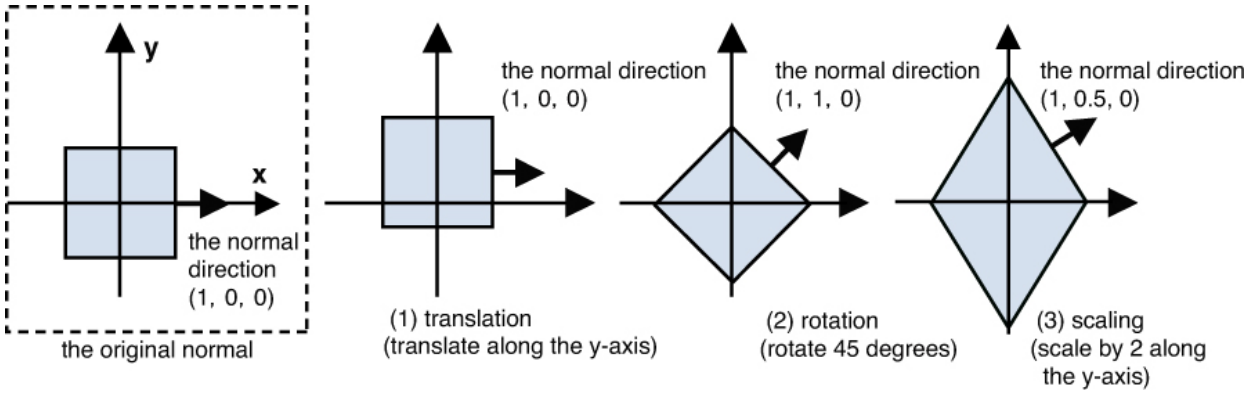
Universidad
Rey Juan Carlos
Escuela Técnica Superior
Ingeniería de Telecomunicación



Gráficos y visualización 3D

4. Transformaciones con WebGL

JOSÉ MIGUEL GUERRERO HERNÁNDEZ

EMAIL: JOSEMIGUEL.GUERRERO@URJC.ES

Índice de contenidos

Índice de contenidos

1. Introducción - Geometría euclidiana

- La **geometría** es una rama de las matemáticas que se ocupa del estudio de las propiedades de las figuras en el plano o el espacio, incluyendo: puntos, rectas, planos, polígonos, poliedros, etc.
- El matemático griego del siglo III a.C. **Euclides** es considerado el padre de la geometría

Euclides

1. Introducción - Geometría euclidiana

- La obra de Euclides titulada "**Los Elementos**" presenta un conjunto de axiomas (llamados postulados) que definen la geometría euclidiana (conocida simplemente como geometría hasta hace 2 siglos)
- Esta obra es considerada una de las obras científicas más importantes de la humanidad, ya que es fundamental en otros campos del conocimiento tales como la física, química, astronomía, ingeniería, etc.

1. Introducción - Geometría euclidiana

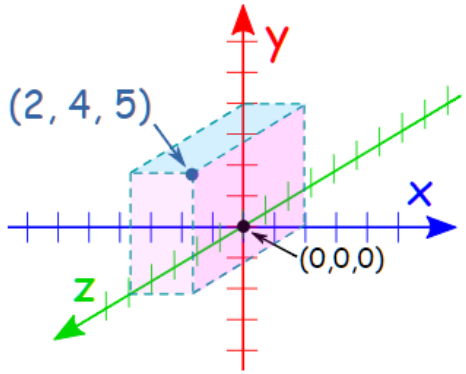
Los 5 postulados de Euclides son:

1. Desde cualquier punto se puede trazar una recta a cualquier otro punto
2. Toda recta se puede prolongar indefinidamente
3. Con cualquier centro y cualquier distancia se puede trazar un círculo
4. Todos los ángulos rectos son iguales
5. Por un punto exterior a una recta se puede trazar una y sola una paralela a dicha recta

https://www.youtube.com/watch?v=EPV-7cj8Ej8

1. Introducción - Geometría euclidiana

- La geometría de Euclides es una abstracción de la realidad que funciona muy bien con el mundo que percibimos (teorema de Pitágoras, física clásica, …)
- Las **coordenadas cartesianas** son un tipo de coordenadas ortogonales usadas en espacios euclídeos

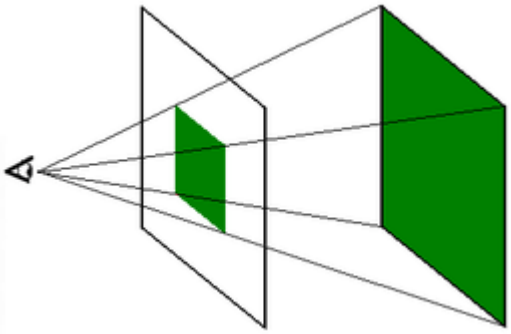


1. Introducción - Geometría no euclidiana

- A partir del siglo XIX diversos matemáticos como Gauss o Lobachevski comenzaron a concebir nuevos tipos de geometría que no cumplían el quinto postulado de Euclides
- De esta forma surgen, por ejemplo:
  - Geometría esférica (usada en la navegación y en la astronomía)
  - Geometría hiperbólica (usada en la teoría general de la relatividad)

