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

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Компьютерная графика»

Тема: Расширения OpenGL, программируемый графический конвейер. Шейдеры.

Студент гр. 0304 _____ Алексеев Р.В.

Преподаватель _____ Герасимова Т.В.

Санкт-Петербург

2023

Цель работы.

- Освоение работы с шейдерами в OpenGL.

Задание.

Разработать визуальный эффект по заданию, реализованный средствами языка шейдеров GLSL.

Вариант Эффекта: 21

Задание 2:

В последующих номерах 20 – 39 необходимо развить предыдущую лабораторную работу, превратив кривую в поверхность на сцене и добавив в программу дополнительный визуальный эффект, реализованный средствами языка шейдеров.

21. Анимация. Цветовые координаты изменяются по синусоидальному закону

Выполнение работы.

При помощи языка C++ и фреймворка Qt была написана программа, отрисовывающая плоскость из сплайна путем растягивания его вдоль оси Z.

Для реализации программы были написаны фрагментный шейдер и вершинный шейдер.

Фрагментный шейдер:

```
#version 460 core
in vec3 color;
out vec4 FragColor;

void main()
{
    FragColor = vec4(
        clamp(0.4 + color.x, 0.0, 1.0),
        clamp(0.1 + color.y, 0.0, 1.0),
        clamp(0.4 + color.z, 0.0, 1.0),
        1.0
```

```
    );
}
```

Вершинный шейдер:

```
#version 460 core
layout (location = 0) in vec3 vPos;
out vec3 color;

uniform mat4 transformations;
uniform float time;

void main()
{
    const vec3 axis = vec3(0.0, 1.0, 0.0);
    vec3 delta = vec3(0.0, 0.0, 0.0);
    const float amp = 0.2;
    //вычисление значения для цветовых координат
    delta += amp * sin(time);

    gl_Position = transformations * vec4(vPos, 1.0);

    color = delta;
}
```

Вершинный шейдер в качестве входных данных принимает вектор с координатыми вершины — *vPos*, матрицу преобразований — *transformations*, время — *time*. В качестве выходных данных шейдер устанавливает вектор цветовых координат — *color*, и изменяет позиции вершин согласно матрице трансформации. Для изменения цветовых координат по синусоидальному закону вычисляется произведение константы и синуса от времени. Фрагментный шейдер принимает вектор с цветовыми координатами и устанавливает их.

