Санкт-Петербургский Государственный
Электротехнический Университет

Кафедра МОЭВМ

Задание для лабораторной работы № 1
"Рисование геометрических объектов"

Выполнил: Калмак Д.А.
Факультет: ФКТИ
Группа: 0303
Преподаватель: Герасимова Т.В.

Санкт-Петербург
2025 г.

Цель работы.

- ознакомление с основами программирования на WebGL.

- проанализировать полученное задание, выделить информационные объекты и действия.

- разработать программу для отрисовки геометрических объектов.

- дополнить программу средствами регулирования параметров объектов.

Основные теоретические положения.

WebGL является 3D графической библиотекой, которая позволяет современным интернет-браузерам отрисовывать 3D-сцены стандартным и эффективным способом. WebGL имеет клиенто-ориентированный подход; элементы, которые составляют части 3D-сцены, обычно загружаются с сервера. Однако, вся дальнейшая обработка, необходимая для получения изображения выполняется локально, с помощью графического оборудования клиента.

Чтобы написать приложение WebGL, первым делом нужно получить объект контекста рендеринга WebGL. Этот объект взаимодействует с буфером рисования WebGL и может вызывать все методы WebGL. Следующие операции выполняются для получения контекста WebGL:

- Создать холст(canvas) HTML-5
- Получить идентификатор холста
- Получить WebGL

Шейдеры – это программы, которые работают на GPU. Шейдеры написаны на OpenGL ES Shader Language (известный как ES SL). ES SL имеет свои собственные переменные, типы данных, классификаторы, встроенные входы и выходы.

Файл HTML для приложения WebGL будет содержать ссылки на необходимые файлы JavaScript и ресурсы. Некоторые ресурсы, такие как шейдеры, могут быть определены в строке.

Файл javascript вдохнет жизнь в ваше приложение WebGL. Он устанавливает контекст рендеринга WebGL, рисует и определяет ответы на различные события.

Шейдер представляет собой часть шейдерной программы, которая заменяет собой часть графического конвейера видеокарты. Тип шейдера зависит от того, какая часть конвейера будет заменена. Каждый шейдер должен вы-полнить свою обязательную работу, т. е. записать какие-то данные и передать их дальше по графическому конвейеру. шейдеры делятся на типы:

– вершинный шейдер (vertex shader);

– геометрический шейдер (geometric shader);

– фрагментный шейдер (fragment shader);

Дополнительно существуют вычислительные (compute) шейдеры, которые выполняются независимо от графического конвейера.

Разные шаги графического конвейера накладывают разные ограничения на работу шейдеров. Поэтому у каждого типа шейдеров есть своя специфика.

Геометрический и тессяляционные шейдеры не являются обязательны-ми. Современный OpenGL требует наличия только вершинного и фрагментного шейдера. Хотя существует сценарий, при котором фрагментный шейдер может отсутствовать.

Данные вершин могут быть двух, трех или четырехмерными. Иногда требуется дополнительное измерение, чтобы правильно перемещать вершины в пространстве. Данные вершин чаще всего представлены в шейдерах с типами данных vec2, vec3 и vec4. Это 2, 3 и 4-компонентные структуры с плавающей точкой. Эти данные должны быть загружены из массивов JavaScript 32-битного типа с плавающей запятой.

Если у вас есть данные вершин, вам нужно загрузить их в буферы. Каждый массив может быть загружен в отдельный буфер, или все массивы могут быть упакованы в один и тот же буфер.

Как только буфер загружен данными, он должен быть подключен к правильному входу в вашей шейдерной программе. Для этого вы запрашиваете ввод по имени, включаете его, а затем присоединяете свои данные в текущем связанном буфере к входу с описанием того, как данные форматируются.

Можно нарисовать только один тип точки:

- POINTS – рисует точку для каждой вершины.

Можно создать три различных линейных примитива:

- LINES – рисует отрезок линии для каждой пары вершин.
- LINE_STRIP – рисует связанную группу отрезков линии от вершины v0 до vn, соединяющую линию между каждой вершиной и следующей в указанном порядке.
- LINE_LOOP – аналогично LINE_STRIP, за исключением того, что закрывает строку от vn до v0, определяя цикл.

