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

Отчет
по лабораторной работе №1
по дисциплине «3D Компьютерная графика»

Студент группы 5304		Лянгузов А. А.
Преподаватель:		Герасимова Т.В.

Санкт-Петербург
2019

Цель работы: ознакомление с основными примитивами WebGL. Требования и рекомендации к выполнению задания: • проанализировать полученное задание, выделить информационные объекты и действия;

• разработать программу с использованием требуемых примитивов и атрибутов.

Задание

Получите удобное рисование с буферами вершин, униформой и шейдерами:

- рисование нескольких вещей с отдельными командами рисования.
- используя разные примитивы.
- изменение размеров линий и точек по умолчанию.
- изменение цвета на лету.

Основные теоретические положения

Для создания приложения WebGL для рисования точек необходимы следующие шаги. **Шаг 1.** Подготовьте холст и получите контекст рендеринга WebGL На этом этапе мы получаем объект контекста рендеринга WebGL, используя метод getContext ().

Шаг 2. Определите геометрию и сохраните ее в буфере объектов Поскольку мы рисуем три точки, мы определяем три вершины с трехмерными координатами и сохраняем их в буферах. **Шаг 3.** Создать и скомпилировать шейдерную программу На этом этапе вам нужно написать программы вершинного шейдера и фрагментного шейдера, скомпилировать их и создать объединенную программу, связав эти две программы. **Шаг 4.** Связать шейдерные программы для буферизации объектов На этом этапе мы связываем объекты буфера с программой шейдера. **Шаг 5.** Рисование необходимого объекта

Ход работы

При последовательном выполнении лабораторных работ должно быть получено итоговое изображение:

Рисунок 1 — Финальная сцена.

Шаг 1. Создана базовая html- страница с холстом (canvas) и подключением базовых шейдоров (index.html).

Шаг 2. Созданы базовые модельные классы: цвет (**color.js**), позиция (**position.js**), вершина (**vertex.js**), фигура (**figure.js**).

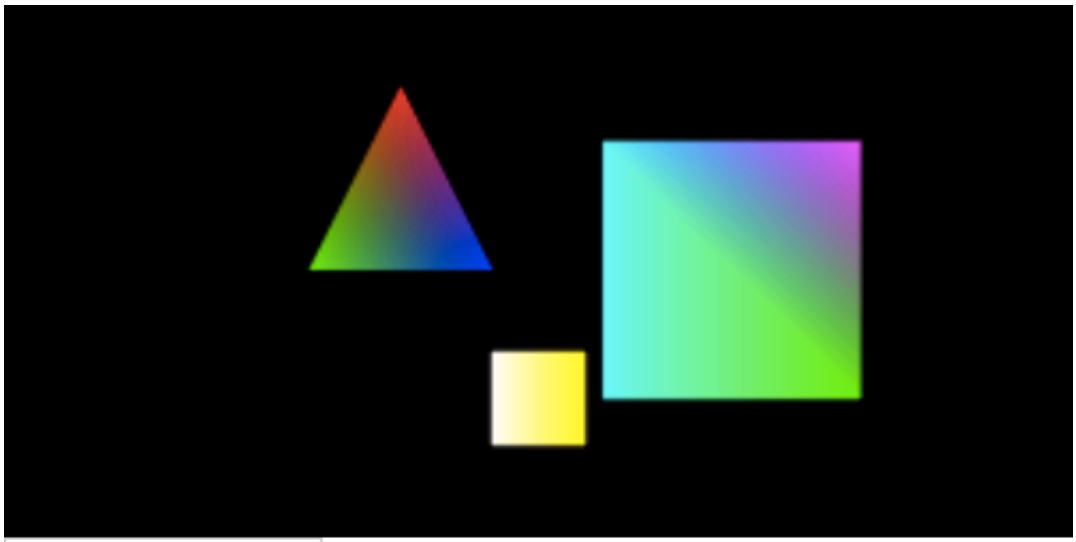
Шаг 3. Создан репозиторий с фигурами (**shapes_repository.js**).

