

# **Object-oriented Graphics Rendering Engine**

## **Sistema de Partículas**

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

# Sistema de partículas

- ❑ Son sistemas dinámicos basados en una colección de elementos móviles (**particles**) generados por un emisor (**emitter**).
  - ❑ Su comportamiento físico lo determinan parámetros como la velocidad, la dirección, el tiempo de vida, . . .
- ❑ Durante el tiempo de vida pueden (opcionalmente) sufrir cambios mediante **affectors** (viento, gravedad, ...)
- ❑ Representación gráfica (renderización).
  - ❑ En Ogre, por defecto, con **billboards** (carteles o vallas publicitarias)
- ❑ Se pueden crear y configurar completamente en código, pero es habitual tenerlos en scripts (archivos de extensión **.particle**) que se pueden generar con un editor gráfico

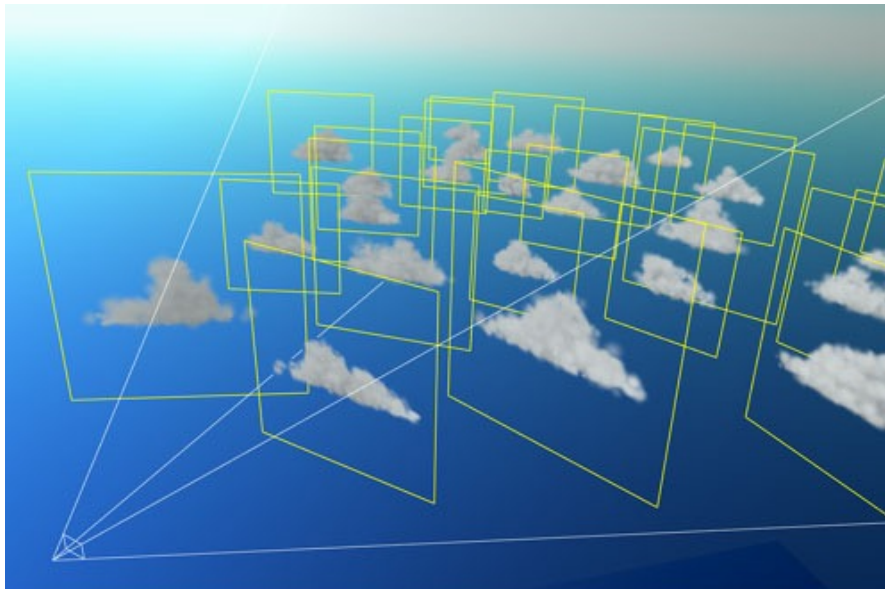
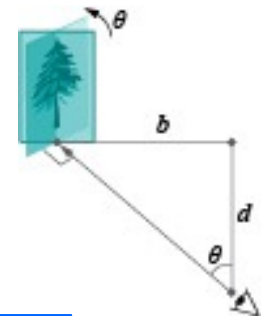


## ❑ Billboard: carteles o vallas publicitarias

Panel (rectángulo) que se orienta hacia la cámara (cada frame).

En Ogre, por motivos de eficiencia, se gestionan en grupos: **BillboardSet**

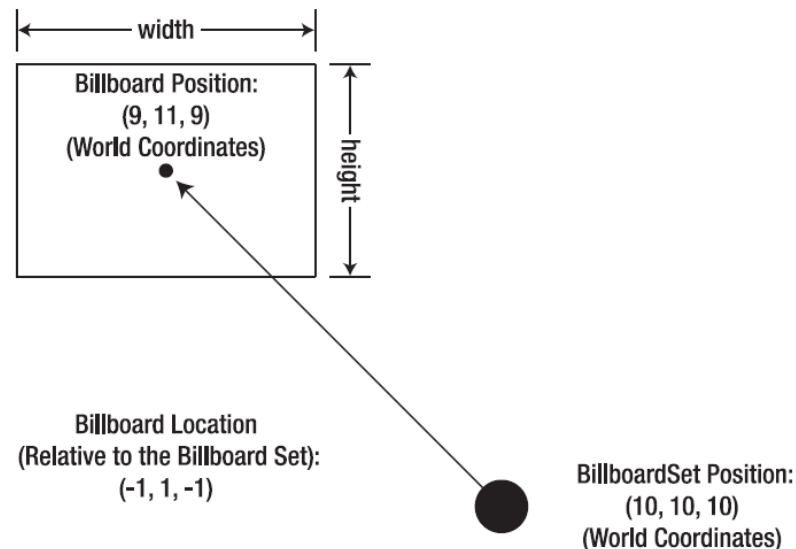
No podemos crear un **Billboard** de forma independiente, debe pertenecer a un conjunto (Set)



