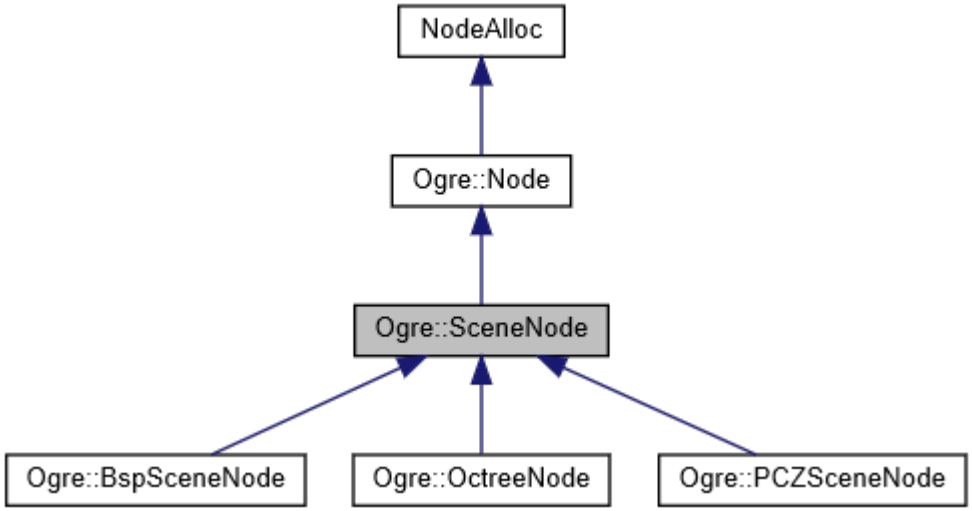
Nodos de la escena

- 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
Caja delimitadora
Booleanos



Objetos -> **MovableObject: Entity (SubEntity), Light, Camera**

Entidades

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

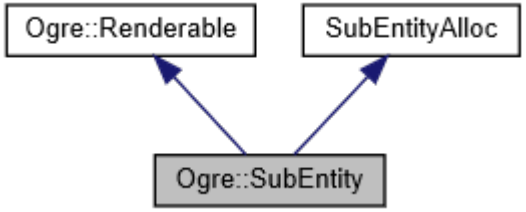
  La clase **SubEntity.** Atributos:

  Malla

  Material

  Cola de renderizado

  Booleanos: Visible



Entidades sobre mallas construidas

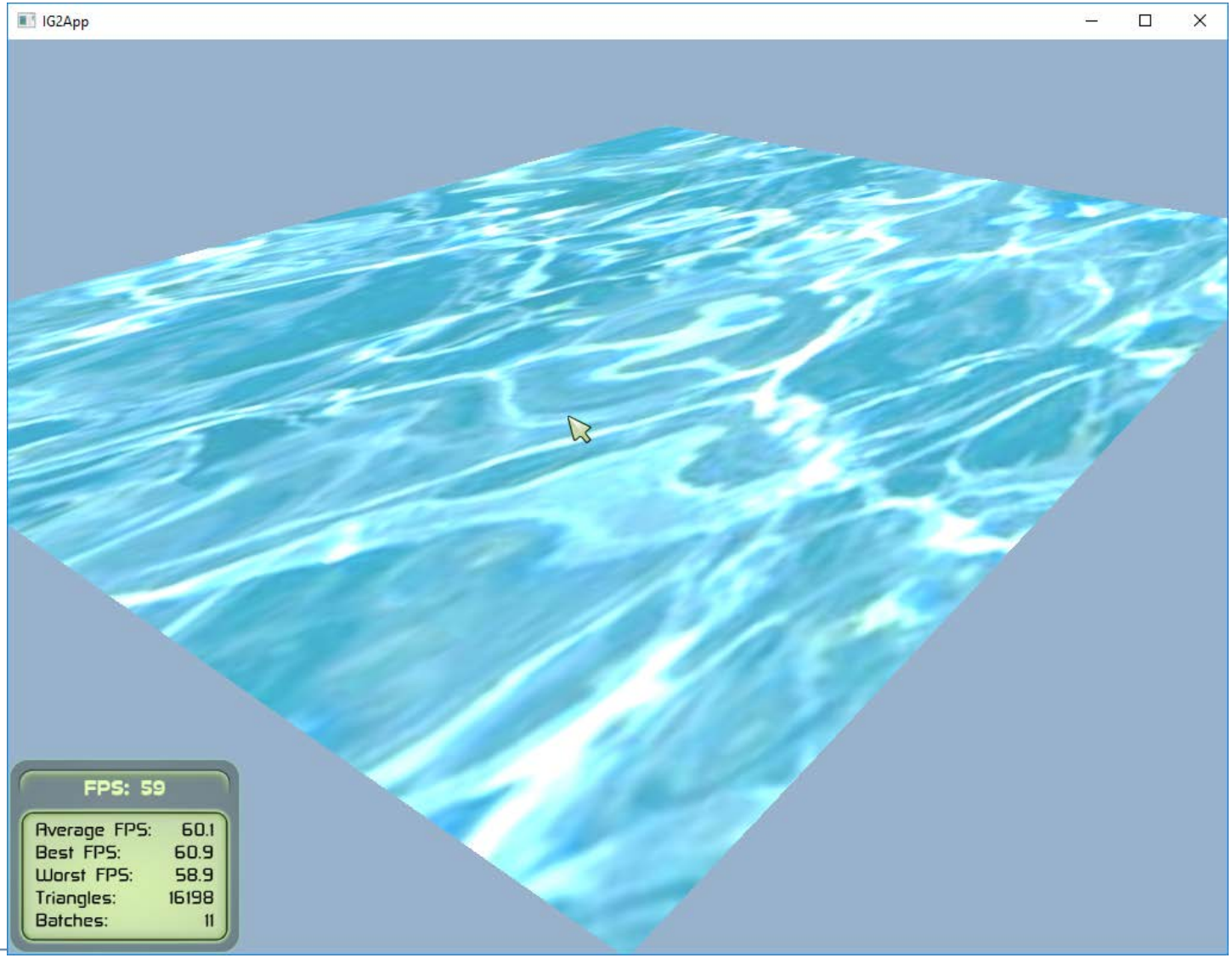
- Se pueden añadir a la escena mallas creadas (las .mesh) y mallas construidas
- Solo hay dos formas de mallas construidas: las superficies planas y las superficies curvadas
- Para **construir la malla del plano** se usa el commando (se muestra un ejemplo de uso):

```
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) y distancia desde el origen, anchura del plano en coordenadas mundiales, idem. altura, número de segmentos del plano en la dirección X, idem. Y, booleano (si true, crea normales perpendiculares al plano), número de conjuntos de coordenadas de textura 2D creadas, número de veces que se repite la textura en la dirección u, idem. v, orientación Up de la textura

Ejemplo de plano con textura



Entidades sobre mallas construidas

- Una vez construida la malla se puede **definir una entidad que la use** y se hace tal como es habitual:

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

- Y entonces 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);
```

Entidad Obj sin heredar de Entity

- **OgreInput:** OgreBites::InputListener

```
class Obj : public OgreBites::InputListener { //#include <OgreInput.h>
    public:
        Obj(Ogre::SceneNode* node, ...);
        ~Obj();
        // métodos de InputListener que podemos redefinir
        virtual bool keyPressed(const OgreBites::KeyboardEvent& evt);
        virtual bool keyReleased(const OgreBites::KeyboardEvent& evt);
        virtual void frameRendered(const Ogre::FrameEvent &  evt);
        virtual bool mousePressed(const OgreBites::MouseButtonEvent& evt);
        virtual bool mouseRelease(const OgreBites::MouseButtonEvent& evt);
        virtual bool mouseMoved(const OgreBites::MouseMotionEvent& evt);
        virtual bool mouseWheelRolled(const OgreBites::MouseWheelEvent& evt);
        ...
}
```

Entidad Obj sin heredar de Entity

Constructora

```
class Obj : public OgreBites::InputListener  {

        // #include <OgreSceneManager.h>
        // #include <OgreEntity.h>

        protected:

        Ogre::SceneNode* mNode;
        Ogre::SceneManager* mSM;
        ...

}
Obj::Obj(Ogre::SceneNode* node) : mNode(node) {

        Ogre::SceneManager* mSM = mNode->getCreator();

        ...

}
```

