



OGRE 3D

Object-oriented Graphics Rendering Engine

<http://www.ogre3d.org>

Material original: Ana Gil Luezas
Adaptación al curso 24/25: Alberto Núñez
Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid

Object-oriented Graphics Rendering Engine

- ❑ Motor gráfico de código abierto (licencia MIT)
- ❑ Implementado en C++
 - ❑ Uso extensivo de mecanismos de O.O. y namespaces (**namespace Ogre**)
- ❑ Multiplataforma: Windows, Linux, MacOS, Android
- ❑ API gráfico seleccionable: OpenGL, Direct3D
- ❑ Diseñado para ser extendido:
 - ❑ Clases abstractas, componentes, herencia (múltiple), polimorfismo y *plug-ins* (complementos).
 - ❑ Permiten ampliar la funcionalidad sin modificar el código

- ❑ Un motor gráfico es un compendio de *gestores*:
 - ❑ Gestor del API gráfico (archivo **ogre.cfg**)
 - ❑ Gestor de recursos (archivo **resources.cfg**)
 - ❑ Mallas, Texturas, Materiales
 - ❑ Shaders (GPU programs), ...
 - ❑ Gestores de escena
 - ❑ Gestor de *log* (archivo **ogre.log**)
 - ❑ Gestor de plugins (archivo **plugins.cfg**)
 - ❑ Gestor de archivos
 - ❑ ...
- ❑ En general, los gestores son ***singleton*** (instancia única)

- ❑ Definición de materiales mediante scripts:
 - ❑ *Uso de shaders (GPU programs: Cg, GLSL, HLSL, ensamblador)*
 - ❑ Selección automática de la versión del material (*technique*)
 - ❑ Nivel de detalle (*LOD*)
- ❑ Comunicación mediante listas de observadores (*listeners*) para recibir notificaciones
- ❑ El objeto **Ogre::Root*** **root** es el punto de entrada al sistema.
 - ❑ Debe ser el primero en crearse y el último en destruirse.
 - ❑ Permite inicializar el sistema e iniciar el bucle de renderizado.

- ❑ Permite el acceso a todos los subsistemas
- ❑ Debe ser el primero en ser creado, y el último en ser destruido
- ❑ Al crear una instancia de `Root` se inicializa Ogre
 - ❑ De la misma forma que se cierra al eliminar su instancia.
- ❑ El constructor tiene la forma:

```
Ogre::Root::Root(const String &pluginFileName = "plugins.cfg",  
                 const String &configFileName = "ogre.cfg",  
                 const String &logFileName = "Ogre.log");
```

donde:

- ❑ `pluginFileName` es el fichero con la información de los *plugins*.
- ❑ `configFileName` es el fichero con la configuración a ser cargada.
- ❑ `logFileName` es el fichero donde se almacena el log.

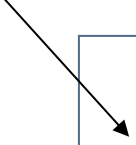
- ❑ El método `Ogre::Root::showConfigDialog` permite configurar el sistema de forma interactiva a través de un diálogo.
 - ❑ Entre otras opciones, podemos configurar la resolución, la profundidad de color, las opciones de pantalla completa, ...
- ❑ Con `Root` se pueden obtener punteros a otros objetos en el sistema
 - ❑ `Ogre::SceneManager`, `Ogre::RenderSystem` y otros administradores de recursos.
- ❑ OGRE se puede ejecutar en modo de *renderizado continuo*
 - ❑ Refrescar siempre todos los elementos de renderizado tan rápido como sea posible
 - ❑ Invocar el método `Ogre::Root::startRendering`
 - ❑ Termina cuando:
 - ❑ Todas las ventanas de renderizado se cierran
 - ❑ Cualquier objeto `Ogre::FrameListener` indique que quiere detener el ciclo

Ficheros de configuración

- ☐ Ogre utiliza varios archivos de configuración (*.cfg) en formato INI.
 - ☐ clave = valor
 - ☐ Comentarios con #
- ☐ Controlan qué *plugins* se cargan y dónde se buscarán los archivos de recursos.
- ☐ Estos archivos se buscan en un conjunto de ubicaciones predefinidas.
 - ☐ En nuestro caso, estos ficheros estarán en el directorio **bin**
 - ☐ También es posible establecer la variable de entorno **OGRE_CONFIG_DIR** para establecer una ubicación alternativa de los archivos de configuración.
- ☐ Estos ficheros son:
 - ☐ plugins.cfg
 - ☐ ogre.cfg
 - ☐ resources.cfg
- ☐ En Windows, los ficheros ogre.cfg y ogre.log también se pueden ubicar en el directorio `MisDocumentos/Ogre/nombreProyecto/ogre.X`

