Санкт-Петербургский Государственный
Электротехнический Университет

Кафедра МОЭВМ

Задание для лабораторной работы № 4
"Текстуры изображения"

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

Санкт-Петербург
2025 г.

Цель работы.

- освоить текстурирование 3D объектов.
- проанализировать полученное задание, выделить информационные объекты и действия.
- отредактировать свою собственную картинку и сделайте ее текстурной картой (3 разных размера), разложить текстуру по изображению, закрепить и отцентрировать свое изображение на объекте.

Основные теоретические положения.

Модуль текстуры, также называемый модулем отображения текстуры (TMU) или модулем обработки текстуры (TPU), является аппаратным компонентом в графическом процессоре, который выполняет выборку. Выборка - это процесс вычисления цвета по текстуре изображения и координатам текстуры.

Объект текстуры – это структура данных, которая содержит данные о цвете для текстуры изображения и, возможно, для набора мип-карт для текстуры, а также значений свойств текстуры, таких как фильтры минимизации и увеличения и режим повтора текстуры. Текстурный модуль должен получить доступ к текстурному объекту, чтобы выполнить свою работу. Единица текстуры – процессор; объект текстуры содержит данные, которые обрабатываются

GLSL поиск текстуры выполняется с помощью переменных сэмплера.

Переменная сэмплера – это переменная в шейдерной программе типа sampler2D или samplerCube. Sampler2D используется, чтобы сделать поиск в стандартном изображении текстуры; samplerCube используется, чтобы сделать поиск в cubemap текстуры Значение переменной сэмплера является ссылкой на единицу текстуры. Значение указывает, какая единица текстуры вызывается, когда переменная сэмплера используется для поиска текстуры. Переменные сэмплера

должны быть объявлены как глобальные однородные переменные. Не допустимо, чтобы шейдерная программа присваивала значение переменной сэмплера. Значение должно исходить из стороны JavaScript.

На стороне JavaScript доступные текстурные блоки пронумерованы 0, 1, 2, ..., где максимальное значение зависит от реализации. Количество единиц может быть определено как значение выражения

gl.getParameter (gl.MAX_COMBINED_TEXTURE_IMAGE_UNITS)

Что касается JavaScript, то значение переменной сэмплера является целым числом. Если вы хотите, чтобы в переменной сэмплера использовалась единица текстуры 2, установите значение переменной сэмплера на 2. Это можно сделать с помощью функции gl.uniform1i.

Чтобы использовать текстуру изображения, вам также необходимо создать текстурный объект и загрузить изображение в текстурный объект.

Возможно, вы захотите установить некоторые свойства объекта текстуры, и вы можете создать набор mip-карт для текстуры. И вам придется связать текстурный объект с текстурным блоком. Все это делается на стороне JavaScript.

Команда для создания объекта текстуры – gl.createTexture(). Он создает один объект текстуры и возвращает ссылку на него. Например,

textureObj = gl.createTexture();

Это просто выделяет некоторую память для объекта. Чтобы использовать его, вы должны сначала «связать» объект текстуры, вызвав gl.bindTexture. Например,

gl.bindTexture (gl.TEXTURE_2D, textureObj);

Первый параметр, gl.TEXTURE_2D, является целью текстуры. Эта цель используется для работы с обычным текстурным изображением. Существует другая цель для текстур кубической карты.

Функция gl.texImage2D используется для загрузки изображения в текущий связанный объект текстуры. Но помните, что эта команда и другие команды всегда применяются к текущему связанному объекту текстуры. Объект текстуры не упоминается в команде; вместо этого объект текстуры должен быть связан до вызова команды.

Вы также должны указать текстурному блоку использовать объект текстуры. Прежде чем вы сможете это сделать, вам нужно сделать текстурный блок «активным», что делается путем вызова функции gl.activeTexture. Параметр является одной из констант gl.TEXTURE0, gl.TEXTURE1, gl.TEXTURE2, ..., которые представляют доступные текстурные единицы. (Значения этих констант не равны 0, 1, 2, ...) Первоначально текстурный блок номер 0 активен. Например, чтобы активировать текстурный блок № 2, используйте

gl.activeTexture (gl.TEXTURE2);

Чтобы связать объект текстуры, когда активен модуль текстуры 2, тогда объект текстуры textureObj связан с модулем текстуры номер 2. Привязка просто сообщает модулю текстуры, какой объект текстуры использовать. То есть блок текстуры 2 будет выполнять обычный поиск текстур с использованием изображения и настроек, которые хранятся в textureObj. Текстурный объект может быть связан с несколькими текстурными единицами одновременно. Тем не менее, данный текстурный блок может иметь только одну связанную TEXTURE_2D за раз.

Изображение можно загрузить в объект текстуры с помощью функции gl.texImage2D. Для использования с WebGL эта функция обычно имеет вид

gl.texImage2D (target, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, изображение);

Цель – .TEXTURE_2D для обычных текстур; Существуют и другие цели для загрузки текстур кубических карт. Второй параметр - это уровень mipmap, который

равен 0 для основного изображения. Хотя можно загружать отдельные mip-карты, это делается редко. Следующие два параметра задают формат текстуры внутри объекта текстуры и в исходном изображении. В WebGL два параметра формата должны иметь одинаковое значение. Поскольку веб-изображения хранятся в формате RGBA, редко требуется что-либо еще. Но вы можете использовать gl.RGB если вам не нужен альфа-компонент. А используя gl.LUMINANCE или gl.LUMINANCE_ALPHA, вы можете преобразовать изображение в оттенки серого. (Яркость является средневзвешенным значением красного, зеленого и синего цветов, которое приблизительно соответствует воспринимаемой яркости цвета.) Четвертый параметр всегда будет gl.UNSIGNED_BYTE, указывая, что цвета в изображении сохраняются с использованием одного байта для каждого компонента цвета. Хотя возможны и другие значения, они не имеют смысла для веб-изображений.

