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

Комп’ютерний практикум №1

з дисципліни «Обробка медичних зображень»
на тему: «Сучасні бібліотеки для роботи з графічними даними»

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. Ознайомитися з сучасними бібліотеками для роботи з графічними даними та теоретичними основами щодо проведення медичної візуалізації.

2. Провести аналіз та обрати середовище розробки програмних застосунків, мову програмування та необхідні сторонні бібліотеки для роботи з графічними даними та для роботи з файлами медичних зображень; обгрунтувати свій вибір.

3. Скласти та захистити звіт по роботі.

Для виконання комп'ютерних практикумів було обрано наступні засоби:

Мова програмування C++:

1. повна сумісність з бібліотекою OpenGL, яка написана на C
2. доступ до ручного керування пам'яттю
3. швидкість роботи за рахунок нативної компіляції для контретної платформи

Середовище розробки CLion:

1. крос-платформність
2. нативна підтримка C++
3. використання CMake для збирання проектів, що дозволить легко змінити середовище розробки при необхідності
4. інтелектуальні системи для доповнення та аналізу коду

Бібліотека для роботи з вікнами FreeGlut:

1. надає інстументи для автоматизації циклу рендера та обробки подій миші та клавіатури
2. велика кількість навчальних матеріалів
3. відкрита ліцензія (на відміну від оригінальної бібліотеки GLUT)

Бібілотека для роботи з DICOM – DCMTK:

1. дозволяє зчитувати, редагувати та створювати файли відповідно до стандарту DICOM
2. містить інструменти для роботи з закодованими зображеннями та їх обробки
3. включає реалізацію мережевого протоколу стандарту DICOM

Приклад роботи з бібліотекою OpenGL

main.cpp

```
#include <GL/glew.h>
#include <GL/freeglut.h>
#include <iostream>
#include "Square.h"

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

Square* square;

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

    glutInit(&argc, argv);
    glutInitContextVersion(4,0);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(400,400);
    glutCreateWindow("Test");

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

    glewInit();

    square = new Square();

    glutMainLoop();

    delete square;

    return 0;
}

void render() {

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

    square->render();

    glutSwapBuffers();

    calculateFPS();

}

void keyboardInput(unsigned char key, int x, int y) {
```

```
    printf("key presed: %i \n", key);

    switch(key) {
        case 'w':
            square->shiftPosition(0.0f, 0.1f);
            break;
        case 's':
            square->shiftPosition(0.0f, -0.1f);
            break;
        case 'a':
            square->shiftPosition(-0.1f, 0);
            break;
        case 'd':
            square->shiftPosition(0.1f, 0);
            break;
    }

}

int framesCount = 0, lastTimeFpsCount = 0;
char title[220];
void calculateFPS() {

    framesCount++;

    int currentTime = glutGet(GLUT_ELAPSED_TIME);

    if (currentTime - lastTimeFpsCount > 1000) {

        float fps = framesCount * 1000.0f / (currentTime - lastTimeFpsCount);
        lastTimeFpsCount = currentTime;
        framesCount = 0;

        sprintf(title, "OpenGLTest, FPS = %4.2f", fps);
        glutSetWindowTitle(title);

    }

}
```

Square.h

```
#ifndef TEST2_SQUARES_AND_TEXTURES_SQUARE_H
#define TEST2_SQUARES_AND_TEXTURES_SQUARE_H

#include <GL/glew.h>

#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>

class Square {

private:
```

```
    float vertexes[8] = {
        0.5, 0.5,
        -0.5, 0.5,
        -0.5, -0.5,
        0.5, -0.5
    };

    glm::mat4 transform;

    GLuint vao;
    GLuint vbo;
    GLuint ebo;
    GLuint shaderProgram;
    GLuint transformUniformLocation;

public:
    Square();
    ~Square();
    void render();
    void shiftPosition(float x, float y);

};

#endif //TEST2_SQUARES_AND_TEXTURES_SQUARE_H
```

Square.cpp

```
#include "Square.h"
#include "ShaderManager.h"

Square::Square() {

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

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

    GLuint vertexShader = ShaderManager::LoadShader("shaders/vertexShader.glsl",
GL_VERTEX_SHADER);
    GLuint fragmentShader = ShaderManager::LoadShader("shaders/fragmentShader.glsl",
GL_FRAGMENT_SHADER);

    shaderProgram = glCreateProgram();
    glAttachShader(shaderProgram, vertexShader);
    glAttachShader(shaderProgram, fragmentShader);
    glBindFragDataLocation(shaderProgram, 0, "outColor");
```

```
    glLinkProgram(shaderProgram);
    glUseProgram(shaderProgram);


    GLint posAttr = glGetAttribLocation(shaderProgram, "position");
    glVertexAttribPointer(posAttr, 2, GL_FLOAT, GL_FALSE, 0, nullptr);
    glEnableVertexAttribArray(posAttr);


    transform = glm::mat4(1.0f);
    transformUniformLocation = glGetUniformLocation(shaderProgram, "transform");
    glUniformMatrix4fv(transformUniformLocation, 1, GL_FALSE, glm::value_ptr(transform));


}


Square::~Square() {
    glDeleteBuffers(1, &vbo);
    glDeleteProgram(shaderProgram);
}


void Square::render() {


    glBindVertexArray(vao);
    glUniformMatrix4fv(transformUniformLocation, 1, GL_FALSE, glm::value_ptr(transform));
    glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, nullptr);


}


void Square::shiftPosition(float x, float y) {
    transform = glm::translate(transform, glm::vec3(x, y, 0.0f));
}
```

