Московский авиационный институт

(Национальный исследовательский университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Компьютерная графика»

Лабораторная работа № 4-6

Тема: Ознакомление с технологией OpenGL. Создание шейдерных анимационных эффектов в OpenGL 2.1

Студентка: Довженко А.А.
Группа: 80-307
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2018

1. Постановка задачи

Создать графическое приложение с использованием OpenGL. Используя результаты Л.Р.No3, изобразить заданное тело (то же, что и в л.р. No3) с использованием средств OpenGL 2.1. Использовать буфер вершин. Точность аппроксимации тела задается пользователем. Обеспечить возможность вращения и масштабирования многогранника и удаление невидимых линий и поверхностей. Реализовать простую модель освещения на GLSL.

Параметры освещения и отражающие свойства материала задаются пользователем в диалоговом режиме.

Для поверхности, созданной в л.р. No5, обеспечить выполнение шейдерного эффекта.

Вариант 7.

Одна из полостей двуполостного гиперболоида. Анимация. Координата X изменяется по закону X=sin(t) для всех вершин, компонента X нормали которых >0.

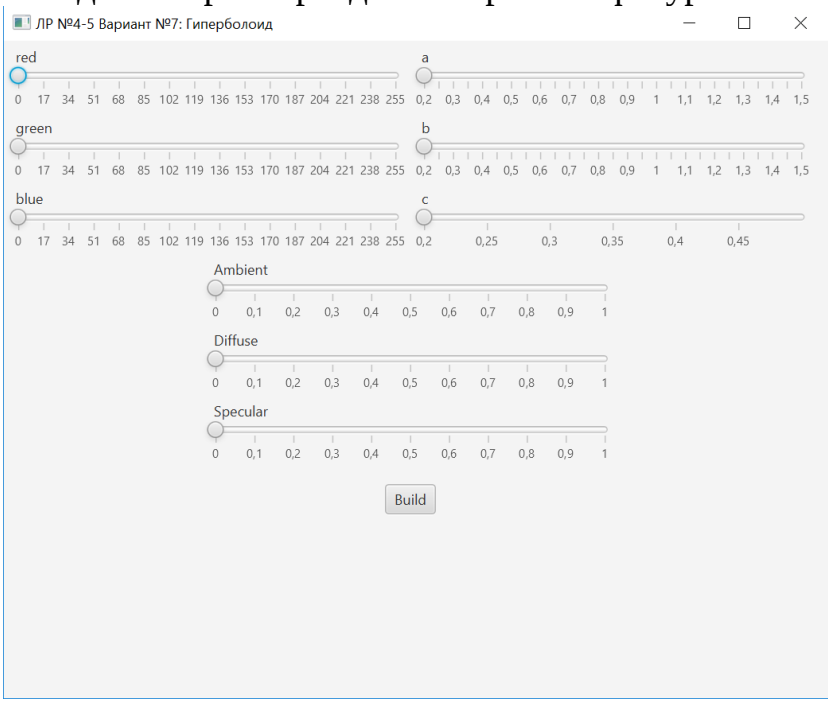
2. Решения задачи

Тело строится по аналогии с предыдущей лабораторной работой, только используя средства OpenGL. Затем необходимо реализовать класс, предоставляющий интерфейс шейдера, реализовать вершинный и фрагментный шейдеры, загрузив их в вышеуказанный класс. Для ввода данных используется окно JavaFX, после выбора входных данных открывается окно OpenGL, в котором происходит отрисовка заданной фигуры. В работе использована библиотека LWJGL. Интерфейс: поворот выполняется перетаскиванием курсора по холсту. Все необходимые параметры устанавливаются в диалоговом окне JavaFX перед построением при помощи слайдеров.

