**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Компьютерная графика»

**Лабораторная работа № 2**

Тема: Каркасная визуализация выпуклого многогранника. Удаление невидимых линий

Студент: Ватулин Валентин Михайлович

Группа: 80-306

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

1. Постановка задачи

Разработать формат представления многогранника и процедуру его каркасной отрисовки в ортографической и изометрической проекциях. Обеспечить удаление невидимых линий и возможность пространственных поворотов и масштабирования многогранника. Обеспечить автоматическое центрирование и изменение размеров изображения при изменении размеров окна.

Вариант 5: обелиск(усеченный клин).

1. Описание программы

Работа написана на JavaScript с использованием технологии WebGL и работает в браузере. Для работы с проекциями и изменением положения объекта в пространстве был написан класс m4, представляющий из себя матрицу размера 4х4 и методы работы с ней. Присутствует возможность изменения положение объекта по трём осям, поворота объекта по трем осям, масштабирования объекта, а также изменения отрисовки объекта: присутствует возможность отрисовывать ортогональную и перспективную проекцию.

Для упрощения работы с WebGL были написаны функции создания шейдера и создания програмы отрисовки (createShader и createProgram соответственно).

Объект для отрисовки представляет из себя JavaScript объект, хранящий всю необходимую информацию о его свойствах (координаты вершин, положение в пространстве и т.д.). Из вершин объекта составляется буфер вершин, а из переменных пространственного положения объекта составляется матрица отображения, которая применяется в шейдере для правильной отрисовки объекта в пространстве. Также каждая вершина объекта хранит случайно выбранный при запуске программы цвет, который применяется к соответствующей грани объекта.

