Informática Gráfica II

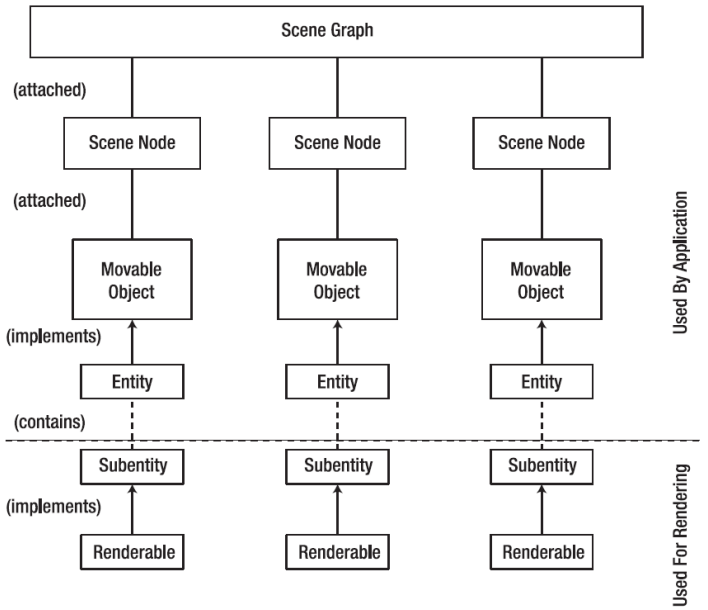
Object-oriented Graphics Rendering Engine

El grafo de la escena

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

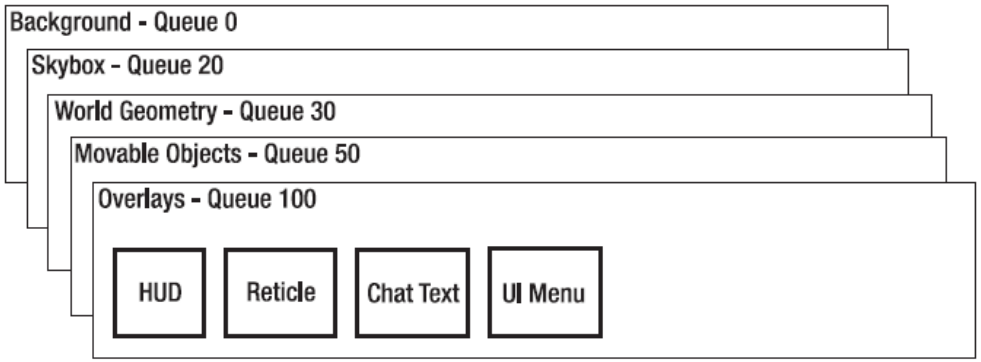
Grafo de la escena

- El grafo de la escena tiene todos los nodos del mismo tipo `SceneNode`.
  - Es un árbol general con un nodo raíz (*RootSceneNode*)
- Un nodo contiene objetos (de la clase abstracta `MovableObject`) y una transformación de modelado común para todos:
  - `Translate(position)`
  - `Rotate(orientation)`
  - `Scale`
- `SubEntity` (malla, material) implementa `Renderable`
- `MovableObject` la implementan
  - `Light`
  - `Camera`
  - `Entity` (subentities)



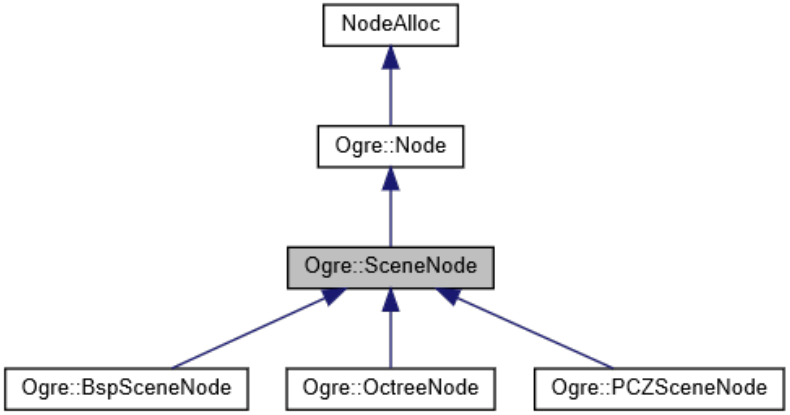
Grafo de la escena

- Una entidad está formada por varias subentidades
  - Cada subentidad contiene una referencia a una **submalla** y una referencia a un **material**.
- Las subentidades son elementos susceptibles de ser renderizados (**Renderable**).
- El gestor de la escena forma colas (Z-order) con estos elementos en el proceso de renderizado (transparentes, opacos, estáticos)



La clase SceneNode

- Atributos:
  - Puntero al padre
  - Punteros a los hijos
  - Nombre
  - Orientación
  - Posición
  - Escala
  - Matriz de transformación afín
  - Gestor de la escena que lo ha creado
  - Objetos -> **MovableObject: Entity (SubEntity), Light, Camera**
  - Caja delimitadora
  - Booleanos



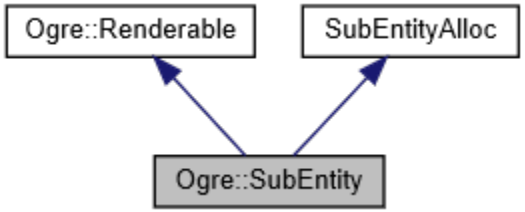
La clase SceneNode

- Los SceneNodes contienen información que se utiliza para todos los objetos que están conectados a él.
  - Una entidad no se renderiza en la escena hasta que no está conectada a un SceneNode.
  - SceneNode no es un objeto visible en la escena.
    - Contiene información abstracta (p. ej. Ubicación u orientación).
    - Sólo cuando está conectado a algo como una entidad se utiliza esa información para representar realmente algo en la escena.
- Los SceneNodes pueden tener más de un objeto conectado a ellos.
  - Por ejemplo, una luz que siga a un personaje en una escena.
    - Adjuntar tanto la entidad del personaje como la luz al mismo SceneNode.
    - Esto hará que ambos compartan la misma información de localización.
    - Podemos incluso adjuntar SceneNodes a otros SceneNodes.
    - Adjuntar elementos en una parte concreta (un arma en la mano de un personaje)
- Su posición es relativa a su SceneNode padre
- Cada **SceneManager** crea un Nodo raíz al que se adjuntan todos los demás SceneNodes.

La clase Entity

- **Entity** está organizada en **SubEntity**

- Atributos de **SubEntity**:
  - Malla
  - Material
  - Cola de renderizado
  - Booleanos: Visible



La clase Entity

- Una Entidad (**Entity**) es un tipo de objeto que puedes renderizar en tu escena.
- Es "cualquier cosa" representada por una malla 3D.
  - Objetos del terreno son entidades muy grandes.
  - Luces, Carteles, Partículas y Cámaras **NO son entidades**.
- Ogre separa los objetos renderizables de información como su ubicación.
- Las entidades no se colocan directamente en una escena.
  - Se coloca un **SceneNode** en su escena
  - Se adjunta una **entidad** a ese **SceneNode**.
  - La **entidad** se renderiza utilizando la información obtenida del **SceneNode**.

Cómo añadir elementos a la escena

- Se añaden en el método **`IG2App::setupScene(void)`** que es invocado al terminar **`IG2App::setup(void)`**
- Aquí empezaremos a crear nuestra escena
- Los elementos los añade el gestor de la escena **`mSM`**
  - Es un atributo de **`IG2App`** que se crea en el **`setup()`**

```
mSM = mRoot->createSceneManager();
```

- El gestor de la escena **NO** es **Singleton** y puede no ser único
- Todos los elementos se añaden a la escena incluyéndolos en un nodo.
  - Objeto **SceneNode**
- **MovableObjects:** elementos que se pueden añadir a nodos de la escena
  - Cámaras, las luces y las entidades.