Треугольники:

Лицевые грани, задние грани и режимы рендеринга:

- cullFace() – если CULL_FACE включен, это указывает, какие стороны треугольника отбрасывать, FRONT, BACK или FRONT_AND_BACK.
- frontFace() – указать , по часовой стрелке, то ли CW или против часовой стрелки, против часовой стрелки для того точка рисунка означает треугольник перед фронтом.

Типы треугольников

- TRIANGLE – рисует ряд отдельных трехсторонних многоугольников
- TRIANGLE_STRIP – рисует полосу соединенных треугольников. Первые три вершины определяют полный треугольник. Каждая последующая вершина завершает треугольник с двумя предыдущими. Порядок отрисовки первых двух точек менял местами каждый треугольник, чтобы помочь поддерживать

порядок отрисовки по часовой стрелке или против часовой стрелки, т.е.: вершины 0-1-2-3-4-5-6 нарисованы так: 0-1-2, 2-1-3, 2-3-4, 4-3-5, 4-5-6

- TRIANGLE_FAN – рисует полоску треугольников, соединенных общим источником. Первые три вершины определяют полный треугольник. Каждая последующая вершина завершает треугольник с предыдущей и первой вершиной.

До сих пор наш шейдер использовал жестко закодированный цвет. Вы можете изменить этот цвет в работающей программе одним из двух способов: униформой и атрибутами. Все цвета будут в формате RGBA – красный, зеленый, синий, альфа. Альфа – это дополнительный термин, используемый в операциях смешивания. Вы можете думать об этом как о «прозрачности».

Цветовые массивы работают так же, как массивы позиций вершин. Вам нужно будет настроить второй вход массива для вашего вершинного шейдера, создать массив цветов, загрузить его в буфер и прикрепить к шейдеру.

Задание.

Разработать программу, реализующую представление куба и пирамиды.

Разработанная программа должна быть пополнена возможностями регулировки параметров фигур с использованием интерфейса, включая смещение, масштабирование и поворот.

Выполнение работы.

Работа выполнена с использованием HTML для веб-страницы, JavaScript и WebGL для логики и рендеринга приложения и gl-matrix – библиотеки для работы с матрицами. Для визуализации разработки использовался браузер Microsoft Edge Версия 133.0.3065.92.

Для изображения сцены используется canvas. Для организации интерфейса веб-страницы созданы div блоки, к которым применены стили. Справа от окна сцены разработана панель управления для каждой фигуры. Панель реализована с ползунками range для регулирования x, y, z смещения, x и y масштабирования, x, y, z поворотов для каждой из фигур. Подключены два скрипта JavaScript: WebGLJavascript.js для логики и рендеринга приложения и gl-matrix.js для работы с матрицами.

Создание холста произведено с использованием методов getElementById() и getContext(). Переменная cubeVertices содержит координаты вершин граней, причем при отрисовки треугольниками каждая грань задана двумя треугольниками. Переменная cubeColors определяет цвета граней куба. Аналогично для пирамиды pyramidVertices и pyramidColors, причем пирамида с квадратным основанием. Поскольку фигур несколько, для создания буфера рационально было создать метод createBuffer(data). В ней создается buffer – пустой буферный объект для хранения буфера вершин, в том числе функция рассчитана и на их цвета. С помощью метода bindBuffer() привязываем к нему соответствующий буфер массива, а с помощью bufferData() – выделяет место в указанном буфере и инициализирует его данными. Создаются отдельные буферы для геометрии и для цветов каждой фигуры: cubeBuffer, pyramidBuffer и cubeColorBuffer и pyramidColorBuffer соответственно. Переменная vertCode содержит код вершинного шейдера, в котором также есть вектор vColor для передачи цвета и vPosition для передачи координат в фрагментный шейдер и матрица преобразований modelMatrix. Фрагментный же шейдер использует vColor и значение переменной gradient, которая получается путем преобразования координат vPosition, для итогового цвета, а точность задана с плавающей точкой с помощью precision mediump float. Созданы объекты вершинного и фрагментного шейдера с помощью метода createShader() с аргументом gl.VERTEX_SHADER или gl.FRAGMENT_SHADER соответственно. Метод