❑ Fichero **plugins.cfg**

- ❑ Un plug-in es un módulo de código (.dll en Windows, .so en Linux)
 - ❑ Implementa uno de los interfaces de plug-in en Ogre
 - ❑ SceneManager o RenderSystem.
- ❑ Contiene la lista de *plugins* que Ogre carga al inicio
- ❑ El nombre del fichero se puede cambiar, siempre que se indique su nueva ruta en el constructor de **Root**.
- ❑ La directiva **PluginFolder** indica el directorio que contiene los *plugins*
 - ❑ En el ejemplo, el directorio de trabajo actual.
 - ❑ En MacOS esta línea se ignora,
 - ❑ Se busca en `Resources/`
- ❑ Cada directiva **Plugin** indica un plug-in a cargar



```
# Defines plugins to load
# Define plugin folder|
PluginFolder=.

# Define plugins
# Plugin=RenderSystem_Direct3D9
# Plugin=RenderSystem_Direct3D11
Plugin=RenderSystem_GL
Plugin=RenderSystem_GLES2
Plugin=Plugin_ParticleFX
Plugin=Plugin_BSPSceneManager
# Plugin=Plugin_CgProgramManager
# Plugin=Codec_EXR
Plugin=Codec_STBI
# Plugin=Codec_FreeImage
Plugin=Plugin_PCZSceneManager
Plugin=Plugin_OctreeZone
Plugin=Plugin_OctreeSceneManager
```


☐ Fichero **ogre.cfg**

- ☐ Fichero que contiene la configuración con la que se iniciará Ogre
- ☐ Este fichero se crea automáticamente
- ☐ Se puede indicar la configuración a través del diálogo
- ☐ Si el fichero no existe, Ogre mostrará el diálogo (ver siguiente slide)
- ☐ Desde el programa, podemos forzar el diálogo

☐ `root->showConfigDialog();`

```
Render System=OpenGL 3+ Rendering Subsystem

[OpenGL Rendering Subsystem]
Colour Depth=32
Display Frequency=60
FSAA=0
Full Screen=No
RTT Preferred Mode=FBO
VSync=Yes
VSync Interval=1
Video Mode=1024 x 768
sRGB Gamma Conversion=No

[OpenGL 3+ Rendering Subsystem]
Colour Depth=32
Display Frequency=60
FSAA=0
Full Screen=No
RTT Preferred Mode=FBO
VSync=Yes
VSync Interval=1
Video Mode=1024 x 768
sRGB Gamma Conversion=No
```

OgreBitesConfigDialog



☐ Fichero **resources.cfg**

- ☐ Contiene una lista de recursos que Ogre debe cargar durante el arranque.
- ☐ Los recursos incluyen:
 - ☐ scripts, mallas, texturas, diseños GUI y otros.
- ☐ En este archivo se pueden utilizar tanto rutas absolutas como relativas
- ☐ No se pueden utilizar variables de entorno.
- ☐ Las rutas relativas se resuelven en relación a la ubicación de **resources.cfg**
- ☐ Ogre no buscará en subdirectorios.
- ☐ Los recursos se pueden cargar desde el sistema de archivos o desde un archivo ZIP.
- ☐ Distintos tipos de entradas:
 - ☐ Zip= indica que el recurso está en un archivo ZIP
 - ☐ FileSystem= indica que queremos cargar el contenido de una carpeta.

```
# Resources required by the sample browser and most samples.
```

```
[Essential]
```

```
Zip=../media/packs/SdkTrays.zip
```

```
Zip=../media/packs/profiler.zip
```

```
FileSystem=../media/thumbnails
```

```
# Common sample resources needed by many of the samples.
```

```
# Rarely used resources should be separately loaded by the samples which require them
```

```
[General]
```

```
# Bites uses the next entry to discover the platform shaders
```

```
FileSystem=../media
```

```
...
```

```
FileSystem=../media/IG2App
```

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

```
# FileSystem=../media/HLMS
```

```
# Zip=../media/packs/DamagedHelmet.zip
```

```
...
```

