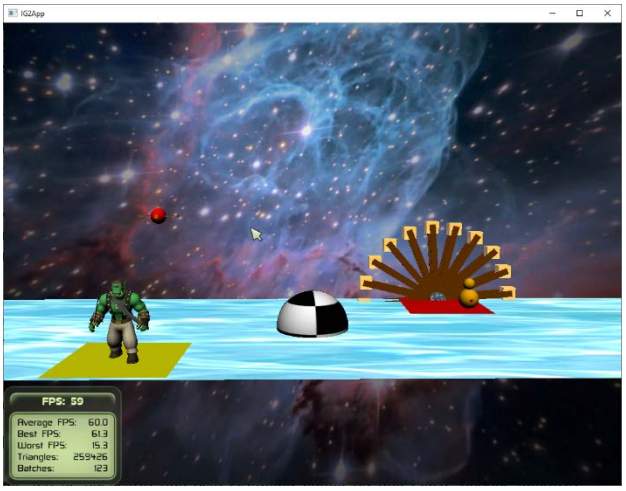
INFORMÁTICA GRÁFICA 2
Grado en desarrollo de videojuegos
Curso 2022-23
Práctica 2

(Entrega 1 de la Práctica 2, apartados 1-)

1. Limpia el código y deja solo la escena con el avión, la bomba, Sinbad, el rio, la plataforma amarilla, el muñeco y la noria. Es decir, elimina las nubes y la niebla. Si algún objeto dispone de foco, elimínalo y deja únicamente la luz direccional, encendida, con dirección (0, -1, -1) y con componente difusa (1.0, 1.0, 1.0). Configura Ogre seleccionando **`OpenGL 3+ Rendering Subsystem`**.

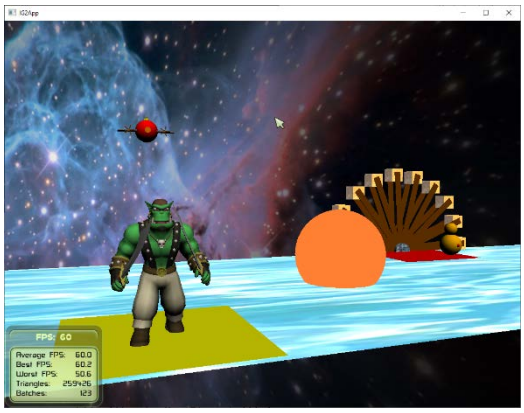
2. Vamos a definir un **`SkyPlane`** plano como combinación de texturas. Para ello define una entrada **`IG2/space`** en el archivo de material que especifique que este material no interactúa con la luz, que modula las texturas **`lightMap.jpg`** y **`spaceSky.jpg`** de las unidades de textura 0 y 1 respectivamente y que estas unidades tienen las mismas coordenadas de textura. Configura una animación de rotación para la unidad de textura 1.

3. Añade a la escena un **`SkyPlane`** vertical que tenga asociado el material **`IG2/space`** definido en el apartado anterior y que se renderice como se muestra en la captura adjunta. No es necesario que este skyplane venga de una clase, puedes añadirlo directamente en el **`setupScene()`** de **`IG2App`**. Experimenta con los parámetros del método **`setSkyPlane()`**; por ejemplo, haz pruebas con la normal y la distancia del plano, con la curvatura, con el zoom, etc.



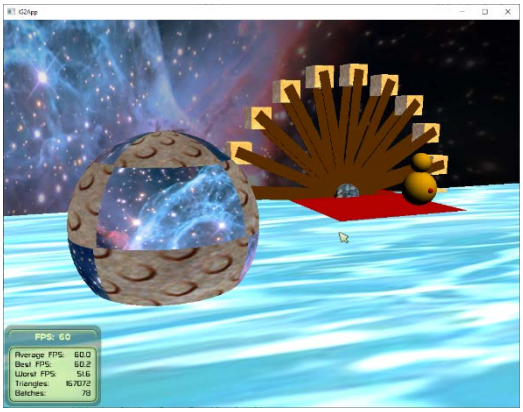
En los apartados que siguen, para el material de los shaders utiliza un archivo que se llame `practica2GLSL.material`. Para la esfera debes utilizar la malla `uv_sphere.mesh` que proporciona, además de los vértices, también normales y coordenadas de textura.

4. Define los primeros shaders que se explican: **`MyFirstShaderVS.glsl`** y **`MyFirstShaderFS.glsl`** y que pintan la bomba de color naranja.