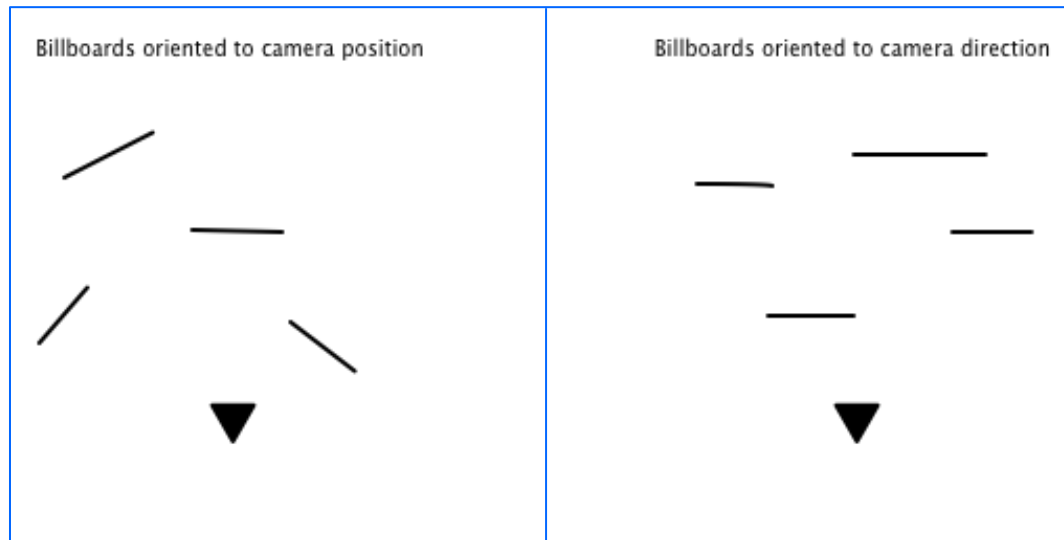
Permiten simular varios efectos: texto, botones, vegetales (impostores), sistemas de partículas (humo, estelas, ...)

**Impostores:** Elementos visibles en la escena que no han sido renderizados con su malla. P. ej. Entornos con vegetación.

- ❑ La clase **BillboardSet** hereda de **MovableObject** y **Renderable**
  - ❑ Todos los **billboards** de un grupo (**BillboardSet**) tienen que tener el mismo tamaño y material.
  - ❑ Las posiciones de cada **billboard** son relativas a la posición del conjunto.
  - ❑ El conjunto se trata como un único objeto.

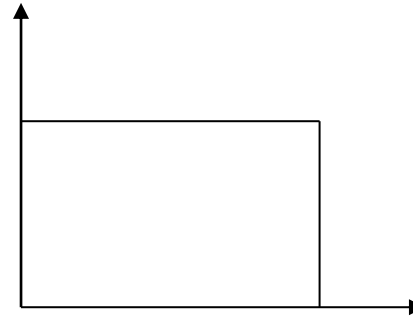
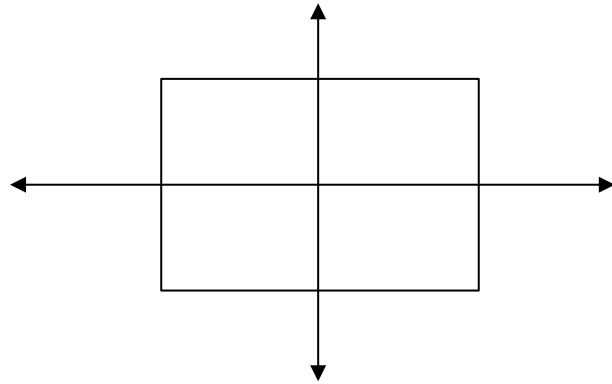


- ❑ Se pueden orientar de distintas formas:

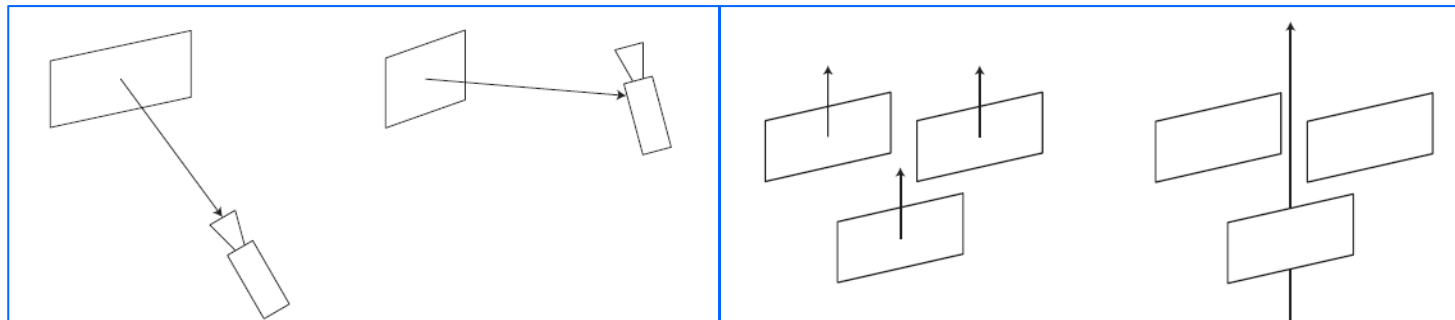


- Podemos configurar el sistema local, la forma de orientarse, ...

