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 _____ Максименко Е.М.

Преподаватель _____ Герасимова Т.В.

Санкт-Петербург
2023

Цель работы.

- Изучение языка шейдеров GLSL

- Изучение способов применения шейдеров в программах OpenGL

Задание.

Задание 2:

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

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.

Вариант Эффекта: 36

Анимация. Сдвиг вдоль нормали пропорционально времени всех вершин, у которых нормаль составляет с осью OZ острый угол.

Выполнение работы.

Работа была выполнена с использованием языка программирования C++ и фреймворка Qt 6. Каркасом программы послужила программа из работы 1.

Для отрисовки плоскости был использован сплайн из работы 4, при этом контрольные точки для сплайна были заданны программно. По контрольным точкам были построены точки сплайна на промежутке $[x_0; x_n]$, где x_0 — наименьший из x, x_n — наибольший из x. Данные точки передавались вершинному шейдеру.

Для передачи данных о вершинах вершинному шейдеру использовались такие структуры, как VAO (Vertex Array Object), VBO (Vertex Buffer Object), EBO (Element Buffer Object). Данные об одной вершине представляются в виде 6 значений Glfloat: первые 3 значения — координаты вершины (нормализованные),

следующие 3 значения — вектор нормали к плоскости в данной точке. Данные о вершинах записывались в следующем порядке: $\{x_0, y_0, 0\}$, $\{x_0, y_0, -1\}$, $\{x_1, y_1, 0\}$, $\{x_1, y_1, -1\}$ и т. д., где x_i и y_i — координаты точки на сплайне. Использование значений $z = 0$ и $z = -1$ обусловлено тем, что плоскость строится из сплайна путем вытягивания его вдоль оси Z. Нормаль для точки с $x=x_i$ и $y=y_i$ вычисляется по точкам $\{x_i, y_i, 0\}$, $\{x_i, y_i, -1\}$, $\{x_{i+1}, y_{i+1}, 0\}$ (см. источник 1).

Данные о вершине записываются в VBO. Также, для указания порядка отрисовки треугольников, используется EBO. В нем указываются для каждого треугольника индексы трех вершин из VBO, из которых составлен данный треугольник.

В шейдер также передается матрица преобразований для задания наклона камеры, чтобы плоскость была видна, а также время, которое определяет величину смещения вершины вдоль нормали. Данные величины передаются как uniform. Реализацию отдельных методов класса GLScene, отвечающих за подготовку данных и передачу их шейдеру см. в листинге 1.

Листинг 1. Реализация методов подготовки данных и передачи их в шейдер.

```
void GLScene::initializeGL()
{
	QColor bgc(255, 255, 255);
	initializeOpenGLFunctions();
	glClearColor(bgc.redF(), bgc.greenF(), bgc.blueF(), bgc.alphaF());

	if (!initShaderProgram())
	{
		std::cerr << _shaderProgram.log().toStdString() << std::endl;
		return;
	}
	generateControlPoints();
	calculateSplinePoints();
	bindVertices();

	glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
	glEnable(GL_DEPTH_TEST);
	glEnable(GL_CULL_FACE);

	connect(_timer, &QTimer::timeout, this, &GLScene::updateTimer);
	_timer->start(30);
}
void GLScene::resizeGL(int w, int h)
{
	_width = w;
	_height = h;

	glViewport(0, 0, w, h);
```

```
      QMatrix4x4 transformMatrix;
      transformMatrix.setToIdentity();
      transformMatrix.rotate(50.f, 1.f, 0.f, 0.f);

      int matrixLocation = _shaderProgram.uniformLocation("transformations");
      _shaderProgram.setUniformValue(matrixLocation, transformMatrix);
}
void GLScene::paintGL()
{
    _vao.bind();
    _vbo.bind();
    _ebo.bind();
    _shaderProgram.bind();
    int vertexLocation = _shaderProgram.attributeLocation("vPos");
    int normalLocation = _shaderProgram.attributeLocation("vNorm");

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

    _shaderProgram.enableAttributeArray(normalLocation);
    _shaderProgram.setAttributeBuffer(normalLocation, GL_FLOAT, 3 *
sizeof(GLfloat), 3, sizeof(GLfloat) * 6);

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

    glDrawElements(GL_TRIANGLE_STRIP, 6 * (_splinePoints.size() - 1),
GL_UNSIGNED_SHORT, nullptr);
}
bool GLScene::initShaderProgram()
{
    _vao.create();
    _vao.bind();

    _vbo.create();
    _ebo.create();
    // compile vertex shader
    if (
        !_shaderProgram.addShaderFromSourceFile(
            QOpenGLShader::Vertex,
            ":/shaders/lab5.vs"
        )
    )
    {
        return false;
    }
    // compile fragment shader
    if (
        !_shaderProgram.addShaderFromSourceFile(
            QOpenGLShader::Fragment,
            ":/shaders/lab5.fs"
        )
    )
    {
        return false;
    }
    // link shaders to program
    if (!_shaderProgram.link())
    {
        return false;
    }
    // bind shader program to use
    if (!_shaderProgram.bind())
    {
        return false;
    }
    return true;
}
void GLScene::bindVertices()
```

