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

# Розрахунково-графічна робота

з дисципліни «Обробка медичних зображень»

Варіант №23

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

Зараховано від ___.___._____

_____
(підпис викладача)

Київ-2019

**Зміст**

## Розділ 1. Постановка завдання

1. Відповідно до свого варіанта розрахувати тривимірні координати пікселя зображення, який розташований в заданому рядку та стовпчику.

2. Розробити програмний застосунок для визначення просторового положення піксельних даних томографічних зображень та визначення тривимірних координат пікселя зображення, який розташований у заданому рядку та стовпчику; розміри програмного вікна застосунку (графічна частина) мають бути достатніми для візуалізації всіх даних, що передбачені в завданні до РГР.

3. Реалізувати вивід текстового повідомлення із тривимірними координатами пікселя зображення, який розташований в заданому відповідно до варіанта рядку та стовпчику.

4. Реалізувати у вікні програмного застосунку графічне відображення координатних осей із початком координат в ізоцентрі сканера; границь об’єму, який визначає отримана під час дослідження серія томографічних зображень; площини томографічного зображення, що визначене в просторі відповідно до варіанта, та положення пікселя, який розташований в заданому рядку та стовпчику цього зображення.

5. Скласти і захистити звіт по роботі.

| Номер варіанта | 0028,0010 Rows | 0028,0011 Columns | 0028,0030 Pixel Spacing | 0018,0050 Slice Thickness | 0018,0088 Spacing Between Slices | Координати пікселя в площині зображення |
|---|---|---|---|---|---|---|
| 23 | 512 | 512 | 0.703125\0.703125 | 5.00 | 1.25 | (367;298) |

| Номер варіанта | 0020,0032 Image Position, шуканий зріз (25й) | 0020,0037 Image Orientation, шуканий зріз |
|---|---|---|
| 23 | -157.4\-180.0\-126.75 | 1.0\0.0\0.0\0.0\1.0\0.0 |

| Номер варіанта | 0020,0032 Image Position, 1й зріз | 0020,0037 Image Orientaion, 1й зріз | Кількість зображень в серії |
|---|---|---|---|
| 23 | -157.4\-180.0\28.25 | 1.0\0.0\0.0\0.0\1.0\0.0 | 79 |

## Розділ 2. Теоретичні відомості

DICOM містить ряд елементів, що дають змогу визначити фізичні координати отрииманих зображень. Розглянемо деякі з них.

Елемент 0020,0032 "Image Position" зберігає інформацію щодо x, y, z координат лівого верхнього кута (першого піксела) зображення в міліметрах. Це надає можливість визначити, де зображення знаходиться в тривимірному просторі.

Елемент 0020,0037 "Image Orientation" зберігає напрямні косинуси тривимірних векторів напрямку рядків vr (перші три значення, що зберігаються в елементі) та стовпців vc (наступні три значення, що зберігаються в елементів) зображення. Ці два вектори, що починаються з координат піксела заданих в Image Position елементі, однозначно визначають площину зображення в тривимірному просторі. Наприклад, значення вектора vr (1,0,0), яке відповідає (cos 0°. cos 90°. cos 90°), означає, що напрямок рядків зображення точно співпадає з напрямком осі Ox, а значення вектора vc (0,1.0), яке відповідає (cos 90°. cos 0°. cos 90°). означає, що напрямок стовпців зображення точно співпадає з напрямком осі Oy.

Для піксела зображення P, який розташований в r-му рядку та в c-му стовпчику, тривимірні координати до ізоцентра сканера розраховуються відповідно до формули:

$$P_{3D} = I_p + r * v_r + c * v_c \quad (1)$$

,де $I_p$ – координати лівого верхнього кута зображення в міліметрах, $v_r$ та $v_c$ – напрямні косинуси тривимірних векторів напрямку рядків та стовпців відповідно

В стандарті DICOM використовується так звана «відносна система координат» (RCS - Reference Coordinates System), яка визначає просторове положення зображення шодо тіла людини під час проведення сканування. Початок координат такої системи розташований в ізоцентрі сканера, що в більшості випадків визначається лазерними пристроями для зовнішнього вирівнювання положення пацієнта на столі перед початком проведення кожного нового дослідження (тобто для початкового положення столу). Ізоцентр сканера – це точка середині магніту з координатами магнітого поля (x, y, z) = 0.