**billboard\_origin:** center | bottom\_left | ...



**billboard\_type:** point | oriented\_self | oriented\_common



Default: point - center

## ❑ Para crear el conjunto (set)

- ❑ Establecer el nombre, el número de elementos, dimensiones, material, ...

```
BillboardSet* bbSet = mSM->createBillboardSet(NameBS, MaxEls);
```

```
bbSet->setDefaultDimensions(w, h);
```

```
bbSet->setMaterialName (...);
```

nombre para el conjunto

máximo nº de elementos

## ❑ Para colocar el BillboardSet en el grafo de la escena

```
node->attachObject(bbSet);
```

## ❑ La posición del nodo será la del BillboardSet

## ❑ Para crear los elementos del conjunto: establecer su posición

### ❑ Posición relativa al grupo (nodo)

```
Billboard* bb = bbSet->createBillboard(Vector3(x, y, z));
```

# Ejemplo para crear el efecto *billboard*

## ❑ Ejemplo de script para crear efecto *billboard*

Desactiva la iluminación dinámica.  
El objeto se ilumina aunque no haya luz

Cuando las coordenadas de la textura superan 1.0, se fijan a 1.0.

```
material IG2App/Panel
{
    technique
    {
        pass
        {
            lighting off
            texture_unit
            {
                texture FicheroTextura
                tex_address_mode clamp
            }
        }
    }
}
```

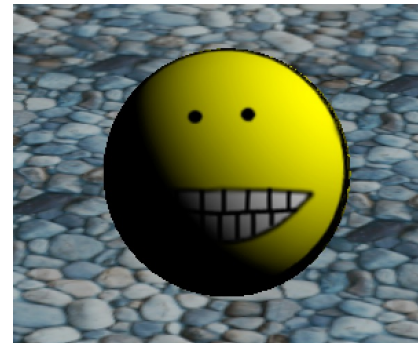
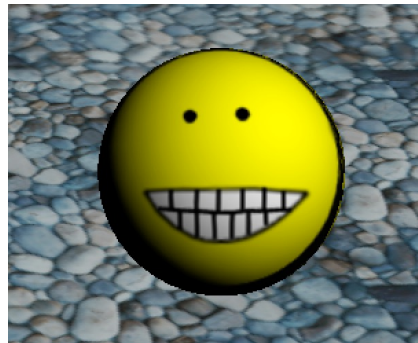


# Ejemplo para crear el efecto *billboard*

- ❑ Ejemplo con la textura "spotlight\_image.png" aplicada a una esfera

```
material example/esferaSmile{
    technique {
        pass {
            texture_unit {
                texture spotlight_image.png
                env_map spherical
            }
        }
    }
}
```

```
Entity * sphereEnt = mSM->createEntity("sphere.mesh");
SceneNode* sphereNode =
    mSM->getRootSceneNode()->createChildSceneNode();
sphereNode->attachObject(sphereEnt);
sphereNode->setPosition(Vector3(0, 100, 0));
sphereEnt->setMaterialName("example/esferaSmile");
```



Aunque rote la cámara, siempre se orienta la textura

# Ejemplo para crear el efecto *billboard*

- ❑ Ejemplo con la textura "10 points.png" aplicada en un *billboardSet*

```
material example/board{
    technique {
        pass{
            lighting off
            texture_unit {
                texture 10points.png
                tex_address_mode clamp
            }
        }
    }
}
```

```
SceneNode* bbNode =
    mSM->getRootSceneNode()->createChildSceneNode();
BillboardSet* bbSet = mSM->createBillboardSet(1);
bbSet->setDefaultDimensions(80, 40);
bbSet->setMaterialName("example/board");
bbNode->attachObject(bbSet);
bbSet->createBillboard({ 0, 50, -50});
```



Aunque rote la cámara, siempre se orienta la textura

- ❑ Los scripts de partículas permiten definir los sistemas de partículas que se van a instanciar en el código
  - ❑ No es necesario codificar los ajustes en el código fuente
  - ❑ Permite realizar cambios muy rápidamente.
  - ❑ Pueden crearse múltiples sistemas reales en tiempo de ejecución.
- ❑ Una vez que los scripts han sido *analizados*
  - ❑ Se pueden instanciar sistemas basados en ellos
  - ❑ Para ello, se usa el método `Ogre::SceneManager::createParticleSystem()`
  - ❑ Como parámetro recibe el nombre para el nuevo sistema o el nombre de sistema en el script.
- ❑ La clase **ParticleSystem** hereda de **MovableObject**
  - ❑ Necesita un emisor y un renderizador

# Sistema de partículas

- ❑ Para indicar que el sistema de partículas, las emita / pare de emitirlas:

```
pSys->setEmitting(true|false);
```

- ❑ Para eliminar las partículas de forma inmediata

```
pSys->clear();
```

- ❑ Ejemplo: crear un sistema a partir de un script (archivo **.particle**)

```
ParticleSystem* pSys = mSM -> createParticleSystem("psSmoke", "example/smoke");
```

```
pSys->setEmitting(false);
```

```
mPSNode->attachObject(pSys);
```

Nombre para el sistema

Nombre de la entrada en el script **.particle**

La posición y la dirección de emisión son relativas al nodo.

# Sistema de partículas (Material para simular humo)

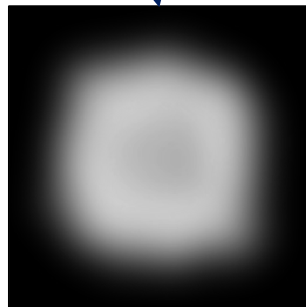
```
material example/smoke{
    technique{
        pass{
            lighting off
            scene_blend alpha_blend
            depth_write off
            diffuse vertexcolour

            texture_unit{
                texture smoke.png
                tex_address_mode clamp
            }
        }
    }
}
```

Establece el tipo de mezcla (blend) que este pase tiene con el contenido existente de la escena. El valor alfa de la salida de renderizado se utiliza como mascara.