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

    // copy verticies to buffer
    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;
    // copy indices to buffer
    for (GLushort i = 0; i < _splinePoints.size() - 1; ++i)
    {
        // add indices to array
        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;

        // calculate normals
        dx1 = _splinePoints[i + 1].x() - _splinePoints[i].x();
        dx2 = 0.f;
        dy1 = _splinePoints[i + 1].y() - _splinePoints[i].y();
        dy2 = 0.f;
        dz1 = 0.f;
        dz2 = -1.f;

        vertices[12 * i + 3] = dy1 * dz2 - dy2 * dz1;
        vertices[12 * i + 9] = dy1 * dz2 - dy2 * dz1;
        vertices[12 * i + 4] = -(dx1 * dz2 - dx2 * dz1);
        vertices[12 * i + 10] = -(dx1 * dz2 - dx2 * dz1);
        vertices[12 * i + 5] = dx1 * dy2 - dx2 * dy1;
        vertices[12 * i + 11] = dx1 * dy2 - dx2 * dy1;
    }
    vertices[2 * iCount + 3] = dy1 * dz2 - dy2 * dz1;
    vertices[2 * iCount + 9] = dy1 * dz2 - dy2 * dz1;
    vertices[2 * iCount + 4] = -(dx1 * dz2 - dx2 * dz1);
    vertices[2 * iCount + 10] = -(dx1 * dz2 - dx2 * dz1);
    vertices[2 * iCount + 5] = dx1 * dy2 - dx2 * dy1;
    vertices[2 * iCount + 11] = dx1 * dy2 - dx2 * dy1;
    // fill vbo and ebo
    _vbo.bind();
    _vbo.allocate(vertices, sizeof(GLfloat) * vCount);

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

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(GLfloat) * 6,
nullptr);

    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(GLfloat) * 6,
reinterpret_cast<void*>(sizeof(GLfloat) * 3));

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

Вершинный шейдер принимает вектор координат вершины, а также вектор нормали к плоскости из данной точки. Также он принимает матрицу преобразований и время. На выходе данный шейдер устанавливает позицию вершины, а также передает значение смещения вершины вдоль нормали для изменения цвета. Косинус угла между нормалью и осью Y (выбрана вместо оси Z, т. к. нормали всех вершин перпендекулярны оси Z) определяется по скалярному произведению данных векторов, деленному на произведение длин данных векторов. Если косинус положительный, то угол острый и вершина будет смещена вдоль нормали. Смещение вдоль нормали определяется следующим образом: вектор нормали нормируется, после чего все компоненты умножаются на синус от величины времени и на амплитуду смещения. Также величина смещения передается в фрагментный шейдер для индикации величины смещения вершин. Исходный код шейдеров см в листингах 2 и 3.