Последний параметр в вызове gl.texImage2D – это изображение. Обычно imag будет DOM-элементом изображения, который был загружен асинхронно с помощью JavaScript. Изображение может также быть <холст> элемент. Это означает, что вы можете рисовать на холсте, используя API-интерфейс 2D-графики HTML-холста, а затем использовать холст в качестве источника изображения текстуры. Вы даже можете сделать это с закадровым холстом, который не виден на веб-странице.

Изображение загружается в объект текстуры, который в данный момент привязан к цели в текущем активном модуле текстуры. Нет текстурного объекта по умолчанию; то есть, если при вызове gl.texImage2D не было привязано ни одной текстуры, возникает ошибка. Активная текстурная единица – это та, которая была выбрана с помощью gl.activeTexture, или это текстурная единица 0, если gl.activeTexture никогда не вызывался. Текстурный объект связан с активным текстурным блоком с помощью gl.bindTexture. Это обсуждалось ранее в этом разделе.

Использование изображений в WebGL осложняется тем, что изображения загружаются асинхронно. То есть команда для загрузки изображения просто запускает процесс загрузки изображения. Вы можете указать функцию обратного вызова, которая будет выполняться после завершения загрузки. Изображение на самом деле не будет доступно для использования до тех пор, пока не будет вызвана функция обратного вызова. При загрузке изображения для использования в качестве текстуры, функция обратного вызова должна загружать изображение в объект текстуры. Часто он также вызывает функцию рендеринга для рисования сцены с изображением текстуры.

Задание.

Разработать программу, реализующую представление куба и пирамиды.

Разработанная программа должна быть пополнена возможностями текстурирования, а также переключением текстур трех размеров.

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 135.0.3179.98.

В интерфейсе был создан новый div блок для управления текстурами. По умолчанию текстурирование не применяется. Если активировать checkbox, то на фигуры наложится текстура. По умолчанию текстура растягивается на всю грань фигуры, однако, активировав checkbox для масштабирования по минимуму отношения размера текстуры и изображения по ширине и высоте, текстура лучше впишется. Также есть три checkbox, которые определяют размер текстуры: 128x128, 256x256 и 512x512.

Были созданы массивы текстурных координат cubeTexCoords и pyramidTexCoords. Расположение текстуры на них соответствует просмотру текстуры отдельно как изображения. Для массивов созданы буферы cubeTexCoordsBuffer и pyramidTexCoordsBuffer. Определено местоположение атрибута “texCoords” в шейдерной программе с помощью метода getAttribLocation(), и включен атрибут enableVertexAttribArray(). Поскольку в предыдущей работе функция отрисовки пирамиды изменилась, ее код стал дублированием функции отрисовки куба. Для оптимизации функции были объединены в одну drawCubeOrPyramid(). В функции drawCubeOrPyramid() был добавлен буфер текстурных координат.

Для создания и загрузки текстуры реализована функция loadTexture(gl, url, width, height, skipPixels = 0, skipRows = 0, scale_min = false). Сразу загружается шесть текстур: три обычные с разными размерами и три с scale_min с разными размерами. В функции создаем объект текстуры и сразу его привязываем к цели. Далее создаем изображение и по его загрузке проводим преобразования над текстурой с помощью функции gl.pixelStorei(). В их числе переворот по Y gl.UNPACK_FLIP_Y_WEBGL, выравнивание gl.UNPACK_ALIGNMENT по 1 байту в строке пикселей, длина строки пикселей gl.UNPACK_ROW_LENGTH по ширине изображения и обрезка изображения gl.UNPACK_SKIP_PIXELS и gl.UNPACK_SKIP_ROWS. Далее создаем canvas для изображения, которое мы центрируем, масштабируем, и загружаем данные в текстуру с помощью gl.texImage2D(). Далее генерируем mip-уровни gl.generateMipmap(). Также применяем фильтрацию с помощью функции gl.texParameteri(). При увеличении текстуры, или gl.TEXTURE_MAG_FILTER, линейная интерполяция. При уменьшении текстуры, или gl.TEXTURE_MIN_FILTER, трилинейная фильтрация. Затем сбрасываем параметры gl.UNPACK_FLIP_Y_WEBGL, gl.UNPACK_ALIGNMENT, gl.UNPACK_ROW_LENGTH,

gl.UNPACK_SKIP_PIXELS и gl.UNPACK_SKIP_ROWS с помощью функции gl.pixelStorei().

Для флага использования текстур и самой текстуры определены местоположения атрибутов useTexture и uTexture с помощью метода getAttribLocation(). В функцию updateView() добавлено считывание значения checkbox useTexture, которое передается в шейдер. Так же считывается checkbox useScaleMin, чтобы в зависимости от активности передать текстуры растянутые определенные по максимуму отношения размера текстуры и изображения по ширине и высоте или определенные по минимуму отношения размера текстуры и изображения по ширине и высоте. Далее активируется текстурный блок с индексом 0 с помощью функции gl.activeTexture(gl.TEXTURE0), затем с помощью функции gl.bindTexture(gl.TEXTURE_2D, currentTexture) связываем текущую выбранную текстуру, причем в зависимости от выбранного размера в одном из трёх checkbox. Также передается индекс текстурного блока в шейдер с помощью gl.uniform1i(uTextureLocation, 0).

