Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Владимирский государственный университет

имени Александра Григорьевича и Николая Григорьевича Столетовых»

(ВлГУ)

Кафедра информационных систем и программной инженерии

Лабораторная работа № 11

по дисциплине "Программирование графических приложений"

ТЕМА РАБОТЫ:

Освещение в WEBGL

Выполнил:

студент гр. ПРИм-124

Парахин К.В.

Принял:

Жигалов И.Е.

Владимир 2024 г.

Цель работы:

Изучение методов работы с освещением в WebGL

Выполнение работы:

Индивидуальный вариант 14

На основе приведенных примеров программ построить объект, аналогичный одному из построенных в результате выполнения лабораторной работы по теме 9, но без использования tree.js, и применить к нему модели отражения и затенения (табл. 1).



Используем указанные в примерах скрипты для создания моделей освещения:  
Модель отражения Ламберта и модель интерполяции Фонга  
  
В качестве предмета для построения можно использовать треугольную призму, использованную в лабораторной работе 9  
  
Листинг программы:  
  
<!doctype html>

<html lang="ru">

<head>

<title>Призма WebGL</title>

<meta charset="utf-8">

<script type="text/javascript" src="webgl-utils\_.js"></script>

<script type="text/javascript" src="gl-matrix-min.js"></script>

<script id="LambertPhongVS" type="x-shader/x-vertex">

    attribute vec3 aVertexPosition;

    attribute vec3 aVertexNormal;

    uniform mat4 uMVMatrix;

    uniform mat4 uPMatrix;

    uniform mat3 uNMatrix;

    varying vec3 vNormal;

    void main(void) {

        vNormal = normalize(uNMatrix \* aVertexNormal);

        gl\_Position = uPMatrix \* uMVMatrix \* vec4(aVertexPosition, 1.0);

    }

</script>

<script id="LambertPhongFS" type="x-shader/x-fragment">

    precision mediump float;

    uniform vec3 uColor;

    varying vec3 vNormal;

    const vec3 vLightDirection = vec3(2.0, -1.0, -1.0);

    void main(void) {

        vec3 L = normalize(vLightDirection);

        vec3 N = normalize(vNormal);

        float lambertComponent = max(dot(N, -L), 0.0);

        vec3 diffuseLight = uColor \* lambertComponent;

        gl\_FragColor = vec4(diffuseLight, 1.0);

    }

</script>

<script type="text/javascript">

    var gl;

    function initGL(canvas) {

        try {

            gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");

            gl.viewportWidth = canvas.width;

            gl.viewportHeight = canvas.height;

        } catch (e) {}

        if (!gl) {

            alert("WebGL не поддерживается.");

        }

    }

    function createProgram(vertexShaderId, fragmentShaderId) {

        var vertexShader = getShader(gl, vertexShaderId);

        var fragmentShader = getShader(gl, fragmentShaderId);

        var shaderProgram = gl.createProgram();

        gl.attachShader(shaderProgram, vertexShader);

        gl.attachShader(shaderProgram, fragmentShader);

        gl.linkProgram(shaderProgram);

        if (!gl.getProgramParameter(shaderProgram, gl.LINK\_STATUS)) {

            alert("Не удалось инициализировать шейдеры.");

        }

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

        gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

        shaderProgram.vertexNormalAttribute = gl.getAttribLocation(shaderProgram, "aVertexNormal");

        gl.enableVertexAttribArray(shaderProgram.vertexNormalAttribute);

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

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

        shaderProgram.nMatrixUniform = gl.getUniformLocation(shaderProgram, "uNMatrix");

        shaderProgram.colorUniform = gl.getUniformLocation(shaderProgram, "uColor");

        return shaderProgram;

    }

    var currentColor;

    var currentProgram;

    function initShaders() {

        currentProgram = createProgram("LambertPhongVS", "LambertPhongFS");

        gl.useProgram(currentProgram);

        currentColor = [0.0, 1.0, 0.0];

        gl.uniform3fv(currentProgram.colorUniform, currentColor);

    }

    var mvMatrix = mat4.create();

    var pMatrix = mat4.create();

    function setMatrixUniforms() {

        gl.uniformMatrix4fv(currentProgram.pMatrixUniform, false, pMatrix);

        gl.uniformMatrix4fv(currentProgram.mvMatrixUniform, false, mvMatrix);

        var normalMatrix = mat3.create();

        mat3.normalFromMat4(normalMatrix, mvMatrix);

        gl.uniformMatrix3fv(currentProgram.nMatrixUniform, false, normalMatrix);