5. Define la modulación de texturas del **`SkyPlane`** que se hizo en el segundo apartado, pero usando shaders que llamaremos **`SpaceSkyVS.glsl`** y **`SpaceSkyFS.glsl`**. Recuerda que las texturas que se modulan son **`lightMap.jpg`** y **`spaceSky.jpg`**.

6. Define un shader de fragmentos que llamaremos **`BombaTeseladaFS.glsl`** que muestre la textura de fondo (**`spaceSky.jpg`**) en las zonas negras de la textura **`checker.png`**, y la textura **`BumpyMetal.jpg`**, en las zonas blancas, tal como se ve en la captura adjunta. En este caso, el shader de vértices **`BombaTeseladaVS.glsl`** se limita a proporcionar las coordenadas de textura que necesita el shader de fragmentos y los vértices en coordenadas de recorte.



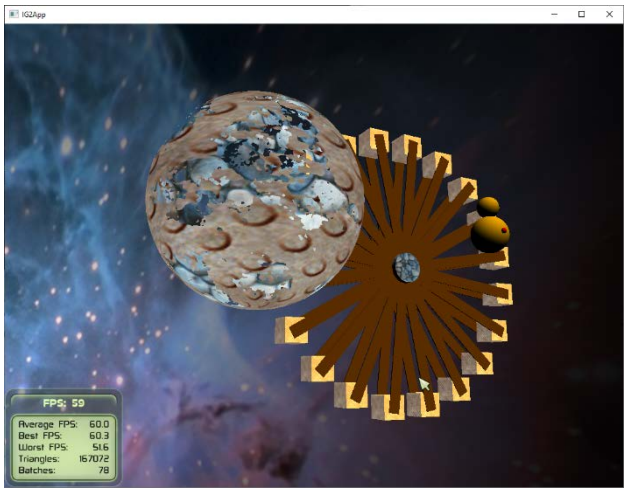
7. Modifica los shaders anteriores y define dos nuevos shaders **`SpaceVS.glsl`** y **`SpaceFS.glsl`** de forma que el primero pasa al segundo dos juegos de coordenadas de textura: **`vUv0`** para la textura **`lightMap.jpg`** y **`vUv1`** para **`spaceSky.jpg`**. A estas últimas se les aplica un zoom centrado de factor **`ZF=0.5`**. Recuerda que el intervalo de las coordenadas de textura es [0, 1] de forma que, para aplicarles un zoom centrado, primero hay que centrarlas, trasladándolas al intervalo [-0.5, 0.5], luego se escalan y por último hay que deshacer la traslación. Es decir, dadas las coordenadas de textura **`c0`**, las nuevas **`c1`** se obtienen así: **`c1=(c0-0.5)*ZF+0.5`**.

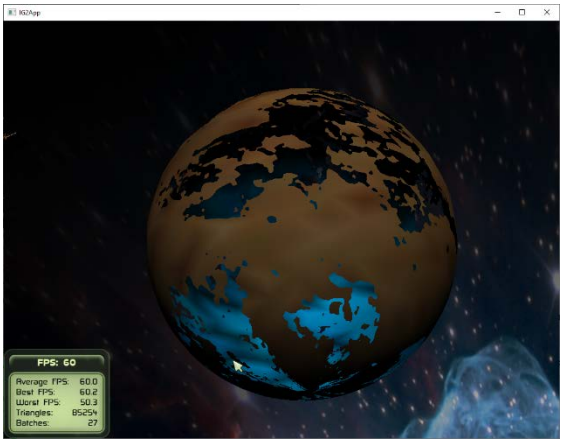
8. Pon animación al escalado de forma que la textura crezca y decrezca sin parar. Usa para ello un factor de escalado **`ZF=st*a+b`**, donde **`st∈[-1,1]`**. Para estos valores, la textura debe pasar desde la mitad de su tamaño a su tamaño original. Calcula los valores **`a`**, **`b`** para que esto suceda y completa el vertex shader. Para dar valor a **`st`** usa una variable **`uniform float sintime`** que toma valores de forma automática con **`sintime_0_2pi`**, repitiéndose con el intervalo de segundos que prefieras.

9. **(NUEVO)** Modifica el shader de vértices del apartado 6 para que se haga zoom animado en la textura **`spaceSky.jpg`** de la bomba teselada.