Для удобной работы был произведен переход с webgl на webgl2. В результате этого перехода было переписаны шейдеры под webgl2. Для этого была указана версия #version 300 es, а также изменено объявление переменных на in, out. В фрагментном шейдере добавляем семплинг цвета текстуры uTexture с учетом координат vTexCoord и применяем в смешении с освещением, если активирован useTexture.

Тестирование.

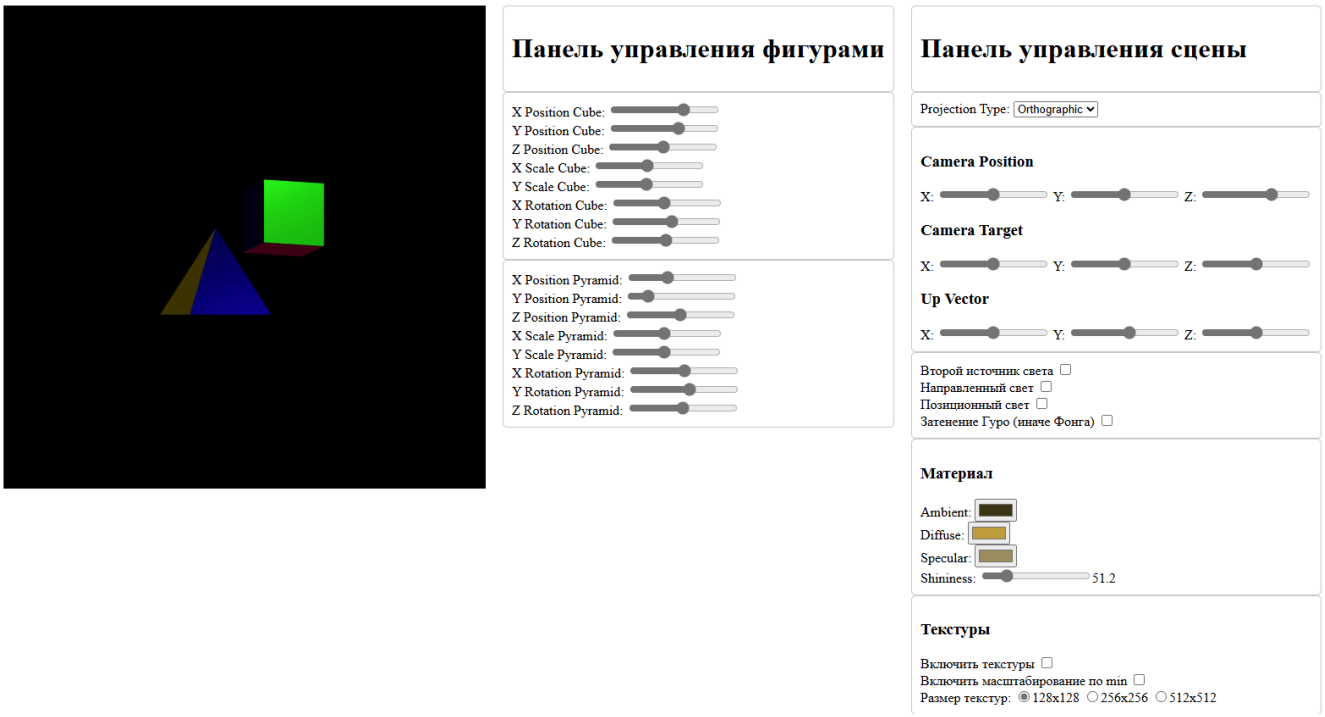

Рисунок 1 – Сцена с кубом и пирамидой без текстур

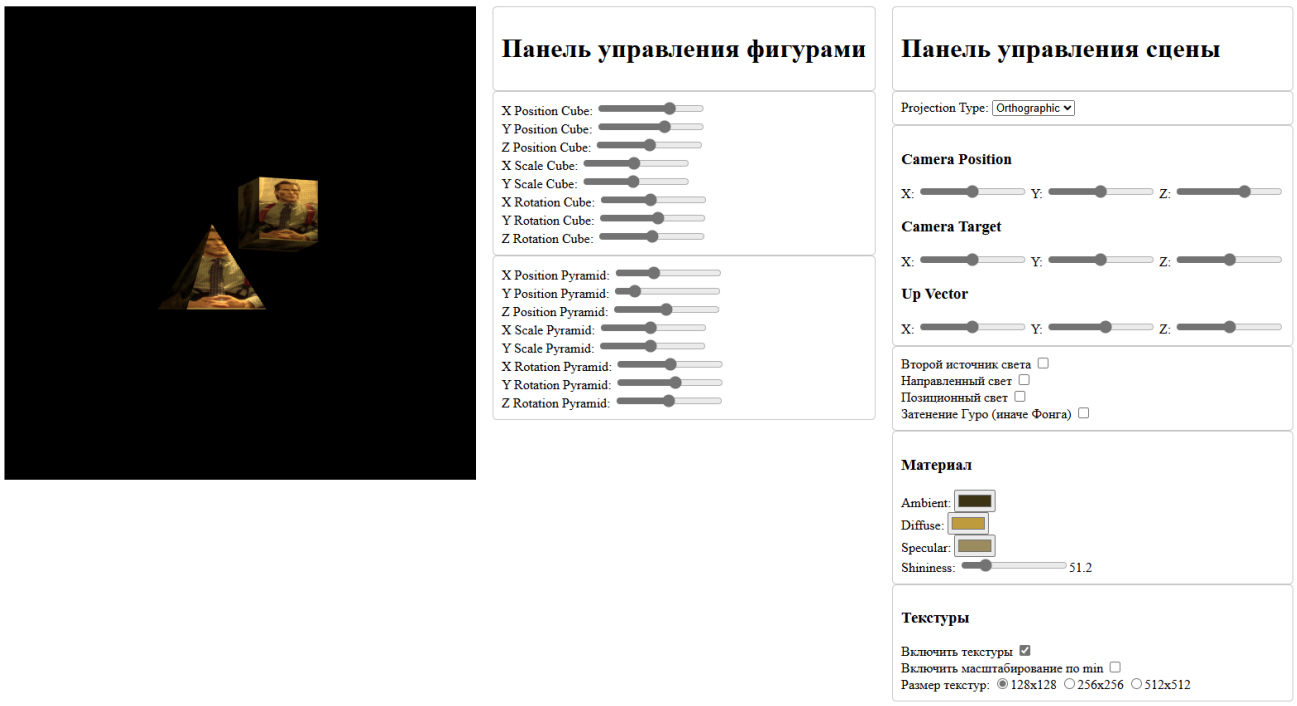

Рисунок 2 – Сцена с кубом и пирамидой с текстурами

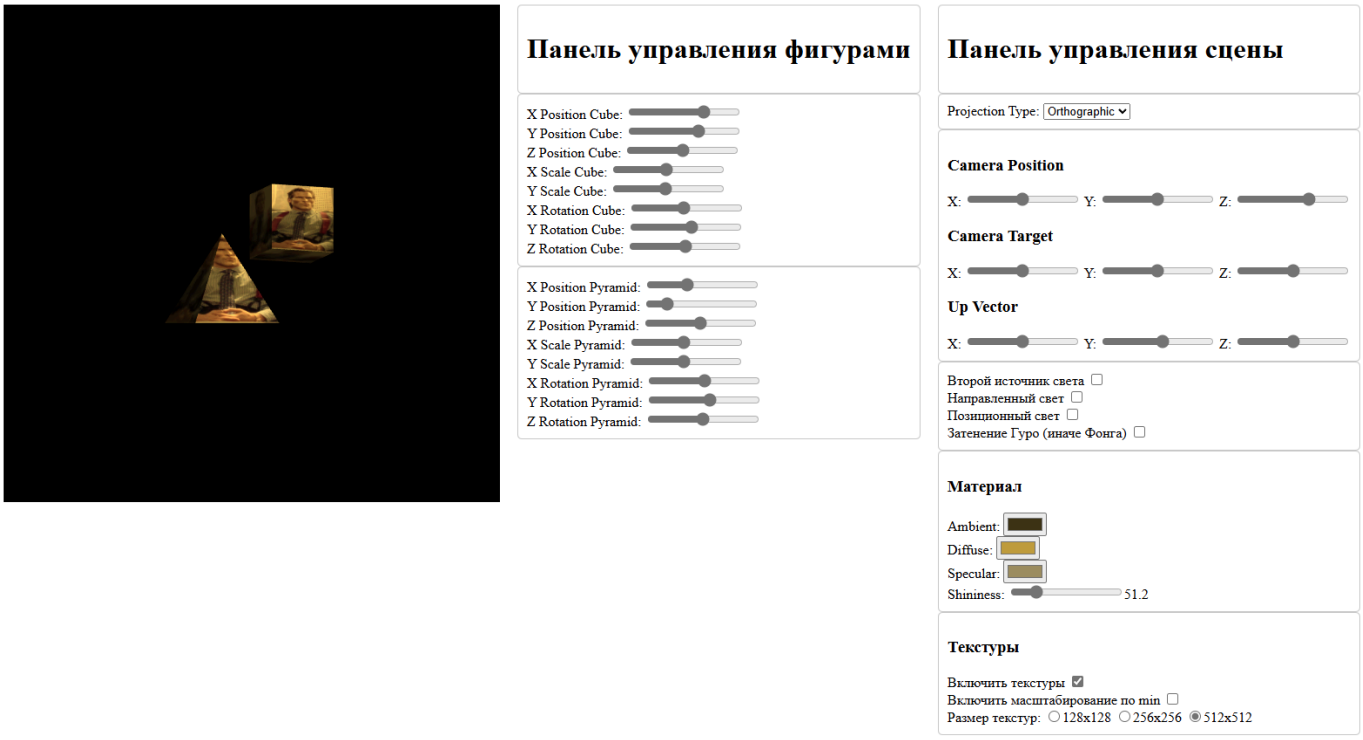

Рисунок 3 – Сцена с кубом и пирамидой с текстурами максимального размера

Вывод.

