

Informática Gráfica II

Práctica 2

Curso 24/25

Desarrollo de un videojuego utilizando Ogre3D (Parte II)

En esta práctica vamos a mejorar el videojuego realizado en la práctica anterior con los efectos vistos en clase, tales como el sistema de partículas, *multitexturas*, *skyplanes* y *shaders*.

Apartado 1. Animación – Creando una *intro* para el juego.

En este apartado vamos a crear una *intro* para el juego. La idea es que, al iniciar el programa, se muestre la animación creada – en bucle – hasta que pulsemos una tecla, por ejemplo, la s, tras lo cual dará comienzo el juego.

Para crear la animación vamos a contar con los siguientes elementos (aunque en apartados posteriores introduciremos algún cambio):

- Un suelo (distinto del suelo empleado en el laberinto).
- Sinbad (con y sin espadas)
- Ogrehead

Sinbad hará movimientos laterales, realizando rotaciones cuando llegue a los extremos, y un baile cuando esté en el centro. Tened en cuenta que, hasta el primer giro, Sinbad no tiene espadas, aparecen al girar la primera vez. Ogrehead hará un movimiento similar, y en su último giro se aplicará una escala de forma que cada vez sea más pequeño, hasta desaparecer cuando llega a la posición de Sinbad. Seguidamente se muestran algunas capturas de la intro. Tenéis disponible un video en el C.V. para ver con más detalle cada una de las acciones.





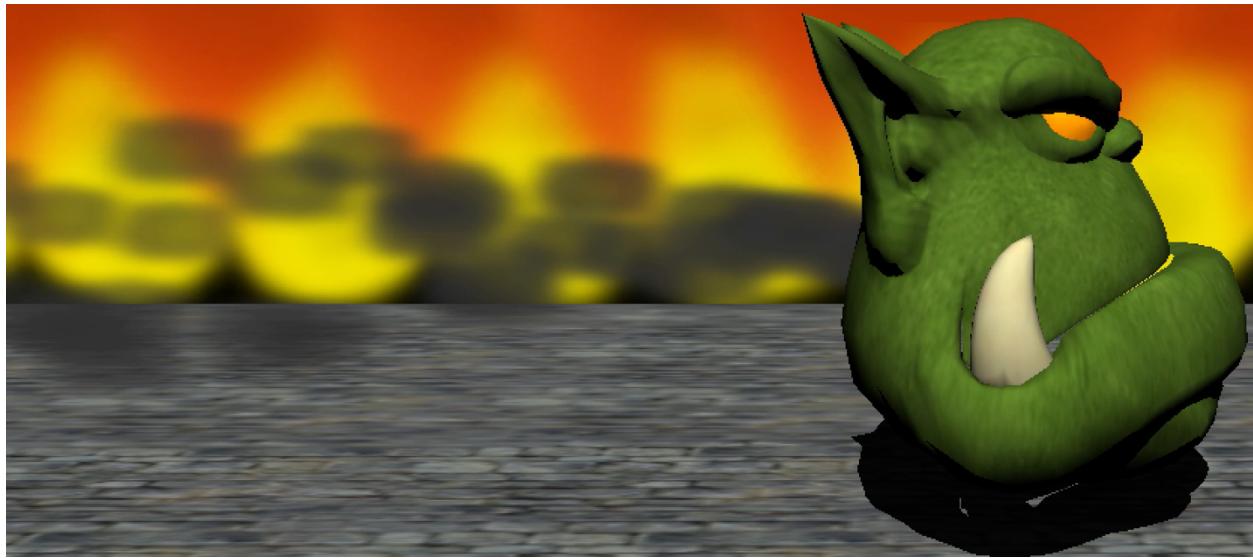
Apartado 2. Sistema de partículas.

Vamos a crear varios efectos utilizando sistemas de partículas. Durante la intro vamos a incluir dos: efecto fuego y una estela de humo para ogrehead. Durante el juego, simularemos humo en el laberinto para dificultar la visibilidad.

Inicialmente, creamos los efectos en la intro. Primero, la estela de ogrehead. Para ello, definiremos primero el material del humo – disponible en el Tema 7 – y seguidamente definiremos el sistema de partículas en un fichero con extensión **.particle**. Tened en cuenta que el efecto debe ser similar. Fijaos en:

- La duración (no acaba nunca)
- El tamaño de cada partícula.
- La dirección de la emisión y su ángulo.
- El color de las partículas.

La siguiente imagen muestra un ejemplo de la estela de humo de ogrehead:



Para el efecto del fuego, crearemos varios sistemas de partículas equidistantes entre sí. Para cada uno, utilizaremos el efecto que vimos en clase utilizando el modificador (affector) **ColourImage** con la imagen **smokecolors.png**. Tened en cuenta:

- La duración (termina un poco antes de que finalice la animación)
- Se repite en bucle.
- La intensidad y ángulo de las partículas.

La siguiente imagen muestra un ejemplo de este efecto.