Cómo añadir elementos a la escena

- Para añadir una **entidad** a la escena se realiza el siguiente proceso:
  1. El gestor de la escena crea la entidad a partir de una malla

```
Ogre::Entity* ent = mSM->createEntity("sphere.mesh");
```

  - Ojo! Las entidades **NO** se comparten.
  - Para añadir, por ejemplo, **dos esferas** a la escena, es necesario crear **dos entidades** con la malla de la esfera.
    - Aunque se puede reutilizar su nombre para crear las instancias de los objetos.
    - Si no se hace así, se produce un error y la escena no se renderiza.
  - En el fichero **`resources.cfg`** se debe indicar el acceso a la malla
    - Una manera recomendable de hacerlo es copiar **`sphere.mesh`**
      - Desde **`media/models`** a **`media/IG2App`**

Cómo añadir elementos a la escena

2. El gestor de la escena crea un nodo vacío
   - A partir de un nodo que ya exista
   - Dándole un nombre que sea único (entre el resto de nodos de la escena).
   - El nodo será hijo del nodo que lo creó.
   - En el siguiente ejemplo, el nuevo nodo es hijo de la raíz

```
Ogre::SceneNode* mSceneNewNode = mSM->getRootSceneNode()->createChildSceneNode("esfera");
```

   - Si se repite un nombre, se produce un error y la escena no se renderiza.
     - Se pueden crear nodos dentro de un bucle utilizando el contador `i`
     - Usar su índice como parte del nombre: **`std::to_string(i)`**
   - También es posible crear nodos sin nombre
     - Mediante **`createChildSceneNode()`** sin parámetros
     - Estos nodos no se pueden recuperar

3. Se adjunta la entidad al nodo
   - `mSceneNewNode->attachObject(ent);`

Cómo eliminar elementos de la escena

- El objeto **SceneManager** puede eliminar elementos como
  - SceneNodes

```
SceneNodesvirtual void Ogre::SceneManager::destroySceneNode(. . .);
```

    - Elimina el SceneNode
    - No se recomienda
    - Es mejor dejar que `SceneManager` elimine los objetos cuando se finalice
    - A su vez, un `SceneNode` puede desvincular una entidad con:

```
virtual void detachObject (MovableObject *obj);
```

  - Entidades

```
virtual void Ogre::SceneManager::destroyEntity (. . .);
```

    - Elimina y borra un objeto `Entity` del `SceneManager`
    - Antes se debe desvincular del `SceneNode` correspondiente
    - Se recomienda dejar que SceneManager elimine los objetos cuando se finalice

La clase Vector3

- La clase Vector3 representa un vector estándar de tres dimensiones
- Se puede crear un Vector3 indicando los valores de cada dimensión

```
Vector3 ejemploVector(valorX, valorY, valorZ);
```

- Se pueden acceder a las coordenadas del vector:

```
ejemploVector.x
ejemploVector.y
ejemploVector.Z
```

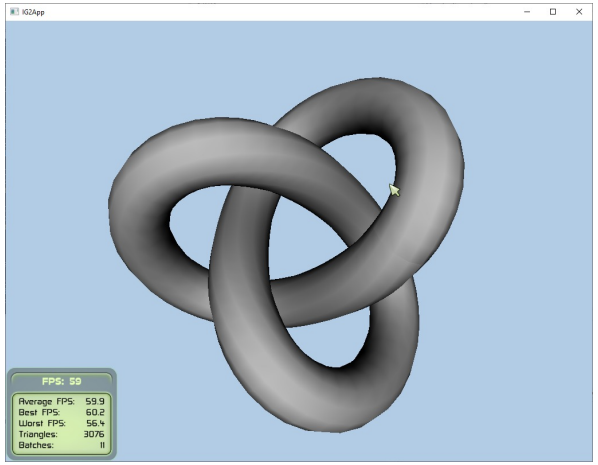
- La clase contiene constantes que representan las direcciones de los ejes y el vector (0,0,0)

```
static const Vector3 ZERO;
static const Vector3 UNIT_X;
static const Vector3 UNIT_Y;
static const Vector3 UNIT_Z;
static const Vector3 NEGATIVE_UNIT_X;
static const Vector3 NEGATIVE_UNIT_Y;
static const Vector3 NEGATIVE_UNIT_Z;
```

Ejemplo: Añadir un elemento a la escena

- Añadir el nudo a la escena

```
Ogre::Entity* ent = mSM->createEntity("knot.mesh");

Ogre::SceneNode* knotNode = mSM->getRootSceneNode()->createChildSceneNode("nudo");

knotNode->attachObject(ent);
```



Acciones sobre los nodos

Acciones opcionales posteriores que se pueden realizar sobre los nodos de la escena:

- Situar el nodo en un lugar determinado de la escena; por ejemplo, fijando una posición

```
mSceneNewNode->setPosition(400, 100, -300);
```

- Realizar transformaciones 3D (traslaciones, rotaciones y escalaciones) sobre el nodo

```
mSceneNewNode->setScale(5, 5, 5);
mSceneNewNode->yaw(Ogre::Degree(-45));
```

- Mostrar la caja delimitadora (*bounding box*) que contiene el nodo

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

- Hacer no visible el nodo en la escena. Por defecto es visible

```
mSceneNewNode->setVisible(false);
```

- Sobre un nodo no podemos comprobar si es o no visible, pero sí sobre una entidad

```
if (ent->isVisible()){…}
```

Entidades sobre mallas construidas

- Se pueden añadir a la escena mallas creadas (las `.mesh`) y mallas construidas
- Solo hay dos formas de crear mallas construidas: superficies planas y superficies curvadas
- Para construir la malla del plano se usa el comando `createPlane`. P. ej:

```
MeshManager::getSingleton().createPlane("mPlane1080x800",
                        ResourceGroupManager::DEFAULT_RESOURCE_GROUP_NAME,
                        Plane(Vector3::UNIT_Y, 0),
                        1080, 800, 100, 80,
                        true, 1, 1.0, 1.0, Vector3::UNIT_Z);
```

- Los parámetros son estos:
  - Nombre de la malla
  - Nombre del grupo de recursos que se le asigna a la malla
  - Orientación del plano (mediante el vector normal al plano)
  - Anchura del plano en coordenadas globales (world). Ídem para la altura.
  - Número de segmentos del plano en la dirección `X`. Ídem para la dirección `Y` (útil en superficies curvas)
  - Booleano (si *true*, crea normales perpendiculares al plano)
  - Número de conjuntos de coordenadas de texturas 2D (por defecto, las esquinas coinciden con las de la textura)
  - Número de veces que se repite la textura en la dirección `u`. Ídem para la dirección `v`
  - Orientación '`Up`' de la textura.

