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

Отчет
по лабораторной работе №2
по дисциплине «КОМПЬЮТЕРНАЯ 3D-ГРАФИКА»

Тема: 3D трансформации.

Студентка гр. 5303 _____ Допира В.Е.

Преподаватель _____ Герасимова Т.В.

Санкт-Петербург
2019

Цель работы: представить 3D сцену и в ней один или несколько объектов из вашего задания (стараться представить все).

Освоить использование стандартных матричных преобразований над этими объектами и осуществить:

- просмотр трансформаций: через lookAt или эквивалентные модельные трансформации;
- трансформацию проекции: через perspective и ortho;

моделирование трансформации rotate, translate, scale и матричный стек.

Ход работы

Шаг 1. Нарисовать куб

1. Описать куб в initBuffers()

Куб можно представить в виде четырех квадратов, состоящих из двух треугольников каждый, но отрисовываются они за один проход. Каждый треугольник описывается отдельно. Цвета описываются для каждой стороны.

Новый буфер индекса вершин для куба:

```
var cubeVertexPositionBuffer;
var cubeVertexColorBuffer;
var cubeVertexIndexBuffer;
```

2. Значения заносятся в буфер координат вершин пирамиды для всех плоскостей:

```
cubeVertexPositionBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
vertices = [
        // Front face
        -1.0, -1.0,  1.0,
        1.0, -1.0,  1.0,
        1.0,  1.0,  1.0,
        -1.0,  1.0,  1.0,

        // Back face
        -1.0, -1.0, -1.0,
        -1.0,  1.0, -1.0,
        1.0,  1.0, -1.0,
        1.0, -1.0, -1.0,

        // Top face
        -1.0,  1.0, -1.0,
        -1.0,  1.0,  1.0,
```

```
        1.0,  1.0,  1.0,
        1.0,  1.0, -1.0,
        // Bottom face
        -1.0, -1.0, -1.0,
        1.0, -1.0, -1.0,
        1.0, -1.0,  1.0,
        -1.0, -1.0,  1.0,

        // Right face
        1.0, -1.0, -1.0,
        1.0,  1.0, -1.0,
        1.0,  1.0,  1.0,
        1.0, -1.0,  1.0,

        // Left face
        -1.0, -1.0, -1.0,
        -1.0, -1.0,  1.0,
        -1.0,  1.0,  1.0,
        -1.0,  1.0, -1.0
];
gl.bufferData(gl.ARRAY_BUFFER,     new     Float32Array(vertices),
gl.STATIC_DRAW);
cubeVertexPositionBuffer.itemSize = 3;
cubeVertexPositionBuffer.numItems = 24;
```

3. Задан буфер цветов:

```
cubeVertexColorBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexColorBuffer);
colors = [
         [0.2, 0.2, 0.2, 1.0], // Front face
         [0.2, 0.2, 0.2, 1.0], // Back face
         [0.5, 0.5, 0.5, 1.0], // Top face
         [0.2, 0.2, 0.2, 1.0], // Bottom face
         [0.2, 0.2, 0.2, 1.0], // Right face
         [0.2, 0.2, 0.2, 1.0]  // Left face
];
var unpackedColors = [];
for (var i in colors) {
        var color = colors[i];
        for (var j=0; j < 4; j++) {
            unpackedColors = unpackedColors.concat(color);
        }
}
gl.bufferData(gl.ARRAY_BUFFER,  new  Float32Array(unpackedColors),
gl.STATIC_DRAW);
cubeVertexColorBuffer.itemSize = 4;
cubeVertexColorBuffer.numItems = 24;
```

4. Определен массив элементов буфера:

```
cubeVertexIndexBuffer = gl.createBuffer();
```

```
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
var cubeVertexIndices = [
        0, 1, 2,      0, 2, 3,    // Front face
        4, 5, 6,      4, 6, 7,    // Back face
        8, 9, 10,     8, 10, 11,  // Top face
        12, 13, 14,   12, 14, 15, // Bottom face
        16, 17, 18,   16, 18, 19, // Right face
        20, 21, 22,   20, 22, 23  // Left face
];
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,                          new
Uint16Array(cubeVertexIndices), gl.STATIC_DRAW);
cubeVertexIndexBuffer.itemSize = 1;
cubeVertexIndexBuffer.numItems = 36;
```

5. Куб добавлен в drawScene()

Сопоставлены буферы координат вершин и цветов куба:

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

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

6. Создан массив элементов буфера, который заполняется необходимыми значениями в функции initBuffers. Он хранит список вершин, используя нумеруемые с нуля ссылки на массивы, которые использованы для координат и цветов.

