Санкт-Петербургский Государственный
Электротехнический Университет

Кафедра МОЭВМ

Задание для лабораторной работы № 2
"3D трансформации"

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

Санкт-Петербург
2025 г.

Цель работы.

- освоить использование стандартных матричных преобразований над 3D объектами.
- проанализировать полученное задание, выделить информационные объекты и действия.
- добавить в программу просмотр трансформаций через lookAt, трансформацию проекции через perspective и ortho и моделирование трансформации rotate, translate, scale и матричный стек.

Основные теоретические положения.

Основные параметры сцены:

- Расположение объектов
- Расположение источников освещения
- Расположение камеры

Простейшие аффинные преобразования/манипуляции:

- Перемещение ($x:=x+\Delta x$; $y:=y+\Delta y$; $z:=z+\Delta z$)
- Поворот (вокруг z: $x' := x \cos\alpha + y \sin\alpha$; $y' := -x \sin\alpha + y \cos\alpha$; $z' := z$)
- Масштабирование ($x := k_x x$)
- Отражение по оси: *(-1)

Некоторые преобразования приходится делать в несколько этапов.

Виды координат:

- декартовы (x, y, z)
- обобщённые (x, y, z, w) — позволяют задать бесконечно удалённые точки (при w=0)

Связь:

$(x, y, z, w)_{об} = (x_{дек}, y_{дек}, z_{дек}, 1)$

$(x, y, z)_{дек} = (x_{об}, y_{об}, z_{об}) / w$ — не всегда возможно

Аффинное преобразование — преобразование, которое можно задать в виде:

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \underbrace{A}_{\text{масштаб (поворот)}} \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + \underbrace{\begin{pmatrix} c_x \\ c_y \end{pmatrix}}_{\text{сдвиг}} \qquad \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} a_{11} & a_{12} & c_x \\ a_{21} & a_{22} & c_y \\ 0 & 0 & 1 \end{pmatrix}}_{M} \begin{pmatrix} x_0 \\ y_0 \\ 1 \end{pmatrix}$$

Комбинацию аффинных преобразований можно представить, как одно аффинное преобразование, матрица которого есть произведение матриц комбинируемых преобразований:

- $x_1 = M_1 x_0$
- $x_2 = M_2 x_1$
- …
- $x_n = M_n x_{n-1} = M_n(M_{n-1} x_{n-2}) = M_n(M_{n-1}(M_{n-2} x_{n-3})) = \ldots = (M_n M_{n-1} \ldots M_1) x_0 = M x_0$

Координаты бывают:

1. Мировые (точка в пространстве).
2. Видовые (получаются путём передвижения мировых).
3. Проекционные (относительно плоскости проецирования, Z — расстояние до объекта).
4. Экранные (координаты пикселей).

Поверхность проецирования — поверхность, на которой строится изображение (обычно плоскость).

Проецирование — процесс (задача) сопоставления каждой точке трёхмерной сцены некоторой точки поверхности проецирования.

Луч проецирования — луч, который идет из объекта к поверхности проецирования.

Виды проецирования:

- Параллельное: все лучи проецирования параллельны друг другу.

  Подвиды:

  Косоугольное: лучи не перпендикулярны плоскости проецирования.

  Аксонометрическое: лучи перпендикулярны плоскости проецирования. Если использовать систему координат (x,y,z), в которой z — это расстояние от поверхности проецирования до объекта, то все лучи будут параллельны оси Z(hi, co). В зависимости от соотношения коэффициентов масштабирования sx, sy, sz (по осям X,Y,Z соответственно) выделяют следующие модификации:

  - Изометрическое: $sx = sy = sz$.
  - Диметрическое: $sx = sy \neq sz$.
  - Триметрическое: $sx \neq sy \neq sz$.
- Перспективное: все лучи проецирования проходят через одну точку, они не параллельны. Чем меньше фокусное расстояние, тем меньше размер проекции на плоскости.