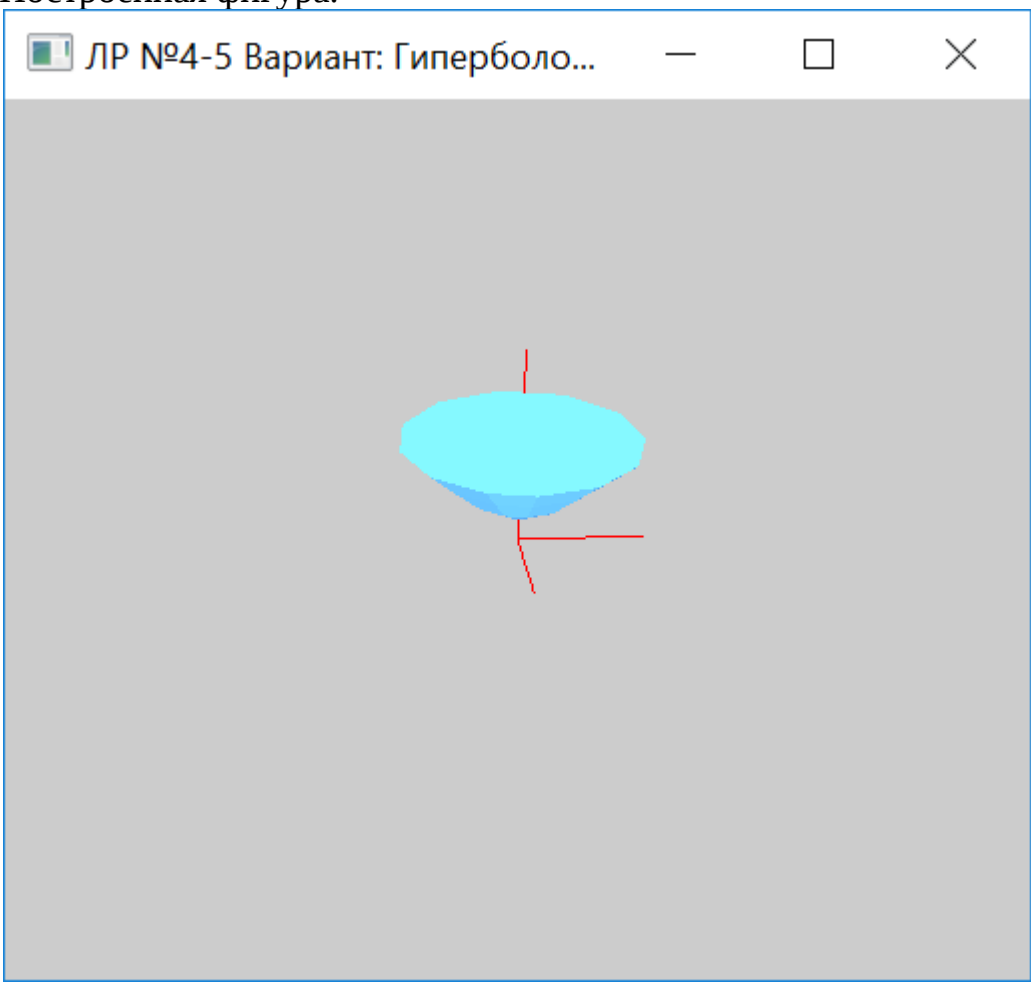
3. Руководство по использованию программы

Запустить IntelliJ, открыть в нём проект с программой и запустить его.

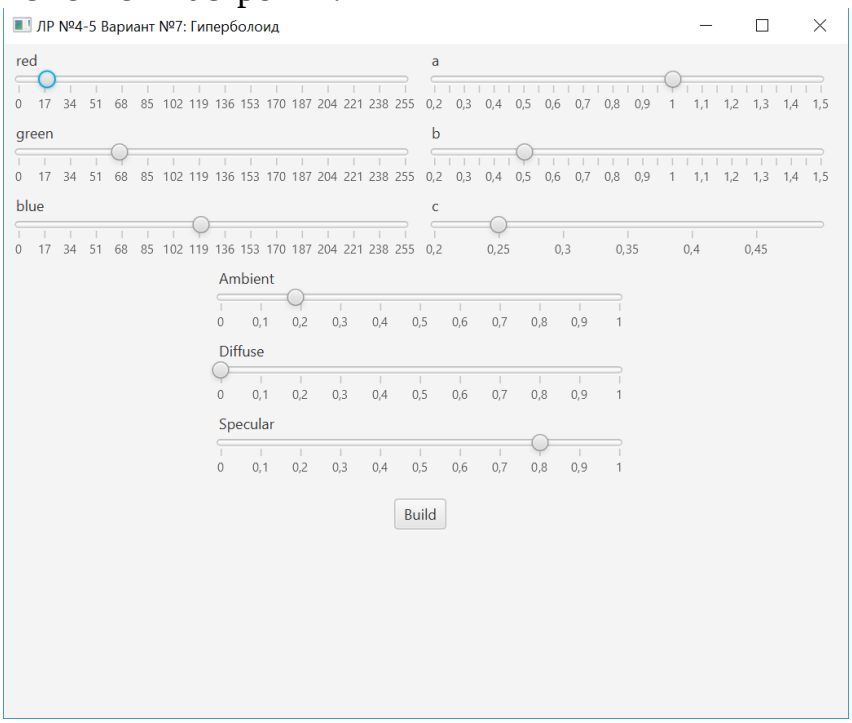
Исходные параметры для построения фигуры.



