Álvaro Holguera
CIFP José Luis Garci
(Madrid)

UD 07
MATERIALES LUCES & SONIDO

Desarrollo de Entornos Interactivos Multidispositivo

Álvaro Holguera
CIFP José Luis Garci
(Madrid)

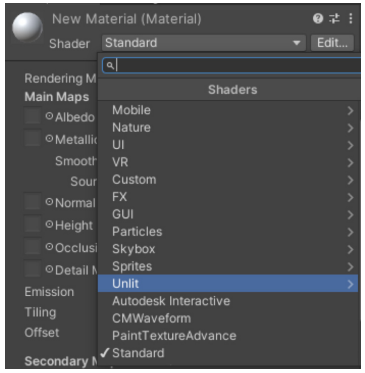
Parte 1
MATERIALES

Ha llegado el momento de dotar de realismo a nuestros juego, de “ponerlos bonitos”. Para ello, usaremos los materiales, la iluminación y el sonido.

SHADER Y MATERIALES

La combinación de materiales y luces es lo que determina el look realista de nuestra escena. Pero para dar esa apariencia "física" a un objeto, tenemos que distinguir entre un "shader" y un "material"

Shaders



Un shader es una porción de código que dice a la superficie cómo debe comportarse cuando la luz incide sobre ella. Aunque no lo vemos, se encuentra en cada material que creamos. Lo veremos en la parte superior.

Por defecto se crea el material con un shader de tipo standard, pero hay muchos más, algunos de ellos con funcionalidades muy específicas (como los optimizados para móviles, para crear efectos de partículas, líquidos, crear cajas de cielos ó skybox, etc.)

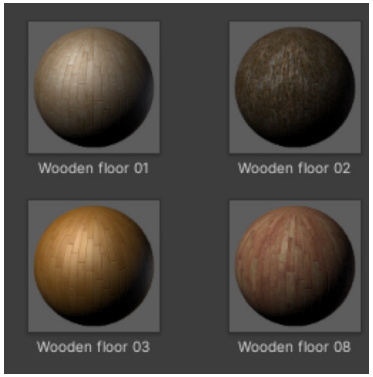
Si queremos aplicar un material a un objeto, pero no queremos que aplique ningún shader, podemos usar el que está en Unlit->Transparent.

NOTA: si trabajamos con URP ó HDRP podemos usar la herramienta "Shader Graph" para crear nuestros propios shaders mediante una herramienta gráfica nodal que veremos más adelante.

Materiales

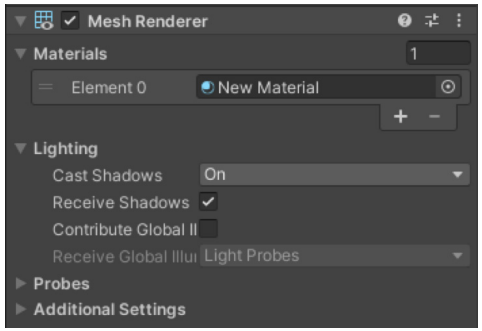
Los materiales reproducen la textura de los objetos 3D, sus propiedades, las cuales serán incluidas en el componente de "mesh renderer" de los GameObjects.

Los materiales debemos crearlos como un objeto más en nuestro proyecto (Botón derecho del ratón->Create->Material). Si tenemos la vista a una columna en el proyecto se mostrarán con un icono en forma de pelota azul, y si la tenemos a dos columnas mostrarán una previsualización de su aspecto.



Podemos arrastrarlo al objeto de la ventana de Jerarquía, y se le asignará al componente de Mesh Renderer, o podemos arrastrarlo directamente a ese componente.

El componente Mesh Renderer es además el que nos permite indicar si el objeto proyecta sombras.



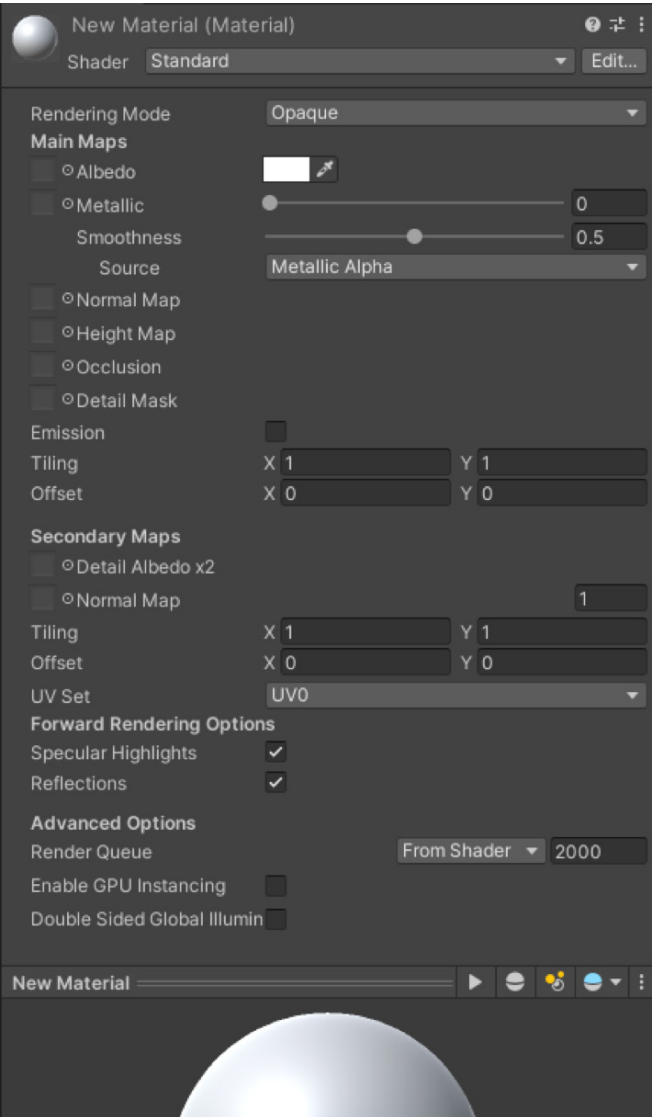
NOTA: Un mesh renderer puede albergar más de un material si el modelo tiene varias mallas.

Las opciones que aparecen en la ventana de inspector al seleccionar el material, dependerá del tipo de shader que elijamos.

El Standard Shader, por ejemplo, está pensado como material habitual para objetos sólidos.

STANDARD MATERIAL

Aunque trabajando en URP y HDRP (como veremos más adelante) tendremos acceso a parámetros y shaders más complejos, veamos las opciones del material que usa Unity por defecto.



Rendering mode

Permite crear objetos opacos con diferente material:

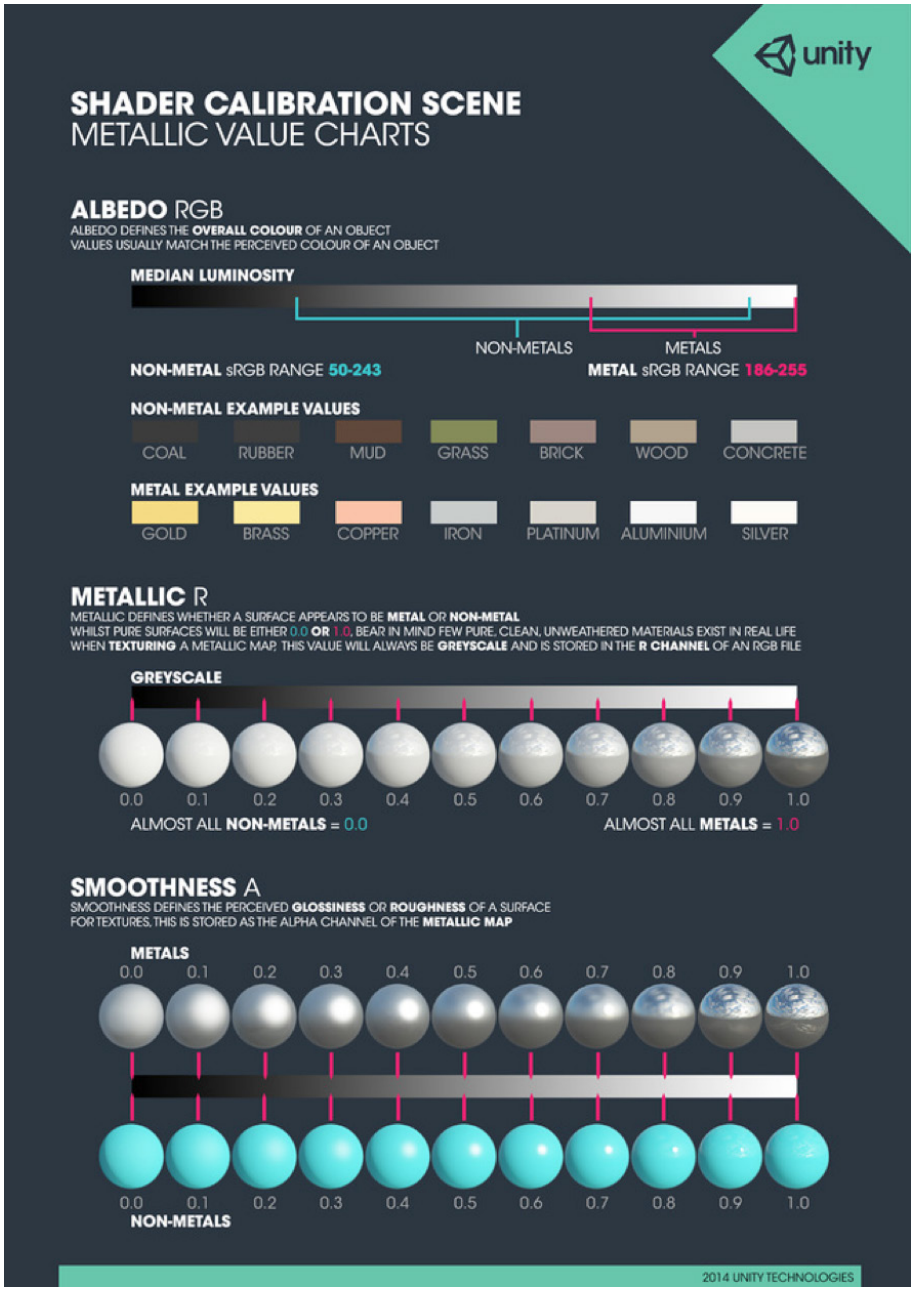
- **Opacos**, el más habitual .
- **Transparentes** (como el cristal).
- **Desvanecidos** (fade), similar a transparente, pero sin mantener propiedades del material.
- **Cutout**, que permite ajustar los bordes del canal alpha de la textura.

Más información en docs.unity3d.com:

Main Maps

Definen las texturas y su brillo. Algunos de estos parámetros aceptan una imagen en lugar de un color.

- **Albedo**: color ó textura por defecto del material. Si añadimos un color, tintará el mapa de ese color.
- **Metallic / Specular:** cuanto más metálico / espécular es un objeto, más refleja las luces de la escena). Dependiendo del shader elegido aparecerá uno y otro.
- **Smoothness**: cómo de pulida está la superficie, y por tanto cómo de direccionales son las reflexiones. Si aplicamos un mapa al parámetro de Metallic, desaparecerá la opción de "Smoothness", ya que este lo controlará el Alpha Channel.
- **Normal Map**: mapa de normales que simula detalles en la malla, dando la apariencia de tener una alta poligonación. En la textura importada indicaremos que es mapa de normale.
- **Height Map**: produce elevaciones en le material. Requiere altas poligonaciones en los objetos.
- **Occlusion**: indica qué zonas de nuestro material deben recibir o no luz indirecta de la escena. Produce efectos poco realistas pero ayudan a crear ambientes.
- **Detail Mask**: máscara aplicada a los mapas de detalle.
- **Emission**: permite simular que partes de nuestro material son emisoras de luz. Si lo activamos, podremos indicar cómo afecta a la iluminación global de la escena. En URP y HDRP no funciona.
- **Tilling / Offset:** cómo se repite el mapa en el material, así como si se desplaza (offset)
- **Secundary maps:** Permite añadir un mapa de Albedo y otro de normales de alta definición, para cuando la cámara se acerque mucho al objeto.



Puedes usar las cartas de referencia de Unity para crear tu material estándar.

La asset-store de Unity tiene un gran nº de materiales disponibles

Animar un material

Aunque en el siguiente tema conoceremos la potente herramienta de Shader Graph, es bueno saber que a través de la clase "Renderer" tenemos acceso al Mesh Renderer de nuestro Game Object, y eso implica los materiales que usa y sus parámetros.

Podemos acceder y/o cambiar cualquier de sus atributos: color, textura, offset, shader, etc.Por ejemplo, a menudo es útil contar con una textura animada, lo cual nos permite, entre otras cosas:

- Evitar animar un objeto, si solo queremos crear una sensación de desplazamiento
- Crear efectos 2D sin recurrir a los sistemas de partículas, solo animando una serie de imágenes con fondo transparente

Se puede animar una textura de dos formas:

- Creando un componente Animation, en el que cambiamos los parámetros de la textura del Game Object, como haríamos con cualquier otra animación
- Mediante código, accediendo a sus propiedades mediante el componente Renderer, el cual nos permite acceder a los atributos del material

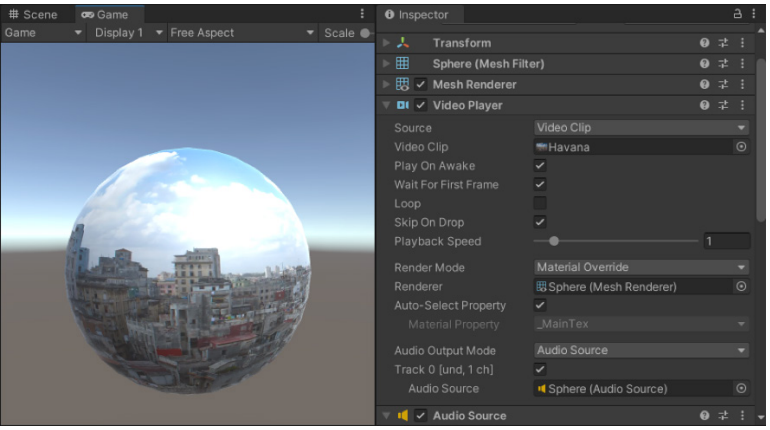
Este script hace que sus texturas de Albedo y de Normales se deplsacen creando la sensación de movimiento:

```
[SerializeField] float scrollSpeed;
Renderer rend;
void Start()
{
    rend = GetComponent<Renderer>();
    scrollSpeed = 0.43f; //Velocidad de desplazamiento
}
void Update()
{
    //Distancia de desplazamiento, según el tiempo transc.
    float offset = Time.time * scrollSpeed;
    //Vector de desplazamiento
    Vector2 despl = new Vector2(0, -offset);
    //Desplazamos la textura albedo y la normal
    rend.material.SetTextureOffset("_MainTex", despl);
    rend.material.SetTextureOffset("_BumpMap", despl);
}
```

PARA NOTA: podemos también crear texturas animadas mediante código para lograr efectos similares a los sistemas de partículas. En este vídeo puedes ver cómo conseguirlo.

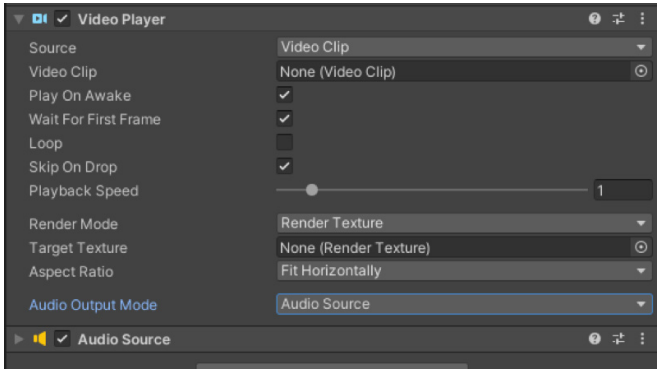
INTEGRAR VÍDEO

En Unity podemos incrustar vídeos mediante el componente "Video Player" (en versiones anteriores se usaba una textura para reproducir el vídeo):



Podemos crear un Empty Object que contendrá el vídeo, aunque también podemos usar otro objeto al que el vídeo envolverá como una textura.

