МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №6

по дисциплине «Компьютерная графика»

Тема: «Реализация трёхмерного объекта с использованием библиотеки OpenGL»

Студент гр. 7381 _____ Минуллин М.А.

Студентка гр. 7381 _____ Машина Ю.Д.

Преподаватель _____ Герасимова Т.В.

Санкт-Петербург

2019

Цель работы.

Разработать программу, реализующую представление разработанного вами трехмерного рисунка, используя предложенные функции библиотеки OpenGL (матрицы видового преобразования, проецирование) и язык GLSL.

Разработанная программа должна быть пополнена возможностями остановки интерактивно различных атрибутов через вызов соответствующих элементов интерфейса пользователя, замена типа проекции, управление преобразованиями, как с помощью мыши, так и с помощью диалоговых элементов.

Индивидуализация.

Вариант 51. Написать программу, рисующую проекцию трехмерного каркасного объекта, представлен на рис. 1.

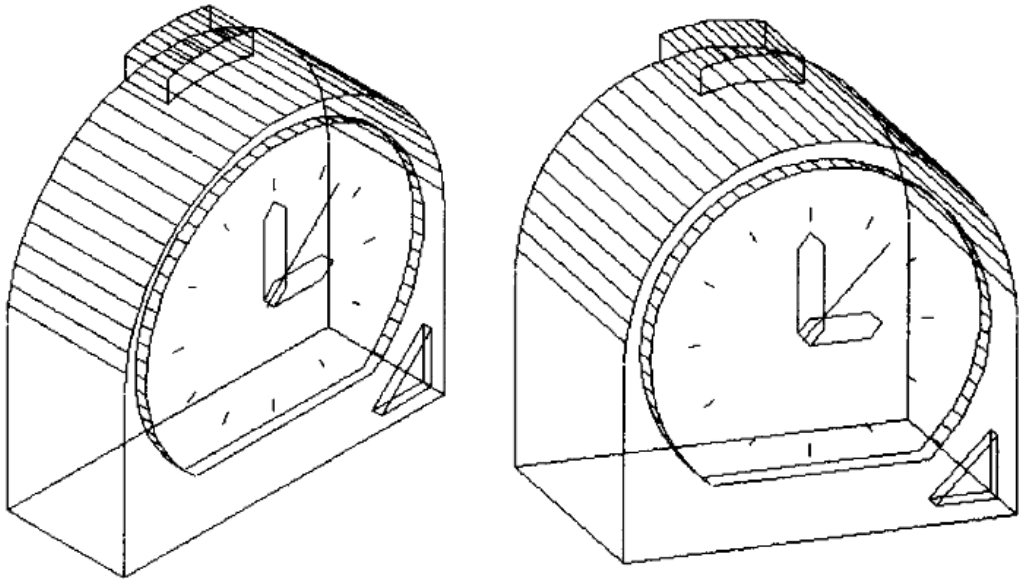


Рисунок 1.

Требования.

1 Грани объекта рисуются с помощью доступных функций рисования отрезка в координатах окна. При этом использовать шейдеры GLSL и OpenGL

2 Ортогональное и перспективное проецирование;

3 Перемещения, повороты и масштабирование многогранника по каждой из осей независимо от остальных.

4 Генерация многогранника с заданной мелкостью разбиения.

5 При запуске программы объект сразу должно быть хорошо виден.

6 Пользователь имеет возможность вращать фигуру (2 степени свободы) и изменять параметры фигуры.

7 Возможно изменять положение наблюдателя.

8 Нарисовать оси системы координат.

9 Все варианты требований могут быть выбраны интерактивно.

Ход работы.

В ходе выполнения работы был создан новый проект. Написан пользовательский графический интерфейс (приложение Б). Задание выполнено в соответствии с большей частью требований. Пример изображения осей координат был взят из книги «Программирование трехмерной графики. Тихомиров». Оттуда же получена информация по рисованию каркасных объектов, основам работы с матрицами проекций, видами преобразований. На основе этого материала был написан код для отображения объекта из задания.

На рис. 2 изображён интерфейс, который видит перед собой пользователь при запуске программы. Как видим, требования №2 («Нарисовать оси системы координат.») и №5 («При запуске программы объект сразу должно быть хорошо виден») вполне выполняются.