Entidades sobre mallas construidas

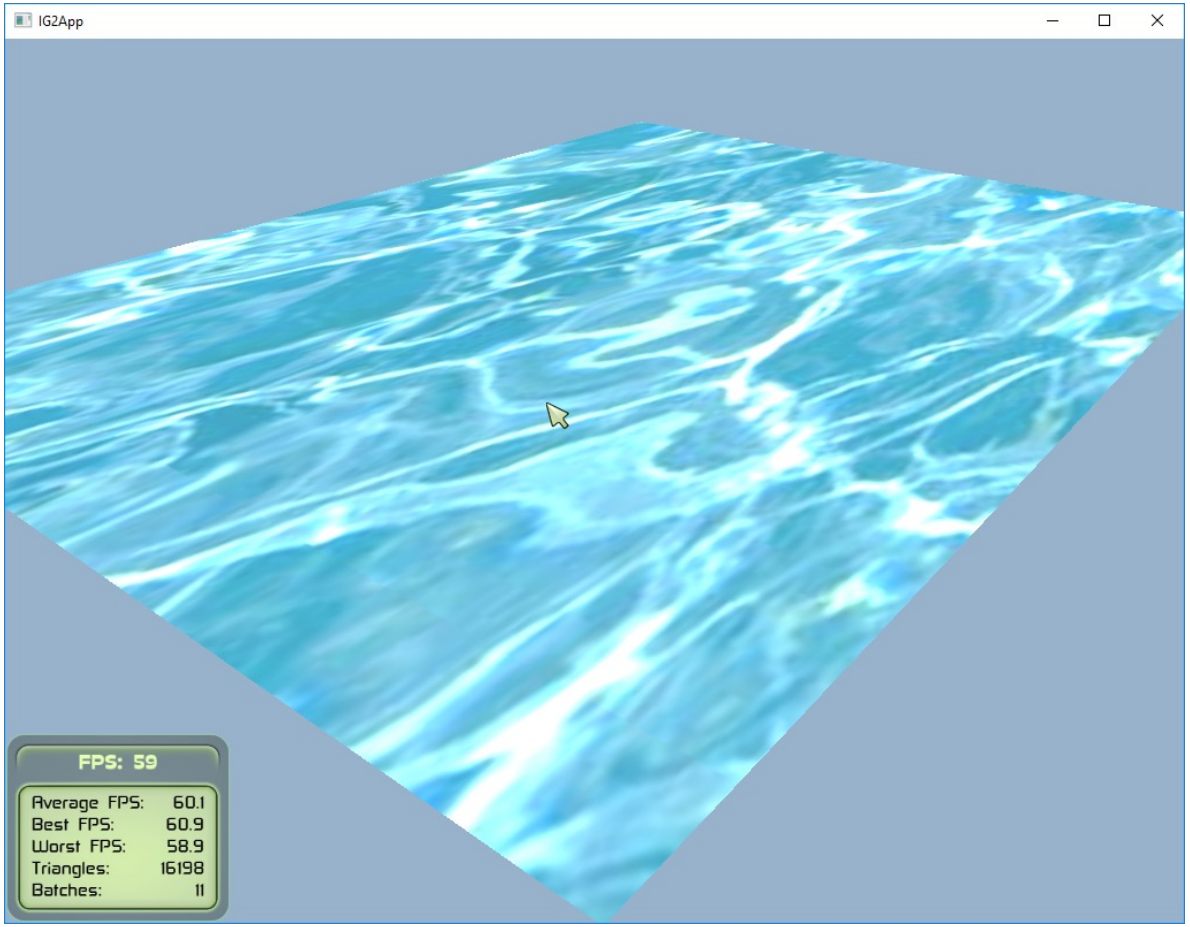
- Una vez construida la malla se puede definir una entidad que la use:

```
Ogre::Entity* plane = mSM->createEntity("mPlane1080x800");
```

- La entidad se puede asociar a un nodo como con el resto de las entidades y de los nodos del grafo de la escena que se han visto hasta ahora:

```
mNode->attachObject(plane);
```

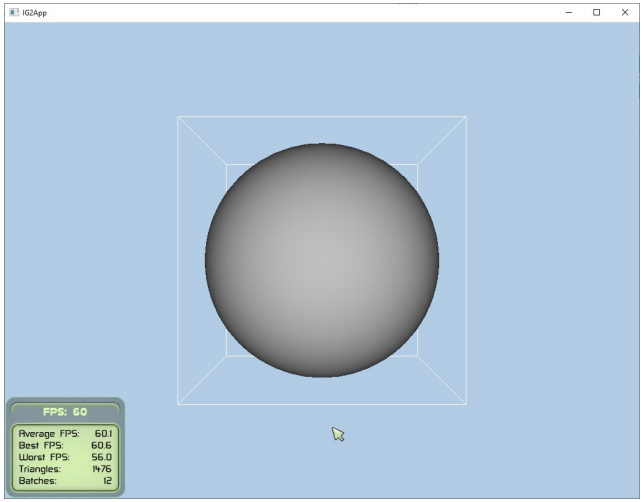
Ejemplo de plano con textura



Bounding Box

- Caja mínima en la cual se encuentra contenida una entidad.
  - **AABB** - **A**xis **A**ligned **B**ounding **B**ox.
  - Esta caja está alineada permanentemente con los ejes del mundo (World axes)
  - Método para obtener el AABB

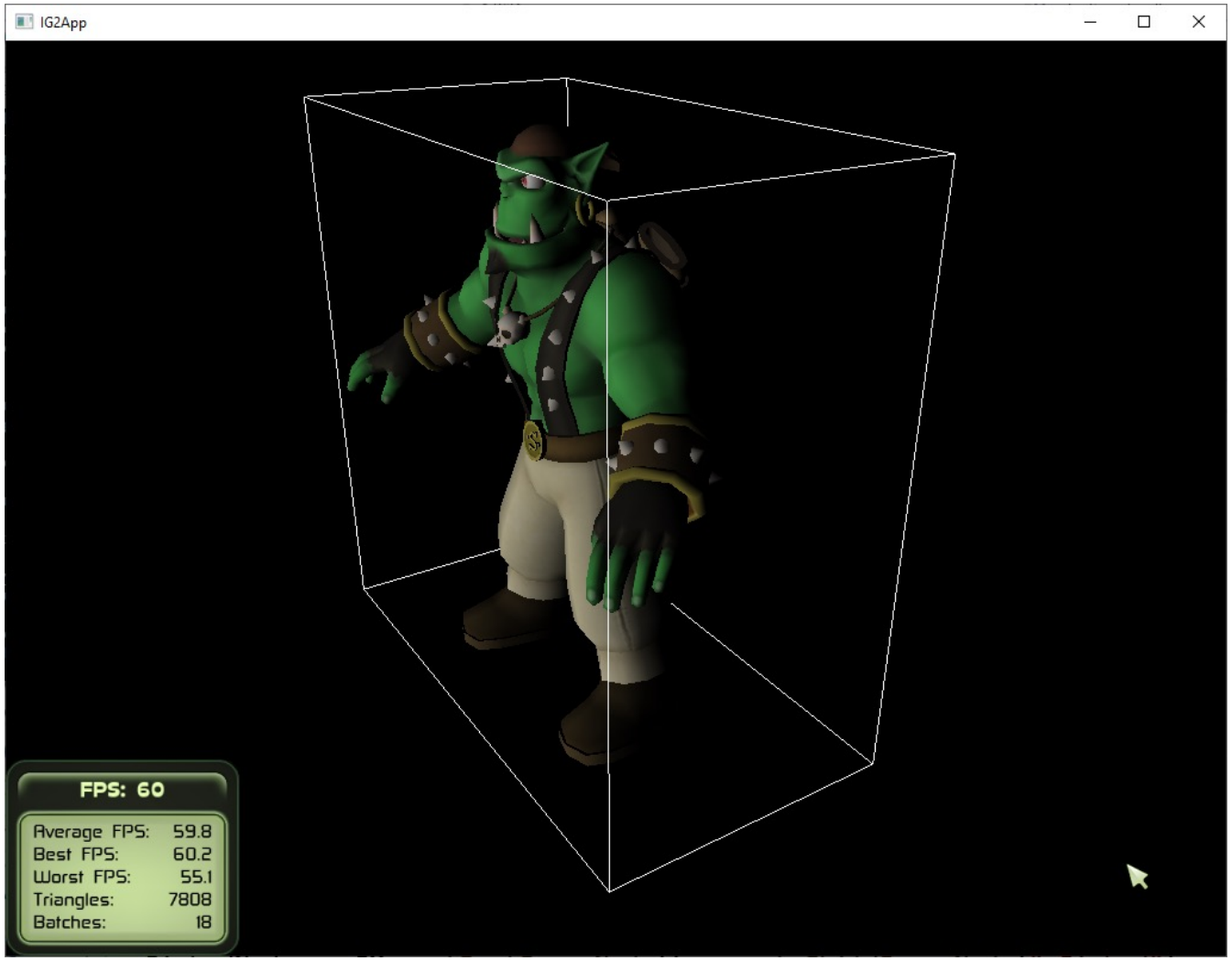
```
const AxisAlignedBox& IG2Object::getAABB(){
    return mNode->_getWorldAABB();
}
```