Шаг 4. Написан драйвер для работы с WebGL (**webgl-driver.js** - просто обертка, инкапсулирующая большинство функций и упрощающая программный интерфейс).

Шаг5. Написан головной файл приложения (**app.js**).

Шаг 6. Настроена смена цвета на лету.

Итоговая картинка:



1,0,0,1
0,1,0,1
0,0,1,1
1,0,1,1
0,1,1,1
0,1,0,1
0,1,1,1
1,1,0,1
1,1,1,1
1,1,0,1
1,1,1,1

Рисунок 2 — Результат работы программы.

Выводы.

В ходе лабораторной работы были изучены основные примитивы WebGL, проанализировано полученное задание, выделены информационные объекты и действия. Разработана программа с использованием требуемых примитивов и атрибутов.

Исходный код.

Файл index.html:

```
<html>
<head>
<title>3D-graphics</title>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
<link rel="stylesheet" href="./app.css" />
<script type="text/javascript" src="./libs/glMatrix-0.9.5.min.js"></script>
<script id="shader-fs" type="x-shader/x-fragment">
    precision mediump float;

    varying vec4 vColor;

    void main(void) {
        gl_FragColor = vColor;
    }
</script>
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec4 aVertexColor;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

    varying vec4 vColor;

    void main(void) {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vColor = aVertexColor;
    }
</script>
<script type="text/javascript" src="./models/webgl-driver.js"></script>
<script type="text/javascript" src="./models/figure.js"></script>
<script type="text/javascript" src="./models/vertex.js"></script>
<script type="text/javascript" src="./models/position.js"></script>
<script type="text/javascript" src="./models/color.js"></script>
<script type="text/javascript" src="./models/shapes_repository.js"></script>
<script type="text/javascript" src="./app.js"></script>
</head>
<body onload="main('canvas-field');">
    <canvas id="canvas-field"></canvas>
    <div id="vertexColors">
    </div>
</body>
</html>
```

Файл color.js:

```
class Color {
    constructor(r, g, b, alpha) {
        this.r = r;
        this.g = g;
        this.b = b;
        this.alpha = alpha;
    }

    getColorCode() {
        return [this.r, this.g, this.b, this.alpha];
    }

    static size() {
        return 4;
    }
}
```

Файл position.js:

```
/**
 * Позиция.
 * Является оберткой над Vertex.
 */
class Position {
    constructor (x, y, z) {
        this.x = x;
        this.y = y;
        this.z = z;
    }

    getX() {
        return this.x;
    }

    setX(value) {
        this.x = value;
    }

    getY() {
        return this.y;
    }

    setY(value) {
```

```
        this.y = value;
    }

    getZ() {
        return this.z;
    }

    setZ(value) {
        this.z = value;
    }

    static size() {
        return 3;
    }

    getCoords() {
        return [this.getX(), this.getY(), this.getZ()];
    }

    getReversedCoords() {
        return [-1 * this.vertex.getX(), -1 * this.vertex.getY(), -1 *
this.vertex.getZ()];
    }
}
```

Файл vertex.js:

```
/**
 * Вершина.
 */
class Vertex {
    constructor (x, y, z, color) {
        this.position = new Position(x, y, z);
        this.color = color;
    }

    getCoords() {
        return this.position.getCoords();
    }

    getColorCode() {
        return this.color.getColorCode();
    }

    static size() {
        return 3;
    }
```

```
}
```

