Paula Díaz Jorge, Franco Alla, Javier Gómez Alayón

SEMINARIO

Mundos virtuales: Introducción a la programación de gráficos 3D

Índice

1. **Qué funciones se pueden usar en los scripts de Unity para llevar a cabo traslaciones, rotaciones y escalados.**
    - Transform.**translate** → Para realizar translaciones
    - Transform.**rotate** → Para realizar rotaciones
    - Transform.**localScale** → Para escalar (Cambiar el tamaño del objeto)
2. **¿Cómo trasladarías la cámara 2 metros en cada uno de los ejes y luego la rotas 30º alrededor del eje Y?. Rota la cámara alrededor del eje Y 30ª y desplázate 2 metros en cada uno de los ejes. ¿Obtendrías el mismo resultado en ambos casos?. Justifica el resultado.**
    a. Primero trasladar y luego rotar

```CSharp
// Trasladar 2 metros en cada eje

transform.Translate(new Vector3(2, 2, 2), Space.Self);

// Rotar 30° alrededor del eje Y

transform.Rotate(0, 30, 0);
```

El resultado de este código es que la cámara primero se mueve a la posición (2,2,2) y luego se rota los 30 grados, esto no afecta en hacia dónde se mueve ya que ya ha realizado el desplazamiento.

    b. Rotar primero y luego trasladar

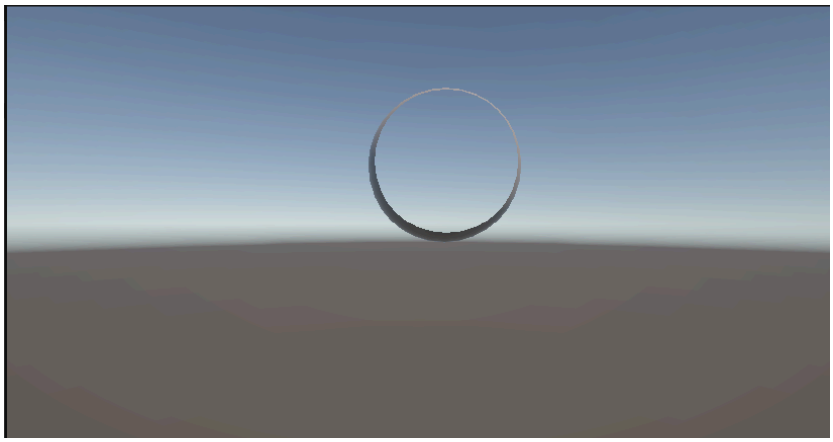
```CSharp
// Rotar 30° alrededor del eje Y

transform.Rotate(0, 30, 0);

// Trasladar 2 metros en cada eje local

transform.Translate(new Vector3(2, 2, 2), Space.Self);
```

El resultado para este caso es diferente al anterior, ya que si al principio rotamos el objeto 30 grados, este cambia su eje de referencia, por lo que el desplazamiento se realiza en base a su nueva orientación.

3. **Sitúa la esfera de radio 1 en el campo de visión de la cámara y configura un volumen de vista que la recorte parcialmente.**

   Para este ejercicio hemos situado la cámara en el origen (0, 0, 0) y la esfera a 5 de distancia (0, 0, 5. Se ajusta el plano cercano (`nearClipPlane`) de la cámara de forma que corte el volumen de la esfera y para que se viese la esfera como en la imagen, el volumen de vista se tuvo que poner de la siguiente manera:

```CSharp
camara.transform.rotation = Quaternion.identity;

camara.nearClipPlane = 4.5f; // Recorta parte frontal de la esfera

camara.farClipPlane = 100f; // Mantiene el plano alejado

camara.fieldOfView = 60f; // Ajusta el campo de visión
```