- Parte del código de **`setupScene()`** de **IG2App** para añadir Sinbad a la escena

```
Ogre::Entity* ent = mSM->createEntity("Sinbad.mesh");
mSinbadNode = mSM->getRootSceneNode()->createChildSceneNode ("nSinbad");
mSinbadNode->attachObject(ent);

mSinbadNode->setPosition(400, 100, -300);
mSinbadNode->setScale(20, 20, 20);
mSinbadNode->yaw(Ogre::Degree(-45));
mSinbadNode->showBoundingBox(true);
```

Sinbad


IG2App
FPS: 60
Average FPS: 59.8
Best FPS: 60.2
Worst FPS: 55.1
Triangles: 7808
Batches: 18

Ejemplo de la estructura del grafo

- Podemos añadir los nodos para crear figuras más complejas
- Invocando **createChildSceneNode** desde otro nodo
  - `mNodeChild = mNodeParent->createChildSceneNode("nSinbadChild");`
  - Creamos el grafo definiendo una estructura específica
- Al aplicar la transformación en el nodo padre, se aplica en los nodos hijo

```
Ogre::Entity* entParent = mSM->createEntity("Sinbad.mesh");
mNodeParent = mSM->getRootSceneNode()->createChildSceneNode("nSinbadParent");
mNodeParent->attachObject(entParent);

Ogre::Entity* entChild = mSM->createEntity("Sinbad.mesh");
mNodeChild = mNodeParent->createChildSceneNode("nSinbadChild");
mNodeChild->attachObject(entChild);

mNodeChild->translate(50, 0, 0);
```

Ejemplo de la estructura del grafo

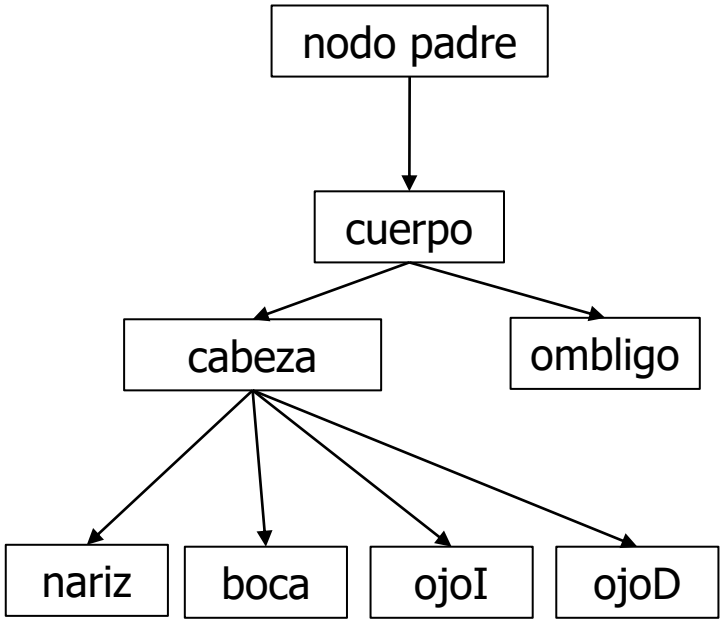
```
mNodeChild->translate(50, 0, 0);
```

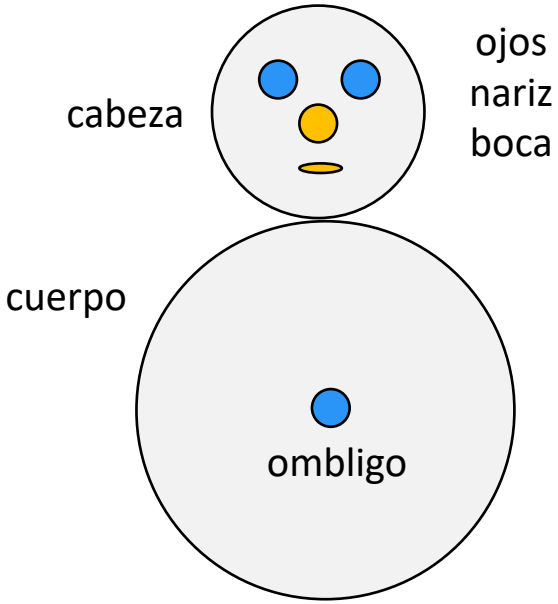
Movemos 50 al hijo (Sinbad derecho) en el eje de las X

```
mNodeParent->yaw(Radian(Math::PI/10.0));
```

Rotamos al padre (Sinbad izquierdo), y también rota el hijo.

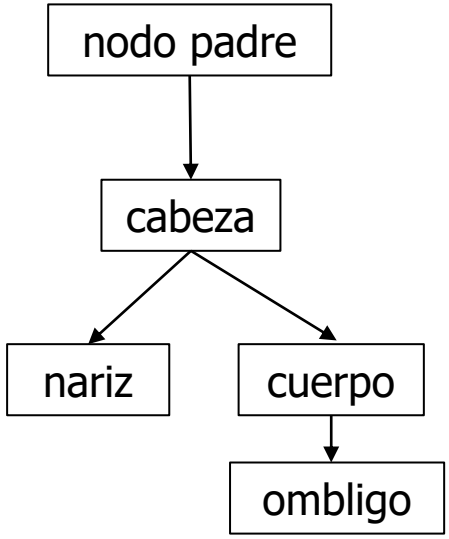
Ejemplo de construcción de una figura compleja: Snowman

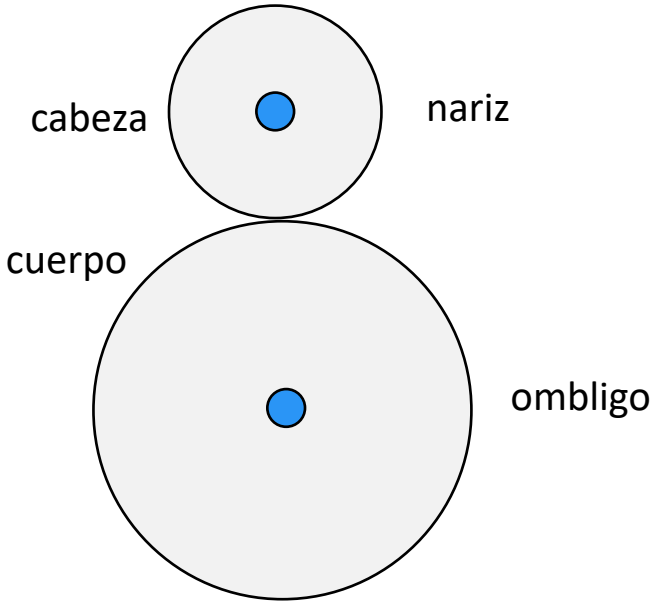




- La cabeza puede girar hacia los lados
  - incluyendo los ojos, nariz y boca, independientemente del cuerpo.
- El cuerpo puede girar para desplazarse, con la cabeza que se desplaza con el cuerpo.
- Todas las entidades son esferas (`sphere.mesh`)

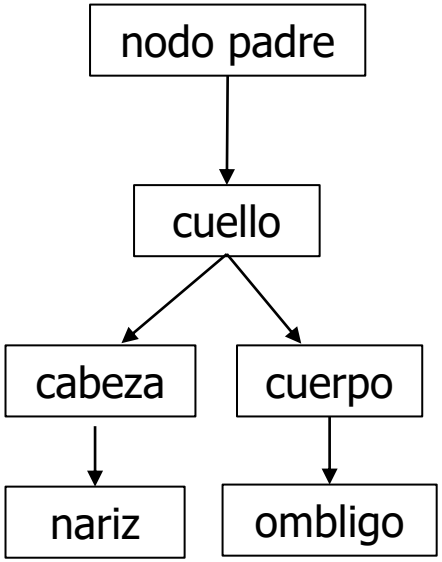
Ejemplo de construcción de una figura compleja: Snowman