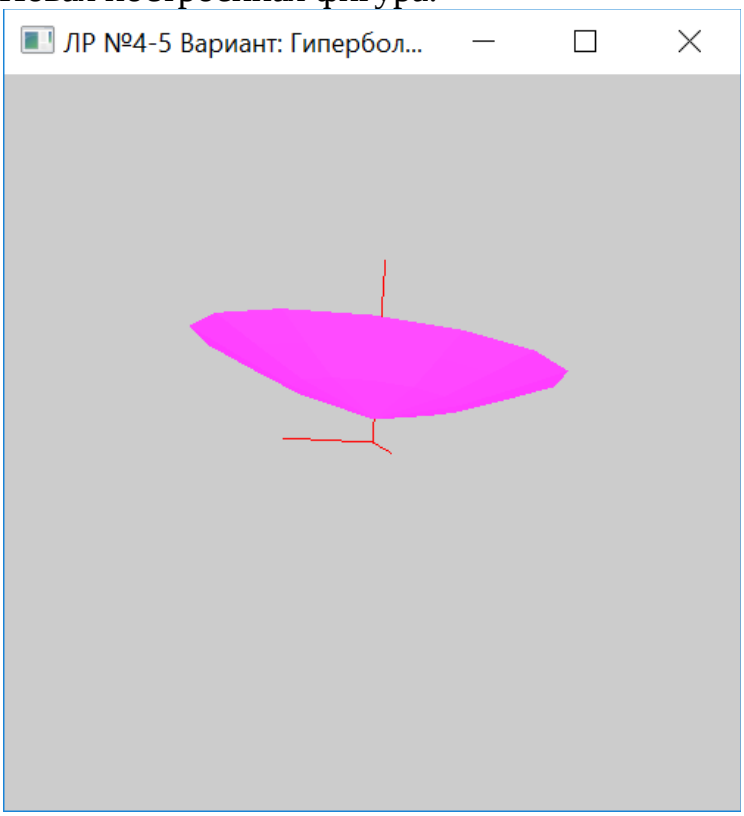
Построенная фигура.



Изменяем настройки.



Новая построенная фигура.