Añadimos el componente Video>Video Player. Entre sus opciones tenemos:

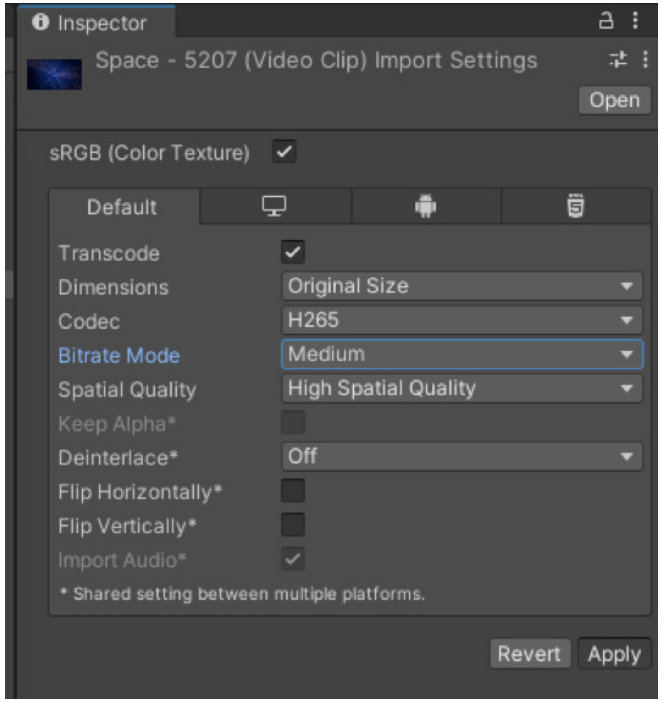


- **Fuente** (si es un clip o una URL con el vídeo en streaming) y el archivo
- Que se reproduzca **automáticamente** (Play on Awake) así como que espere a que esté cargado para reproducirse,
- Reproducción en **bucle** o **velocidad** de reproducción
- **Render Mode**:
  - Puede ser una "Render Texture", lo que permite por ejemplo que el vídeo funcione como una textura rodenado un modelo 3D (hay que indicar la textura de destino)
  - Podemos seleccionar también una cámara, en su "Far Plane" que nos permite poner objetos delante del vídeo, o Near Plane que se reproduce ocupando toda la pantalla. Debemos indicar qué cámara usar, y en la cámara indicada configurar para que el "Rendering Path" sea "Forward".
- Podemos indicar cómo se ajustará el vídeo a la pantalla (Aspect Ratio).

Finalmente debemos seleccionad la salida de audio, que es preferible seleccionar un Audio Source que incorporaremos al objeto como componente adicional.

Otra opción es crear las llamadas "Render Texture" que permiten recoger las imágenes de una cámara de la escena y mostrarlas en la textura, por ejemplo para crear "cámaras de vigilancia" en el juego.

En el inspector del vídeo importado, podemos transcodificar el vídeo para cambiar de codec, calidad, voltearlo, etc.



Videotutoriales en

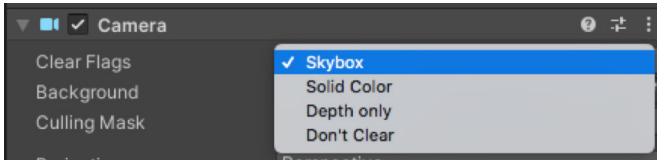
https://learn.unity.com/tutorial/playing-video-in-unity

SKY BOX

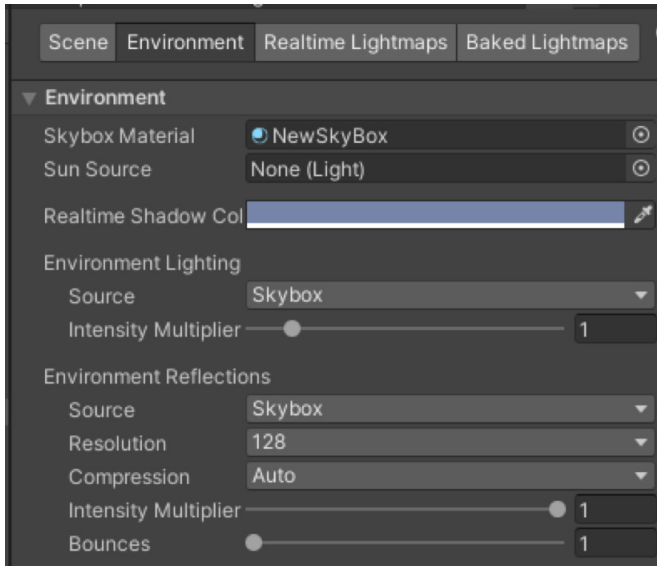
Los materiales de tipo SkyBox permiten crear el entorno en el que se mueven nuestra escena.

Fuente: https://assetstore.unity.com/packages/2d/textures-materials/sky/free-hdr-sky-61217

Para que se muestre, deberemos configurar el parámetro "Clear Flags" de nuestra cámara en "Skybox", que es como viene por defecto:

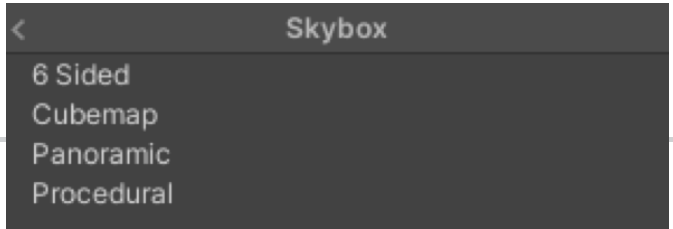


En Window>Rendering>Lighting, en la pestaña de "Environment", podremos indicar qué material de tipo SkyBox usará la cámara, así como activar que la luz ambiental venga de ese material, lo que da más realismo a la escena:



Nosotros podemos crear nuestros propios materiales de tipo SkyBox, pero recuerda: las texturas que usan estos materiales deben ser imágenes de alto rango dinámico (HDR)

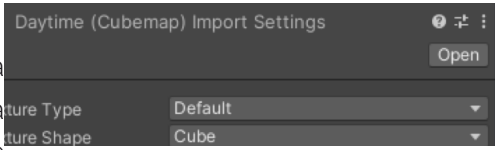
Para crear un nuevo material, en el shader debemos indicar que es Skybox. Tendremos varias opciones, dependiendo de cómo ha sido creada la imagen que servirá a nuestro cielo: cubemap, procedural, 6 sided (requiere 6 imágenes HDR independientes), ó panoramic.



Ejemplo de SkyBox creado con "Cubemap"

Ejemplo de imagen "Panoramic"

La textura que vamos a usar, es decir el archivo HDR, debemos indicar que es "Cube" en el "Texture Shape" si la queremos usar en un skyBox de tipo "cubemap".



CUIDADO: cada archivo HDR pesa mucho y puede hacer que tu proyecto ocupe demasiado en el disco duro. Borra aquellos que no uses.

Una vez creado el material, podremos acceder a ciertos parámetros de nuestro SkyBox, como la exposición, la rotación, virado de color, etc.

TRUCO: a través de la clase "RenderSettings" que nos permite controlar parámetros ambientales de la escena, podemos acceder a los parámetros de nuestro SkyBox y modificarlos, para, por ejemplo, hacer que rote solo en el Update:

```
RenderSettings.skybox.SetFloat ("_Rotation", Time.time * 5f);
```

URP & HDRP

Veamos ahora lo que significa trabajar con Universal Render Pipeline (URP) o High Definition Render Piplein (HDRP)

¿Qué es un “Render Pipeline”?

Los “render pipilenes” hacen referencia a las operaciones que realiza Unity para realizar renderizados y postprocesos a nuestro proyecto.

Debido a que afecta a cómo se comportan los shaders, y que es difícil pasar de uno a otro, tenemos que elegir correctamente el pipeline adecuado según las características de nuestro juego

Cuando hablamos de SRP nos referimos a “scriptable render pipeline”, lo que significa que tendremos total control de nuestras texturas a través de código.

En Unity, además del sistema de trabajar por defecto (el Built-In Render Pipeline, también llamado “Standard”), que tiene limitaciones como veremos, tenemos 2 formas básicas de trabajar con SRP:

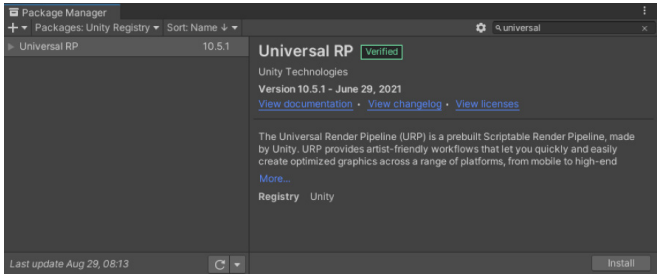
- Universal Render Pipeline: permite crear gráficos optimizados para un amplio rango de plataformas. Usado para juegos 2D y móviles de alta calidad
- High Definition Render Pipeline: permite crear resultados de alta calidad para plataformas de alto rendimiento, como PCs y consolas. Incluye funcionalidades adicionales, así como shaders más complejos y más tipos de luces y efectos atmosféricos.

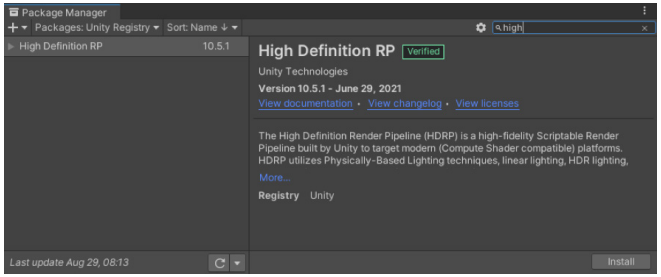
Para decidir cuál usar: https://docs.unity3d.com/2019.3/Documentation/Manual/render-pipelines.html

Una vez instalado los paquetes, podemos configurar un Render Pipeline en “Edit>Project Settings > Quality”

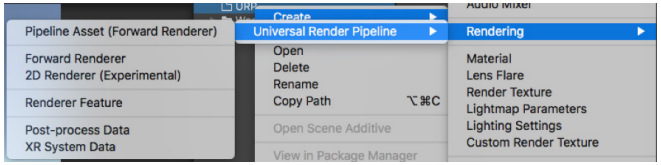
ACTUALIZAR EL RENDER PIPELINE

Si tenemos un proyecto en Built-In Render Pipleine, podemos actualizarlo a través del Package Manager e instalando el paquete correspondiente ("Universal RP" o High Definition RP"). Eso sí, ero entonces hay que actualizar algunas luces y postprocesos.

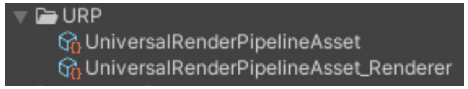




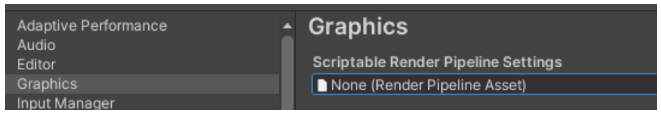
Una vez instalado, debemos crear los assets necesarios en la configuración. Para ello, pulsamos con el botón derecho en el proyecto > Create > Rendering > URP/HDRP > Pipeline Asset.



Creará dos archivos:

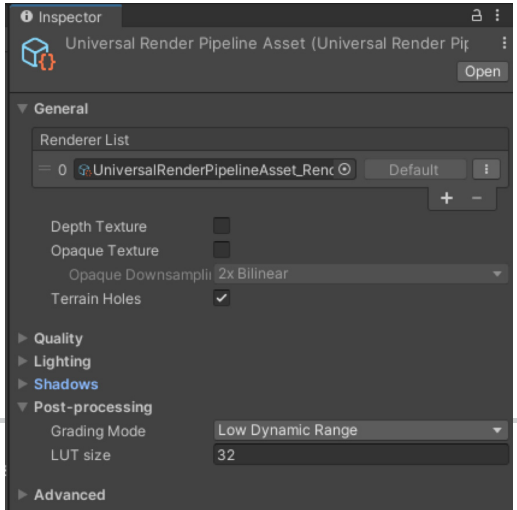


Uno de ellos, el "UniversalRenderPipelineAsset" lo vincularemos en la sección de "Graphics" de nuestro Project Settings:

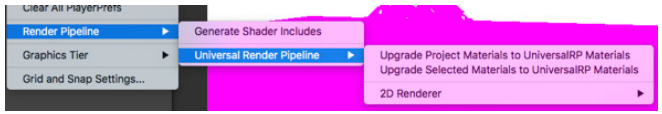


Si hemos creado un proyecto en URP ó HDRP desde Unity, tendrá ya algunos creados y asignados por defecto.

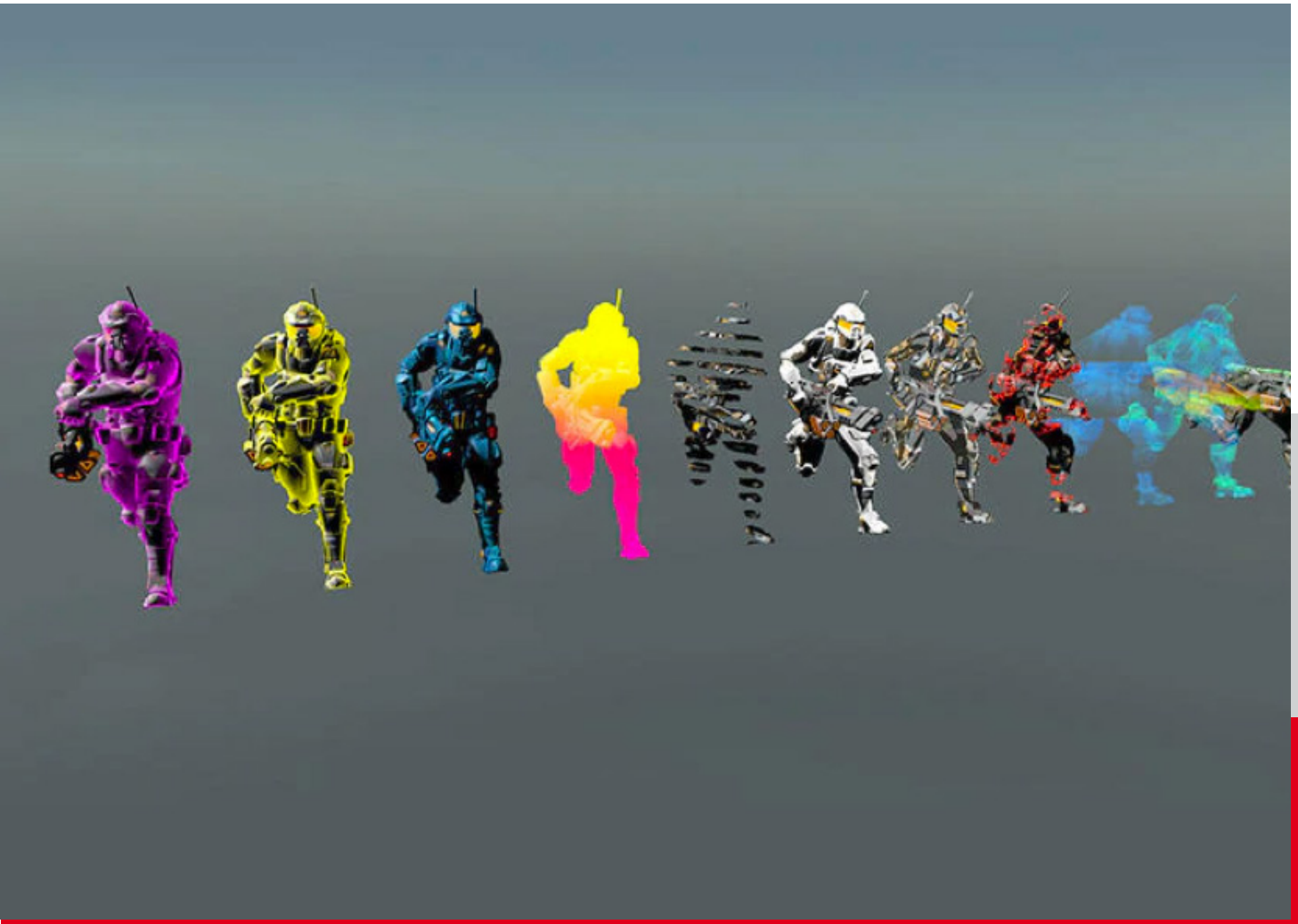
Si miramos ese archivo en el inspector veremos que contiene todas las configuraciones para gestionar la calidad de nuestros gráficos, y por tanto, también el rendimiento:



Si teníamos materiales creados previamente, aparecerán rosa (y es posible que si teníamos terrenos desaparezcan). Por ello es necesario actualizar todos los materiales (o aquellos que seleccionemos): Edit > Render Pipeline > Universal Render Pipeline / HDRP > Update Project Materials / Update Selected Materials



Álvaro Holguera
CIFP José Luis Garci
(Madrid)

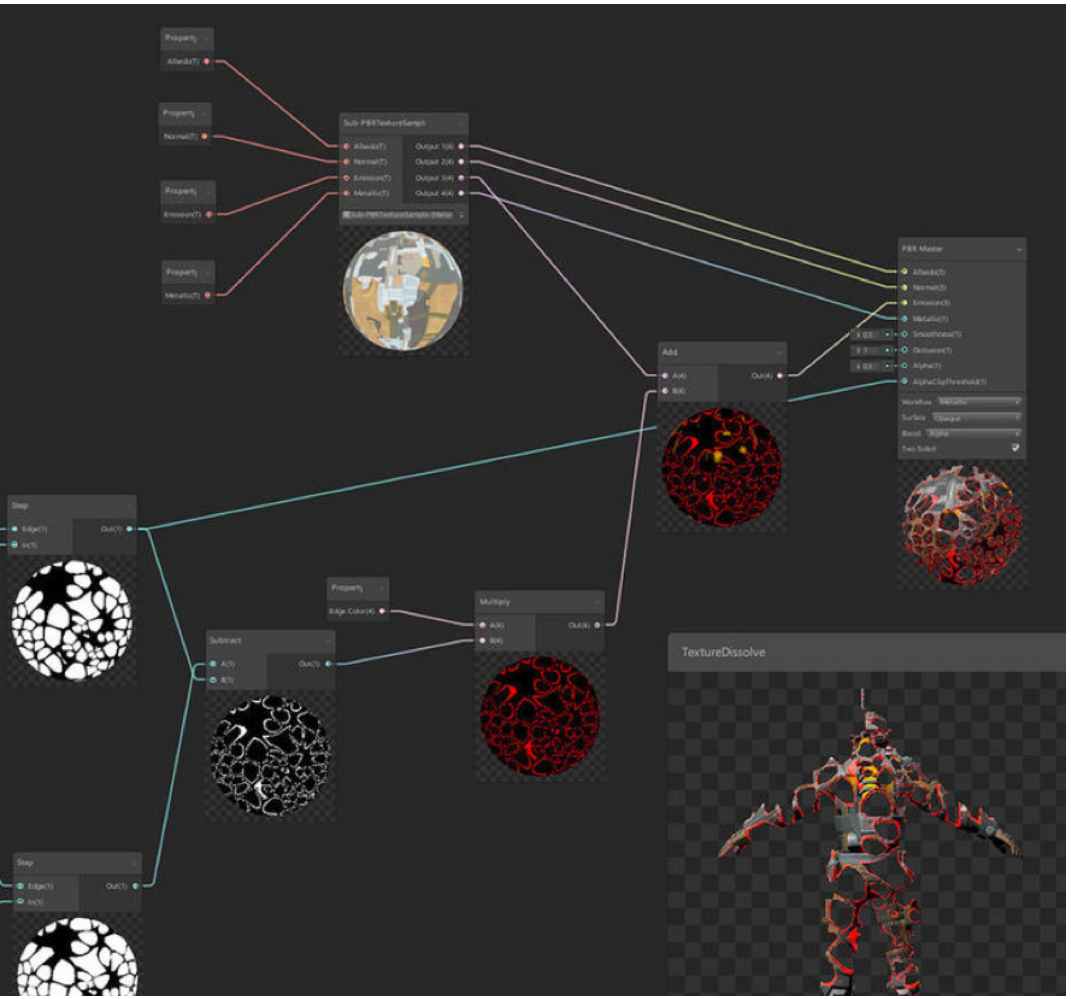


SHADER GRAPH

Conozcamos esta herramienta que nos permite obtener texturas complejas mediante la creación de nuestros shaders propios.

