//librairie pour la gestion des matrices 3x3 // code recupere de https://webglfundamentals.org/
var m4 = {

```
identity: function () {
    return [
        1, 0, 0, 0,
        0, 1, 0, 0,
        0, 0, 1, 0,
        0, 0, 0, 1,
    ];
},


translation: function (tx, ty, tz) {
    return [
        1, 0, 0, 0,
        0, 1, 0, 0,
        0, 0, 1, 0,
        tx, ty, tz, 1,
    ];
},

xRotation: function (angleInRadians) {
    var c = Math.cos(angleInRadians);
    var s = Math.sin(angleInRadians);

    return [
        1, 0, 0, 0,
        0, c, s, 0,
        0, -s, c, 0,
        0, 0, 0, 1,
    ];
},

yRotation: function (angleInRadians) {
    var c = Math.cos(angleInRadians);
    var s = Math.sin(angleInRadians);

    return [
        c, 0, -s, 0,
        0, 1, 0, 0,
        s, 0, c, 0,
        0, 0, 0, 1,
    ];
},
zRotation: function (angleInRadians) {
    var c = Math.cos(angleInRadians);
    var s = Math.sin(angleInRadians);

    return [
        c, s, 0, 0,
        -s, c, 0, 0,
        0, 0, 1, 0,
        0, 0, 0, 1,
    ];
},

scaling: function (sx, sy, sz) {
    return [
        sx, 0, 0, 0,
        0, sy, 0, 0,
        0, 0, sz, 0,
        0, 0, 0, 1,
    ];
},
translate: function (m, tx, ty, tz) {
    return m4.multiply(m, m4.translation(tx, ty, tz));
},

xRotate: function (m, angleInRadians) {
    return m4.multiply(m, m4.xRotation(angleInRadians));
},

yRotate: function (m, angleInRadians) {
    return m4.multiply(m, m4.yRotation(angleInRadians));
},

zRotate: function (m, angleInRadians) {
    return m4.multiply(m, m4.zRotation(angleInRadians));
},

scale: function (m, sx, sy, sz) {
    return m4.multiply(m, m4.scaling(sx, sy, sz));
},
multiply: function (a, b) {
    var b00 = b[0 * 4 + 0];
    var b01 = b[0 * 4 + 1];
    var b02 = b[0 * 4 + 2];
    var b03 = b[0 * 4 + 3];
    var b10 = b[1 * 4 + 0];
    var b11 = b[1 * 4 + 1];
    var b12 = b[1 * 4 + 2];
    var b13 = b[1 * 4 + 3];
    var b20 = b[2 * 4 + 0];
    var b21 = b[2 * 4 + 1];
    var b22 = b[2 * 4 + 2];
    var b23 = b[2 * 4 + 3];
    var b30 = b[3 * 4 + 0];
    var b31 = b[3 * 4 + 1];
    var b32 = b[3 * 4 + 2];
    var b33 = b[3 * 4 + 3];
    var a00 = a[0 * 4 + 0];
    var a01 = a[0 * 4 + 1];
    var a02 = a[0 * 4 + 2];
    var a03 = a[0 * 4 + 3];
    var a10 = a[1 * 4 + 0];
    var a11 = a[1 * 4 + 1];
    var a12 = a[1 * 4 + 2];
    var a13 = a[1 * 4 + 3];
    var a20 = a[2 * 4 + 0];
    var a21 = a[2 * 4 + 1];
    var a22 = a[2 * 4 + 2];
    var a23 = a[2 * 4 + 3];
    var a30 = a[3 * 4 + 0];
    var a31 = a[3 * 4 + 1];
    var a32 = a[3 * 4 + 2];
    var a33 = a[3 * 4 + 3];

    return [
        b00 * a00 + b01 * a10 + b02 * a20 + b03 * a30,
        b00 * a01 + b01 * a11 + b02 * a21 + b03 * a31,
        b00 * a02 + b01 * a12 + b02 * a22 + b03 * a32,
        b00 * a03 + b01 * a13 + b02 * a23 + b03 * a33,
        b10 * a00 + b11 * a10 + b12 * a20 + b13 * a30,
        b10 * a01 + b11 * a11 + b12 * a21 + b13 * a31,
        b10 * a02 + b11 * a12 + b12 * a22 + b13 * a32,
        b10 * a03 + b11 * a13 + b12 * a23 + b13 * a33,
        b20 * a00 + b21 * a10 + b22 * a20 + b23 * a30,
        b20 * a01 + b21 * a11 + b22 * a21 + b23 * a31,
        b20 * a02 + b21 * a12 + b22 * a22 + b23 * a32,
        b20 * a03 + b21 * a13 + b22 * a23 + b23 * a33,
        b30 * a00 + b31 * a10 + b32 * a20 + b33 * a30,
        b30 * a01 + b31 * a11 + b32 * a21 + b33 * a31,
        b30 * a02 + b31 * a12 + b32 * a22 + b33 * a32,
        b30 * a03 + b31 * a13 + b32 * a23 + b33 * a33,
    ];
},
projection: function (width, height, depth) {
    // Note: This matrix flips the Y axis so 0 is at the top.
    return [
        2 / width, 0, 0, 0,
        0, -2 / height, 0, 0,
        0, 0, 2 / depth, 0,
        -1, 1, 0, 1,
    ];
},

orthographic: function (left, right, bottom, top, near, far) {
    return [
        2 / (right - left), 0, 0, 0,
        0, 2 / (top - bottom), 0, 0,
        0, 0, 2 / (near - far), 0,

        (left + right) / (left - right),
        (bottom + top) / (bottom - top),
        (near + far) / (near - far),
        1,
    ];
},

createPerspectiveMatrixFromZ: function (gamma) {
    return [
        1, 0, 0, 0,
        0,1, 0, 0,
        0,0, 1, gamma,
        0,0, 0, 1,
    ];
}
,

perspective: function(fieldOfViewInRadians, aspect, near, far) {
var f = Math.tan(Math.PI * 0.5 - 0.5 * fieldOfViewInRadians);
var rangeInv = 1.0 / (near - far);
return [
  f / aspect, 0, 0, 0,
  0, f, 0, 0,
  0, 0, (near + far) * rangeInv, -1,
  0, 0, near * far * rangeInv * 2, 0
];
```