shaderSource() позволяет прикрепить исходный код шейдера, а compileShader() – скомпилировать. Метод createProgram() использован для создания объекта программы шейдера для хранения комбинированной программы шейдера, метод attachShader() – прикрепления шейдеров, метод linkProgram() – линковки, а метод useProgram() – использования при отрисовки. Определяются местоположения атрибутов “coordinates” и “color” в шейдерной программе с помощью метода getAttribLocation() и включаются атрибуты enableVertexAttribArray(). Создаются две матрицы методом create() у mat4 из gl-matrix: одна для куба cubeModelMatrix и другая для пирамиды pyramidModelMatrix. Обе матрицы инициализируются как единичные identity() методом. Получается uniform-переменная “modelMatrix” из шейдерной программы методом getUniformLocation(), через которую будут передаваться матрицы преобразований, в переменные cubeModelMatrixLocation и pyramidModelMatrixLocation. Для отрисовки куба написана функция drawCube(vertexBuffer, colorBuffer, vertexCount) с буферами вершин и их цветов, а также переменная с их количеством. В функции связываются буферы с координатами вершин и цветами методом bindBuffer(), затем настраивается расположение атрибута методом vertexAttribPointer(). Метод drawArrays() отрисовывает куб треугольниками. Аналогично реализована функция drawPyramid(vertexBuffer, colorBuffer, vertexCount) для отрисовки пирамиды за исключением того, что основание пирамиды отрисовывается отдельно методом drawArrays(gl.TRIANGLE_FAN, 0, 4), поскольку основание квадратное. Устанавливается черный цвет очистки экрана методом clearColor(). Включен тест глубины enable(gl.DEPTH_TEST). Производится очистка буфера цвета и глубины методом clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT). Задаётся область отрисовки, равная размерам canvas, с помощью метода viewport(). Задана функция updateView() для обновления матриц для трансформаций и повторной отрисовки сцены. Функция запускается первый раз для отрисовки, а затем с помощью

обработчика событий addEventListener(), который следит за каждым ползунком в панеле управления. По Id элемента получаются значения с ползунков панели управления каждой фигуры с плавающей точки используя метод getElementById() и функцию parseFloat() в паре. Затем для каждой матрицы фигур соответственно производится сброс, смещение, повороты и масштабирование с помощью методов mat4 из gl-matrix identity(), translate(), rotateX(), rotateY(), rotateZ() и scale(). Очищаются буферы цвета и глубины, а затем с помощью метода uniformMatrix4fv() обновляется значение uniform-переменной “modelMatrix” в шейдере значением матриц у каждой фигуры и происходит отрисовка функциями drawCube() и drawPyramid().

Тестирование.