В результате выполнения лабораторной работы была разработана программа, отображающая геометрические объекты: куб и пирамида вместе с панелями управления фигурами и сценой и текстурированием. В него входит текстуры трех размеров: 128x128, 256x256, 512x512. Также текстуры масштабируются двумя способами: максимум и минимум отношения размера текстуры и изображения по ширине и высоте. Текстуры закреплены за фигурой и отцентрированы. 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 lab4</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 class="light-control">
                    <label>Второй источник света <input type="checkbox"
id="enableLight2"></label><br>
                    <label>Направленный свет <input type="checkbox"
id="enableDirLight"></label><br>
                    <label>Позиционный свет <input type="checkbox"
id="enablePointLight"></label><br>
                    <label>Затенение Гуро (иначе Фонга) <input type="checkbox"
id="gouraudToggle"></label>
                </div>
                <div class="material-control" id="material-control">
                    <h3>Материал</h3>
                    <label>Ambient: <input type="color" id="ambientColor"
value="#3F3313"></label><br>
                    <label>Diffuse: <input type="color" id="diffuseColor"
value="#C09B3A"></label><br>
                    <label>Specular: <input type="color" id="specularColor"
value="#A08D5D"></label><br>
                    <label>Shininess: <input type="range" id="shininess" min="1"
max="256" value="51.2"><span id="shininessValue">51.2</span></label>
                </div>
                <div class="texture-control">
                    <h3>Текстуры</h3>
                    <label>Включить текстуры <input type="checkbox"
id="useTexture"></label><br>
                    <label>Включить масштабирование по min <input type="checkbox"
id="useScaleMin"></label><br>
                    <label>Размер текстур: <label><input type="radio"
name="textureSize" value="128" checked>128x128</label>
                    <label><input type="radio" name="textureSize"
value="256">256x256</label>
                    <input type="radio" name="textureSize" value="512">512x512</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, .light-control,
.material-control, .texture-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('webgl2');

const cubeVertices = [
   // зад (красный)
   -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,

   // перед (зеленый)
   -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 (синий)
   -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 (жёлтый)
   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 (пурпурный)
   -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 (голубой)
   -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 = [
   // зад (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,

   // перед (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 cubeTexCoords = [
    // зад (красный)
    1.0, 0.0, 0.0, 1.0, 0.0, 0.0,
    1.0, 0.0, 1.0, 1.0, 0.0, 1.0,

    // перед (зелёный)
    0.0, 0.0, 1.0, 0.0, 1.0, 1.0,
    0.0, 0.0, 1.0, 1.0, 0.0, 1.0,

    // левая грань (синий)
    0.0, 0.0, 1.0, 1.0, 0.0, 1.0,
    0.0, 0.0, 1.0, 0.0, 1.0, 1.0,

    // правая грань (жёлтый)
    1.0, 0.0, 1.0, 1.0, 0.0, 1.0,
    1.0, 0.0, 0.0, 1.0, 0.0, 0.0,

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

const pyramidVertices = [
    // Основание (оранжевый)
    -0.45, 0.0, -0.45,  0.45, 0.0, -0.45,  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 (зелёный)
    0.0, 0.9, 0.0,  0.45, 0.0, -0.45, -0.45, 0.0, -0.45,
    // право (красный)
    0.0, 0.9, 0.0,  0.45, 0.0, 0.45, 0.45, 0.0, -0.45,
    // зад (синий)
    0.0, 0.9, 0.0,  -0.45, 0.0, 0.45, 0.45, 0.0, 0.45,
    // лево (жёлтый)
    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,  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
];

const pyramidTexCoords = [
   // Основание (оранжевый)
   0.0, 0.0, 1.0, 0.0, 1.0, 1.0,
   0.0, 0.0, 1.0, 1.0, 0.0, 1.0,

   // Боковые грани
   // перед (зелёный)
   0.5, 1.0, 0.0, 0.0, 1.0, 0.0,
   // право (красный)
   0.5, 1.0, 0.0, 0.0, 1.0, 0.0,
   // зад (синий)
   0.5, 1.0, 0.0, 0.0, 1.0, 0.0,
   // лево (жёлтый)
   0.5, 1.0, 0.0, 0.0, 1.0, 0.0,
];

function makeFlatNormals(vertices, start, num, normals) {
   if (num % 9 !== 0) {
      console.warn("Warning: number of floats is not a multiple of 9");
      return;
   }

   for (let i = start; i < start + num; i += 9) {
      const p0 = vec3.fromValues(vertices[i],   vertices[i+1], vertices[i+2]);
      const p1 = vec3.fromValues(vertices[i+3], vertices[i+4], vertices[i+5]);
      const p2 = vec3.fromValues(vertices[i+6], vertices[i+7], vertices[i+8]);

      const v1 = vec3.create();
      const v2 = vec3.create();
      const n = vec3.create();

      vec3.subtract(v1, p1, p0);
      vec3.subtract(v2, p2, p0);
      vec3.cross(n, v1, v2);
      vec3.normalize(n, n);

      normals.push(...n, ...n, ...n);
   }
}
```

```
const cubeNormals = [];
makeFlatNormals(cubeVertices, 0, cubeVertices.length, cubeNormals);

const pyramidNormals = [];
makeFlatNormals(pyramidVertices, 0, pyramidVertices.length, pyramidNormals);

