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

Комп’ютерний практикум №2

з дисципліни «Обробка медичних зображень»
на тему: «Математичний апарат візуалізації геометричних примітивів»

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 роботи з графічною бібліотекою OpenGL.
2. Відповідно до свого варіанта вивести формули для побудови в декартовій системі координат геометричного примітиву із заданою зв'язністю пікселів.
3. Розробити програмний застосунок для відображення геометричного примітиву та переведення координат точок з однієї системи координат в іншу.
4. Розміри вікна програмного застосунку задати довільними; центр координат розмістити в центрі частини вікна програмного застосунку для візуалізації графічних даних (без інтерфейсу користувача); провести візуалізацію осей координат.
5. Розробити інтерфейс користувача для інтерактивного визначення геометричного примітиву в початковій системі координат відповідно до свого варіанта: відрізок задається через координати точок початку та кінця відрізка, коло – через координати центра та розмір радіуса в пікселях екрану.
6. Створити подію, при обробці якої виникає переведення та відображення інтерактивно заданих точок в новій системі координат.
7. Відповідно до свого варіанта побудувати фігуру в декартовій системі координат та відобразити на екрані.
8. Скласти і захистити звіт по роботі.

Номер варіанта	Об'єкт відобаження	Зв'язність пікселів	Початкова система координат	Нова система координат
16	Коло	8	Сферична	Декартова

Хід роботи

Через симетричність кола, для його побудови достатньо побудувати лише його 1/8 частину, решту точок можна отримати з вже отриманих. Побудуємо точки кола у верхній частині першої чверті.

Розглянемо точку (x_i, y_i) . Оскільки маємо 8 зв'язні пікселі, можливі лише 2 варіанти руху до наступної точки: вправо (точка S (x_{i+1}, y_i)), і по діагоналі (точка T (x_{i+1}, y_{i-1})). При виборі кожної точки буде існувати похибка, оскільки обрані точки не лежать прямо на колі, тому обирати потрібно точку, похибка якої найменша.

Розрахуємо похибку (D) для кожної з точок:

$$D(S_i)=(x+1)^2+y^2-R^2$$

$$D(T_i)=(x+1)^2+(y-1)^2-R^2$$

Для вибору точки візьмемо загальну похибку d_i

$$d_i = D(S_i) + D(T_i) = (x_i+1)^2 + y_i^2 - R^2 + (x_i+1)^2 + (y_i-1)^2 - R^2 = 2(x_i+1)^2 + y_i^2 + (y_i-1)^2 - 2R^2$$

Знайдемо d_{i+1}

$$d_{i+1} = D(S_{i+1}) + D(T_{i+1}) = 2(x_i+2)^2 + y_{i+1}^2 + (y_{i+1}-1)^2 - 2R^2$$

Візьмемо різницю похибок між ітераціями

$$d_{i+1} - d_i = 2((x_i+2)^2 - (x_i+1)^2) + (y_{i+1}^2 - y_i^2) + ((y_{i+1}-1)^2 - (y_i-1)^2)$$

$$d_{i+1} = d_i + 2(x_i+2+x_i+1)(x_i+2-x_i-1) + (y_{i+1}+y_i)(y_{i+1}-y_i) + (y_{i+1}-1+y_i-1)(y_{i+1}-1-y_i+1)$$

$$d_{i+1} = d_i + 2(2x_i+3) + (y_{i+1}+y_i)(y_{i+1}-y_i) + (y_{i+1}-1+y_i-1)(y_{i+1}-1-y_i+1)$$

Розглянемо випадок di < 0, отже на кроці і буде обрано точку S

$$y_{i+1} = y_i$$

$$d_{i+1} = d_i + 2(2x_i+3) = d_i + 4x_i + 6$$

Розглянемо випадок di >= 0, отже на кроці і буде обрано точку T

$$y_{i+1} = y_i - 1$$

$$d_{i+1} = d_i + 2(2x_i+3) + (2y_i-1)(-1) + (2y_i-3)(-1)$$

$$d_{i+1} = d_i + 4x_i + 6 - 2y_i - 2y_i + 3 = d_i + 4(x_i - y_i) + 10$$

В якості початкової точки оберемо точку x = 0, y = R, де R – радіус кола.