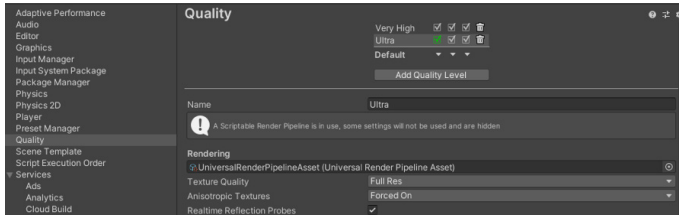
SHADER GRAPH

La herramienta Shader Graph, disponible para Universal Render Pipeline y High Definition Render Pipeline, permite crear nuestros propios shaders mediante SRP pero sin necesidad de programar, ya que se realiza de fora nodal



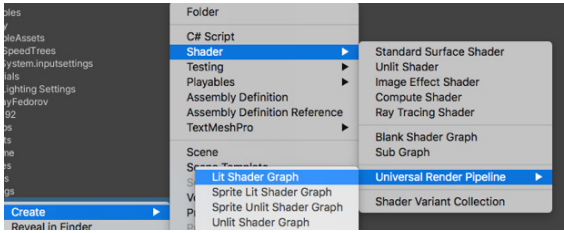
Antes de comenzar, debemos asegurarnos que tenemos el proyecto en URP ó HDRP, ya sea creando uno desde el principio o actualizando uno anterior como hemos visto en el anterior capítulo.

También deberemos vincular el archivo de tipo Renderer que generó a nuestra configuración del proyecto, en la sección de Quality:



Crear un shader

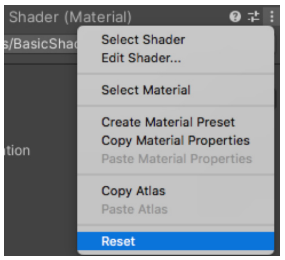
Para utilizar la herramienta de Shader Graph, debemos crear un Shader en el proyecto : Botón derecho > Create > Shader > URP / HDRP. Dependiendo de si trabajamos con URP ó HDRP tendremos disponibles unos shaders u otros.



Creará un archivo con el icono de shader, que si hacemos doble click o pulsamos el botón "Open Shader Editor" del inspecctor nos abrirá el Editor de Shader Graph.

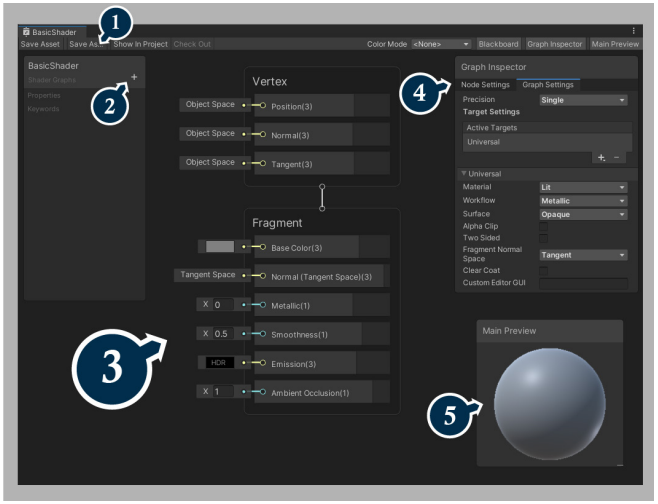
Podemos crear un material a partir de este shader para aplicarlo a un Game Object, pulsando con el botón derecho en el shader > Create > material.

IMPORTANTE: una vez está asociado el material al shader, si cambiamos las propiedades de ese shader, además de guardar los cambios tendremos que resetear el material para verlos (en el menú del inspector del material)

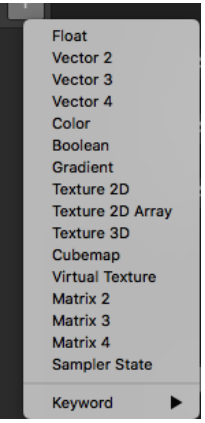


SHADER GRAPH EDITOR

El editor de Shader Graph se compone de:

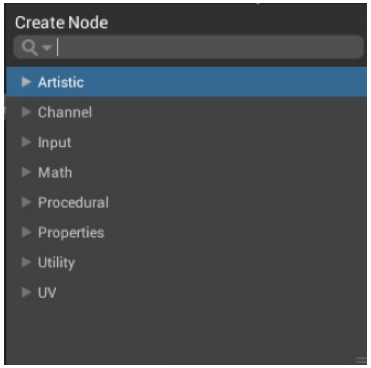


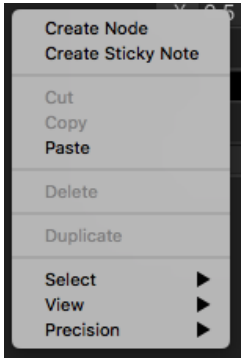
- Barra superior (1) para guardar el asset y mostrar/ocultar los paneles. Es importante guardar los cambios para que se apliquen, y si es necesario, resetear el material con el Shader.



- Panel para añadir propiedades al shader (2). Hay muchos tipos de propiedades y valores, así como palabras clave. Si inspeccionamos el material con este shader, podremos ver allí los atributos creados y sus valores.

- Sistema de nodos (3). Podemos navegar por él, hacer zoom, paneo, etc. (cn la tecla "F" centramos la vista en el nodo seleccionado, o en todos). Para crear un nodo nuevo, pulsaremos la barra espaciadora, o pulsando con el botón derecho del ratón. Otro método es creando un nuevo vínculo desde el nodo y con con el botón derecho del ratón se abrirá el menú contextual.

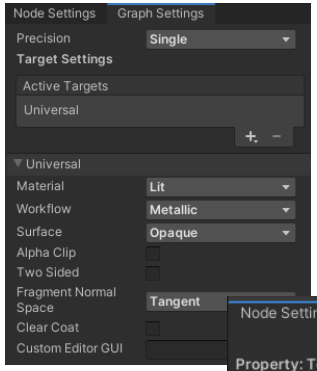


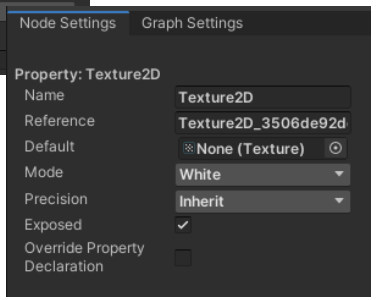


- Graph Inspector (4). Posee dos pestañas:

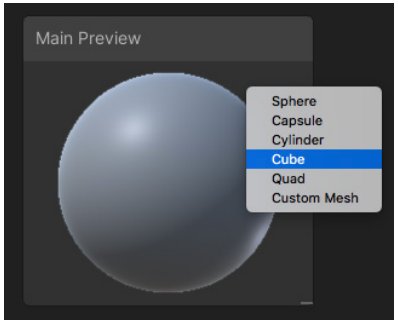
  - Graph Settings: configuración general del shader

  - Node Settings: parámetros del nodo o de la propiedad seleccionada. Dependiendo del tipo de propiedad, saldrán unas opciones u otras. Lo usaremos más adelante también para acceder a él desde código.

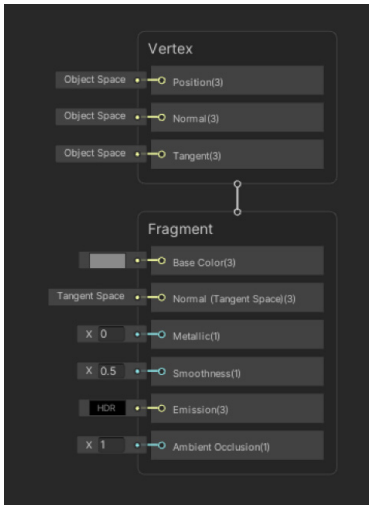




- Previsualización (5): veremos el resultado final. Podemos cambiar la forma de previo haciendo click con el botón derecho, e incluso añadir una malla nuestra de nuestro proyecto. También podemos ampliar la ventana y hacer que gire con el click del ratón.



CREANDO UN SHADER

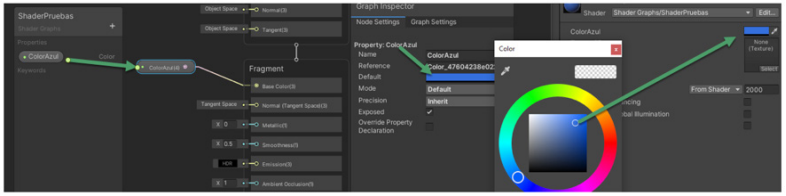


Al crear un shader tendrá 2 nodos por defecto:

- Vertex: afecta a la geometría del objeto, permitiendo crear mallas animadas (por ejemplo, olas)
- Fragment: parámetros básicos del material, como color base, mapa de normales, emisión, AO, etc.

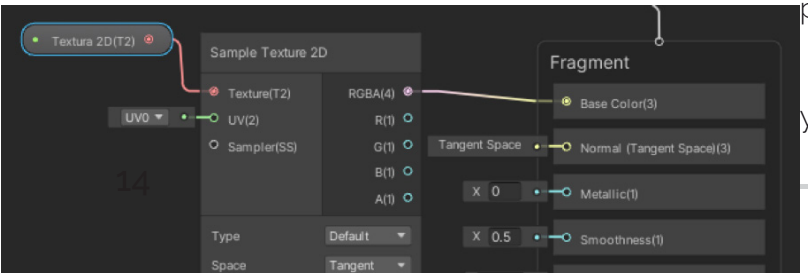
Podemos crear propiedades nuevas para asociar a este shader: pueden ser colores, texturas, vectores (por ejemplo para crear un número crearemos un vector1, etc.).

Al crear un nodo nuevo (barra espaciadora ó arrastrando un enlace) debemos seleccionar el tipo de nodo. Lo mejor es usar el buscador de nodos. Cada nodo tiene sus propios parámetros que pueden vincularse a otros nodos.



Al añadir un atributo (como color), podemos sacarlo como nodo y vincularlo a las características del shader. En el panel de Graph Inpsector>Node settings podemos configurar ese parámetros. También aparecerá como propiedades del material creado a partir del shader (y podremos cambiarlo mediante código)

Por ejemplo: para aplicar una textura al shader, y cambiar el "Base Color" que tiene por defecto, podemos crear un nodo de tipo "Sample Texture 2D", la cuál tiene dos propiedades: la textura, y el mapa de UVs. Para añadir una imagen, crearemos en la ventana de propiedades una nueva de tipo "Texture2D", a a que podremos vincular una imagen en el "Node Settings". Ahora podemos conectar los nodos: la propiedad al nodo de Textura, y la salida RGB de esta al Base Color del Shader. En el previsualizador podremos ver el resultado

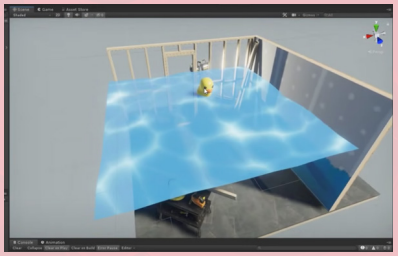


Para romper un vínculo, basta con seleccionarlo y arrastrarlo fuera del vínculo.

AGUA

Uno de los materiales más habituales y complejos es el agua, presente en muchos escenarios.

Puedes buscar assets en Internet, pero para practicar con el editor de Shader Graph, te proponemos los siguientes tutoriales:



https://www.youtube.com/watch?v=Vg0L9aCRWPE

https://www.youtube.com/watch?v=MHdDUqJHJxM

https://www.youtube.com/watch?v=gRq-IdShxpU

ANIMANDO UN SHADER

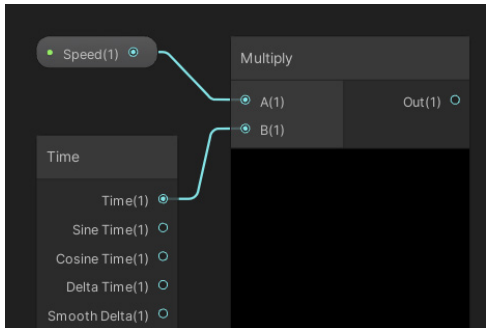
Anteriormente vimos cómo animar una textura accediendo a los parámetros del Mesh Renderer mediante código, ahora veremos cómo añadir elementos temporales en Shader Graph que se pueden vincular a cualquier atributo, por ejemplo, sus coordenadas UV.

Seguiremos estos pasos:

1. Crearemos una propiedad de tipo float que determinará la velocidad.

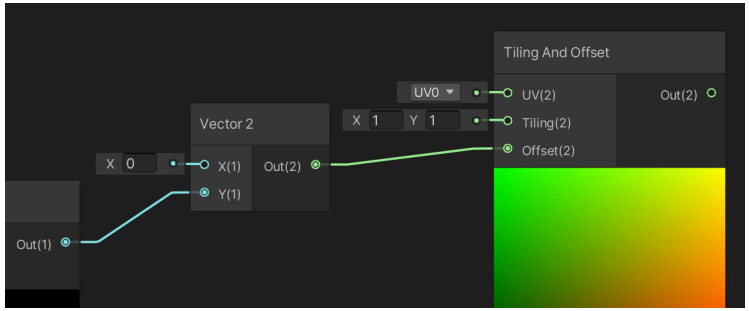
2. Añadiremos un nodo "Time" que permite aplicar cambios temporales.

3. Otro nodo "Multiply" para poder controlar esos cambios temporales mediante la variable "float". Acabamos de crear un número que irá creciendo con el tiempo, a la velocidad que le digamos:



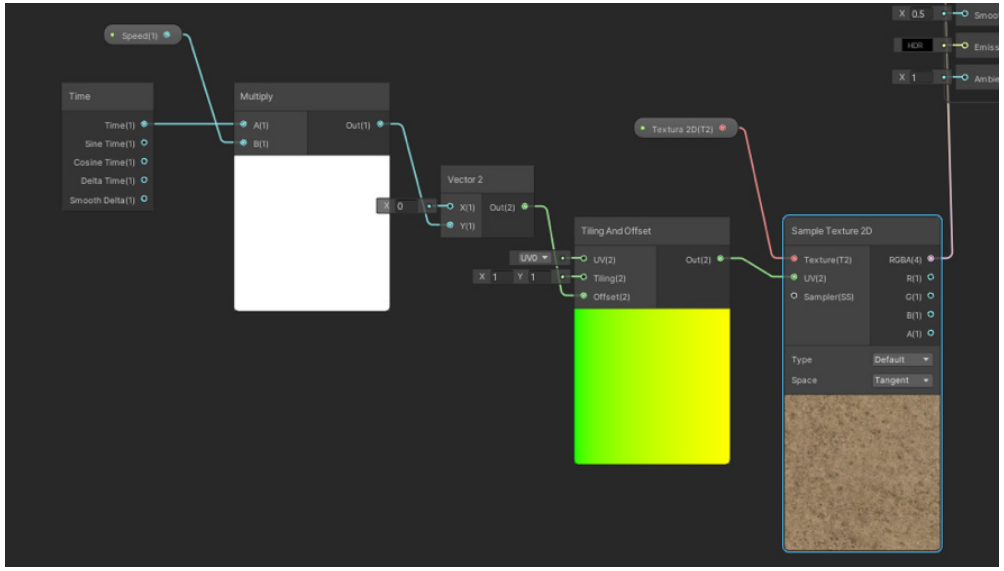
4 Un nodo "Vector 2" para poder aplicar a su valor en "Y" esos cambios temporales, conectando la salida del nodo "Multiply"

5. Ahora, crearemos un nodo "Tiling And Offset" que permite gestionar esos parámetros de una textura. El Vector 2 creado lo vinculamos al Offset.