Текущим буфером назначен массив элементов буфера куба, затем выполняется код для передачи матрицы модель-вид и проекционной матрицы на видеокарту, и затем вызывается drawElements, чтобы нарисовать треугольники:

```
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
setMatrixUniforms();
gl.drawElements(gl.TRIANGLES, cubeVertexIndexBuffer.numItems,
gl.UNSIGNED_SHORT, 0);
```

Шаг 2. Нарисовать шар

1. Определение переменных для буфферов:

```
    var ballVertexPositionBuffer;
    var ballVertexNormalBuffer;
    var ballVertexColordBuffer;
    var ballVertexIndexBuffer;
```

2. Разбить шар на сегменты

Чтобы отобразить набор треугольников, которые аппроксимируются на сферу, необходимо разделить их. Сфера разрезается линиями широты и долготы (см. рис. 1).

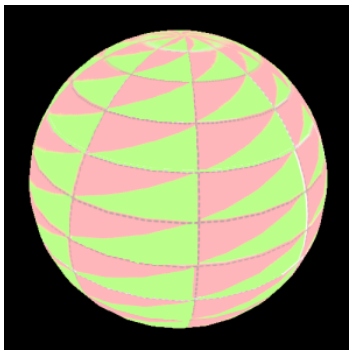

Рисунок 1 — Сегменты шара

Теперь нужно рассчитать все точки пересечения этих линий и использовать их как координаты вершин. Затем можно разделить прямоугольник, сформированный из двух смежных линий широты и двух смежных линий долготы, на два треугольника и затем отрисовать эти треугольники.

Все точки одной конкретной широты, независимо от значений долготы, имеют одинаковую координату Y. Поэтому все точки на n-ной широте сферы с радиусом равным единице и с десятью линиями широты будут иметь координату Y равной cos(nπ / 10).

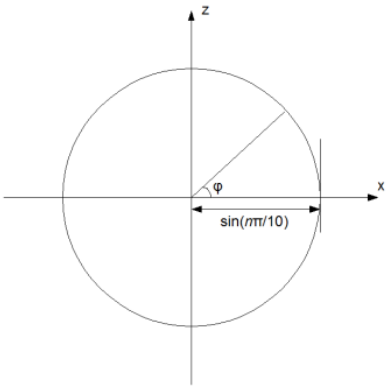

Рисунок 2 — Поиск координат шара

Координата X при Z равной нулю равняется sin(nπ / 10). Рассмотрим срез сферы, как показано на рис. 2 — горизонтальный срез через плоскость n-ной широты. Все точки находятся на окружности с радиусом sin(nπ / 10). Назовем это значение k. Если мы разделим окружность по линиям долготы, количество

которых предположим 10 и учитывая, что будет 2π радиан в окружности, то значениями угла φ, перемещающегося по окружности, будут 2π/10, 4π/10 и так далее. Несложной тригонометрией получено, что координаты X будут равняться kcos(φ), а координаты Z — ksin(φ).

Подводя итог, для сферы радиуса r с m поясами широты и n поясами долготы, можно определить значения x, y и z, разбивая диапазон от 0 до π на m частей значениями θ, и разбивая диапазон от 0 до 2π на n частей значениями φ, и затем вычисляя:

- $x = r \sin\theta \cos\varphi$
- $y = r \cos\theta$
- $z = r \sin\theta \sin\varphi$