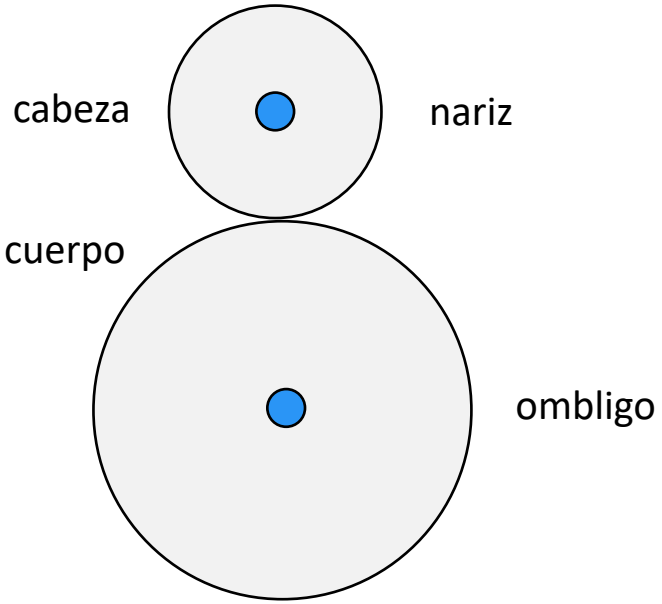




- La cabeza puede girar hacia los lados, incluyendo la nariz y el cuerpo con el ombligo
- El cuerpo puede girar de manera independiente

Ejemplo de construcción de una figura compleja: Snowman





- La cabeza puede girar hacia los lados, incluyendo la nariz, independientemente del cuerpo
- El cuerpo puede girar, incluyendo el ombligo, independientemente de la cabeza
- Con esta estructura, cuerpo y cabeza son independientes
- Utilizando el nodo padre, la transformación afecta a todo el muñeco

Recordatorio: Sí se puede...

- Reutilizar la variable **ent** para referirse a entidades, posiblemente diferentes, en distintos lugares del código:

```
Ogre::Entity* ent = mSM->createEntity("RomanBathLower.mesh");
Ogre::SceneNode* mNode = mSM->getRootSceneNode()->createChildSceneNode();
mNode->attachObject(ent);

ent = mSM->createEntity("RomanBathUpper.mesh");
Ogre::SceneNode* mNode1 = mSM->getRootSceneNode()->createChildSceneNode();
mNode1->attachObject(ent);
```

- Reutilizar la variable **mNode** para referirse a nodos, posiblemente diferentes, en distintos lugares del código

```
Ogre::Entity* ent = mSM->createEntity("RomanBathLower.mesh");
Ogre::SceneNode* mNode = mSM->getRootSceneNode()->createChildSceneNode();
mNode->attachObject(ent);

ent = mSM->createEntity("RomanBathUpper.mesh");
mNode = mSM->getRootSceneNode()->createChildSceneNode();
mNode->attachObject(ent);
```

Recordatorio: NO se puede...

- Reutilizar la malla de una entidad

```
Entity* hour = mSM->createEntity("sphere.mesh");
Ogre::SceneNode* mHourNode = mSM->getRootSceneNode()->createChildSceneNode("Hour 1");
mHourNode->attachObject(hour);

mHourNode = mSM->getRootSceneNode()->createChildSceneNode("Hour 2");
mHourNode->attachObject(hour);    // ERROR
```

- Utilizar el mismo nombre para nodos diferentes

```
Entity* hour = mSM->createEntity("sphere.mesh");
Ogre::SceneNode* mHourNode = mSM->getRootSceneNode()->createChildSceneNode("Hour 1");
mHourNode->attachObject(hour);

hour = mSM->createEntity("sphere.mesh");
mHourNode = mSM->getRootSceneNode()->createChildSceneNode("Hour 1"); // ERROR
mHourNode->attachObject(hour);
```

Transformaciones sobre nodos

- En el tema anterior vimos algunos ejemplos sobre transformaciones
- Los nodos de la escena admiten **tres tipos de transformaciones** de modelado
  - **Traslaciones** para situar el nodo en un lugar determinado de la escena
  - **Rotaciones** para situar el nodo con una orientación determinada
  - **Escalaciones** para mostrar la entidad asociada a un nodo con un tamaño determinado
- Cada una de estas transformaciones admite, además, distintos comandos y muchos de ellos se pueden invocar con respecto a distintos espacios o sistemas de coordenadas
- Cada tipo de transformación se guarda por separado
  - Posición
  - Orientación
  - Escala
- A partir de las tres partes y el árbol, se aplican las transformaciones y se genera la matriz de cada nodo según el orden **T*R*S*vertex**

Sistema de coordenadas

Podemos especificar las transformaciones de modelado (de los nodos de la escena) en:

- **World space (TS_WORLD)**
  - Relativo al sistema de coordenadas global (nodo raíz, si no se ha modificado).
- **Parent space (TS_PARENT)**
  - Relativo al padre del nodo del grafo de la escena.
  - Incluye todas las rotaciones desde la raíz del árbol al padre del nodo.
- **Local space (TS_LOCAL)**
  - Relativo al nodo de la escena.
  - Incluye todas las rotaciones desde la raíz del árbol al propio nodo.
- Lo más habitual:
  - Las traslaciones relativas al padre
  - Las rotaciones en el espacio local (por defecto en Ogre).
  - **Escalaciones** siempre con respecto al **local**
- Las coordenadas de los vértices nunca cambian, están en **espacio del objeto (local)**

Transformaciones

- Traslaciones

```
sceneNode->translate(100.0, 10.0, 0.0);  //Por defecto TS_PARENT
sceneNode->translate(100.0, 10.0, 0.0, Ogre::Node::TS_WORLD);
sceneNode->translate(0.0, 0.0, 100.0, Ogre::Node::TS_LOCAL);
```

- Escalaciones

```
sceneNode->scale(2.0, 1.0, 1.0);
```

- Otros métodos

```
rotate(…);          // Por defecto TS_LOCAL
setPosition (…); // Relativo a TS_PARENT
setOrientation(quaternion);
setRotation(…);
```

- Diferencia entre `translate` y `setPosition`
  - Una traslación se realiza de forma relativa a un espacio de coordinadas (global, padre, local)
  - Por otro lado, `setPosition` se realiza siempre en el espacio de coordenadas del padre

Herencia en las transformaciones

- Se puede especificar que un nodo **no** se vea afectado por algunas transformaciones del nodo padre.
  - `sceneNode -> setInheritOrientation (bool inherit);`
    - Por defecto (true) La orientación del nodo será afectada por la del padre
  - `sceneNode -> setInheritScale (bool inherit);`
    - Por defecto (true) La escala del nodo será afectada por la del padre
- Se aplica recursivamente a todos los nodos hijos del nodo `sceneNode`
  - Ver ejemplo snowman en el Campus Virtual
- Podemos reiniciar las transformaciones de un nodo:
  - `sceneNode->resetToInitialState();`
  - `sceneNode->setInitialState();` // Útil para animaciones
  - `sceneNode->resetOrientation()`

Traslaciones de nodos

- Los nodos se pueden trasladar de dos formas
  - `node->setPosition(x, y, z)`: traslada el nodo al punto de coordenadas **(x, y, z)**.
    - Coordenadas relativas a las coordenadas del **padre del nodo**.
    - Por ejemplo, si `node` está en las coordenadas (1, 2, 3) y hacemos `node->setPosition(4, 5, 6)`, entonces pasa a estar en (4, 5, 6).
  - `node->translate(x, y, z)`: traslada el nodo a una nueva posición.
    - La posición final se calcula sumando el vector **(x,y,z)** a su posición actual.
    - Por ejemplo, si `node` está en las coordenadas (1, 2, 3) y hacemos `node->translate(4, 5, 6)`, entonces pasa a estar en (5, 7, 9)

