# Scripts

- **OGRE** permite el uso de **Scripts** para describir materiales.

  En lugar de escribir código C++ en la aplicación, podemos escribir en archivos independientes en el lenguaje de script. Ogre ejecutará automáticamente el código necesario a partir de la información del archivo.

  De esta forma, modificar un material no requiere modificar código de la aplicación y, por tanto, no es necesario recompilar. Además se pueden reutilizar fácilmente.

- Durante el proceso de inicialización, Ogre analiza todos los archivos que aparecen en **resources.cfg**, pero no los carga. La carga se realiza al crear las entidades que los utilizan.

- Todos los recursos son compartidos.

- Los archivos de mallas **.mesh** pueden contener un enlace (**nombre del material**) al material de cada submalla.

# Script para el material

- Un material puede constar de varias técnicas para adecuar el renderizado a las capacidades de la plataforma. Si la GPU no soporta la primera técnica, se comprueba la segunda y así sucesivamente
- Las técnicas también se pueden usar para especificar el LOD (*Level of Detail*)

# Script para el material

- Cada técnica describe características sobre:
  - **Propiedades del material respecto a la incidencia de la luz en la entidad (componentes ambiente, difusa y especular).**
  - **Propiedades del material respecto a las unidades de textura.**

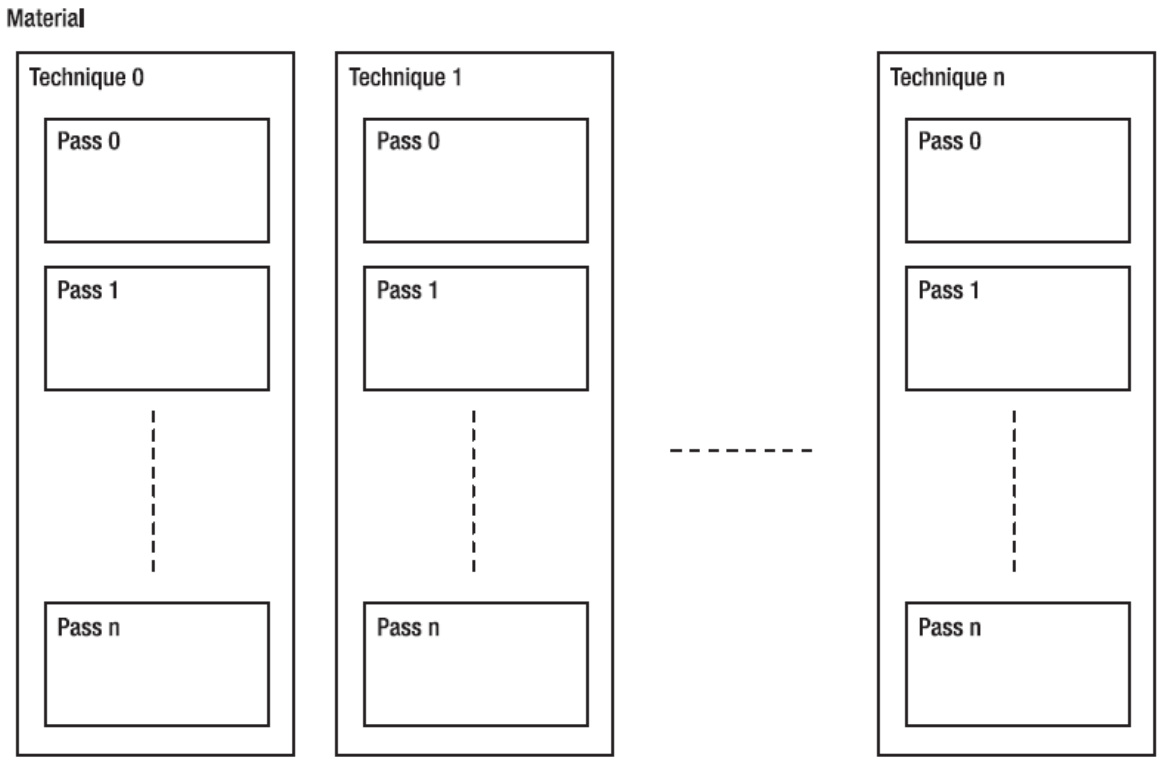
  Y **los shaders** (GPU programs) que se deben utilizar para el renderizado. Por defecto se utilizan los shaders de la *fixed-function pipeline*, del sistema RTTS (incluye Fixed Function emulation)
- Cada técnica puede constar de varias pasadas. Algunos efectos requieren renderizar el objeto varias veces (pasadas) para realizar una composición. Por ejemplo, en el caso de desenfoques (*motion blur*).

# Script para el material

- Los scripts de materiales son archivos de texto con extensión **.material**. Un archivo puede contener varios materiales.

  **Cada material** tiene que tener un **nombre único** y una o varias técnicas. Cada técnica puede constar de una o varias pasadas.