1. Набор тестов

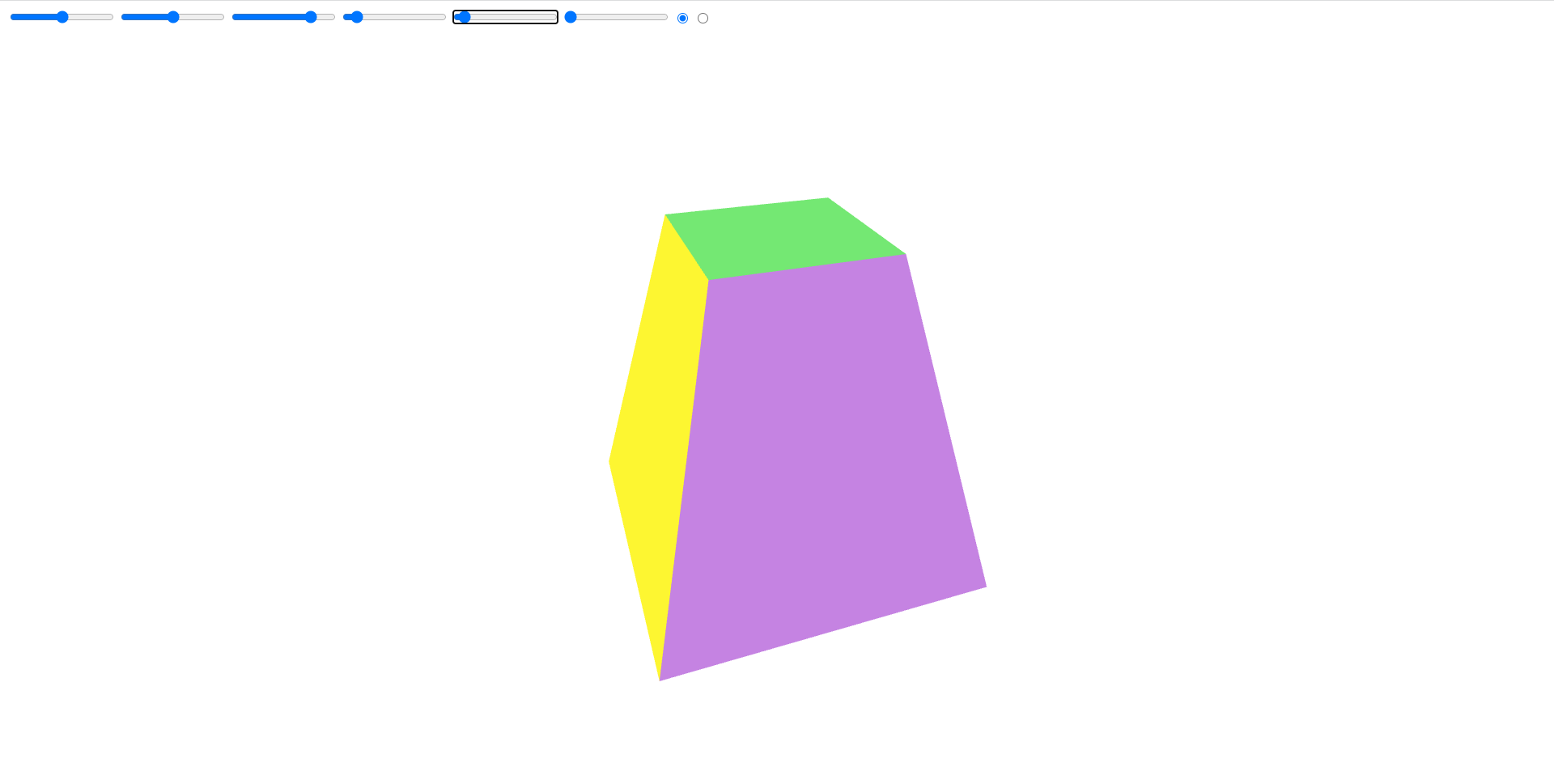


рис. 1 - Перспективная проекция

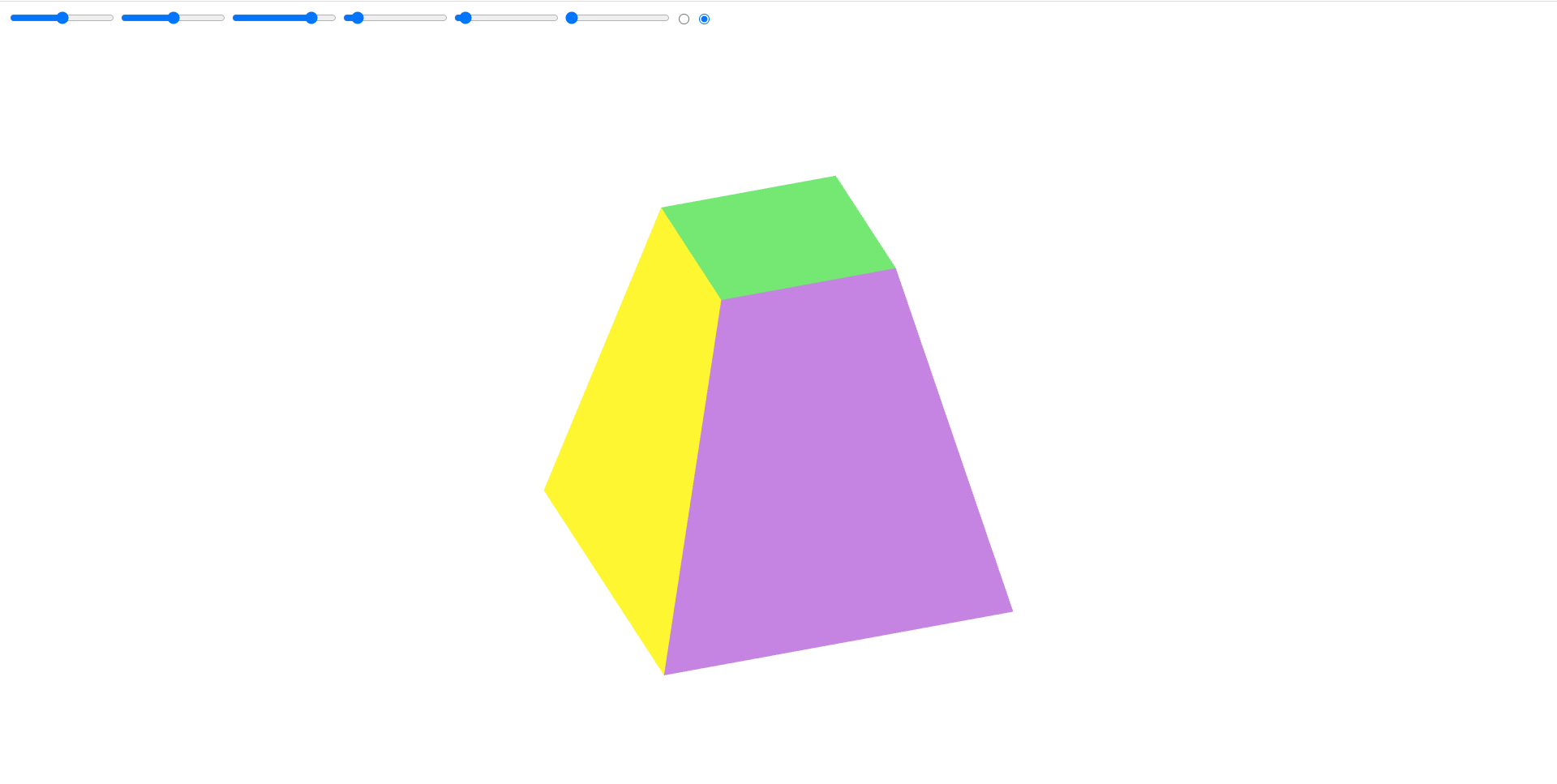


рис. 2 - Ортогональная проекция

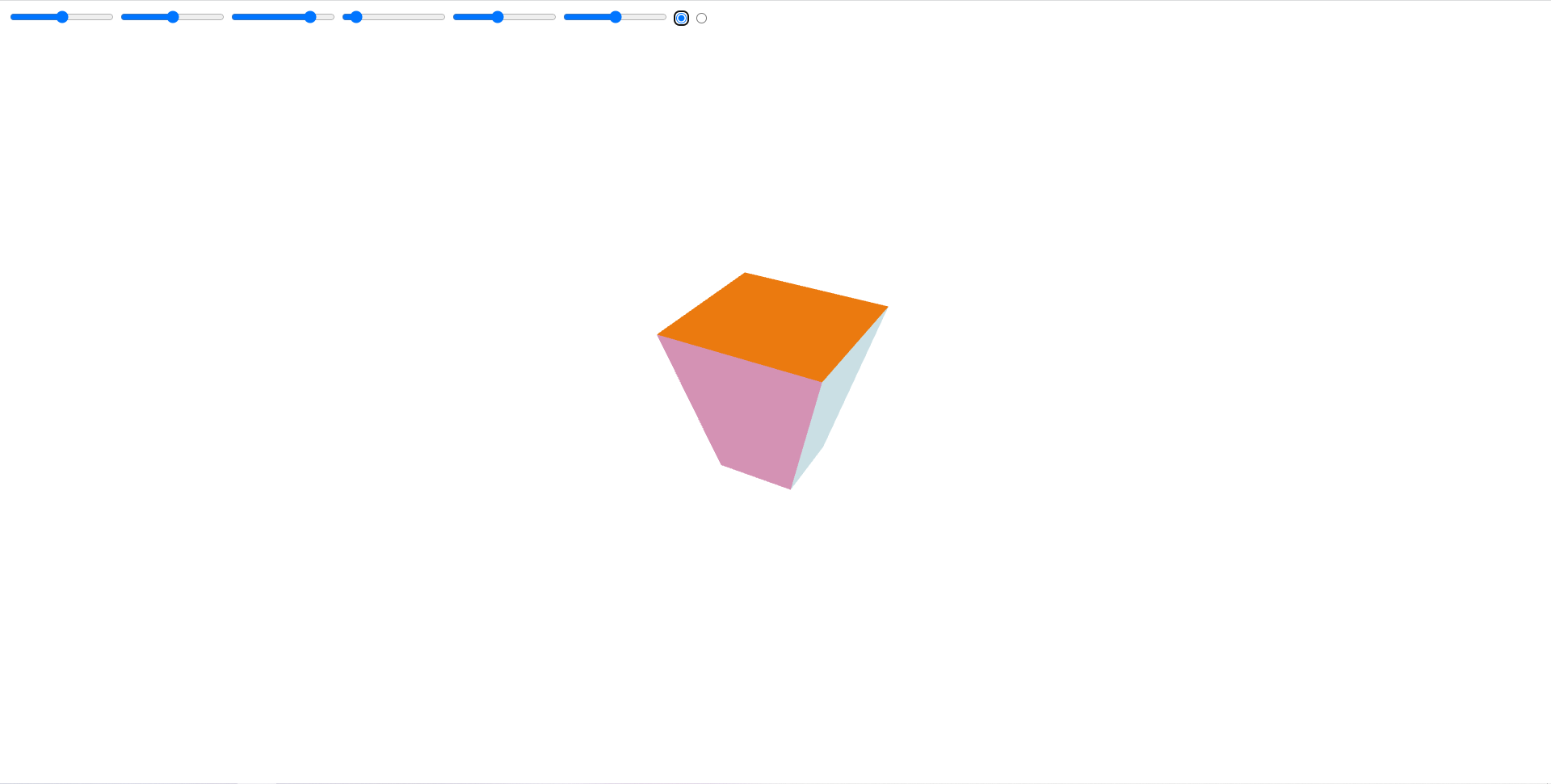


рис. 3 - Перспективная проекция, объект уменьшен и перевернут

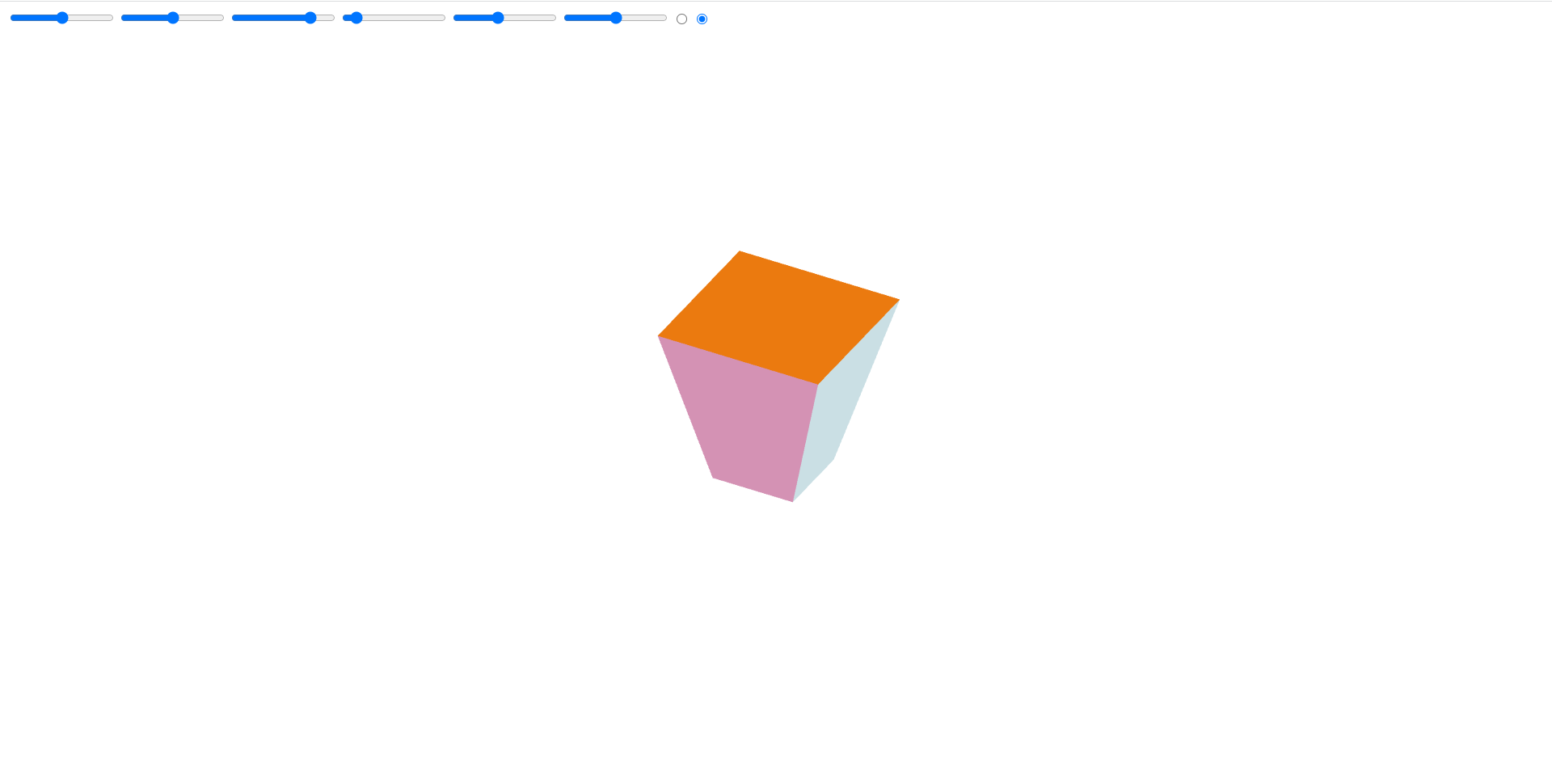


рис. 4 - Ортогональная проекция, объект уменьшен и перевернут

1. Листинг программы

index.html:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>2</title>

<link rel="stylesheet" href="./style.css">

</head>

<body>

<canvas></canvas>

<div id="sliders">

