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

Комп’ютерний практикум №5

з дисципліни «Обробка медичних зображень»
на тему: «Фізичні розміри та просторове положення медичних зображень»

Варіант №16

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

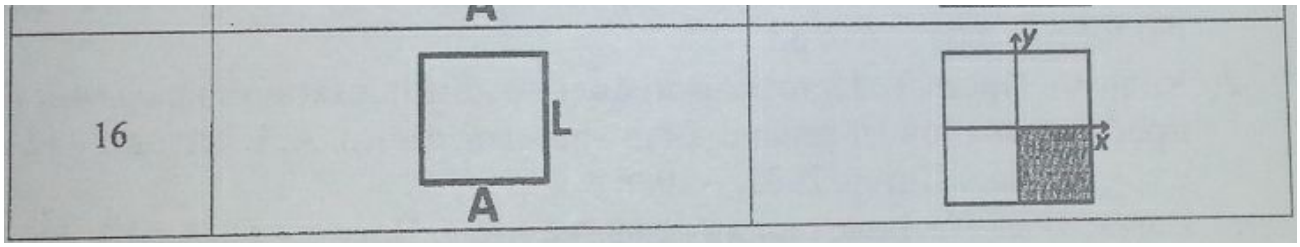
Зараховано від ___.___._____

_____
(підпис викладача)

Київ-2019

Завдання

1. Вивчити теоретичні відомості щодо визначення фізичних розмірів та просторового положення медичних томографічних зображень.
2. Розробити програмний за стосунок для завантаження медичного зображення (томографічного зрізу) в форматі DICOM та визначення тривимірних координат пікселів та просторового положення площини зображення.
3. Завантажити зображення томографічного зрізу в форматі DICOM в оперативну пам’ять та відобразити дані зображення у вікні програми: одному пікселу зображення відповідає один піксел екрану; розміри програмного вікна застосунку (графічна частина) мають бути вдвічі більше відповідних розмірів завантаженого медичного зображення; відображення даних пікселів зображення має бути виконане відповідно до схеми, що зазначена в варіанті завдання; початок координат має бути розташований у центрі.
4. Реалізувати інтерактивне текстове відображення у вікні програмного застосунку значень координат пікселів, що розташовані під курсором миші: під час переміщення миші на даних графічної частини мають відображатися координати у заданій в програмі системі відображення (початок координат – центр графічної частини програмного застосунку); під час переміщення миші на даних завантаженого зображення мають додатково відображатись тривимірні координати в заданій системі координат в заданій під час сканування системі (початок координат – ізоцентр сканера).
5. Реалізувати відображення просторового положення зображення в формі абревіатур необхідних літер (або їх комбінацій) на зображення під час його перегляду.
6. Скласти і захистити звіт по роботі.



Лістинг програми:

main.cpp

```
#include <GL/glew.h>
#include <GL/freeglut.h>
#include <dcmtk/dcmimgle/dcmimage.h>
#include <string>
#include <vector>
#include "Image.h"
#include "TextRenderer.h"
#include "Renderable.h"
#include "DicomFileWrapper.h"
```

```
#include <math.h>

int windowWidth;
int windowHeight;

int imageWidth;
int imageHeight;

std::vector<Renderable *> renderableObjects;
TextRenderer *cursorPositionText;
TextRenderer *scanPositionText;

OFVector<double> imagePosition;
OFVector<double> imageOrientation;

bool leftIsLeft;
bool topIsAnterior;

void render();

void cursorInput(int x, int y);
void displayCursorPosition(int x, int y);
void displayScanPosition(int x, int y);
void swapX(){}
void swapY(){}


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

    // Read image

    DicomFileWrapper dicomFileWrapper("DICOM_Image.dcm");

    imageWidth = dicomFileWrapper.getUShort(DcmTagKey(0x0028, 0x0011));
    imageHeight = dicomFileWrapper.getUShort(DcmTagKey(0x0028, 0x0010));
    auto pixelData = dicomFileWrapper.getUCharArray(DcmTagKey(0x7FE0, 0x0010));

    windowWidth = imageWidth * 2;
    windowHeight = imageHeight * 2;

    imagePosition = dicomFileWrapper.getDoubleArray(DcmTagKey(0x0020, 0x0032), 3);
    imageOrientation = dicomFileWrapper.getDoubleArray(DcmTagKey(0x0020, 0x0037), 6);

    std::string xtext, ytext;
    bool swapX = false, swapY = false;

    if ((abs(imageOrientation.at(0)) + abs(imageOrientation.at(3)) > cos(45 * M_PI / 180)) &&
        (abs(imageOrientation.at(1)) + abs(imageOrientation.at(4)) > cos(45 * M_PI / 180))) //
Axial plane
    {
        xtext.append("L");
        if (imageOrientation.at(0) > 0) {
            leftIsLeft = true;
            swapX = true;
            if (imageOrientation.at(0) != 1) {
                if (abs(imageOrientation.at(1)) != 0) {
                    if (imageOrientation.at(1) > 0)
                        xtext.append("F");
                    else if (imageOrientation.at(1) < 0)
                        xtext.append("H");
                }
                if (abs(imageOrientation.at(2)) != 0) {
                    if (imageOrientation.at(2) > 0)
                        xtext.append("P");
                    else if (imageOrientation.at(2) < 0)
                        xtext.append("A");
                }
            }
        } else if (imageOrientation.at(0) < 0) {
            leftIsLeft = false;
            if (imageOrientation.at(0) != -1) {
                if (abs(imageOrientation.at(1)) != 0) {
                    if (imageOrientation.at(1) > 0)
                        xtext.append("H");
                    else if (imageOrientation.at(1) < 0)
```

```
                    xtext.append("F");
                }
                if (abs(imageOrientation.at(2)) != 0) {
                    if (imageOrientation.at(2) > 0)
                        xtext.append("A");
                    else if (imageOrientation.at(2) < 0)
                        xtext.append("P");
                }
            }
        }
        ytext.append("A");
        if (imageOrientation.at(4) > 0) {
            topIsAnterior = true;
            swapY = true;
            if (imageOrientation.at(4) != 1) {
                if (imageOrientation.at(3) != 0) {
                    if (imageOrientation.at(3) > 0)
                        ytext.append("R");
                    else if (imageOrientation.at(3) < 0)
                        ytext.append("L");
                }
                if (imageOrientation.at(5) != 0) {
                    if (imageOrientation.at(5) > 0)
                        ytext.append("F");
                    else if (imageOrientation.at(5) < 0)
                        ytext.append("H");
                }
            }
        } else if (imageOrientation.at(4) < 0) {
            topIsAnterior = false;
            if (imageOrientation.at(4) != -1) {
                if (imageOrientation.at(3) != 0) {
                    if (imageOrientation.at(3) > 0)
                        ytext.append("L");
                    else if (imageOrientation.at(3) < 0)
                        ytext.append("R");
                }
                if (imageOrientation.at(5) != 0) {
                    if (imageOrientation.at(5) > 0)
                        ytext.append("H");
                    else if (imageOrientation.at(5) < 0)
                        ytext.append("F");
                }
            }
        }
    }

    auto bitsAllocated = dicomFileWrapper.getUShort(DcmTagKey(0x0028, 0x0100));
    auto rescaleIntercept = dicomFileWrapper.getDouble(DcmTagKey(0x0028, 0x1052));
    auto rescaleSlope = dicomFileWrapper.getDouble(DcmTagKey(0x0028, 0x1053));

    GLenum pixelType;

    if (bitsAllocated == 8 && rescaleIntercept == 0 && rescaleSlope == 1)
        pixelType = GL_UNSIGNED_BYTE;
    else if (bitsAllocated == 32 && rescaleIntercept != 0 && rescaleSlope != 1)
        pixelType = GL_FLOAT;
    else
        throw std::runtime_error("Image not supported");

    glutInit(&argc, argv);
    glutInitContextVersion(4, 0);
    glutInitWindowPosition(0, 0);
    glutInitWindowSize(windowWidth, windowHeight);
    glutCreateWindow("Lab5");
    glewInit();

    auto image = new Image(windowWidth, windowHeight, imageWidth, imageHeight,pixelType, pixelData,
swapX, swapY);
    image->setPosition(3.0f * imageWidth / 4.0f, imageHeight / 4.0f);

    renderableObjects.push_back(image);

    cursorPositionText = new TextRenderer("arial.ttf", 14, windowWidth, windowHeight);
    cursorPositionText->setTextPosition(0, windowHeight - 16);
```

```
    cursorPositionText->setTextColor(0, 0, 0);
    renderableObjects.push_back(cursorPositionText);

    scanPositionText = new TextRenderer("arial.ttf", 14, windowWidth, windowHeight);
    scanPositionText->setTextPosition(0, windowHeight - 30);
    cursorPositionText->setTextColor(0, 0, 0);
    renderableObjects.push_back(scanPositionText);

    auto lLabel = new TextRenderer("arial.ttf", 40, windowWidth, windowHeight);
    lLabel->
            setTextPosition(windowWidth
                            - 20, imageHeight / 2);
    lLabel->setText(xtext);
    lLabel->setTextColor(1, 1, 1);
    renderableObjects.
            push_back(lLabel);

    auto aLabel = new TextRenderer("arial.ttf", 40, windowWidth, windowHeight);
    aLabel->
            setTextPosition(imageWidth
                            * 3 / 2 - 10, 0);
    aLabel->setText(ytext);
    aLabel->setTextColor(1, 1, 1);
    renderableObjects.
            push_back(aLabel);

    glutDisplayFunc(render);
    glutPassiveMotionFunc(cursorInput);

    glutMainLoop();

    return 0;
}

void render() {

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

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

    glutSwapBuffers();

}

void cursorInput(int x, int y) {
    x = x - imageWidth;
    y = imageHeight - y;
    displayCursorPosition(x, y);
    displayScanPosition(x, y);
    glutPostRedisplay();
}

void displayCursorPosition(int x, int y) {
    auto text = std::string("Cursor Position: x= ")
            .append(std::to_string(x))
            .append(" y= ")
            .append(std::to_string(y));
    cursorPositionText->setText(text);
}

void displayScanPosition(int x, int y) {

    y = -y;
    double mm_coords[3];

    if (x > 0 && y > 0){
        int imageX, imageY;
        for (int i = 0; i < 3; i++) {
            if (leftIsLeft)
                imageX = x;
            else
                imageX = imageWidth - x;
            mm_coords[i] = imageOrientation.at(i) * imageX;
```

```
            if (topIsAnterior)
                imageY = y;
            else
                imageY = -y + imageHeight;
            mm_coords[i] += imageOrientation.at(i+3) * imageY;
            mm_coords[i] += imagePosition.at(i);
        }

        printf("coords[%i][%i]\n", x, y);
        auto text = std::string("Scan Position: x= ")
                .append(std::to_string(mm_coords[0]))
                .append(" y= ")
                .append(std::to_string(mm_coords[1]))
                .append(" z= ")
                .append(std::to_string(mm_coords[2]));
        scanPositionText->setText(text);

    }
    else
        scanPositionText->setText(std::string("Scan Position: Out of bounds"));
}
```