Выполняется цикл через все срезы по широте, затем внутри этого цикла проходятся все сегменты по долготе и генерируются нормали, текстурные координаты и координаты вершин для каждого. Циклы завершаются, когда индекс больше количества линий долготы/широты.

```
var vertexPositionData = [];
var normalData = [];
for (var latNumber=0; latNumber <= latitudeBands; latNumber++) {
  var theta = latNumber * Math.PI / latitudeBands;
  var sinTheta = Math.sin(theta);
  var cosTheta = Math.cos(theta);

  for (var longNumber=0; longNumber <= longitudeBands; longNumber+
+) {
    var phi = longNumber * 2 * Math.PI / longitudeBands;
    var sinPhi = Math.sin(phi);
    var cosPhi = Math.cos(phi);
    var x = cosPhi * sinTheta;
    var y = cosTheta;
    var z = sinPhi * sinTheta;
    var u = 1 - (longNumber / longitudeBands);
    var v = 1 - (latNumber / latitudeBands);
    normalData.push(x);
    normalData.push(y);
    normalData.push(z);
    vertexPositionData.push(radius * x);
    vertexPositionData.push(radius * y);
    vertexPositionData.push(radius * z);
    }
}
```

Связывание вершин: для этого создаданы индексы вершин, состоящих из шести значений, которые представляют прямоугольник в виде двух примыкающих треугольников.

Цикл проходит по всем вершинам, сохраняя ее индекс в переменной first, затем осуществляется поиск эквивалентной точки на нижнем поясе широты, прибавляя к переменной first longitudeBands + 1 индексов (один дополнительный из-за наложения вершин) и сохраняется это значение в переменной second. Затем создаются два треугольника, как показано на рис. 3.

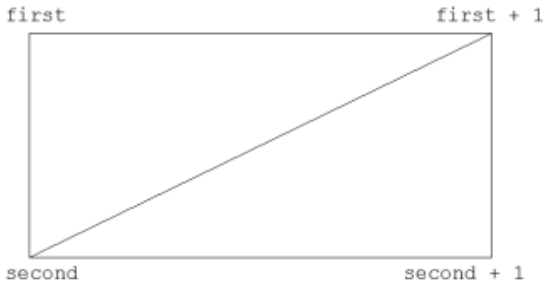

Рисунок 3 — Связывание вершин

```
var indexData = [];
for (var latNumber=0; latNumber < latitudeBands; latNumber++) {
   for (var longNumber=0; longNumber < longitudeBands; longNumber+
+) {
   var first = (latNumber * (longitudeBands + 1)) + longNumber;
   var second = first + longitudeBands + 1;
   indexData.push(first);
   indexData.push(second);
   indexData.push(first + 1);

   indexData.push(second);
   indexData.push(second + 1);
   indexData.push(first + 1);
   }
}
```

3. Аналогично задан буфер цветов:

```
ballVertexColorBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ARRAY_BUFFER, ballVertexColorBuffer);
        colors = [];
        for (var i=0; i < normalData.length / 3; i++) {
            colors = colors.concat([1.0, 0.0, 0.0, 1.0]);
        }
          gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors),
gl.STATIC_DRAW);
        ballVertexColorBuffer.itemSize = 4;
```

```
        ballVertexColorBuffer.numItems = normalData.length / 3;
```

4. Шар добавлен в drawScene()

Подготовлены буферы для рисовки шара:

```
mat4.translate(mvMatrix, [5, 1.0, -5.0]);
mat4.multiply(mvMatrix, ballRotationMatrix);
gl.uniform1i(shaderProgram.samplerUniform, 0);
gl.bindBuffer(gl.ARRAY_BUFFER, ballVertexPositionBuffer);
gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
ballVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
gl.bindBuffer(gl.ARRAY_BUFFER, ballVertexColorBuffer);
gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
ballVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);
gl.bindBuffer(gl.ARRAY_BUFFER, ballVertexNormalBuffer);
gl.vertexAttribPointer(shaderProgram.vertexNormalAttribute,
ballVertexNormalBuffer.itemSize, gl.FLOAT, false, 0, 0);
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, ballVertexIndexBuffer);
setMatrixUniforms();
gl.drawElements(gl.TRIANGLES,       ballVertexIndexBuffer.numItems,
gl.UNSIGNED_SHORT, 0);
```

Результат выполнения программы показан на рисунке 4.

Рисунок 4 — Результат работы программы

Вывод

В ходе лабораторной работы была представлена 3D сцена и объекты куб и шар. Освоены матричные преобразования над объектами.

