# Delivery 2

**Miembros del grupo:** Félix Fernandez-Palacios y Adrián Fuster Lorca

**Tabla de ejercicios:**

| | |
|---|---|
| **Post Processing** | |
| Vignetting and Pixelate | **1pt** |
| **Shader in Unity** | |
| Adding PBR Shadow | **1pt** |
| Create Materials PBR | **2pt** |
| **Compute Shaders** | |
| Boids Implementation | **2pt** |
| **Additional Implementations** | |
| Vertex Shader Animation | **1pt** |
| Texture Animation | **1pt** |
| **Rogue Exercise** | |
| Triplanar Textures | **1pt** |

## Vignetting and Pixelate:

**Proyecto:** Assets/PostProcessing
**Escena:** Si subes la main cámara hacia arriba, comenzarán ambos efectos, simulando la pérdida de oxígeno con la altitud. Tienes que subir la Y de la main camera > 25 para poder observar el cambio. En caso de no estar activo, hay que ir a la jerarquía del proyecto y en el PostPro añadir estos post procesados custom. Utilizamos los dos post procesos de la práctica anterior.

## PBR SHADOW

**Proyecto:** Assets/Shader In Unity/ PBR_Shadow
**Escena:** Lo encuentras en la cabaña de la izquierda si miras de frente la montaña. Podrás observar 2 antorchas al lado de la misma. En la roca el material creado recibe la sombra de la antorcha como se muestra en la siguiente imagen.


Utilizando y corrigiendo el PBR de la práctica anterior hemos cambiado ciertas cosas para crear un PBR_Shadow que recibe sombras pero no las emite. Hemos pasado de usar CGPROGRAM en el shader a utilizar HLSLPROGRAM además de varios includes nuevos y realizamos los cálculos de luz de forma distinta que en el PBR normal.

## Create Materials PBR

**Proyecto:** Assets/Shader In Unity/ PBR -> Materials / Textures
**Escena:** Hemos añadido a todos los elementos de las escena materiales creados a partir de nuestro PBR el cual hemos modificado de la entrega anterior. Tanto los materiales finales como las texturas que utilizamos para crearlos se encuentran en las carpetas Assets/Shader In Unity/Materiales y Assets/Shader In Unity/Textures.

## Boids Implementation

**Proyecto:** Assets/Compute Shaders/ Boid
**Escena:** En un gameobject vacío llamado Boid, el cual utilizamos como eje de rotación para crear el movimiento, añadimos otro gameobject vacío al cual le añadimos el script de boid. A este le pasamos el compute shader, el pez que utilizaremos para crear el boid y ciertos parámetros de configuración. La intención es tratar de simular un banco de peces que da vueltas a la isla.

Se ha implementado el boid siguiendo las tres reglas fundamentales del boid: separación, alineación y cohesión. Para el compute shader hemos utilizado el persistent ya que los peces no van a ser modificados, lo que hace que sea mucho más eficiente que el simple al ahorrar muchos cálculos por iteración.

No nos ha dado tiempo a hacer que los peces detectan colisiones utilizando Raycast ya que nos estaba dando errores y hemos decidido suprimirlo.

## Vertex Shader Animation

**Proyecto:** Assets/Additional Implementation/Vertex Animation
**Escena:** Hemos hecho una serie de peces y un tiburón que simulan que están nadando dentro del agua. Para crear esto hay que hacer un **lerp** entre la posición del pez y su posición animada, para conseguir la posición utilizamos el nodo **position** y ponemos en Space “Object”. Para la animada cogemos la posición igual que antes y la dividimos para sacar los canales R,G,B. Al R le añadimos el nodo **add** y en el otro parámetro le añadimos el tiempo multiplicado por un vector para sacar la velocidad y por otro lado cogemos la posición del mundo y según el eje donde lo queramos que se tambalee accedemos a un canal u otro. Estos dos los unimos por un **add**, que van hacia una función de **seno** para crear la ilusión de movimiento ondulante y multiplicado por un escalar controlamos esto. Esto se unirá al **add** que iba en un principio al canal R.

## Texture Shader Animation

**Proyecto:** Assets/Additional Implementation/Texture Animation
**Escena:** Hemos intentado recrear el mar con las olas llegando a la orilla de la isla. Para ello hemos implementado un texture animation utilizando shader graph. A parte de esto hemos activado el depth texture del URP para crear la sensación de profundidad en los objetos que se encuentran debajo del agua.

