UNITY VIRTUAL REALITY PROJECTS - CAPITULO 9
USING ALL 360 DEGREES: Usando todos los 360 grados

- Rodríguez Contreras, Paulo Cesar **CUI**: 20142373
- Vilca Quico, Brigitte Roxana **CUI**: 20143361

Introducción:

Las fotos y videos de 360 grados son formas diferentes de usar la realidad virtual a las que los consumidores pueden acceder hoy en día, tanto en términos de experiencia como de producción y publicación. Ver imágenes pregrabadas requiere mucho menos poder de cómputo que renderizar escenas en 3D, y esto funciona muy bien en dispositivos móviles de realidad virtual. En este capítulo, exploraremos los siguientes temas:

- Uso de texturas para ver globos, panoramas y esferas de fotos
- ¿Entender qué es un medio de 360 grados?
- Agregar una fotosfera y un video de 360 grados a sus proyectos de Unity
- ¿Qué es un campo de visión y cómo se puede grabar un video de 360 grados?

En general, el término "360 grados" se refiere a la visualización de fotos o videos pregrabados de una manera que le permite girar la dirección de su vista para revelar contenido que estaba justo fuera de su campo de visión.

Los medios de 360 grados sin VR se han vuelto relativamente comunes. Algunos ejemplos de casos alrededor del mundo que usan esta tecnología son:

- ✓ Sitios de listados de bienes raíces ofrecen recorridos panorámicos con un reproductor basado en la web que le permite desplazarse de forma interactiva para ver el espacio.
- ✓ **YouTube** admite la carga y reproducción de videos de 360 grados y proporciona un reproductor con controles interactivos para mirar durante la reproducción.
- ✓ **Google Maps** le permite cargar imágenes de fotosfera de 360 grados, como su herramienta Street View

Con un auricular VR, la visualización de medios de 360 grados es sorprendentemente inmersiva, en este capítulo trabajaremos un claro ejemplo, tú estás parado en el centro de una esfera con una imagen proyectada en la superficie interior, pero sientes que realmente estás allí en la escena capturada.

Simplemente se puede girar la cabeza para mirar alrededor. Es una de esas cosas que hace que las personas se interesen en la realidad virtual la primera vez que la ven, y es una aplicación popular para Google Cardboard y GearVR. En este capítulo, exploraremos una variedad de usos de los medios que aprovechan la capacidad de Unity y VR para usar todos los 360 grados.

Desarrollo:

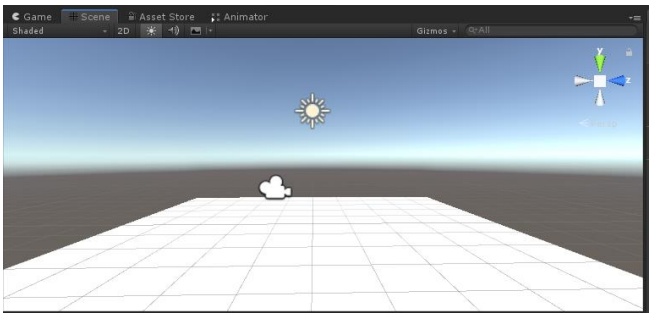
Para lograr todos los objetivos y conocer de manera más detallada cómo trabajar los 360 grados en Realidad Virtual, desarrollaremos nuevas escenas que nos ayudarán a distinguir cada una de estas propiedades, son las siguientes:

1) **360Degrees**: Aplicar una imagen rectangular regular como una textura a una esfera.
2) **MagicalOrbs**: Ver la esfera desde el interior, mapeando una imagen ordinaria en su superficie interior.
3) **Panorama**: Mapear una imagen grande(panoramica) en un cilindro.
4) **InfoGraphics**: Infografías gigantes.
5) **Globes**: Proyectar el globo esférico en un gráfico tridimensional.
6) **PhotoSpheres**: Foto Esferas de 360 grados.

360 Degrees:

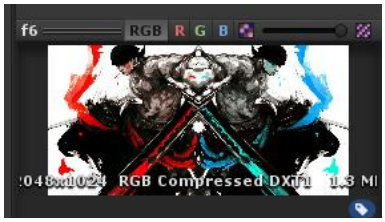
En esta escena aplicaremos una imagen rectangular regular como una textura a una esfera. Realizaremos los siguientes pasos:

a. Creamos una nueva escena
b. Creamos un nuevo plano y lo ajustamos a sus valores predeterminados
c. Insertamos el *prefab* **MeMyselfEye**, y lo colocamos en la posición (0,1,-1)
d. Finalmente eliminamos la cámara principal predeterminada



Seguidamente crearemos una esfera y le añadiremos un *script* para que gire, mediante los siguientes pasos:

a. Crear una nueva esfera y restablecer valores predeterminados
b. Establecer su posición en (0,1.5,0)
c. Arrastrar la textura que deseemos, en nuestro caso una imagen

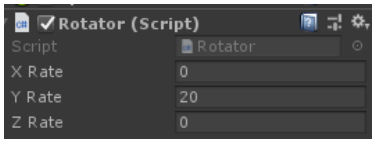


d. Ir a Agregar Componente y crear un nuevo script llamado Rotator

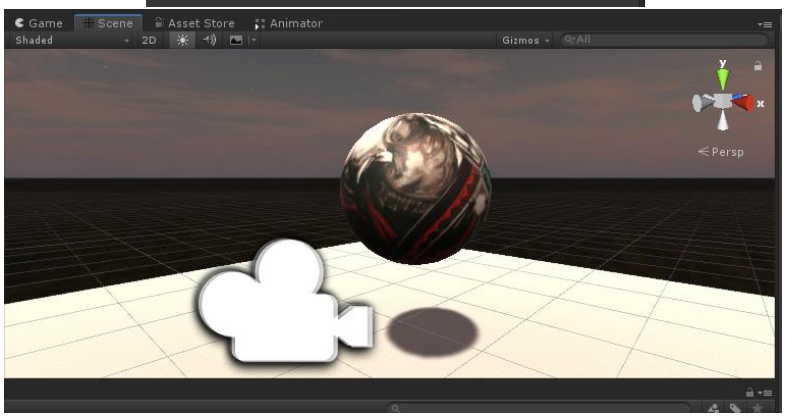
```
using UnityEngine;
using System.Collections;
public class Rotator : MonoBehaviour {
        public float xRate = 0f; // degrees per second
        public float yRate = 0f;
        public float zRate = 0f;
void Update () {
        transform.Rotate (new Vector3 (xRate, yRate, zRate) *
        Time.deltaTime);
}
}
```