4. Листинг программы

```
//lab4.java
// Довженко А.А. М8О-307Б
// Лабораторная работа 4-5
// Вариант№7: Одна из полостей двуполостного гиперболоида.

package main;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;
import org.joml.Matrix4f;
import org.lwjgl.*;
import org.lwjgl.glfw.*;
import org.lwjgl.opengl.*;
import org.lwjgl.system.*;
import shader.Shader;

import java.io.File;
import java.io.FileInputStream;
import java.net.URL;
import java.nio.*;

import static org.lwjgl.glfw.Callbacks.*;
import static org.lwjgl.glfw.GLFW.*;
import static org.lwjgl.opengl.GL11.*;
import static org.lwjgl.system.MemoryStack.*;
import static org.lwjgl.system.MemoryUtil.*;

public class Lab4 extends Application{

    // The window handle
    private long window;
    private Matrix4f matrix;
    private double curX, curY;

    public static float a = 1;
    public static float b = 1;
    public static float c = 0.5f;

    public static float red = 0;
    public static float green = 0.2f;
```

```
    public static float blue = 0.8f;

    public static float ambient = 1f;
    public static float diffuse = 1f;
    public static float specular = 1f;

    private float step = 0.25f;
    private Figure figure;

    public Lab4() {
        matrix = new Matrix4f().identity();
        curX = 0;
        curY = 0;
    }

    public void run() {
        System.out.println("Hello LWJGL " + Version.getVersion() + "!");

        initGLFW();
        loop();

        // Free the window callbacks and destroy the window
        glfwFreeCallbacks(window);
        glfwDestroyWindow(window);

        // Terminate GLFW and free the error callback
        glfwTerminate();
        glfwSetErrorCallback(null).free();
    }

    private void initGLFW() {
        //инициализиция всего необходимого перед началом работы

        GLFWErrorCallback.createPrint(System.err).set();
        if (!glfwInit()) {
            throw new IllegalStateException("Unable to initialize GLFW");
        }

            glfwDefaultWindowHints(); // optional, the current window hints are already the default
           glfwWindowHint(GLFW_VISIBLE, GLFW_TRUE); // the window will stay hidden after creation
            glfwWindowHint(GLFW_RESIZABLE, GLFW_TRUE); // the window
```