    }

    var vertices = [

        -1.0, -1.0,  1.0,

         1.0, -1.0,  1.0,

         0.0, -1.0, -1.0,

        -1.0,  1.0,  1.0,

         1.0,  1.0,  1.0,

         0.0,  1.0, -1.0

    ];

    var vertexNormals = [

        0.0, -1.0, 0.0,

        0.0, -1.0, 0.0,

        0.0, -1.0, 0.0,

        0.0,  1.0, 0.0,

        0.0,  1.0, 0.0,

        0.0,  1.0, 0.0,

    ];

    var indices = [

        0, 1, 2,

        3, 4, 5,

        0, 1, 4,

        0, 4, 3,

        1, 2, 5,

        1, 5, 4,

        2, 0, 3,

        2, 3, 5

    ];

    var vertexBuffer;

    var indexBuffer;

    var vertexNormalBuffer;

    function initBuffers() {

        vertexBuffer = gl.createBuffer();

        gl.bindBuffer(gl.ARRAY\_BUFFER, vertexBuffer);

        gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(vertices), gl.STATIC\_DRAW);

        vertexBuffer.itemSize = 3;

        vertexBuffer.numItems = vertices.length / 3;

        indexBuffer = gl.createBuffer();

        gl.bindBuffer(gl.ELEMENT\_ARRAY\_BUFFER, indexBuffer);

        gl.bufferData(gl.ELEMENT\_ARRAY\_BUFFER, new Uint16Array(indices), gl.STATIC\_DRAW);

        indexBuffer.itemSize = 1;

        indexBuffer.numItems = indices.length;

        vertexNormalBuffer = gl.createBuffer();

        gl.bindBuffer(gl.ARRAY\_BUFFER, vertexNormalBuffer);

        gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(vertexNormals), gl.STATIC\_DRAW);

        vertexNormalBuffer.itemSize = 3;

        vertexNormalBuffer.numItems = vertexNormals.length / 3;

    }

    var rX = 0;

    function drawScene() {

        gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);

        gl.clear(gl.COLOR\_BUFFER\_BIT | gl.DEPTH\_BUFFER\_BIT);

        mat4.perspective(pMatrix, 45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0);

        mat4.identity(mvMatrix);

        mat4.translate(mvMatrix, mvMatrix, [0.0, 0, -5.0]);

        mat4.rotateX(mvMatrix, mvMatrix, degToRad(rX));

        gl.bindBuffer(gl.ARRAY\_BUFFER, vertexBuffer);

        gl.vertexAttribPointer(currentProgram.vertexPositionAttribute, vertexBuffer.itemSize, gl.FLOAT, false, 0, 0);

        gl.bindBuffer(gl.ARRAY\_BUFFER, vertexNormalBuffer);

        gl.vertexAttribPointer(currentProgram.vertexNormalAttribute, vertexNormalBuffer.itemSize, gl.FLOAT, false, 0, 0);

        setMatrixUniforms();

        gl.bindBuffer(gl.ELEMENT\_ARRAY\_BUFFER, indexBuffer);

        gl.drawElements(gl.TRIANGLES, indexBuffer.numItems, gl.UNSIGNED\_SHORT, 0);

    }

    var lastTime = 0;

    function animate() {

        var timeNow = new Date().getTime();

        if (lastTime != 0) {

            var elapsed = timeNow - lastTime;

            rX += (20 \* elapsed) / 1000.0;

        }

        lastTime = timeNow;

    }

    function tick() {

        requestAnimationFrame(tick);

        drawScene();

        animate();

    }

    function webGLStart() {

        var canvas = document.getElementById("canvas");

        initGL(canvas);

        initShaders();

        gl.clearColor(0.2, 0.2, 0.2, 1.0);

        gl.enable(gl.DEPTH\_TEST);

        initBuffers();

        tick();