6. Aplicamos la salida de este nodo al parámetro de UV de nuestra "Simple Texture 2D", al que tenemos vinculada nuestra Textura 2D.

7. Finalmente aplicamos la textura a la Base de nuestro Shader.



Con esta configuración, la textura se desplazará en su coordenada "Y" a la velocidad que establezcamos en nuestro parámetro "speed" (tendrás que cambiarlo, porque por defecto es cero).

NOTA: las animaciones no se ven por defecto en la escena para no consumir recursos, pero si pulsamos "Alt" al pasar por encima de ella, veremos las texturas animadas.

Y acuérdate de guardas los cambios al Shader ("Save Asset").

CONTROLANDO LOS ATRIBUTOS

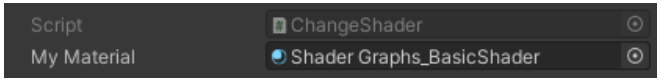
Mediante código odemos acceder a cualquiera de los parámetros creados para nuestro shader y, por tanto, del material.

Para ello debemos crear un script asociado al Game Object con el material y seguir estos pasos:

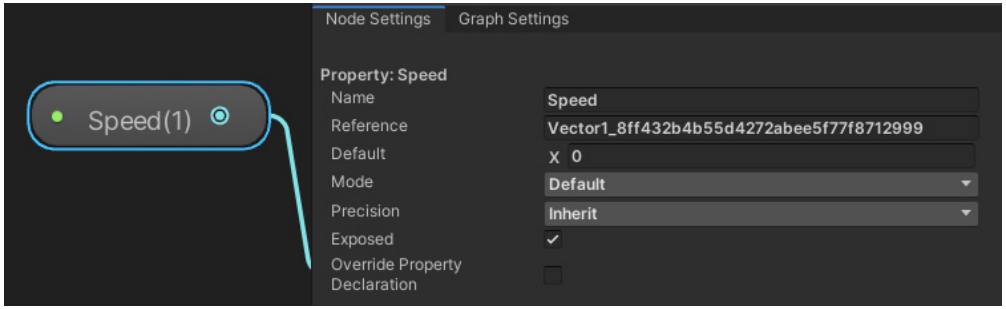
1. Crear una variable de tipo Material (serializado o pública):

```
[SerializeField] Material myMaterial;
```

2. Arrastrar en Unity el material a esa variable, directamente desde el proyecto:



3. Obtener la referencia del parámetro (la encontraremos en la pestaña de "Node Settings" del Graph Inspector", en "Reference")



Mediante código, podemos usar los métodos que nos permiten actualizar los valores de los parámetros a partir de su referencia, mediante este método en el que pondremos la referencia copiada donde pone "referencia":

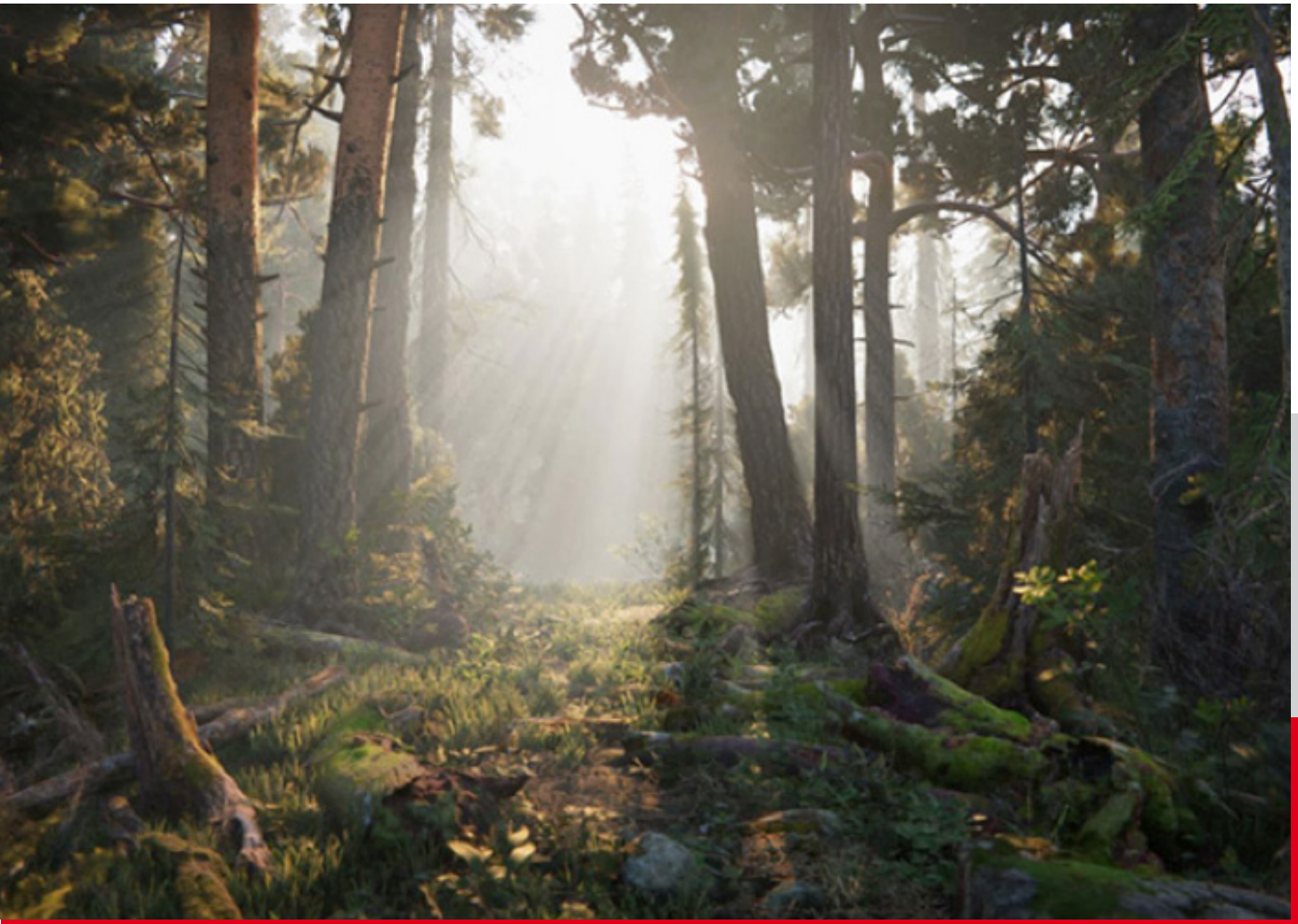
```
variableCreada.SetFloat("referencia", valor);
```

De esta forma, podemos controlar todos los parámetros de nuestro material. Por ejemplo, en el siguiente script haremos que la velocidad de la animación se vaya acelerando a medida que pase el tiempo:

```
[SerializeField] Material myMaterial;

// Update is called once per frame
void Update()
{
    float speedUpdate = Time.time * 0.1f;
    myMaterial.SetFloat("Vector1_8ff432b4b55d4272abee5f77f8712999", speedUpdate);
}
```

Álvaro Holguera
CIFP José Luis Garci
(Madrid)

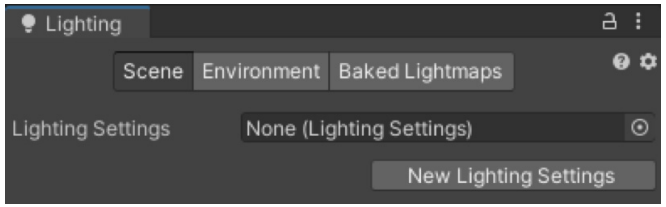


Parte 2
LUCES

Las luces se encargan de dotar de personalidad a una escena. Interactúan con los materiales y dotan de realismo a todo el juego.

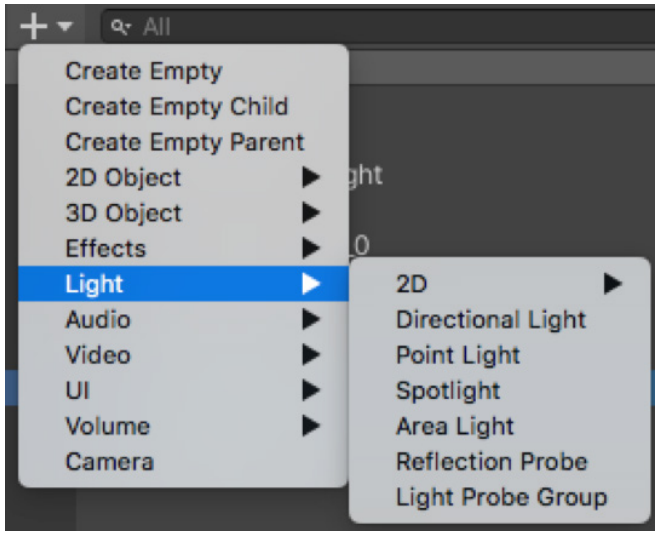
AÑADIENDO LUCES

Antes de comenzar a crear luces, y para poder tener acceso a ciertas funcionalidades, debemos primero establecer unos parámetros generales en la escena, es decir, los "lighting settings". Para ello, en la pestaña "Scene" deberemos crear un nuevo archivo con esos parámetros (si no está creado ya).



A partir de ese momento puedes ajustar la calidad general y el rendimiento, en la pestaña "Scene" del panel Lighting.

Para añadir una luz, iremos al menú Game Object->Light y elegiremos el tipo de luz.



MPORTANTE: para ver el resultado en tiempo real en aquellas luces que lo permiten, debemos activar la bombilla del panel de la escena.

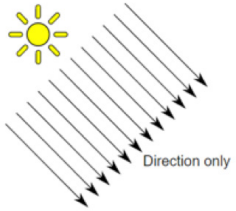
TIPOS DE LUCES

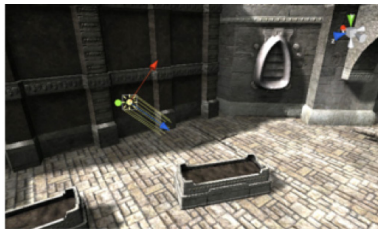
Para este tema conviene tener abierto y disponible el panel “Window > Rendering > Lighting”.

Existen 4 tipos básicos de luces en un entorno 3D:

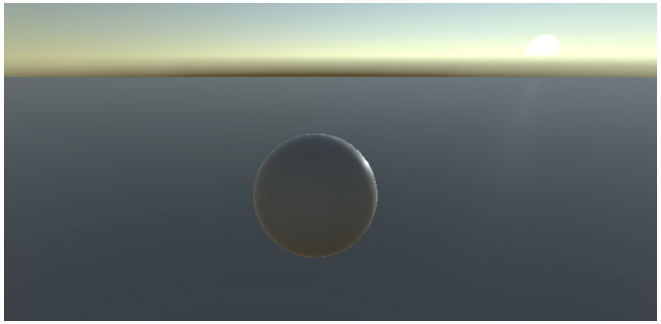
Directional Light

Luz direccional que simula la luz proyectada por el sol. La cantidad de luz que llega a los objetos es independiente de su posición o distancia (como la del sol), y el único parámetro que permite ajustar es la rotación para indicar la dirección de las sombras.



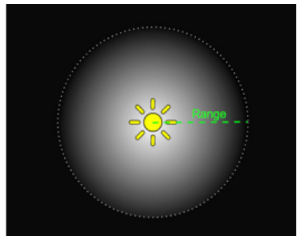


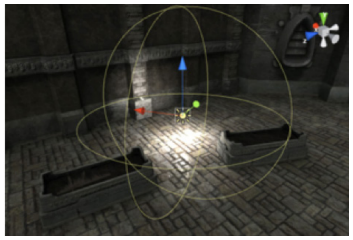
Si giramos mucho la luz en el eje X veremos que se tinta de anaranjado, como una puesta de sol:



Point Light

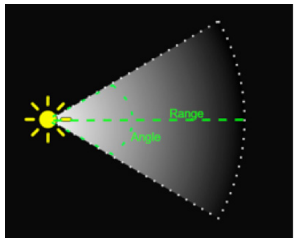
Luz de punto que simula la luz emitida por una bombilla (o una explosión). Emite la luz en forma de esfera, igual en todas las direcciones hasta un rango definido (la cantidad de luz que llega a los objetos se reduce el cuadrado de su distancia).

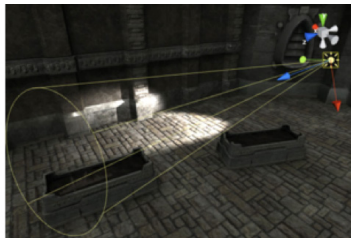




Spot Light

Luz de foco que simula la luz de una linterna. Emite la luz en forma de cono.





La cantidad de luz no solo disminuye con la distancia, también se aleja del punto central del cono, la zona de penumbra. Poodemos ajustarlo en un parámetro que solo tiene esta luz que indica el ángulo:

Area Light

Emite luz en un área rectangular. (solo hacia un lado del rectángulo) Genera luces más realistas, con sombras más suaves.

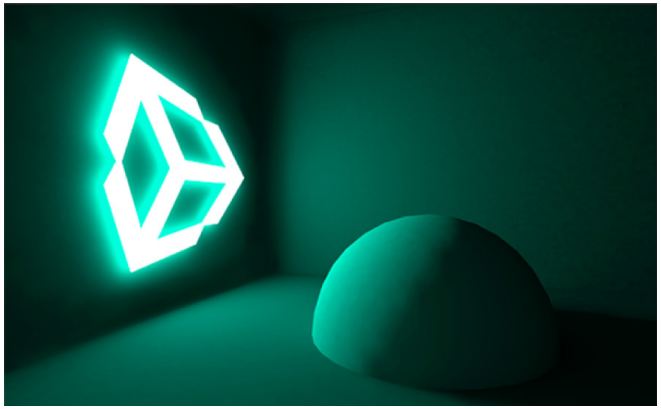
Estas luces consumen mas recursos, por lo que no pueden calcularse en tiempo real (por eso no se ve su luz en la escena al añadirlas) y tendrán que ser “horneadas” (baked) en “lightmaps” como se verá luego.

LUCES ESPECIALES

Otra consecuencia en las lluces de área, al no soportar luz en tiempo real, es que solo iluminan objetos estáticos, como veremos luego. Una alternativa es usar sondas de luz ("light probes").

Pambién podemos recurrir a las texturas que emiten luz, las cuáles sí funcionan en tiempo real.

Estas tendremos que crearlas mediante postprocesos o ajustando parámetros en el Shader Graph, ya que la propiedad "Emission" del material no funciona en URP ni en HDRP. Más información en: https://docs.unity3d.com/Manual/lighting-emissive-materials.html

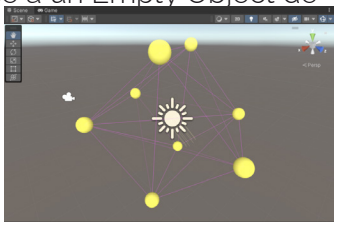


RECUERDA: otra fuente de luz en la escena es el SkyBox. En la pestaña de Environment del panel de Lighting podras indicar si emite luz y cuánta.

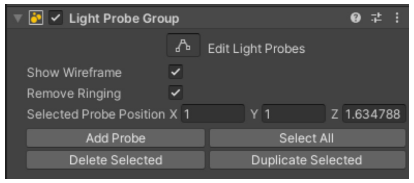
Sondas

Las "light probes" y las "reflection probes" (sondas de luz y de reflexión) permiten capturar información de la luz en una zona determinada, para ofrecer resultados más realistas con menor coste, incluso en objetos NO estáticos.

Funciona a través del componente "Light Probe Group" que suele añadirse a un Empty Object de la escena y que genera automáticamente 8 sondas en formación que podemos ajustar individualmente.



En el inspector del componente, podemos modificar cada sonda de forma individual, añadir nuevas o borrar las creadas. Lo ideal es cubrir toda la escena con el menor número posible de sondas y de grupos.



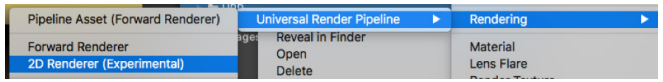
Si tenemos luces estáticas en nuestra escena, apenas necesitamos sondas. Pero si tenemos luces moviéndose y objetos no estáticos, necesitaremos más sondas para cubrir todas las posiciones. Aquí puedes ver más información.

Luces 2D

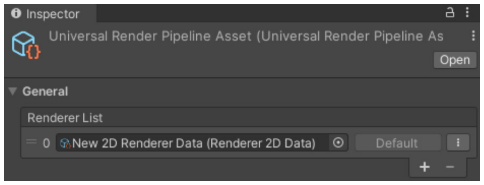
Unity nació como un motor para juegos 3D y por tanto las luces no afectaban a elementos en 2D. Por defecto, los sprites 2D ignoran las luces. Esto obligaba a crear elementos tridimensionales si queríamos iluminarlos con luces en un juego 2D.