❑ Fichero **ogre.log**

- ❑ Fichero que contiene información diagnóstica de la ejecución
- ❑ Resulta útil para comprobar la causa de una parada no esperada o de un fallo
- ❑ Se puede cambiar el nombre en el constructor de **Root**

```
11:37:52: Creating resource group General
11:37:52: Creating resource group OgreInternal
11:37:52: Creating resource group OgreAutodetect
11:37:52: SceneManagerFactory for type 'DefaultSceneManager' registered.
11:37:52: Registering ResourceManager for type Material
11:37:52: Registering ResourceManager for type Mesh
11:37:52: Registering ResourceManager for type Skeleton
11:37:52: MovableObjectFactory for type 'ParticleSystem' registered.
11:37:52: ArchiveFactory for type 'FileSystem' registered
11:37:52: ArchiveFactory for type 'Zip' registered
11:37:52: ArchiveFactory for type 'EmbeddedZip' registered
11:37:52: DDS codec registering
11:37:52: ETC codec registering
11:37:52: ASTC codec registering
11:37:52: Registering ResourceManager for type GpuProgram
11:37:52: Registering ResourceManager for type Compositor
11:37:52: MovableObjectFactory for type 'Entity' registered.
11:37:52: MovableObjectFactory for type 'Light' registered.
11:37:52: MovableObjectFactory for type 'BillboardSet' registered.
11:37:52: MovableObjectFactory for type 'ManualObject' registered.
11:37:52: MovableObjectFactory for type 'BillboardChain' registered.
11:37:52: MovableObjectFactory for type 'RibbonTrail' registered.
11:37:52: MovableObjectFactory for type 'StaticGeometry' registered.
11:37:52: MovableObjectFactory for type 'Rectangle2D' registered.
11:37:52: Loading library /Users/alberto/Desktop/ogre/IG2-Prac0/IG2-Prac0/build/Release/Contents/Frameworks/RenderSystem_GL.dylib
11:37:52: Installing plugin: GL RenderSystem
```

bin\ogre.log

Informe sobre puglins

Informe sobre CPU

Informe sobre GPU

Informe sobre recursos

❏ main

```
#include "IG2App.h"
```

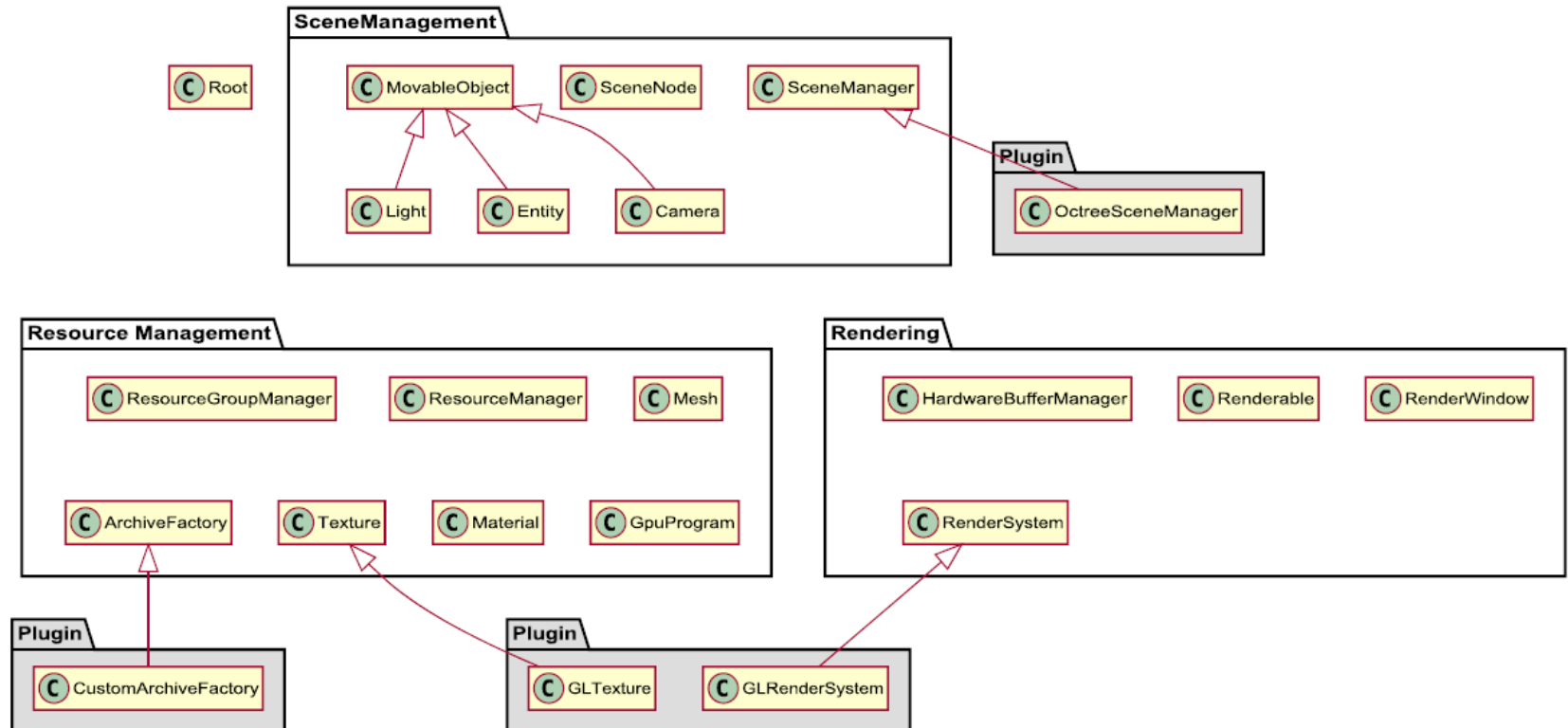
```
int main(int argc, char *argv[]){  
    IG2App app;  
    app.initApp();  
    app.getRoot()->startRendering();  
    app.closeApp();  
    return 0;  
}
```

```
IG2ApplicationContext::initApp(){  
    mRoot = new Root("...plugins.cfg", "...ogre.cfg", "...ogre.log");  
    mOverlaySystem = new OverlaySystem();  
    setup();  
}
```

```
IG2App::setup{  
    IG2ApplicationContext::setup();  
    mSM = mRoot->createSceneManager();  
    mTrayMgr = new OgreBites::TrayManager("TrayGUISystem", mWindow.render);  
    setupScene();  
}
```

```
IG2ApplicationContext::setup{  
    mRoot = new Root("plugins.cfg", "ogre.cfg", "ogre.log");  
    mOverlaySystem = new OverlaySystem();  
}
```