<input type="range" min="-2" max="2" value="0" step="0.01" id="xt">

<input type="range" min="-2" max="2" value="0" step="0.01" id="yt">

<input type="range" min="-10" max="0" value="-2" step="0.01" id="zt">

<input type="range" min="0" max="360" value="0" id="rotX">

<input type="range" min="0" max="360" value="0" id="rotY">

<input type="range" min="0" max="360" value="0" id="rotZ">

<input type="radio" name="projection" value="0" checked>

<input type="radio" name="projection" value="1">

</div>

<script src="./m4.js"></script>

<script src="./main.js"></script>

</body>

</html>

style.css:

body {

margin: 0px;

width: 100vw;

height: 100vh;

}

#sliders {

position: absolute;

top: 10px;

left: 10px;

}

m4.js:

'use strict';

class m4 {

constructor(arr) {

this.array = [...arr];

}

static get n() {

return 4;

}

static get identity() {

return new m4([1, 0, 0, 0,

0, 1, 0, 0,

0, 0, 1, 0,

0, 0, 0, 1]);

}

multiply(other) {

let res = new m4(new Array(m4.n \*\* 2));

for (let i = 0; i < m4.n; ++i) {

for (let j = 0; j < m4.n; ++j) {

res.array[m4.n \* i + j] = 0;

for (let k = 0; k < m4.n; ++k) {

res.array[m4.n \* i + j] += other.array[m4.n \* i + k] \* this.array[m4.n \* k + j];

}

}

}

return res;

}

translate(tx, ty, tz) {

let res = new m4(this.array);

let t\_matrix = new m4([1, 0, 0, 0,

0, 1, 0, 0,

0, 0, 1, 0,

tx, ty, tz, 1]);

return res.multiply(t\_matrix);

}

rotateX(x) {

let res = new m4(this.array);

let c = Math.cos(x);

let s = Math.sin(x);

let rot\_matrix = new m4([1, 0, 0, 0,

0, c, s, 0,

0,-s, c, 0,

0, 0, 0, 1]);

return res.multiply(rot\_matrix);

}

rotateY(x) {

let res = new m4(this.array);

let c = Math.cos(x);

let s = Math.sin(x);

let rot\_matrix = new m4([c, 0,-s, 0,

0, 1, 0, 0,

s, 0, c, 0,

0, 0, 0, 1]);

return res.multiply(rot\_matrix);