Finalmente, crearemos el efecto humo en el laberinto. Con ello pretendemos simular un efecto que dificulte la visibilidad en el juego. Podemos utilizar, como base, el ejemplo visto en clase. En el fichero del laberinto, utilizaremos la letra 's' para indicar que en esa posición colocaremos un sistema de partículas de humo.

Tened en cuenta que también irá una perla en esta posición. Hay que tener cuidado al añadir este sistema de partículas en el laberinto, ya que, si lo añadimos al nodo que contiene el bloque ‘perla’, es posible que la *bounding box* de la misma se modifique, haciendo inconsistente la colisión con el personaje para calcular cuándo Sinbad se come una perla.

La siguiente imagen muestra un ejemplo de un mapa configurado con el efecto humo y su visualización:



Tenéis disponible un vídeo en el C.V. para que veáis con más detalle los efectos de la intro.

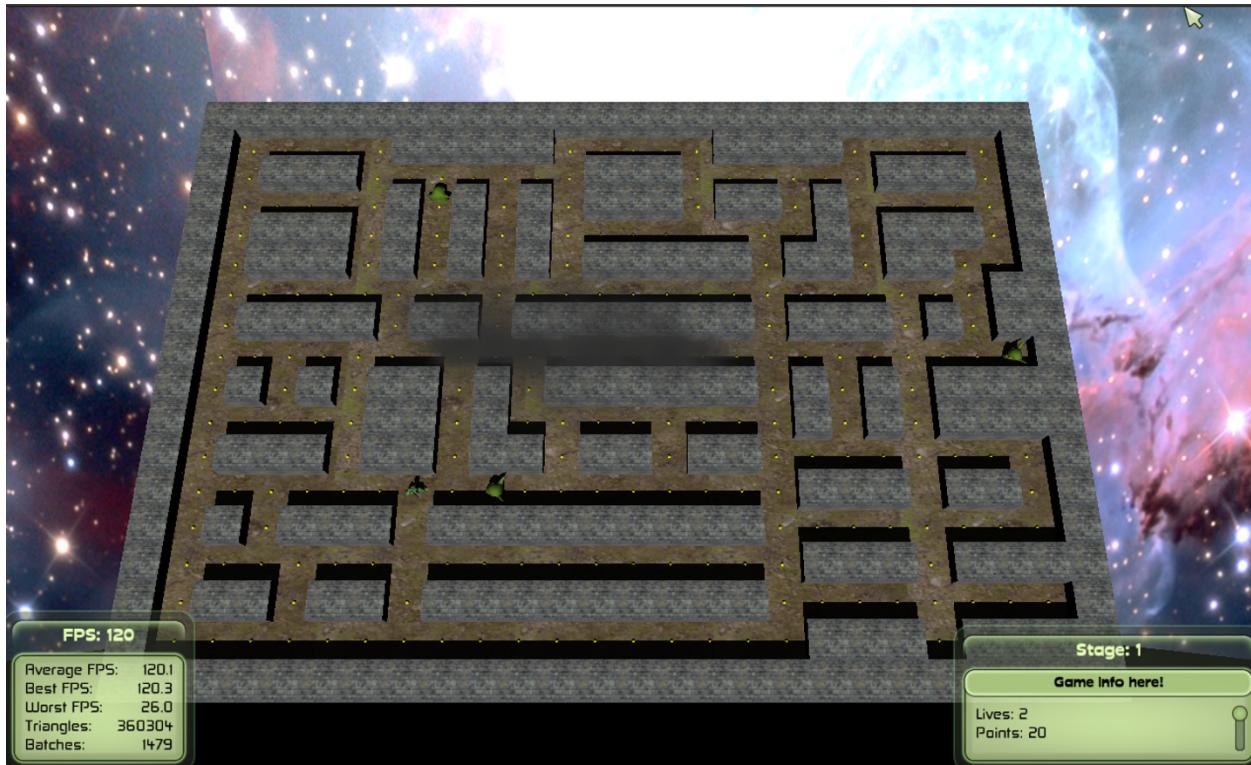
Apartado 3. Efectos con texturas y añadiendo un cielo

En este apartado vamos a aplicar efectos sobre las texturas del juego. Al menos, es necesario aplicarlo sobre un elemento con textura, como por ejemplo, el suelo de la intro. También podéis aplicar otros de los efectos que hemos visto en clase sobre los bloques, o el suelo del laberinto.

El efecto aplicado deberá utilizar dos ficheros de imagen. En la siguiente captura se muestra un efecto utilizando la textura del suelo y `lightMap.jpg` para dar un efecto de sombra.



Seguidamente, crearemos un cielo como se ve en la siguiente imagen. Tened en cuenta que el cielo NO se ve durante la intro, debe aparecer sólo durante el juego. Dependiendo de cómo sea la orientación de vuestro laberinto, tendréis que crear el plano – con la normal correspondiente – y probar con los distintos parámetros. Además, el cielo debe moverse.



