



**OGRE 3D**

**Skeletal Animation  
and  
SceneNode Animation**

**<http://www.ogre3d.org>**

Ana Gil Luezas  
Departamento de Sistemas Informáticos y Computación  
Universidad Complutense de Madrid

- ❑ Modificar el valor de uno o más parámetros de un objeto a lo largo de un periodo de tiempo. Algunos parámetros que se pueden modificar: posición, orientación, tamaño, color, coordenadas de textura.
- ❑ Para especificar cómo varían los valores con el tiempo se puede utilizar una función o un muestreo.

Muestreo. Secuencia (*track*) de instantáneas (*keyframes*): valores de los parámetros en distintos instantes de tiempo

$$Kf0 = \langle t0, v0 \rangle, Kf1 = \langle t1, v1 \rangle, \dots, Kfn = \langle tn, vn \rangle$$

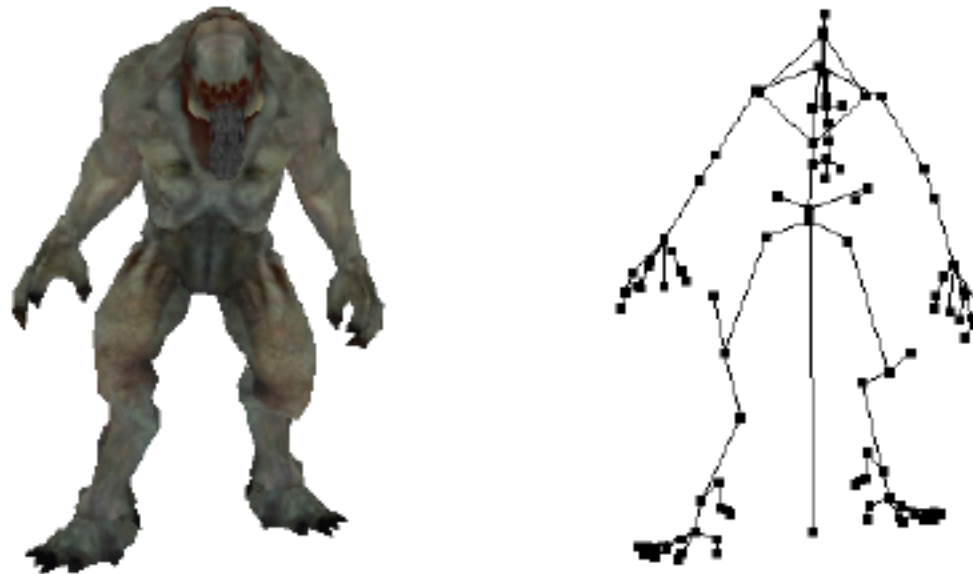
La información necesaria para cada instantánea depende del tipo de animación:

- ❑ Valor numérico
- ❑ Transformación (posición, escala y orientación)
- ❑ Malla

Los valores correspondientes a los puntos intermedios del tiempo se obtienen por interpolación de los valores vecinos

- ❑ En OGRE hay varias clases de animación:
  - ❑ **Numeric Value Animation**. Por ejemplo: intensidad de la luz
  - ❑ **SceneNode Animation**: modifica la posición, orientación y escala de los nodos de la escena. El valor es una transformación  
Transformación: Matriz4x4  
Transformación: posición (vec3), orientación (quaternion -> vec4) y escala (vec3 / float para escalas uniformes)
  - ❑ **Vertex Animation**: modifica los vértices de la malla. El valor es una malla. Hay dos subtipos: Morph y Pose (gestos faciales)
  - ❑ **Skeletal Animation**: El valor es una transformación sobre una articulación de un esqueleto ligado a una malla

