



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

**Комп'ютерний практикум №3**  
з дисципліни «Обробка медичних зображень»  
на тему: «Формалізація уявлень про медичні зображення»

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 основи збереження даних у файлах формату DICOM;
2. Розробити програмний застосунок для завантаження зображення (томографічного зрізу) в форматі DICOM та відображення текстової інформації з елементів DICOM.
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 пікселю зображення відповідає один піксел екрана).
4. Створити події, при обробці яких поверх даних зображення можна відобразити та приховати текстову інформацію з елемента DICOM відповідно до варіанта (варіант 16 — Вага хворого (Patient's Weight)).
5. Скласти і захистити звіт по роботі.

## Хід роботи

### Лістинг програми:

#### main.cpp

```
#include <GL/glew.h>
#include <GL/freeglut.h>
#include <dcmtd/dcmdata/dctk.h>
#include <dcmtd/dcmimgle/dcmimage.h>
#include <iostream>
#include "Image.h"
#include "TextRenderer.h"

int windowHeight = 300;
int windowWidth = 300;

Image* image;
TextRenderer* textRenderer;

void initDrawableObjects();
void render();
void keyboardInput(unsigned char key, int x, int y);

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

    glutInit(&argc, argv);
    glutInitContextVersion(4,0);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(windowWidth, windowHeight);
    glutCreateWindow("Lab3");
    glewInit();

    initDrawableObjects();

    glutDisplayFunc(render);
    glutIdleFunc(render);
    glutKeyboardFunc(keyboardInput);

    glutMainLoop();

    delete image;
    delete textRenderer;

    return 0;
}
```

```

void initDrawableObjects() {
    // Load pixel data and init image

    auto *dicomImage = new DicomImage("DICOM_Image_for_Lab_2.dcm");
    int imageWidth = (int) dicomImage->getWidth();
    int imageHeight = (int) dicomImage->getHeight();
    if(!dicomImage->isMonochrome() || dicomImage->getOutputDataSize() / imageHeight / imageWidth !=
1)
        throw std::runtime_error("Image not supported");
    auto* pixelData = (unsigned char*) dicomImage->getOutputData(8);

    image = new Image(windowWidth, windowHeight, imageWidth, imageHeight, pixelData);

    delete dicomImage;

    // Load patient weight and init text renderer

    DcmFileFormat dcmFileFormat;
    dcmFileFormat.loadFile("DICOM_Image_for_Lab_2.dcm");
    OFString patientWeight;
    DcmTagKey weightTagKey = DcmTagKey(0x0010, 0x1030);
    dcmFileFormat.getDataset()->findAndGetOFString(weightTagKey, patientWeight);

    textRenderer = new TextRenderer("arial.ttf", 14, windowWidth, windowHeight);
    textRenderer->setTextPosition(0, windowHeight - 16);
    textRenderer->setText(std::string("Patient's Weight: ").append(patientWeight.c_str()));
}

void render() {
    glClearColor(1,1,1,1);
    glClear(GL_COLOR_BUFFER_BIT);

    image->render();
    textRenderer->render();

    glutSwapBuffers();
}

void keyboardInput(unsigned char key, int x, int y) {
    textRenderer->toggleVisibility();
}

```

## Shader.h

```

#ifndef LAB3_SHADER_H
#define LAB3_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; }
};

#endif //LAB3_SHADER_H

```

## 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]);
}

```

## Image.h

```

#ifndef LAB3_IMAGE_H
#define LAB3_IMAGE_H

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

class Image {

private:
    Shader *shader;
    GLuint vao;
    GLuint vbo;

    std::vector<float> data;

public:
    Image(int windowHeight, int windowHeight, int imageWidth, int imageHeight, const unsigned char
*pixelData);
    ~Image();

    void render();

};

#endif //LAB3_IMAGE_H

```

## Image.cpp

```

#include "Image.h"

Image::Image(int windowHeight, int windowHeight, int imageWidth, int imageHeight, const unsigned char
*pixelData) {

    // Read pixel data

```

```

    for (int i = 0; i < imageHeight; i++) {
        for (int j = 0; j < imageWidth; j++) {
            unsigned char pixel = *(pixelData + imageWidth * i + j);
            data.push_back(i);
            data.push_back(j);
            data.push_back((float) pixel / 255.0f); //Red
            data.push_back((float) pixel / 255.0f); //Green
            data.push_back((float) pixel / 255.0f); //Blue
        }
    }

    // Init OpenGL Data

    glGenVertexArrays(1, &vao);
    glBindVertexArray(vao);

    glGenBuffers(1, &vbo);
    glBindBuffer(GL_ARRAY_BUFFER, vbo);
    glBufferData(GL_ARRAY_BUFFER, data.size() * sizeof(float), &data[0], GL_STATIC_DRAW);

    auto vertexShader =
        "#version 400\n"
        "in vec2 position; "
        "in vec3 inColor; "
        "out vec3 vertexColor; "
        "uniform mat4 model;"
        "uniform mat4 view;"
        "uniform mat4 projection;"
        "void main()"
        "{"
        "    gl_Position = projection * view * model * vec4(position, 0.0, 1.0);"
        "    vertexColor = inColor;"
        "}";

    auto fragmentShader =
        "#version 400\n"
        "in vec3 vertexColor;"
        "out vec4 FragColor;"
        "void main()"
        "{"
        "    FragColor = vec4(vertexColor, 1.0);"
        "}";

    shader = new Shader(vertexShader, fragmentShader);

    glm::mat4 modelMatrix = glm::mat4(1.0f);
    glm::mat4 viewMatrix = glm::lookAt(glm::vec3(0,0,1), glm::vec3(0,0,0), glm::vec3(0,1,0));
    glm::mat4 projectionMatrix = glm::ortho(0.0f, (float) windowWidth,0.0f,(float) windowHeight,
1.0f, -1.0f);