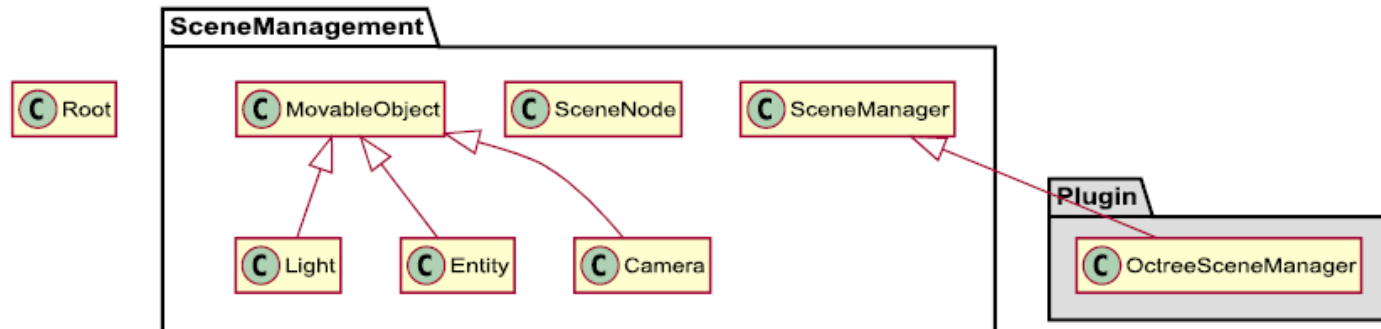
The core objects



ogrecave.github.io/ogre/api/latest/_the_core_objects.html

Scene Management

- ❑ Aparte del objeto `Ogre::Root`, `Ogre::SceneManager` es probablemente la parte más crítica del sistema desde el punto de vista de la aplicación.
- ❑ Generalmente, la mayor parte de la interacción con el SceneManager ocurre durante la configuración de la escena.
- ❑ Será el objeto más utilizado por la aplicación.
- ❑ Puede haber más de una instancia en la aplicación (no es singleton).