    }

</script>

</head>

<body onload="webGLStart();">

    <h4>Модель отражения Ламберта, модель интерполяции Фонга</h4>

    <canvas id="canvas" width="500" height="500"></canvas>

</body>

</html>

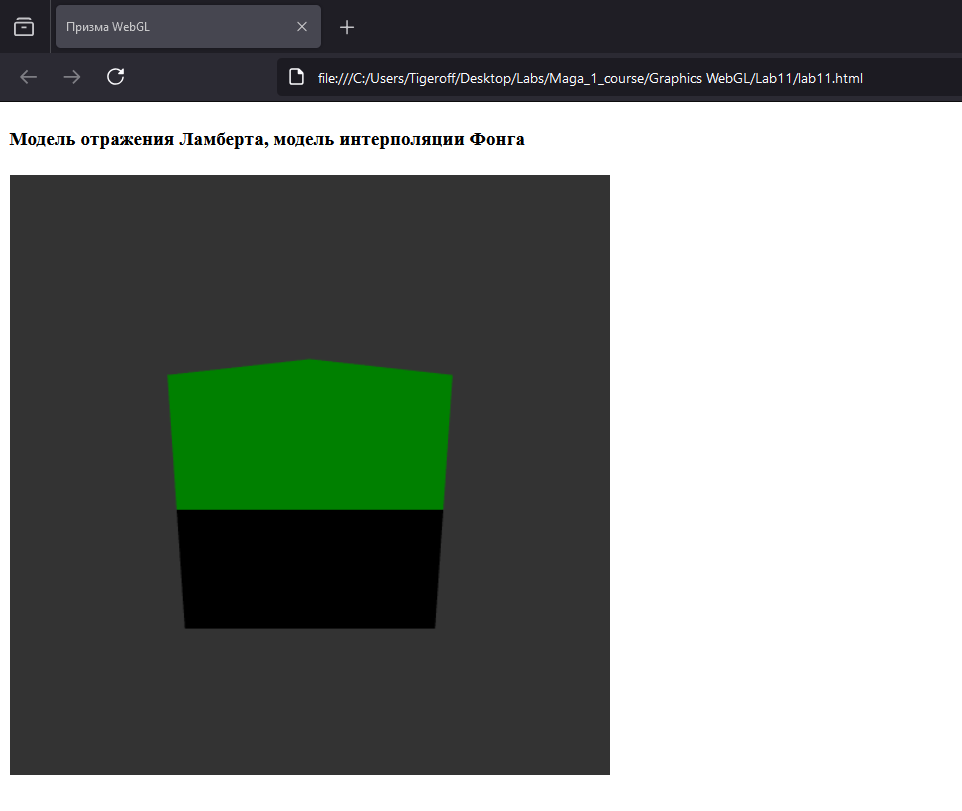


Рисунок 1. Положение сцены 1

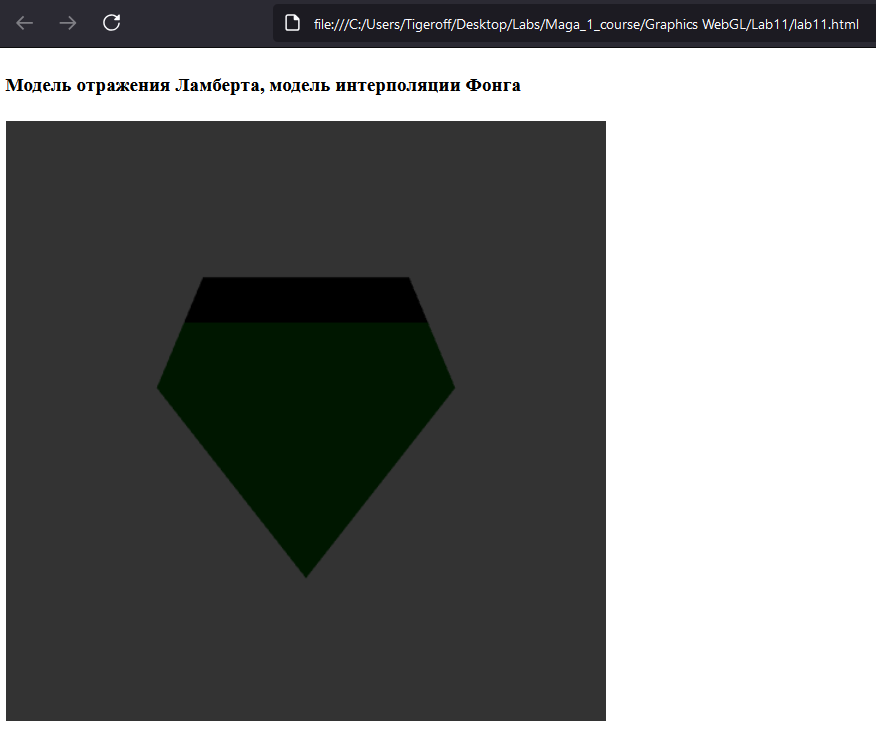


Рисунок 2. Положение сцены 2

Вывод

В результате выполнения работы я провел изучение методов работы с освещением сцены в WEBGL.