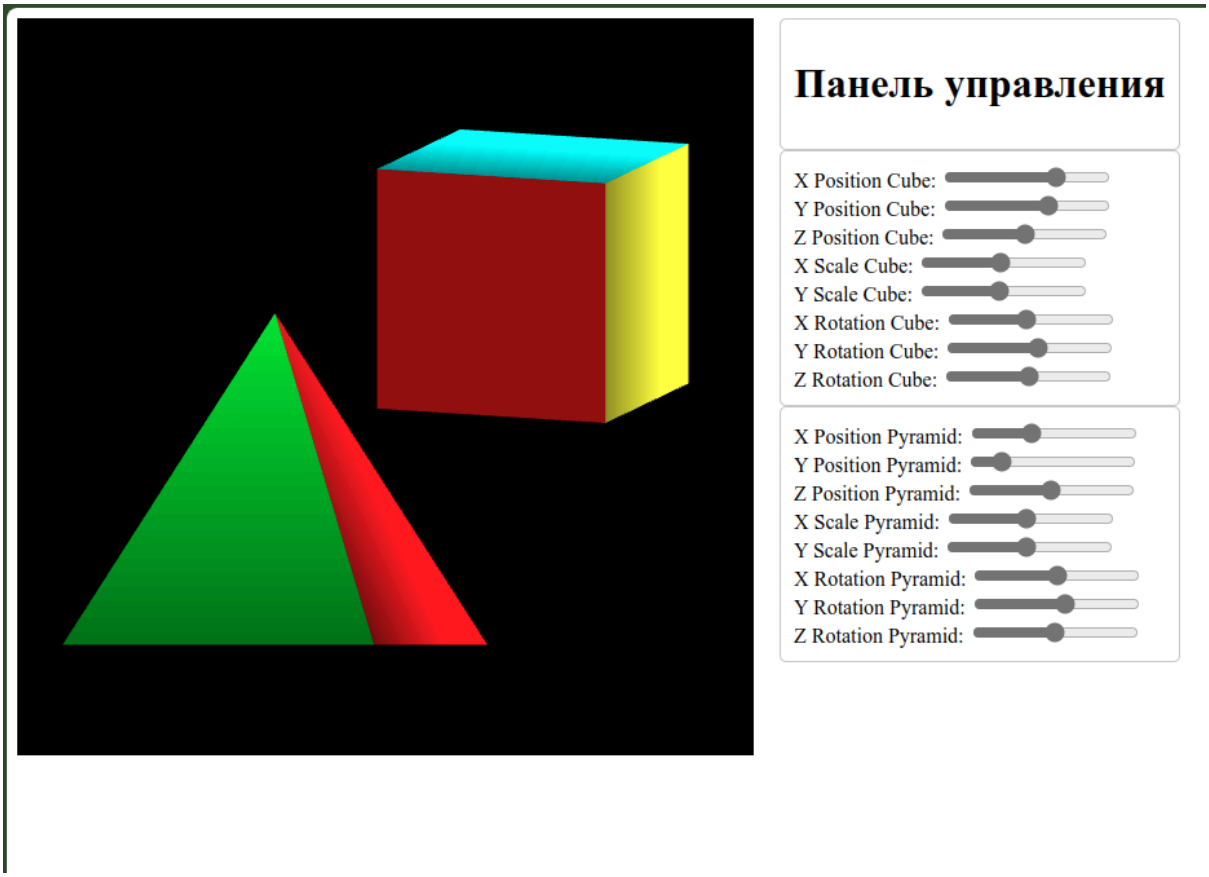

Рисунок 1 – Сцена с кубом и пирамидой с панелью управления

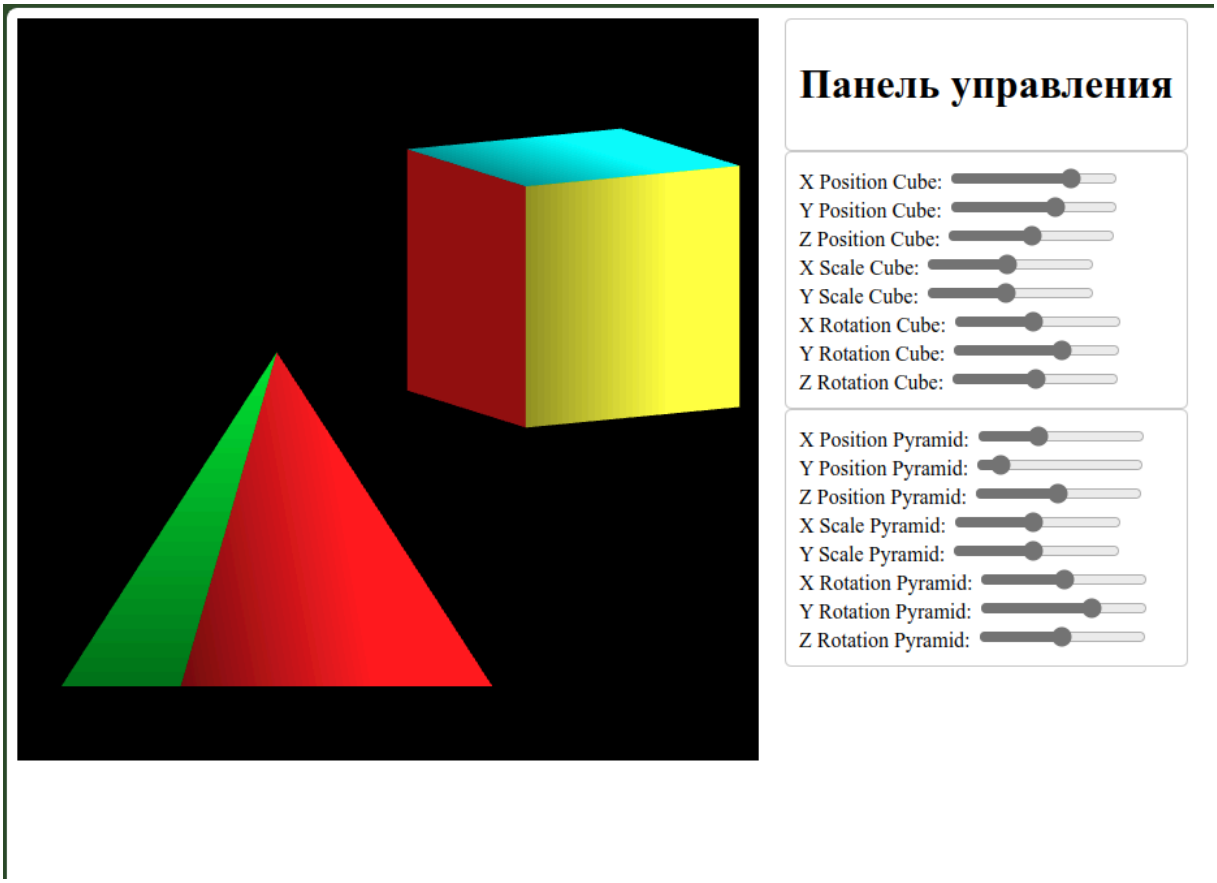

Рисунок 2 – Сцена с кубом и пирамидой с панелью управления, где куб и пирамида повернуты и сдвинуты

Вывод.

В результате выполнения лабораторной работы была разработана программа, отображающая геометрические объекты: куб и пирамида, а также панель управления для каждой из фигур. Программа работает корректно. При выполнении работы были приобретены навыки работы с графической библиотекой WebGL.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

main.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>WebGL CG3D lab1</title>
    </head>
    <body>
        <div class="page">
            <canvas width="570" height="570" id="my_Canvas"></canvas>
            <div class="controls">
                <div class="title-control">
                    <h1>Панель управления</h1>
                </div>
                <div class="cube-control">
                    <label>X Position Cube:</label>
                    <input type="range" id="xPosCube" min="-1" max="1" step="0.1"
value="0.4">
                    <br>
                    <label>Y Position Cube:</label>
                    <input type="range" id="yPosCube" min="-1" max="1" step="0.1"
value="0.3">
                    <br>
                    <label>Z Position Cube:</label>
                    <input type="range" id="zPosCube" min="-1" max="1" step="0.1"
value="0">
                    <br>
                    <label>X Scale Cube:</label>
                    <input type="range" id="xScaleCube" min="0.1" max="2" step="0.1"
value="1">
                    <br>
                    <label>Y Scale Cube:</label>
                    <input type="range" id="yScaleCube" min="0.1" max="2" step="0.1"
value="1">
                    <br>
                    <label>X Rotation Cube:</label>
                    <input type="range" id="xRotateCube" min="-180" max="180" step="1"
value="-10">
                    <br>
                    <label>Y Rotation Cube:</label>