Pero en las últimas versiones ha incluido un renderizador de luces 2D, y con ello, lluces 2D. Está incluido solo para URP y HDRP, los cuales se pueden instalar aunque estemos trabajando en un proyecto 2D.

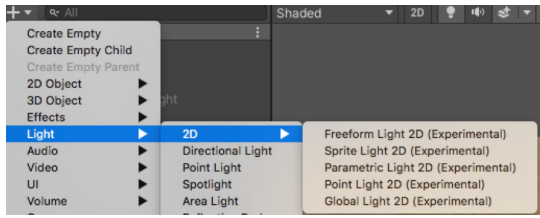
Ahora podremos crear un"2D Renderer" dentro del menú "Rendering > URP /HDRP"



Ahora, deberemos asignar este nuevo "Renderer" a nuestro asset de Render Pipeline (el que creamos al instalar el paquete de URP):



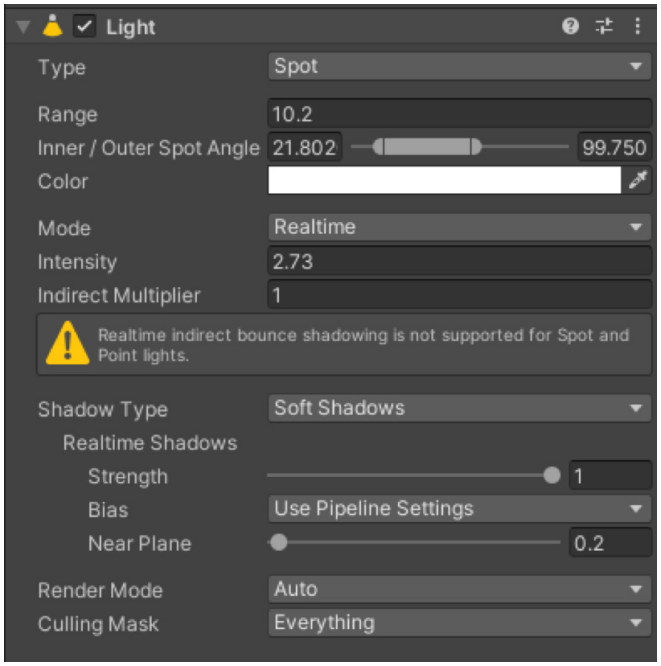
Esto nos permitirá ahora usar las luces 2D.



PARÁMETROS DE LAS LUCES

Casi todos los tipos de luces comparten los mismos parámetros. Veamos cuáles son.

En la ventana de inspector encontraremos las propiedades, las cuales dependerán del tipo de luz elegida. Estos son los parámetros más habituales:

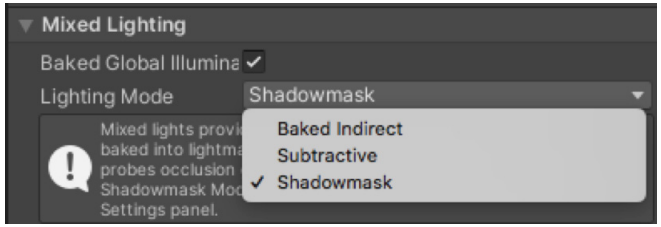


- Rango: Determina la distancia llega la luz emitida. En el caso de la luz direccional (el sol) no está disponible.
- Ángulo: en luces de tipo spot, determina el ancho del cono de luz, tanto el interior coomo el exterior, generando penumbra entre ambos.
- Color: permite tintar una luz.
- Modo: determina cómo Unity calcula la forma en la que la luz interactúa con los elementos:

  - **Realtime**: se actualiza en cada fotograma, sin cálculos previos (no todas las luces lo soportan)

  - **Baked** ("horneado"): el cálculo se hace previamente sobre objetos estáticos, liberando el procesamiento. Crea mapas de textura nuevos en los que se reflejan los efectos de la luz (los podemos ver en el componente Mesh Renderer de los objetos estáticos).

  - **Mixed**: Unity realiza algunos cálculos previamente y otros en tiempo real. En la pestaña de "Scene", en Mixed Lighting tendremos configuraciones que afectan a las luces de tipo "baked" y "mixed" (por ejemplo, "Baked Indirect" consume más recursos que "Substractive") para luces de tipo "mixed".

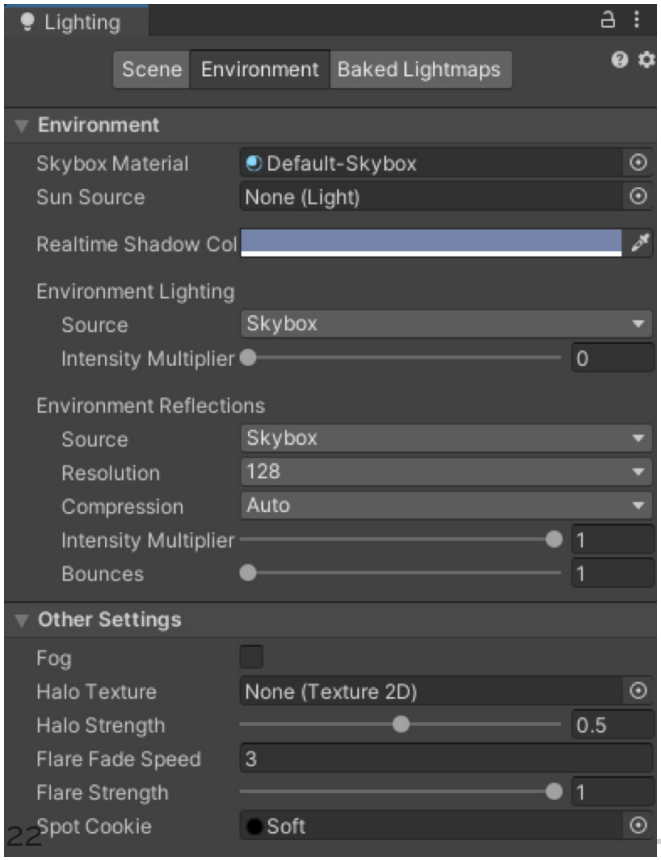


- Intensidad: para luces puntuales, spot y de área el valor por defecto es 1. Para la direccional es 0,5.
- "Indirect Multiplier": cómo se comportan los rayos reflejados en las superficies. Si el valor es menor que 1, los rayos cada vez son más débiles, lo contrario si es mayor que 1
- Sombras: si activamos la opción de proyectar sombras (duras o suaves), y tenemos el modo de la luz en "Realtime" ó "Mixed" podremos configurar otros parámetros para las sombras en tiempo real, como por ejemplos a qué distancia se proyectan (Bias). Esta opción no está disponible para luces puntuales.
- Otros parámetros, como texturas (Cookie), halos, destellos (Flare), prioridad en el renderizado, o incluso indicar a qué capas afecta esta luz (Culling Mask)

LUZ AMBIENTAL

Se conoce como luz ambiental aquella que inunda la escena pero que no viene de una fuente en concreto. Contribuye a crear un ambiente especifico, permitiendo iluminar espacios sin necesidad de añadir luces, o para crear ambientes sin luces pero que se vean.

En el menú Window->Rendering>Lighting settings podemos abrir la ventana de control de luces, concretamente la pestaña de "Environment".

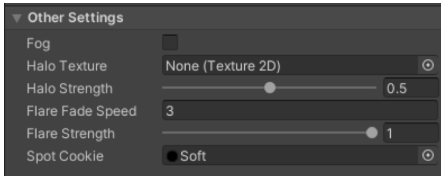


Tendremos varias pestañas. La pestaña de "Scene" controla la Iluminación Global (GI) de la escena. Apartados:

Environment: permite configurar de dónde viene la luz ambiental y cómo se comporta

- Skybox: si creamos un material de este tipo podemos usarlo para que se "rodee" la escena. Si el material es procedural, podemos indicar una luz de la escena como sol.
- Environment Lighting: de dónde viene la luz ambiental, y con qué intensidad. Si en lugar de "SkyBox" elegimos "Gradient", podemos separar colores para el cielo, el ecuador y el suelo
- Environment reflections: configuraciones que afectan a las "Reflection Probe"

En este apartado también tenemos la posibilidad de crear niebla, halos o flares (para estas últimas tenemos que crear un elemento de tipo Lens Flare en nuestro proyecto).

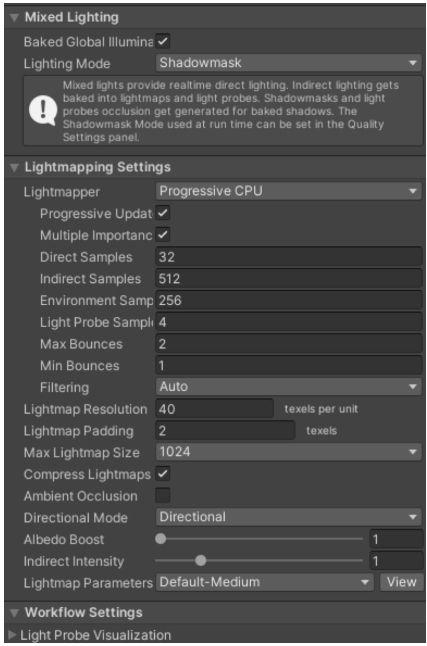


Finalmente, abajo encontraremos dos botones: "Auto Generate" y "Generate Lighting", que permite indicar si queremos que Unity actualice automáticamente los lightmaps, y si queremos que genere de nuevo las texturas con las luces "horneadas".

Global Ilumination (GI)

Por defecto, las luces en tiempo real no contribuyen a la iluminación global de la escena, pero si activamos en la pestaña de "Scene" la casilla de "Baked Global Illumination" permitiremos a esas luces que sus rayos reboten, tantas veces como le hayamos dicho en sus configuraciones.

Tendremos que buscar un equilibrio entre la calidad de la iluminación (su realismo principalmente) y el rendimiento.

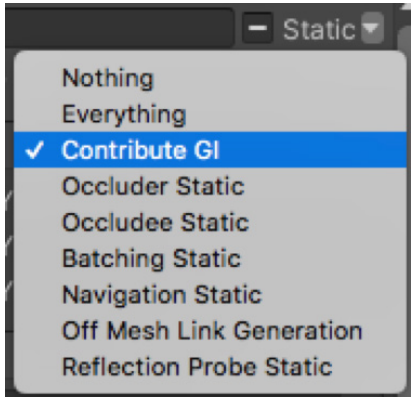


Más información en https://docs.unity3d.com/Manual/realtime-gi-using-enlighten.html

“HORNEADO” DE LUCES

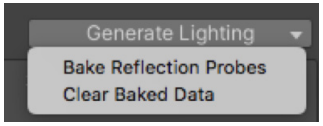
El “horneado” de luces permite procesar cómo se comportan con el entorno y aplicar a las texturas de los materiales sus efectos antes de que el juego se ejecute (los llamados “lightmaps”), reduciendo enormemente los requerimientos del sistema para crear una luz realista.

Para poder realizarlo, lo primero que tenemos que hacer es indicar los objetos que son “estáticos” y que por tanto no van a cambiar de posición durante el juego. En la parte superior derecha del inspector veremos la casilla:

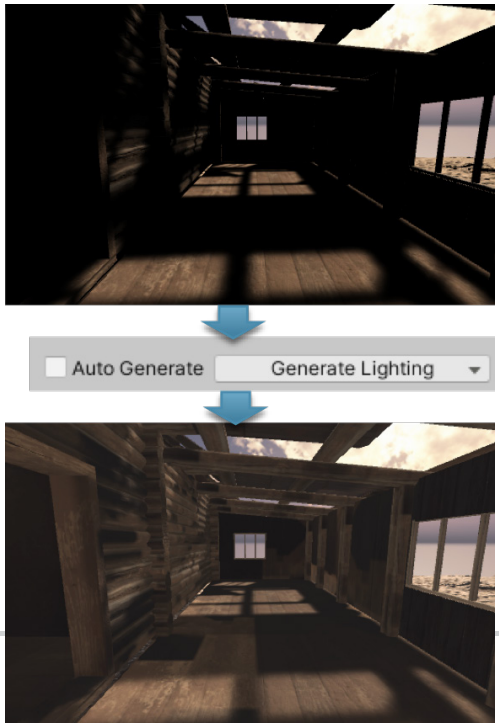


En el desplegable, podemos indicar incluso para qué elementos debe comportarse como estático, en este caso para “Contribute GI” (ó “Lightmap Static” en versiones anteriores) y “Reflection Probe Static”. Si el elemento seleccionado tiene hijos, nos preguntará si queremos aplicarlo a ellos también.

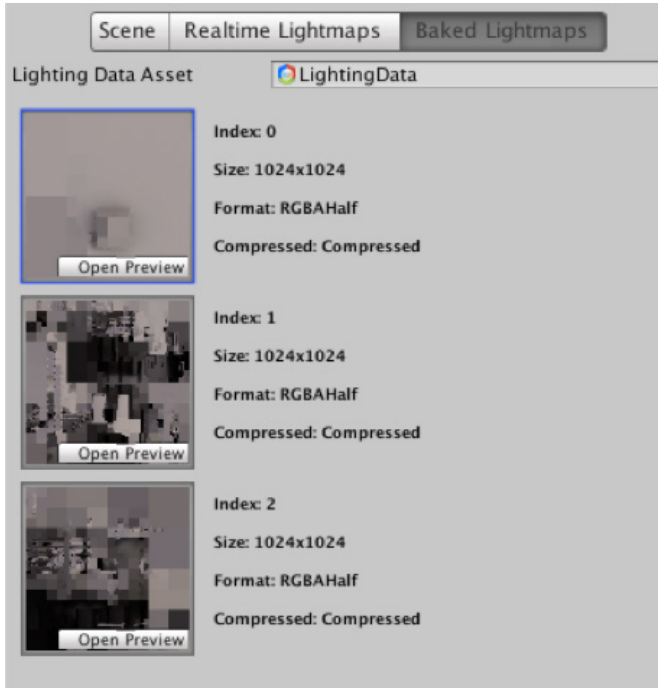
Para que los cambios se apliquen, debemos ir al panel de Lighting y pulsar en “Generate Lighting” (podemos también hornear solo las sondas, o eliminar los mapas generados). El proceso llevará un rato, pero al final veremos cómo se ha generado luz ambiental en los elementos estáticos de nuestra escena.



NOTA: los objetos tienen que tener un correcto mapa de UV’s, si no lo tiene, podemos marcar la casilla de “Generar Lightmap UV’s” en el inspector del objeto importado.



Podemos ver el resultado del “horneado” en forma de imágenes de textura, es decir, los lightmaps, en la pestaña de “Baked Lightmaps”.



TRUCO: el horneado de luces es un proceso que en escenas complejas puede llevar mucho tiempo. Por ello, a veces es preferible bajar la calidad de las luces, los reflejos y reducir el tamaño de los mapas y aumentarlos en un “bakeo” final.

POSTPROCESOS

A menudo, los efectos que queremos generar con las luces es mejor producirlos directamente en la cámara para consumir menos procesamiento Hay que tener en cuenta que si el proceso de postproducción lo hacemos en la imagen 2D que capta la cámara, el consumo de recursos es menor

Algunos efectos: Bloom, profundidad de campo, motion blur, Viñeteado, etc.

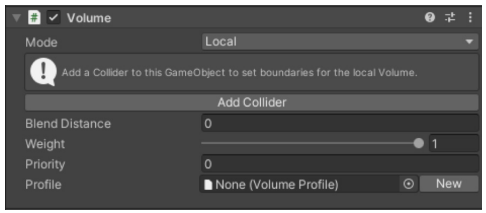
Volume Based Framework

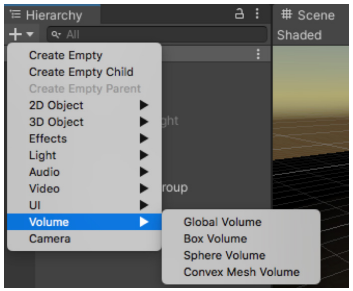
Tanto URP como HDRP introducen lo que se llama un "Volume Based Framework". Permite definir áreas en las que configurar cómo serán renderizadas, y así controlar elementos como nieblas, luces, cielos, post-processing, sombras, etc.

Un volumen puede ser global (afecta a toda la escena) o local (afectando a un área específica)

Podemos crearlo de dos formas:

- Crear un Empty Object y añadirle el componente "Volume". Si decidimos que sea "Local", tendremos que añadir un Collider (de tipo caja, esférico o de malla con el Convex activado (los colisionadores tienen que tener el Is Trigger activado).



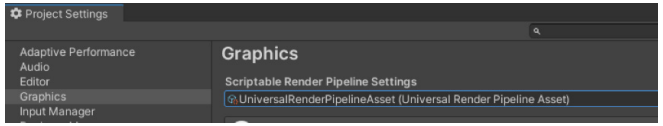


- Más rápido: podemos añadir a la escena un volumen como si fuera un GameObject más, y creará el Empty con todo lo anterior ya añadido.

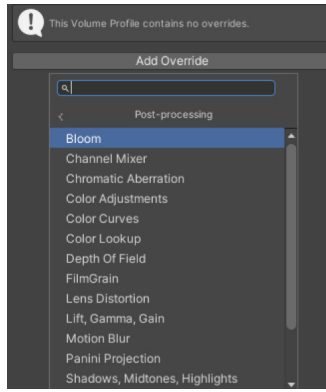
En un volumen local, cuando la cámara está fuera del colisionador no se ve afectada, pero dentro sí. Editando el tamaño del colisionador ajustamos su rango de influencia. Para que la transición sea suave, podemos aumentar el parámetro de "Blend Distance" del componente Volume.

