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

Отчет
по лабораторной работе №1
по дисциплине «КОМПЬЮТЕРНАЯ 3D-ГРАФИКА»
Тема: Рисование геометрических объектов.

Студентка гр. 5303 _____ Допира В.Е.

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.

Задания

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 сведения

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:

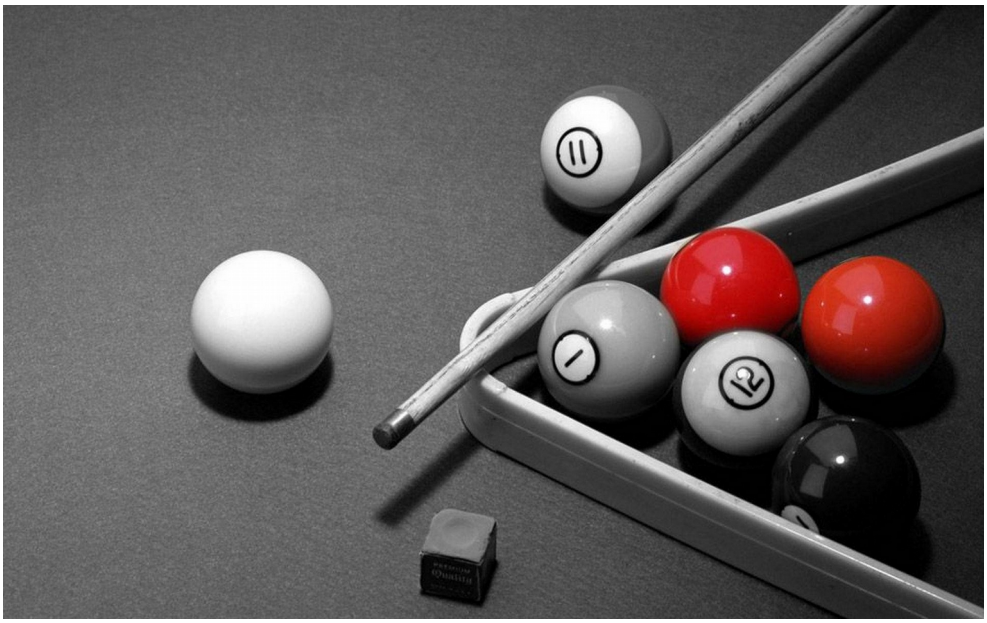

Рисунок 1 — Финальная сцена

Шаг 1. Создан холст(canvas) HTML-5, получен идентификатор холста lab1 и контекст рендеринга WebGL

```
<body onload="webGLStart();">
    <p> Lab 1</p>
    <canvas    id="lab1"    style="border:    none;"    width="700"
height="500"></canvas>
</body>


function initGL(canvas) {
        try {
            gl = canvas.getContext("webgl");
            gl.viewportWidth = canvas.width;
            gl.viewportHeight = canvas.height;
        } catch (e) {
        }
        if (!gl) {
            alert("Could not initialise WebGL, sorry :-(");
        }
    }


function webGLStart() {
        var canvas = document.getElementById("lab1");
        initGL(canvas);
    }
```

Шаг 2. Созданы буферы объектов.

Выделены 3 объекта:

1. Треугольник
2. Окружность
3. Квадрат

Например, для квадрата буфферы:

```
    var squareVertexPositionBuffer;
    squareVertexPositionBuffer = gl.createBuffer();
```

Шаг 3. Создана шейдерная программа

```
    var shaderProgram;
    function initShaders() {
        var fragmentShader = getShader(gl, "shader-fs");
        var vertexShader = 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.vertexPositionAttr
ibute);

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

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

Шаг 4 – Связаны шейдеры и буферы объектов. Например, для квадрата операции будут происходить над следующим (текущим) буфером:

```
gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
```

Координаты:

```
vertices = [
     1.0,  1.0,  0.0,
    -1.0,  1.0,  0.0,
     1.0, -1.0,  0.0,
    -1.0, -1.0,  0.0
];
```

Создан объект Float32Arrays из массива координат, и он используется для заполнения текущего буффера:

```
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);
```

Для объекта устанавливаются два свойства: numItems — количество отдельных координат вершин, каждая из которых состоит из трех чисел (itemSize).

```
squareVertexPositionBuffer.itemSize = 3;
squareVertexPositionBuffer.numItems = 4
```

Шаг 5 – Рисование сцены и объектов. Они раскрашены приближенно к некоторым объектам итогового изображения. Например, для квадрата:

```
gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
squareVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexColorBuffer);
gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
squareVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);

setMatrixUniforms();
gl.drawArrays(gl.TRIANGLE_STRIP, 0,
squareVertexPositionBuffer.numItems);
```

Для треугольника использован примитив gl.TRIANGLES, для квадрата — gl.TRIANGLE_STRIP, для окружности — gl.TRIANGLE_FAN.

Цвета объектов заданы в формате RGBA.

Результат выполнения программы показан на рисунке 2.

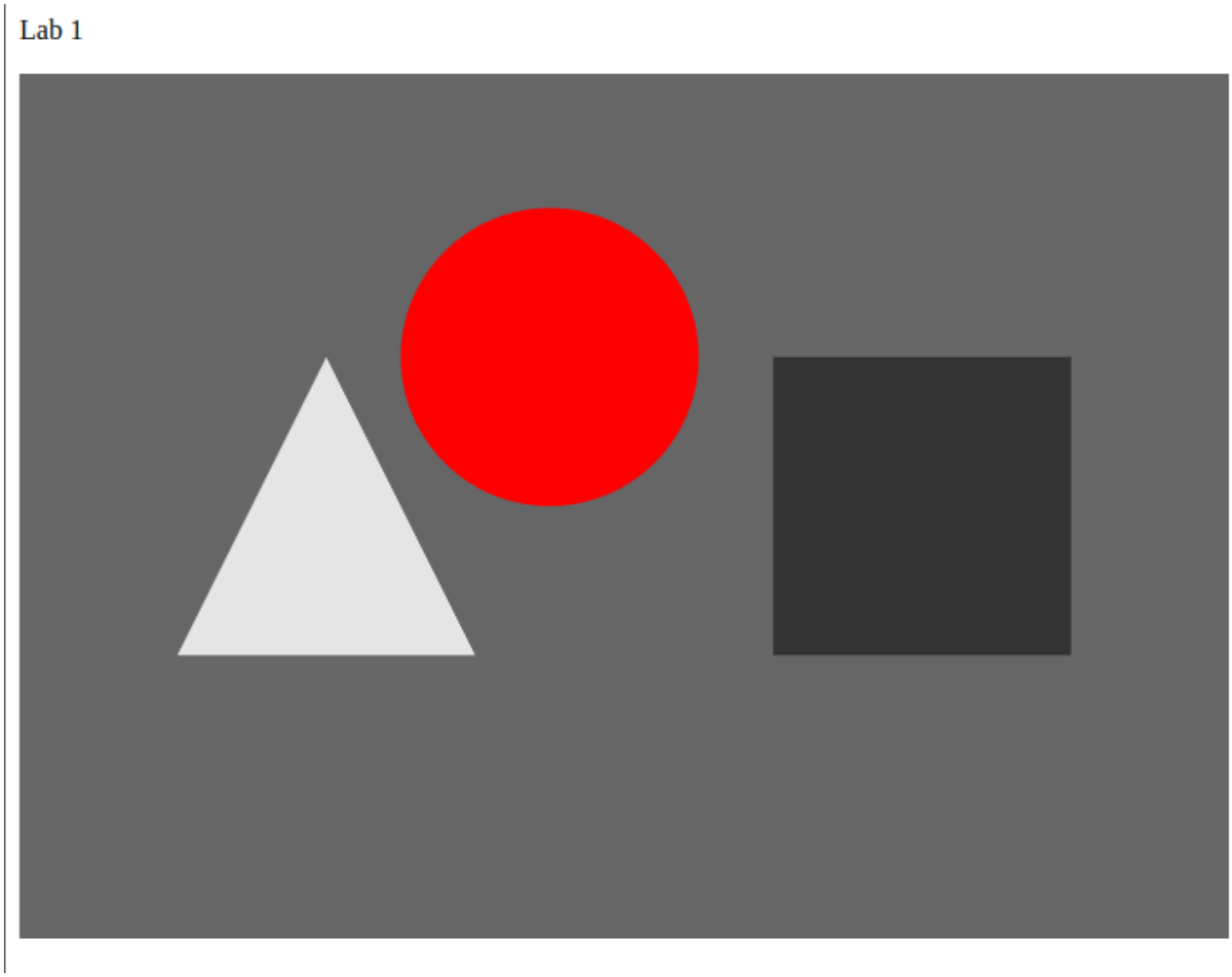

Рисунок 2 — Результат работы программы

Вывод

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.

Приложение 1

Листинг программы

```
<html>

<head>
<title>Lab 1</title>
<meta http-equiv="content-type" content="text/html; charset=ISO-
8859-1">
<script type="text/javascript" src="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; //модель-вид матрица
    uniform mat4 uPMatrix; //проекционная матрица

    varying vec4 vColor;

    void main(void) {
         gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition,
1.0);
        vColor = aVertexColor;
    }
</script>


<script type="text/javascript">

    var gl;
    function initGL(canvas) {
        try {
            gl = canvas.getContext("webgl");
            gl.viewportWidth = canvas.width;
            gl.viewportHeight = canvas.height;
        } catch (e) {
        }
        if (!gl) {
            alert("Could not initialise WebGL, sorry :-(");
```

```
        }
    }
    function getShader(gl, id) {
        var shaderScript = document.getElementById(id);
        if (!shaderScript) {
            return null;
        }

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

        var 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;
    }

    var shaderProgram;
    function initShaders() {
        var fragmentShader = getShader(gl, "shader-fs");
        var vertexShader = 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.vertexPositionAtt
ribute);