Елемент 0028,0030 "Pixel Spacing" зберігає інформацію щодо фізичних розмірів пікселів зображення в міліметрах. Це надає можливість проводити реальні вимірювання відстаней на зображенні. Знаючи роздільність зображення в пікселах, можна визначити його фізичні розміри (область перегляду, FoV).

Інформація щодо товщини томографічного зрізу в міліметрах зберігається в елементі 0018,0050 "Slice Thickness".

Елемент 0018,0088 "Spacing Between Slices" містить інформацію щодо фізичної відстані між сусідніми томографічними зрізами в міліметрах.

## Розділ 3. Розрахунок координат пікселя

Розрахуємо координати пікселя, даного в завданні. Піксель знаходиться в 25 зрізі, перший піксель якого має координати (-157.4; -180.0; -126.75). За даними елемента Image Orientation видно, що площина зрізу співпадає з площиною xOy, отже координата z шуканого пікселя буде такою ж як і в першого пікселя зрізу.

Координати шуканого пікселя (367; 298). Для того щоб знайти його положення в просторі необхідно застосувати формулу (1) з поправкою на значення Pixel Spacing (0.703125; 0.703125). В результаті отримуємо:

x = -157.4 + 298 *1*0.703125 + 367 * 0 *  0.703125 = 52.13125 мм

y = -180 + 298*0* 0.703125 + 367 * 1 *  0.703125 = 78.046875 мм

z = -126.75 мм

Далі розрахуємо об’єм області сканування. Відомо, що перший зріз має координати (-157.4; -180.0; 28.25) і також розташований паралельно до площини xOy. Також відомо кількість зображень в серії (79), відстань між зрізами (1.25) та товщину самих зрізів (5.00). Враховуючи те, що кожен зріз має розмір 512x512 пікселів, можна розрахувати координати останнього пікселя:

x = -157.4 + 512 *1*0.703125 + 512 * 0 *  0.703125 = 202.6 мм

y = -180 + 512*0* 0.703125 + 512 * 1 *  0.703125 = 180 мм

z = 28.25 мм

Оскільки зріз №25 розташований нижче за координатою z від першого зрізу, тому z=28.25 – верхня межа області сканування.

Знайдемо нижню межу:

b = 28.25 – 79 * 1.25 = -70.5 мм

Враховуючи товщину зрізів маємо висоту області сканування:

h = 28.25 + 70.5 + 5 = 103.75 мм

Також можна знайти ширину і глибину вокселя:

width = 512 * 0.703125 = 360 мм

depth = 512 * 0.703125 = 360 мм

Виконавши розрахунки можна зробити висновок, що в заданих даних міститься помилка, оскільки зріз з шуканим пікселем знаходиться поза зоною сканування.