Приложение 1

Листинг программы

```
<html>

<head>
<title>Lab 2</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 type="text/javascript" src="webgl-utils.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 vec3 aVertexNormal;
    attribute vec4 aVertexColor;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;
    uniform mat3 uNMatrix;

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

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

    var mvMatrix = mat4.create();
    var mvMatrixStack = [];
    var pMatrix = mat4.create();

    function mvPushMatrix() {
        var copy = mat4.create();
        mat4.set(mvMatrix, copy);
        mvMatrixStack.push(copy);
    }

    function mvPopMatrix() {
```

```
        if (mvMatrixStack.length == 0) {
            throw "Invalid popMatrix!";
        }
        mvMatrix = mvMatrixStack.pop();
    }

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

        var normalMatrix = mat3.create();
        mat4.toInverseMat3(mvMatrix, normalMatrix);
        mat3.transpose(normalMatrix);
          gl.uniformMatrix3fv(shaderProgram.nMatrixUniform, false,
normalMatrix);
    }


    function degToRad(degrees) {
        return degrees * Math.PI / 180;
    }


    var mouseDown = false;
    var lastMouseX = null;
    var lastMouseY = null;

    var ballRotationMatrix = mat4.create();
    mat4.identity(ballRotationMatrix);

    function handleMouseDown(event) {
        mouseDown = true;
        lastMouseX = event.clientX;
        lastMouseY = event.clientY;
    }


    function handleMouseUp(event) {
        mouseDown = false;
    }


    function handleMouseMove(event) {
        if (!mouseDown) {
            return;
        }
        var newX = event.clientX;
        var newY = event.clientY;
```

```
        var deltaX = newX - lastMouseX
        var newRotationMatrix = mat4.create();
        mat4.identity(newRotationMatrix);
         mat4.rotate(newRotationMatrix, degToRad(deltaX / 10), [0,
1, 0]);

        var deltaY = newY - lastMouseY;
         mat4.rotate(newRotationMatrix, degToRad(deltaY / 10), [1,
0, 0]);

              mat4.multiply(newRotationMatrix,  ballRotationMatrix,
ballRotationMatrix);

        lastMouseX = newX
        lastMouseY = newY;
    }


    var pyramidVertexPositionBuffer;
    var pyramidVertexColorBuffer;
    var cubeVertexPositionBuffer;
    var cubeVertexColorBuffer;
    var cubeVertexIndexBuffer;
    var ballVertexPositionBuffer;
    var ballVertexNormalBuffer;
    var ballVertexColordBuffer;
    var ballVertexIndexBuffer;

    function initBuffers() {
        pyramidVertexPositionBuffer = gl.createBuffer();
                                    gl.bindBuffer(gl.ARRAY_BUFFER,
pyramidVertexPositionBuffer);
        var vertices = [
            // Front face
             0.0,  1.0,  0.0,
            -1.0, -1.0,  1.0,
             1.0, -1.0,  1.0,

            // Right face
             0.0,  1.0,  0.0,
             1.0, -1.0,  1.0,
             1.0, -1.0, -1.0,

            // Back face
             0.0,  1.0,  0.0,
             1.0, -1.0, -1.0,
            -1.0, -1.0, -1.0,

            // Left face
```

```
             0.0,  1.0,  0.0,
            -1.0, -1.0, -1.0,
            -1.0, -1.0,  1.0
        ];
         gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
gl.STATIC_DRAW);
        pyramidVertexPositionBuffer.itemSize = 3;
        pyramidVertexPositionBuffer.numItems = 12;

        pyramidVertexColorBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ARRAY_BUFFER, pyramidVertexColorBuffer);
        var colors = [];/*
            // Front face
            1.0, 0.0, 0.0, 1.0,
            0.0, 1.0, 0.0, 1.0,
            0.0, 0.0, 1.0, 1.0,

            // Right face
            1.0, 0.0, 0.0, 1.0,
            0.0, 0.0, 1.0, 1.0,
            0.0, 1.0, 0.0, 1.0,

            // Back face
            1.0, 0.0, 0.0, 1.0,
            0.0, 1.0, 0.0, 1.0,
            0.0, 0.0, 1.0, 1.0,

            // Left face
            1.0, 0.0, 0.0, 1.0,
            0.0, 0.0, 1.0, 1.0,
            0.0, 1.0, 0.0, 1.0
        ];*/
        for (var i=0; i < 12; i++) {
            colors = colors.concat([0.9, 0.9, 0.9, 1.0]);
        }
          gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors),
gl.STATIC_DRAW);
        pyramidVertexColorBuffer.itemSize = 4;
