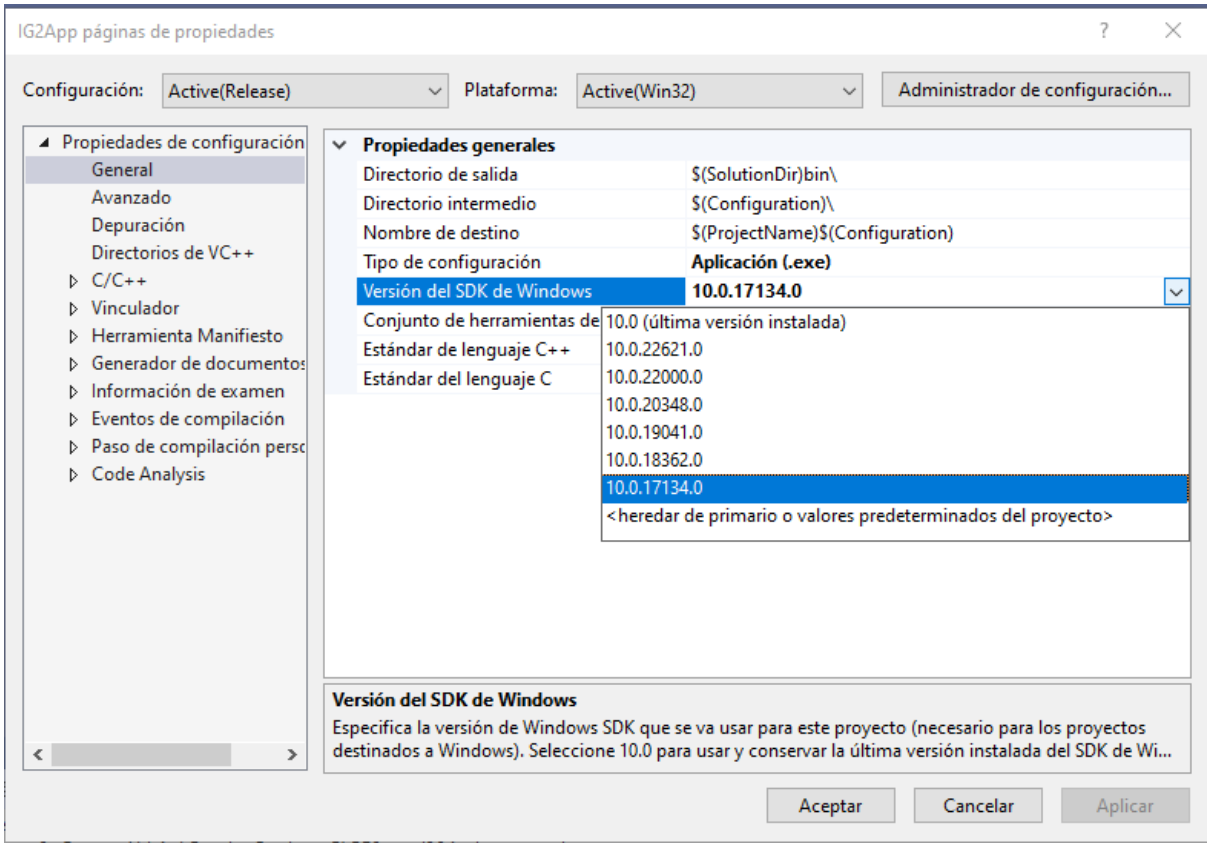
Informática Gráfica II: Primer contacto con Ogre 3D

Clase de laboratorio (11/09/2024)

La finalidad de esta práctica es establecer un primer contacto con el motor gráfico Ogre 3D. Para ello, utilizaremos Visual Studio 22 y el SDK de Microsoft "Windows 10 SDK, versión 1803 (10.0.17134.12)", el cual se puede descargar desde el siguiente enlace: https://developer.microsoft.com/es-es/windows/downloads/sdk-archive/

Con Visual Studio se pueden utilizar distintos SDKs, pudiendo tener instalados varios simultáneamente de forma que se elija en cada momento cuál se utiliza para la compilación.