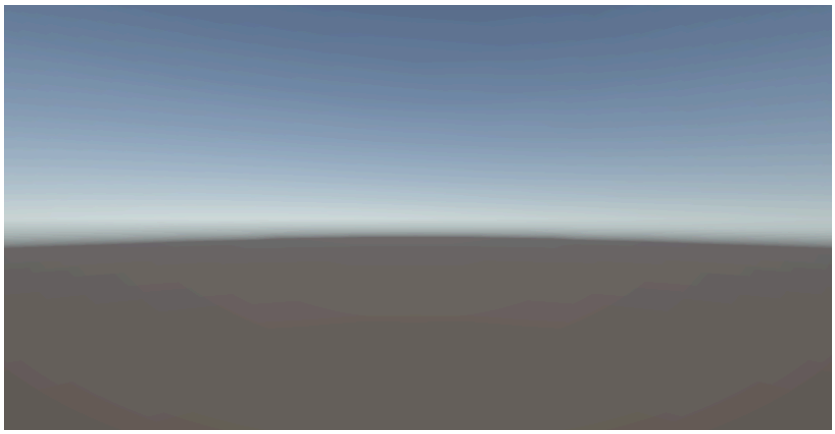


4. **Sitúa la esfera de radio 1 en el campo de visión de la cámara y configura el volumen de vista para que la deje fuera de la vista.**

   En este caso, la cámara se encuentra en el origen (0, 0, 0) mirando hacia el eje Z positivo, y la esfera está situada en (0, 0, 0.5) con radio 1. Al establecer:

```CSharp
camara.transform.rotation = Quaternion.identity;

      camara.nearClipPlane = 1.0f;  // Empieza a renderizar desde
      z = 1

     camara.farClipPlane = 100f;

     camara.fieldOfView = 60f;
```

   El plano cercano de la cámara comienza en z = 1, por lo que toda la esfera queda antes de ese plano y, en consecuencia, fuera del volumen de vista, no siendo visible en la escena.



5. **Cómo puedes aumentar el ángulo de la cámara. Qué efecto tiene disminuir el ángulo de la cámara.**

   El ángulo de la cámara se puede modificar con fieldOfView de la siguiente manera:

```CSharp
Camera.main.fieldOfView = 90f;
```

Con esto lo que se está haciendo es aumentar el ángulo de visión de la cámara.

6. **Es correcta la siguiente afirmación: Para realizar la proyección al espacio 2D, en el inspector de la cámara, cambiaremos el valor de projection, asignándole el valor de orthographic**

   Si, ya que en Unity para realizar una proyección al espacio 2D se tiene que cambiar el tipo de proyección de la cámara a Orthographic, esto se hace para eliminar la perspectiva, lo que es lo mismo, hacer que los objetos no se deformen con la distancia.

7. **Especifica las rotaciones que se han indicado en los ejercicios previos con la utilidad quaternion.**

   Usando cuaterniones, las rotaciones que se han indicado en los ejercicios anteriores serían:

```CSharp
// Rotar 30° alrededor del eje Y
transform.rotation = Quaternion.Euler(0, 30, 0);
```

8. **¿Como puedes averiguar la matriz de proyección en perspectiva que se ha usado para proyectar la escena al último frame renderizado?**

En Unity, la matriz de proyección en perspectiva se obtiene directamente desde la cámara activa.

```CSharp
Matrix4x4 projectionMatrix = Camera.main.projectionMatrix;
```

9. ¿Cómo puedes averiguar la matriz de proyección en perspectiva ortográfica que se ha usado para proyectar la escena al último frame renderizado?

Si la cámara está en modo orthographic, la obtienes de la misma forma:

```CSharp
Matrix4x4 projectionMatrix = Camera.main.projectionMatrix;
```

En este caso, la matriz no aplica perspectiva, es decir, los objetos no disminuyen de tamaño con la distancia.

Se puede comprobar si la cámara es ortográfica de la siguiente manera:

```CSharp
if (Camera.main.orthographic)

       ...
```

10. ¿Cómo puedes obtener la matriz de transformación entre el sistema de coordenadas local y el mundial?

Del sistema de coordenadas local al mundial se hace con la propiedad `localToWorldMatrix` de cualquier objeto `Transform`:

```CSharp
Matrix4x4 localToWorld = transform.localToWorldMatrix;
```

De mundial a local es de la siguiente manera:

```CSharp
Matrix4x4 worldToLocal = transform.worldToLocalMatrix;
```

11. Cómo puedes obtener la matriz para cambiar al sistema de referencia de vista

La matriz de vista transforma las coordenadas del sistema mundial al sistema de referencia de la cámara. Puede obtenerse de dos maneras equivalentes:

```CSharp
Matrix4x4 viewMatrix = Camera.main.worldToCameraMatrix;

Matrix4x4 viewMatrix = Camera.main.transform.worldToLocalMatrix;
```