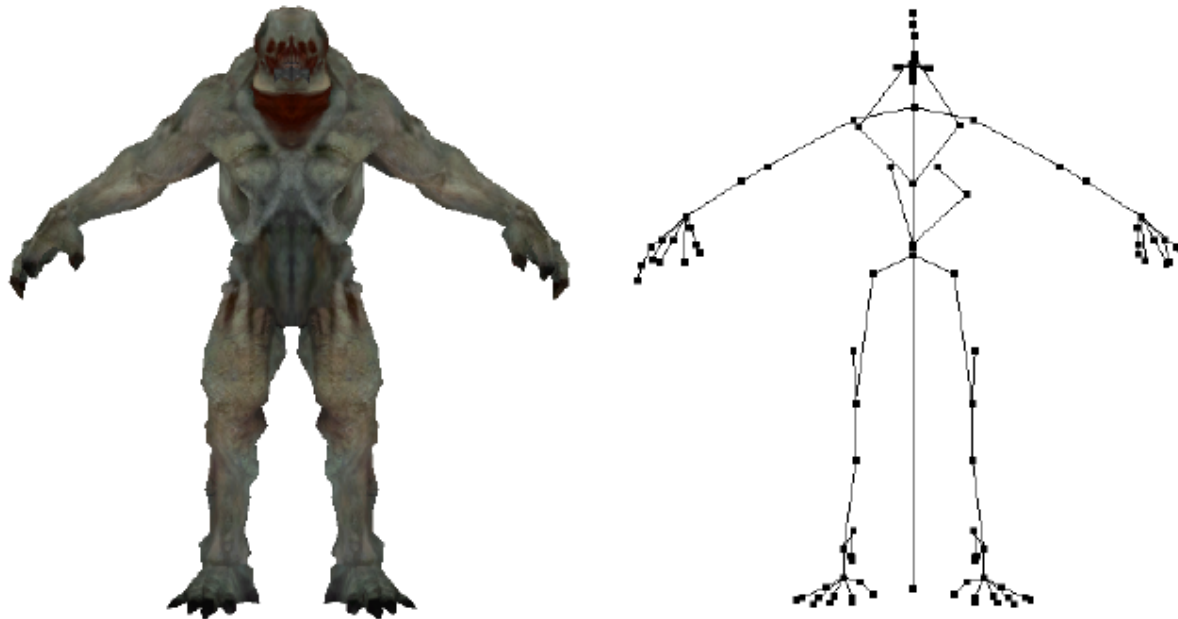
- ❑ **Skeletal trees:** Análogos al grafo de la escena, constan de una jerarquía de articulaciones (**joints / bones**) dadas por una posición, una rotación y una escala. La transformación de una articulación actúa sobre algunos vértices de la malla (los enganchados a la articulación)



Puedes ver, por ejemplo, [robot.skeleton.xml](#): `<bones>` y `<bonehierarchy>`

- ❑ **Skeletal meshes:** Además contienen información sobre la asociación entre vértices y articulaciones.

Al asociar una malla con su esqueleto (**rigging**) se elige una pose adecuada (**bind pose**). Las transformaciones asociadas son el punto de partida de las animaciones.

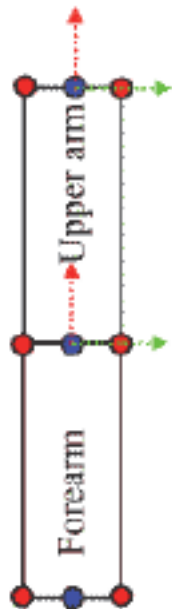


Puedes ver, por ejemplo, [robot.mesh.xml](#): `<boneassignments>`

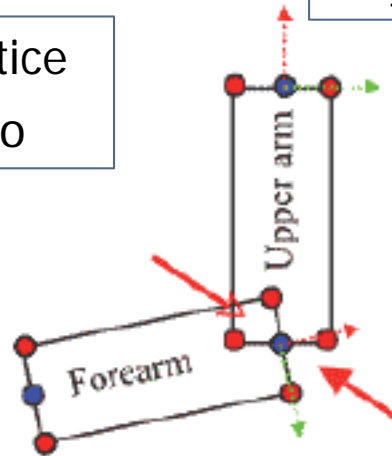
## □ Skeletal meshes: joints (bones) and weights

**Grafo de la escena:** Todos los vértices de la malla se adjuntan a un único nodo.

Todos los vértices de la malla se ven afectados por igual por la transformación del nodo.



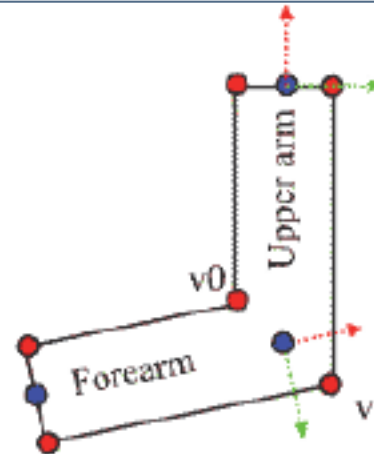
**Rojo:** vértice  
**Azul:** nodo



**Esqueletos:** Cada vértice de la malla se puede enganchar a varias articulaciones con distintos pesos.

Cada vértice de la malla se ve afectado por las transformaciones de las articulaciones a las que está enganchado.