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:

    glm::vec3 scale, translation;
    float rotation = 0;

    Shader *shader;
    GLuint vao;
    GLuint vbo;
    GLuint ebo;
    GLuint texture;

    void updateModelMatrix();

public:
    Image(int windowWidth, int windowHeight, int imageWidth, int imageHeight, GLenum pixelType,
const void *pixelData, bool swapX, bool swapY);
    ~Image();

    void setPosition(float x, float y);
    void setRotation(float degrees);
    void render() override;

};
```

Image.cpp

```
#include "Image.h"

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

    float textCoord[4][2] = {
            {0, 1},
            {1, 1},
            {1, 0},
            {0, 0},
    };
```

```
    if (swapX) {
        for (int i = 0; i < 4; i++) {
            if(textCoord[i][0] == 0)
                textCoord[i][0] = 1;
            else
                textCoord[i][0] = 0;
        }
    }
    if (swapY) {
        for (int i = 0; i < 4; i++) {
            if(textCoord[i][1] == 0)
                textCoord[i][1] = 1;
            else
                textCoord[i][1] = 0;
        }
    }

    float vertexes[] = {
            //Vertices      //Texture Coords
            -0.5, -0.5,     textCoord[0][0], textCoord[0][1],
            0.5, -0.5,      textCoord[1][0], textCoord[1][1],
            0.5, 0.5,       textCoord[2][0], textCoord[2][1],
            -0.5, 0.5,      textCoord[3][0], textCoord[3][1]
    };
    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);
```

```
    scale = glm::vec3(imageWidth, imageHeight, 1);
    translation = glm::vec3(0);
    updateModelMatrix();

    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);

    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);

}

void Image::setPosition(float x, float y) {
    translation = glm::vec3(x, y, 0);
    updateModelMatrix();
}

void Image::setRotation(float degrees) {
    rotation = glm::radians(degrees);
    updateModelMatrix();
}

void Image::updateModelMatrix() {
    glm::mat4 model = glm::mat4(1.0f);
    model = model * glm::translate(model, translation);
    model = model * glm::rotate(model, rotation, glm::vec3(0,0,1));
    model = glm::scale(model, scale);
    shader->setMatrix4("model", model);
}