12. Especifica la matriz de la proyección usado en un instante de la ejecución del ejercicio 1 de la práctica 1.

Con los valores de la cámara:

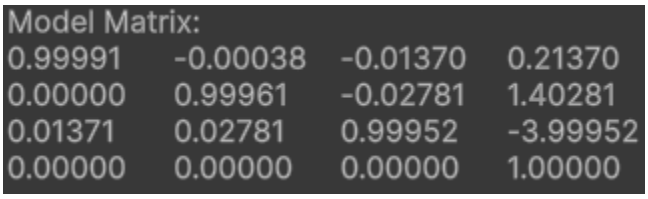
- fieldOfView = 60°
- aspect = 16/9
- nearClipPlane = 0.3f
- farClipPlane = 1000f

La matriz de proyección perspectiva es:

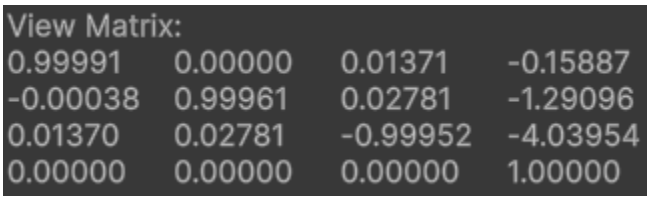
```CSharp
[1.03, 0.00, 0.00, 0.00]

[0.00, 1.83, 0.00, 0.00]

[0.00, 0.00, -1.00, -0.60]

[0.00, 0.00, -1.00, 0.00]
```

13. Especifica la matriz de modelo y vista de la escena del ejercicio 1 de la práctica 1.

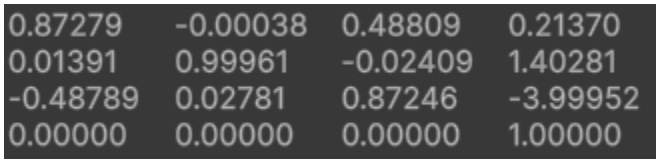
a. Matriz de modelo:



b. Matriz de vista:



14. Aplica una rotación en el start de uno de los objetos de la escena y muestra la matriz de cambio al sistema de referencias mundial.



15. ¿Cómo puedes calcular las coordenadas del sistema de referencia de un objeto con las siguientes propiedades del Transform:?:

Position (3, 1, 1), Rotation (45, 0, 45)

Puedes utilizar la funcionalidad que te dá Unity para crear una matriz en base a una posición, una rotación y una escala de la siguiente manera:

```CSharp
Matrix4x4 modelMatrix = Matrix4x4.TRS(
    new Vector3(3, 1, 1),
    Quaternion.Euler(45, 0, 45),
    Vector3.one
);
Debug.Log(modelMatrix);
```