Explicación: Este script usa como paquetes *UnityEngine* y *System.Collection*. Ambos paquetes permiten la configuración a través de la programación para los objetos creados en Unity. La clase *Rotator* (la cual crearemos) derivara de la clase general de todo Unity la cual es *MonoBehaviour* (comportamiento de los objetos). En este script, se define los grados por el cual rotará tenemos *xRate*, *yRate* y *zRate* todos estos incializados en 0. Esta clase, solo cuenta con un método *update*(). Este método es el encargado de la actualización por *frame* para el *GameObject*. En este caso, realizará el llamado para la rotación con las coordenadas que reciba la clase.

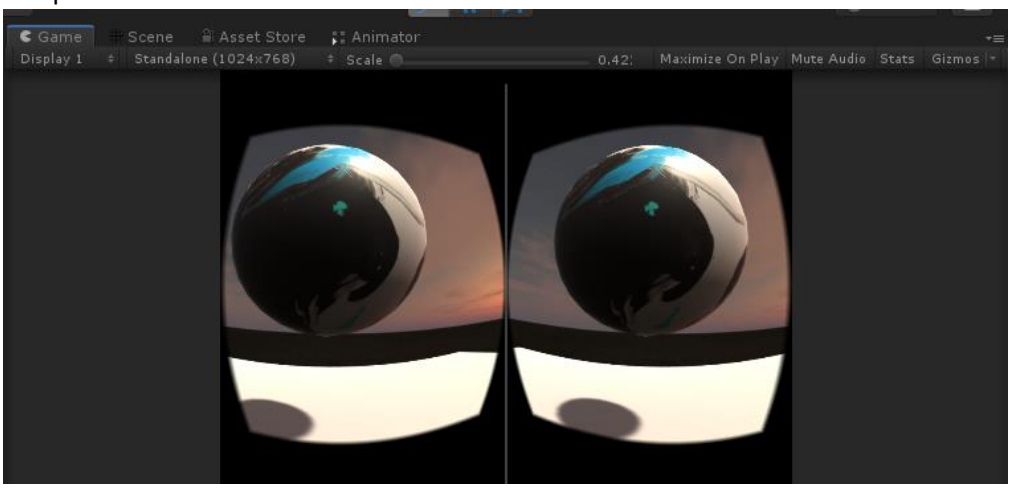
e. Finalmente establecemos la velocidad de rotación en el componente ***Rotator Script***, guardamos y probamos en RV.
El componente ***Rotator Script*** define la velocidad de rotación en el eje que corresponda, ya sea X, Y o Z.



Nuestra escena quedaría de la siguiente manera:



Y al probarlo en Realidad Virtual:



Observaremos la esfera girando, nada más que eso.

Magical Orbs:

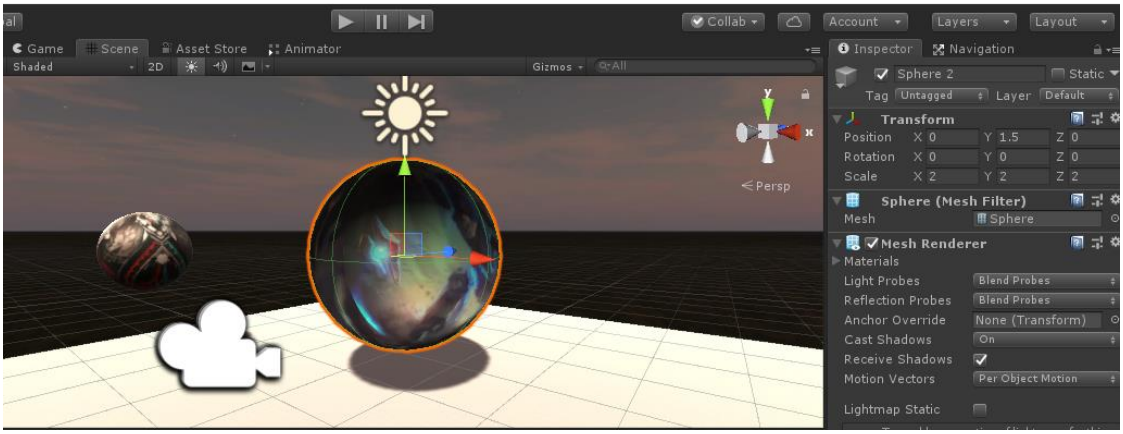
Para el siguiente ejemplo, veremos la esfera desde el interior, mapeando una imagen ordinaria en su superficie interior. Luego, pondremos una capa de color sólido alrededor del exterior. Realizaremos los siguientes pasos:

a. Creamos una nueva escena y movemos la esfera anterior a un costado, cambiando su posición en (0, 1.5, 0)

b. Crear un nuevo material al cual le añadiremos la textura que deseemos en nuestro caso será:

c. Añadir el material que hemos creado a la esfera

En este punto tendremos esto, adicionalmente aumentamos la *escala* de la esfera, al doble para una mejor visualización del efecto a continuación:



Lo que realizaremos a continuación es crear una *textura* que se renderice por el interior de la imagen, para ello aplicaremos un nuevo concepto llamado ***shaders***.