$$d_0 = 2 + R^2 + (R-1)^2 - 2R^2 = 2 + 2R^2 - 2R + 1 - 2R^2 = 3 - 2R$$

Лістинг програми:

main.cpp

```
#include <GL/glew.h>
#include <GL/freeglut.h>
#include <iostream>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include "Shader.h"
#include "CoordinateSystem.h"
#include "Circle.h"

int width = 600;
int height = 600;

Shader* shader;
CoordinateSystem* coordinateSystem;
Circle* circle;

void init(int argc, char* argv[]);
void display(SphericalCoordinates center, int radius);
void render();

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

    SphericalCoordinates center{};
    int radius;

    std::cout << "Circle center radial distance = ";
    std::cin >> center.radialDistance;

    std::cout << "Circle center theta = ";
    std::cin >> center.theta;
    center.theta = glm::radians(center.theta);

    std::cout << "Circle center phi = ";
```

```
    std::cin >> center.phi;
    center.phi = glm::radians(center.phi);

    std::cout << "Circle radius = ";
    std::cin >> radius;


    init(argc, argv);
    display(center, radius);

    return 0;
}

void init(int argc, char* argv[]) {

    glutInit(&argc, argv);
    glutInitContextVersion(4, 0);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(600, 600);
    glutCreateWindow("Lab 2. Spherical to Cortasian");

    glewInit();

    glutDisplayFunc(render);

}

void display(SphericalCoordinates center, int radius) {

    shader = new Shader();

    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(-width/2.0f, width/2.0f, -height/2.0f, height/2.0f,
1.0f, -1.0f);

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

    coordinateSystem = new CoordinateSystem(shader, width, height);
    circle = new Circle(shader);

    circle->setSphericalParameters(center, radius);

    glutMainLoop();

    delete shader;
    delete coordinateSystem;
    delete circle;

}

void render() {

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

    coordinateSystem->render();
    circle->render();

    glutSwapBuffers();

}
```

Shader.h

```
#ifndef LAB2_SHADER_H
#define LAB2_SHADER_H

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

class Shader {
```

```
private:
    GLuint shaderProgram;
    GLuint compileShader(const char *shaderText, GLenum shaderType);
public:
    Shader();
    void setMatrix4(const std::string &name, glm::mat4 matrix);
    GLuint getShader() { return shaderProgram; }

};


#endif //LAB2_SHADER_H
```

Shader.cpp

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

Shader::Shader() {

    GLuint vertexShader = compileShader(
            "#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;"
            "}", GL_VERTEX_SHADER);

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

CoordinateSystem.h

```
#ifndef LAB2_COORDINATESYSTEM_H
#define LAB2_COORDINATESYSTEM_H

#include <GL/glew.h>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
#include "Circle.h"
#include "Shader.h"

class CoordinateSystem {

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

public:
    CoordinateSystem(Shader* shader, int width, int height);
    void render();

};


#endif //LAB2_COORDINATESYSTEM_H
```

CoordinateSystem.cpp

```
#include "CoordinateSystem.h"

CoordinateSystem::CoordinateSystem(Shader* shader, int width, int height) : shader(shader) {

    int vertices[] = {
            //X
            //Coordinates   //Colors
            -width / 2, 0,  0, 0, 0,
            width / 2, 0,   0, 0, 0,
            //Y
            //Coordinates   //Colors
            0, height / 2,  0, 0, 0,
            0, -height / 2, 0, 0, 0
    };

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

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

    glVertexAttribPointer(0, 2, GL_INT, GL_FALSE, 5 * sizeof(int), (void*)0);
    glEnableVertexAttribArray(0);

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

}

void CoordinateSystem::render() {
    glUseProgram(shader->getShader());
    glBindVertexArray(vao);
    glDrawArrays(GL_LINES, 0, 4);
}
```

Circle.h

```
#ifndef LAB2_CIRCLE_H
#define LAB2_CIRCLE_H

#include <GL/glew.h>
#include <vector>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
#include "Shader.h"
```

```
struct CortasianCoordinates {
    float x;
    float y;
    float z;
};

struct SphericalCoordinates {
    float radialDistance;
    float theta;
    float phi;
};

class Circle {

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

    std::vector<float> verticesData;

    void addVertex(float x, float y, float r, float g, float b);

public:

    explicit Circle(Shader* shader);

    void setCortesianParameters(CortasianCoordinates coordinates, float radius);
    void setSphericalParameters(SphericalCoordinates center, float radius);

    void render();

};

#endif //LAB2_CIRCLE_H
```

