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. 5381 _____ Буздина М.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:

1. рисование нескольких вещей с отдельными командами рисования
2. используя разные примитивы
3. изменение размеров линий и точек по умолчанию
4. 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.

Результатом работы данного курса представлен на рис.1:

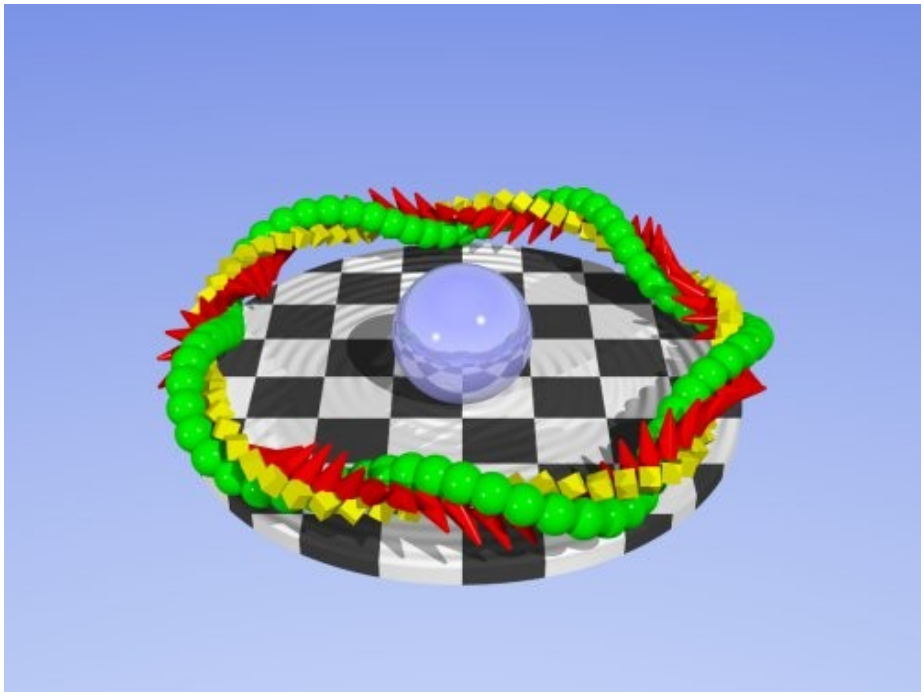

Рисунок 1 — Финальная сцена

На данном этапе было решено определить 3 объекта:

1. Окружность
2. Треугольник
3. Прямоугольник (квадрат)

Эти объекты имеют следующие свойства:

1. Окружность (координаты центра, радиус, цвет)
2. Треугольник (координаты вершин (3 шт), цвет)
3. Прямоугольник (координаты центра, ширина, высота, цвет)

Для отрисовки окружности использовался примитив TRIANGLE_FAN, для квадрата использовался примитив TRIANGLE_STRIPH, для треугольник использовался примитив TRIANGLE. Рисунке 2 представлен результат работы программы — его стартовое состояние.