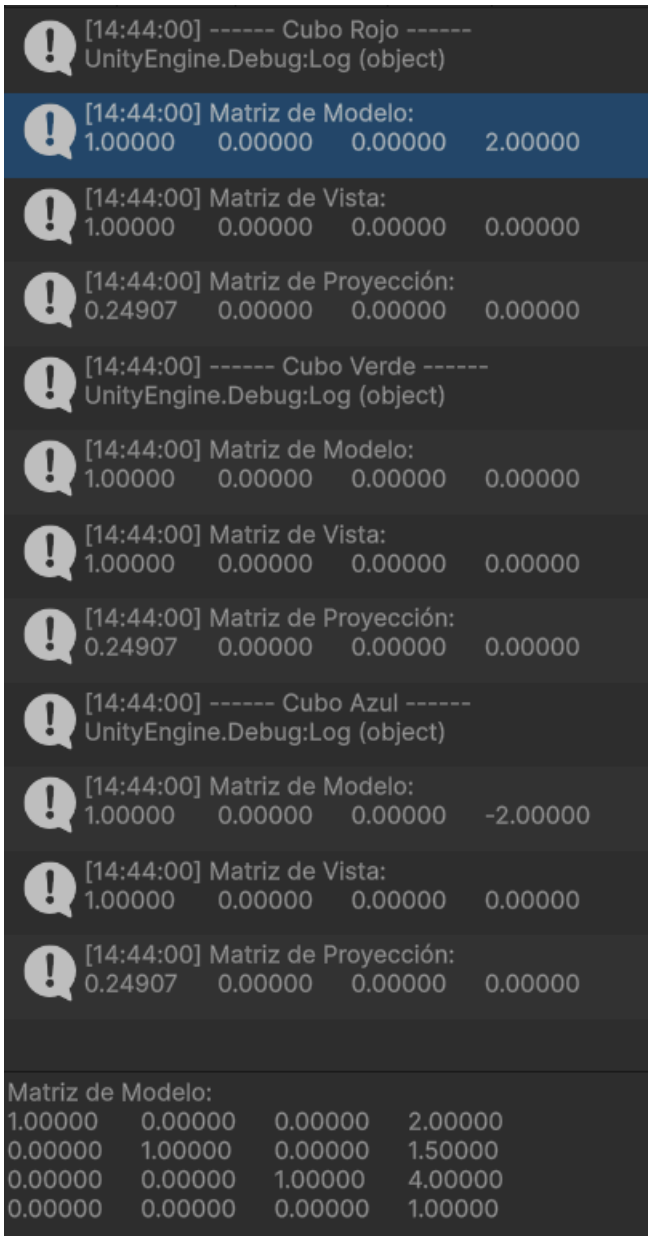
16. **Crea una escena en Unity con los siguientes elementos: cámara principal, plano base (como suelo) y tres cubos de distinto color (rojo, verde, azul) colocados en posiciones distintas en el espacio. Realiza un pequeño script de depuración adjunto a la cámara que permita visualizar en consola o en pantalla las matrices de transformación (Model, View, Projection) y sus resultados sobre un vértice de cada cubo.**

Para responder esta pregunta en vez de poner 9 imágenes voy a poner el código que he utilizado para obtener las respuestas y además una imagen de todos los logs.

CSharp

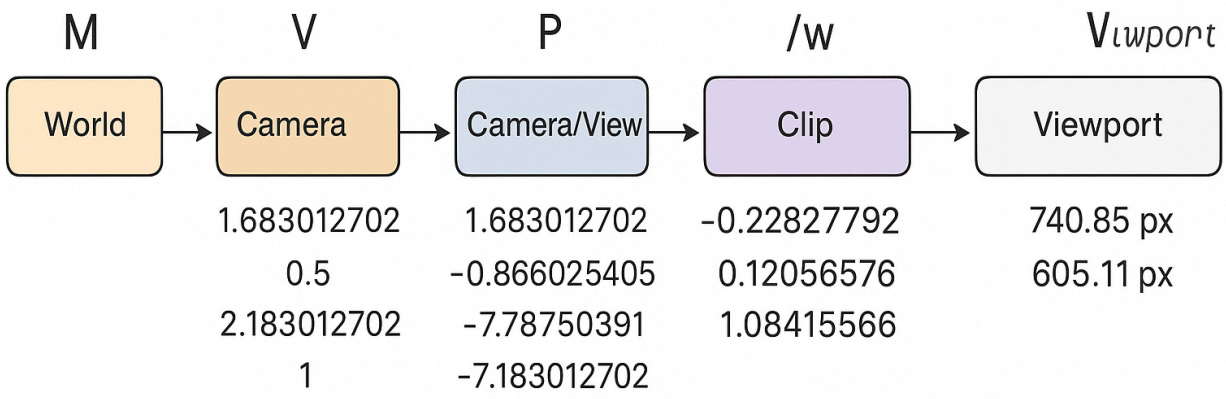
```
// Transforms de los cubos en la escena
    public Transform cuboRojo;
    public Transform cuboVerde;
    public Transform cuboAzul;
    // Vértice local a transformar (esquina del cubo)
    public Vector3 verticeLocal = new Vector3(0.5f, 0.5f, 0.5f);
    void Start()
    {
        ImprimirMatrices(cuboRojo, "Cubo Rojo");
        ImprimirMatrices(cuboVerde, "Cubo Verde");
        ImprimirMatrices(cuboAzul, "Cubo Azul");
    }
    void ImprimirMatrices(Transform objeto, string nombre)
```

```
    {
        Vector3 posicionMundialVertice = objeto.position + verticeLocal; //
Cojo el vértice del cubo.
        Matrix4x4 matrizModelo = objeto.localToWorldMatrix;
        Matrix4x4 matrizVista = Camera.main.worldToCameraMatrix;
        Matrix4x4 matrizProyeccion = Camera.main.projectionMatrix;
        Debug.Log($"------ {nombre} ------");
        Debug.Log("Matriz de Modelo:\n" + matrizModelo);
        Debug.Log("Matriz de Vista:\n" + matrizVista);
        Debug.Log("Matriz de Proyección:\n" + matrizProyeccion);
    }
```