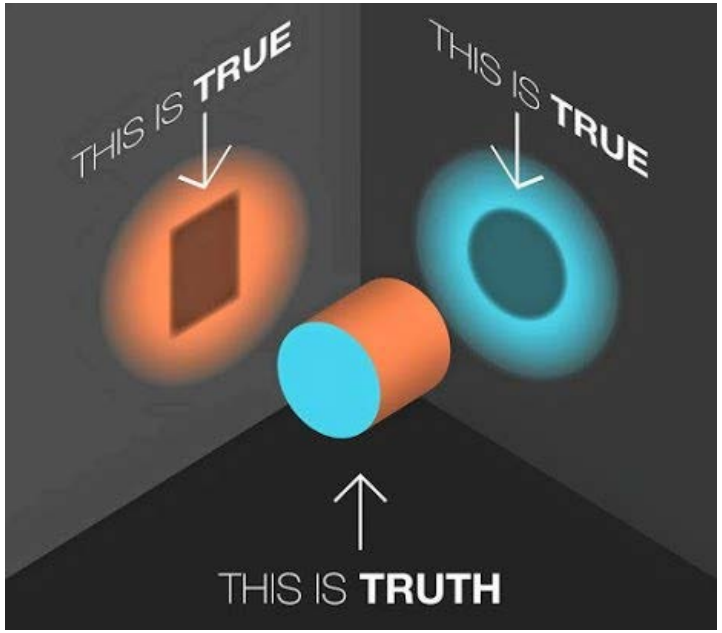
1. Introducción - Geometría proyectiva

- En gráficos 3D nos interesa una forma de geometría llamada **proyectiva**
- La geometría proyectiva se define en el contexto de la geometría euclidiana y tiene sus orígenes en la pintura del Renacimiento, al investigar la visión que nuestro ojo tiene de una figura cuando la vemos en distintos planos colocados entre ella y nosotros (**perspectiva**)



1. Introducción - Geometría proyectiva

- En geométrica proyectiva, el **espacio proyectivo** se relaciona con la forma en la que un ojo o una cámara proyecta una escena 3D sobre una imagen 2D



1. Introducción - Coordenadas homogéneas

- Las **coordenadas homogéneas** son un instrumento usado en geometría para describir un punto en el espacio proyectivo
- Fueron introducidas por el matemático alemán August Ferdinand Möbius en el año 1837
- La representación mediante coordenadas homogéneas de objetos en un espacio dimensional se realiza a través de coordenadas de un espacio (n+1)-dimensional

1. Introducción - Coordenadas homogéneas

- En el sistema de coordenadas homogéneas se introduce una cuarta dimensión **w** denominada componente hiperespacial o factor de escala
- Por ejemplo, dado un punto **p** en el espacio 3D determinado por sus coordenadas cartesianas x, y, z. Ese mismo punto en coordenadas homogéneas se representaría como:

$$p(x, y, z) = p(xw, yw, zw, w)$$

1. Introducción - Coordenadas homogéneas

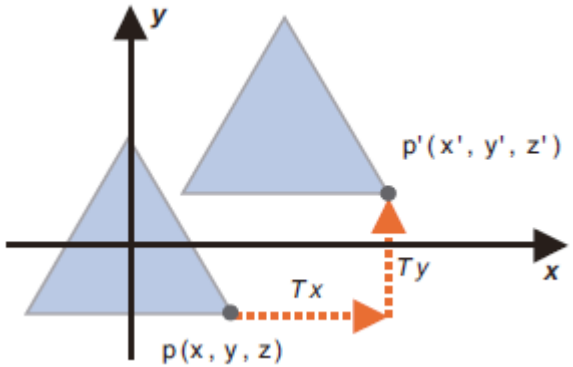
- Las coordenadas homogéneas nos va a permitir realizar transformaciones de manera sencilla al trabajar con matrices

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} aw \\ bw \\ cw \\ w \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \\ 1 \end{bmatrix}$$

Índice de contenidos

2. Transformaciones básicas - traslación

- La **traslación** es una transformación lineal que supone el desplazamiento (Tx, Ty, Tz) de un vértice (x, y, z)



$$x' = x + Tx$$

$$y' = y + Ty$$

$$z' = z + Tz$$

2. Transformaciones básicas - traslación

- Para realizar las operaciones de transformación es muy útil trabajar con matrices y coordenadas homogéneas

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & Tx \\ 0 & 1 & 0 & Ty \\ 0 & 0 & 1 & Tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Coordenadas trasladadas (vector)

Matriz de transformación

Coordenadas originales (vector)