                    <input type="range" id="yRotateCube" min="-180" max="180" step="1"
value="20">
                    <br>
                    <label>Z Rotation Cube:</label>
                    <input type="range" id="zRotateCube" min="-180" max="180" step="1"
value="0">
                </div>
                <div class="pyramid-control">
                    <label>X Position Pyramid:</label>
                    <input type="range" id="xPosPyramid" min="-1" max="1" step="0.1"
value="-0.3">
                    <br>
                    <label>Y Position Pyramid:</label>
```

```
                    <input type="range" id="yPosPyramid" min="-1" max="1" step="0.1"
value="-0.7">
                    <br>
                    <label>Z Position Pyramid:</label>
                    <input type="range" id="zPosPyramid" min="-1" max="1" step="0.1"
value="0">
                    <br>
                    <label>X Scale Pyramid:</label>
                    <input type="range" id="xScalePyramid" min="0.1" max="2" step="0.1"
value="1">
                    <br>
                    <label>Y Scale Pyramid:</label>
                    <input type="range" id="yScalePyramid" min="0.1" max="2" step="0.1"
value="1">
                    <br>
                    <label>X Rotation Pyramid:</label>
                    <input type="range" id="xRotatePyramid" min="-180" max="180"
step="1" value="0">
                    <br>
                    <label>Y Rotation Pyramid:</label>
                    <input type="range" id="yRotatePyramid" min="-180" max="180"
step="1" value="20">
                    <br>
                    <label>Z Rotation Pyramid:</label>
                    <input type="range" id="zRotatePyramid" min="-180" max="180"
step="1" value="0">
                </div>
            </div>
        </div>
        <script src="gl-matrix.js"></script>
        <script src="WebGLJavascript.js"></script>
    </body>
    <style>
        .page {
            display: flex;
            align-items: flex-start;
            gap: 20px;
        }