С фокусным расстоянием связано увеличение и угол обзора. Чем меньше фокусное расстояние, тем больше угол обзора, и наоборот.

Каждая вершина в сцене проходит две основные стадии трансформации:

- Преобразование вида модели (перемещение, вращение и масштабирование объектов, преобразование трехмерного просмотра).
- Проекция (перспективная или орфографическая).

Элементарные преобразования:

Правосторонняя и левосторонняя система координат:

Правосторонняя система координат используется чаще всего. В OpenGL как локальная система координат для объектных моделей (таких как куб, сфера), так и система координат камеры используют правостороннюю систему.

Перенос – где [dx, dy, dz] – вектор переноса.

В результате вызова этой функции создается матрица переноса, определяемая параметрами [dx, dy, dz], которые необходимо объединить с матрицей представления глобальной модели:

$M_{\text{модель зрения}} = M_{\text{модель зрения}} * T(\text{дх, ду, дз});$

Где T (dx, dy, dz)= $\begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$

В общем, новая матрица преобразования всегда присоединяется к глобальной матрице справа.

Поворот – объект способен вращаться на angle в градусах вокруг произвольного вектора. Однако часто легче вращать только одну из основных осей:

ось x: vec3 (1,0,0)

ось y: vec3 (0,1,0)

ось z: vec3 (0,0,1)

Эти простые вращения затем объединяются для получения произвольного желаемого вращения. Например:

mv = mult(mv, rotate(20,vec3(0,1,0))); // Поверните на 20 градусов против часовой

//стрелки вокруг оси Y

$M_{\text{модельное представление}} = M_{\text{модельное представление}} * R_x(a)$; где $R_x(a)$ обозначает матрицу вращения вокруг оси x для степени a:

$$R_x(a) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(a) & -\sin(a) & 0 \\ 0 & \sin(a) & \cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

mv = mult (mv, rotateY (a)); // вращение вокруг оси y

mv = mult (mv, rotateZ (a)); // вращение вокруг оси z

Масштабирование – где sx, sy и sz – коэффициенты масштабирования вдоль каждой оси относительно локальной системы координат модели. Преобразование масштабирования позволяет матрице преобразования изменять размеры объекта путем сжатия или растяжения по основным осям с центром в

начале координат.

Следует отметить, что масштабирование всегда связано с началом вдоль каждого измерения с соответствующими коэффициентами масштабирования. Это означает, что если масштабируемый объект не перекрывает источник, он будет перемещаться дальше, если его масштабировать, и ближе, если он уменьшен

Эффект объединения полученной матрицы в матрицу представления глобальной модели аналогичен перемещению и повороту.

Порядок трансформаций:

Когда вы проводите умножение преобразований, как мы делаем, и как это делается в классическом OpenGL, порядок применения преобразований противоположен порядку, в котором они появляются в программе. Другими словами, последнее указанное преобразование является первым примененным. Это свойство иллюстрируется следующими примерами.

Исходное положение камеры по умолчанию – в начале координат, а объектив смотрит в отрицательном направлении z.

Большинство объектных моделей, таких как кубы или сферы, также определены в начале координат с размером по умолчанию.

Цель преобразования вида модели состоит в том, чтобы позволить пользователю переориентировать и изменить размеры этих объектов и разместить их в любом желаемом месте, а также упростить расположение их относительно друг друга.

Трансформации модели и трансформация вида:

Функция lookAt() позволяет легко перемещать точки «из» и «в» линейно. Например, если вам нужно перемещаться вдоль стены здания, расположенного вдали от начала координат и выровненного по осям, вы можете просто взять точку «до» в качестве одного угла здания и рассчитать «от» как постоянное расстояние от точки "до". Для перемещения по зданию просто измените точку «до».