a. Crearemos un nuevo sombreado personalizado: Assets|Create|Shader

b. Abriremos el archivo para modificarlo con el siguiente contenido:

```
Shader "Custom/InwardShader" {
Properties {
_MainTex ("Albedo (RGB)", 2D) = "white"
{}
}
SubShader {
Tags { "RenderType"="Opaque" }
LOD 200
Cull Front
CGPROGRAM
#pragma surface surf Standard vertex:vert
void vert(inout appdata_full v) {
v.normal.xyz = v.normal * -1; }
sampler2D _MainTex;
struct Input {
float2 uv_MainTex;
void surf (Input IN, inout SurfaceOutputStandard o) {
fixed4 c = tex2D (_MainTex, IN.uv_MainTex);
o.Albedo = c.rgb;
}
ENDCG
}
FallBack "Diffuse"
}}
sampler2D _MainTex;
struct Input {
float2 uv_MainTex;
};
void surf (Input IN, inout SurfaceOutputStandard o) {
fixed4 c = tex2D (_MainTex, IN.uv_MainTex);
o.Albedo = c.rgb;
}
ENDCG
}
FallBack "Diffuse"
}
```

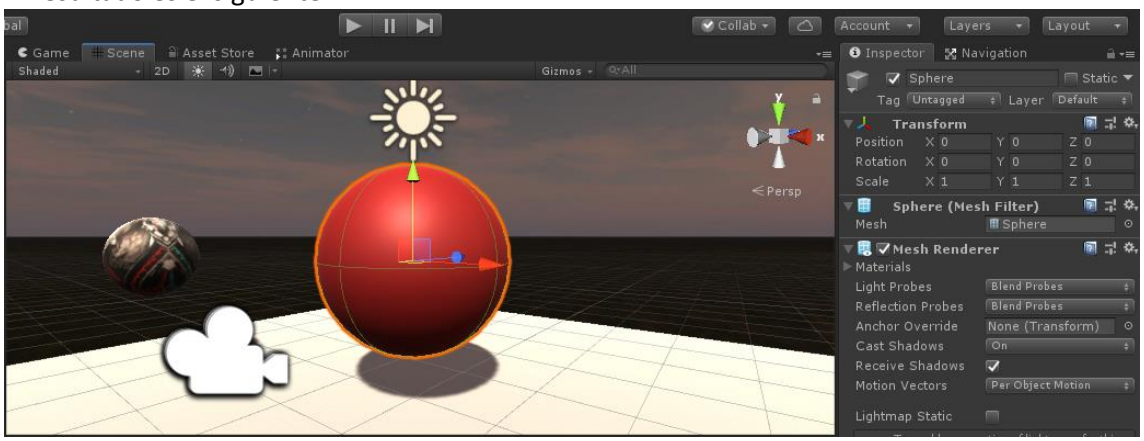
Un ***shader*** de Unity es un archivo script de texto con directivas para la renderización personalizada de Unity.

Explicación: Para entender las características de los *shaders*, debemos saber sus componentes. Los *shaders* se dividen en propiedades y *subshaders*. El primero es el encargado de las adiciones en las propiedades de colores o vectores. En este caso, define las propiedades para el color Albedo que viene predefinido en Unity. Los *subshaders* se encargan de los detalles de las propiedades principales del *shader*. En este caso nos indica el tipo de renderización y cuando deberá ejecutarse.

Finalmente, para probar el ***shader*** haremos lo siguiente:

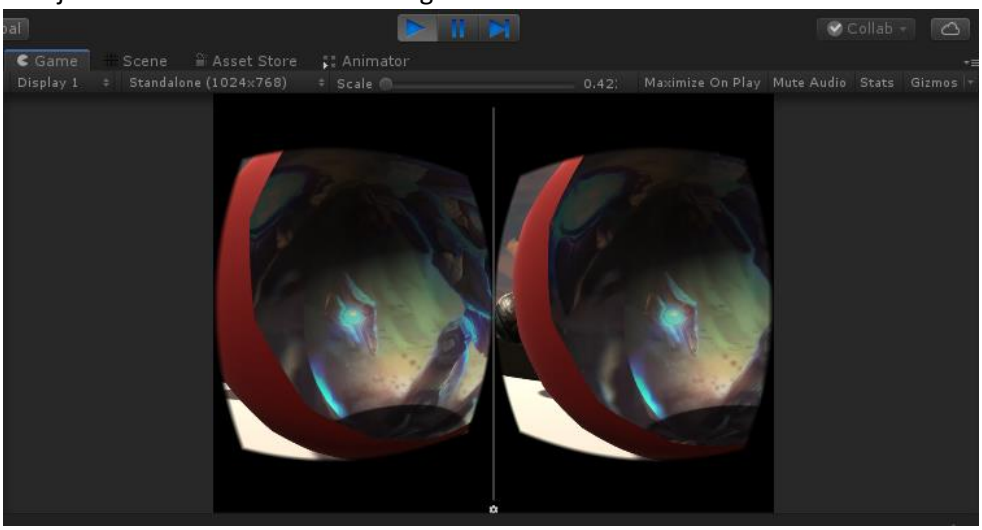
a. Seleccionar la esfera y crear una esfera hija de la esfera en cuestión
b. Deshabilitar el Componente *Collider*
c. Encontrar el material de color Rojo que hemos realizado en anteriores capítulos.

El resultado es el siguiente:



Como dato adicional deberemos añadir el Script "*HeadLookWalk*" que trabajamos anteriormente para que se pueda producir movimiento dirigido por la cámara.

Y al probar en VR, nos metemos al interior de la esfera y podemos observar como al ingresar a la esfera roja nos encontramos con la imagen en su interior:



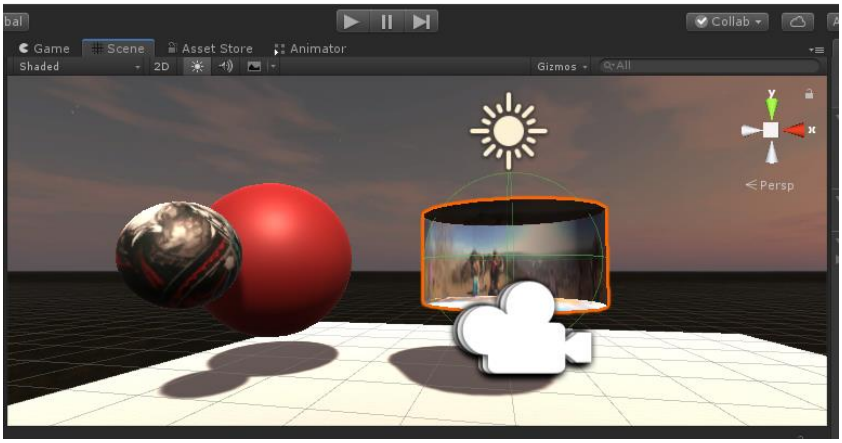
Panorama:

¿Qué sucede si la imagen es realmente amplia, como una foto panorámica que puede tomar con su teléfono? Mapearemos una imagen panorámica en un cilindro de Unity.
Usaremos la imagen que nos proporciona el libro de *Hollywood* tal y como nos cuenta el autor:
Realizaremos los siguientes pasos:

a. De igual manera a la escena anterior, moveremos los objetos ya creados a otra posición para que se visualice de mejor manera.
b. Creamos un nuevo *cilindro* y establecemos su posición en (0, 1.5, 0) y su escala en (2, 0.5, 2)
c. Desactivamos su Capsule *Collider*
d. Arrastramos la textura de *Hollywood* sobre el cilindro

e. Y le añadimos el ***shader*** que creamos anteriormente: Shader|Personalizado|*InwardShader*

Como resultado tendremos lo siguiente:



Y si probamos en RV:

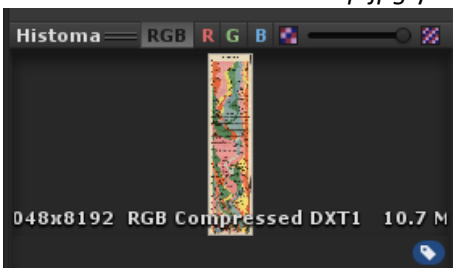
Debemos caminar hacia el centro del cilindro y podremos observar con el cursor toda la imagen solamente girando la cámara.

Obtendremos imágenes parecidas a esta, y podremos observar toda la imagen.

InfoGraphics:

Esta escena se trata de infografía, una infografía es un diagrama o diagrama visual que se utiliza para representar información o datos. Estas pueden ser bastante grandes, lo que dificulta su visualización en la pantalla de una computadora de escritorio, y mucho menos en un teléfono móvil. Sin embargo, tenemos un espacio virtualmente infinito para trabajar en la realidad virtual. Realizaremos los siguientes pasos:

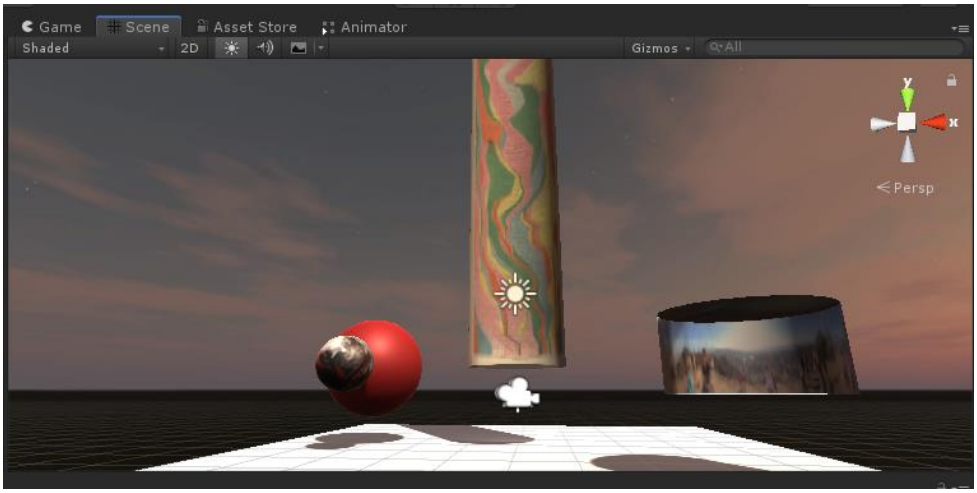
a. Creamos una nueva escena, y movemos el cilindro anterior.
b. Creamos un nuevo cilindro, esta vez en la posición (0, 11.5, 0) y escala (2, 10, 2)
c. Deshabilitamos su *Capsule Collider*
d. Importamos la imagen que viene con el libro *HistoMap.jpg* y la arrastramos al cilindro



e. Seleccionamos el componente cilindro y establecemos su *Tiling* X = -1

f. Finalmente, también le añadiremos un ***shader***: Shader|Personalizado|*InwardShader*

La escena queda de la siguiente manera:



Para completar este proyecto, debemos ofrecer al usuario una forma de moverse hacia arriba y hacia abajo a través del tubo de infografía. Para simplificar, moveremos ***MeMyselfEye*** en primera persona al centro del tubo y escribiremos un sencillo script de búsqueda de la cabeza, de la siguiente manera:

a. Seleccionar *MeMyselfEye* y resetear sus valores predeterminados
b. Creamos un nuevo script llamado "***HeadLookUpDown***" el cual describimos a continuación:

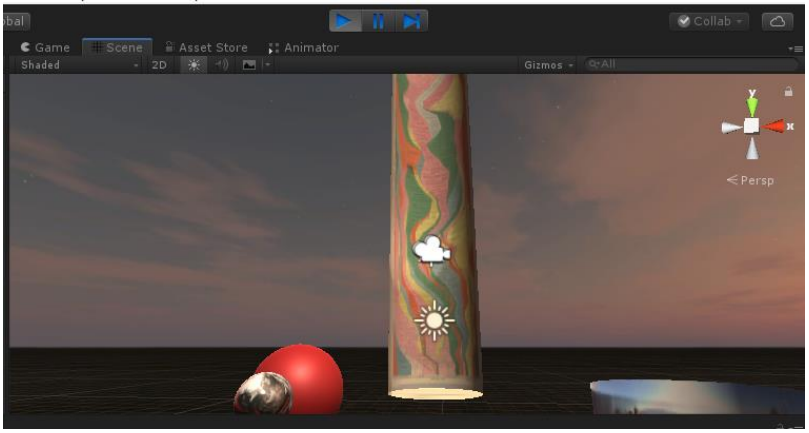
```
using UnityEngine;
using System.Collections;
public class HeadLookUpDown : MonoBehaviour {
public float velocity = 0.7f;

public float maxHeight = 20f;
void Update () {
        float moveY = Camera.main.transform.forward.y * velocity *
        Time.deltaTime;
        float newY = transform.position.y + moveY;
        if (newY >= 0f && newY < maxHeight) {
        transform.position = new Vector3 (transform.position.x,
        newY, transform.position.z);
        }
```