        .title-control, .cube-control, .pyramid-control {
            border: 1px solid #ccc;
            padding: 10px;
            border-radius: 5px;
        }
    </style>
</html>
```

WebGLJavascript.js

```
var canvas = document.getElementById('my_Canvas');
var gl = canvas.getContext('webgl');

var cubeVertices = [
   // перед
   -0.33, -0.33, -0.33,  0.33, -0.33, -0.33,  0.33, 0.33, -0.33,
   -0.33, -0.33, -0.33,  0.33, 0.33, -0.33,  -0.33, 0.33, -0.33,

   // зад
```

```
    -0.33, -0.33, 0.33,  0.33, -0.33, 0.33,  0.33, 0.33, 0.33,
    -0.33, -0.33, 0.33,  0.33, 0.33, 0.33,  -0.33, 0.33, 0.33,

    // лево
    -0.33, -0.33, -0.33,  -0.33, 0.33, -0.33,  -0.33, 0.33, 0.33,
    -0.33, -0.33, -0.33,  -0.33, 0.33, 0.33,  -0.33, -0.33, 0.33,

    // право
    0.33, -0.33, -0.33,  0.33, 0.33, -0.33,  0.33, 0.33, 0.33,
    0.33, -0.33, -0.33,  0.33, 0.33, 0.33,  0.33, -0.33, 0.33,

    // низ
    -0.33, -0.33, -0.33,  0.33, -0.33, -0.33,  0.33, -0.33, 0.33,
    -0.33, -0.33, -0.33,  0.33, -0.33, 0.33,  -0.33, -0.33, 0.33,

    // верх
    -0.33, 0.33, -0.33,  0.33, 0.33, -0.33,  0.33, 0.33, 0.33,
    -0.33, 0.33, -0.33,  0.33, 0.33, 0.33,  -0.33, 0.33, 0.33
];

var cubeColors = [
    // перед (красный)
    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, 0.0,

    // зад (зелёный)
    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,

    // лево (синий)
    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, 0.0, 1.0,

    // право (жёлтый)
    1.0, 1.0, 0.0,  1.0, 1.0, 0.0,  1.0, 1.0, 0.0,
    1.0, 1.0, 0.0,  1.0, 1.0, 0.0,  1.0, 1.0, 0.0,

    // низ (пурпурный)
    1.0, 0.0, 1.0,  1.0, 0.0, 1.0,  1.0, 0.0, 1.0,
    1.0, 0.0, 1.0,  1.0, 0.0, 1.0,  1.0, 0.0, 1.0,

    // верх (голубой)
    0.0, 1.0, 1.0,  0.0, 1.0, 1.0,  0.0, 1.0, 1.0,
    0.0, 1.0, 1.0,  0.0, 1.0, 1.0,  0.0, 1.0, 1.0
];

var pyramidVertices = [
    // Основание (квадратное)
    -0.45, 0.0, -0.45,  0.45, 0.0, -0.45,  0.45, 0.0, 0.45,  -0.45, 0.0, 0.45,

    // Боковые грани
    // перед
    0.0, 0.9, 0.0,  -0.45, 0.0, -0.45,  0.45, 0.0, -0.45,
    // право
    0.0, 0.9, 0.0,   0.45, 0.0, -0.45,  0.45, 0.0, 0.45,
    // зад
    0.0, 0.9, 0.0,   0.45, 0.0, 0.45,  -0.45, 0.0, 0.45,
```

```
   // лево
   0.0, 0.9, 0.0,  -0.45, 0.0, 0.45,  -0.45, 0.0, -0.45
];

var pyramidColors = [
   // Основание (оранжевый)
   1.0, 0.5, 0.0,  1.0, 0.5, 0.0,  1.0, 0.5, 0.0,  1.0, 0.5, 0.0,