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);

let cubeNormalBuffer = createBuffer(cubeNormals);
let pyramidNormalBuffer = createBuffer(pyramidNormals);

let cubeTexCoordBuffer = createBuffer(cubeTexCoords);
let pyramidTexCoordBuffer = createBuffer(pyramidTexCoords);

function loadTexture(gl, url, width, height, skipPixels = 0, skipRows = 0, scale_min =
false) {
   const texture = gl.createTexture();
   gl.bindTexture(gl.TEXTURE_2D, texture);

   const image = new Image();
   image.crossOrigin = "Anonymous";
   image.onload = function() {
      gl.bindTexture(gl.TEXTURE_2D, texture);

      gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
      gl.pixelStorei(gl.UNPACK_ALIGNMENT, 1);
      gl.pixelStorei(gl.UNPACK_ROW_LENGTH, image.width);
      gl.pixelStorei(gl.UNPACK_SKIP_PIXELS, skipPixels);
      gl.pixelStorei(gl.UNPACK_SKIP_ROWS, skipRows);

      const canvas = document.createElement('canvas');
      canvas.width = width;
```

```
      canvas.height = height;
      const ctx = canvas.getContext('2d');

      let scale;
      if (scale_min) {
         scale = Math.min(width / image.width, height / image.height);
      } else {
         scale = Math.max(width / image.width, height / image.height);
      }
      const x = (width - image.width * scale) / 2;
      const y = (height - image.height * scale) / 2;

      ctx.drawImage(image, x, y, image.width * scale, image.height * scale);

      gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, canvas);
      gl.generateMipmap(gl.TEXTURE_2D);

      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_LINEAR);

      gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, false);
      gl.pixelStorei(gl.UNPACK_ALIGNMENT, 4);
      gl.pixelStorei(gl.UNPACK_ROW_LENGTH, 0);
      gl.pixelStorei(gl.UNPACK_SKIP_PIXELS, 0);
      gl.pixelStorei(gl.UNPACK_SKIP_ROWS, 0);
   };
   image.src = url;
   return texture;
}

let url =
"https://steamuserimages-a.akamaihd.net/ugc/2339126078234824539/86A9178BC8ECD7B6E23007D
ED91C686A47517D0E/"
let texture128 = loadTexture(gl, url, 128, 128);
let texture256 = loadTexture(gl, url, 256, 256);
let texture512 = loadTexture(gl, url, 512, 512);
let texture128_min = loadTexture(gl, url, 128, 128, 0, 0, true);
let texture256_min = loadTexture(gl, url, 256, 256, 0, 0, true);
let texture512_min = loadTexture(gl, url, 512, 512, 0, 0, true);

const vertCode = `#version 300 es
   precision mediump float;

   in vec3 coordinates;
   in vec3 color;
   in vec3 normal;
   in vec2 texCoord;

   out vec3 vColor;
   out vec3 vPosition;
   out vec3 vNormal;
   out vec3 vLightColorGouraud;
   out vec2 vTexCoord;