Explicación: Este script usa una vez más los paquetes de *UnityEngine* y *System.Collections*. Nuestra clase *HeadLookUpDown* derivará de la clase Unity

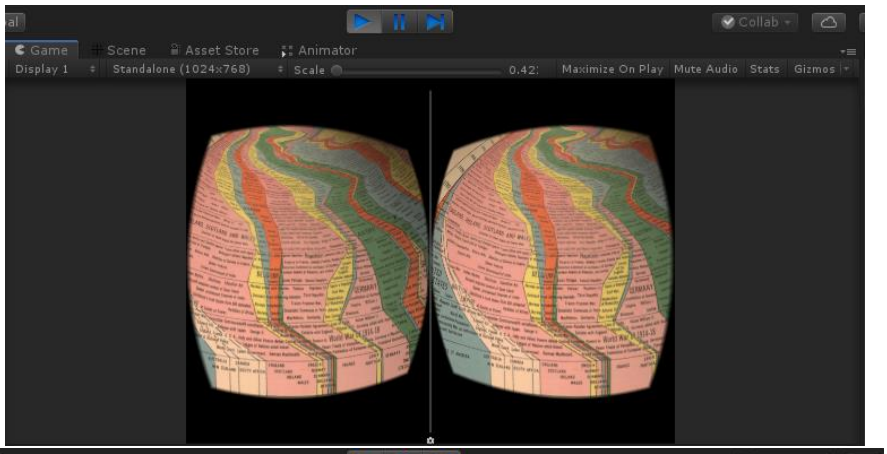
MonoBehaviour. Este script define una variable de velocidad y la altura máxima. El método *update*(), se encargara de actualizar el movimiento con las dos variables creadas. Se moverá con la velocidad indicada y como máximo hasta la altura dada.

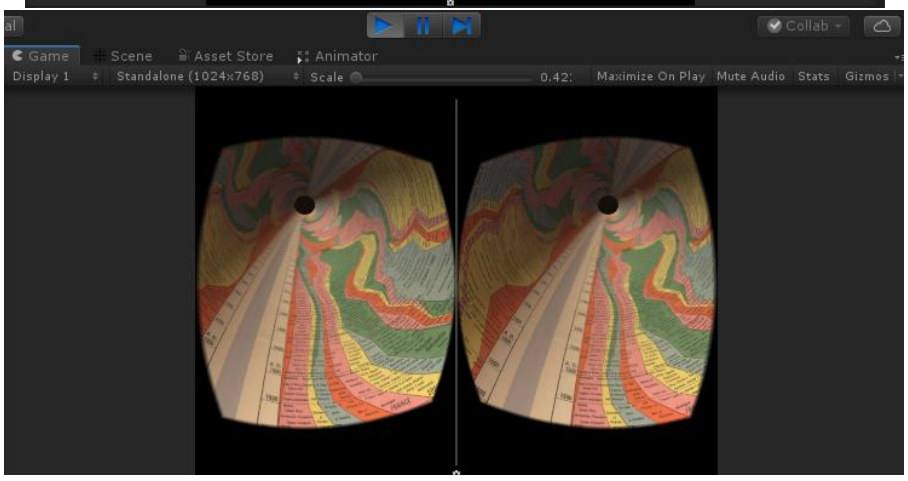
c. Finalmente, con el *script* que hemos creado y luego de realizar unos ajustes de iluminación procedemos a probar la escena en VR:

Como vemos la cámara está elevándose con el fin de observar la infografía.



Al estar en el centro del cilindro se tiene una buena posición para observar todo el texto moviendo sólo la cámara, en el interior del Cilindro sería:

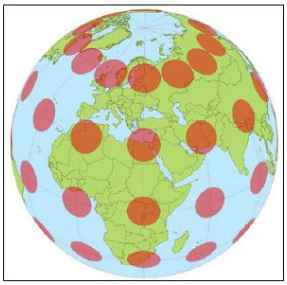




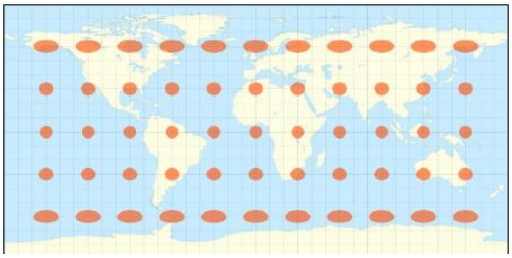
Obviamente la subida que se realiza tiene un límite el cual es el tamaño o altura del cilindro, lo cual definimos en el *script*:

Globes:

Para tratar el siguiente tema debemos hablar de **Proyecciones equirectangulares:**
A modo de una explicación breve diremos que desde que se descubrió que la Tierra es redonda, los cartógrafos y los marineros han tenido dificultades para proyectar el globo esférico en un gráfico bidimensional.
En el modelado gráfico por computadora, se puede realizar un mapeado de la tierra, y con medios de 360 grados de una manera más detallada, esto normalmente se hace usando una proyección equirectangular (o Meridiana)



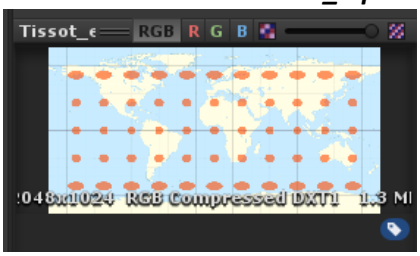
La siguiente imagen muestra el globo sin envolver con una proyección ***equirectangular***:



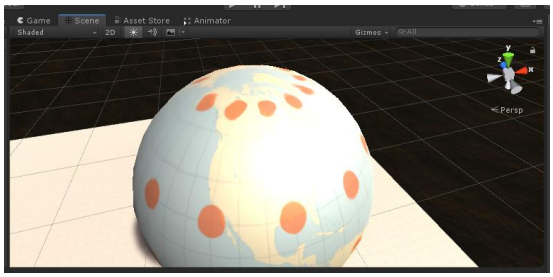
Usaremos una malla ***equirectangular*** para nuestras esferas fotográficas y una imagen proyectada (deformada) para su mapa de textura.