2. Transformaciones básicas - traslación

- La multiplicación de matrices se calcula:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1p} \\ a_{21} & a_{22} & \dots & a_{2p} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mp} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \dots & \dots & \dots & \dots \\ b_{p1} & b_{p2} & \dots & b_{pn} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \dots & \dots & \dots & \dots \\ c_{m1} & c_{m2} & \dots & c_{mn} \end{bmatrix}$$

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + \dots + a_{1p}b_{p1}$$

La multiplicación de matrices se realiza de forma muy eficiente en la GPU

- Aplicado a la multiplicación de traslación:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & Tx \\ 0 & 1 & 0 & Ty \\ 0 & 0 & 1 & Tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

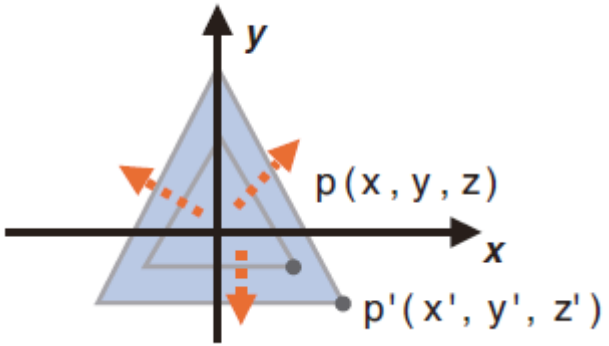
$$x' = x + Tx$$

$$y' = y + Ty$$

$$z' = z + Tz$$

2. Transformaciones básicas - escalado

- El **escalado** es una transformación lineal que aumenta (incrementa) o contrae (disminuye) las coordenadas de un vértice (x, y, z) en un factor (Sx, Sy, Sz)



$$x' = S_x \times x$$

$$y' = S_y \times y$$

$$z' = S_z \times z$$

2. Transformaciones básicas - escalado

- El escalado también se puede representar mediante una matriz de transformación

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

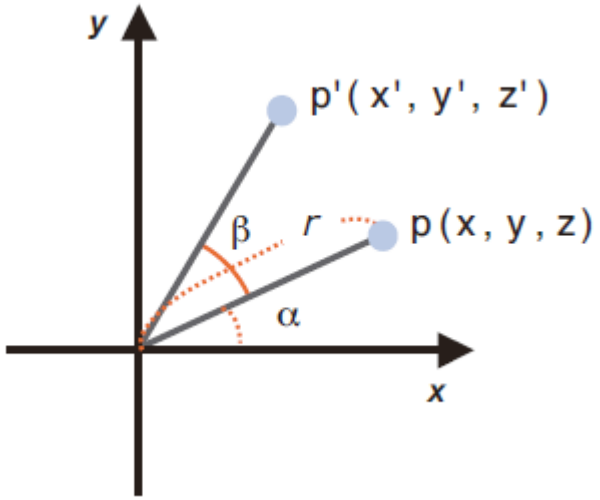
Coordenadas escaladas (vector)

Matriz de transformación

Coordenadas originales (vector)

2. Transformaciones básicas - rotación

- La **rotación** es una transformación lineal que gira las coordenadas de un vértice (x, y, z) un ángulo β



$$x' = x \cos \beta - y \sin \beta$$

$$y' = x \sin \beta + y \cos \beta$$

$$z' = z$$

2. Transformaciones básicas - rotación

- La matrices de transformación de la rotación son:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta & 0 & 0 \\ \sin\Theta & \cos\Theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Rotación en el **eje Z**

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} \cos\Theta & 0 & \sin\Theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\Theta & 0 & \cos\Theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Rotación en el **eje Y**

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\Theta & -\sin\Theta & 0 \\ 0 & \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Rotación en el **eje X**

Índice de contenidos

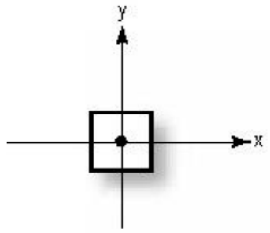
3. Cadena de transformaciones

- Para realizar una cadena de transformación (o sea, realizar una transformación y después otra) simplemente hay que **multiplicar las matrices de transformación** de las transformaciones básicas
- Por ejemplo, para escalar y trasladar:

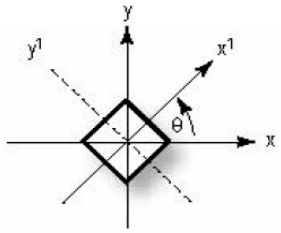
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & Tx \\ 0 & 1 & 0 & Ty \\ 0 & 0 & 1 & Tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

3. Cadena de transformaciones

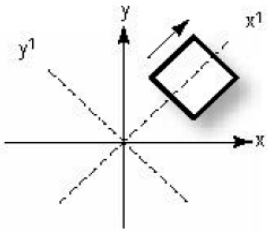
- El orden en el que se aplican las transformaciones sobre los ejes, influye en el resultado final:



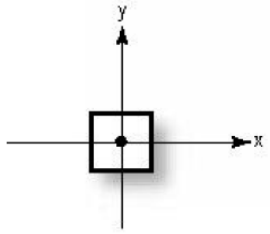
Cuadrado inicial



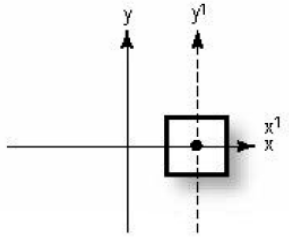
Rotación sobre z, (nuevo eje x^1)



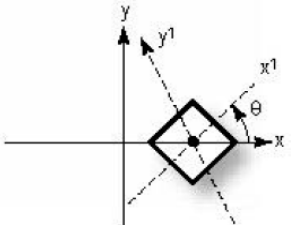
Traslación sobre x (ahora es x^1)



Cuadrado inicial



Traslación sobre x



Se rotan las nuevas coordenadas

Índice de contenidos

4. Librería JavaScript glMatrix - Introducción

- Como hemos visto, para realizar las transformaciones básicas vistas (traslación, escalado y rotación) necesitaremos calcular la matriz de transformación
- Esta matriz se puede en JavaScript mediante arrays de 4 dimensiones
- En lugar de hacerlo de forma manual, vamos a usar una librería JavaScript específica para el uso de matrices: **glMatrix**