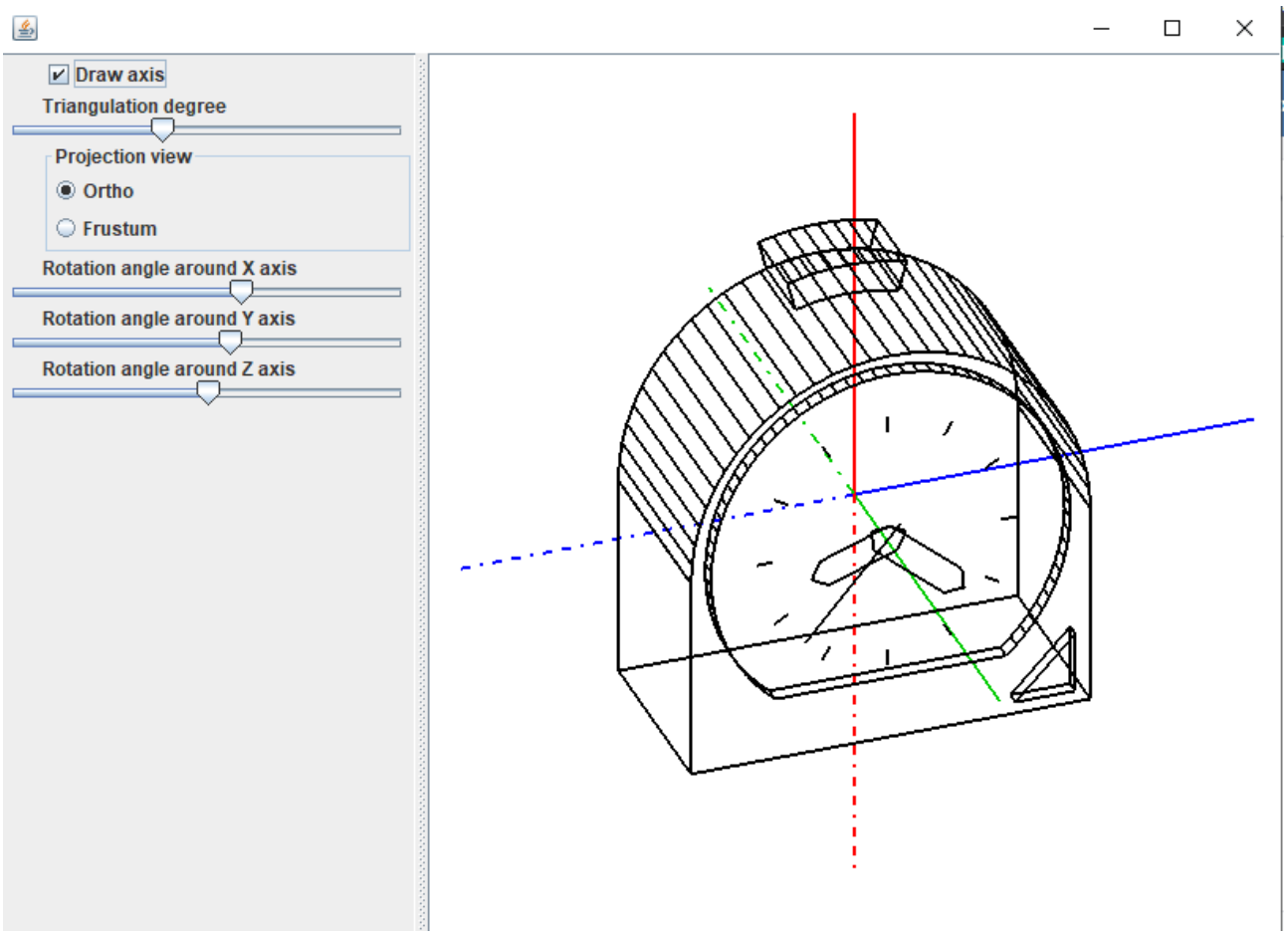
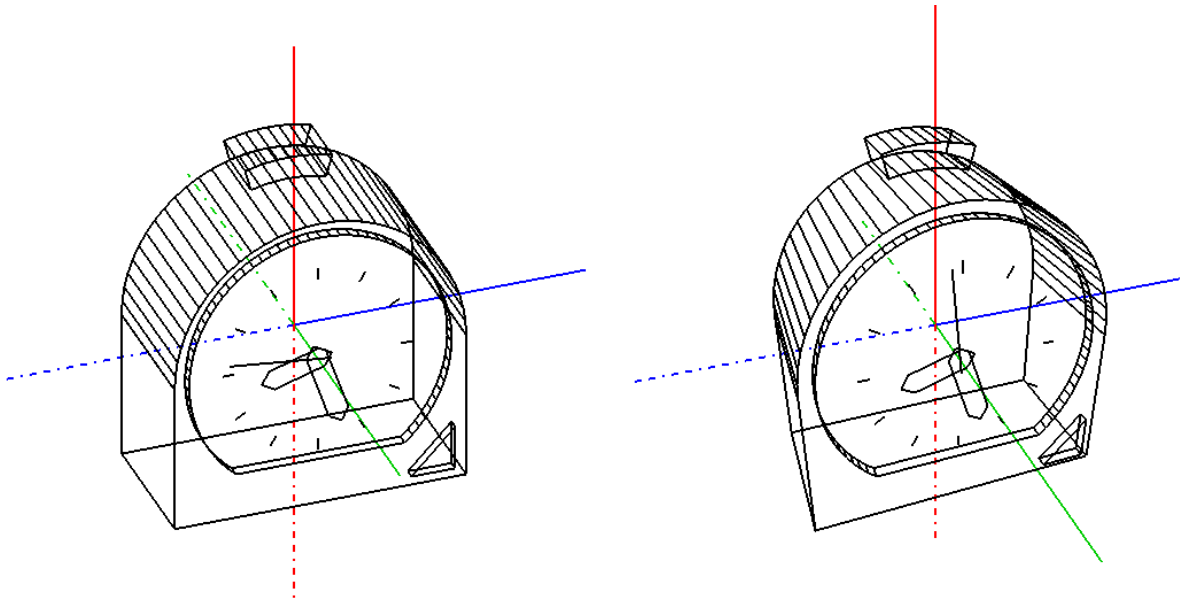