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

    glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 5 * sizeof(float), (void*)0);
    glEnableVertexAttribArray(0);

    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 5 * sizeof(float), (void*)(2 * sizeof(int)));
    glEnableVertexAttribArray(1);

}

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

void Image::render() {

    glUseProgram(shader->getReference());
    glBindVertexArray(vao);
    glDrawArrays(GL_POINTS, 0, data.size() / 5);

}

```

TextRenderer.h

```

#ifndef LAB3_TEXTRENDERER_H
#define LAB3_TEXTRENDERER_H

#include <vector>
#include <string>
#include "Shader.h"
#include <ft2build.h>
#include FT_FREETYPE_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 {
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();
};

#endif //LAB3_TEXTRENDERER_H

```

## 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];

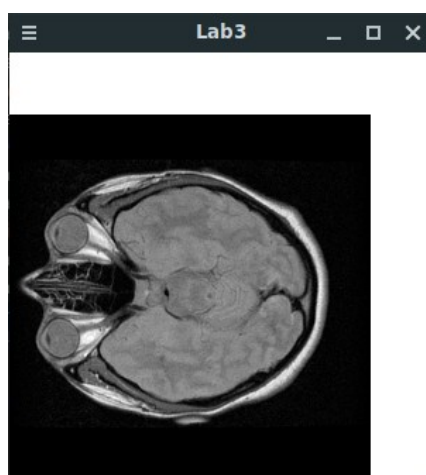
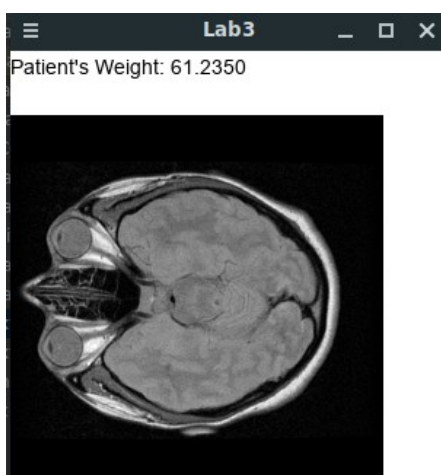
            GLfloat xpos = x + ch.Bearing.x * scale;
            GLfloat ypos = y - (ch.Size.y - ch.Bearing.y) * scale;

            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)
        }
    }
}

```

## Результат роботи:



При натисненні будь-якої клавіші клавіатури надпис змінює видимість



## Контрольні запитання

### 1. Стандарт DICOM, його основні призначення.

DICOM – стандарт, який визначає структуру файлів з медичними зображеннями та стандартизує передачу таких файлів мережею. Метою використання цього стандарту є уніфікація медичних зображень які отримані з пристроїв від різних виробників, збереження медичних зображень та супутньої інформації про пацієнта, пристрій, тощо.

### 2. Навіщо і що саме визначається в описі відповідності стандарту DICOM, що є обов'язковою документацією до пристроїв медичної візуалізації?

Цей опис говорить про те яким чином та які функції виконує даний пристрій медичної візуалізації з метою стандартизувати процес комунікації між різними системами.

### 3. Що саме визначає розмір файлу зображення, що зберігається в форматі DICOM? Відповідь пояснити.

Розмір файлу зображення визначає кількість зображень, кількість даних про пацієнта та діагностичний прилад, алгоритми для стиснення зображень.

### 4. Як реалізувати найпростіший парсер (зчитувач) DICOM файлів, щоб можна було отримати значення елемента за його тегом?

Для отримання інформації за певним тегом необхідно послідовно просканувати файл, що являє собою сукупність елементів даних, в кожному з яких є поля з назвою тега, типом та довжиною даних і поєднанням значенням потрібної інформації.

### 5. Що визначає інформаційна модель за стандартом DICOM?

Вона визначає структуру і організацію інформації, пов'язану з медичними зображеннями. В файлі формату DICOM це представлено у вигляді ієрархії: Пацієнт – дослідження – серія - зображення

### 6. Що за стандартом DICOM визначає термін «складне зображення»?

Складне зображення є елементом серії зображень. Крім самого зображення складне зображення може містити криві, оверлейні дані, таблиці та ін.

### 7. Як завантажити піксельні дані зображення з DICOM файлу в оперативну пам'ять?

Для цього потрібно зчитати інформацію про розмір зображення та його тип, а далі отримати інформацію про кожен піксель з елемента Pixel Data

### 8. Що таке за стандартом DICOM однофреймові та мультифреймові зображення, коли їх можна одержати?

Мультифреймове зображення означає що в файлі присутні декілька зображень. Такі зображення зазвичай створюються за допомогою апаратів УЗД та ангиографів.

### 9. Які модальності зображень визначені за стандартом DICOM?

CR	Computed Radiography
CT	Computed Tomography
MR	Magnetic Resonance
US	Ultrasound
OT	Other
BI	Biomagnetic imaging
CD	Color flow Doppler
DD	Duplex Doppler
DG	Diaphanography
ES	Endoscopy
LS	Laser surface scan

та інші

*10. Навіщо використовують упаковку піксельних даних зображення, чи впливає це на якість?*

Це дозволяє у деяких випадках значно зменшити розмір файлу. Якість залежить від алгоритму упаковки – бувають алгоритми без втрат якості, але деякі алгоритми знижують якість.