}

};

var program; var gl; var translation = [0, -70, -450]; var rotation = [-20, -40, 350]; var scale = [3, 3, 3]; var color;

var near = 1; var far = 2000;

//NEW var fov = 30;

var positionLocation; //var resolutionLocation; var positionBuffer; var colorBuffer; var matrixLocation;

function setupUI() { let sliderContainer = document.getElementById("sliderContainer"); createSliderWithLabel2(sliderContainer, -gl.canvas.width, gl.canvas.width, translation[0], 1, "tx", function (v) { translation[0] = v; updateScene(); }) sliderContainer.appendChild(document.createElement('br')); createSliderWithLabel2(sliderContainer, -gl.canvas.height, gl.canvas.height, translation[1], 1, "ty", function (v) { translation[1] = v; updateScene(); }) sliderContainer.appendChild(document.createElement('br')); createSliderWithLabel2(sliderContainer, -gl.canvas.height, gl.canvas.height, translation[2], 1, "tz", function (v) { translation[2] = v; updateScene(); })

```
sliderContainer.appendChild(document.createElement('hr'));
createSliderWithLabel2(sliderContainer, 0, 360, rotation[0], 1, "rx", function (v) { rotatio
sliderContainer.appendChild(document.createElement('br'));
createSliderWithLabel2(sliderContainer, 0, 360, rotation[1], 1, "ry", function (v) { rotatio
sliderContainer.appendChild(document.createElement('br'));
createSliderWithLabel2(sliderContainer, 0, 360, rotation[2], 1, "rz", function (v) { rotatio

sliderContainer.appendChild(document.createElement('hr'));
createSliderWithLabel2(sliderContainer, 0, 5, scale[0], 0.2, "sx", function (v) { scale[0] =
sliderContainer.appendChild(document.createElement('br'));
createSliderWithLabel2(sliderContainer, 0, 5, scale[1], 0.2, "sy", function (v) { scale[1] =
sliderContainer.appendChild(document.createElement('br'));
createSliderWithLabel2(sliderContainer, 0, 5, scale[2], 0.2, "sz", function (v) { scale[2] =


//---
sliderContainer.appendChild(document.createElement('hr'));
createSliderWithLabel2(sliderContainer, 0, 1000, near, 1, "near", function (v) { near = v; u
sliderContainer.appendChild(document.createElement('br'));
createSliderWithLabel2(sliderContainer, -400, 0, far, 1, "far", function (v) { far = v; upda
sliderContainer.appendChild(document.createElement('br'));

//---
sliderContainer.appendChild(document.createElement('hr'));
createSliderWithLabel2(sliderContainer, 10, 60, fov, 0.1, "fov", function (v) { fov = v; upd
```