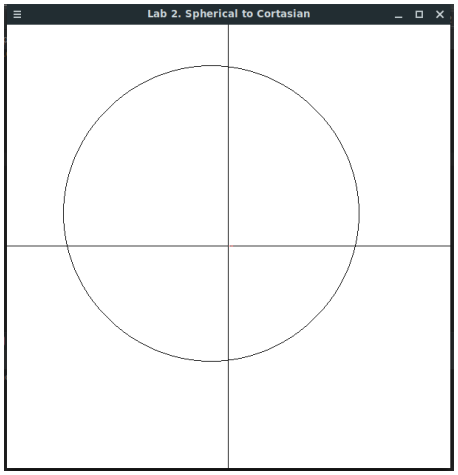
Circle.cpp

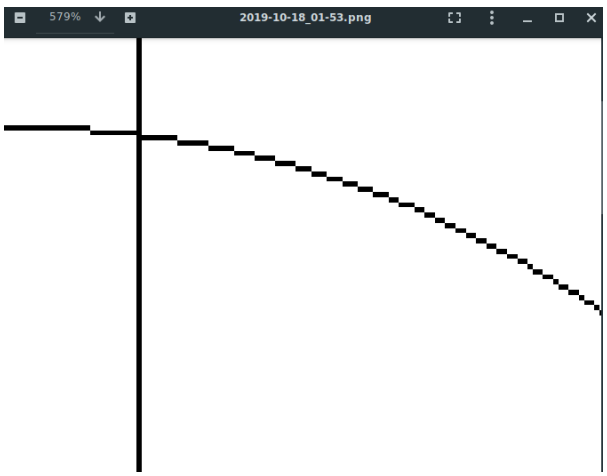
```
#include <cmath>
#include <vector>
#include "Circle.h"

Circle::Circle(Shader *shader) : shader(shader) {

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

    glGenBuffers(1, &vbo);
    glBindBuffer(GL_ARRAY_BUFFER, vbo);

}

void Circle::setSphericalParameters(SphericalCoordinates center, float radius) {

    CortasianCoordinates cortasianCenter{};

    cortasianCenter.x = center.radialDistance * std::sin(center.theta) * std::cos(center.phi);
    cortasianCenter.y = center.radialDistance * std::sin(center.theta) * std::sin(center.phi);
    cortasianCenter.z = center.radialDistance * std::cos(center.phi);

    setCortesianParameters(cortasianCenter, radius);

}

void Circle::setCortesianParameters(CortasianCoordinates center, float radius) {

    int x = 0;
    int y = radius;
    float d = 3 - 2 * radius;

    do {
        //Create vertices
        addVertex(center.x + x, center.y + y, 0, 0, 0);
        addVertex(center.x - x, center.y - y, 0, 0, 0);
        addVertex(center.x + x, center.y - y, 0, 0, 0);
        addVertex(center.x - x, center.y + y, 0, 0, 0);

        addVertex(center.x + y, center.y + x, 0, 0, 0);
```

```
        addVertex(center.x - y, center.y - x, 0, 0, 0);
        addVertex(center.x + y, center.y - x, 0, 0, 0);
        addVertex(center.x - y, center.y + x, 0, 0, 0);

        x = x + 1;

        if(d < 0) {
            d = d + 4 * x + 6;
        }
        else {
            y = y - 1;
            d = d + 4 * (x - y) + 10;
        }

    } while (y >= x);

    addVertex(0, 0, 1, 1, 1);
    addVertex(1, 0, 1, 0, 0);
    addVertex(3, 0, 1, 0, 0);
    addVertex(5, 0, 1, 0, 0);

    glBindBuffer(GL_ARRAY_BUFFER, vbo);
    glBufferData(GL_ARRAY_BUFFER, verticesData.size() * sizeof(float), &verticesData[0],
GL_DYNAMIC_DRAW);

    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);

}

void Circle::addVertex(float x, float y, float r, float g, float b) {
    verticesData.push_back(x);
    verticesData.push_back(y);
    verticesData.push_back(r);
    verticesData.push_back(g);
    verticesData.push_back(b);
}

void Circle::render() {
    glUseProgram(shader->getShader());
    glBindVertexArray(vao);
    glDrawArrays(GL_POINTS, 0, verticesData.size() / 5);
}
```

Результат роботи:

```
"/home/syt0r/CLionProjects/16 - Shuliak/16_Pract_2/16_Pract_2_Code/cmake-build-debug/Lab2"
Circle center radial distance = 50
Circle center theta = 90
Circle center phi = 118
Circle radius = 200
```





Контрольні запитання

1. *Чим пояснюється необхідність отримання растрового подання геометричних об'єктів при розробці графічних програмних застосунків?*

Растрове подання необхідне для економії ресурсів для обчислень та дозволяє вивести результат на монітор, який використовує працює за таким принципом.

2. *В чому полягають недоліки найпростіших алгоритмів отримання растрового подання лінії?*

   В цих алгоритмах на кожному кроці виконуються операції з дробовими числами та операції округлення, які є значно ресурсоємнішими за операції додавання цілих чисел в алгоритмі Брезенхейма

3. *В чому полягає основна ідея алгоритму Брезенхейма для отримання растрового подання лінії?*

   Ідея полягає у ітеративному використанні змінної зі значенням похибки, яка дозволяє визначити наступну точку з мінімальним відхиленням і лише з використанням операцій додавання та множення.

4. *В чому полягає основна ідея алгоритму Брезенхейма для отримання растрового подання кола?*

   Ідея полягає у використанні властивості симетричності кола, завдяки чому потрібно знайти лише точки з 1/8 частини кола, та ітеративному використанні змінної похибки, що дозволяє вибрати наступну точку яка мінімізує відхилення.