   // Боковые грани
   // перед (зелёный)
   0.0, 1.0, 0.0,  0.0, 1.0, 0.0,  0.0, 1.0, 0.0,
   // право (красный)
   1.0, 0.0, 0.0,  1.0, 0.0, 0.0,  1.0, 0.0, 0.0,
   // зад (синий)
   0.0, 0.0, 1.0,  0.0, 0.0, 1.0,  0.0, 0.0, 1.0,
   // лево (жёлтый)
   1.0, 1.0, 0.0,  1.0, 1.0, 0.0,  1.0, 1.0, 0.0
];

function createBuffer(data) {
   var buffer = gl.createBuffer();
   gl.bindBuffer(gl.ARRAY_BUFFER, buffer);
   gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(data), gl.STATIC_DRAW);
   return buffer;
}

var cubeBuffer = createBuffer(cubeVertices);
var pyramidBuffer = createBuffer(pyramidVertices);

var cubeColorBuffer = createBuffer(cubeColors);
var pyramidColorBuffer = createBuffer(pyramidColors);

var vertCode = `
   attribute vec3 coordinates;
   attribute vec3 color;
   varying vec3 vColor;
   varying vec3 vPosition;
   uniform mat4 modelMatrix;
   void main(void) {
      gl_Position = modelMatrix * vec4(coordinates, 1.0);
      vColor = color;
      vPosition = coordinates;
   }
`;

var vertShader = gl.createShader(gl.VERTEX_SHADER);
gl.shaderSource(vertShader, vertCode);
gl.compileShader(vertShader);

var fragCode = `
   precision mediump float;
   varying vec3 vColor;
   varying vec3 vPosition;
   void main(void) {
      float gradient = vPosition.z + 0.9;
      gl_FragColor = vec4(vColor * gradient, 1.0);
   }
```

```
`;

var fragShader = gl.createShader(gl.FRAGMENT_SHADER);
gl.shaderSource(fragShader, fragCode);
gl.compileShader(fragShader);

var shaderProgram = gl.createProgram();
gl.attachShader(shaderProgram, vertShader);
gl.attachShader(shaderProgram, fragShader);
gl.linkProgram(shaderProgram);
gl.useProgram(shaderProgram);

var coordLocation = gl.getAttribLocation(shaderProgram, "coordinates");
gl.enableVertexAttribArray(coordLocation);
var colorLocation = gl.getAttribLocation(shaderProgram, "color");
gl.enableVertexAttribArray(colorLocation);

var cubeModelMatrix = mat4.create();
mat4.identity(cubeModelMatrix);
var cubeModelMatrixLocation = gl.getUniformLocation(shaderProgram, "modelMatrix");

var pyramidModelMatrix = mat4.create();
mat4.identity(pyramidModelMatrix);
var pyramidModelMatrixLocation = gl.getUniformLocation(shaderProgram, "modelMatrix");

function drawCube(vertexBuffer, colorBuffer, vertexCount) {
   gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
   gl.vertexAttribPointer(coordLocation, 3, gl.FLOAT, false, 0, 0);

   gl.bindBuffer(gl.ARRAY_BUFFER, colorBuffer);
   gl.vertexAttribPointer(colorLocation, 3, gl.FLOAT, false, 0, 0);

   gl.drawArrays(gl.TRIANGLES, 0, vertexCount);
}

function drawPyramid(vertexBuffer, colorBuffer, vertexCount) {
   gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
   gl.vertexAttribPointer(coordLocation, 3, gl.FLOAT, false, 0, 0);

   gl.bindBuffer(gl.ARRAY_BUFFER, colorBuffer);
   gl.vertexAttribPointer(colorLocation, 3, gl.FLOAT, false, 0, 0);

   gl.drawArrays(gl.TRIANGLE_FAN, 0, 4);
   gl.drawArrays(gl.TRIANGLES, 4, vertexCount - 4);
}

gl.clearColor(0.0, 0.0, 0.0, 1.0);
gl.enable(gl.DEPTH_TEST);
gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
gl.viewport(0, 0, canvas.width, canvas.height);

function updateView() {
   const xPosCube = parseFloat(document.getElementById('xPosCube').value);
   const yPosCube = parseFloat(document.getElementById('yPosCube').value);
   const zPosCube = parseFloat(document.getElementById('zPosCube').value);
   const xScaleCube = parseFloat(document.getElementById('xScaleCube').value);
   const yScaleCube = parseFloat(document.getElementById('yScaleCube').value);
```