}

// Dessiner la lettre T dans un espace 3D function fillGeometryCoordinates(gl) { let depth = 10; //TODO function sideRectangle(p1, p2, direction) { let r = [];

```
    if (direction) {
        r = [
            p2[0], p2[1], p2[2],
            p1[0], p1[1], p1[2],
            p1[0], p1[1], p1[2] + depth,

            p1[0], p1[1], p1[2] + depth,
            p2[0], p2[1], p2[2] + depth,
            p2[0], p2[1], p2[2],

        ];
    }
    else {
        r = [
            p2[0], p2[1], p2[2],
            p1[0], p1[1], p1[2] + depth,
            p1[0], p1[1], p1[2],

            p1[0], p1[1], p1[2] + depth,
            p2[0], p2[1], p2[2],
            p2[0], p2[1], p2[2] + depth,

        ];

    }
    return r;
}
//TODO
function getRectanglePoints(x, y, z, w, h, direction) {
    let r = [];
    if (direction) {
        r = [
            x, y, z,
            x, y + h, z,
            x + w, y, z,
            x + w, y, z,
            x, y + h, z,
            x + w, y + h, z,
        ]

    } else {
        r = [
            x, y, z,
            x + w, y, z,
            x, y + h, z,
            x + w, y, z,
            x + w, y + h, z,
            x, y + h, z,
        ]

    }

    return r;
}


gl.bufferData(
    gl.ARRAY_BUFFER,
    new Float32Array([
        ...getRectanglePoints(0, 0, 0, 50, 10, true),
        ...getRectanglePoints(20, 10, 0, 10, 40, true),
        ...getRectanglePoints(0, 0, depth, 50, 10, false),
        ...getRectanglePoints(20, 10, depth, 10, 40, false),

        ...sideRectangle([0, 0, 0], [0, 10, 0], true),
        ...sideRectangle([50, 0, 0], [50, 10, 0], false),
        ...sideRectangle([20, 10, 0], [20, 10 + 40, 0], true),
        ...sideRectangle([20 + 10, 10, 0], [20 + 10, 10 + 40, 0], false),

        ...sideRectangle([0, 0, 0], [50, 0, 0], false),

        ...sideRectangle([0, 10, 0], [20, 10, 0], true),
        ...sideRectangle([30, 10, 0], [50, 10, 0], true),

        //...sideRectangle([20, 10, 0], [30, 10, 0], false),
        ...sideRectangle([20, 50, 0], [30, 50, 0], true),


    ]),
    gl.STATIC_DRAW);
```

}

function updateScene() { webglUtils.resizeCanvasToDisplaySize(gl.canvas); gl.viewport(0, 0, gl.canvas.width, gl.canvas.height); gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT); gl.enable(gl.CULL_FACE); gl.enable(gl.DEPTH_TEST);

```
gl.useProgram(program);

gl.enableVertexAttribArray(positionLocation);
gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);

var size = 3;          // 3 elements par iteration
var type = gl.FLOAT;   // type
var normalize = false; // pas de normalisation de vecteur
var stride = 0;        // 0 de décalage entre composants
var offset = 0;        // 0 décalage de départ
gl.vertexAttribPointer(
    positionLocation, size, type, normalize, stride, offset
);

// Turn on the color attribute
gl.enableVertexAttribArray(colorLocation);
// Bind the color buffer.
gl.bindBuffer(gl.ARRAY_BUFFER, colorBuffer);

// Tell the attribute how to get data out of colorBuffer (ARRAY_BUFFER)
var size = 3;                 // 3 components per iteration
var type = gl.UNSIGNED_BYTE;  // the data is 8bit unsigned values
var normalize = true;         // normalize the data (convert from 0-255 to 0-1)
var stride = 0;               // 0 = move forward size * sizeof(type) each iteration to get
var offset = 0;               // start at the beginning of the buffer
gl.vertexAttribPointer(
    colorLocation, size, type, normalize, stride, offset);

// Compute the matrix


let matrix = m4.identity();

//TODO
var aspect = gl.canvas.clientWidth / gl.canvas.clientHeight;

matrix = m4.multiply( matrix, m4.perspective(fov,aspect,near,far));
//---
matrix = m4.translate(matrix, translation[0], translation[1], translation[2]);
matrix = m4.xRotate(matrix, rotation[0] * Math.PI / 180);
matrix = m4.yRotate(matrix, rotation[1] * Math.PI / 180);
matrix = m4.zRotate(matrix, rotation[2] * Math.PI / 180);
matrix = m4.scale(matrix, scale[0], scale[1], scale[2]);

// Set the matrix.
gl.uniformMatrix4fv(matrixLocation, false, matrix);

var primitiveType = gl.TRIANGLES;
var offset = 0;
var count = 12 * 2 * 3;
gl.drawArrays(primitiveType, offset, count);
```