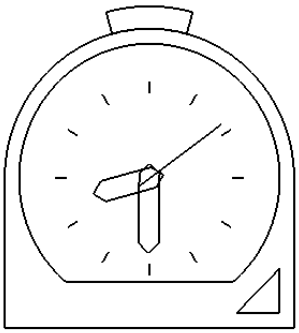
Рисунок 2 – Внешний вид интерфейса программы.

На рис. 3 можно увидеть различия между ортографической и перспективной проекциями.

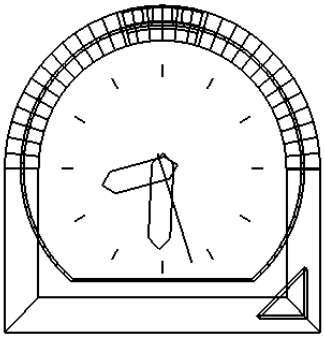


а б

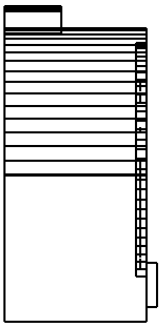
Рисунок 3 – Ортографическая (а) и перспективная (б) проекции.



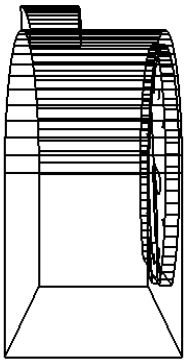
а



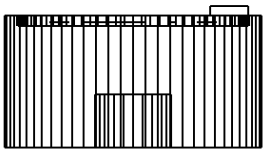
б



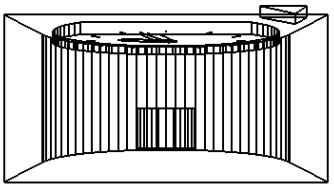
в



г



д



е

На рис. 4 можно увидеть проекции (ортографическую и перспективную) объекта на разные плоскости.

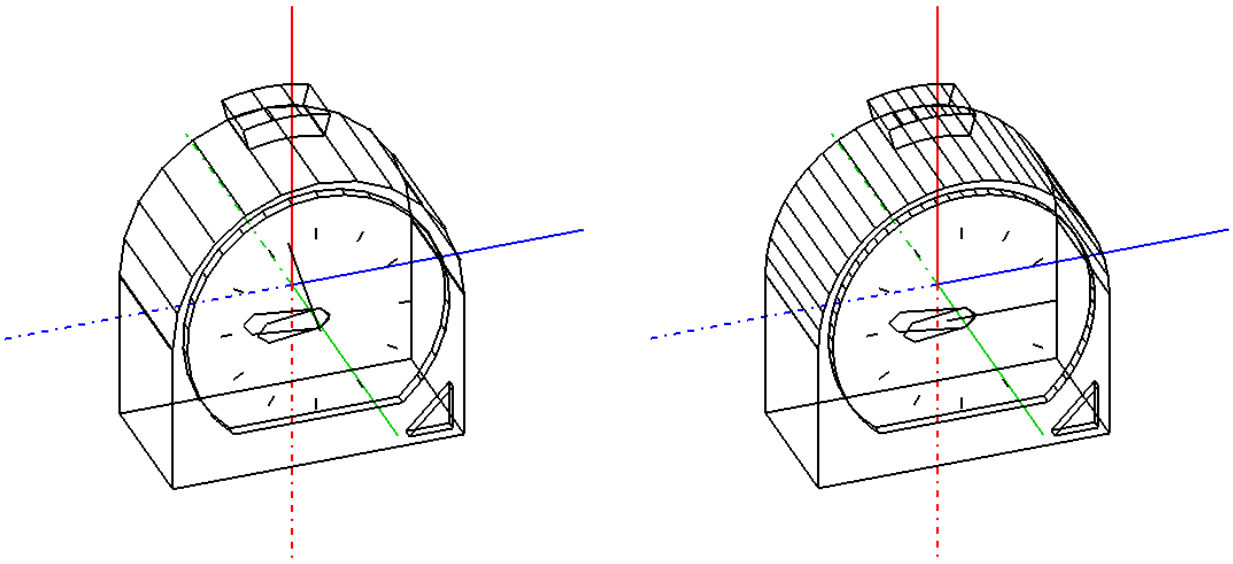
Поскольку в качестве объекта для реализации выступает будильник, было принято решение выводить на него актуальное время.