Файл figure.js:

```
/**
 * Фигура - базовый примитив.
 */
class Figure {
    /**
     * Конструктор класса.
     * @param {Position} positionFromZero - позиция фигуры относительно нуля.
     * @param {Array<Vertex>} vertices - массив координат вершин.
     * @param {Color} color - цвет вершины.
     */
    constructor(positionFromZero, vertices, color) {
        this.positionFromZero = positionFromZero;
        this.vertices = vertices;
        this.vertexBuffer = undefined;
        this.colorBuffer = undefined;
    }

    /**
     * Возвращает координаты фигуры относительно нуля.
     * @returns Position.
     */
    getPositionFromZero() {
        return this.positionFromZero;
    }

    /**
     * Возвращает вершины фигуры.
     */
    getVertices() {
        return this.vertices;
    }

    /**
     * Возвращает вершинный буфер фигуры - место в памяти видеокарты.
     */
    getVertexBuffer() {
        return this.vertexBuffer;
    }

    getColorBuffer() {
        return this.colorBuffer;
    }
```

```
    /**
     * Возвращает количество вершин фигуры.
     */
    getVerticesCount() {
        return this.vertices.length;
    }

    /**
     * Рисует фигуру на холсте, который использует драйвер.
     */
    draw() {
        this.vertexBuffer = WEBGL_DRIVER.initBuffer(this.vertices,
BUFFER_TYPE.vertex);
        this.colorBuffer = WEBGL_DRIVER.initBuffer(this.vertices,
BUFFER_TYPE.color);
        WEBGL_DRIVER.drawFigure(this);
    }

    /**
     * Возвращает вершины фигуры в виде одномерного массива.
     */
    static joinVerticesPositions(vertices) {
        let result = [];

        for(let vertex of vertices) {
            result.push(...vertex.getCoords());
        }

        return result;
    }

    static joinVerticesColors(vertices) {
        let result = [];

        for(let vertex of vertices) {
            result.push(...vertex.getColorCode());
        }

        return result;
    }
}
```