Для инициализации программы и подключения шейдеров использован следующие методы класса GLWidget:

```
void GLWidget::initializeGL()
{
    QColor bgc(255, 255, 255);
```

```
    initializeOpenGLFunctions();
    glClearColor(bgc.redF(), bgc.greenF(), bgc.blueF(), bgc.alphaF());
    // подключение шейдеров
    if(!initShaderProg())
    {
        std::cerr << _shaderProg.log().toStdString() << std::endl;
        return;
    }

    // генерация точек сплайна
    generatePoints();
    createSpline();
    bindVert();
    // выбор режима растрирования
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);
    // подключение к таймеру
    connect(_timer, &QTimer::timeout, this, &GLWidget::updateTimer);
    _timer->start(200);
}

void GLWidget::resizeGL(int w, int h)
{
    _width = w;
    _height = h;

    glViewport(0, 0, w, h);
    // задание матрицы пребразований
    QMatrix4x4 transformMatrix;
    transformMatrix.setToIdentity();
    transformMatrix.rotate(15.f, 1.f, 0.f, 0.f);

    int matrixLocation = _shaderProg.uniformLocation("transformations");
    _shaderProg.setUniformValue(matrixLocation, transformMatrix);
}

void GLWidget::paintGL()
{
    _vao.bind();
    _vbo.bind();
    _ebo.bind();
```

```
    _shaderProg.bind();

    // установка переменных для передачи значений в шейдеры
    int vertexLocation = _shaderProg.attributeLocation("vPos");

    _shaderProg.enableAttributeArray(vertexLocation);
          _shaderProg.setAttributeBuffer(vertexLocation,  GL_FLOAT,  0,  3,
sizeof(GLfloat)*6);

    int timeLocation = _shaderProg.uniformLocation("time");
    _shaderProg.setUniformValue(timeLocation, _time);

             glDrawElements(GL_TRIANGLE_STRIP,   6*(_splinePoints.size()-1),
GL_UNSIGNED_SHORT, nullptr);
}

bool GLWidget::initShaderProg()
{
    _vao.create();
    _vao.bind();

    _vbo.create();
    _ebo.create();

    // подключаем вершинный шейдер
              if(!_shaderProg.addShaderFromSourceFile(QOpenGLShader::Vertex,
":/shader.vs"))
    {
        return false;
    }
    // подключаем фрагментный шейдер
            if(!_shaderProg.addShaderFromSourceFile(QOpenGLShader::Fragment,
":/shader.fs"))
    {
        return false;
    }
    if(!_shaderProg.link())
    {
        return false;
    }
    if(!_shaderProg.bind())
    {
```

```
        return false;
    }

    return true;
}

void GLWidget::generatePoints()
{
    _points.push_back(QPointF{-0.8, 0.7});
    _points.push_back(QPointF{-0.5, -0.5});
    _points.push_back(QPointF{0.0, 0.0});
    _points.push_back(QPointF{0.2, -0.9});
    _points.push_back(QPointF{0.3, 0.4});
    _points.push_back(QPointF{0.6, -0.5});
    _points.push_back(QPointF{0.9, 0.1});
}

void GLWidget::createSpline()
{
    for(int i = 0; i < _points.size(); i++)
    {
        _splinePoints.clear();
        for(float t = 0; t < 1.0; t += 0.01)
        {
            _splinePoints.push_back(getPoint(i, t));
        }
    }
}

QPointF GLWidget::getPoint(int num, float t)
{
    QVector<QPointF> newPoints = _points;

    int i = num;

    while (i > 0) {
        for(int k = 0; k < i; k++)
        {
                    newPoints[k] = newPoints[k] + t * (newPoints[k+1] -
newPoints[k]);
        }
```

```
        i--;
    }

    return newPoints[0];
}

void GLWidget::bindVert()
{
    int vCount = 12 * _splinePoints.size();
    int iCount = 6 * (_splinePoints.size() - 1);
    GLfloat* vertices = new GLfloat[vCount];
    GLushort* indices = new GLushort[iCount];

    // копируем вершины в буфер
    for (GLushort i = 0; i < _splinePoints.size(); ++i)
    {
        // {x, y, 0, n1}
        vertices[12 * i + 0]  = _splinePoints[i].x();
        vertices[12 * i + 1]  = _splinePoints[i].y();
        vertices[12 * i + 2]  = 0.f;
        // {x, y, -1, n2}
        vertices[12 * i + 6]  = _splinePoints[i].x();
        vertices[12 * i + 7]  = _splinePoints[i].y();
        vertices[12 * i + 8]  = -1.f;
    }
    GLfloat dx1, dx2, dy1, dy2, dz1, dz2;
    // копируем индексы в буфер
    for (GLushort i = 0; i < _splinePoints.size() - 1; ++i)
    {
        // добавляем индексы в массив
        indices[6 * i + 0] = 2 * i + 0;
        indices[6 * i + 1] = 2 * i + 2;
        indices[6 * i + 2] = 2 * i + 1;
        indices[6 * i + 3] = 2 * i + 3;
        indices[6 * i + 4] = 2 * i + 0;
        indices[6 * i + 5] = 2 * i + 2;
    }

    _vbo.bind();
    _vbo.allocate(vertices, sizeof(GLfloat) * vCount);

    _ebo.bind();
```

```
    _ebo.allocate(indices, sizeof(GLushort) * iCount);

    _vbo.release();
    _ebo.release();
    delete [] vertices;
    delete [] indices;
}
```

В шейдер передается матрица преобразования, с помощью которого задается наклон камеры, также передается время, чтобы цвет наглядно изменялся по синусоидальному закону.

Данные о вершинах — координаты, сохраняются в переменную *_vbo*, для порядка отрисовки используется *_ebo*, в котором сохранена информация о индексах вершин, из которых состоят треугольники.

Тестирование.

Результаты тестирования в разные моменты времени представлены на рис. 1-3.