# Розділ 4. Лістинг програми

main.cpp

```
#include <GL/glew.h>
#include <GL/freeglut.h>
#include <vector>
#include <glm/glm.hpp>
#include "Drawable.h"
#include "Point.h"
#include "Shader.h"
#include "Slice.h"
#include "CoordinateSystem.h"
#include "Cube.h"

//Screen size
int windowWidth = 600;
int windowHeight = 600;

int rows = 512, columns = 512;

Coordinate pixelSpacing = {0.703125, 0.703125};
float sliceThickness = 5.0;
float slicesSpacing = 1.25;
Coordinate pixelCoords = {367, 298};

Coordinate imagePositionSlice25 = {-157.400, -180.0, -126.750};
float imageOrientationSlice25[] = {1.0, 0.0, 0.0, 0.0, 1.0, 0.0};

Coordinate imagePositionSlice1 = {-157.400, -180.0, 28.250};
float imageOrientationSlice1[] = {1.0, 0.0, 0.0, 0.0, 1.0, 0.0};

int numOfImageInSeries = 79;

Shader *shader;
std::vector<Drawable *> drawables;

void initDrawableObjects();
void render();
void keyboardInput(unsigned char key, int x, int y);
void updateViewPoint(glm::vec3 cameraPosition);

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

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

    initDrawableObjects();

    glutDisplayFunc(render);
    glutKeyboardFunc(keyboardInput);

    glutMainLoop();

    return 0;

}

void initDrawableObjects() {

    Coordinate pixelPhysicalLocation{
            imagePositionSlice25.x + pixelCoords.y * imageOrientationSlice25[0] * pixelSpacing.y +
pixelCoords.x * imageOrientationSlice25[3] * pixelSpacing.x,
            imagePositionSlice25.y + pixelCoords.y * imageOrientationSlice25[1] * pixelSpacing.y +
pixelCoords.x * imageOrientationSlice25[4] * pixelSpacing.x,
            imagePositionSlice25.z + pixelCoords.y * imageOrientationSlice25[2] * pixelSpacing.y +
pixelCoords.x * imageOrientationSlice25[5] * pixelSpacing.x
    };
    printf("Pixel location(x,y,z) = (%.2f,%.2f,%.2f) ", pixelPhysicalLocation.x,
pixelPhysicalLocation.y, pixelPhysicalLocation.z);

    float scanBottomZ = imagePositionSlice1.z - numOfImageInSeries * slicesSpacing; //Bottom because
slice 1 is higher than slice 25
```

```
    Coordinate slice25EndPoint{
            imagePositionSlice25.x + rows * imageOrientationSlice25[0] * pixelSpacing.y + columns *
imageOrientationSlice25[3] * pixelSpacing.x,
            imagePositionSlice25.y + rows * imageOrientationSlice25[1] * pixelSpacing.y + columns *
imageOrientationSlice25[4] * pixelSpacing.x,
            imagePositionSlice25.z + rows * imageOrientationSlice25[2] * pixelSpacing.y + columns *
imageOrientationSlice25[5] * pixelSpacing.x
    };

    float scanZoneHeight = abs(scanBottomZ - imagePositionSlice1.z) + sliceThickness;
    float scanZoneWidth = abs(slice25EndPoint.x - imagePositionSlice25.x);
    float scanZoneDepth = abs(slice25EndPoint.y - imagePositionSlice25.y);

    Coordinate scanZoneCenter{
            (slice25EndPoint.x + imagePositionSlice25.x) / 2,
            (slice25EndPoint.y + imagePositionSlice25.y) / 2,
            (scanBottomZ + imagePositionSlice1.z) / 2
    };

    shader = new Shader();

    glm::mat4 projection = glm::perspective(glm::radians(40.0f), (float) windowWidth / (float)
windowHeight, 0.1f,4000.0f);
    glm::mat4 model = glm::mat4(1.0f);

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

    updateViewPoint(glm::vec3(1200));

    drawables.push_back(new CoordinateSystem(shader, 500));
    drawables.push_back(new Cube(shader, scanZoneHeight, scanZoneWidth, scanZoneDepth,
scanZoneCenter));
    drawables.push_back(
            new Slice(
                    shader,
                    {imagePositionSlice25.x, imagePositionSlice25.y, imagePositionSlice25.z},
                    {slice25EndPoint.x, imagePositionSlice25.y, imagePositionSlice25.z},
                    {slice25EndPoint.x, slice25EndPoint.y, imagePositionSlice25.z},
                    {imagePositionSlice25.x, slice25EndPoint.y, imagePositionSlice25.z}
            )
    );
    drawables.push_back(new Point(shader, pixelPhysicalLocation, 10));

}

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

    for (auto d : drawables)
        d->draw();

    glutSwapBuffers();
}

void updateViewPoint(glm::vec3 cameraPosition) {
    glm::vec3 cameraTarget(0,0,0);
    glm::vec3 cameraDirection = glm::normalize(cameraPosition - cameraTarget);
    glm::vec3 up(0.0f, 0.0f, 1.0f);
    glm::vec3 cameraRight = glm::normalize(glm::cross(up, cameraDirection));
    glm::vec3 cameraUp = glm::cross(cameraDirection, cameraRight);

    glm::mat4 view = glm::lookAt(
            cameraPosition,
            cameraTarget,
            cameraUp
    );

    shader->setMatrix4("view", view);
}


void keyboardInput(unsigned char key, int x, int y) {
    switch (key) {
        case 'x':
            updateViewPoint(glm::vec3(1500, 0,0));
            break;
```

```
        case 'y':
            updateViewPoint(glm::vec3(0, 1500,0));
            break;
        case 'z':
            updateViewPoint(glm::vec3(1, 1,1000));
            break;
        case 'd':
            updateViewPoint(glm::vec3(1200));
            break;
    }
    glutPostRedisplay();
}
```