4. Librería JavaScript glMatrix - Introducción

- glMatrix es una librería *open source* que nos permite trabajar con matrices y vectores en JavaScript de manera sencilla
- Para usar glMatrix, simplemente hay que enlazar la librería JavaScript, por ejemplo usando la URL de una CDN:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/gl-matrix/2.8.1/gl-matrix-min.js"></script>
```

http://glmatrix.net/

4. Librería JavaScript glMatrix - Introducción

- En particular, nos va a interesar la creación de matrices de transformación de 4 dimensiones, necesarias para las transformaciones básicas (traslación, escalado y rotación) en coordenadas homogéneas
- Para ello usaremos el objeto `mat4` creado por glMatrix al cargar la librería JavaScript

4. Librería JavaScript glMatrix - Referencia

(static) **create()** → {mat4}

Creates a new identity mat4

Source: mat4.js, line 13

Returns:

a new 4x4 matrix

Type

mat4

Crea la matriz **identidad** de 4 dimensiones:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

http://glmatrix.net/docs/module-mat4.html

4. Librería JavaScript glMatrix - Referencia

(static) fromTranslation(out, v) → {mat4}

Creates a matrix from a vector translation This is equivalent to (but much faster than): mat4.identity(dest); mat4.translate(dest, dest, vec);

Parameters:

Name	Type	Description
out	mat4	mat4 receiving operation result
v	vec3	Translation vector

Source: mat4.js, line 675

Returns:

out

Type
mat4

Crea la matriz de transformación para realizar la **traslación** a la posición (Tx, Ty, Tz)

http://glmatrix.net/docs/module-mat4.html

4. Librería JavaScript glMatrix - Referencia

(static) fromScaling(out, v) → {mat4}

Creates a matrix from a vector scaling This is equivalent to (but much faster than): mat4.identity(dest); mat4.scale(dest, dest, vec);

Parameters:

Name	Type	Description
out	mat4	mat4 receiving operation result
v	vec3	Scaling vector

Source: mat4.js, line 706

Returns:

out

Type
mat4

Crea la matriz de transformación para realizar un **escalado** de factor en un factor (Sx, Sy, Sz)

http://glmatrix.net/docs/module-mat4.html

4. Librería JavaScript glMatrix - Referencia

(static) fromXRotation(out, rad) → {mat4}

Creates a matrix from the given angle around the X axis This is equivalent to (but much faster than): mat4.identity(dest); mat4.rotateX(dest, dest, rad);

Parameters:

Name	Type	Description
out	mat4	mat4 receiving operation result
rad	Number	the angle to rotate the matrix by

Source: mat4.js, line 785

Returns:

out

Type
mat4

Crea la matriz de transformación para realizar una **rotación** en el eje **X** en un ángulo rad

http://glmatrix.net/docs/module-mat4.html

4. Librería JavaScript glMatrix - Referencia

(static) fromYRotation(out, rad) → {mat4}

Creates a matrix from the given angle around the Y axis This is equivalent to (but much faster than): mat4.identity(dest); mat4.rotateY(dest, dest, rad);

Parameters:

Name	Type	Description
out	mat4	mat4 receiving operation result
rad	Number	the angle to rotate the matrix by

Source: mat4.js, line 820

Returns:

out

Type
mat4

Crea la matriz de transformación para realizar una **rotación** en el eje **Y** en un ángulo rad

http://glmatrix.net/docs/module-mat4.html

4. Librería JavaScript glMatrix - Referencia

(static) fromZRotation(out, rad) → {mat4}

Creates a matrix from the given angle around the Z axis This is equivalent to (but much faster than): mat4.identity(dest); mat4.rotateZ(dest, dest, rad);

Parameters:

Name	Type	Description
out	mat4	mat4 receiving operation result
rad	Number	the angle to rotate the matrix by

Source: mat4.js, line 855

Returns:

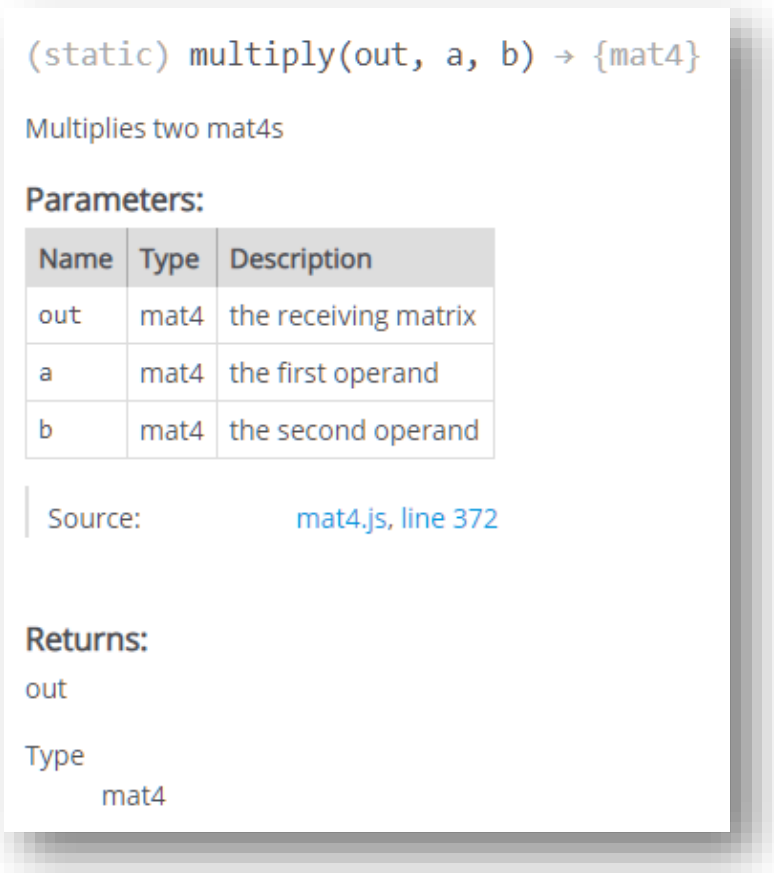
out

Type
mat4

Crea la matriz de transformación para realizar una **rotación** en el eje **Z** en un ángulo rad

http://glmatrix.net/docs/module-mat4.html

4. Librería JavaScript glMatrix - Referencia



(static) multiply(out, a, b) → {mat4}

Multiplies two mat4s

Parameters:

Name	Type	Description
out	mat4	the receiving matrix
a	mat4	the first operand
b	mat4	the second operand

Source: mat4.js, line 372

Returns:

out

Type
mat4

Multiplica dos matrices

http://glmatrix.net/docs/module-mat4.html

Índice de contenidos

5. Referencia API WebGL

`gl.uniformMatrix4fv``(location, transpose, array)`

Assign the 4×4 matrix specified by *array* to the uniform variable specified by *location*.

Parameters	location	Specifies the storage location of the uniform variable.
	Transpose	Must be `false` in WebGL.[3]
	array	Specifies an array containing a 4×4 matrix in column major order (typed array).
Return value	None	
Errors	INVALID_OPERATION	There is no current program object.
	INVALID_VALUE	*transpose* is not `false`, or the length of *array* is less than 16.

[3] This parameter specifies whether to transpose the matrix or not. The transpose operation, which exchanges the column and row elements of the matrix (see Chapter 7), is not supported by WebGL's implementation of this method and must always be set to `false`.

Índice de contenidos

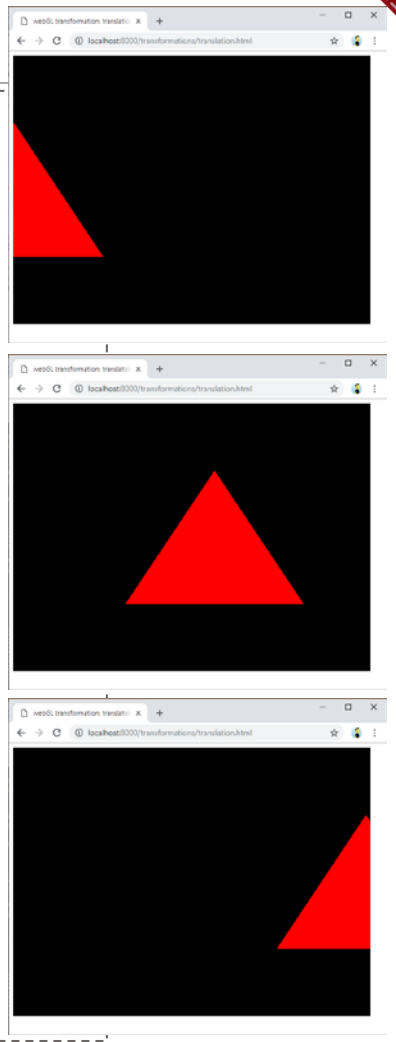
6. Ejemplo: traslación