}

function setupGL() { var canvas = document.getElementById('canvas'); gl = canvas.getContext('webgl');

```
// Shader sources
var vertexShaderSource = `
            attribute vec4 a_position;
            attribute vec4 a_color;

            varying vec4 v_color;

            uniform mat4 u_matrix;
            void main() {
            // Transformation & Project
                vec4 position = u_matrix * a_position;
                //TODO
                gl_Position = position;

                v_color = a_color;
            //---
            }
        `;

var fragmentShaderSource = `
            precision mediump float;
            varying vec4 v_color;

            void main() {
                gl_FragColor = v_color;
            }
        `;

// Compile shader
function compileShader(source, type) {
    var shader = gl.createShader(type);
    gl.shaderSource(shader, source);
    gl.compileShader(shader);
    if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
        console.error('Erreur de compilation du shader: ' + gl.getShaderInfoLog(shader));
        gl.deleteShader(shader);
        return null;
    }
    return shader;
}

var vertexShader = compileShader(vertexShaderSource, gl.VERTEX_SHADER);
var fragmentShader = compileShader(fragmentShaderSource, gl.FRAGMENT_SHADER);

// Link shaders into a program
program = gl.createProgram();
gl.attachShader(program, vertexShader);
gl.attachShader(program, fragmentShader);
gl.linkProgram(program);

positionLocation = gl.getAttribLocation(program, "a_position");
colorLocation = gl.getAttribLocation(program, "a_color");
matrixLocation = gl.getUniformLocation(program, "u_matrix");
gammaLocation = gl.getUniformLocation(program, "u_gamma");


positionBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
fillGeometryCoordinates(gl);

// Create a buffer for colors.
colorBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, colorBuffer);
// Put the colors in the buffer.
setColors(gl);

gl.enable(gl.DEPTH_TEST);
```

}

const colors = [{ r: 255, g: 0, b: 0 }, // Rouge { r: 0, g: 255, b: 0 }, // Vert { r: 0, g: 0, b: 255 }, // Bleu { r: 255, g: 255, b: 0 }, // Jaune { r: 0, g: 255, b: 255 }, // Cyan { r: 255, g: 0, b: 255 }, // Magenta { r: 192, g: 192, b: 192 },// Gris { r: 128, g: 0, b: 0 }, // Marron { r: 128, g: 128, b: 0 }, // Olive { r: 0, g: 128, b: 0 }, // Vert foncé { r: 128, g: 0, b: 128 }, // Pourpre { r: 0, g: 128, b: 128 } // Sarcelle];

function colorForRectangle(colorIndex) { let r = []; for (let i = 0; i < 6; i++) { r.push(colors[colorIndex].r, colors[colorIndex].g, colors[colorIndex].b); } return r; } function setColors(gl) { gl.bufferData(gl.ARRAY_BUFFER, new Uint8Array([...colorForRectangle(0), ...colorForRectangle(1), ...colorForRectangle(2), ...colorForRectangle(3), ...colorForRectangle(4), ...colorForRectangle(5), ...colorForRectangle(6), ...colorForRectangle(7), ...colorForRectangle(8), ...colorForRectangle(9), ...colorForRectangle(10), ...colorForRectangle(11)]) , gl.STATIC_DRAW); }

function main() { setupGL(); setupUI(); updateScene(); }

```
main();
```