}

rotateZ(x) {

let res = new m4(this.array);

let c = Math.cos(x);

let s = Math.sin(x);

let rot\_matrix = new m4([c, s, 0, 0,

-s, c, 0, 0,

0, 0, 1, 0,

0, 0, 0, 1]);

return res.multiply(rot\_matrix);

}

rotate(x, y, z) {

let res = new m4(this.array);

return res.rotateX(x).rotateY(y).rotateZ(z);

}

scale(sx, sy, sz) {

let res = new m4(this.array);

let s\_matrix = new m4([sx, 0, 0, 0,

0, sy, 0, 0,

0, 0, sz, 0,

0, 0, 0, 1]);

return res.multiply(s\_matrix);

}

static perspective(fov, aspect, near, far) {

let f = 1 / Math.tan(fov/2);

return new m4([f/aspect, 0, 0, 0,

0, f, 0, 0,

0, 0, -(far+near)/(far-near), -1,

0, 0, -2\*(far\*near)/(far-near), 0]);

}

}

main.js:

'use strict';

let shaderSources = {

vertexShader: `#version 300 es

in vec4 a\_position;

in vec4 a\_color;

uniform mat4 u\_matrix;

out vec4 v\_color;

void main() {

v\_color = a\_color;

gl\_Position = u\_matrix \* a\_position;

}

`,

fragmentShader: `#version 300 es

precision highp float;

in vec4 v\_color;

out vec4 outColor;

void main() {

outColor = v\_color;

}

`

};

function createShader(gl, type, source) {

let shader = gl.createShader(type);

gl.shaderSource(shader, source);

gl.compileShader(shader);

let success = gl.getShaderParameter(shader, gl.COMPILE\_STATUS);

if (success) {

return shader;

}

console.log(gl.getShaderInfoLog(shader));

gl.deleteShader(shader);

}

function createProgram(gl, shader1, shader2) {

let program = gl.createProgram();

gl.attachShader(program, shader1);

gl.attachShader(program, shader2);

gl.linkProgram(program);

let success = gl.getProgramParameter(program, gl.LINK\_STATUS);

if (success) {

return program;

}

console.log(gl.getProgramInfoLog(program));

gl.deleteProgram(program);

}

let rand = Math.random;

let mode = 0;

function main() {

let obj = {

x: 0,

y: 0,

z: -2,

rotX: 0,

rotY: 0,

rotZ: 0,

scale: 0.25,

vertices: [

-1, -1, -1,

1, -1, -1,

0.5, 1, -0.5,

-0.5, 1, -0.5,

-1, -1, 1,

1, -1, 1,

0.5, 1, 0.5,

-0.5, 1, 0.5,

-1, -1, -1,

-0.5, 1, -0.5,

-0.5, 1, 0.5,

-1, -1, 1,

1, -1, -1,

0.5, 1, -0.5,

0.5, 1, 0.5,

1, -1, 1,

-0.5, 1, -0.5,

0.5, 1, -0.5,

0.5, 1, 0.5,

-0.5, 1, 0.5,

-1, -1, -1,

1, -1, -1,

1, -1, 1,

-1, -1, 1,

],

indices: [

0, 2, 1,

0, 3, 2,

4, 5, 6,

4, 6, 7,

8, 10, 9,

8, 11, 10,

12, 13, 14,

12, 14, 15,

16, 18, 17,

16, 19, 18,

20, 21, 22,

20, 22, 23

],

colors: []

};

for (let i = 0; i < 6; ++i) {

let rand\_color1 = rand();

let rand\_color2 = rand();

let rand\_color3 = rand();

for (let j = 0; j < 4; ++j) {

obj.colors.push(rand\_color1);

obj.colors.push(rand\_color2);

obj.colors.push(rand\_color3);