InputListener de la Entidad Obj

- virtual bool keyPressed(const OgreBites::KeyboardEvent& evt);

```
bool Obj::keyPressed(const OgreBites::KeyboardEvent& evt) {
    // #include <SDL_keycode.h>
    if (evt.keysym.sym == SDLK_??) {
            ...
    }
    return true;
}
void Obj::frameRendered(const Ogre::FrameEvent &  evt) {
    // evt.timeSinceLastFrame -> en segundos (float/double)
    // evt.timeSinceLastEvent  -> en segundos (float/double)
    ...
}
```

La entidad Obj en IG2App

```
void IG2App::setupScene(void) {
  ...
  nodoObjeto = ...;
  // Por ejemplo, nodo = mSM->getRootSceneNode()
  //                                ->createChildSceneNode("...");
  obj1 = new Obj1(nodoObjeto));
  addInputListener(obj1);
  ...
}
void IG2App::shutdown() {
  ...
  delete obj1;
}
```

EntidadIG.h

```
class EntidadIG :public OgreBites::InputListener {
        public:
                //Constructora y destructora
                EntidadIG(SceneNode* node, ...);
                ...
                //Vector estático de listeners
                static std::vector<EntidadIG*> appListeners;
                //Añadir entidad como listener al vector con push_back()
                static void addListener(EntidadIG* entidad) { ... };
        protected:
                Ogre::SceneNode* mNode;
                Ogre::SceneManager* mSM;

                virtual bool keyPressed(const OgreBites::KeyboardEvent& evt)
                        { return false; };
};
```

EntidadIG.cpp

```
// Inicialización del vector de listeners
std::vector<EntidadIG*> EntidadIG::appListeners =
    std::vector<EntidadIG*>(0, nullptr);

EntidadIG::EntidadIG(SceneNode *nodo, …) {
    mNode=nodo;
    mSM= mNode->getCreator();
}

EntidadIG::~EntidadIG() {
    …
}
```

frameRendered()

- El método debe añadirse a la clase **EntidadIG**

```
virtual void frameRendered(const Ogre::FrameEvent& evt) {};
```

e implementarse en aquellas clases que lo requieran.

- En la definición del método se puede usar el tiempo transcurrido desde el último frame renderizado:

```
Ogre::Real time = evt.timeSinceLastFrame;
```

para definir aquellos efectos que se necesiten. Por ejemplo, giros o desplazamientos en los que el espacio desplazado o el ángulo girado es función de este tiempo transcurrido:

```
Ogre::Real time = evt.timeSinceLastFrame;
distance = 80 * time;
mNode->translate(distance, 0, distance, Ogre::Node::TS_LOCAL);
```

Envío / recepción de eventos

- Para que una entidad pueda mandar eventos a todos los oyentes del array **`appListeners`**, se añaden dos métodos a la clase **`EntidadIG`**:

```
void sendEvent(EntidadIG* entidad);

virtual void receiveEvent(EntidadIG* entidad) {};
```

- Usualmente el envío de eventos es el mismo para todas las entidades y consiste en mandar el evento a todos los oyentes del array **`appListeners`**:

```
void EntidadIG::sendEvent(EntidadIG* entidad) {
    for (EntidadIG* e : appListeners)
        e->receiveEvent(this);
}
```

- La recepción de eventos se programa según se desee. Por ejemplo:

```
void Molino::receiveEvent(EntidadIG* entidad) {
    ...
}
```

Envío / recepción de eventos

- Para la Práctica 1, recuerda añadir las entidades de la escena (plano, molino, avión) como oyentes al array estático de oyentes
- Usa para ello el commando **`addListener()`**. Por ejemplo:

```
EntidadIG::addListener(molino);
```

- En la implementación del método **`receiveEvent(e)`** para ciertas entidades puede ser que no se use la entidad **`e`**, pero no tiene por qué ser ese siempre el caso.
- Una forma más general de paso de notificaciones entre entidades es:

```
void sendEvent(MessageType msgType, EntidadIG* entidad);
void receiveEvent(MessageType msgType, EntidadIG* entidad);
```

donde **`enum MessageType { ... };`** es un tipo enumerado

- De esta manera, una entidad puede actuar de diferente forma según sea el tipo del mensaje y la entidad que se lo manda

Envío / recepción de eventos

- El array de oyentes en esta escena incluye la noria, el muñeco y el plano

```
EntidadIG::addListener(muñeco);
...
```

- Al pulsar una tecla determinada, cada entidad reacciona de una manera