Каркасное изображение получено с использованием шейдеров OpenGL. (требование №1).

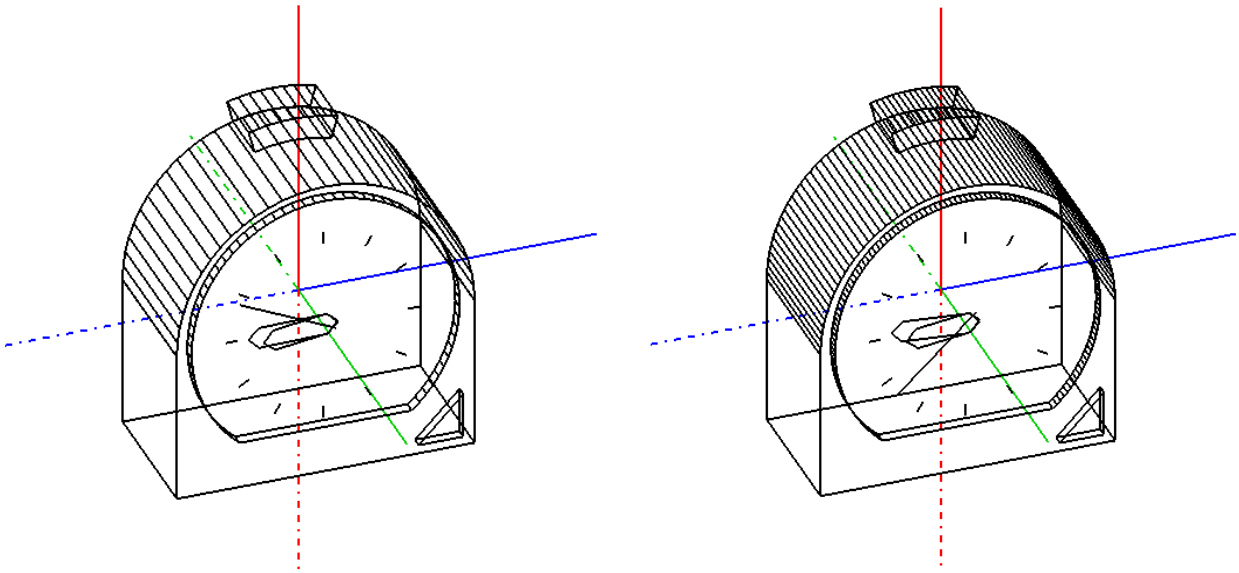
```
gl.glPolygonMode(GL.GL_FRONT_AND_BACK, GL2.GL_LINE);
```

Повороты вдоль каждой из осей можно производить независимо (требование №3).

Для выполнения требования №4 был реализован собственный метод рисования дуги с заданной мелкостью разбиения. Достойные результаты для данного объекта получаются при количестве отрезков для дуги ≥ 12. Полученные результаты для разных мелкостей разбиения представлены на рис. 5.



а б



в г

Рисунок 5 – Полученные результаты для разбиения на 12 (а), 24 (б), 32 (в), 64 (г) дуги верхней крышки будильника.

Требование №6 выполняется для 3 степеней свободы.

Выводы.

В ходе выполнения лабораторной работы были получены навыки работы с матрицами преобразований в OpenGL.

ПРИЛОЖЕНИЕ А
ТОЧКА ВХОДА ПРИЛОЖЕНИЯ

```
public class App {

    public static void main( String[] args ) {
        new GUI().run();
    }
}
```