Establece si este pase renderiza con la escritura en el buffer de profundidad activada o no. En este caso (off) los píxeles se escriben sin actualizar el buffer de profundidad. Se utiliza cuando se renderiza una colección de objetos transparentes al final de una escena para que se solapen entre sí correctamente.

Hace que la reflectancia difusa siga el color del vértice definido en la malla en lugar de los valores de color.



smoke.png

# Sistema de partículas (Ejemplo de script .particle)

- ❑ Puede tener atributos de nivel superior
  - ❑ Rendered, quota, material, . . .
- ❑ Los *emitter* (que crean partículas) y los *affector* (modifican partículas) se añaden como definiciones anidadas dentro del script.
  - ❑ Los parámetros dependen totalmente del tipo de *emitter* / *affector*.

```
particle_system example/smokeParticle{  
  
    billboard_type    point  
    particle_width    35  
    particle_height   35  
    quota             500  
    material          example/smoke  
  
    emitter Point{  
        // Configuración del emisor  
    }  
  
    affector ColourFader{  
        // Configuración del modificador  
    }  
  
    affector Scaler{  
        // Configuración del modificador  
    }  
}
```

Podéis consultar la lista completa de parámetros en:

[https://ogrecave.github.io/ogre/api/latest/\\_particle-\\_scripts.html](https://ogrecave.github.io/ogre/api/latest/_particle-_scripts.html)

# Atributos utilizados en cualquier sistema de partículas

## ❑ `renderer`

- ❑ Los sistemas de partículas no se renderizan a sí mismos
  - ❑ Lo hacen a través de clases `ParticleRenderer`.
- ❑ El *renderer* por defecto en Ogre3D está basado en `billboards`
  - ❑ Es posible añadir más mediante *plugins*.
- ❑ Los *renderers* de partículas se registran con un nombre único
- ❑ Se puede usar ese nombre para determinar el *renderer* a usar.

Formato: **renderer** <nombre\_renderer>

Por defecto: `billboard`

## ❑ `quota`

- ❑ Número máximo de partículas.
- ❑ Una vez se alcanza el límite, los emisores no podrán emitir más partículas hasta que se destruyan.

Formato: **quota** <partículas\_max>

Por defecto: `10`

# Atributos utilizados en cualquier sistema de partículas

## ❑ material

- ❑ Establece el nombre del material que utilizarán todas las partículas de este sistema.
- ❑ Todas las partículas de un sistema utilizan **el mismo material**
  - ❑ Cada partícula puede matizar este material mediante el uso de su propiedad de color.

Formato: **material** <nombre\_material>

Por defecto: none (material en blanco)

## ❑ iteration\_interval

- ❑ Normalmente, los sistemas de partículas se actualizan en función del ratio de *frames*.
- ❑ Esto puede dar resultados variables (particularmente en ratios bajos).
- ❑ Esta opción hace que la frecuencia de actualización sea un intervalo fijo
  - ❑ A velocidades de fotogramas bajas, la actualización de partículas se repetirá en el intervalo fijo hasta que se agote el tiempo de fotogramas.
  - ❑ Un valor de 0 significa la iteración de tiempo utilizando el ratio de *frames* actual.

Formato: **iteration\_interval** <secs>

Por defecto: 0



# Atributos utilizados en cualquier sistema de partículas

## ❑ `particle_width`

- ❑ Establece la anchura de las partículas en coordenadas globales.
- ❑ Es absoluta cuando `billboard_type` es `'point'` o `'perpendicular_self'`
- ❑ Se escala por la longitud del vector de dirección cuando `billboard_type` es `'oriented_common'`, `'oriented_self'` o `'perpendicular_common'`.

Formato: `particle_width` <ancho>

Por defecto: 100

## ❑ `particle_height`

- ❑ Establece la altura de las partículas en coordenadas globales.
- ❑ Tiene las mismas propiedades en cuanto al tipo de `billboard_type`

Formato: `particle_height` <alto>

Por defecto: 100

# Atributos utilizados en cualquier sistema de partículas

## ☐ `local_space`

- ☐ Por defecto, las partículas se emiten en el espacio de coordenadas global (world)
- ☐ Si se transforma el nodo al que está unido el sistema, no afectará a las partículas
  - ☐ Pero sí a los emisores.
- ☐ Para crear algunos efectos es posible que desee que las partículas permanezcan unidas al espacio local en el que se encuentra el emisor y que las sigan directamente.

Formato: **local\_space** <true|false>

Por defecto: false

## ☐ `sorted`

- ☐ Por defecto, las partículas no están ordenadas.
- ☐ Estableciendo este atributo a true, las partículas se ordenarán con respecto a la cámara
  - ☐ La más lejana primero.
- ☐ Puede mejorar ciertos efectos de renderizado con un pequeño coste de ordenación.

Formato: **sorted** <true|false>

Por defecto: false

# Atributos utilizados en cualquier sistema de partículas

## ☐ `cull_each`

- ☐ Todos los sistemas de partículas están delimitados mediante la *bounding box* que contiene todas las partículas del sistema.
- ☐ Esto suele ser suficiente para los sistemas de partículas locales
  - ☐ La mayoría de las partículas son visibles, o no visibles, juntas.
- ☐ Para sistemas que extienden las partículas sobre un área más amplia (lluvia), es posible que se desee eliminar cada partícula individualmente.
  - ☐ Se ahorra tiempo de cómputo
  - ☐ Es mucho más probable que sólo un subconjunto de las partículas sean visibles.
  - ☐ Para ello, se debe el parámetro `cull_each` a `true`.