Con el volumen de la escena seleccionado, en el componente "Volume" creamos un nuevo "Profile" (o le asignamos uno si ya tenemos alguno creado). Se creará por defecto en una subcarpeta de la escena en la que estamos:

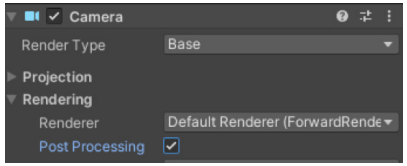
NOTA: tenemos que asegurarnos de que en nuestros Project Settings en el apartado de Graphics tenemos un asset de renderer asitnado.



Seleccionamos el profile creado en el proyecto, y añadimos "Overrides" (los que están en "post Processing). Verás bastantes opciones:

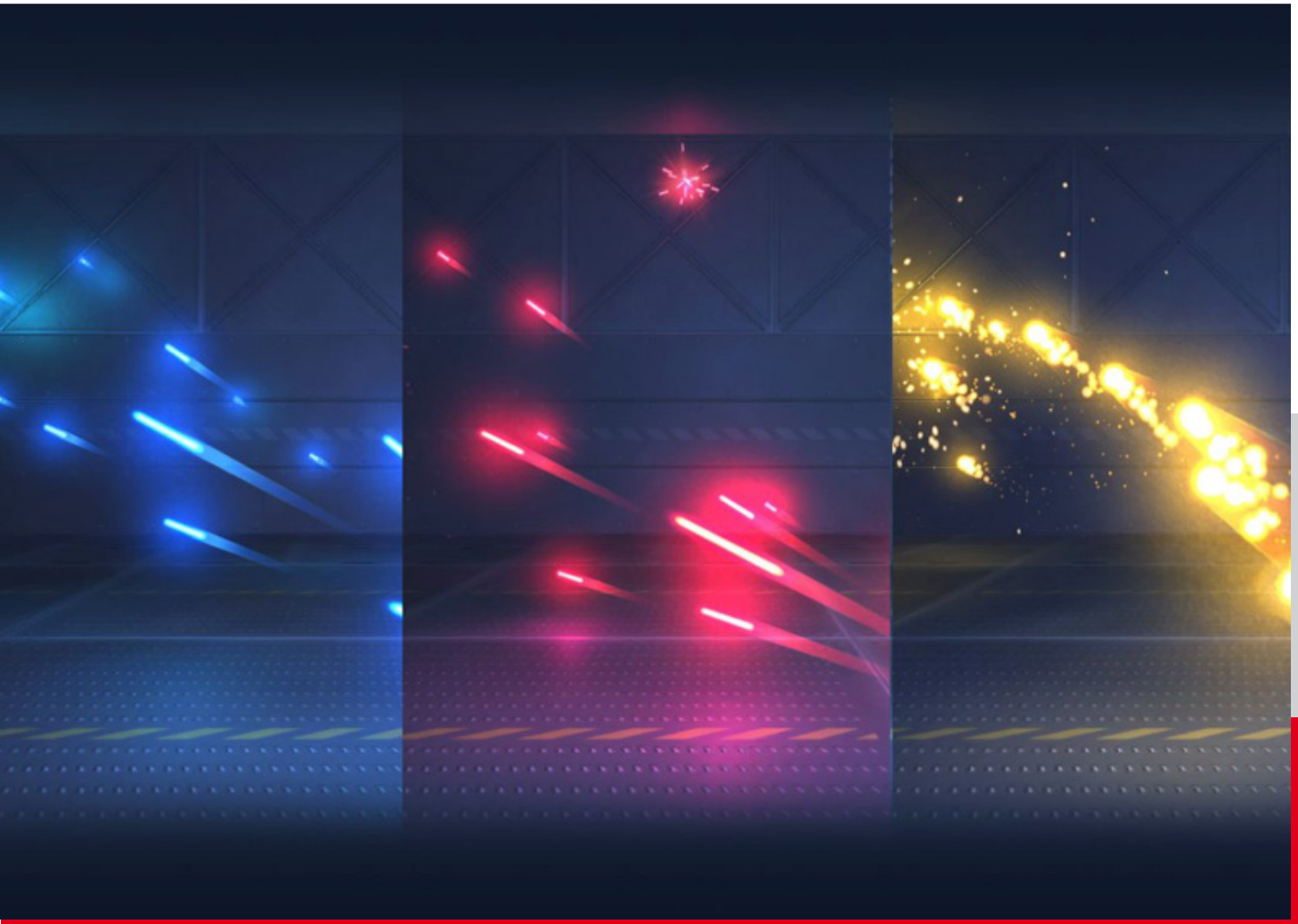


Por último, debemos activar en la cámara la casilla de "Post Processing" y veremos los resultados del profile.



NOTA: Existe un paquete que permite realizar efectos de post-proceso de forma similar y que se puede instalar aunque no se trabaje en URP ó HDRP. Puedes ver como funciona en este enlace: https://docs.unity3d.com/Packages/com.unity.postprocessing@3.2/manual/index.html

Álvaro Holguera
CIFP José Luis Garci
(Madrid)

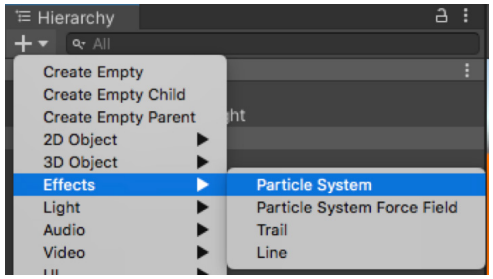


SISTEMAS DE PARTÍCULAS

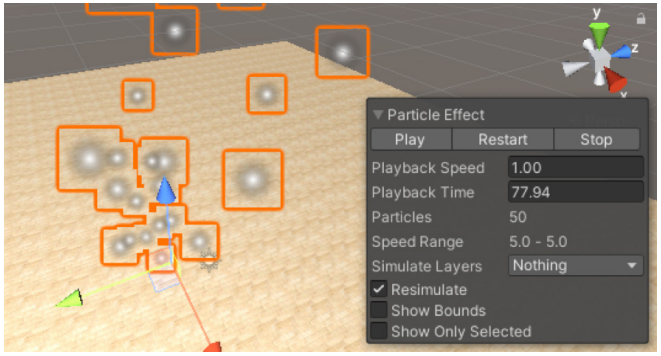
Los sistemas de partículas permiten crear efectos como explosiones, fuego, nubes, etc.

SISTEMAS DE PARTÍCULAS

Para crearlos debemos añadir un Game Object de tipo Effects > Particle System.



Automáticamente se crea un Empty Object que con el componente "Particle System" añadido, y que cuando lo seleccionamos comienza a emitir partículas.

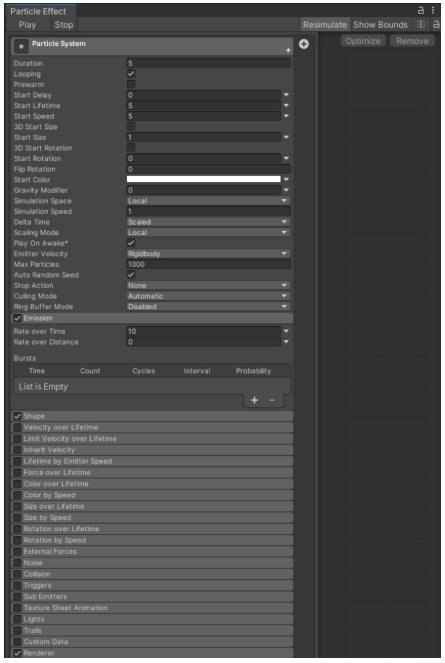


En el panel de escena podemos ver el panel flotante de las partículas donde podemos ajustar algunos de sus parámetros.

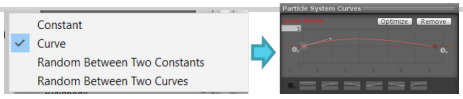
Otra opción es añadirlo como componente a un objeto del escenario, lo que nos permite convertir su malla en un emisor de partículas (podemos desactivar su Mesh Renderer para que solo se muestren las partículas que emite).

En el inspector del objeto podemos ver el panel con los parámetros del sistema de partículas, aunque es recomendable abrir el panel (pulsando el botón "Open Editor" o directamente en "Window > Panels > Particle Effect")

Los parámetros del inspector podemos desplegarlos para ver todas las opciones.



Muchos de los modificadores podemos cambiarlos para usar valores absolutos, escalas, aleatorios, curvas, etc.



Parámetros

Veremos algunos –no todos- de los parámetros que podemos modificar.

El primer apartado, y el más importante, es el de "Particle System", Donde podremos ajustar algunos parámetros básicos:

Duración: tiempo en segundos que durará

Loop: se repite indefinidamente

Start Delay: tiempo que pasa antes de que comiencen a emitirse las partículas

Start Lifetime: segundos que durará la partícula emitida

Start speed: velocidad a la que se mueven al comenzar (se podrá reducir con el tiempo)

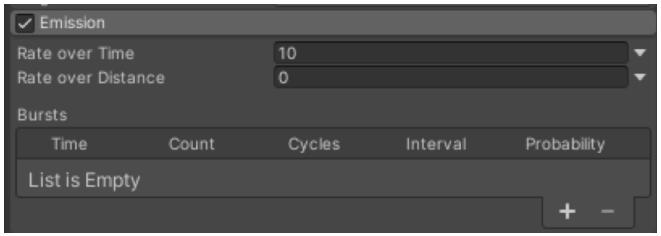
Start Size: tamaño al inicio

Start color: podemos elegir varios patrones. Al hacer click, accederemos a la configuración de cada opción: Color plano, degradado, aleatorio, o aleatorio entre dos colores o dos degradados.

Gravity Modifier: cómo afecta la gravedad a las partículas (si es 0, no le afecta).

SISTEMAS DE PARTÍCULAS (parte 2)

El siguiente apartado es la emisión:

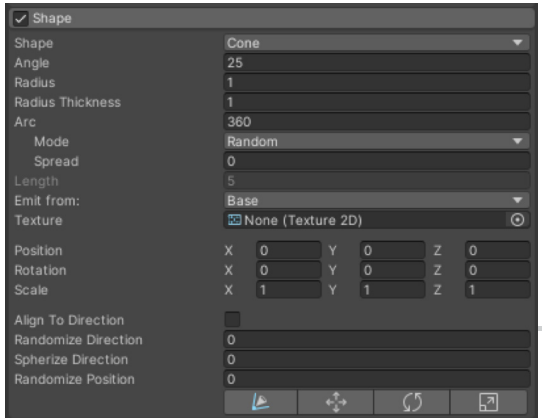


Rate Over Time: tiempo que pasa entre que una partícula es emitida con respecto a la anterior.

Rate Over Distance: similar a la anterior pero usado con el parámetro de distancia cuando un objeto está en movimiento, muy útil para estelas de fuego o de humo, ya que permite aumentar la emisión en caso de que se mueva a gran velocidad.

Bursts (ráfagas): cuántas partículas puede lanzar de una sola vez, en ráfagas regulares.

A continuación, determinamos la forma del emisor (Shape):



Shape (caja, esfera, cono, donut, etc) Podemos añadir una malla como emisor.

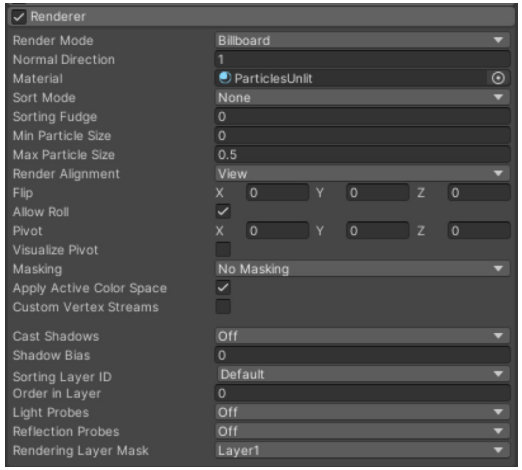
Podemos configurar los **parámetros de la forma**: radio, arco, posición, rotación, etc.

Desde dónde se emite: desde la superficie, desde el volumen, etc.

Texture: permite añadir una imagen 2D para que las partículas tomen el color de ella.

Los **parámetros de dirección** permiten indicar cómo se van a alinear las partículas en relación con la forma que las emite, útil cuándo esta se mueve.

Otro de los apartados importantes es el "Renderer", donde se define la forma de las partículas:



Es importante tener en cuenta que podemos usar imágenes 2D, incluso creadas por nosotros, para incorporarlas como materiales a nuestro sistema de partículas.

Render Mode: elegimos si queremos una imagen siempre mirando a cámara (Billboard y todas sus variantes), o usando una malla que elijamos. Si elegimos una malla, deberemos asignarle un material, y un shader.

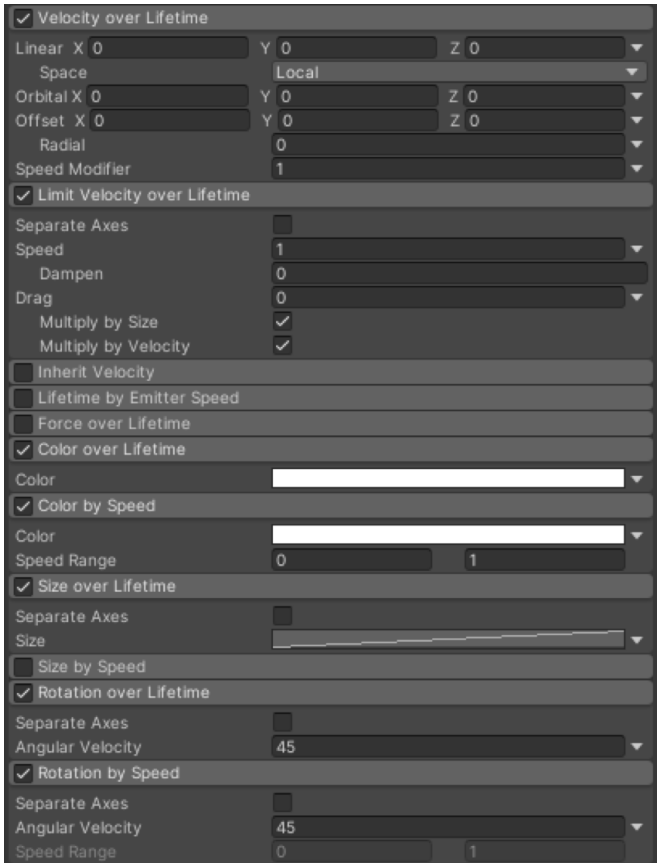
Veremos que podemos elegir una serie de Shaders de tipo "Particles", que definen cómo se verá ese material. Se recomiendo usar "Particles/ Standard Surface" si no queremos usar un shader en particular

Material: deberá ser un tipo de material para partículas (Shader de tipo "Particles > Standard Surface", y en Rendering Mode mejor en Additive)

Trail Material: podemos crear efectos de estela basado también en un material. Una vez asignados los materiales, podemos ajustar múltiples parámetros, como tamaños, rotaciones, si recibe las sombras, etc.

SISTEMAS DE PARTÍCULAS (parte 3)

Más apartados interesantes: los que controlan cómo se comportan en base al tiempo de vida de las partículas:



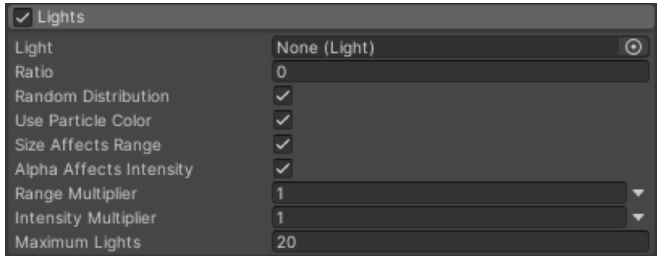
Size/Color over Lifetime: que cambie su escala/color a medida que avance el tiempo

Velocity over Lifetime: que se aceleren las partículas a medida que avanza el tiemmpo

Limit Velocity over Lifetime: lo contrario a lo anterior, las partículas se van frenando. Interesante en explosiones por ejemplo)

Color/Rotation by Speed: en lugar de cambiar el color o la rotación según el tiempo que ha pasado, lo hacen según la velocidad. Por ejemplo, una chispa de una explosión a medida que se decelera, cambia de incandescente a apagada

Lights: otro apartado interesante es el que nos permite añadir una luz a nuestro sistema de partículas, muy útil para explosiones o efectos de fuego.



Deberemos crear una luz en la escena y adjuntarla, preferiblemente creando con ella un "prefab" para poder quitarla de la escena.

En el parámetro ratio podemos indicar cuántas partículas tienen esa luz adjuntada.

Otros parámetros más complejos:

Noise: permite añadir turbulencias a las partículas.

Collision: permite que las partículas choquen con otros objetos. Nos permite ajustar la forma en la que "rebotan".

- World: chocan con cualquier objeto, pero consume muchos recursos

- Planes: le indicamos el plano sobre el que tiene que colisionar.