ShaderManager.h

```
#ifndef TEST2_SQUARES_AND_TEXTURES_SHADERMANAGER_H
#define TEST2_SQUARES_AND_TEXTURES_SHADERMANAGER_H


#include <GL/glew.h>
#include <iostream>


class ShaderManager {


private:
    static std::string readFileToString(const std::string& fileName);


public:
    static GLuint LoadShader(const std::string& fileName, GLenum shaderType);


};


#endif //TEST2_SQUARES_AND_TEXTURES_SHADERMANAGER_H
```

ShaderManager.cpp

```
#include "ShaderManager.h"
#include <fstream>


std::string ShaderManager::readFileToString(const std::string& fileName) {
```

```
    std::ifstream file(fileName);

    if(!file.is_open())
        throw std::invalid_argument("File " + fileName + " not found");

    std::string contents((std::istreambuf_iterator<char>(file)),
                         std::istreambuf_iterator<char>());

    return contents;
}

GLuint ShaderManager::LoadShader(const std::string& fileName, GLenum shaderType) {

    //Load and Compile Shader

    auto shaderScript = readFileToString(fileName);
    auto constScript = shaderScript.c_str();
    GLuint shader = glCreateShader(shaderType);
    glShaderSource(shader, 1, &constScript, 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("Could load shader " + fileName + " " + buffer);
    }

    return shader;
}
```

vertexShader.glsl

```
#version 400

uniform mat4 transform;

in vec2 position;

void main()
{
    gl_Position = transform * vec4(position, 0.0, 1.0);
}
```

fragmentShader.glsl

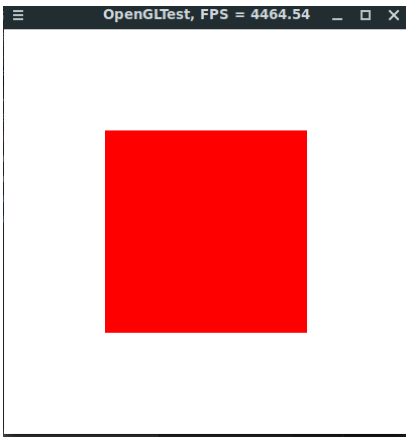
```
#version 400

out vec4 outColor;

void main()
{
```

```
    outColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

Результат роботи:



Приклад роботи з бібліотекою DCMTK

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

using namespace std;

int main() {

    string in_file {"77654033/20010101/CR1/6154"};

    DcmFileFormat file_format;
    OFCondition status = file_format.loadFile(in_file.c_str());

    if (status.bad()) {
        cerr << "Problem opening file:" << in_file << endl;
        return 1;
    }

    DcmDataset* dataset = file_format.getDataset();

    OFString patient_name;
    OFCondition condition;
    condition = dataset->findAndGetOFStringArray(DCM_PatientName, patient_name);

    if (condition.good())  {
        cout << "Patient name is: " << patient_name << endl;
    } else {
        cerr << "Could not get patient name" << endl;
    }
```

```
    return 0;
}
```

Результат роботи:

```
/home/syt0r/CLionProjects/Lab1/cmake-build-debug/Lab1
Patient name is: Doe^Archibald

Process finished with exit code 0
```

Контрольні запитання

1. Коротко опишіть архітектуру бібліотеки OpenGL і організацію конвеєра графічних перетворень.

Всі доступні функції і константи зберігаються в основній бібліотеці OpenGL, константи починаються з префіксу GL_, а функції з gl. Оскільки бібліотека OpenGL являє собою лише API для маніпуляцій зображенням то для роботи з віконною системою ОС, математичних розрахунків та решти дій використовуються сторонні бібліотеки. Також OpenGL підтримує розширення для оптимізації або додавання нового функціоналу які можуть розробляти окремі компанії, але оскільки вони не входять до стандартної специфікації то доступні лише для використання лише у випадку підтримки такого функціоналу встановленим драйвером.

При рендері зображення за допомогою OpenGL виконується певна послідовність кроків, яка називається Rendering Pipeline. Виконуються такі кроки:

1. Обробка вершин: включає отримання інформації про вершини з буфера і обробка кожної вершини за допомогою шейдерів типу vertex або geometry.
2. Пост-обробка вершин
3. Створення примітивів
4. Перетворення та інтерполяція параметрів примітивів
5. Обробка фрагментів за допомогою шейдерів
6. Обробка семплів: обробка отриманих даних з шейдерів та запис даних до різних буферів

2. Назвіть категорії базових команд (функцій) бібліотеки.

1. Функції опису примітивів
2. Функції опису джерел світла
3. Функції завдання атрибутів
4. Функції візуалізації
5. Функції геометричних перетворень

3. Навіщо потрібні різні варіанти команд OpenGL, що відрізняються тільки типами параметрів?

Оскільки бібліотека OpengGL написана мовою C, яка на відміну від C++ не підтримує перевантаження фунцій, для різних типів параметрів потрібні різні функції. Так наприклад для предачі даних до шейдерів є функції glUniform1f, glUniform1ui, glUniformMatrix2fv та інші.

4. Чому організацію OpenGL часто порівнюють із скінченним автоматом?

OpenGL містить набір змінних, що визначають яким чином мають відбуватися дії після встановлення певного режиму. Набір станів, які задано такими змінними, використовується для відображення різних типів даних в межах одного вікна. При використанні OpenGL відбувається зміна різних типів станів, маніпуляція буферами даних і рендер за допомогою використання обраного стану.