        pyramidVertexColorBuffer.numItems = 12;

        cubeVertexPositionBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
        vertices = [
            // Front face
            -1.0, -1.0,  1.0,
             1.0, -1.0,  1.0,
             1.0,  1.0,  1.0,
            -1.0,  1.0,  1.0,

            // Back face
```

```
            -1.0, -1.0, -1.0,
            -1.0,  1.0, -1.0,
             1.0,  1.0, -1.0,
             1.0, -1.0, -1.0,

            // Top face
            -1.0,  1.0, -1.0,
            -1.0,  1.0,  1.0,
             1.0,  1.0,  1.0,
             1.0,  1.0, -1.0,

            // Bottom face
            -1.0, -1.0, -1.0,
             1.0, -1.0, -1.0,
             1.0, -1.0,  1.0,
            -1.0, -1.0,  1.0,

            // Right face
             1.0, -1.0, -1.0,
             1.0,  1.0, -1.0,
             1.0,  1.0,  1.0,
             1.0, -1.0,  1.0,

            // Left face
            -1.0, -1.0, -1.0,
            -1.0, -1.0,  1.0,
            -1.0,  1.0,  1.0,
            -1.0,  1.0, -1.0
        ];
         gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
gl.STATIC_DRAW);
        cubeVertexPositionBuffer.itemSize = 3;
        cubeVertexPositionBuffer.numItems = 24;

        cubeVertexColorBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexColorBuffer);
        colors = [
            [0.2, 0.2, 0.2, 1.0], // Front face
            [0.2, 0.2, 0.2, 1.0], // Back face
            [0.5, 0.5, 0.5, 1.0], // Top face
            [0.2, 0.2, 0.2, 1.0], // Bottom face
            [0.2, 0.2, 0.2, 1.0], // Right face
            [0.2, 0.2, 0.2, 1.0]  // Left face
        ];
        var unpackedColors = [];
        for (var i in colors) {
            var color = colors[i];
            for (var j=0; j < 4; j++) {
                unpackedColors = unpackedColors.concat(color);
            }
```

```
        }
                                gl.bufferData(gl.ARRAY_BUFFER,    new
Float32Array(unpackedColors), gl.STATIC_DRAW);
        cubeVertexColorBuffer.itemSize = 4;
        cubeVertexColorBuffer.numItems = 24;

        cubeVertexIndexBuffer = gl.createBuffer();
                               gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER,
cubeVertexIndexBuffer);
        var cubeVertexIndices = [
            0, 1, 2,      0, 2, 3,    // Front face
            4, 5, 6,      4, 6, 7,    // Back face
            8, 9, 10,     8, 10, 11,  // Top face
            12, 13, 14,   12, 14, 15, // Bottom face
            16, 17, 18,   16, 18, 19, // Right face
            20, 21, 22,   20, 22, 23  // Left face
        ];
                           gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,   new
Uint16Array(cubeVertexIndices), gl.STATIC_DRAW);
        cubeVertexIndexBuffer.itemSize = 1;
        cubeVertexIndexBuffer.numItems = 36;


        var latitudeBands = 30;
        var longitudeBands = 30;
        var radius = 2;

        var vertexPositionData = [];
        var normalData = [];
              for  (var  latNumber=0;  latNumber  <=  latitudeBands;
latNumber++) {
            var theta = latNumber * Math.PI / latitudeBands;
            var sinTheta = Math.sin(theta);
            var cosTheta = Math.cos(theta);

              for (var longNumber=0; longNumber <= longitudeBands;
longNumber++) {
                           var  phi  =  longNumber  *  2  *  Math.PI  /
longitudeBands;
                var sinPhi = Math.sin(phi);
                var cosPhi = Math.cos(phi);

                var x = cosPhi * sinTheta;
                var y = cosTheta;
                var z = sinPhi * sinTheta;
                var u = 1 - (longNumber / longitudeBands);
                var v = 1 - (latNumber / latitudeBands);

                normalData.push(x);
                normalData.push(y);
```

```
                normalData.push(z);
                // textureCoordData.push(u);
                // textureCoordData.push(v);
                vertexPositionData.push(radius * x);
                vertexPositionData.push(radius * y);
                vertexPositionData.push(radius * z);
            }
        }

        var indexData = [];
                for (var latNumber=0; latNumber < latitudeBands;
latNumber++) {
                for (var longNumber=0; longNumber < longitudeBands;
longNumber++) {
