Informática Gráfica II

Object-oriented Graphics Rendering Engine

Scripts: Definición de materiales

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

Script para definir materiales

- **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.
  - Modificar un material **no requiere modificar código de la aplicación.**
    - No es necesario recompilar.
    - 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 definir materiales

- Un material puede constar de varias **técnicas** para adecuar el renderizado a las capacidades de la plataforma.
- Una técnica es una forma de conseguir el **efecto** deseado.
- Es possible proporcionar más de una técnica para disponer de enfoques alternativos
  - Si la tarjeta gráfica no tiene la capacidad de renderizar la técnica preferida
  - Se desea definir versiones del material con menor nivel de detalle
- Las técnicas también se pueden usar para especificar el LOD (*Level of Detail*)
- Cada técnica describe características sobre:
  - Propiedades del material respecto a la incidencia de la luz en la entidad
    - Como las componentes ambiente, difusa y especular
  - Propiedades del material respecto a las unidades de textura.
- Cada técnica consta de uno o varios **pases** (pass).
  - Si la GPU no soporta la primera técnica, se comprueba la segunda y así sucesivamente

Script para definir materiales

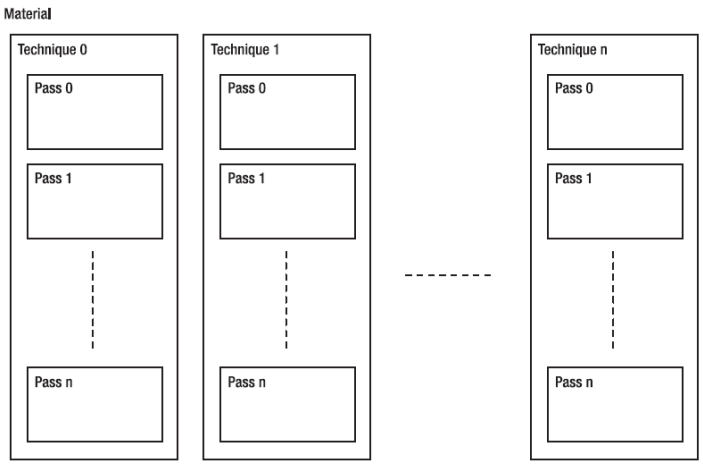
- Ogre también puede dividir los pases definidos en otros (con mayor cantidad de) pases en tiempo de ejecución.
  - Si un pase utiliza demasiadas unidades de textura para la tarjeta en la que se está ejecutando actualmente
- Un pase es **un único renderizado** de la geometría en cuestión
  - Una única llamada a la API de renderizado con un determinado conjunto de propiedades de renderizado
- Cada pase tiene una serie de atributos de nivel superior.
  - P. Ej. *ambiente*, para establecer la cantidad y el color de la luz ambiente reflejada por el material.
- Dentro de cada pase, puede haber cero o varias Unidades de Textura (`texture_unit`)

Script para definir materiales

- **Los shaders** (GPU programs) se deben utilizar para el renderizado.
  - Por defecto se utilizan los shaders de la *fixed-function pipeline*, del sistema RTTS (incluye Fixed Function emulation)
  - Los veremos en temas posteriores.
- Algunos efectos requieren renderizar el objeto varias veces (pass) para realizar una composición.
  - Por ejemplo, en el caso de desenfoques (*motion blur*).
- Cuando se utiliza un material por primera vez, se "compila".
  - Esto implica escanear las técnicas que se han definido y marcar cuáles de ellas son compatibles con la API de renderizado y la tarjeta gráfica.
  - Si ninguna técnica es compatible, el material se renderizará en blanco ☹

Script para definir materiales

- 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 uno o varios pases.
    - Las técnicas deben enumerarse por orden de preferencia
    - Las que aparecen antes tienen preferencia sobre las que aparecen posteriormenente.
    - Las técnicas más avanzadas y exigentes se enumeran en primer lugar
- Para asociar un material a una **entidad**: `setMaterialName(name)`
  - Las técnicas no nombradas adoptarán un nombre que será el índice de la técnica.
  - Por ejemplo: la primera técnica en un material es el índice 0: su nombre sería "0"



Atributos de los pases (pass)