}

}

/\*\* @type {HTMLCanvasElement} \*/

let canvas = document.querySelector('canvas');

canvas.width = window.innerWidth;

canvas.height = window.innerHeight;

let gl = canvas.getContext('webgl2');

gl.clearColor(1, 1, 1, 1);

gl.viewport(0, 0, canvas.width, canvas.height);

gl.enable(gl.DEPTH\_TEST);

gl.enable(gl.CULL\_FACE);

window.addEventListener('resize', function(event) {

canvas.width = window.innerWidth;

canvas.height = window.innerHeight;

gl.viewport(0, 0, canvas.width, canvas.height);

draw();

});

document.querySelector('#xt').addEventListener('input', function(event) {

obj.x = this.value;

draw();

});

document.querySelector('#yt').addEventListener('input', function(event) {

obj.y = this.value;

draw();

});

document.querySelector('#zt').addEventListener('input', function(event) {

obj.z = this.value;

draw();

});

addEventListener('wheel', function(event) {

obj.scale -= event.deltaY \* 0.0002;

obj.scale = Math.max(obj.scale, 0);

draw();

});

document.querySelector('#rotX').addEventListener('input', function(event) {

obj.rotX = this.value/180 \* Math.PI;

draw();

});

document.querySelector('#rotY').addEventListener('input', function(event) {

obj.rotY = this.value/180 \* Math.PI;

draw();

});

document.querySelector('#rotZ').addEventListener('input', function(event) {

obj.rotZ = this.value/180 \* Math.PI;

draw();

});

let radios = document.querySelectorAll('input[type="radio"]');

for (let i of radios) {

i.addEventListener('click', function(event) {

if (this.value === '0') {

mode = 0;

draw();

}

else {

mode = 1;

draw();

}

});

}

let vertexShader = createShader(gl, gl.VERTEX\_SHADER, shaderSources.vertexShader);

let fragmentShader = createShader(gl, gl.FRAGMENT\_SHADER, shaderSources.fragmentShader);

let program = createProgram(gl, vertexShader, fragmentShader);

gl.useProgram(program);

let positionLocation = gl.getAttribLocation(program, 'a\_position');

let colorLocation = gl.getAttribLocation(program, 'a\_color');

let matrixLocation = gl.getUniformLocation(program, 'u\_matrix');

let vao = gl.createVertexArray();

gl.bindVertexArray(vao);

let positionBuff = gl.createBuffer();

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

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

gl.enableVertexAttribArray(positionLocation);

gl.vertexAttribPointer(positionLocation, 3, gl.FLOAT, false, 0, 0);

let colorBuff = gl.createBuffer();

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

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

gl.enableVertexAttribArray(colorLocation);

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

let indexBuff = gl.createBuffer();

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

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

draw()

function draw() {

let zval;

let normal = m4.identity;

if (canvas.width > canvas.height) {

normal = normal.scale(canvas.height/canvas.width, 1, 1);

}

else {

normal = normal.scale(1, canvas.width/canvas.height, 1);

}

let matrix;

if (mode === 0) {

matrix = m4.perspective(45, canvas.width/canvas.height, 0.1, 100);

zval = obj.z;

}

else {

matrix = normal.multiply(new m4([1, 0, 0, 0,

0, 1, 0, 0,

0, 0,-1, 0,

0, 0, 0, 1]));

zval = 0;

}

matrix = matrix.translate(obj.x, obj.y, zval).rotate(obj.rotX, obj.rotY, obj.rotZ).scale(obj.scale, obj.scale, obj.scale);

gl.uniformMatrix4fv(matrixLocation, false, matrix.array);

gl.drawElements(gl.TRIANGLES, 36, gl.UNSIGNED\_SHORT, 0);

}

}

main();

1. Вывод

В ходе данной лабораторной работы я научился работать с технологией WebGL, и освежил свои навыки работы с матрицами отображения.

ЛИТЕРАТУРА

1. WebGL Fundamentals [Электронный ресурс]  
   URL: <https://webgl2fundamentals.org/>  
   (дата обращения: 10.10.2021)
2. WebGL: 2D and 3D graphics for the web [Электронный ресурс]  
   URL: <https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API>  
   (дата обращения: 10.10.2021)