- ❑ Todo lo que aparece en la pantalla es gestionado por el **Ogre : : SceneManager**
- ❑ Mantiene un registro de las ubicaciones y otros atributos de los objetos de la escena
- ❑ Gestiona las cámara que son añadidas a la escena.
- ❑ Se encarga del contenido de la escena que va a ser renderizada por el motor
 - ❑ Organiza los contenidos usando cualquier técnica que considere mejor
 - ❑ Crear y gestionar
 - ❑ Todas las cámaras, Objetos móviles (entidades), luces y materiales.
 - ❑ Gestión de la «geometría del mundo»
- ❑ **SceneManager** mantiene un conjunto nombrado de todos los objetos de la escena
 - ❑ Es posible acceder a ellos a través de la instancia del Scene Manager
- ❑ El **SceneManager** también envía la escena al objeto **RenderSystem** cuando es el momento de renderizar la escena.
- ❑ Nunca se debe llamar al método **Ogre : : SceneManager : : _renderScene** directamente

- ❑ Existen múltiples tipos de Scene Managers.
 - ❑ Se diferencian en cómo dividen la escena para la selección y búsqueda de nodos.
 - ❑ Hay gestores que implementan el esquema *Octtree* y otros que utilizan portales.
- ❑ Si no se especifica ningún parámetro, OGRE usará el SceneManager por defecto, que es adecuado para escenas de tamaño pequeño y moderado.

```
SceneManager* Ogre::Root::createSceneManager();  
SceneManager* Ogre::Root::createSceneManager(const String & typeName,  
                                             const String &instanceName = BLANKSTRING);
```

- ❑ Donde:
 - ❑ typeName: Tipo del SceneManager a crear
 - ❑ instanceName: Nombre para la instancia del SceneManager.

❑ Tipos de Scene Managers

❑ ST_GENERIC

- ❑ Implementación mínima. No está optimizado para escenas o estructuras en particular.

❑ ST_INTERIOR

- ❑ Optimizado para escenas interiores con una alta densidad de elementos.

❑ ST_EXTERIOR_CLOSE

- ❑ Apropiado para exteriores con visibilidad cercana-media.

❑ ST_EXTERIOR_REAL_FAR

- ❑ Adaptado para paisajes.

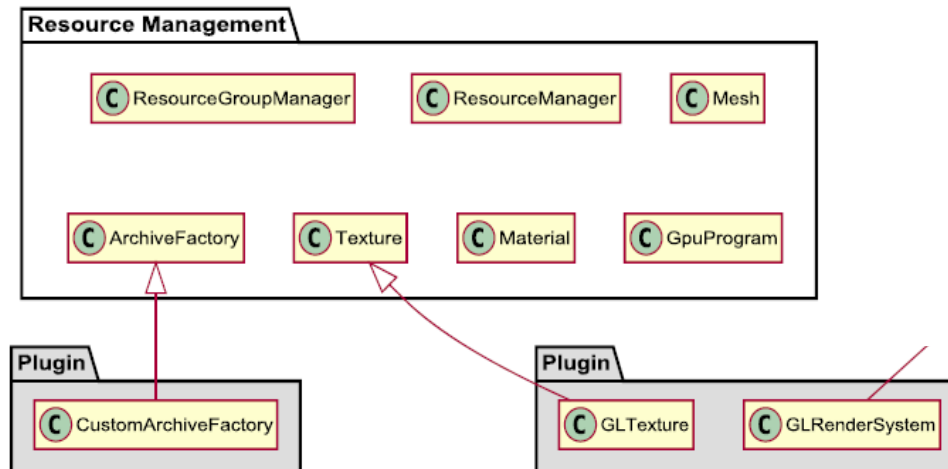
Resource Management

- ❑ El objeto `Ogre::ResourceGroupManager` (es singleton)
 - ❑ Gestiona los grupos de recursos (p.ej.: Essential, General, Tests).
 - ❑ Cada uno agrupa distintas clases de recursos: Mesh, Texture, Material, GPUProgram, Compositor, Font. (archivo [resources.cfg](#))
 - ❑ Todos los recursos son compartidos.

```
Ogre::TextureManager::getSingleton().someMethod()
```

```
Ogre::MeshManager::getSingleton().someMethod()
```

- ❑ Es posible indicar a los gestores de recursos dónde buscar recursos.
 - ❑ `Ogre::ResourceGroupManager::addResourceLocation`