will be resizable

```
            window = glfwCreateWindow(500, 500, "ЛР №4-5 Вариант: Гиперболоид", NULL, NULL);
        if (window == NULL) {
            throw new RuntimeException("Failed to create the GLFW window");
        }

        glfwSetKeyCallback(window, (window, key, scancode, action, mods) -> {
            if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS) {
                glfwSetWindowShouldClose(window, true);
            }
            if (key == GLFW_KEY_UP && action == GLFW_PRESS) {
                matrix.rotateX((float) Math.toRadians(-10));
            }
            if (key == GLFW_KEY_DOWN && action == GLFW_PRESS) {
                matrix.rotateX((float) Math.toRadians(10));
            }
            if (key == GLFW_KEY_LEFT && action == GLFW_PRESS) {
                matrix.rotateY((float) Math.toRadians(-10));
            }
            if (key == GLFW_KEY_RIGHT && action == GLFW_PRESS) {
                matrix.rotateY((float) Math.toRadians(10));
            }
        });

        glfwSetFramebufferSizeCallback(window, (window, w, h) -> {
            glViewport(0, 0, w, h);
            glOrtho(0, w, h, 0, -1, 1);
        });

        try (MemoryStack stack = stackPush()) {
            IntBuffer pWidth = stack.mallocInt(1); // int*
            IntBuffer pHeight = stack.mallocInt(1); // int*

            // Get the window size passed to glfwCreateWindow
            glfwGetWindowSize(window, pWidth, pHeight);

            // Get the resolution of the primary monitor
                                                    GLFWVidMode vidmode = glfwGetVideoMode(glfwGetPrimaryMonitor());

            // Center the window
```

```
            glfwSetWindowPos(
                    window,
                    (vidmode.width() - pWidth.get(0)) / 2,
                    (vidmode.height() - pHeight.get(0)) / 2
            );
        }

        glfwMakeContextCurrent(window);
        glfwSwapInterval(0);
        glfwShowWindow(window);
    }

    private void loop() {
        //цикл, в котором происходит отрисовка

        GL.createCapabilities();
        System.out.println(glGetString(GL_VERSION));

        glEnable(GL_TEXTURE_2D);
        glClearColor(0.8f, 0.8f, 0.8f, 0.0f);

        Shape x = new Shape(new float[]{0, 0, 0, 1, 0, 0});
        Shape y = new Shape(new float[]{0, 0, 0, 0, 1, 0});
        Shape z = new Shape(new float[]{0, 0, 0, 0, 0, 2});

        FloatBuffer floatBuffer = BufferUtils.createFloatBuffer(16);
        matrix.scale(0.25f);

        Shader shader = new Shader();
        shader.set("ambientK", ambient);
        shader.set("diffuseK", diffuse);
        shader.set("specularK", specular);

        figure = new Figure(a,b,c);
        figure.countFigure(step);
        double time = 0;

        while (!glfwWindowShouldClose(window)) {
            glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

            glLoadMatrixf(matrix.get(floatBuffer));

            shader.set("matrix", matrix);
```

```
            if(glfwGetMouseButton(window,GLFW_MOUSE_BUTTON_1) ==
GLFW_PRESS) {
            DoubleBuffer xBegin = BufferUtils.createDoubleBuffer(1);
            DoubleBuffer yBegin = BufferUtils.createDoubleBuffer(1);
            glfwGetCursorPos(window, xBegin, yBegin);

            double xB = xBegin.get(0);
            double yB = yBegin.get(0);

            int dx = (int) (xB - curX);
            int dy = (int) (yB - curY);

            if (dx > 30 || dx < -30) {
               dx = 0;
            }

            if (dy > 30 || dy < -30) {
               dy = 0;
            }
            matrix.rotateX((float) Math.toRadians(dx));
            matrix.rotateY((float) Math.toRadians(dy));
            curX = xB;
            curY = yB;

         }

         shader.bindShader();

         shader.set("axis",1);
         shader.setColor(1.0f, 0.0f, 0.0f);
         x.draw(GL_LINE_STRIP);
         shader.setColor(1.0f, 0.0f, 0.0f);
         y.draw(GL_LINE_STRIP);
         shader.setColor(1.0f, 0.0f, 0.0f);
         z.draw(GL_LINE_STRIP);
         shader.set("axis",0);

                                    //shader.setColor((float)Math.abs(Math.sin(time)),
(float)Math.abs(Math.sin(time * 2)), (float)Math.abs(Math.sin(time / 3)));
                                     shader.setColor((float)Math.abs(Math.sin(time)),
(float)Math.abs(Math.sin(time * 2)), (float)Math.abs(Math.sin(time / 3)));
         figure.drawFigure(matrix, shader);
```

```
            glfwSwapBuffers(window);
            glfwPollEvents();

            try {
                Thread.sleep(35);
            } catch (InterruptedException e) {
                //e.printStackTrace();
            }
            time += 0.05;
            if(time >= 360)  {
                time -= 360;
            }
        }
    }


    @Override
    public void start(Stage stage) throws Exception {
                                                            final   Parent   root   =
FXMLLoader.load(getClass().getResource("/fxWindow/window.fxml"));
        final Scene scene = new Scene(root);

        stage.setTitle("ЛР №4-5 Вариант №7: Гиперболоид");
        stage.setScene(scene);
        stage.show();
        stage.setMinHeight(scene.getWindow().getHeight());
        stage.setMinWidth(scene.getWindow().getWidth());
    }


    public static void main(String[] args) {
        //при запуска отрывается окно JavaFX, в котором задаются параметры,
а после его закрытия открывается окно с фигурой
        Lab4 lab4 = new Lab4();
        launch(args);
        lab4.run();
    }
}

//                                                                     figure.java