Apartado 4. Shaders

En este apartado vamos a definir tres shaders.

4.1 Shader para el cielo con animación de zoom y luz

Implementa un shader para el cielo, de forma que se aplique un factor de zoom entre [0.3 – 1]. Además, se debe aplicar un efecto sobre el cielo para que se oscurezca a medida que se aplica el zoom. Esto puede conseguirse pasando el valor de una variable asignada desde la aplicación – similar al del zoom – para que también aplique un factor entre [0.3 – 1] al color final. Así, cuando el factor aplicado sea 0.3, el cielo deberá verse más oscuro, y cuando sea 1, más claro. La idea es que – tal y como se muestra en el vídeo disponible en el C.V. – la animación pase por los valores intermedios, dando una sensación de sombreado en el tiempo.

Define el material `practica2/spaceSkyZoomLightShader` en el fichero de materiales. Los shaders de vértices y fragmentos estarán en los ficheros, respectivamente, `SpaceSkyZoomLightVS.gls1` y `SpaceSkyZoomLightFS.gls1`.

4.2 Shader para hacer hueca una esfera

Para este shader vamos a crear un efecto aplicado a una esfera similar al efecto que provoca el óxido en los metales, de forma que las partes con corrosión, nos permitan ver el interior de la esfera. Para ello, primero crearemos una esfera en la animación desarrollada en el primer apartado de esta práctica. Tenéis disponible un vídeo en el C.V. para que podáis analizar la posición y escala – aproximada – de la misma.

El material en cuestión – al que llamaremos `practica2/SphereHoles` – utilizará tres unidades de textura, en este orden: `corrosion.jpg`, `BumpyMetal.jpg`, y `Material_dirt.jpg`. La idea es dar un efecto de corrosión, y que a la parte exterior de la esfera se aplique la textura `BumpyMetal.jpg`, mientras que a la parte interior de la esfera aplicaremos la textura `Material_dirt.jpg`. Podéis utilizar otros ficheros con imágenes distintas, pero el efecto conseguido debe ser el mismo que se describe en este apartado.

Hay que tener en cuenta varios aspectos. Inicialmente, el *culling*, ya que nos vamos a encargar en los shaders de calcular qué color final tendrá cada cara (*front* y *back*). Recordad que para esto es necesario indicar `cull.hardware none` y `cull.software none` en el material. Además, debemos saber qué cara del fragmento estamos procesando. Para ello, pasaremos desde la aplicación el siguiente parámetro: `param_named_auto flipping render_target_flipping`.

El shader de vértices y fragmentos se codificarán en los ficheros `SphereHolesVS.gls1` y `SphereHolesFS.gls1`, respectivamente.

La siguiente imagen muestra el efecto conseguido:



4.3 Efecto de onda en VS y en FS para el suelo

Para el último shader vamos a simular el efecto del movimiento del mar. Este efecto lo crearemos definiendo un nuevo material – llamado **practica2/wavesShader** – que aplicaremos al suelo de la presentación, tal y como se ve en la siguiente imagen:



En este caso, el shader de fragmentos (**wavesShaderFS.gls1**) es sencillo, basta con obtener el color a partir de las coordenadas de textura pasadas por el shader de vértices y la unidad de textura – que utilizará el fichero **Water02.jpg** – pasada a través de la aplicación. No se le aplica ninguna modificación al color obtenido.

Al shader de vértices (**wavesShaderVS.gls1**) le pasamos desde la aplicación: la matriz de proyección (**worldviewproj_matrix**), un valor que varíe en el tiempo en el intervalo [0, 1], y el centro del plano donde se va a aplicar el shader.

En esencia, el efecto se consigue variando la componente **y** del vértice. Existen múltiples soluciones para conseguir el efecto deseado. Una aproximación es la siguiente:

```
vertexCoord.y += sin(vertexCoord.x + (time*a)) * b + sin(vertexCoord.z + (distanceCenter) + (time*a)) * b;
```

donde **vertexCoord** es una variable auxiliar con las coordenadas de **vertex**, **time** es la variable de tipo **uniform** que le pasamos desde la aplicación y se modifica de forma automática, y **distanceCenter** es la distancia desde el vértice actual al centro de la malla donde aplicaremos el efecto. Los valores de **a** y **b** puedes ajustarlos para conseguir un resultado similar al mostrado en el vídeo que está disponible en el C.V.

Fecha de entrega.

La práctica deberá entregarse, a través del C.V., antes del día **4 de diciembre de 2024 a las 11:00 horas**. Únicamente será necesario realizar una entrega por grupo. La defensa de la práctica se realizará **en la clase de laboratorio del día 4 de diciembre**. Para realizar la defensa, **los dos miembros del grupo** (si se ha realizado en pareja) deben asistir presencialmente a clase. Además, se utilizará el orden de entrega para realizar la defensa, de forma que el primer grupo en entregar la práctica será el primero en realizar la defensa.