- ❑ Los gestores de recursos garantizan que los recursos se carguen una sola vez y se compartan en todo el motor OGRE.
 - ❑ Gestionan los requisitos de memoria de los recursos.
 - ❑ Buscan en varias ubicaciones los recursos que necesitan, incluyendo múltiples rutas de búsqueda y archivos comprimidos (ficheros ZIP).
- ❑ Generalmente no se interactúa directamente con los gestores de recursos.
 - ❑ Serán llamados por otras partes del sistema OGRE cuando sea necesario
 - ❑ P.Ej: Al solicitar que una textura sea añadida a un Material

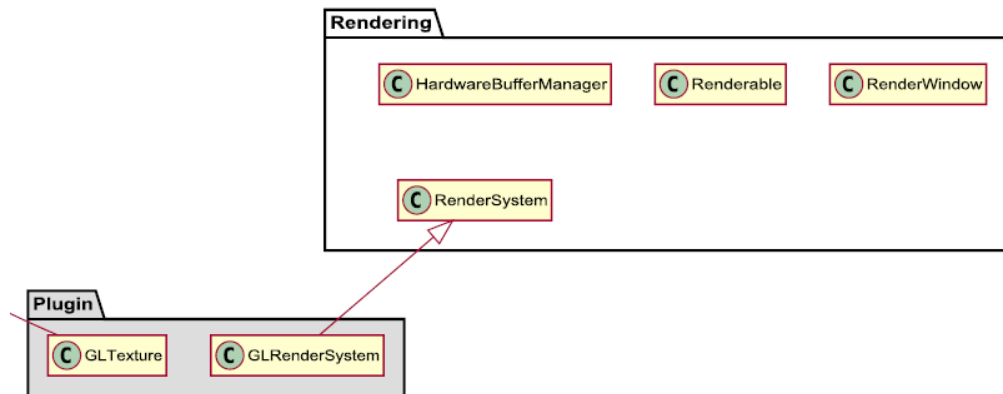
- ❑ **Ogre::RenderSystem** (clase abstracta)

- ❑ Interfaz común para las diferentes APIs gráficas 3D que subyacen en sus implementaciones (OpenGL, Direct3D)

- ❑ Habitualmente se utiliza a través del gestor de la escena

- ❑ También se puede acceder directamente a través de Root:

```
Ogre::RenderSystem* pRS = root -> getRenderSystem();
```

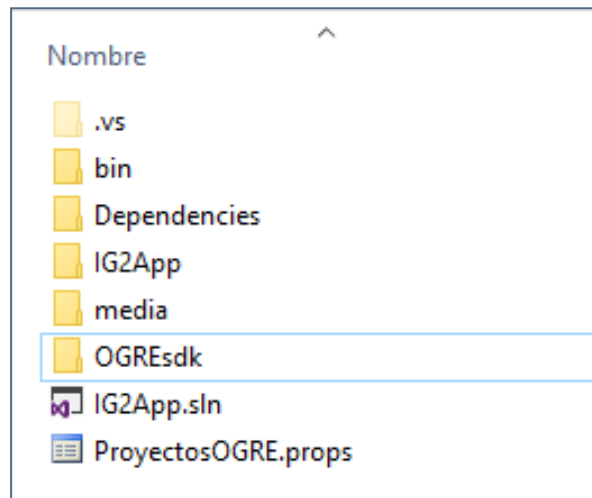


- ❑ Es responsable de enviar las operaciones de renderizado a la API y establecer todas las opciones de renderizado.
- ❑ Esta clase es abstracta porque toda la implementación es específica de la API de renderizado
 - ❑ Subclases específicas para cada API de renderizado
 - ❑ P. Ej: D3DRenderSystem para Direct3D
- ❑ Normalmente no se debería manipular el objeto `ogre::RenderSystem` directamente
- ❑ Todo lo que necesario para renderizar objetos y personalizar configuraciones debería estar disponible en el `Ogre::SceneManager`, `Material` y otras clases orientadas a la escena.

