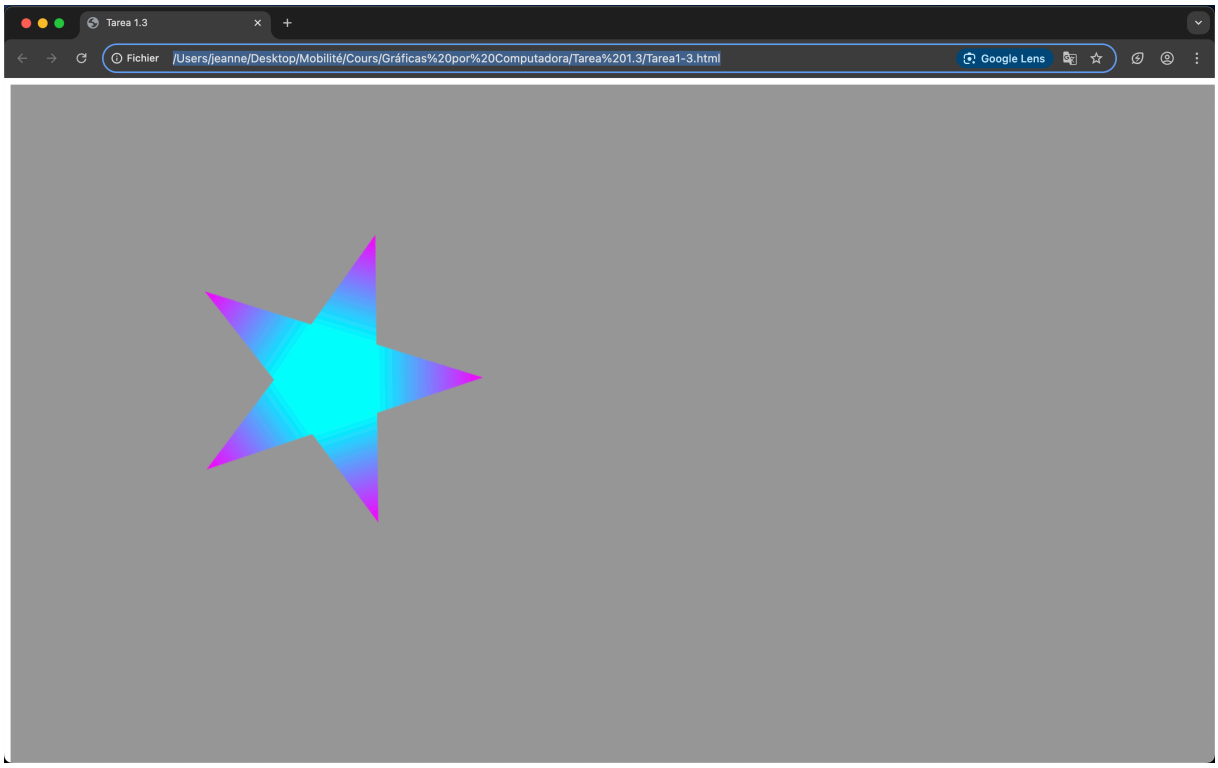
Tarea 1.3: Movimiento de un objeto 2D

Alumno: Jeanne LE GALL-BOTTE

El objetivo de este programa es generar una figura utilizando WebGL, aplicándo dos tipos de animaciones: por un lado una rotación constante sobre sí misma y, por otro, un movimiento de traslación dentro de los límites de la pantalla.



```html
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width,initial-scale=1" />
    <title>Tarea 1.3</title>
</head>

<body>
    <canvas id="canvas"></canvas>

    <script>
        var canvas = document.getElementById('canvas');
        var gl = canvas.getContext('webgl') || canvas.getContext('experimental-webgl');

        /*========== Resize function ==========*/
        var proj_matrix = new Float32Array(16);

        function resizeCanvas() {
            canvas.width = window.innerWidth;
            canvas.height = window.innerHeight;
            gl.viewport(0, 0, canvas.width, canvas.height);

            var aspect = canvas.width / canvas.height;
            if (aspect >= 1) {
                proj_matrix = new Float32Array([
                    1/aspect, 0, 0, 0,
                    0, 1, 0, 0,
                    0, 0, 1, 0,
                    0, 0, 0, 1
                ]);
            } else {
                proj_matrix = new Float32Array([
                    1, 0, 0, 0,
                    0, aspect, 0, 0,
                    0, 0, 1, 0,
                    0, 0, 0, 1
                ]);
            }
        }
        window.addEventListener('resize', resizeCanvas);
        resizeCanvas();
```