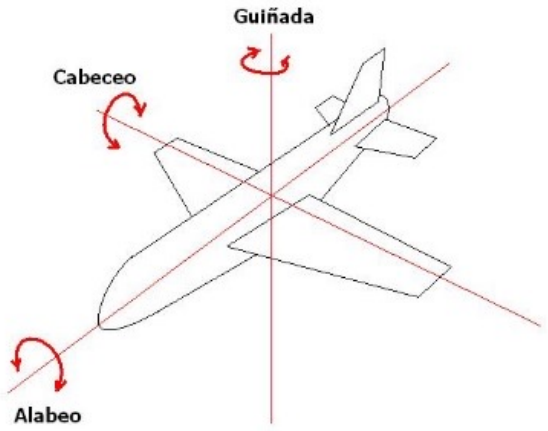
Rotaciones de nodos

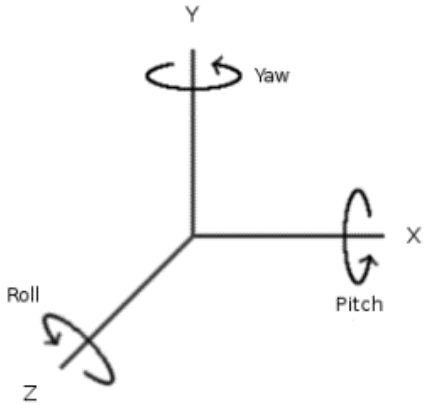
- Los nodos se pueden rotar de tres formas (en el parámetro es obligatorio poner los grados mediante **Degree** o **Radian**)
  - `node->pitch(Radian(Ogre::Math::HALF_PI))`
    - Gira en sentido anti-horario con respecto al eje **X** local de node
  - `node->yaw(Degree(90.0f))`
    - Gira en sentido anti-horario con respecto al eje **Y** local de **node**
  - `node->roll(Ogre::Radian(Ogre::Math::HALF_PI))`
    - Gira en sentido anti-horario con respecto al eje **Z** local de **node**
- Por ejemplo, si `node` tiene un hijo `node2`, ambos nodos contienen a `Sinbad.mesh`, y están situados en el mismo punto entonces:

```
node2->translate(10, 0, 0);
node2->pitch(Radian(Math::HALF_PI));
```

Rotaciones de nodos

```
// Por defecto TS_LOCAL: rota alrededor del eje Y del objeto
sceneNode->yaw(Ogre::Radian(1.0));

// Rota de forma relativa con el eje X del padre
sceneNode->pitch(Ogre::Radian(1.0), Ogre::Node::TS_PARENT);

// Rota de forma relativa con el eje Z del mundo
sceneNode->roll(Ogre::Degree(-45), Ogre::Node::TS_WORLD);
```



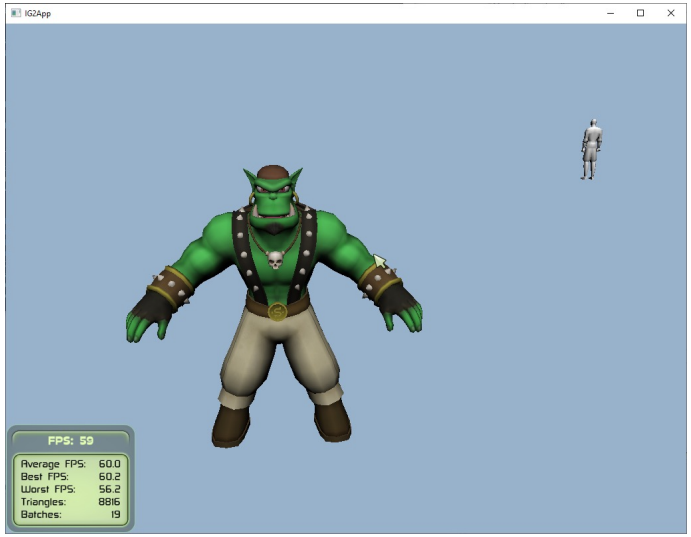


Escalaciones de nodos

- Los nodos se pueden escalar de dos formas
  - `node->setScale(sx, sy, sz);`
    - Escala el nodo según los factores de escalación
      - **sx** para el eje **X**
      - **sy** para el eje **Y**
      - **sz** para el eje **Z**
      - Obviamente lo que se escala es la entidad adjunta al nodo, no el nodo mismo
  - `node->scale(sx, sy, sz);`
    - Escala el nodo según los factores **sx**, **sy** y **sz**.
    - La diferencia con el comando anterior son las mismas que las que hay entre `setPosition()` y `translate()`.
    - `setScale()` escala de forma absoluta y `scale()` lo hace de forma relativa al tamaño ya existente

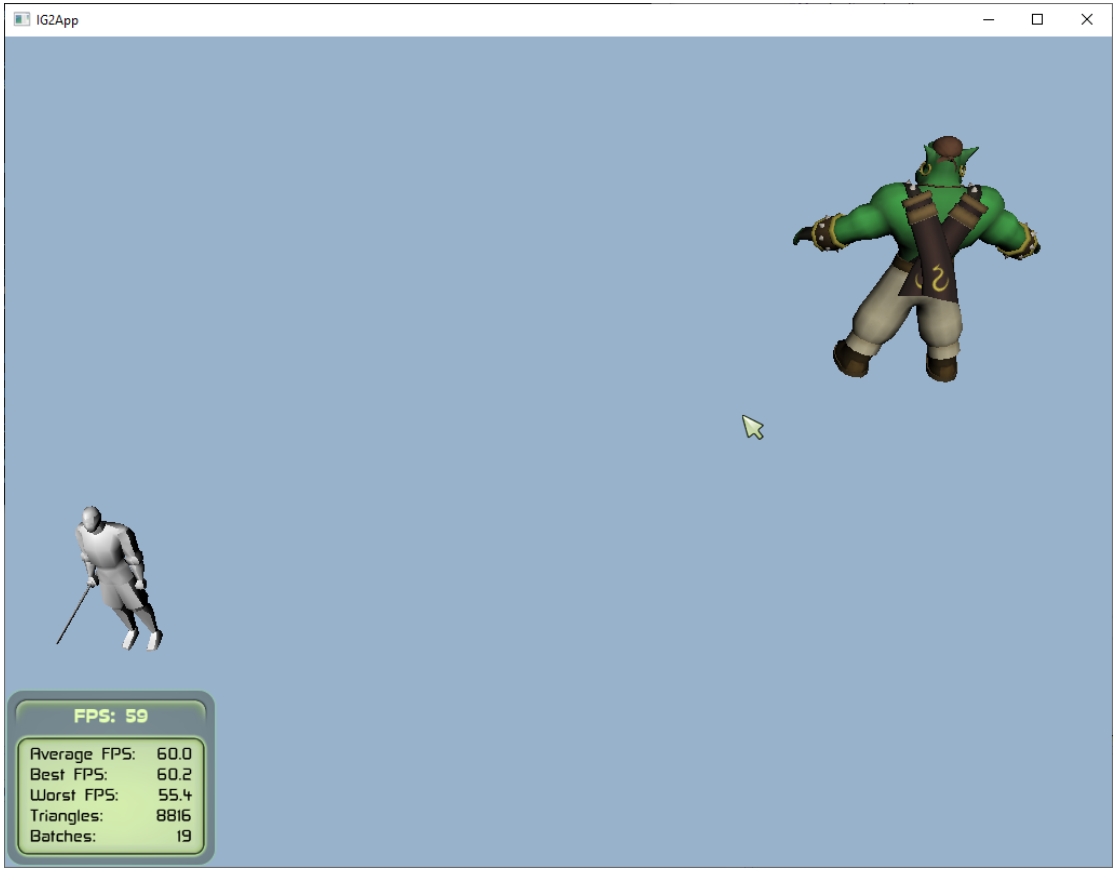
Ejemplo

```
// Creamos a Sinbad en (10, 10, 10) y lo escalamos

SceneNode* node = mSM->getRootSceneNode()->createChildSceneNode();
Entity* ent = mSM->createEntity("Sinbad.mesh");
node->attachObject(ent);
node->setPosition(10, 10, 0);
node->setScale(50, 50, 50);

// Creamos un ninja (hijo de Sinbad) escalado

SceneNode* node2 = node->createChildSceneNode();
ent = mSM->createEntity("ninja.mesh");
node2->attachObject(ent);
node2->translate(20, 0, -20);
node2->scale(0.02, 0.02, 0.02);
```