```
<!DOCTYPE html>
<html>
<head>
    <title>WebGL transformations: translation</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/gl-matrix/2.8.1/gl-matrix-min.js"></script>
    <script id="shaderVs" type="x-shader/x-vertex">
        attribute vec4 a_Position;
        uniform mat4 u_Matrix;
        void main() {
            gl_Position = u_Matrix * a_Position;
        }
    </script>
    <script id="shaderFs" type="x-shader/x-fragment">
        void main() {
            gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
        }
    </script>
    <script>
        var gl;
        var count = 0.0;

        function init() {
            // ...
        }
    </script>
</head>

<body onload="init()">
    <canvas id="myCanvas" width="640" height="480"></canvas>
</body>

</html>
```



6. Ejemplo: traslación

```
<!DOCTYPE html>
<html>
<head>
    <title>WebGL transformations: translation</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/gl-matrix/2.8.1/gl-matrix-min.js"></script>
    <script id="shaderVs" type="x-shader/x-vertex">
        attribute vec4 a_Position;
        uniform mat4 u_Matrix;
        void main() {
            gl_Position = u_Matrix * a_Position;
        }
    </script>
    <script id="shaderFs" type="x-shader/x-fragment">
        void main() {
            gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
        }
    </script>
    <script>
        var gl;
        var count = 0.0;

        function init() {
            // ...
        }
    </script>
</head>

<body onload="init()">
    <canvas id="myCanvas" width="640" height="480"></canvas>
</body>

</html>
```

Se obtiene la posición inicial de los vértices mediante un vector de tipo `attribute`

Se obtiene la matriz de transformación mediante una matriz de tipo `uniform`

Se calcula la traslación mediante la multiplicación de matrices

Se inicializa la variable global JavaScript `count` que hará de contador

6. Ejemplo: traslación

```
function init() {
    // Get canvas,
    // init WebGL context,
    // and shaders ...
    // Draw Scene
    drawScene();
}

function drawScene() {
    // Clear canvas
    gl.clear(gl.COLOR_BUFFER_BIT);

    // Calcualte position
    var position = Math.sin(count);
    var matrix = mat4.fromTranslation(mat4.create(), [position, 0.0, 0.0]);

    // Set uniform value (u_Matrix) in vertex shader
    var mvMatrixUniform = gl.getUniformLocation(gl.program, "u_Matrix");
    gl.uniformMatrix4fv(mvMatrixUniform, false, matrix);

    // Draw
    gl.drawArrays(gl.TRIANGLES, 0, 3);

    // Call drawScene again in the next brows
    count += 0.01;
    requestAnimationFrame(drawScene);
}
```

Usamos el método `fromTranslation` de `mat4` (glMatrix) para la creación de la matriz de transformación

Se escribe el valor de la matriz de transformación en la variable *uniform* del vertex shader llamada `u_Matrix`

Se invoca el método de dibujo WebGL (`drawArrays`) y se vuelve a invocar el mismo método cuando el navegador esté en disposición de dibujar otra vez (`requestAnimationFrame`) incrementando una variable contador (`count`)

Índice de contenidos

7. Ejemplo: escalado

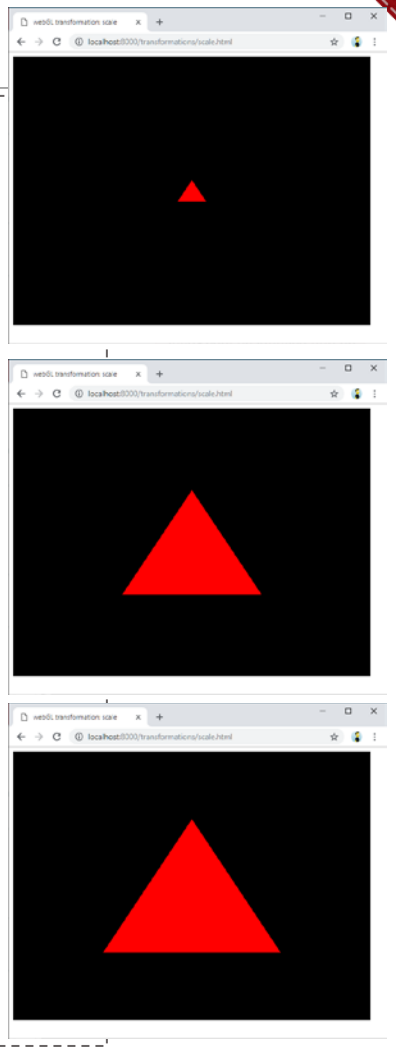
```
<!DOCTYPE html>
<html>
<head>
    <title>WebGL transformations: translation</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/gl-matrix/2.8.1/gl-matrix-min.js"></script>
    <script id="shaderVs" type="x-shader/x-vertex">
        attribute vec4 a_Position;
        uniform mat4 u_Matrix;
        void main() {
            gl_Position = u_Matrix * a_Position;
        }
    </script>
    <script id="shaderFs" type="x-shader/x-fragment">
        void main() {
            gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
        }
    </script>
    <script>
        var gl;
        var count = 0.0;