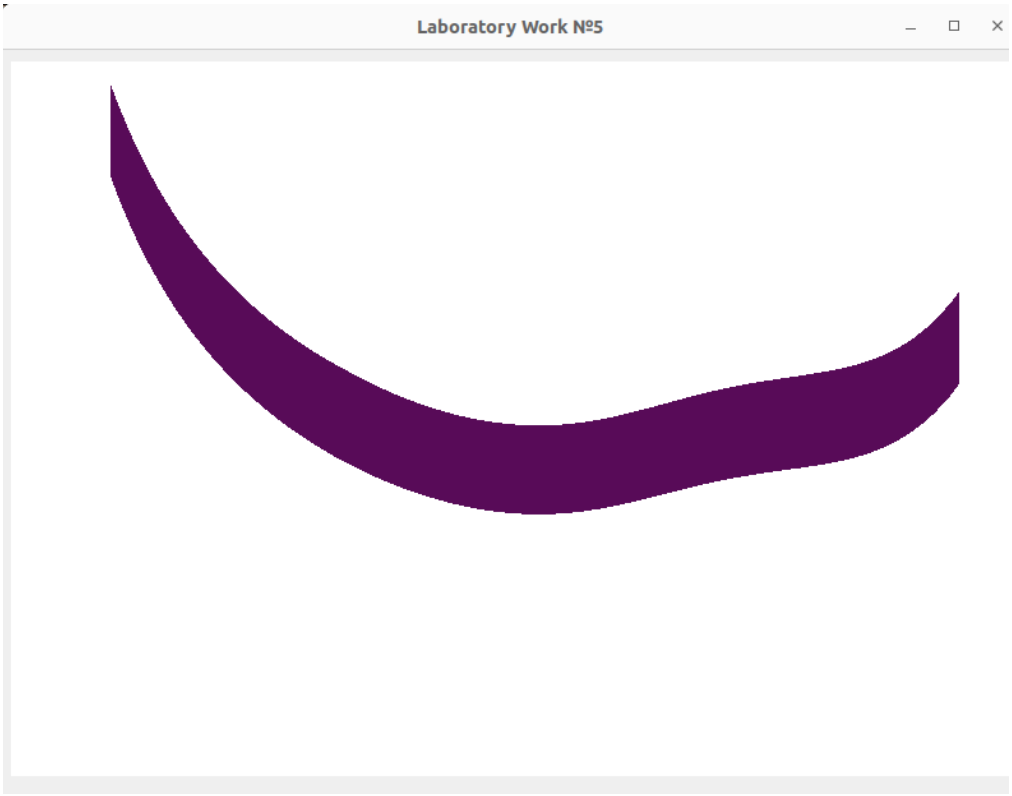

Рисунок 1.