Файл webgl-driver.js:

```
// Контекст WebGL.
let gl;
```

```
// Шейдеры.
let shaderProgram;

// Модельно-видовая матрица.
let mvMatrix = mat4.create();

// Проекционная матрица.
let pMatrix = mat4.create();

// Временный буфер.
let tmpBuffer = undefined;

let BUFFER_TYPE = {vertex : 0, color: 1};

/**
 * Драйвер WebGL.
 */
let WEBGL_DRIVER = {
    // public

    /**
     * Инициализирует драйвер.
     * @param {string} canvas_id - идентификатор canvas на странице.
     */
    init: function(canvas_id) {
        let canvas = document.getElementById(canvas_id);
        WEBGL_DRIVER._initContext(canvas);
        WEBGL_DRIVER._initShaders();

        this.resetScene();
    },

    resetScene() {
        gl.clearColor(0.0, 0.0, 0.0, 1.0);
        gl.enable(gl.DEPTH_TEST);

        gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
        gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
        mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0,
pMatrix);
        mat4.identity(mvMatrix);
    },

    /**
     * Инициализирует вершинный буфер.
     * @param {Array<any>} vertices - массив вершин.
     */
    initBuffer: function(vertices, type) {
        let buffer = this._createEmptyBuffer();
```

```
        this._setCurrentArrayBuffer(buffer);

        switch(type) {
            case BUFFER_TYPE.vertex:
                this._fillVertexBuffer(vertices);
                break;
            case BUFFER_TYPE.color:
                this._fillColorBuffer(vertices);
                break;
        }

        this._resetCurrentBuffer();

        return buffer;
    },

    /**
     * Рисует фигуру на холсте.
     * @param {Figure} figure - фигура.
     */
    drawFigure: function(figure) {
        this._setCurrentPosition(figure.getPositionFromZero());
        this._setCurrentArrayBuffer(figure.getVertexBuffer());
        gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
Position.size(), gl.FLOAT, false, 0, 0);

        this._setCurrentArrayBuffer(figure.getColorBuffer());
        gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
Color.size(), gl.FLOAT, false, 0, 0);

        this._setMatrixUniforms();
        gl.drawArrays(gl.TRIANGLE_STRIP, 0, figure.getVerticesCount());

        this._resetCurrentBuffer();
        this._setCurrentPositionToZero();
    },

    //private

    /**
     * Инициализирует контекст WebGL.
     * @param {string} canvas - идентификатор canvas на странице.
     */
    _initContext: function(canvas) {
        try {
            gl = canvas.getContext("experimental-webgl");
            gl.viewportWidth = canvas.width;
            gl.viewportHeight = canvas.height;
```

```
        } catch (e) {
        }
        if (!gl) {
            alert("Could not initialise WebGL, sorry :-(");
        }
    },

    /**
     * Инициализирует шейдеры.
     */
    _initShaders: function() {
        let fragmentShader = WEBGL_DRIVER._getShader(gl, "shader-fs");
        let vertexShader = WEBGL_DRIVER._getShader(gl, "shader-vs");

        shaderProgram = gl.createProgram();
        gl.attachShader(shaderProgram, vertexShader);
        gl.attachShader(shaderProgram, fragmentShader);
        gl.linkProgram(shaderProgram);

        if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {
            alert("Could not initialise shaders");
        }

        gl.useProgram(shaderProgram);

        shaderProgram.vertexPositionAttribute =
gl.getAttribLocation(shaderProgram, "aVertexPosition");
        gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

        shaderProgram.vertexColorAttribute =
gl.getAttribLocation(shaderProgram, "aVertexColor");
        gl.enableVertexAttribArray(shaderProgram.vertexColorAttribute);

        shaderProgram.pMatrixUniform = gl.getUniformLocation(shaderProgram,
"uPMatrix");
        shaderProgram.mvMatrixUniform = gl.getUniformLocation(shaderProgram,
"uMVMatrix");
    },

    /**
     * Инициализирует шейдер.
     * @param gl - контекст WebGL.
     * @param id - идентификатор тэга script, содержащего шейдер.
     * @returns объект шейдера.
     */
    _getShader: function(gl, id) {
        let shaderScript = document.getElementById(id);
        if (!shaderScript) {
            return null;
```

```
        }

        let str = "";
        let k = shaderScript.firstChild;
        while (k) {
            if (k.nodeType == 3) {
                str += k.textContent;
            }
            k = k.nextSibling;
        }

        let shader;
        if (shaderScript.type == "x-shader/x-fragment") {
            shader = gl.createShader(gl.FRAGMENT_SHADER);
        } else if (shaderScript.type == "x-shader/x-vertex") {
            shader = gl.createShader(gl.VERTEX_SHADER);
        } else {
            return null;
        }

        gl.shaderSource(shader, str);
        gl.compileShader(shader);

        if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
            alert(gl.getShaderInfoLog(shader));
            return null;
        }

        return shader;
    },

    /**
     * Создает пустой буфер.
     */
    _createEmptyBuffer: function() {
        return gl.createBuffer();
    },

    /**
     * Устанавливает буфер в качестве активного.
     * @param {WebGLBuffer} buffer - буфер.
     */
    _setCurrentArrayBuffer: function(buffer) {
        gl.bindBuffer(gl.ARRAY_BUFFER, buffer);
    },

    /**
     * Заполняет буфер данными о вершинах.
     * @param {Array<Vertex>} vertices - вершины.
```

```
     */
    _fillVertexBuffer: function(vertices) {
        gl.bufferData(gl.ARRAY_BUFFER, new
Float32Array(Figure.joinVerticesPositions(vertices)), gl.STATIC_DRAW);
    },

    /**
     * Заполняет буфер данными о цвете.
     * @param {Array<Color>} vertices - вершины.
     */
    _fillColorBuffer: function(vertices) {
        gl.bufferData(gl.ARRAY_BUFFER, new
Float32Array(Figure.joinVerticesColors(vertices)), gl.STATIC_DRAW);
    },

    /**
     * Сбрасывает текущий буфер.
     * Устанавливается временный буфер.
     */
    _resetCurrentBuffer: function() {
        if(tmpBuffer === undefined) {
            tmpBuffer = this._createEmptyBuffer();
        }
        this._setCurrentArrayBuffer(tmpBuffer);
    },

    /**
     * Устанавливает позицию для рисования.
     * @param {Position} positionFromZero - позиция относительно нуля.
     */
    _setCurrentPosition(positionFromZero) {
        this._setCurrentPositionToZero();
        mat4.translate(mvMatrix, positionFromZero.getCoords());
    },

    /**
     * Перемещает позицию для рисования в нулевые координаты.
     */
    _setCurrentPositionToZero() {
        mat4.identity(mvMatrix);
    },

    /**
     * Инициализирует поля программы шейдеров (uniform-переменные) объектами
JS,
     * соответствующими матрице проекции и матрице модели.
     */
    _setMatrixUniforms: function () {
        gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
```

```
        gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);
    }
}
```