Функция lookAt() определяет преобразование просмотра

mat4 lookAt (vec3 eye, vec3 at, vec3 up), параметры

- eye: указывает положение точки глаза
- at: указывает положение контрольной точки
- up: указывает направление вектора вверх

Компоненты матрицы моделирования. Вся эта путаница с преобразованием моделирования и просмотра проистекает из того факта, что мы имитируем классический OpenGL, который использует одну матрицу для представления всех матриц преобразования, как моделирования, так и просмотра, для всего, что рисуется – представление Modelview. Преобразование, используемое для описания модели, и преобразование, используемое для описания местоположения и ориентации точки обзора, сосуществуют в этой одной матрице. Умножение матриц не является коммутативным, а скорее ассоциативным, что означает, что

произведение ((AB) C) такое же, как (A (BC)). Таким образом, матрица OpenGL Modelview является логически продуктом матрицы просмотра и матрицы моделирования.

$M_{modelview} = M_{просмотр} * M_{моделирование}$

Это означает, что ваши преобразования просмотра должны быть введены в матрицу Modelview до моделирования преобразований.

Задание.

Разработать программу, реализующую представление куба и пирамиды.

Разработанная программа должна быть пополнена возможностями просмотра трансформаций через lookAt, трансформации проекции через perspective и ortho и моделирования трансформации rotate, translate, scale и матричный стек.

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 134.0.3124.83.

В интерфейсе был создан новый div блок для управления параметрами сцены. Выбор типы проекции perspective или ortho реализован через выпадающий список select. По умолчанию проекция ortho, но если выбрать perspective, то появляется под выпадающим списком три ползунка с регулировкой перспективы: FOV, Near и Far. Для регулировки lookAt трансформацией созданы девять ползунков для регулировки позицией камеры, точки, на которую она смотрит, и вектора, определяющего верх камеры.

Создан матричный стек matrixStack для хранения матриц в связи с преобразованиями. В качестве функций выступают pushMatrix() для сохранения

матриц и popMatrix() для извлечения. Созданы три матрицы: modelViewMatrix для матрицы моделирования, projectionMatrix для матрицы проекций и normalMatrix для матрицы преобразования нормалей. Матрицы также добавлены в вершинный шейдер. Фигуры благодаря стеку используют теперь матрицы с одним названием. Получаем uniform-переменные из шейдерной программы методом getUniformLocation(), через которую будут передаваться матрицы преобразований, в переменные modelViewMatrixLocation, projectionMatrixLocation и normalMatrixLocation.

Для обновления проекций создана функция updateProjection(), она вызывается из updateView(). В ней определяется отображение скрытого блока управления для перспективной проекции, если выбран соответствующий тип. Матрица проекций инициализируется и получает значения с использованием функций mat4.perspective() и mat4.ortho() в зависимости от ползунков. Затем обновляется uniform-переменная в шейдере.

Как было сказано ранее, в функции updateView() вызывается функция updateProjection(). В функцию добавлена трансформация lookAt. Переменные eye, center и up регулируются с ползунков. Матрицу вида применяем к матрице моделирования и сохраняем. Все изменения с фигурами теперь происходят со стеком, поэтому используются матрицы modelViewMatrix и normalMatrix. Трансформации rotate, translate, scale были разработаны в прошлой лабораторной работе. Были добавлены матрицы преобразования нормалей для будущего использования. Матрица нормалей получается с помощью последовательного выполнения функций mat4.copy, mat4.invert и mat4.transpose, начиная с копирования modelViewMatrix. Матрица моделирования возвращается к сохраненному состоянию после отрисовки одной из фигур и приступлением к работе над следующей.

Упрощено слежение за изменениями ползунков с помощью использования querySelectorAll для input. Для projectionType добавлен свой обработчик событий change для запуска updateView().

Тестирование.