Листинг 2. Исходный код вершинного шейдера.

```
#version 460 core

layout (location = 0) in vec3 vPos;
layout (location = 1) in vec3 vNorm;
out vec3 color;

uniform mat4 transformations;
uniform float time;

float angleBetweenVectors(vec3 v1, vec3 v2)
{
    return dot(v1, v2) / (length(v1) * length(v2));
}

void main()
{
    const vec3 axis = vec3(0.0, 1.0, 0.0);
    vec3 delta = vec3(0.0, 0.0, 0.0);

    if (angleBetweenVectors(vNorm, axis) > 0)
    {
        delta += 0.05 * normalize(vNorm) * sin(time);
    }

    gl_Position = transformations * vec4(
        vPos + delta,
        1.0
    );
    color = 1.5 * delta;
}
```

Листинг 3. Исходный код фрагментного шейдера.

```
#version 460 core

in vec3 color;
```

```
out vec4 FragColor;

void main()
{
    FragColor = vec4(
        clamp(0.2 + color.x, 0.0, 1.0),
        clamp(0.6 + color.y, 0.0, 1.0),
        clamp(0.1 + color.z, 0.0, 1.0),
        1.0
    );
}
```

Тестирование.

Программа содержит анимацию смещения вершин вдоль нормалей. Кадры из данной анимации см. на рис. 1-3.

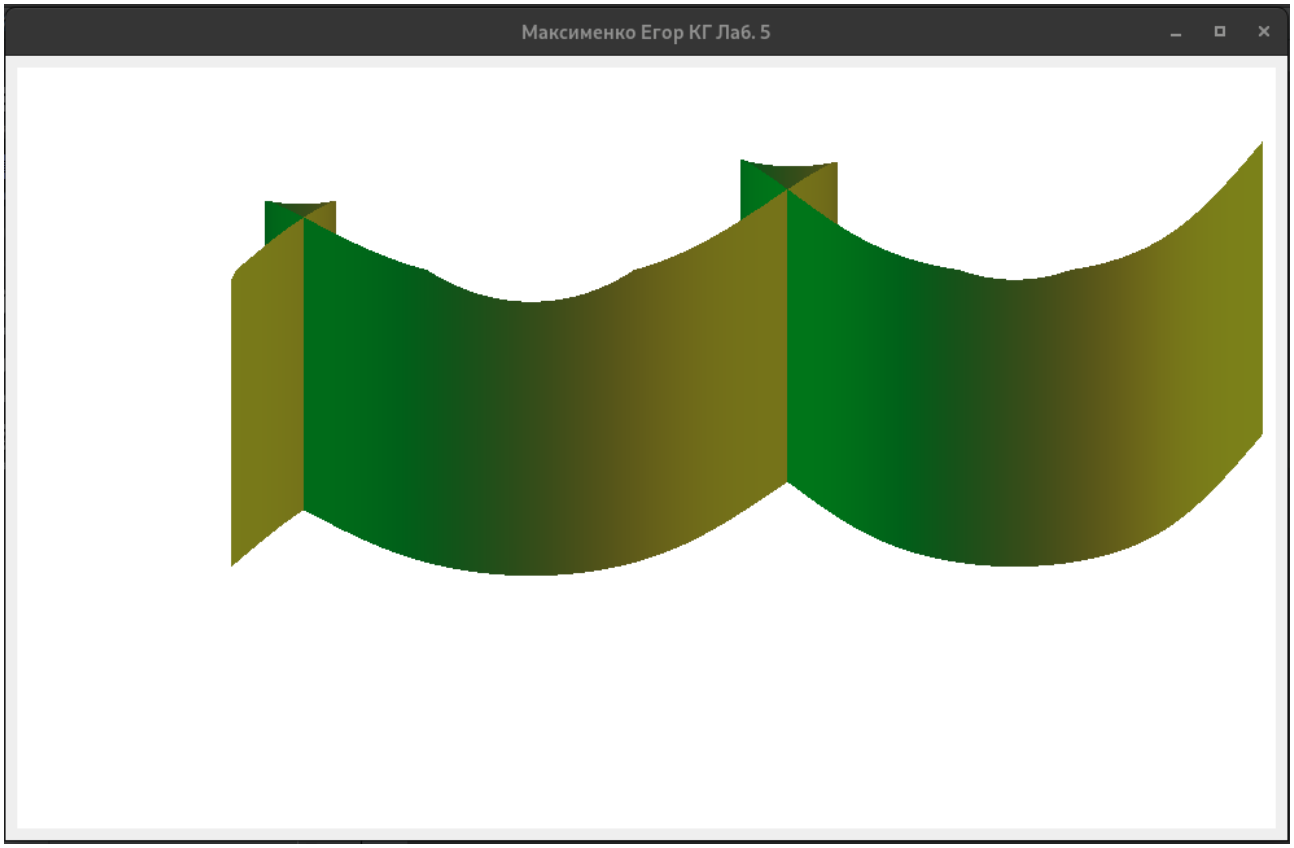

Рисунок 1.