Файл shapes_repository.js:

```
/**
 * Репозиторий для всех объектов на сцене.
 */
let SHAPES_REPOSITORY = {
    data: [],

    init: function() {
        SHAPES_REPOSITORY.data = [
            new Figure(
                new Position(-1.5, 1.0, -7.0),
                [
                    new Vertex(0.0,  1.0,  0.0, new Color(1.0, 0.0, 0.0,
1.0)),
                    new Vertex(-1.0, -1.0,  0.0, new Color(0.0, 1.0, 0.0,
1.0)),
                    new Vertex(1.0, -1.0,  0.0, new Color(0.0, 0.0, 1.0, 1.0))
                ]
            ),
            new Figure(
                new Position(1.5, 0.0, -5.0),
                [
                    new Vertex(1.0,  1.0, 0.0, new Color(1.0, 0.0, 1.0, 1.0)),
                    new Vertex(-1.0,  1.0, 0.0, new Color(0.0, 1.0, 1.0,
1.0)),
                    new Vertex(1.0, -1.0, 0.0, new Color(0.0, 1.0, 0.0, 1.0)),
                    new Vertex(-1.0, -1.0, 0.0, new Color(0.0, 1.0, 1.0, 1.0))
                ]
            ),
            new Figure(
                new Position(0.0, -3.0, -14.0),
                [
                    new Vertex(1.1,  1.1,  -1.1, new Color(1.0, 1.0, 0.0,
1.0)),
                    new Vertex(-1.1,  1.1,  -1.1, new Color(1.0, 1.0, 1.0,
1.0)),
                    new Vertex(1.1, -1.1,  -1.1, new Color(1.0, 1.0, 0.0,
1.0)),
                    new Vertex(-1.1, -1.1,  -1.1, new Color(1.0, 1.0, 1.0,
1.0))
                ]
            )
```

```
        ];
    }
}
```

Файл app.js:

```
function init(canvas_node_id) {
    WEBGL_DRIVER.init(canvas_node_id);
    SHAPES_REPOSITORY.init();
}

function update() {
    WEBGL_DRIVER.resetScene();
    let container = document.getElementById('vertexColors');
    container.innerHTML = '';

    SHAPES_REPOSITORY.data.forEach((figure) => {
        figure.draw();
    });

    setupUI();
}

function main(canvas_node_id) {
    init(canvas_node_id);
    update();
}

function setupUI() {
    let container = document.getElementById('vertexColors');

    createVertexColorNode = function(figure, figureIndex) {
        let vertexIndex = 0;
        figure.getVertices().forEach((vertex) => {
            let wrapper = document.createElement('div');
                let input = document.createElement('input');
                input.setAttribute('id',
`vertex_${figureIndex}_${vertexIndex}`);
                input.value = String(vertex.getColorCode());
                input.addEventListener('keydown', function(e) {
                    if (e.keyCode === 13) {
                        colorChanged(e.target.id, this.value);
                    }
                });
                wrapper.appendChild(input);
            container.appendChild(wrapper);
            vertexIndex++;
```

```
        });
    }

    let i = 0;
    SHAPES_REPOSITORY.data.forEach((figure) => {
        createVertexColorNode(figure, i);
        i++;
    });
}

function colorChanged(node_id, value) {
    let meta = node_id.split('_');

    let figureIndex = meta[1];
    let vertexIndex = meta[2];

    meta = value.split(',');
    let r = meta[0];
    let g = meta[1];
    let b = meta[2];
    let a = meta[3];
    SHAPES_REPOSITORY.data[figureIndex].vertices[vertexIndex].color = new
Color(r, g, b, a);

    update();
}
```