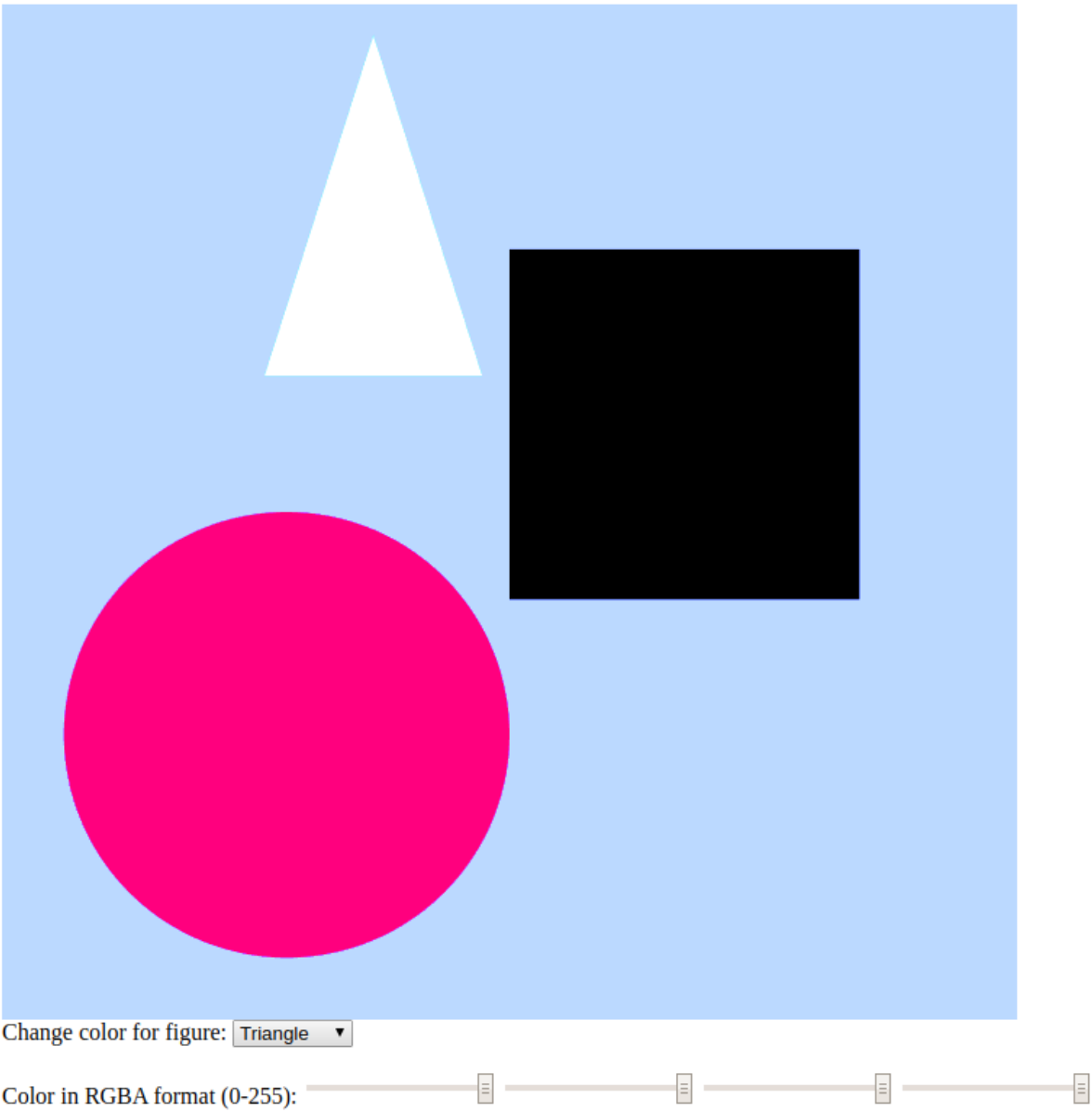

Рисунок 2 — Результат работы программы.

В данной программе обеспечено динамическое изменение цвета. Есть возможность поменять цвет для любой выбранной фигуры. Пример представлен на рисунке 3.