Formato: `cull_each` <true|false>

Por defecto: `false`

# Atributos utilizados en el renderizador Billboard

## ❑ billboard\_type

- ❑ Hay varias formas de orientar *billboards*.
- ❑ El enfoque clásico es que el *billboard* mire directamente a la cámara:
  - ❑ Es el comportamiento por defecto.
- ❑ En ocasiones es necesario configurar que cada partícula tenga una orientación propia

Formato: **billboard\_type** <tipo>

Por defecto: point

### point

Las partículas siempre miran completamente a la cámara.

### oriented\_common

Las partículas se orientan alrededor de un vector de dirección **común**, que actúa como su **eje Y local**. Útil para tormentas de lluvia, campos de estrellas, etc,

### oriented\_self

Las partículas se orientan alrededor de su **propio** vector de dirección, que actúa como su **eje Y local**. Útil para efectos láser, fuegos artificiales, . . .

### perpendicular\_common

Las partículas son perpendiculares a un vector de dirección **común**, que actúa como su **eje Z local**, y su **eje Y local coplanario** con la dirección común y el vector común hacia arriba.

Útil para aureolas, anillos, etc donde las partículas serán perpendiculares al suelo.

### perpendicular\_self

Las partículas son perpendiculares a su **propio** vector de dirección, que actúa como su **eje Z local**, y su **eje Y local coplanario** con su propio vector de dirección y el vector común hacia arriba. Bueno para la pila de anillos, etc, donde las partículas se perpendicular a su dirección de desplazamiento.

# Atributos utilizados en el renderizador Billboard

## ❑ billboard\_origin

- ❑ Esta opción controla el ajuste de dónde aparece un *billboard* en relación a su posición.
  - ❑ Por ejemplo, el centro para una esfera, o el borde inferior, para un árbol

Formato: **billboard\_origin**

<top\_left|top\_center|top\_right|center\_left|center|center\_right|bottom\_left|bottom\_center|bottom\_right>

Por defecto: center

## ❑ billboard\_rotation\_type

- ❑ Por defecto, las partículas del *billboard* rotarán las coordenadas de la textura.
- ❑ Esto produce que las esquinas del *billboard* se llenarán con un área de textura no deseada.
- ❑ Esta configuración te permite especificar otro tipo de rotación.

Formato: **billboard\_rotation\_type** <vertex|texcoord>

Por defecto: texcoord

### vertex

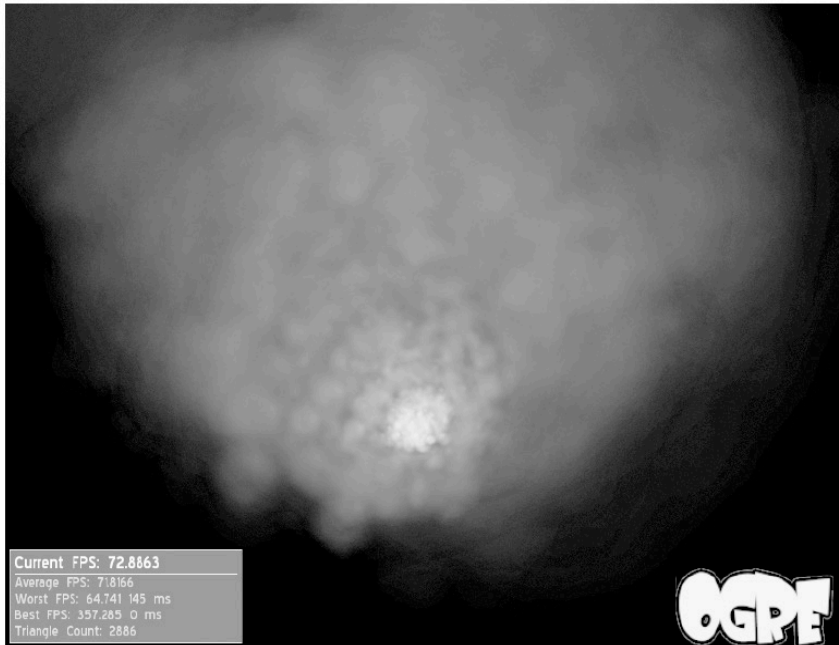
Las partículas del *billboard* rotarán los vértices alrededor de su dirección de orientación de acuerdo con la rotación de la partícula. Garantiza que las esquinas de la textura coincidan exactamente con las esquinas del *billboard*.