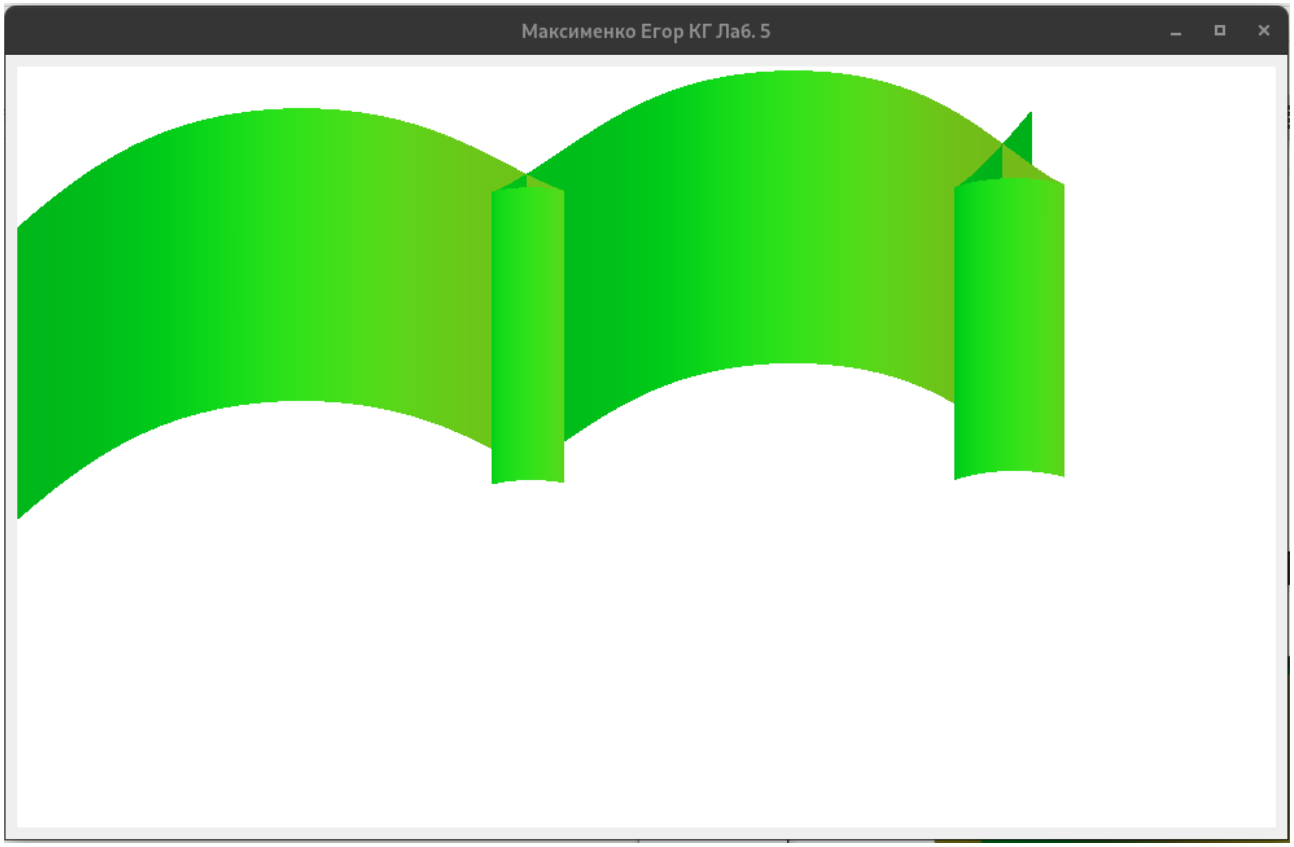

Рисунок 2.