//Довженко А.А. М8О-307Б
//Класс, выполняющий рассчет необходимых точек гиперболоида и его
отрисовку
```

```
package main;

import org.joml.Matrix3f;
import org.joml.Matrix4f;
import org.joml.Vector3f;
import shader.Shader;

import java.util.Vector;

import static org.lwjgl.opengl.GL11.*;

public class Figure {

    private Vector<Vector<Vector3f>> figure;

    private float a;
    private float b;
    private float c;

    public Figure(float a, float b, float c) {
        figure = new Vector<>();
        this.a = a;
        this.b = b;
        this.c = c;
    }

    private Vector<Vector3f> countCircle(float z, float step) {
        Vector<Vector3f> res = new Vector<>();
        //double r = Math.sqrt(1 - z * z);
        //double r = 0.5 * z;

        double step1 = 0;
        while (step1 < Math.PI * 2) {
                res.add(new Vector3f((float) (a * Math.sqrt(((z*z) / (c*c)) - 1) *
Math.cos(step1)), (float) (b * Math.sqrt((z*z) / (c*c) - 1) * Math.sin(step1)), z));
            step1 += 2 * step;
        }
        step1 = 0;
                res.add(new Vector3f((float) (a * Math.sqrt((z*z) / (c*c) - 1) *
Math.cos(step1)), (float) ((b * Math.sqrt((z*z) / (c*c) - 1) *  Math.sin(step1))),
z));
        return res;
    }
```

```
    public void countFigure(float step) {
        float counter = c;
        float step1 = step;

        float begin = c;

        int n = Math.round( (1f - begin) / step);
        step1 = (1f - begin) / (float)n;
        for (int i = 0; i < n; i++) {
            figure.add(countCircle(counter,step1));
            counter += step1;
        }
        figure.add(countCircle(counter,step1));
        //doTriangles();
    }

    public void drawFigure(Matrix4f matrix, Shader shader) {
        for (int i = 0; i < figure.size() - 1; i++) {
            Vector<Vector3f> vec1 = figure.get(i);
            Vector<Vector3f> vec2 = figure.get(i + 1);
            for (int j = 0; j < vec1.size() - 1; j++) {
                                //drawTriangle(vec1.get(j),  vec1.get(j  +  1),  vec2.get(j),
Color.BLACK);
                drawTriangle(new Shape(new float[]{
                        vec2.get(j + 1).x, vec2.get(j + 1).y, vec2.get(j + 1).z,
                        vec1.get(j + 1).x, vec1.get(j + 1).y, vec1.get(j + 1).z,
                        vec2.get(j).x, vec2.get(j).y, vec2.get(j).z
                }), matrix, shader);

                drawTriangle(new Shape(new float[]{
                        vec1.get(j).x, vec1.get(j).y, vec1.get(j).z,
                        vec1.get(j + 1).x, vec1.get(j + 1).y, vec1.get(j + 1).z,
                        vec2.get(j).x, vec2.get(j).y, vec2.get(j).z
                }), matrix, shader);
            }
        }

        Vector<Vector3f> v = figure.get(figure.size() - 1);
        for (int i = 0; i < v.size() - 1; i++) {

            drawTriangle( new Shape( new float[]{
```

```
                v.get(i).x, v.get(i).y, v.get(i).z,
                v.get(i + 1).x, v.get(i + 1).y, v.get(i + 1).z,
                0,0,1
            }), matrix, shader);
        }
    }


    private boolean drawTriangle(Shape kek, Matrix4f matrix, Shader shader) {
        float k = triangleVisible(kek.getVerts(), matrix, shader);
        if(k > 0) {
                //shader.setColor(Math.min(0.5f * k, 1f) , Math.min(0.5f * k, 1f),
Math.min(0.5f * k, 1f));
            kek.draw(GL_POLYGON);
            return true;
        }
        return false;
    }


     private float triangleVisible(float[] vertices, Matrix4f matrix, Shader shader)
{
        float d;
        Matrix3f m = new Matrix3f();
        m = matrix.get3x3(m);

        Vector3f p1 = new Vector3f(vertices[0], vertices[1], vertices[2]);
        Vector3f p2 = new Vector3f(vertices[3], vertices[4], vertices[5]);
        Vector3f p3 = new Vector3f(vertices[6], vertices[7], vertices[8]);
        Vector3f inner = new Vector3f(0, 0, c + 0.01f);
        m.transform(p1);
        m.transform(p2);
        m.transform(p3);
        m.transform(inner);

        Vector3f v1 = p2.sub(p1);
        Vector3f v2 = p3.sub(p1);
        Vector3f n = v1.cross(v2);//crossProduct(sb1, sb2);
        Vector3f tmp = inner.sub(p1);

        if (n.dot(tmp) > 0) {
            n.negate();
        }
```

```
        Vector3f view = new Vector3f(0, 0, 50);
        if (n.dot(view) > 0) {
            d = Math.abs(n.dot(view) / (view.length() * n.length()));
            shader.set("normal",n);
        } else {
            d = -1;
        }

        return d;
    }

    public void setA(float a) {
        this.a = a;
    }

    public void setB(float b) {
        this.b = b;
    }

    public void setC(float c) {
        this.c = c;
    }

}

// //Довженко А.А. М8О-307Б
//Класс-контроллер для JavaFX окна
package main;

import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.Slider;

public class Controller {
    @FXML
    private Slider aSlider;

    @FXML
    private Slider bSlider;

    @FXML
    private Slider cSlider;
```

```
    @FXML
    private Slider redSlider;

    @FXML
    private Slider greenSlider;

    @FXML
    private Slider blueSlider;

    @FXML
    private Slider ambSlider;

    @FXML
    private Slider diffSlider;

    @FXML
    private Slider specSlider;

    @FXML
    private Button button;

    @FXML
    private void initialize() {
        button.setOnAction(event -> {
            Lab4.red = (float)redSlider.getValue();
            Lab4.green = (float)greenSlider.getValue();
            Lab4.blue = (float)blueSlider.getValue();

            Lab4.a = (float)aSlider.getValue();
            Lab4.b = (float)bSlider.getValue();
            Lab4.c = (float)cSlider.getValue();