El programa comienza creando un canvas que ocupa todo el tamaño de la ventana. La función *resizeCanvas* se asegura de que, al cambiar las dimensiones de la ventana, el área de dibujo se ajuste correctamente y la estrella no se deforme. Para lograrlo, se calcula el aspect ratio y se modifica la matriz de proyección. Si la pantalla es más ancha que alta, se reduce la escala en el eje X; si es más alta que ancha, se corrige el eje Y. Así, la estrella conserva sus proporciones originales en cualquier dispositivo.

```
/*========== Geometry ==========*/
var vertices = [
    0.1, -0.15, 0.0,
    -0.1, -0.15, 0.0,
    -0.16, 0.04, 0.0,
    0.0, 0.15, 0.0,
    0.16, 0.04, 0.0,
    -0.26, 0.35, 0.0,
    0.26, 0.35, 0.0,
    0.42, -0.15, 0.0,
    0.0, -0.46, 0.0,
    -0.42, -0.15, 0.0
];

var positionStar = {x : 0, y : 0};
var speed = {x: 0.01, y : 0.01};

var indices = [0,2,3, 0,3,4, 0,1,2, 2,3,5, 3,4,6, 0,4,7, 0,1,8, 1,2,9];

var colors = [0,1,1, 0,1,1, 0,1,1, 0,1,1, 0,1,1, 1,0,1, 1,0,1, 1,0,1, 1,0,1, 1,0,1]

var vertex_buffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, vertex_buffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);

var index_buffer = gl.createBuffer();
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, index_buffer);
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(indices), gl.STATIC_DRAW);

var color_buffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, color_buffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors), gl.STATIC_DRAW);
```

La estrella está formada por diez puntos. Estos puntos se almacenan en un buffer de vértices. Sin embargo, WebGL solo sabe dibujar triángulos, por lo que se necesita un arreglo de índices que indica qué vértices deben unirse para formar las superficies de la estrella. Por ejemplo, el índice 0,2,3 representa un triángulo formado por los vértices 0, 2 y 3.

Además, a cada vértice se le asigna un color. De esta manera, cuando WebGL interpola los colores entre los vértices de cada triángulo, la estrella adquiere un aspecto con matices en lugar de ser de un único color plano.

```
/*========== Shaders ==========*/
var vertCode =
    'uniform vec4 translation;' +
    'uniform mat4 Pmatrix;' +
    'uniform mat4 Vmatrix;' +
    'uniform mat4 Mmatrix;' +
    'attribute vec3 position;' +
    'attribute vec3 color;' +
    'varying vec3 vColor;' +
    'void main(void) {' +
    ' gl_Position = Pmatrix*Vmatrix*Mmatrix*vec4(position, 1.0) + translation;' +
    ' vColor = color;' +
    '}';

var fragCode =
    'precision mediump float;'+
    'varying vec3 vColor;'+
    'void main(void) {'+
    ' gl_FragColor = vec4(vColor, 1.0);'+
    '}';

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

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

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

var Pmatrix = gl.getUniformLocation(shaderProgram, "Pmatrix");
var Vmatrix = gl.getUniformLocation(shaderProgram, "Vmatrix");
var Mmatrix = gl.getUniformLocation(shaderProgram, "Mmatrix");
gl.bindBuffer(gl.ARRAY_BUFFER, vertex_buffer);
```

El vertex shader recibe como atributos la posición y el color de cada vértice. Aplica tres matrices Pmatrix, Vmatrix y Mmatrix que representan respectivamente la proyección, la vista y el modelo. Después añade un vector de traslación para mover toda la figura. Finalmente, envía el color interpolado hacia el siguiente paso de la canalización gráfica.

El fragment shader es sencillo. Su única función es tomar el color interpolado recibido del vertex shader y asignarlo a cada fragmento, es decir, a cada píxel que compone la figura.