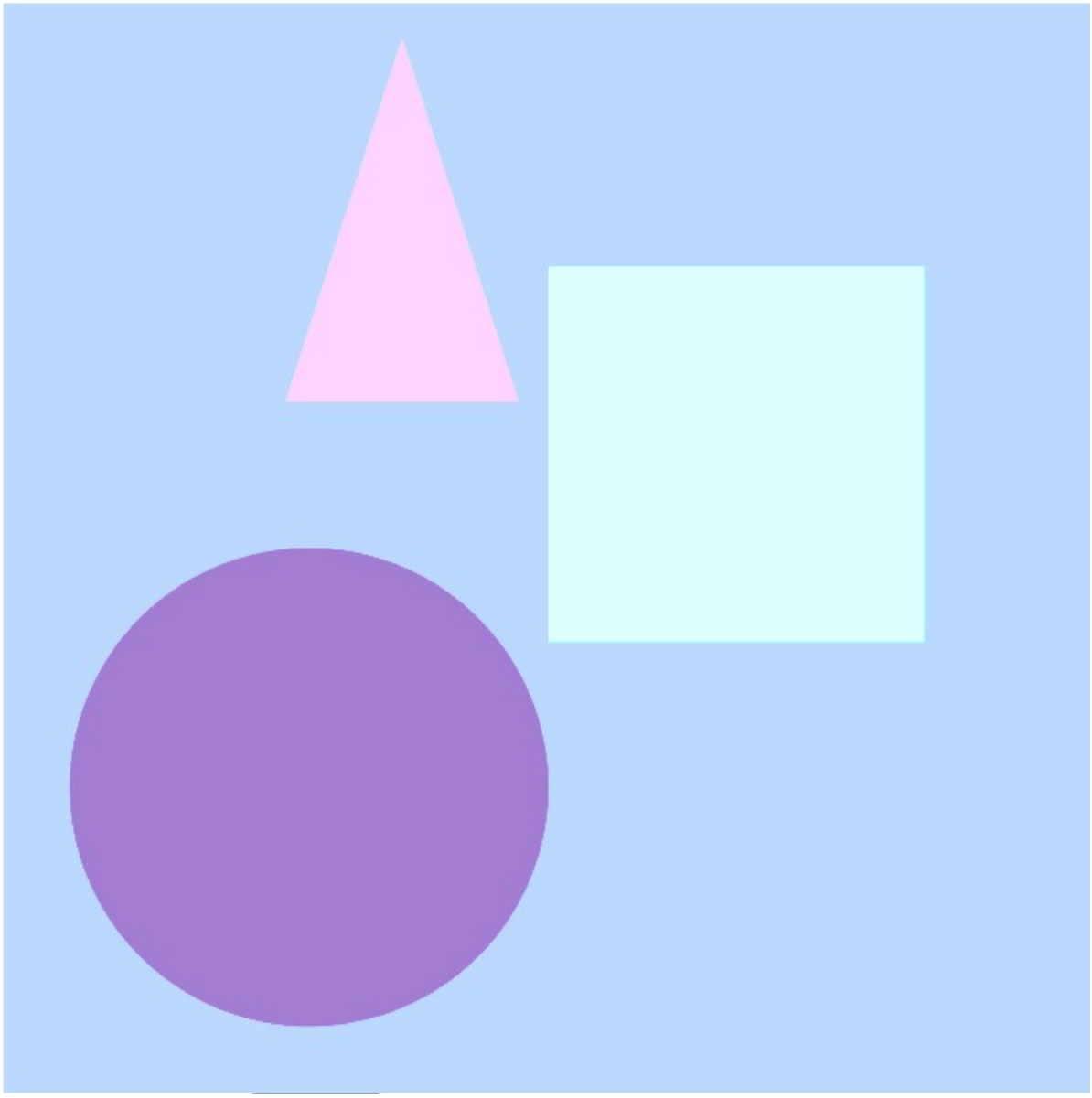

Рисунок 3 — Результат работы программы. Изменение цвета сетки

Выводы.

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

ПРИЛОЖЕНИЕ А
ЛИСТИНГ ПРОГРАММЫ

main.html

```
<html xmlns="http://www.w3.org/1999/html">

<head>
   <title>Great 3D scene</title>
   <meta content="text/html" http-equiv="content-type">
   <script src="sources/glMatrix-0.9.5.min.js" type="text/javascript"></script>
   <script src="utilities.js" type="text/javascript"></script>
   <script src="Figures/Figure.js" type="text/javascript"></script>
   <script src="Figures/Circle.js" type="text/javascript"></script>
   <script src="Figures/Triangle.js" type="text/javascript"></script>
   <script src="Figures/Rectangle.js" type="text/javascript"></script>
   <script src="webGL.js" type="text/javascript"></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>
</head>
<body onload="webGLStart();">
<div>
   <canvas height="700" id="central_canvas" style="border: none;" width="700"></canvas>
</div>
<div>
   <form>
      <label>
         Change color for figure:
         <select id="cur_figure" onchange="(this.value)">
            <option value="Triangle">Triangle</option>
            <option value="Circle">Circle</option>
            <option value="Rectangle">Rectangle</option>
         </select>
      </label>
   </form>

   <form>
      Color in RGBA format (0-255):
      <label>
```

```
            <input type="range" name="color" min="0" max="255" step="1" value="255" onchange="update_scene()">
            <input type="range" name="color" min="0" max="255" step="1" value="255" onchange="update_scene()">
            <input type="range" name="color" min="0" max="255" step="1" value="255" onchange="update_scene()">
            <input type="range" name="color" min="0" max="255" step="1" value="255" onchange="update_scene()">
        </label>
    </form>
<!--    <button onclick="update_scene()">Let's go</button>-->
</div>
</body>
</html>
```