Realizaremos los siguientes pasos:

a. Crearemos una nueva escena y moveremos el cilindro anterior a un costado cambiando su posición.
b. Crearemos una nueva esfera con posición (0, 1.5, 0) y le añadimos el script *Rotator* que trabajamos anteriormente.
c. Finalmente insertamos la textura llamada "***Tissot_equirectangular***" en la esfera



Tenga en cuenta que, desafortunadamente, los círculos ***Tissot*** son ovalados, no circulares, excepto a lo largo del ecuador. Resulta que la esfera predeterminada provista en Unity no encaja bien en los mapas de textura equirectangular. En cambio, proporcioné uno diseñado específicamente para este propósito, ***PhotoSphere.fbx*** (que es el modelo de esfera predeterminado en 3D Studio Max).



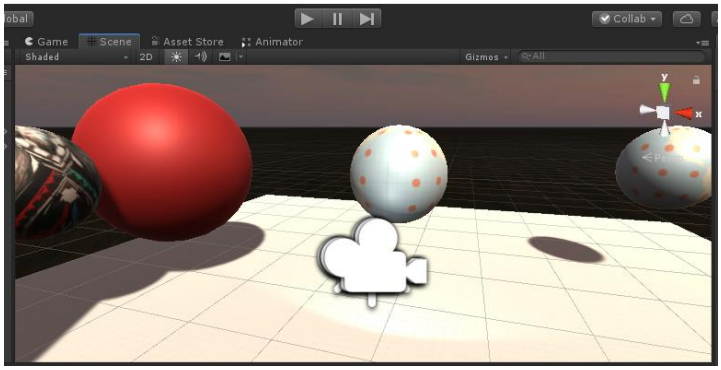
Podemos observar que los círculos de la esfera no son tan regulares.

Vamos a resolver ese problema mediante los siguientes pasos:

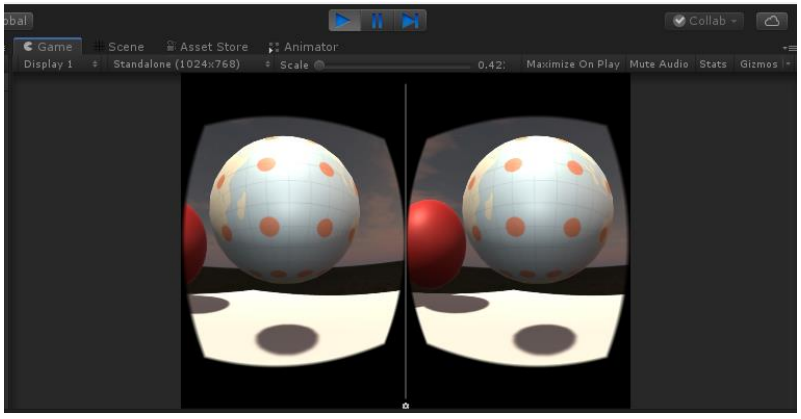
a. Movemos la esfera anterior a otra posición (3, 1.5, 1.5)
b. Importamos el archivo "***PhotoSphere.fbx***"
c. Creamos una nueva esfera arrastrando el prefab *PhotoSphere* de models

d. Establecemos su posición en (0, 1.5, 0) y le agregamos el script ***Rotator*** de igual manera.
e. Arrastramos la textura "***Tissot_equirectangular***" a la esfera.

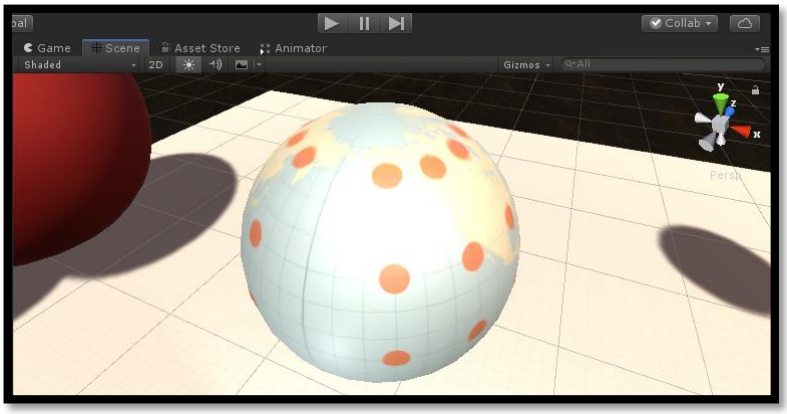
Finalmente, en la escena tenemos:



Y al probarlo en RV:



Veremos que el problema fue solucionado y se ve mucho mejor:



PhotoSpheres:

Esta escena tratará un tema nuevo en la actualidad que son las fotos en fotosferas de 360grados. Sólo necesitaremos una foto de 360 grados.

Realizaremos los siguientes pasos:

a. Crear una nueva escena completamente ***vacía***
b. Crear una esfera ***equirectangular*** arrastrando el model que teníamos

c. Restablecer los *valores predeterminados*
d. Cambiamos su escala a (10, 10, 10)
e. Creamos un nuevo material llamada "***PhotoSphereMat***" al cual le añadiremos el **shader**|InwardShader tal como hemos hecho en este capitulo.
f. Desactivamos las sombras, en *Mesh Renderer* deseleccionamos *Receive Shadows*.
g. Desactivamos la luz direccional global de la escena
h. Iluminamos el interior de la esfera: GameObject | Light | Point Light
i. Eliminamos la *Camara* principal de la escena
j. Seguidamente colocamos el prefab ***MeMyself Eye*** y lo colocamos en la posición (0, -0.4, 0)
k. Importamos la foto que queramos (tiene que ser en 360 grados), elegimos el tamaño máximo de importación 8192.
l. Finalmente arrastramos la textura al material que creamos anteriormente para la esfera.