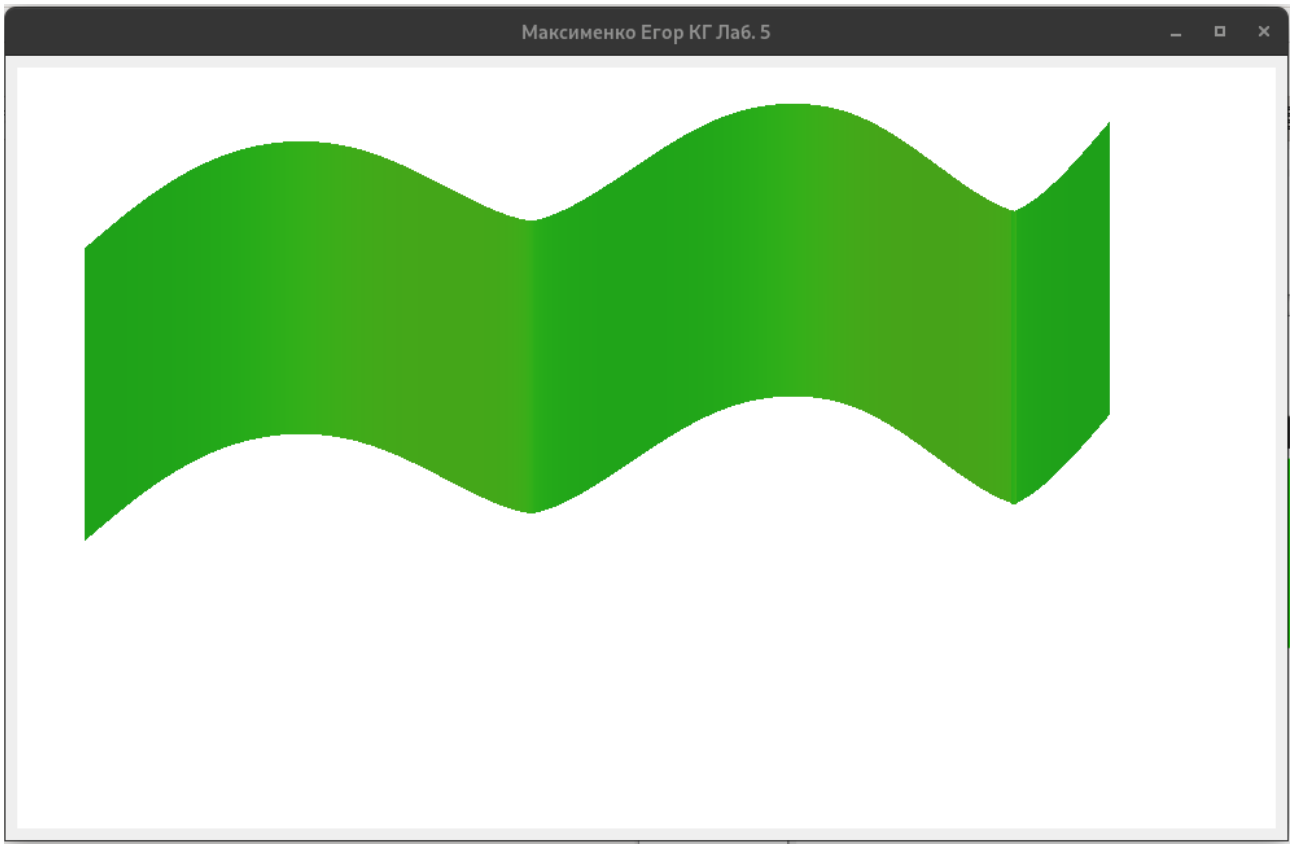

Рисунок 3.

Вывод.

В ходе работы был изучен графический конвейер OpenGL и работа с шейдерами.

Была разработана программа, выполняющая анимацию смещения вершин вдоль нормали по синусоидальному закону.

СПИСОК ИСТОЧНИКОВ

1. WEB ресурс mathprofi.ru: http://mathprofi.ru/uravnenie_ploskosti.html

2. WEB ресурс learnopengl.com: https://learnopengl.com/Getting-started/Hello-Triangle