webGL.js

```
let gl;
let figures = [];
let shaderProgram;
const mvMatrix = mat4.create();
const pMatrix = mat4.create();

function webGLStart() {
    const canvas = document.getElementById("central_canvas");
    initGL(canvas);
    initShaders();
    figures = [new Rectangle(new Point3(1.0, 0.0, 0.0), 2, 2, [0.0, 0.0, 0.0, 1.0]),
        new Net(new Point3(0.0, 0.0, -1.0), 4, 4, 20, [1.0, 1.0, 1.0, 1.0]),
        new Circle(new Point3(-1.0, -1.0, -2.0), 3, [1.0, 0.0, 0.5, 1.0])
    ];
    initBuffers();
    gl.clearColor(0.0, 0.0, 0.0, 0.5);
    gl.lineWidth(3);
    gl.enable(gl.DEPTH_TEST);

    drawScene();
}

function initGL(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 :-(");
    }
}


function getShader(gl, id) {
    const shaderScript = document.getElementById(id);
    if (!shaderScript) {
        alert('No shader programme');
        return null;
    }

    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, shaderScript.text);
    gl.compileShader(shader);

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

    return shader;
}


function initShaders() {
    const fragmentShader = getShader(gl, "shader-fs");
    const 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);let gl;
let figures = [];
let shaderProgram;
const mvMatrix = mat4.create();
const pMatrix = mat4.create();

function webGLStart() {
    const canvas = document.getElementById("central_canvas");
    initGL(canvas);
    initShaders();
    figures = [new Rectangle(new Point3(1.0, 0.5, 0.0), 2, 2, [0.0, 0.0, 0.0, 1.0]),
        new Triangle(new Point3(-1.0, 3.5, -2.0), new Point3(-1.8, 1.0, -2.0), new Point3(-0.2, 1.0, -2.0), [1.0, 1.0, 1.0,
1.0]),
        new Circle(new Point3(-2.0, -2.0, -4.0), 2, [1.0, 0.0, 0.5, 1.0])
    ];
    initBuffers();
    gl.clearColor(0.0, 0.5, 1.0, 0.5);
    gl.lineWidth(3);
    gl.enable(gl.DEPTH_TEST);

    drawScene();
}

function initGL(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 :-(");
    }
}
```

```
function getShader(gl, id) {
   const shaderScript = document.getElementById(id);
   if (!shaderScript) {
      alert('No shader programme');
      return null;
   }

   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, shaderScript.text);
   gl.compileShader(shader);

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

   return shader;
}


function initShaders() {
   const fragmentShader = getShader(gl, "shader-fs");
   const 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.vertexPositionAttribute);

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

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

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

function initBuffers() {
   for (let i = 0; i < figures.length; i++) {
```

```
        figures[i].initBuffers();
    }
}

function drawScene() {
    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);

    //Draw Rectangle
    mat4.translate(mvMatrix, [0.0, 0.0, -7.0]);
    setBuffersToShaders(figures[0].getPositionBuffer(), figures[0].getColorBuffer());
    gl.drawArrays(gl.TRIANGLE_STRIP, 0, figures[0].getPositionBuffer().numItems);

    //Draw Net
    setBuffersToShaders(figures[1].getPositionBuffer(), figures[1].getColorBuffer());
    gl.drawArrays(gl.TRIANGLES, 0, figures[1].getPositionBuffer().numItems);

    //Draw Circle
    setBuffersToShaders(figures[2].getPositionBuffer(), figures[2].getColorBuffer());
    gl.drawArrays(gl.TRIANGLE_FAN, 0, figures[2].getPositionBuffer().numItems);

}

function setBuffersToShaders(pos_buffer, color_buffer) {
    gl.bindBuffer(gl.ARRAY_BUFFER, pos_buffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute, pos_buffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, color_buffer);
    gl.vertexAttribPointer(shaderProgram.vertexColorAttribute, color_buffer.itemSize, gl.FLOAT, false, 0, 0);
    setMatrixUniforms();
}

function update_scene() {
    let cur_figure = document.getElementById("cur_figure").value;
    let color_list = document.getElementsByName("color");
    let color = [];
    for (let i = 0; i < color_list.length; i++) {
        color.push(color_list[i].value / 255)
    }

    for (let i = 0; i < figures.length; i++) {
        if (cur_figure == figures[i].name) {
            figures[i].color = color;
            figures[i].initBuffers();
        }
    }
    drawScene();
}
```