Si estamos utilizando un dispositivo de seguimiento posicional como **Oculus Rift** debemos desactivarlo, para ello usaremos el siguiente script:

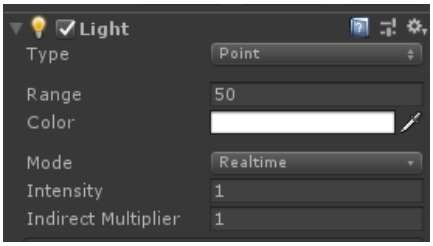
a. Agregamos el script "*DisablePositionalTracking*" al objeto MeMyselfEye

```
using UnityEngine;
using System.Collections;
public class DisablePositionalTracking : MonoBehaviour {
private Vector3 position;
void Start () {
        position = Camera.main.transform.position;
}
void Update () {
        Camera.main.transform.position = position;
}
}
```

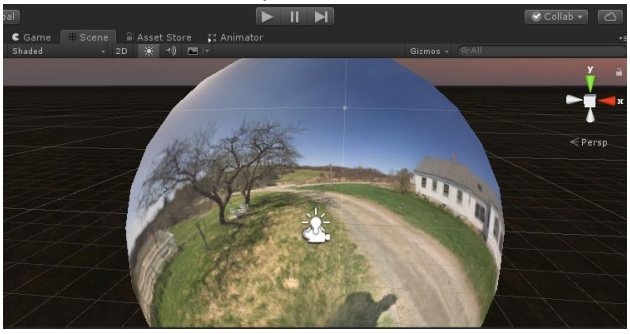
Explicación: Seguimos usando los mismos paquetes de *UnityEngine* y *System.Collections*. Nuestra clase *DisablePositionalTracking* derivará de la clase Unity *MonoBehaviour*. Este script tiene una variación, pues están los métodos *update*() y *start*(). El método *start*() es inicializador, es decir antes de hacer uso del método update recopilará los datos. En este caso, capturara la posición de la cámara. Y el método *update*() actualizará esta ubicación de esta manera se realizará el seguimiento posicional.

Al correr la escena podemos observar que nos encontramos dentro de la imagen y nos da la sensación como si estuviéramos dentro de esa escena y lugar:

Range (Point Light) es la fuerza de iluminación dentro de la esfera, a más valor más iluminación:



Estableciendo el valor en 50 notaremos que ahora se encuentra correctamente iluminada:

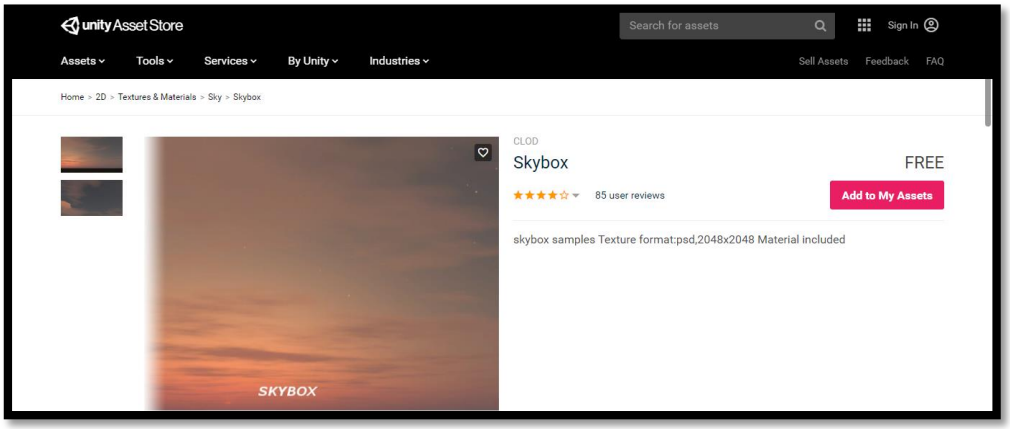


Y al probarlo en VR:

Mejoras Implementadas:

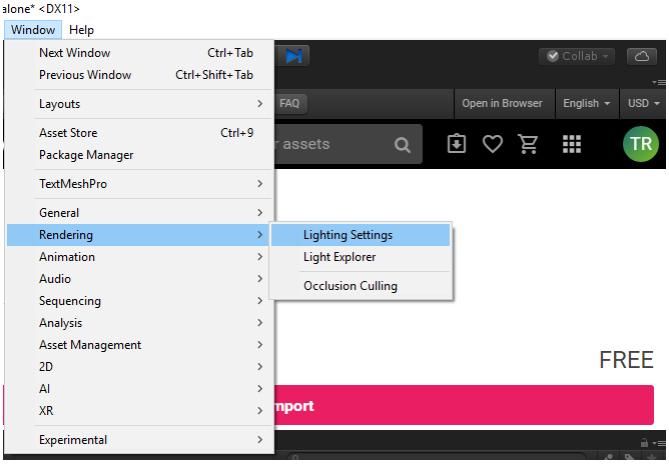
1) SKYBOX

Para añadir una implemetación de SkyBox de acuerdo a nuestros gustos, nos dirigiremos a la tienda de Unity donde podremos encontrar varias opciones, en nuestro caso descargamos la siguiente de https://assetstore.unity.com/packages/2d/textures-materials/sky/skybox-4183

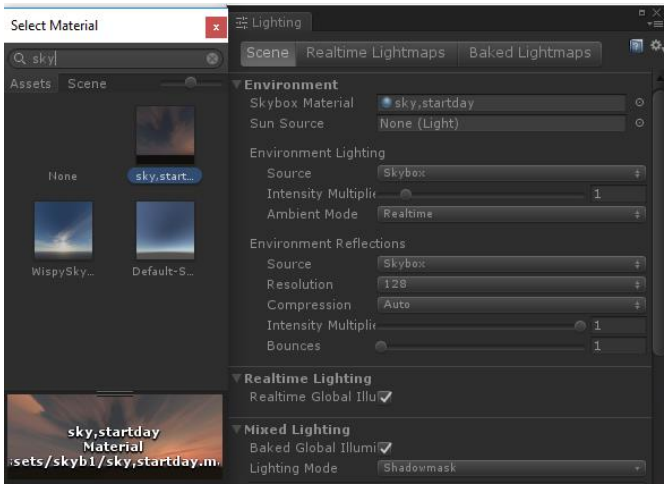


Para utilizarlo debemos seguir el siguiente procedimiento:

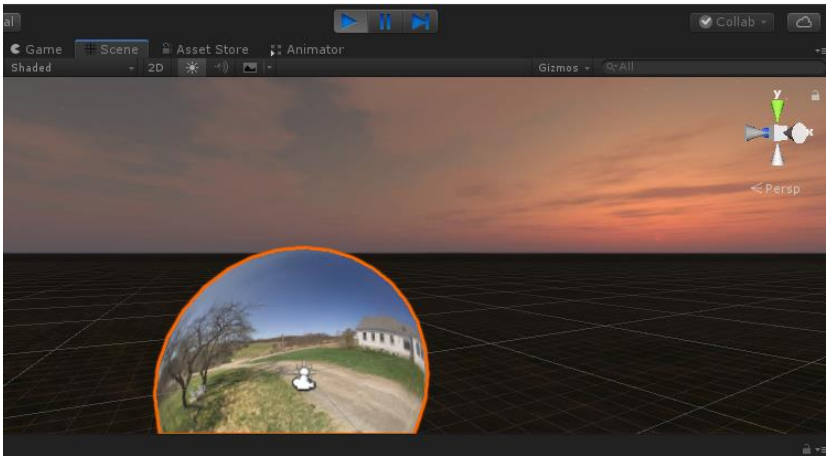
a. Ir a Windows|Rendering|Lighting Settings



b. En Lighting seleccionar el redondo al lado de *SkyBoxMaterial* y escoger el que descargamos:



c. Veremos que en nuestra escena se podrá observar el fondo que elegimos:

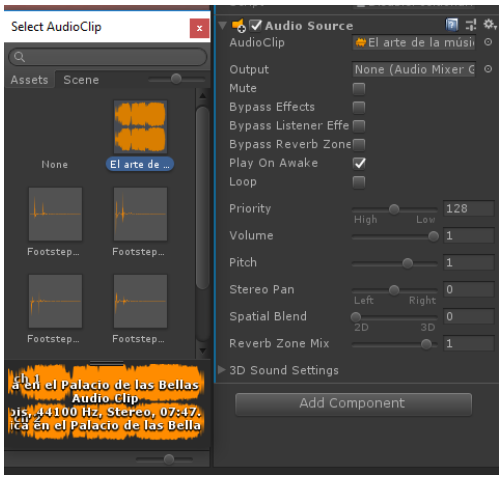


2) MUSICA

Para agregar música a nuestras escenas bastará con descargarla en formato *.mp3* y copiarla al proyecto, para en nuestro caso la guardamos en una carpeta llamada MUSICA:

a. En cada escena añadiremos un componente en cualquier lugar, en este caso en ***MeMyselfEye***: AddComponent|AudioSource

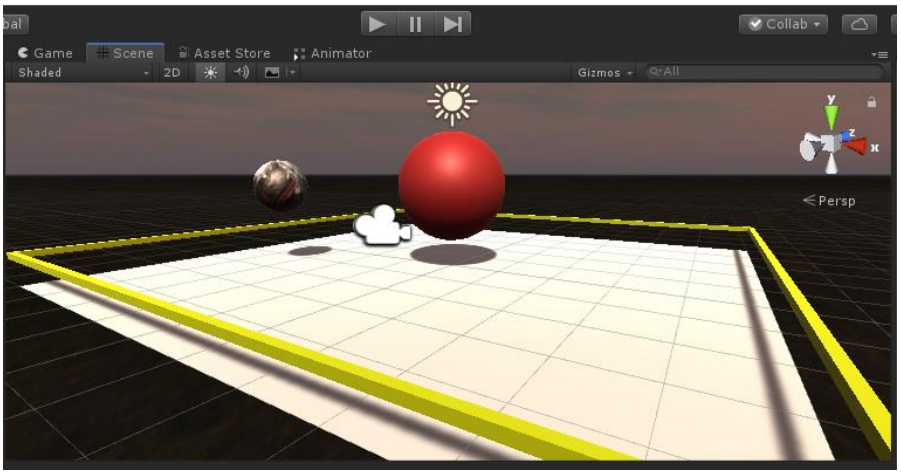
b. Y dentro elegiremos la pista .mp3 de nuestra música de fondo, al darle Play la reproducirá junto a nuestra escena



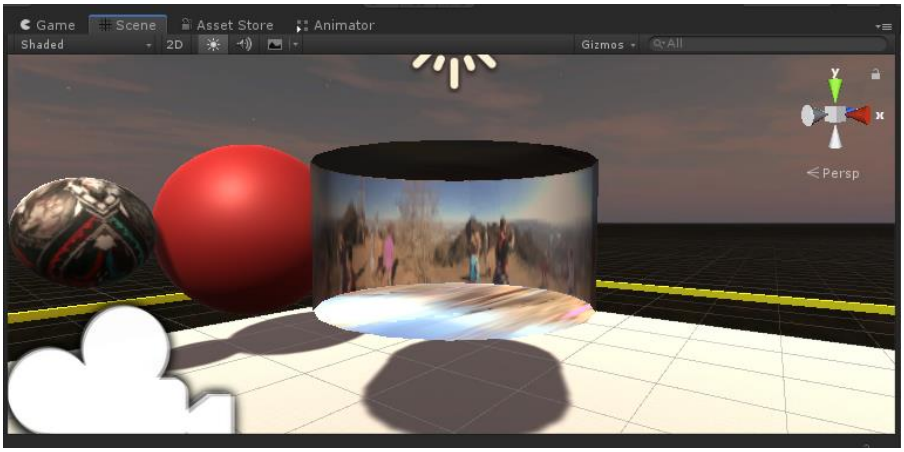
3) LIMITES DEL MUNDO

Agregaremos límites a los mundos (escenas) en los cuales se tenga el movimiento de la cámara guiada por nosotros, con el fin de evitar que se caiga al vacío tal y como aprendimos en capítulos anteriores:

- ***MagicalOrbs:***



- ***Panorama***



- ***Globes***