ПРИЛОЖЕНИЕ Б

КОД КЛАССА ГРАФИЧЕСКОГО ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

```
import com.jogamp.opengl.awt.GLCanvas;
import com.jogamp.opengl.util.FPSAnimator;

import javax.swing.*;
import javax.swing.border.TitledBorder;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import java.awt.*;

public class GUI extends JFrame {

    private final FPSAnimator animator;

    public GUI() {
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setSize(830, 600);

        final ClockRenderer clockRenderer = new ClockRenderer();

        final GLCanvas clockCanvas = new GLCanvas();
        clockCanvas.addGLEventListener(clockRenderer);

        animator = new FPSAnimator(clockCanvas, 60);

        final JLabel labelTriangulationDegree = new JLabel("Triangulation
degree");
```

```
        final JSlider sliderTriangulationDegree = new JSlider(12, 64,
32);
        sliderTriangulationDegree.addChangeListener(new ChangeListener()
{
            @Override
            public void stateChanged(ChangeEvent e) {

clockRenderer.setTriangulationDegree(sliderTriangulationDegree.getValue()
);
            }
        });


        final JLabel labelRotationAngleXAxis = new JLabel("Rotation angle
around X axis");


        final DoubleSlider sliderRotationAngleXAxis = new DoubleSlider(-
180.0, 180.0, 30.0, 360);
        sliderRotationAngleXAxis.addChangeListener(new ChangeListener() {
            @Override
            public void stateChanged(ChangeEvent e) {

clockRenderer.setRotationAngleXAxis(sliderRotationAngleXAxis.getDoubleVal
ue());
            }
        });


        final JLabel labelRotationAngleYAxis = new JLabel("Rotation angle
around Y axis");


        final DoubleSlider sliderRotationAngleYAxis = new DoubleSlider(-
180.0, 180.0, 20.0, 360);
        sliderRotationAngleYAxis.addChangeListener(new ChangeListener() {
            @Override
            public void stateChanged(ChangeEvent e) {
```

```
clockRenderer.setRotationAngleYAxis(sliderRotationAngleYAxis.getDoubleVal
ue());
            }
        });

        final JLabel labelRotationAngleZAxis = new JLabel("Rotation angle
around Z axis");

        final DoubleSlider sliderRotationAngleZAxis = new DoubleSlider(-
180.0, 180.0, 0.0, 360);
        sliderRotationAngleZAxis.addChangeListener(new ChangeListener() {
            @Override
            public void stateChanged(ChangeEvent e) {

clockRenderer.setRotationAngleZAxis(sliderRotationAngleZAxis.getDoubleVal
ue());
            }
        });

        final JCheckBox checkBoxNeedToDrawAxis = new JCheckBox("Draw
axis", true);
        checkBoxNeedToDrawAxis.addChangeListener(new ChangeListener() {
            @Override
            public void stateChanged(ChangeEvent e) {

clockRenderer.setNeedToDrawAxis(checkBoxNeedToDrawAxis.isSelected());
            }
        });

        final JRadioButton radioButtonOrtho = new JRadioButton("Ortho",
true);
        final JRadioButton radioButtonFrustum = new
JRadioButton("Frustum");
```

```
        ButtonGroup buttonGroupProjections = new ButtonGroup();
        buttonGroupProjections.add(radioButtonOrtho);
        buttonGroupProjections.add(radioButtonFrustum);

        radioButtonOrtho.addChangeListener(new ChangeListener() {
            @Override
            public void stateChanged(ChangeEvent e) {
                if (radioButtonOrtho.isSelected()) {
                    clockRenderer.setProjection(Projection.ORTHO);
                } else {
                    clockRenderer.setProjection(Projection.FRUSTUM);
                }
            }
        });

        JPanel panelProjections = new JPanel();
        panelProjections.setBorder(new TitledBorder("Projection view"));
        panelProjections.setLayout(new BoxLayout(panelProjections,
BoxLayout.Y_AXIS));
        panelProjections.setAlignmentX(Component.LEFT_ALIGNMENT);
        panelProjections.add(Box.createHorizontalGlue());
        panelProjections.add(radioButtonOrtho);
        panelProjections.add(radioButtonFrustum);

        JPanel panelProperties = new JPanel();
        panelProperties.setLayout(new BoxLayout(panelProperties,
BoxLayout.Y_AXIS));
        panelProperties.add(checkBoxNeedToDrawAxis);
        panelProperties.add(labelTriangulationDegree);
        panelProperties.add(sliderTriangulationDegree);
        panelProperties.add(panelProjections);
        panelProperties.add(labelRotationAngleXAxis);
```

```
        panelProperties.add(sliderRotationAngleXAxis);
        panelProperties.add(labelRotationAngleYAxis);
        panelProperties.add(sliderRotationAngleYAxis);
        panelProperties.add(labelRotationAngleZAxis);
        panelProperties.add(sliderRotationAngleZAxis);

        JSplitPane splitPaneGUI = new JSplitPane();
        splitPaneGUI.setLeftComponent(panelProperties);
        splitPaneGUI.setRightComponent(clockCanvas);

        add(splitPaneGUI);
    }

    public void run() {
        setVisible(true);
        animator.start();
    }
}
```