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);

}
```

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;

    glm::vec3 textColor;

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 setTextColor(float r, float g, float b);
    void render() override ;
};
```

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;"
            "uniform vec3 textColor;"
            "void main()"
            "{"
            "    vec4 sampled = vec4(1.0, 1.0, 1.0, texture(text, TexCoords).r);"
            "    color = vec4(textColor, 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::setTextColor(float r, float g, float b) {
```

```
    textColor = glm::vec3(r,g,b);
}

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());
        shader->setVec3("textColor", textColor);

        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)

        }

    }

}
```

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);
    void setVec3(const std::string &name, glm::vec3 vector);
    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]);
}

void Shader::setVec3(const std::string &name, glm::vec3 vector) {
    glUniform3fv(glGetUniformLocation(shaderProgram, name.c_str()), 1, &vector[0]);
}
```

DicomFileWrapper.h

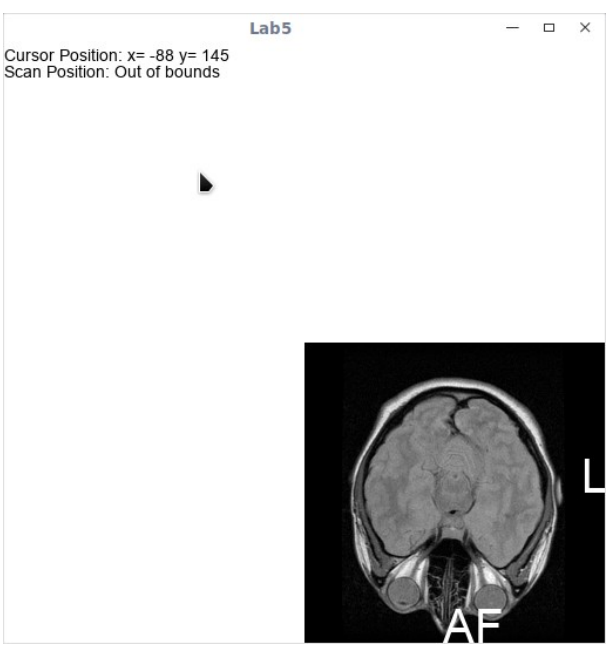
```
#include <iostream>
#include <dcmtk/dcmdata/dctk.h>
#include <dcmtk/dcmimgle/dcmimage.h>

class DicomFileWrapper {
private:
    DcmFileFormat *dcmFileFormat;
public:
    DicomFileWrapper(const std::string &imagePath);
    unsigned short getUShort(const DcmTagKey &dcmTagKey);
    double getDouble(const DcmTagKey& dcmTagKey);
    OFVector<double> getDoubleArray(const DcmTagKey &dcmTagKey, unsigned long size);
    const unsigned char* getUCharArray(const DcmTagKey &dcmTagKey);

};
```