Mayor coste en tiempo y espacio.



**Rojo:** vértice  
**Azul:** articulación

## □ Skeletal meshes: joints (bones) and weights

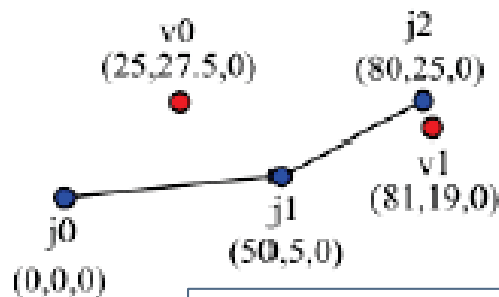
Position
Texture Coordinates
Normal
Bone ID0   Weight 0
Bone ID1   Weight 1
Bone ID2   Weight 2
Bone ID3   Weight 3

Información en la malla para cada vértice: posición, coordenadas de textura, normal, enganches a las articulaciones (articulación y peso).

$$\sum t(v).w$$

Coordenadas del vértice

Transformación y peso de la articulación



*Vertex v0*  
 Vertex pos: (0,25,0)  
 Vertex joint A: j0  
 Vertex weight A: 0.5  
 Vertex joint B: j1  
 Vertex weight B: 0.5

*Vertex v1*  
 Vertex pos: (10,0,0)  
 Vertex joint A: j1  
 Vertex weight A: 0.3  
 Vertex joint B: j2  
 Vertex weight B: 0.7

Rojo: vértice

Azul: articulación

$$v0: (0, 25, 0) + 0.5 (50, 5, 0) = (25, 27.5, 0)$$

$$v1: (10, 0, 0) + 0.3 j1 + 0.7 j2 = (81, 19, 0)$$

## ❑ Animating skeletal meshes (skinning)

Mover el esqueleto: Definir las animaciones del esqueleto

Cada animación consta de una secuencia de **key frames** con las transformaciones que se quieren aplicar al esqueleto

Puedes ver, por ejemplo, [robot.skeleton.xml](#): `<animations>`

Se pueden combinar varias animaciones para obtener animaciones compuestas:

Piernas corriendo + mover los brazos

Piernas corriendo + disparar

Para que se desplace hay que añadir una animación al nodo:  
**SceneNode Animation**

Intermediate tutorial 1 ([wiki.ogre3d.org](http://wiki.ogre3d.org))  
Chapter 9: Pro OGRE 3D Programming



- ❑ Una animación se define con un **AnimationState\*** **animationState** que puede tomar valor de varias formas:
  - ❑ **mSM->createAnimationState(name);** **//mSM** gestor de la escena de la clase
  - ❑ **entity->getAnimationState(name);** **//entity** se construye sobre una **mesh**  
si el modelo es una **mesh** que tiene asociado un esqueleto con sus propias animaciones (**name**) predefinidas (**Sinbad.mesh**, por ejemplo)
- ❑ Las animaciones se activan y se repiten mediante:  
**animationState-> setEnabled(true);**  
**animationState-> setLoop(true);**
- ❑ Para que una animación avance es necesario indicarle al gestor de la animación el tiempo transcurrido. Para ello se puede usar el método **addTime()** de **AnimationState**, en **frameRendered()**

```
void ...::frameRendered(const Ogre::FrameEvent & evt) {  
    animationState-> addTime(evt.timeSinceLastFrame);  
}
```

- ❑ Se puede averiguar el **name** de todas las animaciones predefinidas de una malla mediante el siguiente código:

```
AnimationStateSet * aux = ent->getAllAnimationStates();  
auto it = aux->getAnimationStateIterator().begin();  
while (it != aux->getAnimationStateIterator().end())  
    { auto s = it->first; ++it; }
```

- ❑ Algunos nombres de las animaciones de **Sinbad** son **Dance** (con el que Sinbad baila y saca la lengua), **RunBase y RunTop** (con el que Sinbad mueve las piernas y corre, mientras la parte de arriba mueve los brazos acompasadamente), ...
- ❑ Para que, además de mover el esqueleto, la entidad se desplace, se reoriente, se escale, ... hay que añadir animación al nodo con **SceneNode Animation**