Al cargar el proyecto, aparecerá un diálogo para elegir SDK. Esto también podremos hacerlo en las propiedades del proyecto, tal y como se muestra en la siguiente imagen:



El primer paso consistirá en bajar – desde el Campus Virtual – el proyecto Ogre3D que utilizaremos como plantilla (ver Sección Prácticas). Una vez descargado, se descomprime en la ruta elegida para poder cargarlo en Visual Studio. Tened en cuenta que en los laboratorios de la FdI no tenemos acceso a todos los directorios, por lo que es recomendable guardar nuestros proyectos en `hlocal`.

En el proyecto, nuestro programa estará en la carpeta IG2App. Dentro de esta carpeta tenemos varios ficheros:

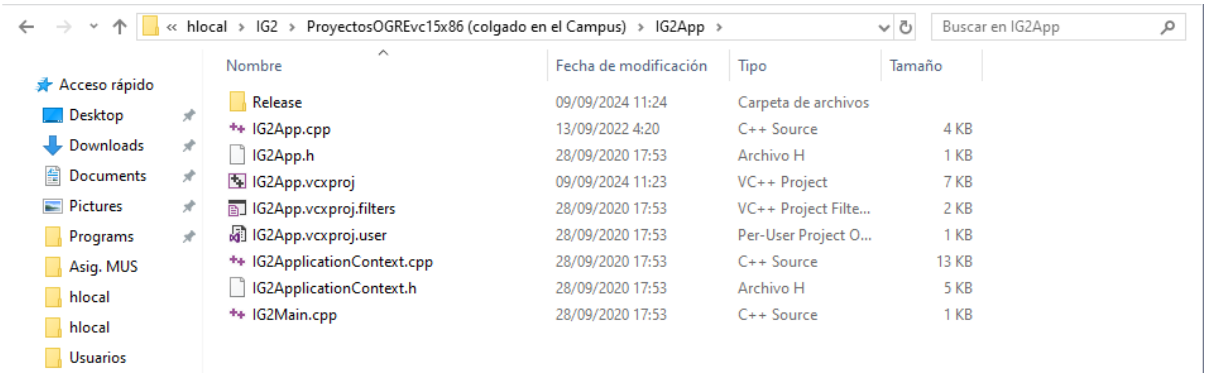
- IG2Main.cpp: Fichero que contiene el punto de entrada al programa (función main).

- IG2App.h: Definición de la clase principal del programa que contiene los métodos y atributos. Por ahora, en esta clase definiremos, como atributos, los elementos que vamos a utilizar en la escena, aunque posteriormente utilizaremos una estructura más compleja y estos elementos se distribuirán en más clases.

- IG2App.cpp: Implementación de los métodos definidos en la clase principal.

- IG2ApplicationContext.h: Definición de la clase que configura y gestiona el contexto de la aplicación. Entre otras acciones, se encarga de cargar los ficheros de configuración, localizar los recursos, y crear el elemento *Root* del sistema Ogre.

- IG2ApplicationContext.cpp: Implementación de los métodos definidos en la clase IG2ApplicationContext.



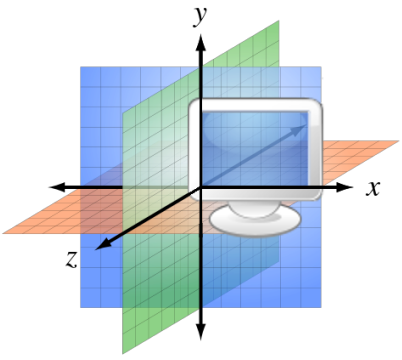
Para compilar el código fuente es necesario utilizar el modo `Release` en `x86`.

En esta práctica, modificaremos únicamente los ficheros IG2App.h y IG2App.cpp. En clase se explicará en detalle lo que hace cada parte del programa, como por ejemplo, el subprograma `setup()` que, esencialmente, configura el entorno para que podamos renderizar la escena.

En el directorio *bin* podremos encontrar los ficheros plugins.cfg y resources.cfg. Estos ficheros son importantes, ya que indican la ubicación de los *plugins* y los recursos que utilizaremos en la escena. Generalmente, el fichero plugins.cfg no se modifica, pero sí modificaremos el fichero resources.cfg para indicar dónde localizar materiales, mallas o contenidos que utilicemos en la escena. En clase se explicará con detalle el contenido de este fichero.

Otro fichero importante es ogre.cfg. En este fichero se guardan los parámetros de configuración para renderizar la escena como, entre otros, la resolución o el sistema de renderizado. Si borramos este fichero, la siguiente vez que ejecutemos el programa, aparecerá el diálogo de configuración que nos permitirá indicar estos parámetros.