        function init() {
            // ...
        }
    </script>
</head>

<body onload="init()">
    <canvas id="myCanvas" width="640" height="480"></canvas>
</body>

</html>
```

La definición de los shaders y la estructura del código JavaScript es idéntica que en el ejemplo anterior



7. Ejemplo: escalado

```
function init() {
    // Get canvas, init WebGL context, and shaders ...

    // Draw Scene
    drawScene();
}

function drawScene() {
    // Clear canvas
    gl.clear(gl.COLOR_BUFFER_BIT);

    // Calcualte new scale
    var scale = Math.abs(Math.sin(count));
    var matrix = mat4.fromScaling(mat4.create(), [scale, scale, 0.0]);

    // Set uniform value (u_Matrix) in vertex shader
    var mvMatrixUniform = gl.getUniformLocation(gl.program, "u_Matrix");
    gl.uniformMatrix4fv(mvMatrixUniform, false, matrix);

    // Draw
    gl.drawArrays(gl.TRIANGLES, 0, 3);

    // Call drawScene again in the next browser repaint
    count += 0.01;
    requestAnimationFrame(drawScene);
}
```

Usamos el método `fromScaling` de `mat4` (glMatrix) para la creación de la matriz de transformación

El resto de la lógica de este método es igual que en los ejemplos anteriores

Índice de contenidos

8. Ejemplo: rotación

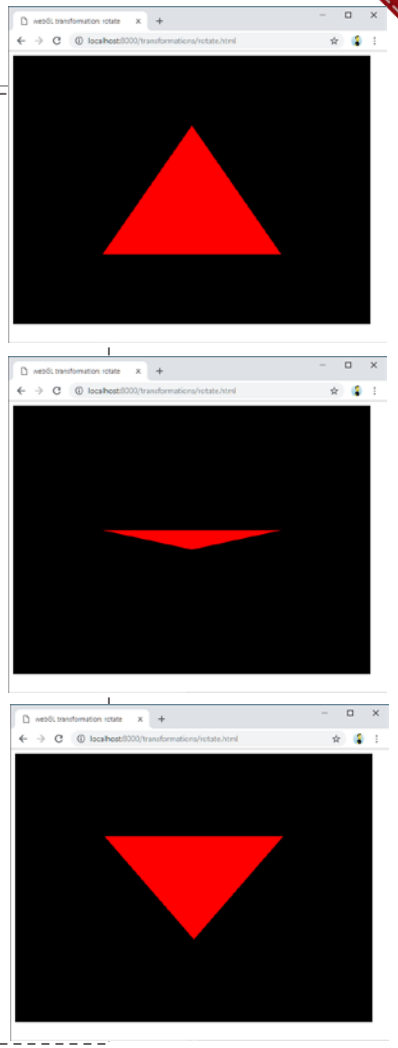
```
<!DOCTYPE html>
<html>
<head>
    <title>WebGL transformations: translation</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/gl-matrix/2.8.1/gl-matrix-min.js"></script>
    <script id="shaderVs" type="x-shader/x-vertex">
        attribute vec4 a_Position;
        uniform mat4 u_Matrix;
        void main() {
            gl_Position = u_Matrix * a_Position;
        }
    </script>
    <script id="shaderFs" type="x-shader/x-fragment">
        void main() {
            gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
        }
    </script>
    <script>
        var gl;
        var count = 0.0;

        function init() {
            // ...
        }
    </script>
</head>

<body onload="init()">
    <canvas id="myCanvas" width="640" height="480"></canvas>
</body>

</html>
```

La definición de los shaders y la estructura del código JavaScript es idéntica que en los ejemplos anteriores



8. Ejemplo: rotación

```
function init() {
    // Get canvas, init WebGL context, and shaders ...