```
   const xRotateCube = parseFloat(document.getElementById('xRotateCube').value) *
Math.PI / 180;
   const yRotateCube = parseFloat(document.getElementById('yRotateCube').value) *
Math.PI / 180;
   const zRotateCube = parseFloat(document.getElementById('zRotateCube').value) *
Math.PI / 180;

   mat4.identity(cubeModelMatrix);
   mat4.translate(cubeModelMatrix, cubeModelMatrix, [xPosCube, yPosCube, zPosCube]);
   mat4.rotateX(cubeModelMatrix, cubeModelMatrix, xRotateCube);
   mat4.rotateY(cubeModelMatrix, cubeModelMatrix, yRotateCube);
   mat4.rotateZ(cubeModelMatrix, cubeModelMatrix, zRotateCube);
   mat4.scale(cubeModelMatrix, cubeModelMatrix, [xScaleCube, yScaleCube, 1]);

   const xPosPyramid = parseFloat(document.getElementById('xPosPyramid').value);
   const yPosPyramid = parseFloat(document.getElementById('yPosPyramid').value);
   const zPosPyramid = parseFloat(document.getElementById('zPosPyramid').value);
   const xScalePyramid = parseFloat(document.getElementById('xScalePyramid').value);
   const yScalePyramid = parseFloat(document.getElementById('yScalePyramid').value);
   const xRotatePyramid = parseFloat(document.getElementById('xRotatePyramid').value) *
Math.PI / 180;
   const yRotatePyramid = parseFloat(document.getElementById('yRotatePyramid').value) *
Math.PI / 180;
   const zRotatePyramid = parseFloat(document.getElementById('zRotatePyramid').value) *
Math.PI / 180;

   mat4.identity(pyramidModelMatrix);
   mat4.translate(pyramidModelMatrix, pyramidModelMatrix, [xPosPyramid, yPosPyramid,
zPosPyramid]);
   mat4.rotateX(pyramidModelMatrix, pyramidModelMatrix, xRotatePyramid);
   mat4.rotateY(pyramidModelMatrix, pyramidModelMatrix, yRotatePyramid);
   mat4.rotateZ(pyramidModelMatrix, pyramidModelMatrix, zRotatePyramid);
   mat4.scale(pyramidModelMatrix, pyramidModelMatrix, [xScalePyramid, yScalePyramid,
1]);

   gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

   gl.uniformMatrix4fv(cubeModelMatrixLocation, false, cubeModelMatrix);
   drawCube(cubeBuffer, cubeColorBuffer, cubeVertices.length / 3);

   gl.uniformMatrix4fv(pyramidModelMatrixLocation, false, pyramidModelMatrix);
   drawPyramid(pyramidBuffer, pyramidColorBuffer, pyramidVertices.length / 3);
}

document.getElementById('xPosCube').addEventListener('input', updateView);
document.getElementById('yPosCube').addEventListener('input', updateView);
document.getElementById('zPosCube').addEventListener('input', updateView);
document.getElementById('xScaleCube').addEventListener('input', updateView);
document.getElementById('yScaleCube').addEventListener('input', updateView);
document.getElementById('xRotateCube').addEventListener('input', updateView);
document.getElementById('yRotateCube').addEventListener('input', updateView);
document.getElementById('zRotateCube').addEventListener('input', updateView);

document.getElementById('xPosPyramid').addEventListener('input', updateView);
document.getElementById('yPosPyramid').addEventListener('input', updateView);
document.getElementById('zPosPyramid').addEventListener('input', updateView);
document.getElementById('xScalePyramid').addEventListener('input', updateView);
```

```
document.getElementById('yScalePyramid').addEventListener('input', updateView);
document.getElementById('xRotatePyramid').addEventListener('input', updateView);
document.getElementById('yRotatePyramid').addEventListener('input', updateView);
document.getElementById('zRotatePyramid').addEventListener('input', updateView);

updateView();
```