### texcoord

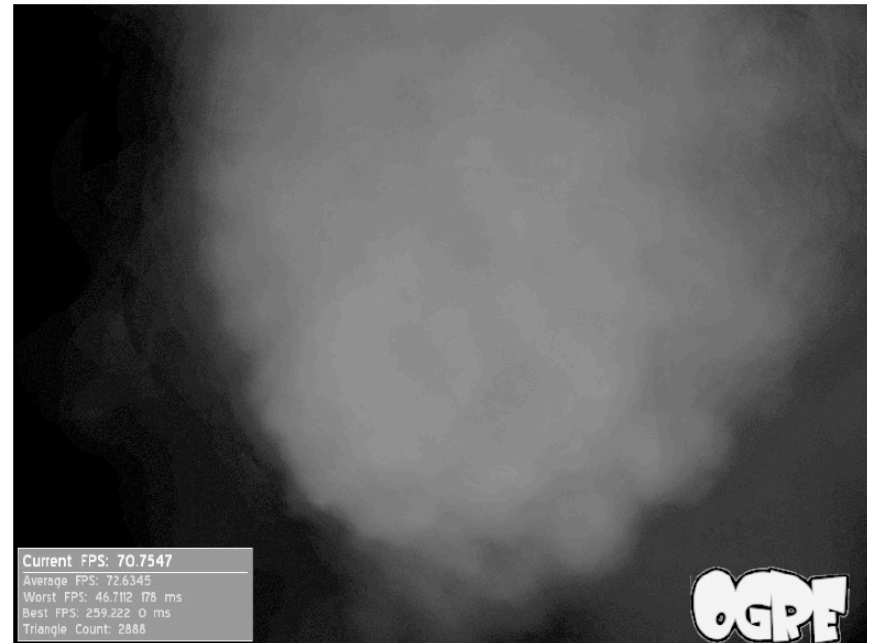
Las partículas del *billboard* rotarán las coordenadas de la textura de acuerdo con la rotación de la partícula. Rotar las coordenadas de la textura es más rápido que rotar los vértices, pero tiene algunas desventajas mencionadas anteriormente.

# Ejemplo de sistema de partículas (ordenación)

- ❑ Ejemplo de la ordenación de las partículas con respecto a la distancia a la cámara



Sorting= false



Sorting=true

- ❑ Los emisores de partículas se clasifican por **tipo**
  - ❑ Los emisores **Point** emiten desde un único punto
  - ❑ Los emisores **Box** emiten aleatoriamente desde un área.
- ❑ Se pueden añadir nuevos emisores a Ogre creando *plugins*.
- ❑ Actualmente Ogre soporta emisores de tipo
  - ❑ Point
  - ❑ Box
  - ❑ Cylinder
  - ❑ Ellipsoid
  - ❑ HollowEllipsoid
  - ❑ Ring

# Atributos de los Emisores (Emitters)

## ❑ angle

- ❑ Establece el ángulo máximo (en grados) con el que las partículas emitidas pueden desviarse de la dirección del emisor.

Formato: **angle** <grados>  
Por defecto: 0

## ❑ colour

- ❑ Establece un color para todas las partículas emitidas. Se puede utilizar `colour_range_start` y `colour_range_end` para configurar rangos de colores. El formato de color es <r g b [a]>, donde cada componente es un valor entre 0 y 1, siendo la componente alfa opcional.

Formato: **colour** <r> <g> <b> [<a>]  
Por defecto: 1 1 1 1

0: Transparente  
0.5: Translúcido  
1: Opaco

## ❑ direction

- ❑ Establece la dirección del emisor (relativo al SceneNode al que está unido el Sistema)

Formato: **direction** <x> <y> <z>  
Por defecto: 1 0 0

## ❑ position

- ❑ Establece la posición del emisor relativa al SceneNode al que está unido el sistema.

Formato: **position** <x> <y> <z>  
Por defecto: 0 0 0



# Atributos de los Emisores (Emitters)

## ❑ velocity

- ❑ Establece una velocidad constante para todas las partículas en el momento de la emisión.
- ❑ Con los atributos `velocity_min` y `velocity_max`, se puede establecer un intervalo de velocidades en lugar de una fija.

Formato: **velocity** <unidades\_por\_segundo>

Por defecto: 1

## ❑ time\_to\_live

- ❑ Establece el número de segundos que cada partícula vivirá antes de ser destruida.
- ❑ Es posible que los modificadores (`affectors`) de partículas alteren este valor.
- ❑ Con los atributos `time_to_live_min` y `time_to_live_max`, se puede establecer un intervalo de vida en lugar de uno fijo.

Formato: **time\_to\_live** <segundos>

Por defecto: 5

# Atributos de los Emisores (Emitters)

## ❑ duration

- ❑ Establece el número de segundos que el emisor está activo.
- ❑ Un valor de 0 indica duración infinita.
- ❑ Un valor <0 emite todas las partículas en el siguiente frame.
- ❑ El emisor puede iniciarse de nuevo, (P. ej. con `repeat_delay`).
- ❑ Además, con los atributos `duration_min` y `duration_max`, se puede establecer un intervalo de duración en lugar de uno fijo.

Formato: **duration** <segundos>

Por defecto: 0

## ❑ emission\_rate

- ❑ Establece cuántas partículas por segundo deben emitirse.
- ❑ El comportamiento depende del emisor.
- ❑ También estará limitado por la configuración de `quota` del sistema de partículas.

Formato: **emission\_rate** <partículas\_por\_segundo>

Por defecto: 10

# Modificadores (Affectors)

- ❑ Los modificadores (**affectors**) de partículas modifican las partículas durante de su vida.
- ❑ Se clasifican por tipo. Los que vamos a utilizar son:
  - ❑ `ColourFader` alteran el color de las partículas en vuelo.
  - ❑ `ColourImage` modifica el color de las partículas en vuelo utilizando los colores de la imagen del archivo indicado.
  - ❑ `Scaler` escala las partículas en vuelo.
  - ❑ `Rotator` rota las partículas en vuelo.
  - ❑ `DirectionRandomiser` aplica aleatoriedad al movimiento de las partículas.
- ❑ Se pueden añadir nuevos efectos a Ogre mediante la creación de *plugins*.
- ❑ Los **affectors** no tienen atributos universales
  - ❑ Todos son específicos del tipo que se utilice.