ПРИЛОЖЕНИЕ В
КЛАСС, ОТВЕЧАЮЩИЙ ЗА РЕНДЕР СЦЕНЫ

```
import com.jogamp.opengl.GL;
import com.jogamp.opengl.GL2;
import com.jogamp.opengl.GLAutoDrawable;
import com.jogamp.opengl.GLEventListener;

import java.util.Calendar;

public class ClockRenderer implements GLEventListener {

    private Projection projection = Projection.ORTHO;

    private double rotationAngleXAxis = 30.0;
    private double rotationAngleYAxis = 20.0;
    private double rotationAngleZAxis = 0.0;

    private int triangulationDegree = 32;

    private boolean needToDrawAxis = true;

    public void setTriangulationDegree(int triangulationDegree) {
        this.triangulationDegree = triangulationDegree;
    }

    public void setRotationAngleXAxis(double rotationAngleXAxis) {
        this.rotationAngleXAxis = rotationAngleXAxis;
    }
```

```
public void setRotationAngleYAxis(double rotationAngleYAxis) {
    this.rotationAngleYAxis = rotationAngleYAxis;
}

public void setRotationAngleZAxis(double rotationAngleZAxis) {
    this.rotationAngleZAxis = rotationAngleZAxis;
}

public void setNeedToDrawAxis(boolean needToDrawAxis) {
    this.needToDrawAxis = needToDrawAxis;
}

public void setProjection(Projection projection) {
    this.projection = projection;
}

@Override
public void init(GLAutoDrawable glAutoDrawable) {
    GL2 gl = glAutoDrawable.getGL().getGL2();

    gl.glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
}

@Override
public void dispose(GLAutoDrawable glAutoDrawable) {

}

@Override
public void display(GLAutoDrawable glAutoDrawable) {
```

```
GL2 gl = glAutoDrawable.getGL().getGL2();

gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);

gl.glPushMatrix();
gl.glPolygonMode(GL.GL_FRONT_AND_BACK, GL2.GL_LINE);

gl.glMatrixMode(GL2.GL_PROJECTION);
gl.glLoadIdentity();
if (projection == Projection.ORTHO) {
    gl.glOrtho(-2, 2, -2, 2, -20, 20);
    gl.glTranslated(0.0, 0.0, 0.0);
} else {
    gl.glFrustum(-2, 2, -2, 2, 4.0, 25.0);
    gl.glTranslated(0.0, 0.0, -9.6);
    gl.glScaled(2.3, 2.3, 2.3);
}

gl.glMatrixMode(GL2.GL_MODELVIEW);

gl.glRotated(rotationAngleXAxis, 1.0, 0.0, 0.0);
gl.glRotated(rotationAngleYAxis, 0.0, 1.0, 0.0);
gl.glRotated(rotationAngleZAxis, 0.0, 0.0, 1.0);

gl.glLineWidth(2);

if (needToDrawAxis) {
    drawAxis(gl);
}
```

```
        drawBox(gl);

        drawArc1(gl, 0.42, 0.9, -50.0, 230.0, 2 * triangulationDegree);
        drawArc1(gl, 0.50, 0.9, -50.0, 230.0, 2 * triangulationDegree);
        drawArc1(gl, 0.42, 0.9, 230.0, 310.0, 1);
        drawArc1(gl, 0.50, 0.9, 230.0, 310.0, 1);
        drawArc2(gl, 0.46, 0.9, -50.0, 230.0, 2 * triangulationDegree,
0.08);

        drawTop(gl);
        drawBell(gl);
        drawRisks(gl);
        drawClockHands(gl);
        drawButton(gl);

        gl.glPopMatrix();
    }

    private void drawBell(GL2 gl) {
        drawArc1(gl, -0.5, 1.0, 75.0, 105.0, triangulationDegree / 2);
        drawArc1(gl, -0.1, 1.0, 75.0, 105.0, triangulationDegree / 2);
        drawArc1(gl, -0.5, 1.15, 75.0, 105.0, triangulationDegree / 2);
        drawArc1(gl, -0.1, 1.15, 75.0, 105.0, triangulationDegree / 2);
        drawArc2(gl, -0.30, 1.15, 75.0, 105.0, triangulationDegree / 4,
0.4);
        drawArc2(gl, -0.30, 1.0, 75.0, 75.0, 1, 0.4);
        drawArc2(gl, -0.30, 1.0, 105.0, 105.0, 1, 0.4);
        drawVertLine(gl, -0.5, 75.0);
        drawVertLine(gl, -0.1, 75.0);
        drawVertLine(gl, -0.5, 105.0);
        drawVertLine(gl, -0.1, 105.0);
```

```
}

private void drawVertLine(GL2 gl, double z, double angle) {