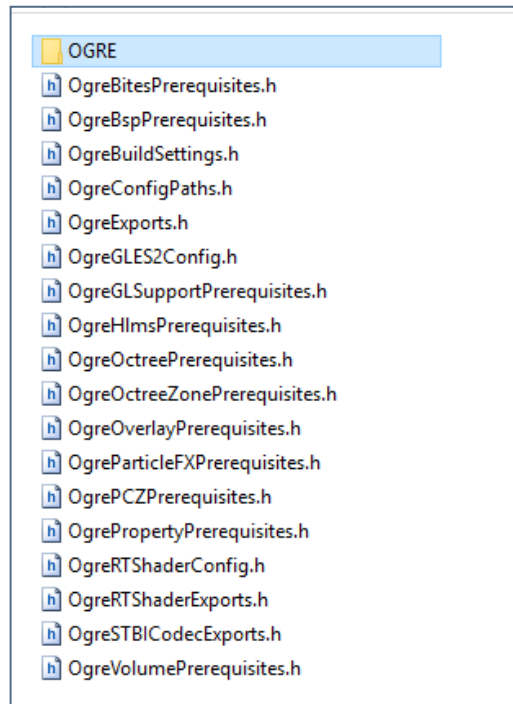
Proyecto-esqueleto OGRE3D

❑ **OgreSDK** es un grupo de herramientas para la programación de aplicaciones.