10. Define shaders para poner agujeros en la bomba. El novedoso es el de fragmentos que se llamará **`HolesFS.glsl`**. Utiliza la textura **`corrosión.jpg`** para descartar aquellos fragmentos cuyo téxel tenga componente roja mayor que 0.6. Este shader utiliza los parámetros uniformes **`texFront`** y **`texBack`** para las texturas exterior e interior de la bomba, respectivamente. Así mismo, el shader utiliza la variable booleana predefinida **`gl_FrontFacing`** para saber si el fragmento es de una cara frontal o de una trasera, tal como se explica en transparencias. Las texturas que dan valor a los

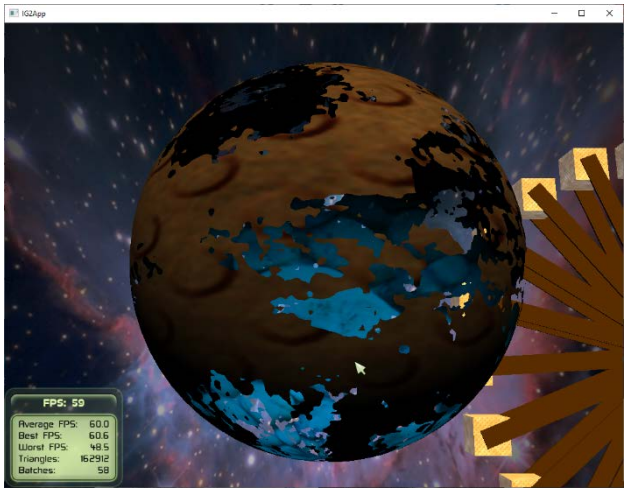
parámetros uniformes referidos a texturas son **`BumpyMetal.jpg`** y **`BeachStones.jpg`**, respectivamente. Ten cuidado con el **`culling`**. Abajo tienes una captura de la bomba con agujeros desde debajo del rio. Observa que los agujeros permiten ver tanto el interior de la esfera como el skyplane.





11. Define los shaders **`HolesAndVertexLightingVS.glsl`** y **`HolesAndVertexLightingFS.glsl`** para iluminar la bomba con agujeros del apartado anterior, pero hazlo en el shader de vértices, tal como se explicó en clase. Recuerda que se hace usando coordenadas de vista, aunque puedes usar otras. La luz es la suministrada con el proyecto que es direccional, con dirección (0, -1, -1) y luz blanca como componente difusa. No añadas componente ambiente. El color de las caras frontales es ocre siena (0.72 0.57 0.35) y el de las caras traseras es azul cerúleo (0.0, 0.6, 0.83). Una vez determinado el color resultante de un vértice por la luz, modúlalo por el color de la textura, según sea cara frontal o trasera, siendo las texturas las del apartado anterior. Aquí tienes una captura de lo que resulta.

12. Define los shaders **`HolesAndLightingVS.glsl`** y **`HolesAndLightingFS.glsl`** para iluminar la bomba con agujeros del apartado anterior, **pero en el shader de fragmentos**. Puedes hacerlo usando las coordenadas que quieras. El vertex shader pasa al fragment shader el vértice y la normal en las coordenadas que hayas elegido. El fragment shader calculará el color correspondiente a la iluminación usando la componente difusa de la luz junto con su posición/dirección, en las coordenadas que hayas elegido. Como coeficiente de reflexión difusa del material, el fragment shader utiliza el color correspondiente de la cara, según sea frontal o trasera, dados en el apartado anterior. Las texturas irán moduladas entonces con estos coeficientes de reflexión difusa. Aquí tienes una captura del resultado. Compárala con la del apartado anterior.



13. Define los shaders **`SpotLightVS.glsl`** y **`SpotLightFS.glsl`** para simular el efecto de un foco situado sobre una esfera que se halla inmóvil encima del rio. El shader de vértices pasa vértices y normales en coordenadas globales, más coordenadas de textura. El shader de fragmentos calcula si el fragmento de la esfera está dentro del cono de emisión de un foco (luz posicional) situado encima de ella y que emite luz hacia abajo. Para averiguarlo, recuerda usar el (coseno del) ángulo entre el vector normal y el vector a la fuente de luz.

La amplitud del cono del foco (cutoff) es un parámetro uniforme del shader de fragmentos que varía cíclicamente con el tiempo, entre el coseno de 0 y el de 2 veces pi (**`costime_0_2pi`**). La intensidad de la luz del fragmento se atenúa por un factor que vale 1.0 (no hay atenuación) en los fragmentos dentro del cono de emisión, y 0.2, en los que están fuera del cono. La textura usada en la esfera es **`BeachStones.jpg`**. Las capturas de más abajo intentan mostrar cómo se ve iluminada la esfera según varía la amplitud del cono de emisión del foco con el tiempo.