                  var first = (latNumber * (longitudeBands + 1)) +
longNumber;
                var second = first + longitudeBands + 1;
                indexData.push(first);
                indexData.push(second);
                indexData.push(first + 1);

                indexData.push(second);
                indexData.push(second + 1);
                indexData.push(first + 1);
            }
        }

        ballVertexNormalBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ARRAY_BUFFER, ballVertexNormalBuffer);
                            gl.bufferData(gl.ARRAY_BUFFER,    new
Float32Array(normalData), gl.STATIC_DRAW);
        ballVertexNormalBuffer.itemSize = 3;
        ballVertexNormalBuffer.numItems = normalData.length / 3;

        ballVertexColorBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ARRAY_BUFFER, ballVertexColorBuffer);
        colors = [];
        for (var i=0; i < normalData.length / 3; i++) {
            colors = colors.concat([1.0, 0.0, 0.0, 1.0]);
        }
          gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors),
gl.STATIC_DRAW);
        ballVertexColorBuffer.itemSize = 4;
        ballVertexColorBuffer.numItems = normalData.length / 3;

        ballVertexPositionBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ARRAY_BUFFER, ballVertexPositionBuffer);
                            gl.bufferData(gl.ARRAY_BUFFER,    new
Float32Array(vertexPositionData), gl.STATIC_DRAW);
        ballVertexPositionBuffer.itemSize = 3;
```

```
                                ballVertexPositionBuffer.numItems    =
vertexPositionData.length / 3;

        ballVertexIndexBuffer = gl.createBuffer();
                                gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER,
ballVertexIndexBuffer);
                           gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,   new
Uint16Array(indexData), gl.STATIC_DRAW);
        ballVertexIndexBuffer.itemSize = 1;
        ballVertexIndexBuffer.numItems = indexData.length;
    }


    var rPyramid = 0;
    var rCube = 0;
    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, [-3, 0.0, -8.0]);
        mvPushMatrix();
        mat4.rotate(mvMatrix, degToRad(rCube), [1, 1, 1]);

        gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
         gl.vertexAttribPointer(shaderProgram.vertexPositionAttribu
te, cubeVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

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

                                gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER,
cubeVertexIndexBuffer);
        setMatrixUniforms();
                                        gl.drawElements(gl.TRIANGLES,
cubeVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);
        mvPopMatrix();


        mat4.translate(mvMatrix, [5, 1.0, -5.0]);

        // mat4.translate(mvMatrix, [0, 0, -6]);

        mat4.multiply(mvMatrix, ballRotationMatrix);
```

```
        gl.uniform1i(shaderProgram.samplerUniform, 0);

        gl.bindBuffer(gl.ARRAY_BUFFER, ballVertexPositionBuffer);
         gl.vertexAttribPointer(shaderProgram.vertexPositionAttribu
te, ballVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

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

        gl.bindBuffer(gl.ARRAY_BUFFER, ballVertexNormalBuffer);
         gl.vertexAttribPointer(shaderProgram.vertexNormalAttribute
, ballVertexNormalBuffer.itemSize, gl.FLOAT, false, 0, 0);

                                gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER,
ballVertexIndexBuffer);
        setMatrixUniforms();
                                        gl.drawElements(gl.TRIANGLES,
ballVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);
    }


    function tick() {
        requestAnimFrame(tick);
        drawScene();
    }


    function webGLStart() {
        var canvas = document.getElementById("lab2");
        initGL(canvas);
        initShaders();
        initBuffers();

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

        canvas.onmousedown = handleMouseDown;
        document.onmouseup = handleMouseUp;
        document.onmousemove = handleMouseMove;

        tick();
    }

</script>


</head>
```

```
<body onload="webGLStart();">
    <p> Lab 2</p>

          <canvas   id="lab2"   style="border:   none;"   width="700"
height="500"></canvas>
</body>

</html>
```