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

```
   uniform vec3 light1Direction;
   uniform vec3 light2Color;
   uniform vec3 light2Direction;
   uniform vec3 dirLightDirection;
   uniform vec3 dirLightColor;
   uniform vec3 pointLightPosition;
   uniform vec3 pointLightColor;

   uniform bool useGouraudShading;
   uniform bool useTexture;

   uniform vec3 materialAmbient;
   uniform vec3 materialDiffuse;
   uniform vec3 materialSpecular;
   uniform float materialShininess;

   void main(void) {
      vec4 pos = modelViewMatrix * vec4(coordinates, 1.0);
      vColor = color;
      vPosition = pos.xyz;
      vec3 transformedNormal = normalize(mat3(normalMatrix) * normal);
      vNormal = transformedNormal;
      vTexCoord = texCoord;

      if (useGouraudShading) {
         vec3 lightColor = vec3(0.0);
         vec3 viewDir = normalize(-pos.xyz);

         vec3 light1 = normalize(light1Direction);
         float diffuse1 = max(dot(transformedNormal, light1), 0.0);
         vec3 reflect1 = reflect(-light1, transformedNormal);
         float spec1 = pow(max(dot(viewDir, reflect1), 0.0), materialShininess);
         lightColor += materialAmbient +
                       materialDiffuse * diffuse1 +
                       materialSpecular * spec1;

         vec3 light2Dir = normalize(light2Direction);
         float light2Diffuse = max(dot(transformedNormal, light2Dir), 0.0);
         vec3 light2Reflect = reflect(-light2Dir, transformedNormal);
         float light2Spec = pow(max(dot(viewDir, light2Reflect), 0.0),
materialShininess);
         lightColor += light2Color * (materialDiffuse * light2Diffuse +
materialSpecular * light2Spec);

         vec3 dirDir = normalize(-dirLightDirection);
         float dirDiffuse = max(dot(transformedNormal, dirDir), 0.0);
         vec3 dirReflect = reflect(-dirDir, transformedNormal);
         float dirSpec = pow(max(dot(viewDir, dirReflect), 0.0), materialShininess);
         lightColor += dirLightColor * (materialDiffuse * dirDiffuse + materialSpecular
* dirSpec);

         vec3 pointDir = normalize(pointLightPosition - pos.xyz);
         float pointDiffuse = max(dot(transformedNormal, pointDir), 0.0);
         vec3 pointReflect = reflect(-pointDir, transformedNormal);
         float pointSpec = pow(max(dot(viewDir, pointReflect), 0.0),
materialShininess);
```

```
         float distance = length(pointLightPosition - pos.xyz);
         float attenuation = 1.0 / distance;
         lightColor += pointLightColor * attenuation * (materialDiffuse * pointDiffuse
+ materialSpecular * pointSpec);

         if (useTexture) {
               vLightColorGouraud = lightColor;
         } else {
               vLightColorGouraud = vColor * lightColor;
         }
      }

      gl_Position = projectionMatrix * pos;
   }
`;

let vertShader = gl.createShader(gl.VERTEX_SHADER);
gl.shaderSource(vertShader, vertCode);
gl.compileShader(vertShader);
if (!gl.getShaderParameter(vertShader, gl.COMPILE_STATUS)) {
   console.error("Vertex Shader Error:\n", gl.getShaderInfoLog(vertShader));
}

const fragCode = `#version 300 es
   precision mediump float;

   in vec3 vColor;
   in vec3 vPosition;
   in vec3 vNormal;
   in vec3 vLightColorGouraud;
   in vec2 vTexCoord;

   out vec4 fragColor;

   uniform vec3 light1Direction;
   uniform vec3 light2Color;
   uniform vec3 light2Direction;
   uniform vec3 dirLightDirection;
   uniform vec3 dirLightColor;
   uniform vec3 pointLightPosition;
   uniform vec3 pointLightColor;

   uniform vec3 materialAmbient;
   uniform vec3 materialDiffuse;
   uniform vec3 materialSpecular;
   uniform float materialShininess;

   uniform bool useGouraudShading;
   uniform bool useTexture;

   uniform sampler2D uTexture;

   void main(void) {
      if (useGouraudShading) {
         if (useTexture) {
               vec4 texColor = texture(uTexture, vTexCoord);
               fragColor = vec4(texColor.rgb * vLightColorGouraud, 1.0);
```

```
        } else {
              fragColor = vec4(vLightColorGouraud, 1.0);
        }
      } else {
        vec3 normal = vNormal;
        vec3 viewDir = normalize(-vPosition);
        vec3 lightColor = vec3(0.0);

        vec3 light1 = normalize(light1Direction);
        float diffuse1 = max(dot(normal, light1), 0.0);
        vec3 reflect1 = reflect(-light1, normal);
        float spec1 = pow(max(dot(viewDir, reflect1), 0.0), materialShininess);
        lightColor += materialAmbient + materialDiffuse * diffuse1 + materialSpecular
* spec1;

        vec3 light2Dir = normalize(light2Direction);
        float light2Diffuse = max(dot(normal, light2Dir), 0.0);
        vec3 light2Reflect = reflect(-light2Dir, normal);
        float light2Spec = pow(max(dot(viewDir, light2Reflect), 0.0),
materialShininess);
        lightColor += light2Color * (materialDiffuse * light2Diffuse +
materialSpecular * light2Spec);

        vec3 dirDir = normalize(-dirLightDirection);
        float dirDiffuse = max(dot(normal, dirDir), 0.0);
        vec3 dirReflect = reflect(-dirDir, normal);
        float dirSpec = pow(max(dot(viewDir, dirReflect), 0.0), materialShininess);
        lightColor += dirLightColor * (materialDiffuse * dirDiffuse + materialSpecular
* dirSpec);

        vec3 pointDir = normalize(pointLightPosition - vPosition);
        float pointDiffuse = max(dot(normal, pointDir), 0.0);
        vec3 pointReflect = reflect(-pointDir, normal);
        float pointSpec = pow(max(dot(viewDir, pointReflect), 0.0),
materialShininess);
        float distance = length(pointLightPosition - vPosition);
        float attenuation = 1.0 / distance;
        lightColor += pointLightColor * attenuation * (materialDiffuse * pointDiffuse
+ materialSpecular * pointSpec);

        vec3 finalColor;
        if (useTexture) {
              vec4 texColor = texture(uTexture, vTexCoord);
              finalColor = texColor.rgb * lightColor;
        } else {
              finalColor = vColor * lightColor;
        }
        fragColor = vec4(finalColor, 1.0);
      }
   }
`;

let fragShader = gl.createShader(gl.FRAGMENT_SHADER);
gl.shaderSource(fragShader, fragCode);
gl.compileShader(fragShader);
if (!gl.getShaderParameter(fragShader, gl.COMPILE_STATUS)) {
   console.error("Fragment Shader Error:\n", gl.getShaderInfoLog(fragShader));
```

```
}

let shaderProgram = gl.createProgram();
gl.attachShader(shaderProgram, vertShader);
gl.attachShader(shaderProgram, fragShader);