## ❑ Altera el color de las partículas en vuelo.

- ❑ **red**: Establece el ajuste que se realizará en el componente rojo del color de las partículas por segundo.

Formato: **red** <delta\_value>

Por defecto: 0

- ❑ **green**: Establece el ajuste que se realizará en el componente verde del color de las partículas por segundo.

Formato: **green** <delta\_value>

Por defecto: 0

- ❑ **blue**: Establece el ajuste que se realizará en el componente azul del color de las partículas por segundo.

Formato: **blue** <delta\_value>

Por defecto: 0

- ❑ **alpha**: Establece el ajuste que se realizará en el componente alfa de las partículas por segundo. Los valores negativos aumentan la transparencia, y los positivos aumentan la opacidad.

Formato: **alpha** <delta\_value>

Por defecto: 0

```
affector ColourFader{  
    red 0.06  
    green 0.06  
    blue 0.06  
    alpha -0.06  
}
```

# ColourImage Affector

- ❑ Modifica el color de las partículas en vuelo, pero en lugar de definir los colores mediante programación, los colores se toman de un archivo de imagen.

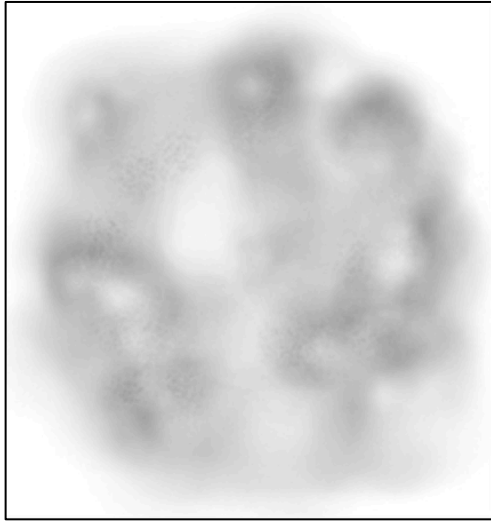
- ❑ **image:** El inicio de un rango de velocidad de rotación que se asignará a las partículas emitidas.

Formato: **image** <nombre\_de\_la\_imagen>

Por defecto: ninguna

```
affector ColourImage{  
    image    smokecolors.png  
}
```

# ColourImage Affector



Textura 2D para las partículas (RGBA):  
smoke.png -> dada en el material

Se combinará el color del téxel de la textura 2D del material con un color de la textura 1D (array de colores dado en el [affector](#)), en función del tiempo de vida de la partícula.

Textura 1D con los colores: [smokecolors.png](#)



## ❑ Escala las partículas en vuelo.

- ❑ **rate**: Cantidad en la que se escalan las partículas en las direcciones x e y por segundo.

Formato: **rate** <cantidad>

- ❑ **scale\_range**: Rango del factor de escala que se aplicará a las partículas emitidas.

Formato: **scale\_range** <min> <max>

Por defecto: 1 1

```
affector Scaler{  
    rate    50  
}
```

## ❑ Rota las partículas en vuelo.

- ❑ **rotation\_speed\_range\_start**: Inicio de un rango de velocidades de rotación que se asignarán a las partículas emitidas.

Formato: **rotation\_speed\_range\_start** <grados\_por\_segundo>

Por defecto: 0

- ❑ **rotation\_speed\_range\_end**: Final de un rango de velocidades de rotación que se asignarán a las partículas emitidas.

Formato: **rotation\_speed\_range\_end** <grados\_por\_segundo>

Por defecto: 0

- ❑ **rotation\_range\_start**: Inicio de un rango de ángulos de rotación que se asignarán a las partículas emitidas.

Formato: **rotation\_range\_start** <grados>

Por defecto: 0

- ❑ **rotation\_range\_end**: Final de un rango de ángulos de rotación que se asignarán a las partículas emitidas.

Formato: **rotation\_range\_end** <grados>

Por defecto: 0

```
affector Rotator{  
    rotation_speed_range_start    -30  
    rotation_speed_range_end      55  
    rotation_range_start          0  
    rotation_range_end            180  
}
```



# DirectionRandomiser Affector

- ❑ Aplica aleatoriedad al movimiento de las partículas.
  - ❑ **randomness**: Cantidad de aleatoriedad a introducir en cada dirección.  
Formato: **randomness** <cantidad>  
Por defecto: 1
  - ❑ **scope**: Porcentaje de partículas afectadas en cada pasada.  
Formato: **scope** <porcentaje>  
Por defecto: 1.0
  - ❑ **keep\_velocity**: Determina si la velocidad de las partículas no varía.  
Formato: **keep\_velocity** <true|false>  
Por defecto: false

```
affector DirectionRandomiser{  
    randomness 5  
}
```