Coordinate.h

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

Drawable.h

```
class Drawable {
private:
    Shader* shader;
    GLuint vao;
    GLuint vbo;
    GLenum drawMode;
    int verticesToDraw;
public:
    Drawable(Shader* shader, GLenum drawMode);
    void setVertices(std::vector<float> vertices);
    void draw();
};
```

Drawable.cpp

```
#include "Drawable.h"

Drawable::Drawable(Shader *shader, GLenum drawMode) : shader(shader), drawMode(drawMode) {}

void Drawable::setVertices(std::vector<float> vertices) {

    verticesToDraw = (int) vertices.size() / 7;

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

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

    glUseProgram(shader->getReference());

    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 7 * sizeof(float), (void*)0);
    glEnableVertexAttribArray(0);

    glVertexAttribPointer(1, 4, GL_FLOAT, GL_FALSE, 7 * sizeof(float), (void*)(3 * sizeof(float)));
    glEnableVertexAttribArray(1);

}

void Drawable::draw() {
    glUseProgram(shader->getReference());
    glBindVertexArray(vao);
    glDrawArrays(drawMode, 0, verticesToDraw);
}
```

CoordinateSystem.h

```
#include "Drawable.h"

class CoordinateSystem : public Drawable {
public:
    CoordinateSystem(Shader* shader, float axisLength);
};
```

CoordinateSystem.cpp

```
#include "CoordinateSystem.h"

CoordinateSystem::CoordinateSystem(Shader *shader, float axisLength) : Drawable(shader, GL_LINES) {

    std::vector<float> vertices = {
            //X
```

```
            0,0,0, 1,0,0,1,
            axisLength,0,0, 1,0,0,1,

            0,0,0, 0,1,0,1,
            0,axisLength,0, 0,1,0,1,

            0,0,0, 0,0,1,1,
            0,0,axisLength, 0,0,1,1
    };

    Drawable::setVertices(vertices);

}
```

Point.h

```
class Point : public Drawable {
public:
    Point(Shader* shader, Coordinate center, float size);
};
```

Point.cpp

```
#include "Point.h"

Point::Point(Shader* shader, Coordinate center, float size) : Drawable(shader, GL_POINTS) {

    std::vector<float> vertexData = {
            //Coordinates                   //Colors
            center.x, center.y, center.z,     0.0f, 0.0f, 0.0f, 1.0f
    };

    Drawable::setVertices(vertexData);

    glEnable(GL_PROGRAM_POINT_SIZE);
    glPointSize(size);

}
```

Slice.h

```
#include <GL/glew.h>
#include "Shader.h"
#include "Coordinate.h"
#include "Drawable.h"

class Slice : public Drawable {
public:
    Slice(Shader* shader, Coordinate a, Coordinate b, Coordinate c, Coordinate d);
};
```