**Análisis Shader Graph:**

Lo primero que hacemos es distorsionar una textura con **gradient noise** para generar el efecto de movimiento. Utilizando el **gradient noise** y **UV** y sumándolos generamos distorsión en los UV. Luego esta operación se divide para generar distintos niveles de distorsión con el nodo **divide**. El nodo recibe como segundo parámetro un slider que creamos entre 1 y 10 para ajustar la distorsión. Este es accesible desde el inspector en la escena. Después conectamos el nodo **tiling and offset** por su parámetro de entrada Offset conectándolo con el resultado de la división anterior. El siguiente paso es añadir movimiento a la distorsión que hemos creado. Para ello conectamos un **vector2D** a los UV de gradient noise y con un nodo **position**, en world space, utilizando el out hacemos un **split** y conectamos al vector2D la coordenada X y la coordenada Z sumada a la operación que se explica a continuación. Para generar la animación utilizamos el nodo **Time** y ayudándonos de un valor que podremos modificar en la jerarquía, variando la velocidad con la que se mueve nuestra textura.

Ahora creamos un nuevo **gradient noise** para generar un efecto de brillo. A este gradient noise le sumamos un **normal vector** y el resultado de la operación lo pasamos como primer parámetro de un **dot product**. Como segundo valor, añadimos un **vector3D** con el valor de Y en 1 para que la iluminación apunte siempre hacia arriba. Después creamos dos nodos **smoothstep** que serán los que nos permitan endurecer el efecto del brillo en la cascada a la vez que dejamos un degradado en la orilla. El resultado de **dot product** lo añadimos como input del primer smoothstep y como primer edge del segundo. Esto lo hacemos porque queremos cambiar la dureza del brillo ya que luego vamos a multiplicar los nodos y obtener la diferencia entre ambos. Ahora sumamos esta operación a la textura principal.

Por último utilizamos el depth texture para añadir profundidad a los elementos que están por debajo del agua como hemos mencionado anteriormente y crearemos una animación para el brillo. Para ello creamos un nuevo nodo **gradient noise** el cual utilizaremos para deformar el offset de los UV en el gradient noise que genera el brillo. Creamos un nodo **UV** y lo sumamos con el gradient noise anterior y como antes dividimos esta operación. Añadimos un nodo **Tilling and Offset** y utilizamos el resultado de la división para añadirlo al offset de este nodo y conectamos el output con el gradient noise del brillo. A continuación copiamos lo que hicimos anteriormente de animación y se lo colocamos al nuevo gradiant noise en la UV. Para el depth texture tendremos que poner el nodo master en transparente.

Lo primero que hacemos es utilizar el nodo **Screen Position** para acceder a los vértices en relación a la pantalla. Ponemos la configuración en **RAW** para que no divida la componente W. Después lo conectamos con el nodo **Split** y hacemos un **Subtract** de **Scene Depth** (el cual tiene que estar configurado en **eye**) y el output **A** del split. El output de esta operación lo pasamos a un nodo **One Minus** para cambiar su signo. Añadimos un nuevo nodo **Smoothstep** para disminuir la saturación y el resultado de este lo añadimos a un nodo **Blend,** conectamos el add anterior con el subtract recién creado y el resultado lo conectamos con el master. El blend debe de estar configurado en modo **Screen**.

## Triplanar Texture

**Proyecto:** Assets/Additional Implementation/Triplanar Textures
**Escena:** Se ha implementado en la geometría del lugar para que el terreno tenga 3 texturas según la altura, césped tierra y rocas. Para empezar necesitamos un nodo **position** con space object, a continuación el nodo **Split** para sacar los canales y creamos tres **Vector2** para unirlos y que estén todos unidos entre ellos, el output de cada uno irá a cada una de las 3 texturas. Crearemos un nodo **Normal vector** con space Object, lo unimos al nodo **Absolute** y después **Power** después de esto a **Normalice,** una vez que tenemos esto utilizamos **Split**, entonces utilizaremos **Multiply** para multiplicar el output de la textura con el canal el cual no haya utilizado, es decir el primero que tiene X e Y lo multiplicaremos por B, así con los 3, después usamos **Add** para unir los tres, el resultado de esto irá al base o albedo del nodo **Master**