Antes de empezar con la creación de la escena, es importante tener en cuenta el sistema de coordenadas que utiliza Ogre:



En nuestro caso, el código que creará la escena estará en el subprograma `setupScene`. Lo primero que haremos será crear una cámara.

La posición de la cámara se indica en la sentencia:

```
mCamNode->setPosition(0, 0, 1000);
```

Mientras que el punto donde mira la cámara se establece con:

```
mCamNode->lookAt(Ogre::Vector3(0, 0, 0), Ogre::Node::TS_WORLD);
```

Seguidamente, crearemos una luz. En Ogre existen varios tipos de luces, que veremos durante el curso. En este caso, utilizaremos una luz direccional. La dirección de la luz se indica en la sentencia:

```
mLightNode->setDirection(Ogre::Vector3(0, -1, -1));
```

Una vez creada la cámara y la luz, vamos a crear algunos objetos y mostrarlos en la escena.

Primero, mostraremos a Sinbad. La malla `Sinbad.mesh` se encuentra en los elementos proporcionados por Ogre3D. En el fichero resources.cfg indicamos su ubicación con la línea:

```
Zip=../Media/packs/Sinbad.zip
```

En este punto podemos rotar la cámara para ver a Sinbad desde distintos ángulos, dejando presionado el botón izquierdo del ratón. Además, con el *scroll* del ratón podemos acercar o alejar la cámara.

Podéis hacer pruebas modificando la posición de la cámara y la luz.

La siguiente línea hace que se muestre la *bounding box* de Sinbad:

```
mSinbadNode->showBoundingBox(true);
```

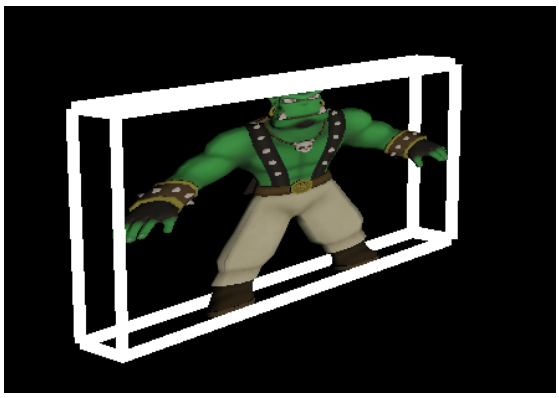
Probad a establecer a falso ese valor, para que podáis ver que no se muestra en este caso.

La escala de Sinbad se puede modificar en la siguiente sentencia:

```
mSinbadNode->setScale(x, y, z);
```

donde cada parámetro hace referencia a los ejes `x`, `y`, y `z`, respectivamente.

Probad a modificar la escala de Sinbad, modificando únicamente un parámetro, y observaréis cómo se modifica en función del eje modificado.



La posición de Sinbad también podemos configurarla con la siguiente sentencia:

```
mSinbadNode->setPosition(x, y, z);
```

Modificad estos parámetros para que veáis que la posición de Sinbad variará en función de los mismos.

En Ogre3D es posible imprimir mensaje con información de los elementos que forman la escena. Además, en la clase IG2App contamos con el método:

```
bool IG2App::keyPressed(const OgreBites::KeyboardEvent& evt);
```

el cual se ejecuta cuando se ha pulsado una tecla. En este caso, al pulsar ESC, se finalizará la ejecución del programa. Si añadimos el siguiente código, cada vez que pulsemos la tecla 'k', se mostrará por pantalla la posición de Sinbad y de la cámara.

```
else if (evt.keysym.sym == SDLK_k){
    cout << "Position of Sinbad: " <<  mSinbadNode->getPosition() << endl;
    cout << "Position of the camera: " << mCamNode->getPosition() << endl;
}
```

Ahora vamos a mostrar un dragón. Para ello, incluimos el .zip que contiene los datos relativos al dragón, como su malla. Si abrimos el fichero resources.cfg, debemos quitar el comentario de la línea que contiene el pack `dragón.zip`. Seguidamente, como hicimos con Simbad, creamos la entidad con la malla `dragon.mesh`, creamos un nodo que llamaremos *nDragon*, adjuntamos la entidad al nodo y lo mostraremos con su *bounding box*.

Es posible que, dependiendo de la posición, no aparezca directamente en la escena y tengamos que desplazar la cámara para verlo.