Ambos shaders son compilados y unidos en un programa que será usado para todos los dibujos.

```
var position = gl.getAttribLocation(shaderProgram, "position");
gl.vertexAttribPointer(position, 3, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(position);

gl.bindBuffer(gl.ARRAY_BUFFER, color_buffer);
var color = gl.getAttribLocation(shaderProgram, "color");
gl.vertexAttribPointer(color, 3, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(color);

gl.useProgram(shaderProgram);

//Rotation
var mov_matrix = [1,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,1];
function rotateZ(m, angle) {
    var c = Math.cos(angle);
    var s = Math.sin(angle);
    var mv0 = m[0], mv4 = m[4], mv8 = m[8];

    m[0] = c*m[0]-s*m[1];
    m[4] = c*m[4]-s*m[5];
    m[8] = c*m[8]-s*m[9];
    m[1] = c*m[1]+s*mv0;
    m[5] = c*m[5]+s*mv4;
    m[9] = c*m[9]+s*mv8;
}

var view_matrix = [
    1,0,0,0,
    0,1,0,0,
    0,0,1,0,
    0,0,0,1
];
```

La rotación está controlada por una matriz llamada *mov_matrix*. Al inicio esta matriz es la identidad, es decir, no aplica ninguna transformación. En cada cuadro de la animación se llama a la función *rotateZ*, que modifica la matriz para girar todos los vértices alrededor del eje Z.

El ángulo de rotación no es fijo, sino que depende del tiempo transcurrido entre un cuadro y el siguiente (*dt*). De este modo, aunque la velocidad de la computadora o la tasa de refresco varíe, la estrella mantiene siempre una rotación fluida y constante.

```
//Translation
var translation = gl.getUniformLocation(shaderProgram, "translation");

/*========== Draw function ==========*/
var time_old = 0;
var animate = function(time) {
    var dt = time-time_old;
    rotateZ(mov_matrix, dt*0.002);
    time_old = time;

    // Calcul du ratio actuel
    var aspect = canvas.width / canvas.height;

    // Limites de l'étoile adaptées à l'aspect
    var limitX = 0.42;
    var limitY = 0.405;
    if(aspect >= 1) limitX /= aspect;
    else limitY *= aspect;

    positionStar.x += speed.x;
    positionStar.y += speed.y;

    if(positionStar.x + limitX > 1 || positionStar.x - limitX < -1) speed.x *= -1;
    if(positionStar.y + limitY > 1 || positionStar.y - limitY < -1) speed.y *= -1;

    gl.uniform4f(translation, positionStar.x, positionStar.y, 0.0, 0.0);
```

```
    gl.enable(gl.DEPTH_TEST);
    gl.depthFunc(gl.LEQUAL);
    gl.clearColor(0.5, 0.5, 0.5, 0.9);
    gl.clearDepth(1.0);
    gl.viewport(0, 0, canvas.width, canvas.height);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    gl.uniformMatrix4fv(Pmatrix, false, proj_matrix);
    gl.uniformMatrix4fv(Vmatrix, false, view_matrix);
    gl.uniformMatrix4fv(Mmatrix, false, mov_matrix);

    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, index_buffer);
    gl.drawElements(gl.TRIANGLES, indices.length, gl.UNSIGNED_SHORT, 0);

    window.requestAnimationFrame(animate);
}
animate(0);
```

Además de rotar, la estrella se traslada por la pantalla. Esto se hace actualizando sus coordenadas centrales (*positionStar.x* y *positionStar.y*) sumando en cada paso una velocidad (*speed.x*, *speed.y*).

Para que la estrella no salga de la ventana, se definen unos límites (*limitX*, *limitY*) que corresponden al radio de la figura. Cuando la posición sumada al límite sobrepasa el valor 1 o -1, se invierte la velocidad en ese eje. Así se consigue el efecto de rebote: la estrella choca contra el borde y cambia de dirección automáticamente.

Toda la animación se desarrolla dentro de la función *animate*, llamada continuamente por *requestAnimationFrame*. En cada ejecución se realizan cuatro pasos:

- Se calcula el tiempo transcurrido y se actualiza la rotación.
- Se incrementa la posición de la estrella y se comprueba si toca los bordes para invertir su velocidad.
- Se limpia la pantalla y se reenvían las matrices de transformación al shader.
- Se dibuja la estrella mediante los índices que forman sus triángulos.

Este ciclo se repite indefinidamente, dando lugar a un movimiento fluido donde la estrella gira y rebota sin cesar.