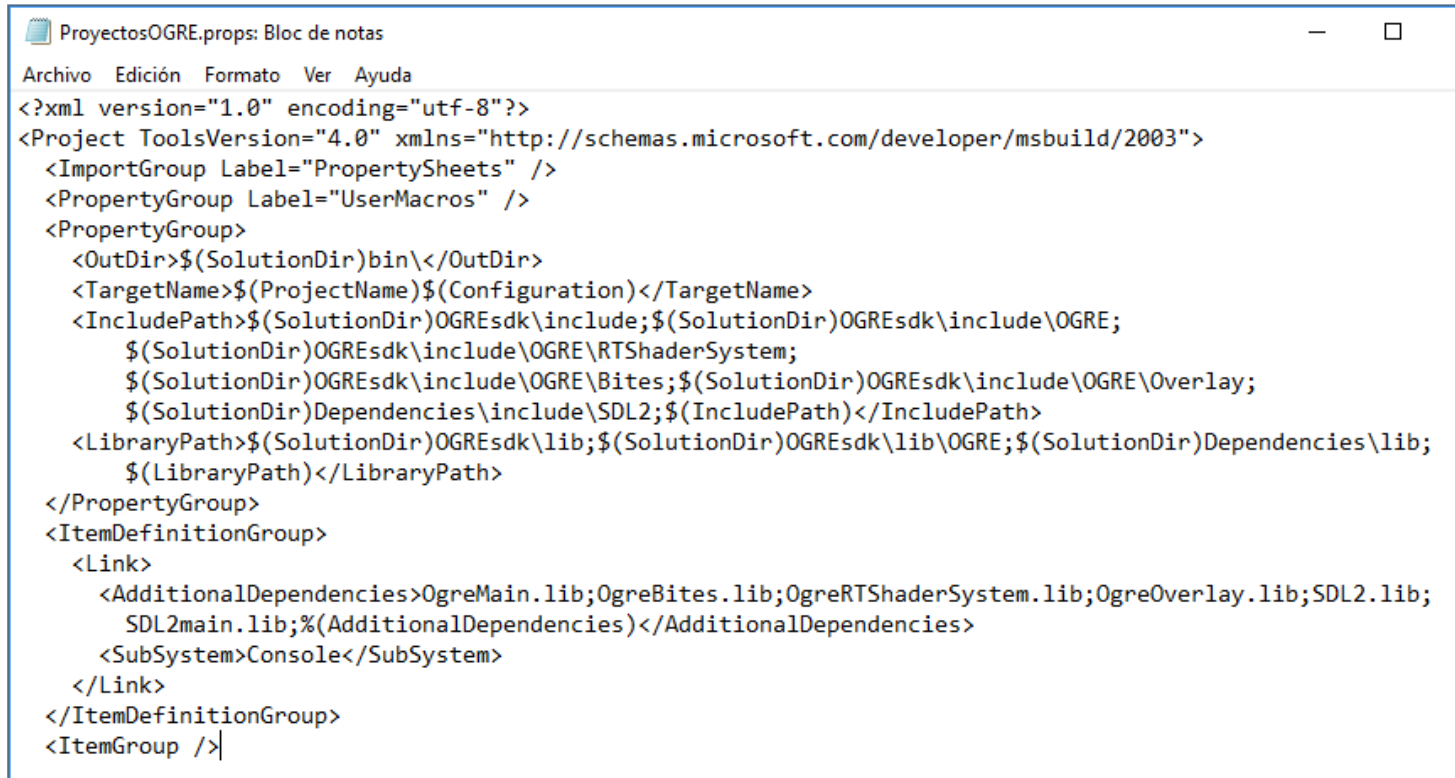
Recomiendan **CMake**



❑ OGREsdk\include



❑ ProyectosOGRE.props



```
ProyectosOGRE.props: Bloc de notas
Archivo Edición Formato Ver Ayuda
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <ImportGroup Label="PropertySheets" />
  <PropertyGroup Label="UserMacros" />
  <PropertyGroup>
    <OutDir>$(SolutionDir)bin\</OutDir>
    <TargetName>$(ProjectName)$(Configuration)</TargetName>
    <IncludePath>$(SolutionDir)OGREsdk\include;$(SolutionDir)OGREsdk\include\OGRE;
      $(SolutionDir)OGREsdk\include\OGRE\RTShaderSystem;
      $(SolutionDir)OGREsdk\include\OGRE\Bites;$(SolutionDir)OGREsdk\include\OGRE\Overlay;
      $(SolutionDir)Dependencies\include\SDL2;$(IncludePath)</IncludePath>
    <LibraryPath>$(SolutionDir)OGREsdk\lib;$(SolutionDir)OGREsdk\lib\OGRE;$(SolutionDir)Dependencies\lib;
      $(LibraryPath)</LibraryPath>
  </PropertyGroup>
  <ItemDefinitionGroup>
    <Link>
      <AdditionalDependencies>OgreMain.lib;OgreBites.lib;OgreRTShaderSystem.lib;OgreOverlay.lib;SDL2.lib;
        SDL2main.lib;%(AdditionalDependencies)</AdditionalDependencies>
      <SubSystem>Console</SubSystem>
    </Link>
  </ItemDefinitionGroup>
</ItemGroup />
```

❑ Utiliza utilidades de OgreBites:

- ❑ OgreApplicationContext
- ❑ OgreBitesConfigDialog
- ❑ OgreInput
- ❑ OgreTrays
- ❑ OgreCameraMan

❑ IG2App contiene las siguientes clases:

- ❑ **IG2ApplicationContext** (adaptación de **OgreApplicationContext**): crea **Root**, crea la ventana de renderizado, inicia los gestores,... Implementa **FrameListener** e informa de eventos de entrada a los observadores (objetos del tipo **InputListener***) suscritos.
- ❑ **IG2App**: hereda de **IG2ApplicationContext** e **InputListener**
- ❑ **main**: lanza el bucle de renderizado

- ❑ **media** contiene directorios para:
 - ❑ materials (texturas, scripts y shaders)
 - ❑ models (mallas)

- ❑ **media** contiene un directorio **IG2App** que reúne los elementos de materials models que usa nuestra aplicación. Inicialmente está vacío.

- ❑ **bin** contiene los tres archivos de configuración
 - ❑ ogre.cfg
 - ❑ resources.cfg
 - ❑ plugins.cfg

❑ OgreTrays (Overlay system)



- ❑ Permiten renderizar elementos 2D y 3D sobre el contenido normal de la escena
- ❑ Se crean efectos como pantallas de visualización (HUD), sistemas de menús, paneles de estado, etc.
 - ❑ El panel de estadísticas de velocidad de fotogramas que viene de serie con OGRE.
 - ❑ Pueden contener elementos 2D o 3D.
- ❑ Los elementos 2D se usan para HUDs
- ❑ Los elementos 3D pueden usarse para crear cabinas o cualquier otro objeto 3D que se desee renderizar sobre el resto de la escena.
- ❑ Se pueden crear superposiciones a través del método `Ogre::OverlayManager::create`
- ❑ También es posible definir las en un script [.overlay](#)

Setting up an OGRE project

- ❑ www.ogre3d.org

Download OGRE (versión $\geq 1.11.2$)

- ❑ github.com/OGRECave/ogre -> BuildingOgre.md

Utiliza **CMake** (cross platform make -> cmake.org/download), herramienta multiplataforma de generación de código, de más alto nivel que el sistema Make de Unix.

Compilador C++: VS2022, ...

Dependencias: SDL2 (ventana y eventos), FreeType, ...

Plugins: Octree, BSP, ParticleFX (archivo **plugins.cfg**)

APIs gráficos: GL y GL3Plus (archivo **ogre.cfg**)

- ❑ **OgreSDK** (Software Development Kit)