    // Draw Scene
    drawScene();
}

function drawScene() {
    // Clear canvas
    gl.clear(gl.COLOR_BUFFER_BIT);

    // Calculate angle
    var matrix = mat4.fromXRotation(mat4.create(), count);

    // Set uniform value (u_Matrix) in vertex shader
    var mvMatrixUniform = gl.getUniformLocation(gl.program, "u_Matrix");
    gl.uniformMatrix4fv(mvMatrixUniform, false, matrix);

    // Draw
    gl.drawArrays(gl.TRIANGLES, 0, 3);

    // Call drawScene again in the next browser repaint
    count += 0.01;
    requestAnimationFrame(drawScene);
}
```

Usamos el método `fromXRotation` de `mat4` (glMatrix) para la creación de la matriz de transformación

El resto de la lógica de este método es igual que en los ejemplos anteriores

Índice de contenidos

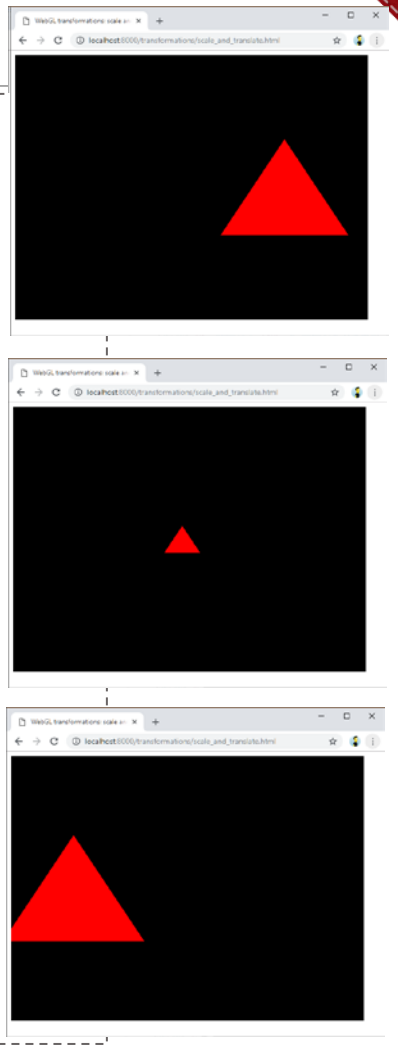
9. Ejemplo: traslación y escalado

```
<!DOCTYPE html>
<html>
<head>
    <title>WebGL transformations: translation</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/gl-matrix/2.8.1/gl-matrix-min.js"></script>
    <script id="shaderVs" type="x-shader/x-vertex">
        attribute vec4 a_Position;
        uniform mat4 u_Matrix;
        void main() {
            gl_Position = u_Matrix * a_Position;
        }
    </script>
    <script id="shaderFs" type="x-shader/x-fragment">
        void main() {
            gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
        }
    </script>
    <script>
        var gl;
        var count = 0.0;

        function init() {
            // ...
        }
    </script>
</head>

<body onload="init()">
    <canvas id="myCanvas" width="640" height="480"></canvas>
</body>

</html>
```



9. Ejemplo: traslación y escalado

```
function drawScene() {
        // Clear canvas
        gl.clear(gl.COLOR_BUFFER_BIT);

        // Calcualte new scale
        var scale = Math.abs(Math.sin(count));
        var position = Math.sin(count);

        var scaled = mat4.fromScaling(mat4.create(), [scale, scale, 0.0]);
        var translated = mat4.fromTranslation(mat4.create(), [position, 0.0, 0.0]);
        var matrix = mat4.multiply(mat4.create(), scaled, translated);

        // Set uniform value (u_Matrix) in vertex shader
        var mvMatrixUniform = gl.getUniformLocation(gl.program, "u_Matrix");
        gl.uniformMatrix4fv(mvMatrixUniform, false, matrix);

        // Draw
        gl.drawArrays(gl.TRIANGLES, 0, 3);

        // Call drawScene again in the next brows
        count += 0.01;
        requestAnimationFrame(drawScene);
    }
```

En primer lugar calculamos las matrices de transformación para cada uno de las transformaciones básicas (traslación y escalado)

Después hacemos uso del método `multiply` de glMatrix para multiplicar las matrices

La multiplicación de matrices es un proceso costoso incluso para las GPUs. Por esta razón, para gráficos complejos se intenta reutilizar la cadena de transformaciones en la medida de lo posible

Índice de contenidos

10. Resumen

- A un conjunto de vértices en WebGL se le pueden aplicar varias transformaciones básicas:
  - **Traslación**: movimiento (Tx, Ty, Tz)
  - **Escalado**: aumento o contracción (Sx, Sy, Sz)
  - **Rotación**: giro (βx, βy, βz)
- A cada una de estas transformaciones básicas le corresponde una **matriz de transformación**
- La generación de estas matrices es muy sencilla en JavaScript usando una librarías, como glMatrix

10. Resumen

- El uso de **coordenadas homogéneas** en WebGL (4 dimensiones para describir un punto del espacio 3D) hace muy sencillo realizar el cálculo de las transformaciones básicas multiplicando las coordenadas originales y la matriz de transformación
- Para **encadenar** transformaciones básicas se multiplican las matrices de transformación