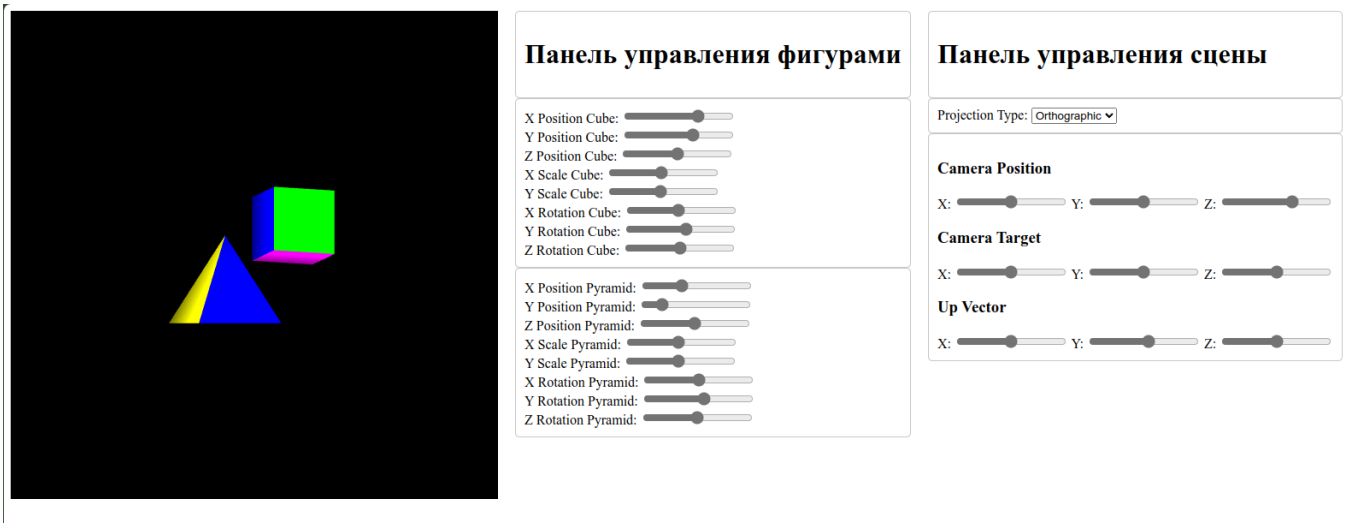

Рисунок 1 – Сцена с кубом и пирамидой с панелями управления и ortho проекцией

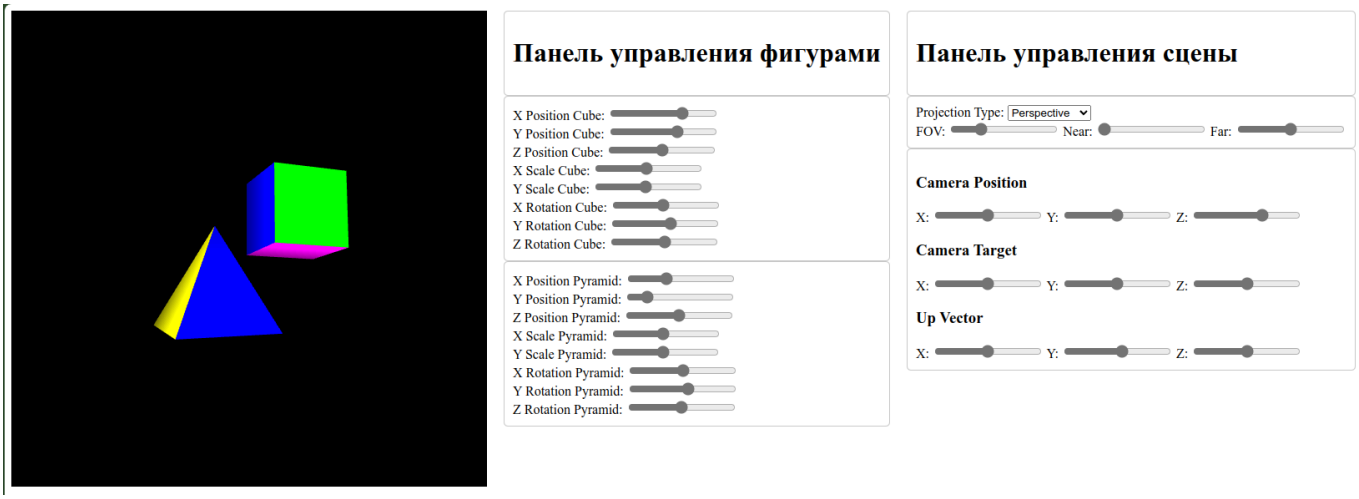

Рисунок 2 – Сцена с кубом и пирамидой с панелями управления и perspective проекцией

Вывод.