            Lab4.ambient = (float)ambSlider.getValue();
            Lab4.diffuse = (float)diffSlider.getValue();
            Lab4.specular = (float)specSlider.getValue();
        });
    }
}

//Довженко А.А. М8О-307Б
//Класс для отрисовки контуров

package main;
```

```
import org.joml.Vector3f;
import org.lwjgl.BufferUtils;

import java.nio.FloatBuffer;

import static org.lwjgl.opengl.GL11.*;
import static org.lwjgl.opengl.GL15.*;

public class Shape {
    private int vertexArrayPtr;
    private int n;

    public float[] getVerts() {
        return verts;
    }
    private float[] verts;

    public Shape(float[] vertexArray) {
        if(vertexArray.length > 7) {
            for (int i = 0; i < 3; i++) {
                    if (vertexArray[3 * i] == 0 && vertexArray[3 * i + 1] == 0 &&
vertexArray[3 * i + 2] == 0) {
                        System.out.println(vertexArray[0] + " " + vertexArray[1] + " " +
vertexArray[2]);
                        System.out.println(vertexArray[3] + " " + vertexArray[4] + " " +
vertexArray[5]);
                        System.out.println(vertexArray[6] + " " + vertexArray[7] + " " +
vertexArray[8]);
                    System.out.println(" ");
                }
            }
        }

        verts = vertexArray;
        n = vertexArray.length;
        FloatBuffer floatBuffer = BufferUtils.createFloatBuffer(n);
        n /= 3;

        floatBuffer.put(vertexArray);
        floatBuffer.flip();

        vertexArrayPtr = glGenBuffers();
```

```
        glBindBuffer(GL_ARRAY_BUFFER, vertexArrayPtr);
        glBufferData(GL_ARRAY_BUFFER, floatBuffer, GL_STATIC_DRAW);
        glBindBuffer(GL_ARRAY_BUFFER, 0);
    }

    public void draw(int mode) {

        glEnableClientState(GL_VERTEX_ARRAY);
        glBindBuffer(GL_ARRAY_BUFFER, vertexArrayPtr);

        glVertexPointer(3, GL_FLOAT, 0, 0);
        glDrawArrays(mode, 0, n);

        glBindBuffer(GL_ARRAY_BUFFER, 0);
        glDisableClientState(GL_VERTEX_ARRAY);
    }
}

//                                                                                shader.java
//Довженко А.А. М8О-307Б
//Класс, предоставляющий интерфейс взаимодействия с шейдером

package shader;

import org.joml.Matrix4f;
import org.joml.Vector3f;
import org.lwjgl.BufferUtils;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.nio.FloatBuffer;


import static org.lwjgl.opengl.GL20.*;

public class Shader {
    private final String FOLDER = "shader/";
    private final String VERTEX_SHADER_FILE = "shader.vs";
    private final String FRAGMENT_SHADER_FILE = "shader.fs";

    private int program;
```

```
    private int vertexShader;
    private int fragmentShader;

    public Shader() {
        program = glCreateProgram();
        vertexShader = glCreateShader(GL_VERTEX_SHADER);
                            glShaderSource(vertexShader,    getSource(FOLDER    +
VERTEX_SHADER_FILE));
        glCompileShader(vertexShader);
        if (glGetShaderi(vertexShader, GL_COMPILE_STATUS) != 1) {
            System.err.println(glGetShaderInfoLog(vertexShader));
            System.exit(1);
        }

        fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
                          glShaderSource(fragmentShader,    getSource(FOLDER    +
FRAGMENT_SHADER_FILE));
        glCompileShader(fragmentShader);
        if (glGetShaderi(fragmentShader, GL_COMPILE_STATUS) != 1) {
            System.err.println(glGetShaderInfoLog(fragmentShader));
            System.exit(2);
        }

        glAttachShader(program, vertexShader);
        glAttachShader(program, fragmentShader);

        glBindAttribLocation(program, 0, "vertex");

        glLinkProgram(program);
        if (glGetProgrami(program, GL_LINK_STATUS) != 1) {
            System.err.println(glGetProgramInfoLog(program));
            System.exit(3);
        }

        glValidateProgram(program);
        if (glGetProgrami(program, GL_VALIDATE_STATUS) != 1) {
            System.err.println(glGetProgramInfoLog(program));
            System.exit(4);
        }
    }


    public void setColor(float red, float green, float blue) {
```

```
    set("red", red);
    set("green", green);
    set("blue", blue);
}

public void set(String name, Vector3f vector) {
    int pointer = glGetUniformLocation(program, name);
    if (pointer != -1) {
        glUniform3f(pointer, vector.x, vector.y, vector.z);
    }
}

public void set(String name, int value) {
    int pointer = glGetUniformLocation(program, name);
    if(pointer != -1) {
        glUniform1i(pointer,value);
    }
}

public void set(String name, float value) {
    int pointer = glGetUniformLocation(program,name);
    if( pointer != -1) {
        glUniform1f(pointer, value);
    }
}

public void set(String name, Matrix4f matrix) {
    int pointer = glGetUniformLocation(program,name);
    if(pointer != -1) {
        FloatBuffer buffer = BufferUtils.createFloatBuffer(16);
        matrix.get(buffer);
        glUniformMatrix4fv(pointer, false, buffer);
    }
}

public void bindShader() {
    glUseProgram(program);
}

private String getSource(String filename) {
    //получение кода шейдера из файла
    StringBuilder res = new StringBuilder();
    BufferedReader reader;
```

```
        try {
            reader = new BufferedReader(new FileReader(new File(filename)));

            String str;
            while((str = reader.readLine()) != null) {
                res.append(str);
                res.append("\n");
            }
            reader.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        //System.out.println(res.toString());
        return res.toString();
    }
}
```