Slice.cpp

```
#include "Slice.h"
Slice::Slice(Shader* shader, Coordinate a, Coordinate b, Coordinate c, Coordinate d) :
Drawable(shader, GL_TRIANGLE_FAN) {

    std::vector<float> vertexData = {
            //Coordinates           //Colors
            a.x, a.y, a.z,          0.0f, 0.0f, 0.0f, 0.5f,
            b.x, b.y, b.z,          0.0f, 0.0f, 0.0f, 0.5f,
            c.x, c.y, c.z,          0.0f, 0.0f, 0.0f, 0.5f,
            d.x, d.y, d.z,          0.0f, 0.0f, 0.0f, 0.5f
    };

    Drawable::setVertices(vertexData);

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

}
```

Cube.h

```
#include <GL/glew.h>
#include "Shader.h"
#include "Coordinate.h"
#include "Drawable.h"

class Cube : public Drawable {
public:
    Cube(Shader* shader, float height, float width, float depth, Coordinate center);
};
```

Cube.cpp

```
#include "Cube.h"
Cube::Cube(Shader* shader, float height, float width, float depth, Coordinate center) :
Drawable(shader, GL_LINE_STRIP) {

    std::vector<float> vertexData = {
            //Coordinates                                                                   //Colors
            center.x - width / 2, center.y + depth / 2, center.z + height / 2,      0.0f, 0.0f, 0.0f,
1.0f,
            center.x + width / 2, center.y + depth / 2, center.z + height / 2,      0.0f, 0.0f, 0.0f,
1.0f,
            center.x + width / 2, center.y - depth / 2, center.z + height / 2,      0.0f, 0.0f, 0.0f,
1.0f,
            center.x - width / 2, center.y - depth / 2, center.z + height / 2,      0.0f, 0.0f, 0.0f,
1.0f,

            center.x - width / 2, center.y + depth / 2, center.z + height / 2,      0.0f, 0.0f, 0.0f,
1.0f,
            center.x - width / 2, center.y + depth / 2, center.z - height / 2,      0.0f, 0.0f, 0.0f,
1.0f,
            center.x - width / 2, center.y - depth / 2, center.z - height / 2,      0.0f, 0.0f, 0.0f,
1.0f,
            center.x - width / 2, center.y - depth / 2, center.z + height / 2,      0.0f, 0.0f, 0.0f,
1.0f,

            center.x - width / 2, center.y - depth / 2, center.z - height / 2,      0.0f, 0.0f, 0.0f,
1.0f,
            center.x + width / 2, center.y - depth / 2, center.z - height / 2,      0.0f, 0.0f, 0.0f,
1.0f,
            center.x + width / 2, center.y - depth / 2, center.z + height / 2,      0.0f, 0.0f, 0.0f,
1.0f,

            center.x + width / 2, center.y - depth / 2, center.z - height / 2,      0.0f, 0.0f, 0.0f,
1.0f,
            center.x + width / 2, center.y + depth / 2, center.z - height / 2,      0.0f, 0.0f, 0.0f,
1.0f,
            center.x + width / 2, center.y + depth / 2, center.z + height / 2,      0.0f, 0.0f, 0.0f,
1.0f,

            center.x + width / 2, center.y + depth / 2, center.z - height / 2,      0.0f, 0.0f, 0.0f,
1.0f,
            center.x - width / 2, center.y + depth / 2, center.z - height / 2,      0.0f, 0.0f, 0.0f,
1.0f
    };

    Drawable::setVertices(vertexData);

}
```