В результате выполнения лабораторной работы была разработана программа, отображающая геометрические объекты: куб и пирамида вместе с панелями управления фигурами и сценой. В них входят трансформации через lookAt, трансформации проекции через perspective и ortho и моделирование трансформации rotate, translate, scale с матричным стеком. 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 фигурами</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 class="controls">
                <div class="title-control">
                    <h1>Панель управления сцены</h1>
                </div>
                <div class="projection-control">
                    <div class="projection-type">
                        <label>Projection Type:</label>
                        <select id="projectionType">
                            <option value="ortho">Orthographic</option>
                            <option value="perspective">Perspective</option>
                        </select>
                    </div>
                    <div class="perspective-controls">
                        <label>FOV:</label>
                        <input type="range" id="fov" min="0" max="180" step="1"
value="45">
                        <label>Near:</label>
                        <input type="range" id="near" min="0.1" max="10" step="0.1"
value="0.1">
                        <label>Far:</label>
                        <input type="range" id="far" min="1" max="200" step="1"
value="100">
                    </div>
                </div>
                <div class="lookat-control">
                    <h3>Camera Position</h3>
                    <label>X: <input type="range" id="eyeX" min="-15" max="15"
step="0.1" value="0"></label>
```

```
                    <label>Y: <input type="range" id="eyeY" min="-15" max="15"
step="0.1" value="0"></label>
                    <label>Z: <input type="range" id="eyeZ" min="-15" max="15"
step="0.1" value="5"></label>
                    <h3>Camera Target</h3>
                    <label>X: <input type="range" id="centerX" min="-5" max="5"
step="0.1" value="0"></label>
                    <label>Y: <input type="range" id="centerY" min="-5" max="5"
step="0.1" value="0"></label>
                    <label>Z: <input type="range" id="centerZ" min="-5" max="5"
step="0.1" value="0"></label>
                    <h3>Up Vector</h3>
                    <label>X: <input type="range" id="upX" min="-10" max="10"
step="0.1" value="0"></label>
                    <label>Y: <input type="range" id="upY" min="-10" max="10"
step="0.1" value="1"></label>
                    <label>Z: <input type="range" id="upZ" min="-10" max="10"
step="0.1" value="0"></label>
                </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, .projection-control, .lookat-control, .cube-control,
.pyramid-control {
            border: 1px solid #ccc;
            padding: 10px;
            border-radius: 5px;

        }
    </style>
</html>
```

WebGLJavascript.js

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

const 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
];

const 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
];

const 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
];

const 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
];

let matrixStack = [];
function pushMatrix(m) {
   matrixStack.push(mat4.clone(m));
}
function popMatrix() {
   if (matrixStack.length === 0) {
       throw "Ошибка: Стек матриц пуст.";
   }
   return matrixStack.pop();
}
let modelViewMatrix = mat4.create();
let projectionMatrix = mat4.create();
let normalMatrix = mat4.create();

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

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

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

const vertCode = `
   attribute vec3 coordinates;
   attribute vec3 color;
   varying vec3 vColor;
   varying vec3 vPosition;

   uniform mat4 modelViewMatrix;
   uniform mat4 projectionMatrix;
   uniform mat4 normalMatrix;

   void main(void) {
      gl_Position = projectionMatrix * modelViewMatrix * vec4(coordinates, 1.0);
      vColor = color;
      vPosition = coordinates;
   }
`;
```

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

const 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);
   }
`;

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

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

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

let modelViewMatrixLocation = gl.getUniformLocation(shaderProgram, "modelViewMatrix");
let projectionMatrixLocation = gl.getUniformLocation(shaderProgram,
"projectionMatrix");
let normalMatrixLocation = gl.getUniformLocation(shaderProgram, "normalMatrix");