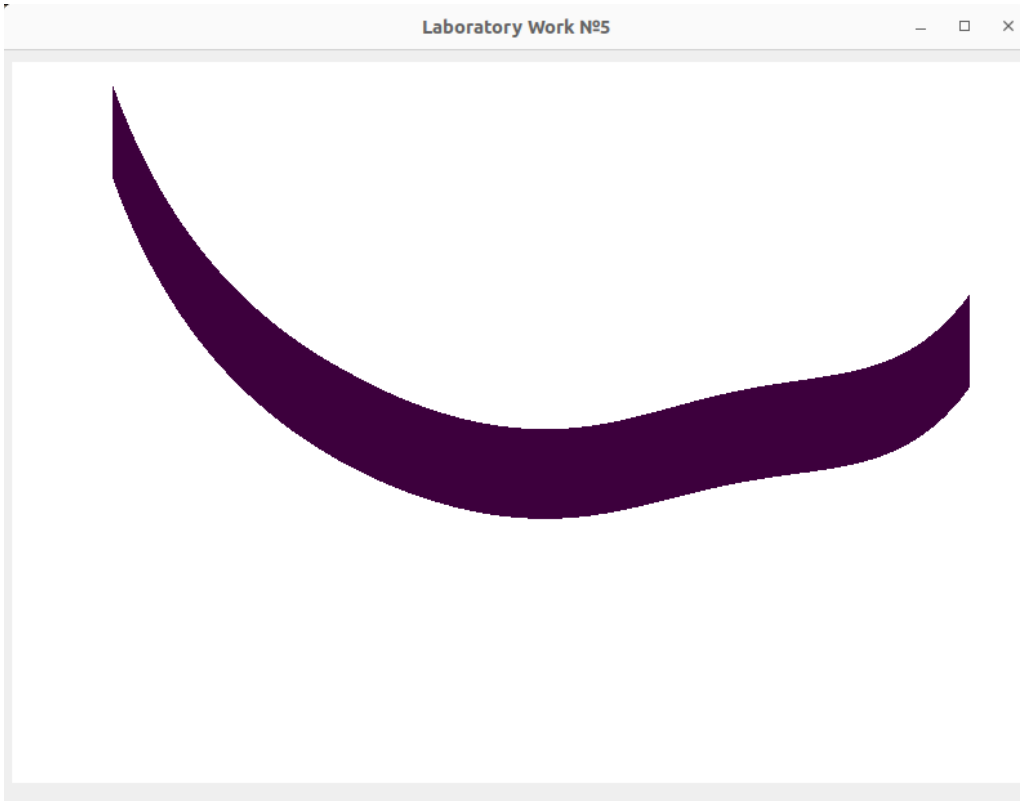

Рисунок 2.

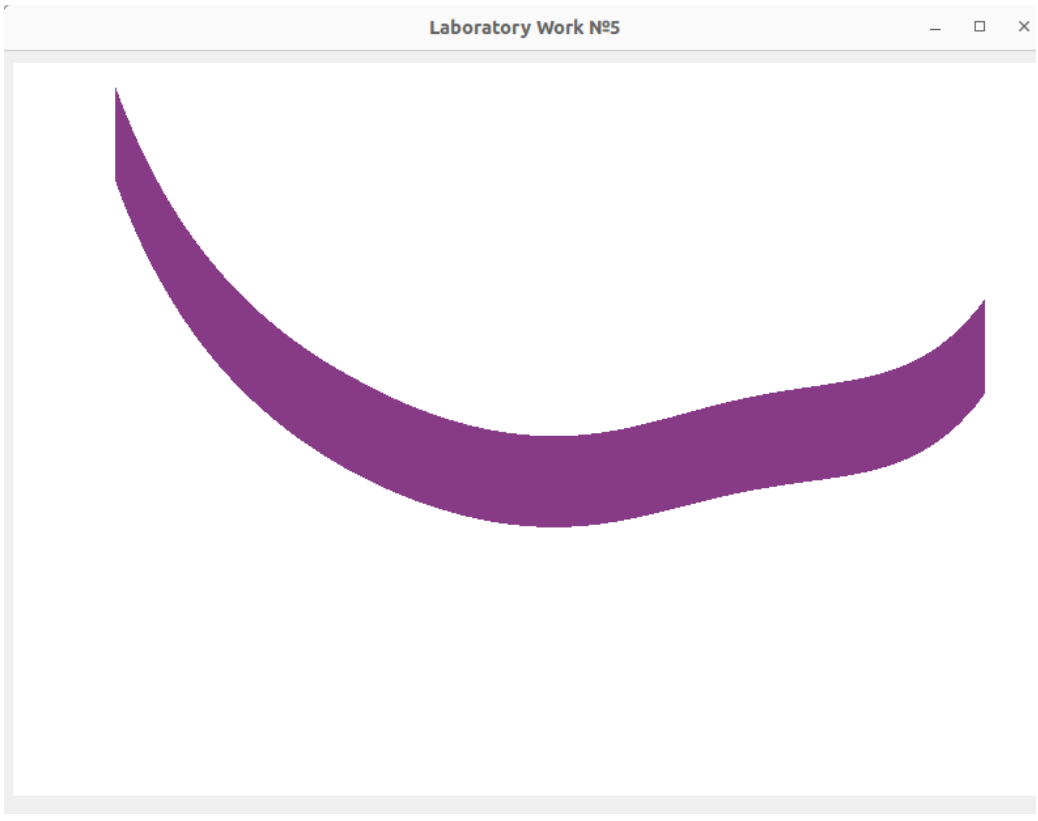

Рисунок 3.

Выводы.

В ходе работы была разработана программа, реализующая анимацию по средствам шейдеров OpenGL.

Был получен опыт работы с шейдерами OpenGL.