Shader.h

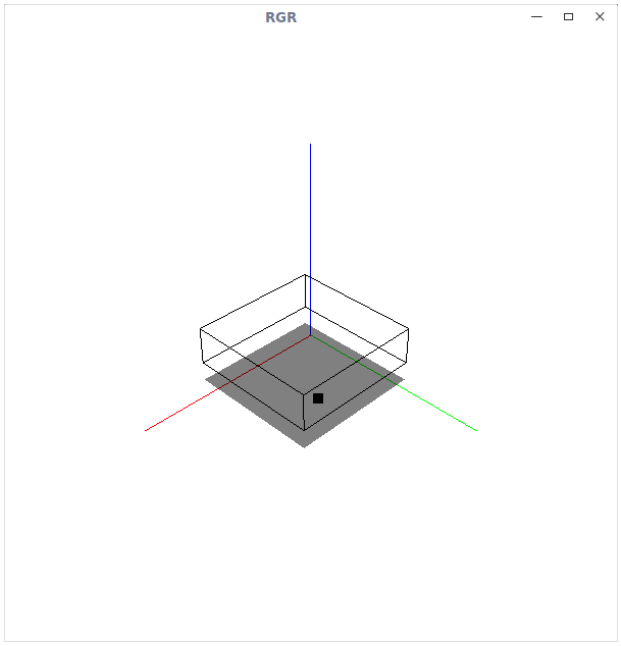
```
#include <GL/glew.h>
#include <string>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>

class Shader {

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

public:
    Shader();
    void setMatrix4(const std::string &name, glm::mat4 matrix);
    void setVec3(const std::string &name, glm::vec3 vector);
    GLuint getReference() { return shaderProgram; }

};
```

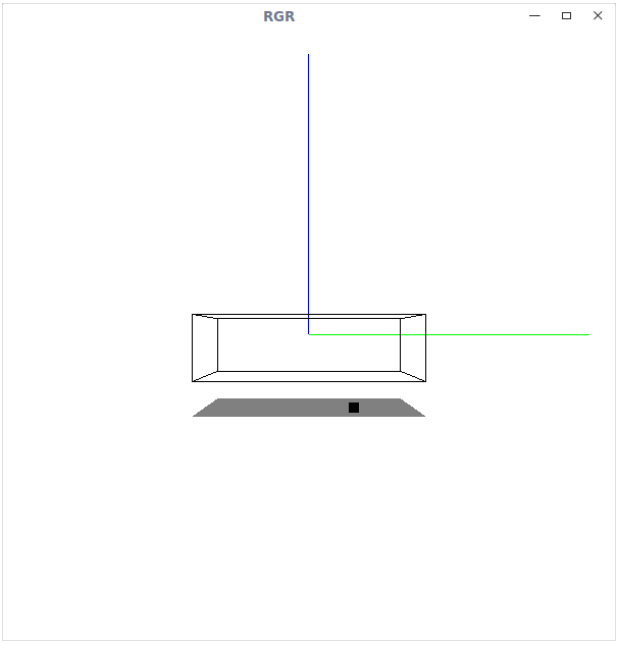
Shader.cpp

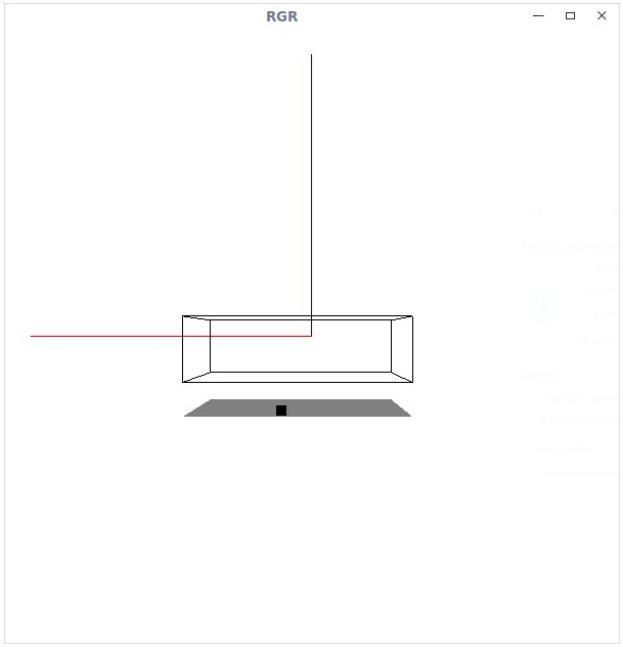
```
#include "Shader.h"
#include <iostream>

Shader::Shader() {

    auto vertexShaderScript =
            "#version 400\n"
            "uniform mat4 projection;"
            "uniform mat4 model;"
```