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 updateProjection() {
   let aspect = canvas.width / canvas.height;
   let projectionType = document.getElementById('projectionType').value;

   let perspectiveControls = document.querySelector('.perspective-controls');
   perspectiveControls.style.display = (projectionType === 'perspective') ? 'block' :
'none';

   mat4.identity(projectionMatrix);

   if (projectionType === 'perspective') {
      let fov = parseFloat(document.getElementById('fov').value) * Math.PI / 180;
      let near = parseFloat(document.getElementById('near').value);
      let far = parseFloat(document.getElementById('far').value);
      mat4.perspective(projectionMatrix, fov, aspect, near, far);
   } else {
      let distance = parseFloat(document.getElementById('eyeZ').value);
      mat4.ortho(projectionMatrix, -aspect * distance / 2, aspect * distance / 2,
-distance / 2, distance / 2, 0.1, 100.0);
   }

   gl.uniformMatrix4fv(projectionMatrixLocation, false, projectionMatrix);
}

function updateView() {
   gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
   gl.viewport(0, 0, canvas.width, canvas.height);

   updateProjection();

   let eye = [
      parseFloat(document.getElementById('eyeX').value),
      parseFloat(document.getElementById('eyeY').value),
      parseFloat(document.getElementById('eyeZ').value)
   ];

   let center = [
      parseFloat(document.getElementById('centerX').value),
      parseFloat(document.getElementById('centerY').value),
      parseFloat(document.getElementById('centerZ').value)
   ];

   let up = [
      parseFloat(document.getElementById('upX').value),
      parseFloat(document.getElementById('upY').value),
      parseFloat(document.getElementById('upZ').value)
   ];

   let viewMatrix = mat4.create();
   mat4.lookAt(viewMatrix, eye, center, up);

   mat4.identity(modelViewMatrix);
   mat4.copy(modelViewMatrix, viewMatrix);
```

```
   pushMatrix(modelViewMatrix);

   let xPosCube = parseFloat(document.getElementById('xPosCube').value);
   let yPosCube = parseFloat(document.getElementById('yPosCube').value);
   let zPosCube = parseFloat(document.getElementById('zPosCube').value);
   let xScaleCube = parseFloat(document.getElementById('xScaleCube').value);
   let yScaleCube = parseFloat(document.getElementById('yScaleCube').value);
   let xRotateCube = parseFloat(document.getElementById('xRotateCube').value) * Math.PI
/ 180;
   let yRotateCube = parseFloat(document.getElementById('yRotateCube').value) * Math.PI
/ 180;
   let zRotateCube = parseFloat(document.getElementById('zRotateCube').value) * Math.PI
/ 180;

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

   mat4.copy(normalMatrix, modelViewMatrix);
   mat4.invert(normalMatrix, normalMatrix);
   mat4.transpose(normalMatrix, normalMatrix);

   gl.uniformMatrix4fv(modelViewMatrixLocation, false, modelViewMatrix);
   gl.uniformMatrix4fv(normalMatrixLocation, false, normalMatrix);

   drawCube(cubeBuffer, cubeColorBuffer, cubeVertices.length / 3);

   modelViewMatrix = popMatrix();

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

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

   mat4.copy(normalMatrix, modelViewMatrix);
   mat4.invert(normalMatrix, normalMatrix);
   mat4.transpose(normalMatrix, normalMatrix);

   gl.uniformMatrix4fv(modelViewMatrixLocation, false, modelViewMatrix);
   gl.uniformMatrix4fv(normalMatrixLocation, false, normalMatrix);
```

```
   drawPyramid(pyramidBuffer, pyramidColorBuffer, pyramidVertices.length / 3);
}


document.getElementById('projectionType').addEventListener('change', updateView);
document.querySelectorAll('input').forEach(input => {
   input.addEventListener('input', updateView);
});

updateView();
```