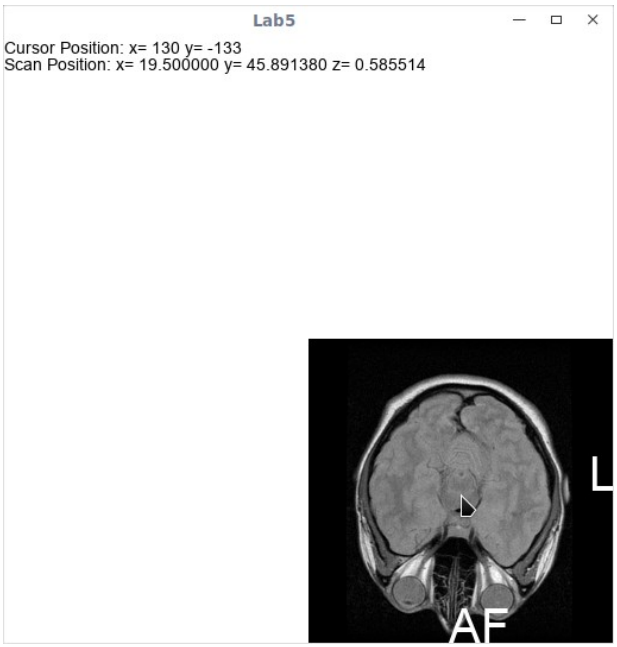
DicomFileWrapper.cpp

```
#include "DicomFileWrapper.h"

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

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

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

OFVector<double> DicomFileWrapper::getDoubleArray(const DcmTagKey& dcmTagKey, unsigned long size) {
    const char * stringValue = nullptr;
    auto condition = dcmFileFormat->getDataset()->findAndGetString(dcmTagKey, stringValue);
    OFVector<Float64> arrayValue;
    OFString ofstr(stringValue);
    DcmDecimalString str(DcmTag(dcmTagKey), ofstr.size());
    str.putString(stringValue);
    auto condition2 = str.getFloat64Vector(arrayValue);
    return arrayValue;
}

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

Результат роботи:





Контрольні запитання

1. *Що таке ізоцентр сканера, в який чин він визначається?*

Ізоцентр сканера – це точка середині магніту з координатами магнітого поля (x, y, z) = 0. В даній точці магнітне поле має напруженість B0 і резонансну частоту 0.

2. *Як розрахувати тривимірні координати певного піксела зображення, який розташований в r-му рядку та в c-му стовпчику?*

$P_{3D}=I_p+r*v_r+c*v_c$,де I_p – координати лівого верхнього кута зображення в міліметрах, v_r та v_c – напрямні косинуси тривимірних векторів напрямку рядків та стовпців відповідно.

3. *Як розрахувати відстань до ізоцентра сканера від певного піксела зображення, який розташований в r-му рядку та в c-му стовпчику?*

Необхідно знайти координати пікселя за формулою, наведеною вище і застосувати формулу для знаходження відстані між точками
PO=√((P_x-0)^2+(P_y-0)^2+(P_z-0)^2)

4. *За якими елементами DICOM визначається інформація щодо фізичних розмірів відсканованого об’єму?*

За елементом Pixel Spacing визначається фізична відстань між пікселями, елемент Specing Between Slices вказує фізичну відстань між зрізами, а Slice Thickness — товщину зрізів.

5. *Чому не можна визначати фізичний розмір відсканованого об'єму за третім виміром, тобто вздовж осі сканування, за інформацією, яка зберігається в елементі DICOM 0018.0050 "Slice Thickness”.*

Оскільки Slice Thickness характеризує лише товщину зрізів, не можна точно встановити розмір відсканованого об’єму, бо сусідні зрізи можуть знаходитись на відстані один від одного, або ж перетинатись.

6. *За якими елементами DICOM визначається інформація щодо просторового положення медичного томографічного зображення?*

Елемент Image Position зберігає фізичну координату роташування першого пікселя в зрізі, а за допомогою елемента Image Orientation, який зберігає напрямні косинуси тривимірних векторів напрямку рядків та стовпців, можна визначити просторову орієнтацію зрізу відносно осей координат.

7. *Визначити напрямні косинуси тривимірних векторів напрямку рядків vr та стовпців vc зображення для косої корональної проекції у випадку її нахилу до трансверсальної площини (окремо для кожного випадку SA, SP, IA, IP).*

SA: (cos 60, cos 90, cos 150) (cos 90, cos 0, cos 90),
SP: (cos 60, cos 90, cos 30) (cos 90, cos 0, cos 90),
IA: (cos 90, cos 0, cos 90) (cos 60, cos 90, cos 150),
IP: (cos 90, cos 0, cos 90) (cos 60, cos 90, cos 15),

8. *Схематично зобразити необхідні літери (або їх комбінацію) для відображення просторової орієнтації зображення, для якого напрямні косинуси, що зберігаються в значенні елементу 0020,0037 “Image Orientation", мають значення 0.5\0\0.8660254\0\1\0.*

(cos 60, cos 90, cos 30) (cos 90, cos 0, cos 90) AHP

9. *За яким елементом D1COM визначається інформація щодо положення пацієнта на столі під час проведення дослідження?*

Елемент Patient position визначає позицію пацієнта по відношенню до сканера.

10. *Визначити напрямні косинуси тривимірних векторів напрямку рядків vc та стовпців vr зображення для восьми можливих положень пацієнта під час сканування: HFP (Head First Prone), HFS (Head First - Supine), HFDR (Head First — Decubitus Right), HFDL (Head First - Decubitus Left), FFDR (Feet First - Decubitus Right), FFDL (Feet First Decubitus Left), FFP (Feet First Prone), FFS (Feet First Supine).*

HFP: vr (-1,0,0) vc (0,-1,0)
HFS: vr (-1,0,0) vc (0,1,0)
HFDR: vr (0,1,0) vc (-1,0,0)
HFDL: vr (0,-1,0) vc (1,0,0)
FFDR: (0,-1,0,-1,0,0)
FFDL: (0,1,0,1,0,0)
FFP: (1,0,0,0,-1,0)
FFS: (1,0,0,0,1,0)