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

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


    var mvMatrix = mat4.create();
    var pMatrix = mat4.create();
    function setMatrixUniforms() {
          gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false,
pMatrix);
         gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false,
mvMatrix);
    }


    var triangleVertexPositionBuffer;
    var triangleVertexColorBuffer;
    var squareVertexPositionBuffer;
    var squareVertexColorBuffer;
    var circleVertexPositionBuffer;
    var circleVertexColorBuffer;

    function initBuffers() {
        triangleVertexPositionBuffer = gl.createBuffer();
                                    gl.bindBuffer(gl.ARRAY_BUFFER,
triangleVertexPositionBuffer);
        var vertices = [
             0.0,  1.0,  0.0,
            -1.0, -1.0,  0.0,
             1.0, -1.0,  0.0
        ];
         gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
gl.STATIC_DRAW);
               triangleVertexPositionBuffer.itemSize  =  3;  //each
coordinate consists of three numbers
```

```
            triangleVertexPositionBuffer.numItems = 3; //three
separate vertex coordinates

        triangleVertexColorBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexColorBuffer);
        var colors = [];
        for (var i=0; i < 3; i++) {
            colors = colors.concat([0.9, 0.9, 0.9, 1.0]);
        }
          gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors),
gl.STATIC_DRAW);
        triangleVertexColorBuffer.itemSize = 4;
        triangleVertexColorBuffer.numItems = 3;


        squareVertexPositionBuffer = gl.createBuffer();
                                     gl.bindBuffer(gl.ARRAY_BUFFER,
squareVertexPositionBuffer);
        vertices = [
             1.0,  1.0,  0.0,
            -1.0,  1.0,  0.0,
             1.0, -1.0,  0.0,
            -1.0, -1.0,  0.0
        ];
         gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
gl.STATIC_DRAW);
        squareVertexPositionBuffer.itemSize = 3;
        squareVertexPositionBuffer.numItems = 4;

        squareVertexColorBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexColorBuffer);
        colors = [];
        for (var i=0; i < 4; i++) {
            colors = colors.concat([0.2, 0.2, 0.2, 1.0]);
        }
          gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors),
gl.STATIC_DRAW);
        squareVertexColorBuffer.itemSize = 4;
        squareVertexColorBuffer.numItems = 4;

        circleVertexPositionBuffer = gl.createBuffer();
                                     gl.bindBuffer(gl.ARRAY_BUFFER,
circleVertexPositionBuffer);
        vertices = [0.0,  0.0];
        for (i = 0; i <= 100; i++){
            vertices.push(Math.cos(i*2*Math.PI/100))
            vertices.push(Math.sin(i*2*Math.PI/100));
        }
         gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
gl.STATIC_DRAW);
```

```
        circleVertexPositionBuffer.itemSize = 2;
        circleVertexPositionBuffer.numItems = 102;

        circleVertexColorBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ARRAY_BUFFER, circleVertexColorBuffer);
        colors = [];
        for (var i=0; i < 102; i++) {
            colors = colors.concat([1.0, 0.0, 0.0, 1.0]);
        }
          gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors),
gl.STATIC_DRAW);
        circleVertexColorBuffer.itemSize = 4;
        circleVertexColorBuffer.numItems = 102;
    }


    function drawScene() {
        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);

        mat4.translate(mvMatrix, [-2, 0.0, -7.0]);
                                        gl.bindBuffer(gl.ARRAY_BUFFER,
triangleVertexPositionBuffer);
         gl.vertexAttribPointer(shaderProgram.vertexPositionAttribu
te, triangleVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
        gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexColorBuffer);
         gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
triangleVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);
        setMatrixUniforms();
                                    gl.drawArrays(gl.TRIANGLES,    0,
triangleVertexPositionBuffer.numItems);

        mat4.translate(mvMatrix, [4.0, 0.0, 0.0]);
                                        gl.bindBuffer(gl.ARRAY_BUFFER,
squareVertexPositionBuffer);
         gl.vertexAttribPointer(shaderProgram.vertexPositionAttribu
te, squareVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
        gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexColorBuffer);
         gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
squareVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);
        setMatrixUniforms();
                                gl.drawArrays(gl.TRIANGLE_STRIP,    0,
squareVertexPositionBuffer.numItems);

        mat4.translate(mvMatrix, [-2.5, 1.0, 0.0]);
                                        gl.bindBuffer(gl.ARRAY_BUFFER,
circleVertexPositionBuffer);
```

```
         gl.vertexAttribPointer(shaderProgram.vertexPositionAttribu
te, circleVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
        gl.bindBuffer(gl.ARRAY_BUFFER, circleVertexColorBuffer);
         gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
circleVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);
        setMatrixUniforms();
                                gl.drawArrays(gl.TRIANGLE_FAN,    0,
circleVertexPositionBuffer.numItems);
    }


    function webGLStart() {
        var canvas = document.getElementById("lab1");
        initGL(canvas);
        initShaders();
        initBuffers();

        gl.clearColor(0.4, 0.4, 0.4, 1.0);
        gl.enable(gl.DEPTH_TEST);

        drawScene();
    }

</script>


</head>


<body onload="webGLStart();">
    <p> Lab 1</p>

          <canvas   id="lab1"   style="border:   none;"   width="700"
height="500"></canvas>
</body>

</html>
```