// shader.fs

```
//Довженко А.А. М8О-307Б
//фрагментный шейдер, отвечающий за рассчет цвета

#version 120

uniform int axis;

uniform float red = 0;
uniform float green = 0;
uniform float blue = 0;

uniform vec3 light = vec3(0,0,1);
uniform vec3 normal;

uniform float ambientK = 1;
uniform float diffuseK = 1;
uniform float specularK = 1;

vec4 countColor(){
    vec3 col = vec3(red,green,blue);
    vec3 ambient = ambientK * col;

    vec3 n = normalize(normal);
    float diff = max(dot(n,light),0);
    vec3 diffuse = diffuseK * diff * col;
```

```
    float spec = pow(dot(n,light),32);
    vec3 specular = specularK * spec * vec3(1,1,1);

    return vec4(ambient + diffuse + specular,1);
}

void main() {
    if(axis == 1) {
        gl_FragColor = vec4(red,green,blue,1);
    } else {
        gl_FragColor = countColor();
    }
}
```

shader.vs

```
//Довженко А.А. М8О-307Б
//вершинный шейдер, отвечающий за расположение точек

#version 120

attribute vec3 vertex;
uniform vec3 normal;

uniform int axis;

uniform mat4 matrix;

void main() {
    gl_Position = matrix * vec4(vertex, 1);
}
```

5. Используемые источники

1. Документация библиотеки JavaFX [Электронный ресурс]. URL: https://docs.oracle.com/javafx/2/ (дата обращения: 21.12.2018).
2. Статья по OpenGL [Электронный ресурс]. URL: https://habr.com/post/111175/ (дата обращения: 20.12.2018).