gl.linkProgram(shaderProgram);
gl.useProgram(shaderProgram);
if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {
   console.error("Program Link Error:\n", gl.getProgramInfoLog(shaderProgram));
}

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

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

const light1DirectionLocation = gl.getUniformLocation(shaderProgram,
'light1Direction');
const light2ColorLocation= gl.getUniformLocation(shaderProgram, 'light2Color');
const light2DirectionLocation= gl.getUniformLocation(shaderProgram, 'light2Direction');
const dirLightDirectionLocation= gl.getUniformLocation(shaderProgram,
'dirLightDirection');
const dirLightColorLocation= gl.getUniformLocation(shaderProgram, 'dirLightColor');
const pointLightDirectionLocation= gl.getUniformLocation(shaderProgram,
'pointLightPosition');
const pointLightColorLocation= gl.getUniformLocation(shaderProgram, 'pointLightColor');
const useGouraudLocation = gl.getUniformLocation(shaderProgram, "useGouraudShading");
const materialAmbientLocation = gl.getUniformLocation(shaderProgram,
"materialAmbient");
const materialDiffuseLocation = gl.getUniformLocation(shaderProgram,
"materialDiffuse");
const materialSpecularLocation = gl.getUniformLocation(shaderProgram,
"materialSpecular");
const materialShininessLocation = gl.getUniformLocation(shaderProgram,
"materialShininess");

const useTextureLocation = gl.getUniformLocation(shaderProgram, "useTexture");
const uTextureLocation = gl.getUniformLocation(shaderProgram, "uTexture");

function drawCubeOrPyramid(vertexBuffer, colorBuffer, normalBuffer, texCoordBuffer,
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.bindBuffer(gl.ARRAY_BUFFER, normalBuffer);
```

```
   gl.vertexAttribPointer(normalLocation, 3, gl.FLOAT, false, 0, 0);

   gl.bindBuffer(gl.ARRAY_BUFFER, texCoordBuffer);
   gl.vertexAttribPointer(texCoordLocation, 2, gl.FLOAT, false, 0, 0);

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

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 hexToVec3(hex) {
   const bigint = parseInt(hex.slice(1), 16);
   const r = ((bigint >> 16) & 255) / 255;
   const g = ((bigint >> 8) & 255) / 255;
   const b = (bigint & 255) / 255;
   return vec3.fromValues(r, g, b);
}

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

   gl.uniform3fv(light1DirectionLocation, [1.0, 1.0, 1.0]);

   const enablelight2 = document.getElementById('enableLight2').checked;
   const enableDir = document.getElementById('enableDirLight').checked;
   const enablePoint = document.getElementById('enablePointLight').checked;
   const useGouraud = document.getElementById("gouraudToggle").checked;

   // Второй свет
```

```
   if (enablelight2) {
      gl.uniform3fv(light2ColorLocation, [0.67, 0.4, 0.8]);
      gl.uniform3fv(light2DirectionLocation, [-1.0, 0.0, -1.0]);
   } else {
      gl.uniform3fv(light2ColorLocation, [0.0, 0.0, 0.0]);
   }

   // Третий — направленный
   if (enableDir) {
      gl.uniform3fv(dirLightColorLocation, [1.0, 0.67, 0.2]);
      gl.uniform3fv(dirLightDirectionLocation, [0.0, 1.0, 0.0]);
   } else {
      gl.uniform3fv(dirLightColorLocation, [0.0, 0.0, 0.0]);
   }

   // Четвёртый — позиционный
   if (enablePoint) {
      gl.uniform3fv(pointLightDirectionLocation, [0.5, 0.5, 0.0]);
      gl.uniform3fv(pointLightColorLocation, [0.4, 0.8, 1.0]);
   } else {
      gl.uniform3fv(pointLightColorLocation, [0.0, 0.0, 0.0]);
   }

   gl.uniform1i(useGouraudLocation, useGouraud ? 1 : 0);

   const ambient = hexToVec3(document.getElementById("ambientColor").value);
   const diffuse = hexToVec3(document.getElementById("diffuseColor").value);
   const specular = hexToVec3(document.getElementById("specularColor").value);
   const shininess = parseFloat(document.getElementById("shininess").value);

   gl.uniform3fv(materialAmbientLocation, ambient);
   gl.uniform3fv(materialDiffuseLocation, diffuse);
   gl.uniform3fv(materialSpecularLocation, specular);
   gl.uniform1f(materialShininessLocation, shininess);

   const useTexture = document.getElementById('useTexture').checked;
   gl.uniform1i(useTextureLocation, useTexture ? 1 : 0);

   const useScaleMin = document.getElementById('useScaleMin').checked;

   if (useTexture) {
      const textureSize =
document.querySelector('input[name="textureSize"]:checked').value;
      let currentTexture;
      if (useScaleMin) {
         switch(textureSize) {
            case '128': currentTexture = texture128_min; break;
            case '256': currentTexture = texture256_min; break;
            case '512': currentTexture = texture512_min; break;
         }
      }
      else {
         switch(textureSize) {
            case '128': currentTexture = texture128; break;
            case '256': currentTexture = texture256; break;
            case '512': currentTexture = texture512; break;
         }
```

```
      }

      gl.activeTexture(gl.TEXTURE0);
      gl.bindTexture(gl.TEXTURE_2D, currentTexture);
      gl.uniform1i(uTextureLocation, 0);
   }

   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);

   drawCubeOrPyramid(cubeBuffer, cubeColorBuffer, cubeNormalBuffer, cubeTexCoordBuffer,
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);

   drawCubeOrPyramid(pyramidBuffer, pyramidColorBuffer, pyramidNormalBuffer,
pyramidTexCoordBuffer, pyramidVertices.length / 3);
}


document.getElementById('projectionType').addEventListener('change', updateView);
document.getElementById('enableLight2').addEventListener('change', updateView);
document.getElementById('enableDirLight').addEventListener('change', updateView);
document.getElementById('enablePointLight').addEventListener('change', updateView);
document.getElementById('gouraudToggle').addEventListener('change', updateView);
document.getElementById('useTexture').addEventListener('change', updateView);
document.getElementById('useScaleMin').addEventListener('change', updateView);
document.querySelectorAll('input[name="textureSize"]').forEach(radio => {
   radio.addEventListener('change', updateView);
});
document.querySelectorAll('input').forEach(input => {
   input.addEventListener('input', updateView);
});
document.querySelectorAll("#material-control input").forEach(input => {
   input.addEventListener("input", () => {
      document.getElementById("shininessValue").textContent =
document.getElementById("shininess").value;
   });
```

```
});

updateView();
```