Sub emitter: cada partícula se convierte en un emisor. Muy útil para fuegos artificiales, o efectos como gotas de lluvia sobre el terreno.

Texture Sheet Animator: a partir de hojas de sprites podemos crear efectos interesantes 2D.

Trails: crea estelas en las partículas. Necesita un material para la estela.

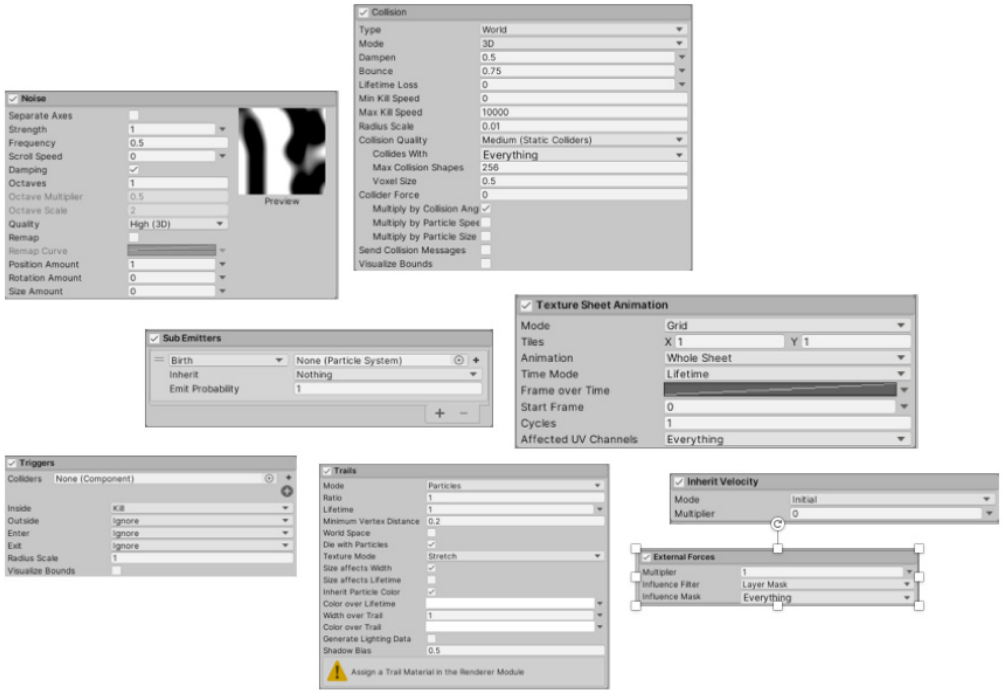
Triggers: permite indicar cómo deben comportarse al chocar con otro collider. Debe contener a su vez un componente collider. Incluso se puede vincular a un script mediante el método "OnParticleTrigger".

Inherit Velocity: el emisor de las partículas transfiere su velocidad a las partículas emitidas.

External Forces: las partículas se ven afectadas por el "Wind Zone" de nuestro juego.

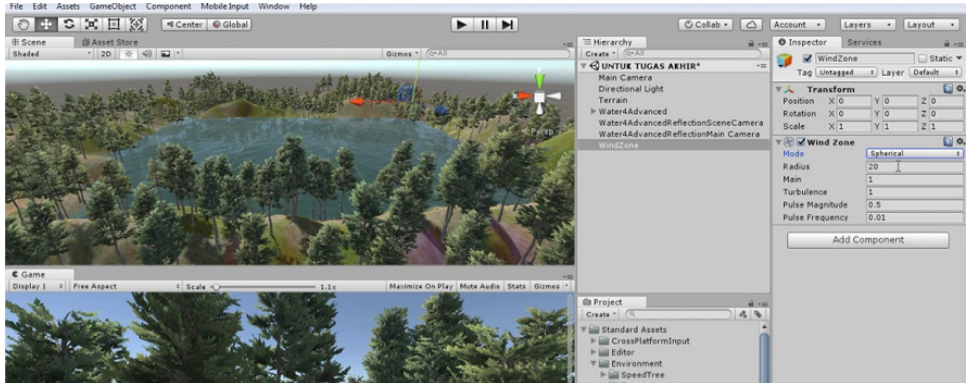
En la página siguiente verás un ejemplo:

SISTEMAS DE PARTÍCULAS (parte 4)

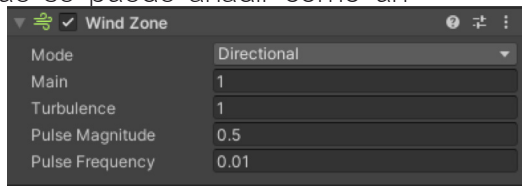


Ejemplo de paneles con su configuración para un sistema de partículas

Wind Zone



Aunque no es un sistema de partículas como tal, hablaremos aquí del Wind Zone. Se trata de un elemento que se puede añadir como un elemento más de la escena (GameObject > 3D > WindZone), o bien añadirlo como un componente más (por ejemplo, en un terreno).



Automáticamente, simulará turbulencias en nuestra escena que afectará a elementos como árboles o sistemas de partículas.

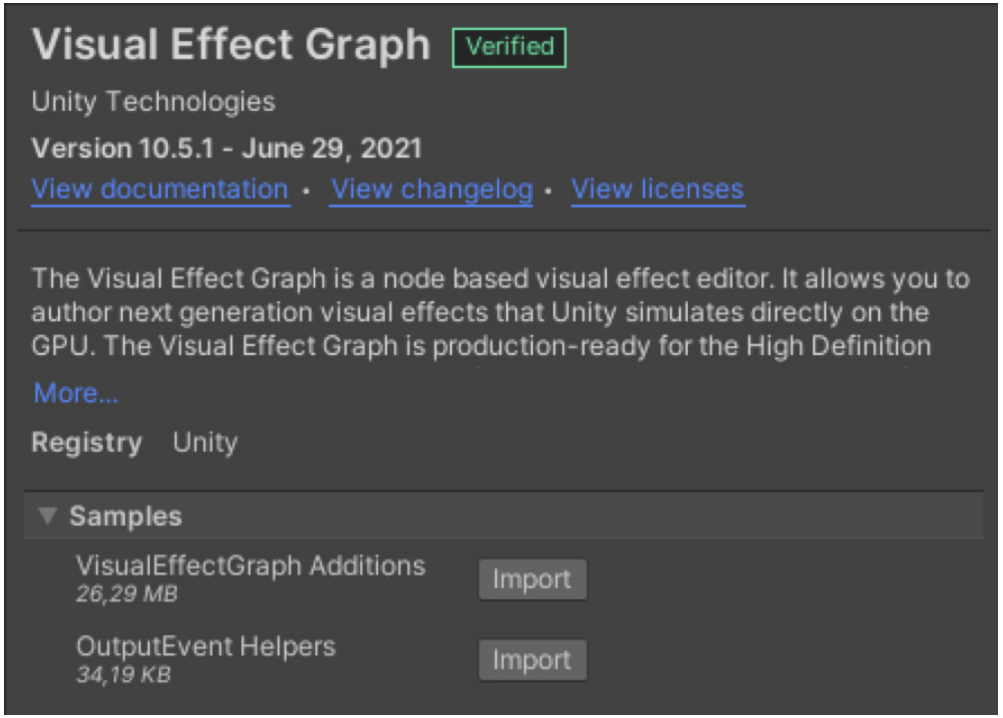
Su configuración es bastante simple:

- Modo: puede ser direccional (afecta a todo el terreno por igual, más realista) o esférico (sopla desde el centro de una esfera que definiremos por su radio)
- Main: fuerza del viento
- Turbulence: un valor más alto generará más variaciones aleatorias en la fuerza del viento
- Pulse Magnitude/Frequency: frecuencia e intensidad de las ráfagas de viento

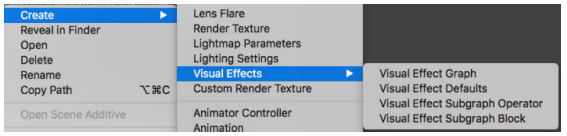
VFX

A menudo necesitamos efectos visuales en tiempo real mas complejos de los que un sistema de partículas nos puede ofrecer, como por ejemplo simulación de humos o fluidos, o emisiones complejas de chispas.

Para ello, Unity tiene un paquete que permite crear VFX que se llama "Visual Effects Graphs" y que tendremos que instalar:

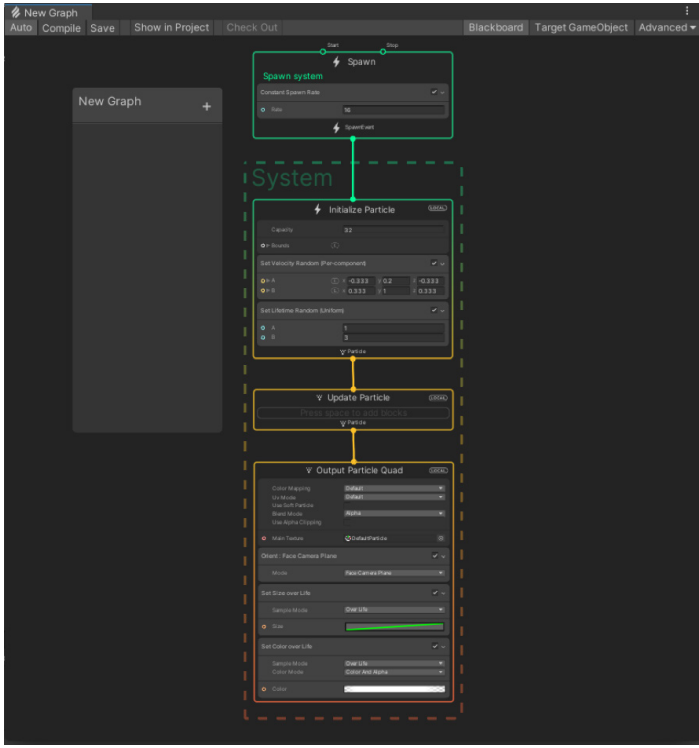


Una vez instalado, puedes crear en tu proyecto un "Visual Effects Graph".



Para editarlo, utiliza un sistema de nodos y bloques, similar al Shader Graph, mediante el cual podremos crear y editar los "assets" con efectos visuales. Puedes abrir el editor en "Window > Visual Effects > Visual Effect Graph", o bien seleccionando el asset creado y pulsar el botón "Open" (deberrás guardar el nuevo asset creado si no lo está ya).

Este es un ejemplo del panel para editar el VFX que aparece al comienzo:



Una de las grandes ventajas de este sistema de VFX frente al sistema de partículas, es que en lugar de la CPU usa principalmente la capacidad de proceso de la GPU para sus cálculos en tiempo real.

La complejidad en la creación de VFX hace que quede fuera de este tema, pero te animo a explorar más en:

https://unity.com/es/visual-effect-graph

EJERCICIO

Ya estamos preparados para dar realismo a nuestro juego de Zaxxon. Para ello, vamos a trabajar en distintos aspectos.

Materiales

Busca texturas en Internet o en la Asset Store para crear tus propios materiales con los que revestir tus obstáculos y tu nave. Si es necesario, sustitúyelos por modelos también de Internet.

Puedes incluso crear un material que desplace su textura a la velocidad que le indiques, de forma que cree la sensación de movimiento en el suelo.

Añade también un SkyBox que vaya con el estilo que has elegido para el juego.

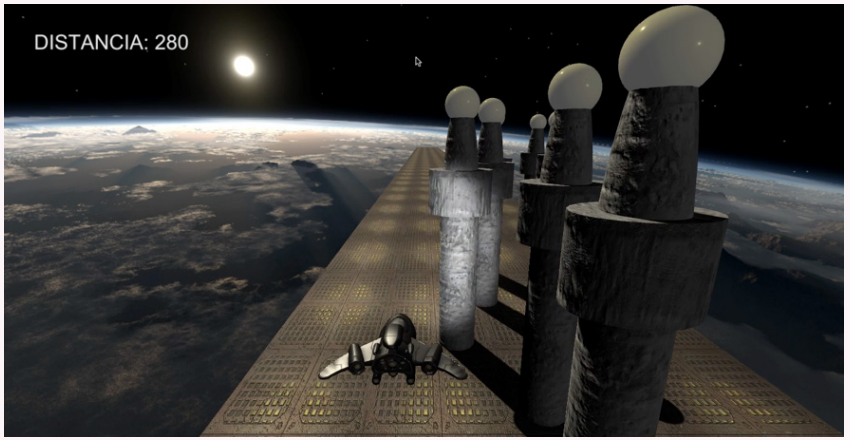
Luces

Crea luces que den el ambiente al escenario. Pueden ser en tiempo real (como faros en la nave) o estáticos para hornear, por ejemplo si creas prefabs con el escenario.

Sistemas de partículas

Añade algún tipo de efecto para dar más vistosidad al juego, ya sea creando una estela para los motores o unos proyectiles en forma de bolas de fuego.

¿Recuerdas que al chocar la nave se detenía todo? Pues busca en Internet una explosión apropiada para tu nave y allí donde se haya chocado coloca un prefab con esa explosión.



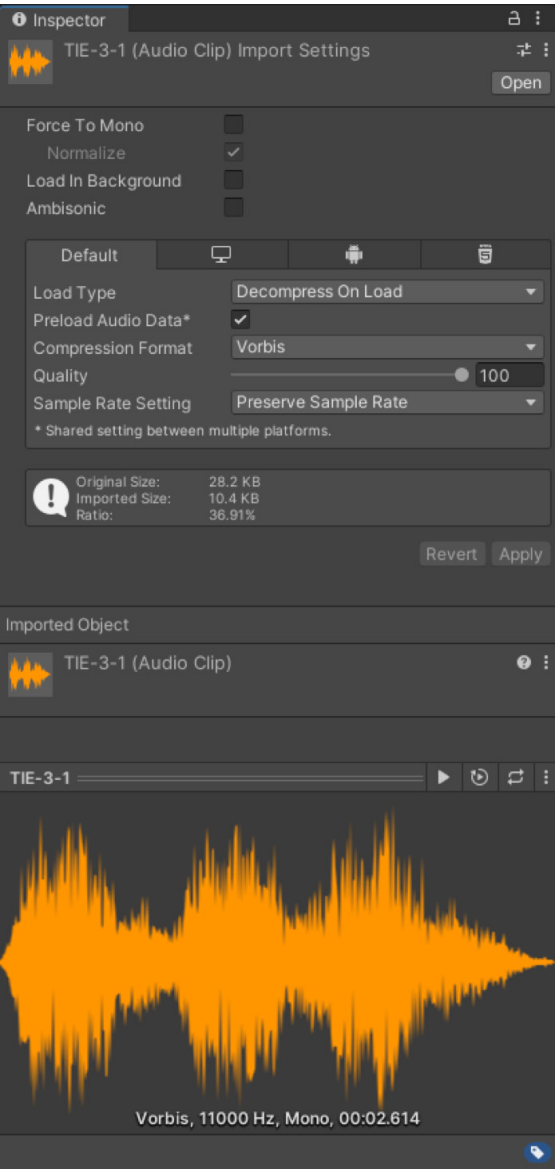
Álvaro Holguera
CIFP José Luis Garci
(Madrid)

Parte 3
SONIDO

Llegamos al último apartado de este tema: el sonido.

IMPORTANDO SONIDO

El sonido es uno de los elementos más importantes en un videojuego, ya que da realismo y profundidad a la escena.



En nuestro proyecto podemos importar los archivos de sonido (Audio clip) que necesitemos en los formatos ,aif, .wav, .mp3, y .ogg

En el inspector podemos escucharlo, o cambiar los parámetros de importación y precarga.

- **Force To Mono**: convierte un archivo estéreo o multipista en uno mono.
- **Load in background**: permite ejecutarlo de forma inmediata. Se puede usar en combinación con "Preload Audio Data".
- **Ambisonic**: para archivos de sonido especiales 360º.

A continuación tenemos la configuración que depende de la plataforma para la que se hará la Build. Como con las imágenes, esto nos permite mejorar el rendimiento.

Load type:

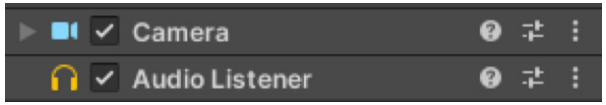
- "Decompressed On Load". Recomendable para archivos pequeños, ya que espera a cargar el archivo para descomprimirlo.
- "Compressed in Memory": Recomendable para archivos grandes de sonido, ya que tiene menor sobrecarga en el rendimiento. Pero si le pedimos que lo cargue en memoria en proyectos grandes puede sobrecargarla.
- En el caso de streaming, se guardará en el disco duro y se ejecutará desde allí. El consumo de memoria es mínimo. Útil para audios muy prolongados cuya sincronización no es crítca.

- **Preload Audio Data**: carga los archivos de audio antes de que se inicie la escena.
- **Compression format:** dependerá del tamaño del archivo y la calidad que necesite. Por ejemplo, PCM da mayor calidad pero consume más espacio, por lo que se recomienda para efectos de sonido de corta duración. Vorbis/mp3 se recomienda para efectos y música de duración mediana, y ADPCM para archivos de larga duración. Los archivos Vorbis cuesta más descomprimirlos, por lo que mejor cargarlos en memoria.
- Sample Rate Settings: algunos permiten optimizar la frecuencia de muestreo.