    gl.glPushMatrix();
    gl.glRotated(-angle, 0.0, 0.0, 1.0);

    gl.glBegin(GL.GL_LINES);
    gl.glColor3d(0.0, 0.0, 0.0);
    gl.glVertex3d(-1.0, 0.0, z);
    gl.glVertex3d(-1.15, 0.0, z);
    gl.glEnd();

    gl.glPopMatrix();
}

private void drawButton(GL2 gl) {
    gl.glPushMatrix();
    gl.glBegin(GL2.GL_QUAD_STRIP);

    gl.glColor3d(0.0, 0.0, 0.0);

    gl.glVertex3d(0.9, -0.9, 0.5);
    gl.glVertex3d(0.9, -0.9, 0.57);
    gl.glVertex3d(0.6, -0.9, 0.5);
    gl.glVertex3d(0.6, -0.9, 0.57);
    gl.glVertex3d(0.9, -0.6, 0.5);
    gl.glVertex3d(0.9, -0.6, 0.57);
    gl.glVertex3d(0.9, -0.9, 0.5);
    gl.glVertex3d(0.9, -0.9, 0.57);
```

```
        gl.glEnd();


        gl.glPopMatrix();
    }


    private void drawArc1(GL2 gl, double z, double R, double angle1,
double angle2, int triangulationDegree) {
        double angle = (angle2 - angle1);
        double segmentAngle = angle / triangulationDegree;
        double segmentHalfLength = R *
Math.sin(Math.toRadians(segmentAngle) / 2.0);
        double r = Math.sqrt(R * R - segmentHalfLength *
segmentHalfLength);


        gl.glPushMatrix();
        gl.glColor3d(0.0, 0.0, 0.0);


        gl.glRotated(-angle1 + segmentAngle / 2.0, 0.0, 0.0, 1.0);
        for (int i = 0; i < triangulationDegree; ++i) {
            gl.glRotated(-segmentAngle, 0.0, 0.0, 1.0);
            gl.glBegin(GL.GL_LINES);
            gl.glVertex3d(-r, -segmentHalfLength, z);
            gl.glVertex3d(-r, segmentHalfLength, z);
            gl.glEnd();
        }


        gl.glPopMatrix();
    }


    private void drawArc2(GL2 gl, double z, double r, double angle1,
double angle2, int triangulationDegree, double width) {
```

```
        double angle = (angle2 - angle1);
        double segmentAngle = angle / triangulationDegree;
        double segmentHalfLength = r *
Math.sin(Math.toRadians(segmentAngle) / 2.0);

        gl.glPushMatrix();
        gl.glColor3d(0.0, 0.0, 0.0);

        gl.glRotated(-angle1 + segmentAngle / 2.0, 0.0, 0.0, 1.0);
        for (int i = 0; i < triangulationDegree + 1; ++i) {
            gl.glRotated(-segmentAngle, 0.0, 0.0, 1.0);
            gl.glBegin(GL.GL_LINES);
            gl.glVertex3d(-r, -segmentHalfLength, z - width / 2.0);
            gl.glVertex3d(-r, -segmentHalfLength, z + width / 2.0);
            gl.glEnd();
        }

        gl.glPopMatrix();
    }

    private void drawAxis(GL2 gl) {
        gl.glBegin(GL.GL_LINES);
        gl.glColor3d(0.0, 0.0, 2.0);
        gl.glVertex3d(0.0, 0.0, 0.0);
        gl.glVertex3d( 2.0, 0.0, 0.0);
        gl.glColor3d(2.0, 0.0, 0.0);
        gl.glVertex3d(0.0, 0.0, 0.0);
        gl.glVertex3d(0.0, 2.0, 0.0);
        gl.glColor3d(0.0, 0.8f, 0.0);
```

```
        gl.glVertex3d(0.0, 0.0, 0.0);
        gl.glVertex3d(0.0, 0.0, 2.0);
        gl.glEnd();

        gl.glLineStipple(2,(short)7239);
        gl.glEnable(GL2.GL_LINE_STIPPLE);

        gl.glBegin(GL2.GL_LINES);

        gl.glColor3d(0.0f, 0.0, 2.0);
        gl.glVertex3d( 0.0, 0.0, 0.0);
        gl.glVertex3d(-2.0, 0.0, 0.0);
        gl.glColor3d(2.0, 0.0, 0.0);
        gl.glVertex3d(0.0, 0.0, 0.0);
        gl.glVertex3d(0.0, -2.0, 0.0);
        gl.glColor3d(0.0, 0.8f, 0.0);
        gl.glVertex3d(0.0, 0.0, 0.0);
        gl.glVertex3d(0.0, 0.0, -2.0);

        gl.glEnd();

        gl.glDisable(GL2.GL_LINE_STIPPLE);
    }

    private void drawBox(GL2 gl) {

        gl.glPushMatrix();
        gl.glTranslated(0.0, -1.0, 0.0);
        gl.glBegin(GL2.GL_QUAD_STRIP);
```

```
        gl.glColor3d(0.0, 0.0, 0.0);

