень пікселних даних із виду, в якому вони зберігаються в файлі DICOM, до виду, в якому вони мають бути представлені у пам'яті програми для відображення.

6. Навести та пояснити формулу, відповідно якої відбувається представлення пікселних даних в пам'яті програми для відображення.

$$U = m * SV + b \quad ,$$

Де U – значення пікселних даних для відображення;
m – значення коефіцієнта масштабування, що зберігається в елементі Rescale Slope;
SV – значення пікселних даних зображення, що зберігається в файлі DICOM;
b – значення коефіцієнта зсуву, що зберігається в елементі Rescale Intercept.

7. У яких випадках застосовується лінійне масштабування?

Лінійне масштабування застосовується у випадках, коли пікселні дані зображення мають великий діапазон значень, а кількість бітів, що виділено для її зберігання, недостатня для охоплення всього діапазону без виникнення помилок квантування. Наприклад, зображення з пухлиною може мати значно вищі значення порівняно до зображень, що відображають здорові тканини. Таким чином, значення елементів Rescale Intercept і Rescale Slope можуть бути різними для різних зображень навіть в межах однієї отримуваної серії.

8. Пояснити, чому значення елементів Rescale Intercept і Resalale Slope можуть бути різними для різних зображень однієї серії.

При використанні лінійного масштабування зображення з пухлиною може мати значно вищі значення порівняно до зображень, що відображають здорові тканини. Таким чином, значення елементів Rescale Intercept і Rescale Slope можуть бути різними для різних зображень навіть в межах однієї отримуваної серії.

9. Що таке відповідно до стандарту DICOM похідне зображення, як його можна отримати?

Відповідно до стандарту DICOM похідним зображенням вважається будь-яке зображення, для якого пікселні дані були отримані з пікселних даних іншого або декількох інших зображень. Таким чином будь-які операції над пікселними даними зображеннями, у випадку необхідності збереження результату, вимагають створення нового зображення, тобто похідного зображення.

10. Як програмно задати значення унікального ідентифікатору UID, що відповідає стандарту DICOM?

UID за стандартом складається не більш ніж з 64 символів та має первну структуру, наприклад: "1.2.840.xxxxx.3.152.235.2.12.187636473"

За допомогою dcmtk можна автоматично генерувати UID за допомогою функції

```
char *dcmGenerateUniqueIdentifier(char *uid, const char* prefix=NULL)
```