Figure.js

```
class Figure {
    constructor(center) {
        this.vertexPositionBuffer = gl.createBuffer();
        this.vertexColorBuffer = gl.createBuffer();
        this.vertices = [];
        this.colors = [];
```

```
        this.center = center;
    }

    initPositionBuffer(){
        gl.bindBuffer(gl.ARRAY_BUFFER, this.vertexPositionBuffer);
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(this.vertices), gl.STATIC_DRAW);
        this.vertexPositionBuffer.itemSize = 3;
        this.vertexPositionBuffer.numItems = this.vertices.length / 3;
    }

    initColorBuffer(){
        gl.bindBuffer(gl.ARRAY_BUFFER, this.vertexColorBuffer);
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(this.colors), gl.STATIC_DRAW);
        this.vertexColorBuffer.itemSize = 4;
        this.vertexColorBuffer.numItems = this.colors.length / 4;
    }

    getPositionBuffer(){
        return this.vertexPositionBuffer;
    }

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

Circle.js

```
class Circle extends Figure {
    constructor(center, radius, color) {
        super(center);
        this.name = "Circle";
        this.radius = radius;
        this.color = color;
    }
    initBuffers(){
        this.generateVerticesMatrix();
        this.generateColorMatrix();
        this.initPositionBuffer();
        this.initColorBuffer();
    }

    generateColorMatrix(){
        let colors = [];
        for (let i=0; i < this.vertices.length / 3; i++) {
            colors = colors.concat(this.color);
        }
        this.colors = colors;
    }

    generateVerticesMatrix(){
        let vertices = this.center.toArray();
        for(let theta = 0; theta < 2 * Math.PI; theta += 0.01) {
            let x1  = this.center.x + this.radius * Math.cos(theta);
            let y1 = this.center.y + this.radius * Math.sin(theta);
            let x2  = this.center.x + this.radius * Math.cos(theta + 0.01);
            let y2 = this.center.y + this.radius * Math.sin(theta + 0.01);
            vertices.push(x1, y1, this.center.z, x2, y2, this.center.z)
        }
        vertices.concat(this.center.toArray());
        this.vertices = vertices;
    }
}
```

Rectangle.js

```
class Rectangle extends Figure {
    constructor(center, a, b, color) {
        super(center);
        this.name = "Rectangle";
        this.aSide = a;
        this.bSide = b;
        this.color = color;
    }

    initBuffers(){
        this.generateVerticesMatrix();
        this.generateColorMatrix();
        this.initPositionBuffer();
        this.initColorBuffer();
    }

    // Special color like on picture
    generateColorMatrix(){
        let colors = [];

        for (let i=0; i < this.vertices.length/3 ; i++) {
            colors = colors.concat(this.color);
        }
        // colors = colors.concat([1.0, 1.0, 1.0, 1.0]);
        this.colors = colors;
    }

    generateVerticesMatrix(){
        let vertices = this.center.toArray();
        vertices = vertices.concat([
            this.center.x + this.aSide / 2, this.center.y + this.bSide / 2, this.center.z,
            this.center.x + this.aSide / 2, this.center.y - this.bSide / 2, this.center.z,
            this.center.x - this.aSide / 2, this.center.y - this.bSide / 2, this.center.z,
            this.center.x - this.aSide / 2, this.center.y + this.bSide / 2, this.center.z,
            this.center.x + this.aSide / 2, this.center.y + this.bSide / 2, this.center.z
        ]);
        vertices = vertices.concat(this.center.toArray());
        this.vertices = vertices;
    }
}
```

Triangle.js

```
class Triangle extends Figure {
    constructor(pointA, pointB, pointC,  color) {
        super(new Point3(0, 0, 0));
        this.name = "Triangle";
        this.pointA = pointA;
        this.pointB = pointB;
        this.pointC = pointC;
        this.color = color;
    }

    initBuffers(){
        this.generateVerticesMatrix();
        this.generateColorMatrix();
        this.initPositionBuffer();
        this.initColorBuffer();
    }
```

```
    generateColorMatrix(){
        let colors = [];
        for (let i=0; i < this.vertices.length / 3; i++) {
            colors = colors.concat(this.color);
        }
        this.colors = colors;
    }

    generateVerticesMatrix(){
        this.vertices = this.pointA.toArray().concat(this.pointB.toArray().concat(this.pointC.toArray()));
    }
}
```

utilites.js

```
class Point3 {
    constructor(x=0, y=0, z=0){
        this.x = x;
        this.y = y;
        this.z = z
    }

    toArray(){
        return [this.x, this.y, this.z];
    }
}
```