17. **Dibujar en un programa de dibujo el recorrido de las coordenadas de un vértice específico del cubo rojo:**

    Local → World → Camera/View → Clip → NDC → Viewport. Indicar cómo cambia su valor en cada espacio. Aplicar la transformación manualmente a un punto (por ejemplo, el vértice (0.5, 0.5, 0.5)) y registrar los resultados paso a paso.



El vértice local (0.5, 0.5, 0.5, 1) se transforma por el modelo M (aquí asumido identidad) manteniéndose como World = (0.5, 0.5, 0.5, 1); la matriz de vista lleva ese punto al espacio cámara como Eye = (1.683012702, 0.5, 2.183012702, 1); la matriz de proyección produce el vector de clip Clip = (1.683012702, −0.866025405, −7.78750391, −7.183012702), cuyo componente w (w_clip = −7.183012702) controla la perspectiva; al dividir por w se obtiene NDC = (x_ndc, y_ndc, z_ndc) = (−0.22827792, 1.02056576, 1.08415566), valores fuera del rango típico [−1,1] en y y z que indican que el vértice queda fuera del frustum; finalmente, mapeando NDC a pantalla con width = 740.85 px y height = 605.11 px resulta x_px ≈ 285.9 px y y_px ≈ −6.2 px (origen superior), por lo que la proyección coloca el punto fuera del área visible; la causa principal de los grandes cambios numéricos es la multiplicación por la matriz de proyección seguida de la división no lineal por w, que amplifica o invierte componentes según la profundidad y las convenciones de cámara.

18. Mover o rotar uno de los cubos y mostrar cómo cambian los valores de su matriz de modelo. Rotar la cámara y mostrar cómo se modifica la matriz de vista. Cambiar entre proyección ortográfica y perspectiva y comparar las diferencias numéricas en la matriz de proyección.

Vamos a dividir la respuesta en tres:

a. Mover o rotar un cubo:
   i. Normal

```
Matriz de Modelo:
1.00000   0.00000   0.00000   0.00000
0.00000   1.00000   0.00000   1.50000
0.00000   0.00000   1.00000   4.00000
0.00000   0.00000   0.00000   1.00000
```

   ii. Movido

```
Matriz de modelo tras trasladar el cubo:
1.00000   0.00000   0.00000   1.00000
0.00000   1.00000   0.00000   1.50000
0.00000   0.00000   1.00000   4.00000
0.00000   0.00000   0.00000   1.00000
```

   iii. Rotado

```
Matriz de modelo tras rotar el cubo:
0.70711   0.00000   0.70711   0.00000
0.00000   1.00000   0.00000   1.50000
-0.70711  0.00000   0.70711   4.00000
0.00000   0.00000   0.00000   1.00000
```

b. Rotación de la cámara
   i. Normal

```
Matriz de vista inicial:
1.00000   0.00000   0.00000   0.00000
0.00000   1.00000   0.00000   -1.00000
0.00000   0.00000   -1.00000  -10.00000
0.00000   0.00000   0.00000   1.00000
```

ii. Rotada

```
Matriz de vista tras rotar la cámara:
0.70711   0.00000   -0.70711  -7.07107
0.00000   1.00000   0.00000   -1.00000
-0.70711  0.00000   -0.70711  -7.07107
0.00000   0.00000   0.00000   1.00000
```

c. Cambio de perspectiva de la cámara

i. Normal

```
Matriz de proyección actual:
0.49931   0.00000   0.00000   0.00000
0.00000   1.73205   0.00000   0.00000
0.00000   0.00000   -1.00060  -0.60018
0.00000   0.00000   -1.00000  0.00000
```

ii. Ortográfica

```
Matriz de proyección actual:
0.05766   0.00000   0.00000   0.00000
0.00000   0.20000   0.00000   0.00000
0.00000   0.00000   -0.00200  -1.00060
0.00000   0.00000   0.00000   1.00000
```