Ejemplo

```
// Rotamos Sinbad. ¿Qué ocurre con el Ninja?
node->yaw(Radian(Math::PI));
```

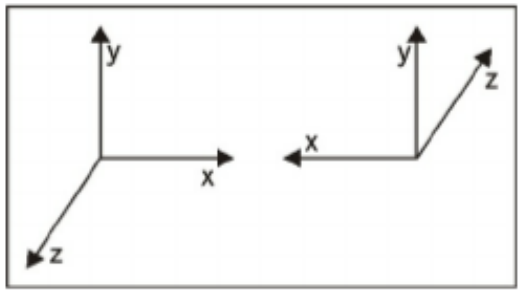


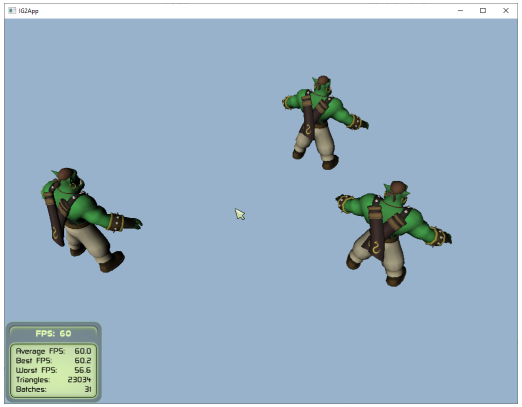
Ejemplo 1: Traslación con respecto al espacio del padre

```
SceneNode* node = mSM->getRootSceneNode()->createChildSceneNode();
Entity* ent = mSM->createEntity("Sinbad.mesh");
node->attachObject(ent);
node->setPosition(0, 0, 100);

// Las escalaciones (node) siempre son con respecto al espacio local (node)
node->scale(50, 50, 50);

// Las rotaciones (node) son, por defecto, con respecto al espacio local (node)
node->yaw(Degree(180.0));
SceneNode* node2 = node->createChildSceneNode();
ent = mSM->createEntity("Sinbad.mesh");
node2->attachObject(ent);
node2->setPosition(10, 0, 0);

// Las traslaciones (node2) son, por defecto, con respecto al padre (node)
node2->translate(0, 0, 10);
SceneNode* node3 = node->createChildSceneNode();
ent = mSM->createEntity("Sinbad.mesh");
node3->attachObject(ent);
node3->setPosition(20, 0, 0);
node3->translate(0, 0, 10);
```

Ejemplo 2: Traslación con respecto al espacio global

```
SceneNode* node = mSM->getRootSceneNode()->createChildSceneNode();
Entity* ent = mSM->createEntity("Sinbad.mesh");
node->attachObject(ent);
node->setPosition(0, 0, 100);
node->scale(50, 50, 50);
node->yaw(Degree(180.0));

SceneNode* node2 = node->createChildSceneNode();
ent = mSM->createEntity("Sinbad.mesh");
node2->attachObject(ent);
node2->setPosition(10, 0, 0);
node2->translate(0, 0, 10);

SceneNode* node3 = node->createChildSceneNode();
ent = mSM->createEntity("Sinbad.mesh");
node3->attachObject(ent);
node3->setPosition(20, 0, 0);
node3->translate(0, 0, 500, Ogre::Node::TS_WORLD);
```





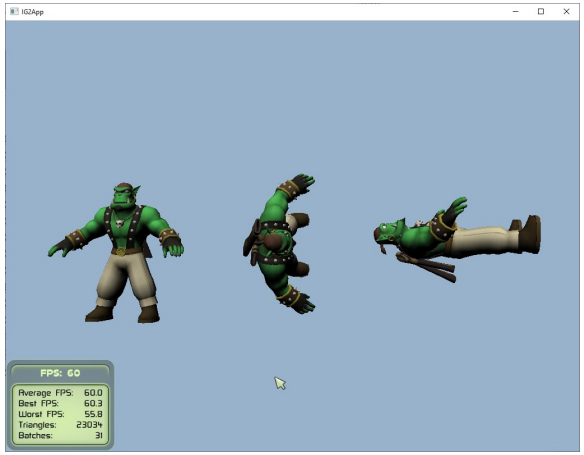
Ejemplo 3: Traslación con respecto al espacio local

```
SceneNode* node = mSM->getRootSceneNode()->createChildSceneNode();
Entity* ent = mSM->createEntity("Sinbad.mesh");
node->attachObject(ent);
node->setPosition(0, 0, 400);
node->scale(50, 50, 50);
node->yaw(Degree(180.0));

SceneNode* node2 = node->createChildSceneNode();
ent = mSM->createEntity("Sinbad.mesh");
node2->attachObject(ent);
node2->yaw(Ogre::Degree(45));
node2->translate(0, 0, 20);

SceneNode* node3 = node->createChildSceneNode();
ent = mSM->createEntity("Sinbad.mesh");
node3->attachObject(ent);
node3->yaw(Ogre::Degree(45));
node3->translate(0, 0, 20, Ogre::Node::TS_LOCAL);
```

Ejemplo 4: Rotación con respecto al espacio global

```
SceneNode* node = mSM->getRootSceneNode()->createChildSceneNode();
Entity* ent = mSM->createEntity("Sinbad.mesh");
node->attachObject(ent);
```

```
SceneNode* node2 = mSM->getRootSceneNode()->createChildSceneNode();
ent = mSM->createEntity("Sinbad.mesh");
node2->attachObject(ent);
node2->setPosition(10, 0, 0);
node2->yaw(Ogre::Degree(90));
node2->roll(Ogre::Degree(90));
```

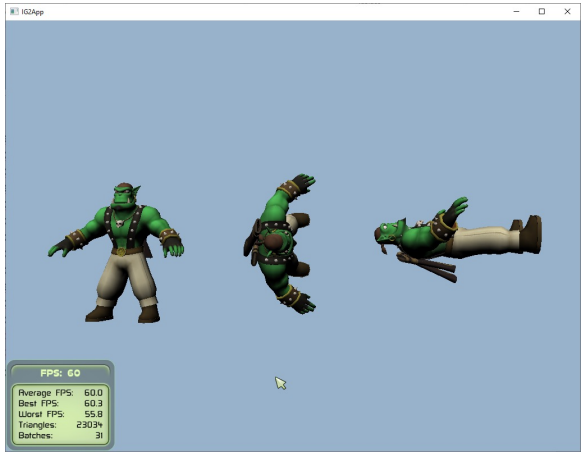



```
SceneNode* node3 = node->createChildSceneNode();
ent = mSM->createEntity("Sinbad.mesh");
node3->setPosition(20, 0, 0);
node3->attachObject(ent);
node3->yaw(Ogre::Degree(90), Ogre::Node::TS_WORLD);
node3->roll(Ogre::Degree(90),Ogre::Node::TS_WORLD);
```

Ejemplo 5: Rotación con respecto al espacio del padre

```
SceneNode* node = mSM->getRootSceneNode()->createChildSceneNode();
Entity* ent = mSM->createEntity("Sinbad.mesh");
node->attachObject(ent);
```

```
SceneNode* node2 = mSM->getRootSceneNode()->createChildSceneNode();
ent = mSM->createEntity("Sinbad.mesh");
node2->attachObject(ent);
node2->setPosition(10, 0, 0);
node2->yaw(Ogre::Degree(90));
node2->roll(Ogre::Degree(90));
```