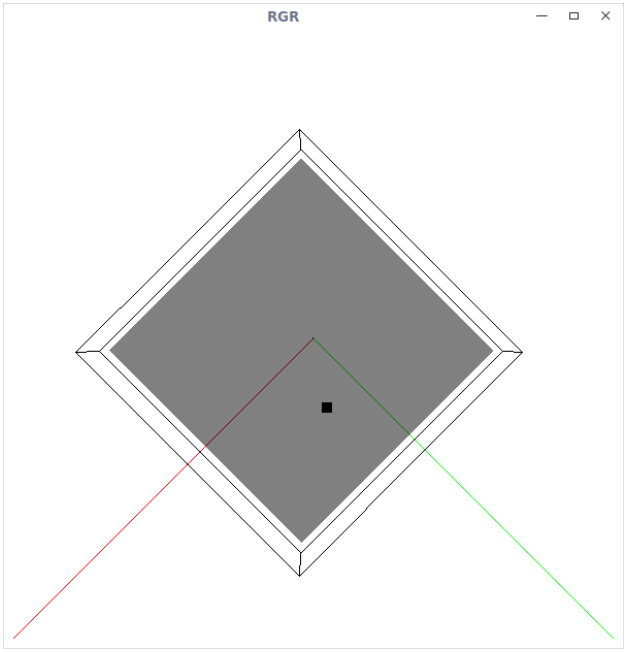
```
            "uniform mat4 view;"
            "layout (location = 0) in vec3 position;"
            "layout (location = 1) in vec4 color_in;"
            "out vec4 fragColor;"
            "void main()"
            "{"
            "    gl_Position = projection * view * model * vec4(position, 1.0);"
            "    fragColor = color_in;"
            "}";

    auto fragmentShaderScript =
            "#version 400\n"
            "in vec4 fragColor;"
            "out vec4 FragColor;"
            "void main()"
            "{"
            "    FragColor = vec4(fragColor);"
            "}";

    GLuint vertexShader = compileShader(vertexShaderScript, GL_VERTEX_SHADER);
    GLuint fragmentShader = compileShader(fragmentShaderScript, 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]);
}
```

## Розділ 4. Результати роботи програмного застосунку




Вивід в консоль:
Pixel location(x,y,z) = (52.13,78.05,-126.75)

# Висновки

В результаті виконання розрахунково-графічної роботи було проведено розрахунки фізичного розташування області сканування, шуканого зрізу та координати пікселя. Також створено програмний застосунок для проведення автоматичних розрахунків та візуалізації.

Під час проведення роботи вдалося поглибити теоретичні знання для роботи з медичними зображення, а також отримати додатковий досвід з візуалізації трьохвимірного простору та об’єктів різного типу.

## Список використаної літератури


1. Learn OpenGL [Електронний ресурс]. – Режим доступу: https://learnopengl.com/ (дата звернення 29.12.2019)
2. DCMTK Documentation [Електронний ресурс]. – Режим доступу: https://support.dcmtk.org/docs/ (дата звернення 29.12.2019)
3. Обробка медичних зображень. Робота з даними та алгоритми для медичної візуалізації : метод. вказівки до викон. комп. практикумів для студ. Спец. 122 “Комп’ютерні науки” спеціалізації “Інформаційні технології в біології та медицині” / Уклад. С. М. Алхімова. – Київ: КПІ ім. Ігоря Сікорського, Вид-во “Політехніка”, 2018. – 40 с.