  Para asociar un material a una **entidad**: **setMaterialName(name)**



# Ejemplos de scripts

## ❑ Ejemplo:

```
// This is a comment

material VerySimple // nombre
{
   technique // al menos una
   {
      pass  // al menos uno
      {
         diffuse 0.5 0.5 0.5
      }
   }
}
```

```
material NotQuiteAsSimple {
   technique  {  // first, preferred technique
      pass {  // first pass
         diffuse 0.5 0.5 0.5
         ambient 0.1 0.2 0.3
         specular 0.8 0.8 0.8 68
         texture_unit {
            texture ReallyCool.jpg
            colour_op modulate
         }
      }
   }
   technique {  // Second technique
      pass {
         diffuse 0.5 0.5 0.5
      }
   }
}
```

# Archivo Practica1.material

- El archivo para definir los materiales de la Práctica 1 podemos llamarlo **`Practica1.material`**
- Este archivo se sitúa de forma que sea accesible según las directivas especificadas en el archivo **`resources.cfg`**
- Este archivo debe "compilar", en particular debe estar equilibrado en llaves abiertas y cerradas
- Es buena política situar los archivos de material y las mallas y texturas que se usen en el proyecto, todos juntos en el directorio **`media/IG2App`** (que quizás lo tengas aún vacío)
- Cada material lleva un nombre (no uses ñ) que es único entre todos los nombres que aparecen en todos los archivos **`.material`** del proyecto. Una forma de hacerlo es llamarlo así: **`nombre_del_archivo/nombre_del_material`**
- Ejemplo:

```
material Practica1/plano {…}
```

# Archivo Practica1.material

- Cada material tiene que tener, al menos, una técnica y cada técnica puede tener un nombre (único) o ser indexada por su posición en el texto
- Cada técnica tiene que tener, al menos, una pasada
- **Para especificar las características de las texturas** hay que tener en cuenta que:
  - Las imágenes de las texturas que se usan en la Práctica 1 están en el directorio **`media/materials/textures`**
  - Se puede especificar solamente la textura, escribiendo el nombre de su archivo con la extensión, tras la palabra "**`texture_unit`**". Por ejemplo:

```
texture_unit {
   texture agua.jpg
}
```

# Archivo Practica1.material

- Antes de la textura se puede especificar un color
- Para decir cómo se combina la textura con este posible color de la entidad se utiliza el atributo "**color_op**" seguido de una de varias opciones. Las más usadas son **color_op add** y **color_op modulate**. El significado de estas opciones es el mismo que en OpenGL. En el primer caso se suman y en el segundo se multiplican. Por ejemplo:

```
texture_unit {
   texture agua.jpg
   colour_op add
}
```

# Archivo Practica1.material

- Si se quiere movimiento en la textura se especifica el atributo **scroll_anim**. Con **scroll_anim 0.1 0.0** se mueve la textura a lo largo del eje **X**. Cambiando el valor **0.1**, se puede mover la textura a lo largo del eje **Y**, o a lo largo de los dos. Por ejemplo:

```
texture_unit {
   texture agua.jpg
   colour_op add
   scroll_anim -0.1 0.0
}
```

- Si se quiere aplicar una textura a una entidad esférica puedes usar el atributo **env_map**. Con **env_map spherical** se generan coordenadas de textura de forma automática al precio de arrastrar un efecto billboard

# Archivo Practica1.material

- Los atributos principales de los pases relacionados con el color y sus valores por defecto son los siguientes:
  - `ambient 1.0 1.0 1.0 1.0  // default`
  - `diffuse 1.0 1.0 1.0 1.0  // default`
  - `specular 0.0 0.0 0.0 0.0 0.0    // default`

Ejemplo final:

```
material Practica1/cabeza {
  technique {
    pass {
      diffuse 0.5 0.5 0.5
      texture_unit {
         texture checker.jpg
         colour_op add
         env_map spherical
      }
    }
  }
}
```

# Incidencias

- Se escribe la instrucción:

```
ent->setMaterialName(“Mat/azulete");
```

y el archivo **Mat.material** no existe o la entrada **azulete** no aparece en el archivo. Entonces el proyecto compila, pero la entidad **ent** sale gris

- El archivo **Mat.material** contiene dos entradas

```
material Mat/azulete { … }
```

Entonces el proyecto compila, pero en ejecución se lanza una excepción de puntero nulo. La ventana de ejecución lo explica: **Resource with the name Mat/azulete already exists**

- La entrada tiene declaraciones redundantes o distintas:

```
material Mat/azulete
{technique{pass{diffuse 0.0 0.8 1.0
                diffuse 1.0 0.8 1.0}}}
```

No hay problema. En este caso se queda con la última (comp. difusa)

- El archivo **Mat.material** no compila. Por ejemplo, porque no está equilibrado en paréntesis al faltarle un paréntesis de cierre. Entonces el proyecto compila, pero en ejecución se lanza una excepción de puntero nulo. La ventana de ejecución lo explica: **no matching closing bracket '}' for open bracket '{' at line 47 in …**