COMPONENTE AUDIO SOURCE

Hay que distinguir dos componentes básicos:

Audio Listener: nuestros oídos en el juego. Cada escena debe tener un solo Listener. Por defecto está añadido como componente a cada cámara que creemos.



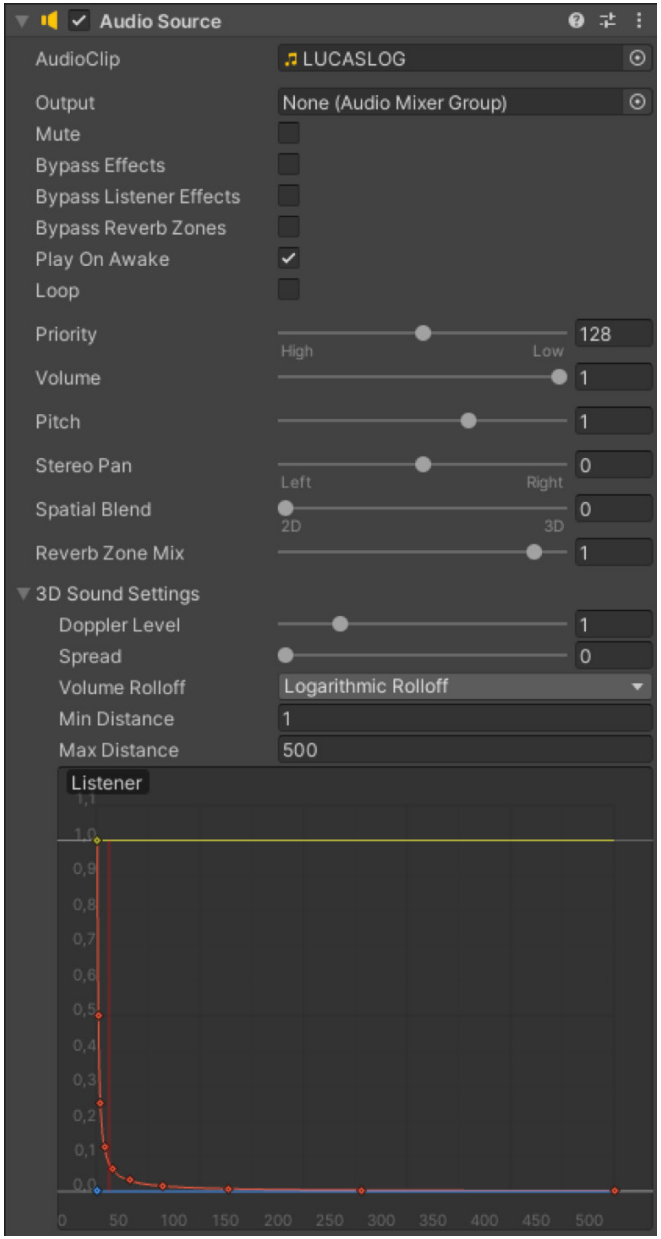
Audio Source: las fuentes de sonido.

Podemos añadirlo a cualquier GameObject (incluso a uno vacío para la música de fondo) Si arrastramos un clip de audio a un elemento de la escena, le crea automáticamente el componente. Una vez creado, insertamos el Audio Clip desde nuestro proyecto:

En el inspector podremos ajustar parámetros como el volumen, el pitch, el paneo, etc. Por supuesto, estos parámetros están disponibles desde código.

Es importante activar o desactivar la opción de Play on Awake (comenzará a sonar automáticamente), así como su reproducción en bucle (Loop).

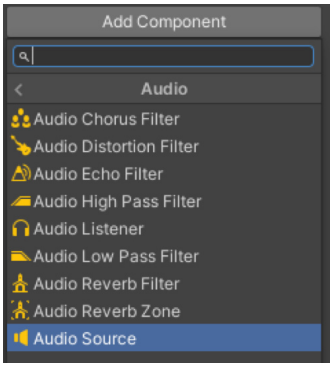
La opción de "Spatial Blend" determina si el sonido es interpretado por Unity dentro del escenario 3D. Por ejemplo, la música de fondo es mejor que esté en 2D, pero las voces de los personajes y los objetos en 3D, para que el jugador los ubique en la escena.



PROCESADORES Y SONIDO 3D

Filtros de sonido

No olvidemos que el sonido es uno de los aspectos fundamentales a la hora de hacer creíble un videojuego o cualquier producto audiovisual. Por ello, es importante conocer la naturaleza del sonido y cómo podemos actuar sobre sus parámetros para lograr el efecto psicológico desado.



Podemos añadir varios componentes de audio para controlar ese sonido, mediante Filtros de Sonido. Se pueden aplicar tanto a Audio sources como a Audio Listener, y añaden efectos o filtros de sonido. Veamos algunos ejemplos:

Filtros de sonido: se pueden aplicar tanto a Audio sources como a Audio Listener, y añaden efectos o filtros de sonido. Algunos ejemplos:

1. **EQ**: filtros pasa bajos o pasa altos, que cortan a partir de una frecuencia para eliminar ruidos indeseados o "colorear" el sonido.
2. **Eco & Coro**
3. **Distorsión**
4. **Reverb**: simula cómo el sonido se refleja en las paredes, creando ambientes de interior.

Como sabemos, la reverberación es lo que permite identificar las condiciones del espacio físico. Por ello, existen las Zonas de Reverberación, las cuales podemos añadir a un Empty Objecto, y permite crear zonas en las que los audio clips se reproducen con una reverberación específica (por ejemplo, una cueva):

Algunos efectos pueden consumir muchos recursos, algo que podemos monitorizar en el Audio Profiler Menu

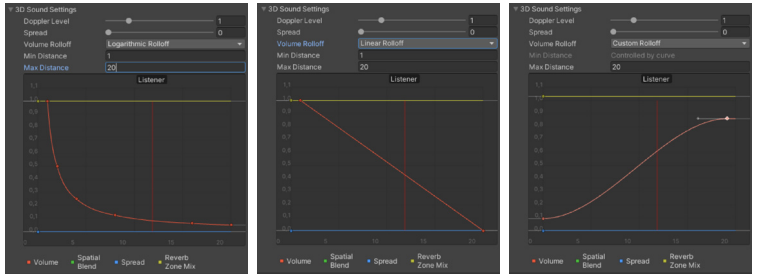
Sonido 3D

Cualquier sonido puede interactuar con el ambiente, siempre que en el "Spatial Blend" lo hayamos configurado como 3D

Parámetros que podremos configurar:

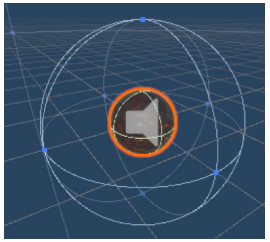
- **Doppler Level**: el efecto Doppler es el cambio de frecuencia que se produce cuando el oyente esté en movimiento relativo con respecto a la fuente del sonido
- **Spread**: cuánto se propaga el sonido en todas direcciones, medio en ángulos
- **Volume rolloff**: la forma en la que se comporta según la distancia (logarítmico, lineal o a medida), quedará reflejado en las gráficas
- **Min/Max distance**: las distancias a las que el audio se propaga

Jugando con el tipo de rolloff, y las distancias de propagación, podemos hacer que el sonido se comporte a nuestro gusto en el entorno 3D, atenuándose o aumentando según a distancia a la que se encuentre la fuente del sonido. Podemos hacer incluso que suene más fuerte cuanto más lejos está:



Ejemplos de cómo podemos cambiar el volumen en función de la distancia

Podemos también cambiar la distancia a la que se oye el sonido moviendo el Gizmo azul que aparece en el objeto



EJECUTAR SONIDO CON CÓDIGO

Como es lógico, tenemos que poder ejecutar desde un script los sonidos del juego.

Para poder hacerlo, deberemos crear una variable de tipo "AudioSource":

```
AudioSource audioSource;
```

En el método Start lo vinculamos al audio source del objeto:

```
audioSource = GetComponent<AudioSource>();
```

Cuando lo deseemos, lo reproducimos, o bien accedemos a cualquiera de los otros atributos:

```
audioSource.volumen = 1;
audioSource.Play();
```

En el siguiente ejemplo, ejecutamos un sonido cuando nos impacta una bala (por ejemplo, una explosión). Además, accedemos al Mesh Renderer para desactivarlo y así desaparecer.

```
//Cargamos el sonido de la explosión
AudioSource audioSource;
//Creamos una variable para contener el Mesh Renderer
public Renderer rend;

void Start()
{
    audioSource = GetComponent<AudioSource>();
    rend = GetComponent<Renderer>();
    //Activamos el Mesh Renderer, por si acaso
    rend.enabled = true;
}

void OnCollisionEnter(Collision collision)
{
    if (collision.gameObject.tag == "bala")
    {
        //Ejecutamos el sonido
        audioSource.Play();
        //Ocultamos el objeto
        rend.enabled = false;
        Destroy(gameObject, audioSource.clip.length);
    }
}
```

NOTA: como el sonido no puede ejecutarse si el elemento que lo contiene ha desaparecido, en su lugar deja de renderizarlo, y tras pasar el tiempo que dura el sonido lo destruye. El método Destroy admite un 2º parámetro de tipo float: el tiempo de espera antes de destruir el objeto, En este caso, directamente lo sacamos del atributo del clip, esto asegura que esperará a que termine el sonido para quitarlo de la escena.

Ejecutar varios sonidos en un mismo Audio Source.

Se pueden asociar varios sonidos a un mismo Game Object, para luego ejecutarlos de forma independiente.

Para lograrlo, usaremos la función PlayOneShot de Unity, en lugar de Play, lo que permite ejecutar dos clips de audio de forma simultánea (el método play al ejecutarse anula el anterior sonido).

Además de seguir los pasos anteriores, crearemos tantas variables públicas/serializadas como queramos, de tipo AudioClip, que en la interfaz de Unity asociaremos con los clips de audio que queramos, arrastrando directamente los archivos de audio del proyecto.

```
[SerializeField] AudioClip disparo1;
[SerializeField] AudioClip disparo2;
```

Una vez asignados los clips de audio, podemos ejecutar el método PlayOneShot a la variable de AudioSource, indicando 2 parámetros: el clip que debe sonar, y el volumen (un nº float de 0 a 1). Puedes ejecutarlo antes de que deje de sonar el clip anterior, porque se solaparán.

```
audioSource.PlayOneShot(disparo1, 0.7f);
```

RECUERDA: Si lo incluimos en un método público dentro del script, podremos acceder desde otros Game Objects para ejecutar el sonido.

EJERCICIO

¿Te acuerdas cuando hicimos botar una pelota mediante animaciones? Vamos ahora a añadirle el sonido, pero asegurándonos de que suena siempre que la pelota toca el suelo.

Pelota botando

Es un buen momento para recordar que podemos ejecutar scripts desde nuestro panel de animaciones de Unity. Usar eventos para que suene un sonido en una animación es realmente sencillo

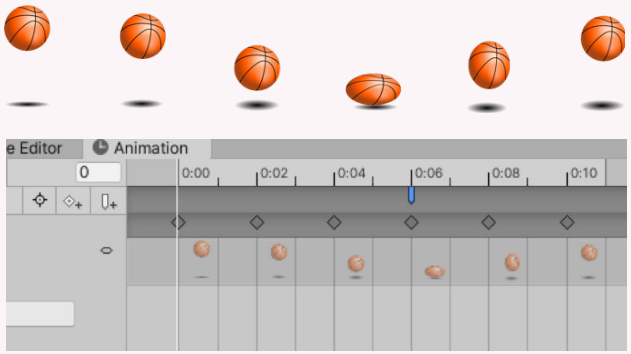
¿Recuerdas la animación de una pelota que hicimos unos temas más atrás? Es un buen momento para añadir un sonido de bote.

Si al GameObject que contiene la animación añadimos un script con métodos públicos, estos estarán disponibles al crear el evento.

Recuerda, no es necesario añadir el clip de audio al AudioSource en Unity ya que lo añadiremos mediante código (esto nos permitirá crear múltiples funciones, cada una llamando a un sonido diferente).

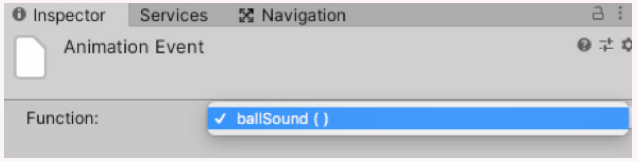
Acuérdate de desactivar el Play On Awake

Esto lo podemos hacer también con una animación de Sprites:



A continuación, hemos creado un script con la variable pública del Audio Source, y la del clip que queremos ejecutar (pueden ser varias, cada una vinculada a una función distinta)

Finalmente, al crear un evento en la animación, justo en el momento que la pelota toca el suelo, le decimos que ejecute la función correspondiente.



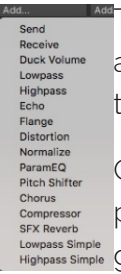

```
public void ballSound()
{
    audioSource.clip = audioClip;
    audioSource.Play();
}
```

AUDIO MIXER

Cuando tenemos varios sonidos en la escena, podemos enviarlos a una mesa de mezclas virtual, en la que podremos controlar sus niveles y aplicar efectos en tiempo real para controlar mejor el resultado.

En el menú "Window->Audio" podemos abrir la ventana de Audio Mixer (o seleccionando la mesa y pulsando el botón "Open". Se abrirá el panel con la mesa:

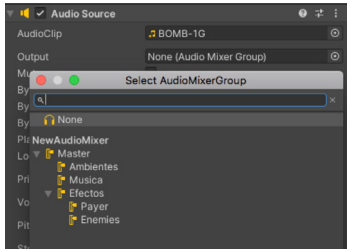
Podemos añadir grupos y subgrupos (1) para poder controlar el volumen (2) de múltiples sonidos a la vez

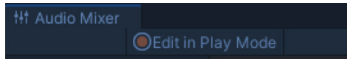


También podemos añadir uno o varios efectos (3) a cada grupo, lo que permite por ejemplo que todos los efectos tengan la misma reverb.

Cuando creamos un componente AudioSource, podemos indicar que lo envía a uno de esos grupos y así controlarlo a través de la mesa de sonido, como se muestra a continuación:

Si ejecutamos el juego, podemos monitorizar y ajustar los niveles en tiempo real, viendo los vúmetros, o usando los botones (4) para mutear o poner en SOLO los grupos (la B es para desactivar los efectos), y si activamos la opción de "Edit in Play Mode" que aparece durante la ejecución, los cambios se guardarán.





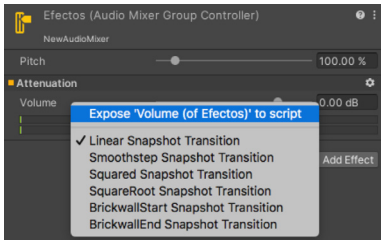
Controlar parámetros de la mesa mediante código

Podemos controlar el volumen de los grupos (o incluso del máster) de la mesa de sonido mediante código, para ello tenemos que añadir la librería "UnityEngine.Audio", crear una variable AudioMixer a la que arrastraremos la mesa.

```
[SerializeField] AudioMixer mixer;
```

Para poder acceder a cada grupo, debemos añadirlo a la lista de "Exposed Parameters" que aparecen en la esquina superior derecha de la mesa (5). Para ello, seleccionamos el grupo y en el panel Inspector, hacemos click con el botón derecho donde pone Volume, y hacemos click en "Expose Volume... to Script"

Ahora aparecerá en la lista de Exposed Parameters, donde podemos cambiarle el nombre:



Ya podemos usar el método "SetFloat" de la mesa de sonido, indicando el fader al que nos dirigimos mediante el nombre indicado:


```
mixer.SetFloat("MyExposedParam", 10);
```

IMPORTANTE: el volumen de la mesa de sonido, como buena mesa, se mide en decibelios (dBs), donde 0dBs es el volumen normal, -80dBs es silencio, y 20dBs es el máximo volumen. Y por si no lo sabes, el decibelio es una unidad logarítmica, así que si vinculas un Slider, que es lineal, al volumen de la mesa, deberás hacer un ajuste matemático:

```
dB = 20.0f * Mathf.Log10(lineal);
```

(aquí "lineal" es el valor del sliider)

Según esta formula, el valor lineal de "1" daría 0dBs,ya que el logaritmo en base 10 de 1 es cero. Y un valor lineal de 10 daría 20dBs (el máximo), ya que el logaritmo de 10 es 1 multiplicado por 20.

ENLACES A VÍDEOS y MATERIALES

Aquí podrás acceder a los vídeos que explican y/o profundizan sobre los conceptos vistos, por si te son de utilidad.

Materiales y luces

Uso básico de los materiales

https://www.youtube.com/watch?v=CADj5Lmyj2s

Uso básico de luces

https://www.youtube.com/watch?v=L_e4FTUdQsI

Bakeado de luces

https://www.youtube.com/watch?v=okYhs6kQ0xw

Sonido

https://www.youtube.com/watch?v=_QFcPsj5S1I

Sistemas de partículas

1.- Repaso a los parámetros disponibles en el sistema de partículas

https://youtu.be/FEA1wTMJAR0

2. Ejemplo de cómo crear una bola de fuego a partir de los Standard Assets

https://www.youtube.com/watch?v=OWShSR6Tr50

3.- Crear un "fire spells" a partir de imágenes creadas en photoshop

https://www.youtube.com/watch?v=-P09r-ALN38