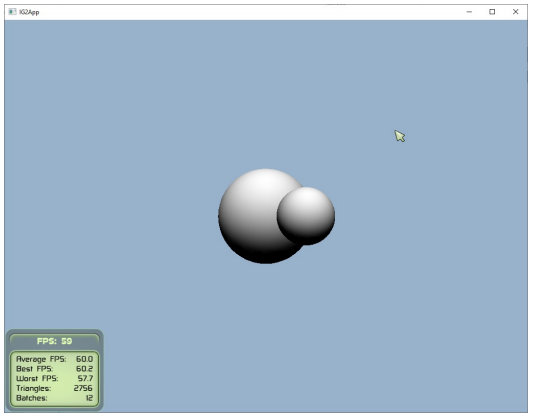



```
SceneNode* node3 = node->createChildSceneNode();
ent = mSM->createEntity("Sinbad.mesh");
node3->setPosition(20, 0, 0);
node3->attachObject(ent);
node3->yaw(Ogre::Degree(90), Ogre::Node::TS_PARENT);
node3->roll(Ogre::Degree(90), Ogre::Node::TS_PARENT);
```

Ejemplo: Rotación de un nodo alrededor de otro

- Mover una esfera (la tierra) alrededor de otra (el sol) mediante evento de teclado
- Soluciones:

1. Hacer que el nodo de la tierra sea hijo del nodo del sol. Entonces, cuando rota el nodo del sol, rota también el nodo de la tierra
2. Supongamos que el nodo de la tierra **NO** puede ser hijo del nodo del sol.
   i. Reposicionamiento: se aumenta **alfa** y se reposiciona la tierra en el punto **(cos(alfa),0,sin(alfa))** como tratamiento del evento
   ii. Nodo ficticio: se crea un nodo ficticio al nivel del nodo del sol y que sea padre del de la tierra. Se rota el nodo ficticio como tratamiento del evento
   iii. Truco: Haciendo la rotación del nodo de la tierra en el origen, trasladándolo allí antes y volviéndolo a llevar a su órbita después, como tratamiento del evento



Ejemplo: Rotación de un nodo alrededor de otro

- Situación inicial. La tierra está en un nodo que **NO** es hijo del nodo del sol

```
solNode = mSM->getRootSceneNode()->createChildSceneNode("sol");
Entity* sol = mSM->createEntity("sphere.mesh");
solNode->attachObject(sol);

tierraNode = mSM->getRootSceneNode()->createChildSceneNode("tierra");
Entity* tierra = mSM->createEntity("sphere.mesh");
tierraNode->attachObject(tierra);
```

Ejemplo: Rotación de un nodo alrededor de otro

- Reposicionamiento
  - En la escena:

```
tierraNode->setPosition(200*Ogre::Math::Cos(Ogre::Degree(alpha)), 0,
                        200*Ogre::Math::Sin(Ogre::Degree(alpha)));
```

  - En el tratamiento del evento:

```
alpha -= 5;
tierraNode->setPosition(200*Ogre::Math::Cos(Ogre::Degree(alpha)), 0,
                        200*Ogre::Math::Sin(Ogre::Degree(alpha)));
```

Ejemplo: Rotación de un nodo alrededor de otro

- Nodo ficticio

  - En la escena:

```
tierraNode2 = mSM->getRootSceneNode()->createChildSceneNode("tierra fake");
tierraNode = tierraNode2->createChildSceneNode("tierra");
Entity* tierra = mSM->createEntity("sphere.mesh");
tierraNode->translate(200, 0, 0);
tierraNode->scale(0.5, 0.5, 0.5);
tierraNode->attachObject(tierra);
```

  - En el tratamiento del evento:

```
tierraNode2->yaw(Ogre::Degree(-5));
```

Ejemplo: Rotación de un nodo alrededor de otro

- Truco
  - En la escena:

```
tierraNode->translate(200, 0, 0);
tierraNode->scale(0.5, 0.5, 0.5);
```

  - En el tratamiento del evento:

```
tierraNode->translate(-200, 0, 0, SceneNode::TS_LOCAL);
tierraNode->yaw(Ogre::Degree(-5));
tierraNode->translate(200, 0, 0, SceneNode::TS_LOCAL);
```

Bucle de renderizado

```
#include "IG2App.h"
 int main(int argc, char *argv[]){

       IG2App app;
       app.initApp();

       app.getRoot()->startRendering();

       app.closeApp();
       return 0;
}
```

→

```
...
while( !mQueuedEnd )
{
   mQueuedEnd = renderOneFrame();
}
```

Bucle de renderizado

```
mActiveRenderer->_initRenderTargets();
clearEventTimes();
mQueuedEnd = false;

// Infinite loop, until broken out of by frame listeners
// or break out by calling queueEndRendering()
while( !mQueuedEnd )
{
    mQueuedEnd = renderOneFrame();
}
```

```
_fireFrameStarted();          // avisa a los FrameListener registrados  -> pollEvents();
_updateAllRenderTargets();    // el renderizado -> frameRenderingQueued();
_fireFrameEnded();            // avisa a los FrameListener registrados
```

Cómo añadir eventos (de teclado) a la escena

- Se modifica el método **keyPressed()** de **IG2App**
- Por ejemplo, se incluye

```
else if (evt.keysym.sym == SDLK_g) {
        knotNode->roll(Ogre::Degree(3));
  }
```

hace girar el nudo alrededor de su eje **Z** cuando se pulsa la tecla **g**

- Es necesario añadir el objeto para que pueda recibir eventos:
  - `addInputListener (obj);`
- Este método está en la clase `IG2ApplicationContext`
  - `void addInputListener(InputListener* lis) { mInputListeners.insert(lis); }`
- El objeto **obj** tiene que heredar de `OgreBites::InputListener`
  - E implementar los métodos correspondientes (como `keyPressed`)

La clase InputListener

- OgreInput::OgreBites::InputListener

```
#include <OgreInput.h>
 // Métodos de InputListener que podemos redefinir

 virtual void frameRendered(const Ogre::FrameEvent& evt) { }
 virtual bool keyPressed(const KeyboardEvent& evt) { return false;}
 virtual bool keyReleased(const KeyboardEvent& evt) { return false; }
 virtual bool touchMoved(const TouchFingerEvent& evt) { return false; }
 virtual bool touchPressed(const TouchFingerEvent& evt) { return false; }
 virtual bool touchReleased(const TouchFingerEvent& evt) { return false; }
 virtual bool mouseMoved(const MouseMotionEvent& evt) { return false; }
 virtual bool mouseWheelRolled(const MouseWheelEvent& evt) { return false; }
 ...
}
```

La clase InputListener

```
// #include <SDL_keycode.h>
```

- Evento lanzado cuando se presiona una tecla:

```
bool Obj::keyPressed(const OgreBites::KeyboardEvent& evt) {
   if (evt.keysym.sym == SDLK_?){
         …
   }
   return true;
}
```

- Evento lanzado cuando se *renderiza* un frame:

```
void Obj::frameRendered(const Ogre::FrameEvent &  evt) {
     // Update
}
```

La clase FrameListener

- Clase "interface" que define los métodos que se utilizan para recibir notificaciones de los eventos relativos a los frames

- `virtual bool frameEnded (const FrameEvent &evt)`
  - Invocada justo después haber renderizado un frame.

- `virtual bool frameRenderingQueued (const FrameEvent &evt)`
  - Invocado después de que se hayan emitido los comandos de renderizado de todos los objetivos, pero antes de volcar los búferes de las ventanas.

- `virtual boolframeStarted (const FrameEvent &evt)`
  - Invocado cuando un frame está "a punto" de empezar a renderizarse