        gl.glVertex3d(-1.0, 1.0, -0.5);
        gl.glVertex3d(-1.0, 1.0, 0.5);
        gl.glVertex3d(-1.0, 0.0, -0.5);
        gl.glVertex3d(-1.0, 0.0, 0.5);
        gl.glVertex3d(1.0, 0.0, -0.5);
        gl.glVertex3d(1.0, 0.0, 0.5);
        gl.glVertex3d(1.0, 1.0, -0.5);
        gl.glVertex3d(1.0, 1.0, 0.5);

        gl.glEnd();

        gl.glPopMatrix();
    }

    private void drawTop(GL2 gl) {
        drawArc1(gl, -0.5, 1.0, 0.0, 180.0, triangulationDegree);
        drawArc1(gl, +0.5, 1.0, 0.0, 180.0, triangulationDegree);
        drawArc2(gl, +0.0, 1.0, 0.0, 180.0, triangulationDegree, 1.0);
    }

    private void drawRisks(GL2 gl) {
        gl.glPushMatrix();
        gl.glColor3d(0.0, 0.0, 0.0);
        gl.glTranslated(0.0, 0.0, 0.45);

        for (int i = 0; i < 12; ++i) {
            gl.glRotated(30, 0.0, 0.0, 1.0);
            gl.glBegin(GL.GL_LINES);
```

```
            gl.glVertex2d(0.0, 0.65);
            gl.glVertex2d(0.0, 0.57);
            gl.glEnd();
        }

        gl.glPopMatrix();
    }


    private void drawClockHands(GL2 gl) {
        Calendar calendar = Calendar.getInstance();
        double seconds = calendar.get(Calendar.SECOND);
        double minutes = calendar.get(Calendar.MINUTE) + seconds / 60.0;
        double hours = calendar.get(Calendar.HOUR) + minutes / 60.0;

        gl.glPushMatrix();

        drawHourHand(gl, hours);
        drawMinuteHand(gl, minutes);
        drawSecondHand(gl, seconds);

        gl.glPopMatrix();
    }

    private void drawHourHand(GL2 gl, double hours) {
        gl.glPushMatrix();

        gl.glRotated(-30.0 * hours, 0.0, 0.0, 1.0);
        gl.glTranslated(0.0, -0.1, 0.45);
```

```
        gl.glBegin(GL2.GL_POLYGON);
        gl.glVertex2d(0.0, 0.0);
        gl.glVertex2d(0.07, 0.07);
        gl.glVertex2d(0.07, 0.43);
        gl.glVertex2d(0.0, 0.5);
        gl.glVertex2d(-0.07, 0.43);
        gl.glVertex2d(-0.07, 0.07);
        gl.glEnd();

        gl.glPopMatrix();
    }

    private void drawMinuteHand(GL2 gl, double minutes) {
        gl.glPushMatrix();

        gl.glRotated(-6.0 * minutes, 0.0, 0.0, 1.0);
        gl.glTranslated(0.0, -0.1, 0.45);

        gl.glBegin(GL2.GL_POLYGON);
        gl.glVertex2d(0.0, 0.0);
        gl.glVertex2d(0.07, 0.07);
        gl.glVertex2d(0.07, 0.53);
        gl.glVertex2d(0.0, 0.6);
        gl.glVertex2d(-0.07, 0.53);
        gl.glVertex2d(-0.07, 0.07);
        gl.glEnd();

        gl.glPopMatrix();
    }
```

```
    private void drawSecondHand(GL2 gl, double seconds) {
        gl.glPushMatrix();

        gl.glRotated(-6.0 * seconds, 0.0, 0.0, 1.0);
        gl.glTranslated(0.0, -0.1, 0.45);

        gl.glBegin(GL.GL_LINES);
        gl.glVertex2d(0.0, 0.0);
        gl.glVertex2d(0.0, 0.71);
        gl.glEnd();

        gl.glPopMatrix();
    }

    @Override
    public void reshape(GLAutoDrawable glAutoDrawable, int x, int y, int
width, int height) {

    }
}
```

ПРИЛОЖЕНИЕ Е

КОД КЛАССА УДОБНОГО ПОЛЗУНКА

```
import javax.swing.*;

public class DoubleSlider extends JSlider {

    private final double minimum;
    private final double maximum;
    private final double scale;

    public DoubleSlider(double min, double max, double value, int scale)
{
        super(0, scale, (int)(scale * (value - min) / (max - min)));

        this.minimum = min;
        this.maximum = max;
        this.scale = scale;
    }

    public double getDoubleValue() {
        return (minimum + (maximum - minimum) * getValue() / scale);
    }
}
```