# Sistema de partículas (Ejemplo para simular humo)

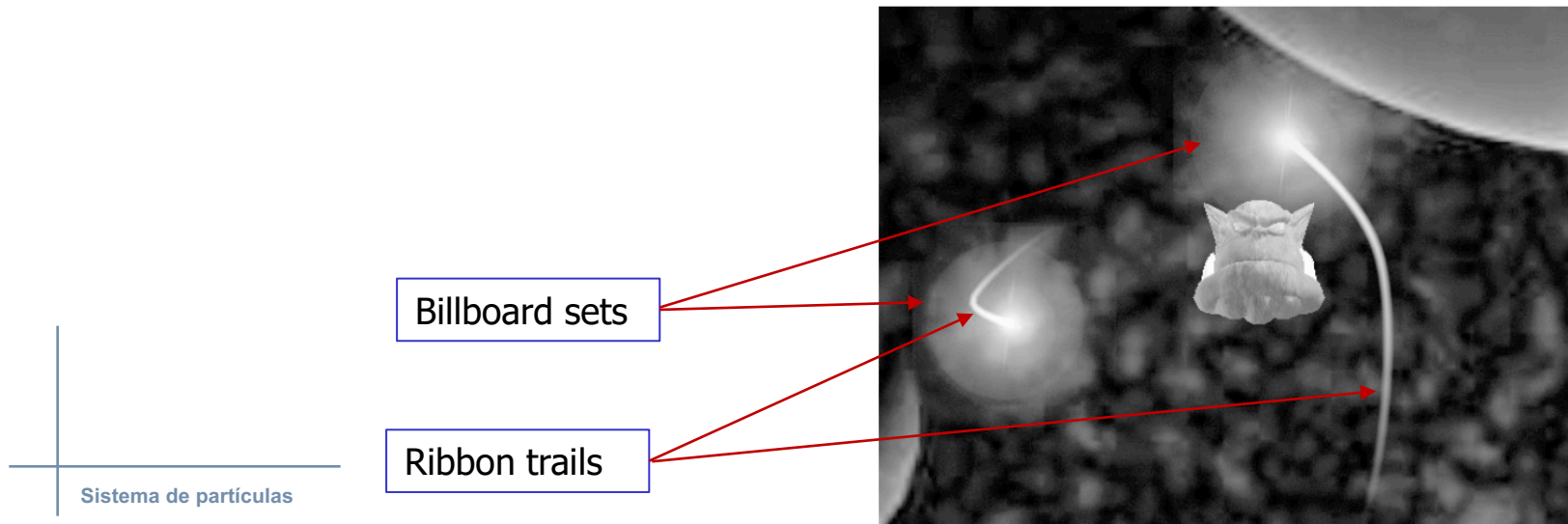
- ❑ El siguiente script simula el humo en la escena
- ❑ Utilizamos:
  - ❑ Un material: "example/smoke"
  - ❑ Un emisor: Point
  - ❑ Dos modificadores: ColourFader y Scaler

```
particle_system example/smokeParticle{  
  
    billboard_type    point  
    particle_width    35  
    particle_height   35  
    quota             500  
    material          example/smoke  
  
    emitter Point{  
  
        direction      0 1 0  
        position        0 0 0  
        angle          180  
        emission_rate   50  
        time_to_live    10  
        velocity_min    50  
        velocity_max    80  
        colour          0.4 0.4 0.4 0.4  
        duration        10  
    }  
}
```

```
affector ColourFader{  
    red    0.06  
    green  0.06  
    blue   0.06  
    alpha  -0.06  
}  
  
affector Scaler{  
    rate  50  
}  
}
```

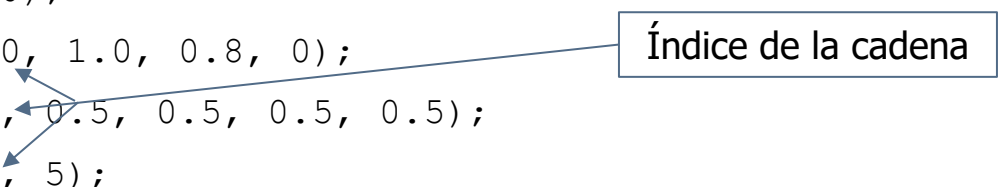
# BillboardChain y Ribbon trails

- ❑ Permiten definir estelas que dejan los objetos al moverse.
- ❑ **BillboardChain**: Análogo a un **BillboardSet**, pero los elementos forman cadenas. Hay que actualizarlos manualmente.
- ❑ **RibbonTrail** subclase de **BillboardChain**: implementa el posicionamiento de los elementos de la cadena siguiendo a un objeto.
  - ❑ Se puede establecer el número de cadenas, el máximo número de elementos por cadena, longitud de la estela, material, ....
  - ❑ Se puede generar un desvanecimiento alterando el color, transparencia, tamaño.
  - ❑ Y hay que determinar los nodos a los que tiene que seguir



# Ejemplo: Ribbon trails

```
RibbonTrail* trail = mSM->createRibbonTrail("RibbTrail");  
mSM->getRootSceneNode()->createChildSceneNode()->attachObject(trail);  
  
trail->setNumberOfChains(1);  
trail->setMaxChainElements(80);  
trail->setMaterialName("LightRibbonTrail");  
trail->setTrailLength(400);  
trail->setInitialColour(0, 1.0, 0.8, 0);  
trail->setColourChange(0, 0.5, 0.5, 0.5, 0.5);  
trail->setInitialWidth(0, 5);  
SceneNode* animNode = mSceneMgr->getRootSceneNode()->createChildSceneNode();  
  
// Nodo al que va a seguir  
trail->addNode(animNode);
```



Índice de la cadena