- **`ambient`**
  - Establece las propiedades de reflectancia del color ambiente
  - `ambient (<red> <green> <blue> [<alpha>]| vertexcolour)`
    - Los valores de color válidos están entre 0.0 y 1.0.
    - Determina cuánta luz ambiental (luz global sin dirección) se refleja.
    - El valor por defecto es blanco total, lo que significa que los objetos están completamente iluminados globalmente.
    - Para ver efectos de luz difusa o especular es necesario reducer este valor.
    - Por defecto: `ambient 1.0 1.0 1.0 1.0`
- **`diffuse`**
  - Establece las propiedades de reflectancia del color difuso
  - `diffuse (<red> <green> <blue> [<alpha>]| vertexcolour)`
    - Los valores de color válidos están entre 0.0 y 1.0.
    - Determina cuánta luz difusa (luz de instancias de la clase Luz) se refleja.
    - El valor por defecto es blanco total, lo que significa que los objetos reflejan la máxima luz blanca que pueden de los objetos Luz.
    - Por defecto: `diffuse 1.0 1.0 1.0 1.0`

Atributos de los pases (pass)

- **`specular`**
  - Establece las propiedades de reflectancia del color especular
  - `specular (<red> <green> <blue> [<alpha>]| vertexcolour) <shininess>`
    - Los valores de color válidos están comprendidos entre 0,0 y 1,0.
    - El brillo (shininess) puede ser cualquier valor mayor que 0.
    - Determina cuánta luz especular (reflejos de instancias de la clase Luz) se refleja.
    - Por defecto no se refleja ninguna luz especular.
    - El color de los reflejos especulares está determinado por los parámetros de color, y el tamaño de los reflejos por el parámetro separado de brillo.
    - Cuanto mayor sea el valor del parámetro de brillo, más nítido será el reflejo.
    - Cuidado con utilizar valores de brillo en el rango de 0 a 1 ya que esto hace que el color especular se aplique a toda la superficie que tiene el material aplicado.
    - Cuando el ángulo de visión de la superficie cambia, también se producen parpadeos cuando el brillo está en el rango de 0 a 1. Los valores de brillo entre 1 y 128 funcionan mejor tanto en DirectX como en OpenGL
    - Por defecto: `specular 0.0 0.0 0.0 0.0 0.0`

Sección texture_unit

- Para especificar las características de las texturas hay que tener en cuenta que:
  - Las imágenes de las texturas están en el directorio `media/materials/textures`
  - Se puede especificar solamente la textura, escribiendo el nombre de su archivo con la extensión, tras la sección **texture_unit**
    - Por ejemplo:

```
texture_unit {

    texture agua.jpg

}
```

  - Este parámetro es mutuamente excluyente con el atributo `anim_texture`.
  - Tened en cuenta que el nombre del archivo de la textura no puede incluir espacios.
  - ¡Ojo con las mayúsculas en las extensiones! (Es case sensitive)

Sección texture_unit

- Para combinar la textura con un color de la entidad se utiliza el atributo color_op seguido 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.
    - **color_op add**: se suman las componentes
    - **color_op modulate**: se multiplican las componentes

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

Sección texture_unit

- Si se quiere movimiento en la textura se especifica el atributo **scroll_anim**
- Útil para crear efectos de desplazamiento constante en una textura
- `scroll_anim <uSpeed> <vSpeed>`
  - *uSpeed*: Número de bucles horizontales por segundo
    - +u=moverse a la derecha, -u = moverse a la izquierda
  - *vSpeed*: Número de bucles verticales por segundo
    - +v=moverse hacia arriba, -v= moverse hacia abajo

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

- Para aplicar una textura a una entidad esférica se debe usar el atributo **env_map**
  - Con `env_map spherical` se generan coordenadas de textura de forma automática al precio de arrastrar un efecto *billboard*

Ficheros .material

- Los ficheros donde se definen los materiales tendrán nombres `nombreFichero.material`
- Estos ficheros se sitúan de forma que sea accesible según las directivas especificadas en el archivo `resources.cfg` (como ocurre con las mallas)
- Los ficheros deben "compilar", en particular deben estar equilibrados en llaves abiertas y cerradas
- Es buena política situar los archivos de material, las mallas y texturas que se usen en el proyecto, todos juntos en el mismo directorio
  - Por ejemplo, en la práctica, en **`media/IG2App`**
- Cada material lleva un nombre (no uses ñ ni tildes)
  - Debe ser único entre todos los nombres que aparecen en todos los archivos **`.material`** del proyecto.
  - Una forma de evitar colisiones de nombres es llamarlos: nombreArchivo/nombreMaterial
- Ejemplo:

```
material Practica1/plano {…}
```

Ejemplos de scripts de materiales

Ejemplo:

```
// This is a comment

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

Incidencias

- Se escribe la instrucción `ent->setMaterialName("Mat/azulete");`
  - 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
  - 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)

Incidencias

- 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 …*