- Podemos adjuntar una entidad “independiente” a una articulación de otra entidad con esqueleto mediante el siguiente comando

```
entity->attachObjectToBone("BoneName", MovableObject* );
```

Articulación del esqueleto

Entidad que se quiere enlazar

- Por ejemplo, para añadir espada al brazo derecho de Sinbad

```
entity->attachObjectToBone("Handle.R", sword);
```

donde **sword** es una entidad construida con **Sword.mesh**

- Los elementos se “desadjuntan” con

```
entity->detachObjectFromBone(MovableObject*);
```

- Para consultar el nombre de las articulaciones o huesos se utiliza

```
auto skeleton = mesh -> getSkeleton(); //entity -> getMesh()
```

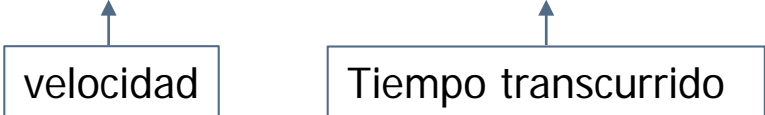
```
auto numBones = skeleton ->getNumBones();
```

```
for ( ... i ...) { skeleton -> getBone(i)->getName(); }
```

- ❑ Aplicando una transformación al nodo en función del tiempo, usando el método `frameRendered()` de `OgreBites::InputListener`.

Ejemplo:


```
void Obj::frameRendered(const FrameEvent & evt)
{
    mNode->yaw(Ogre::Degree(10 * evt.timeSinceLastFrame));
}
```



- ❑ **Muestreo**: Secuencia (*track*) de instantáneas (*keyframes*)

Kf=<time, value> // Los valores son transformaciones

Kf0=<0, v0>      Kf1=<t1, v1>      Kf2=<t2, v2>      Kf3=<duración\_total, v3>



Los valores correspondientes a los puntos intermedios del tiempo se obtienen por interpolación de los valores vecinos. El valor de cada keyframe se da con respecto al estado inicial

## SceneNode Animation (NodeAnimationTrack)

- ❑ Usamos un objeto de la clase **Animation** (creado mediante el método **createAnimation(name, duración)** de la clase **SceneManager**) para especificar caminos/secuencias (**tracks**), mediante el método **createNodeTrack(short)** (short es el número de camino)
- ❑ Los caminos son de la clase **AnimationTrack**, pero para animaciones de nodos usamos la subclase **NodeAnimationTrack**
- ❑ Un camino se define por una secuencia de puntos por los que pasa y el tiempo que se tarda en alcanzarlos

En una animación se pueden definir varios caminos. Todos tienen que tener la misma duración total, aunque no la misma longitud (número de keyframes)

Intermediate tutorial 1 ([wiki.ogre3d.org](http://wiki.ogre3d.org))

# SceneNode Animation (NodeAnimationTrack)

- ❑ Un camino es una secuencia de instantáneas **KeyFrames** y una instantánea es un objeto que guarda información de (instante de tiempo, valor(=transformación) asociado al instante)
- ❑ Los **keyFrames** son objetos de la clase **TransformKeyFrame** que es subclase de **KeyFrame**, y se crean mediante

**track->createNodeKeyFrame(time)**

donde **time** es el momento en que se toma la instantánea

- ❑ Los valores de las instantáneas asociadas a un tiempo determinado son transformaciones que se obtienen mediante los métodos **setTranslate**, **setRotation** y **setScale**

Intermediate tutorial 1 ([wiki.ogre3d.org](http://wiki.ogre3d.org))

# SceneNode Animation (NodeAnimationTrack)

## ❑ Ejemplo: desplazamiento vaivén arriba y abajo

```
Obj::Obj(Ogre::SceneNode* node) : mNode(node), mSM(node->getCreator()) {
```

```
    Entity * ent = mSM -> createEntity(...);
```

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

```
    ...
```

```
    Animation * animation = mSM -> createAnimation("animVV", duracion);
```

```
    NodeAnimationTrack * track = animation -> createNodeTrack(0);
```

```
    track -> setAssociatedNode(mNode);
```

```
    Vector3 keyframePos(-10., 0., 100.);
```

```
    Real durPaso = duracion / 4.0; // uniformes
```

```
    TransformKeyFrame * kf; // 5 keyFrames: origen(0), arriba, origen, abajo, origen(4)
```

```
    kf = track -> createNodeKeyFrame(durPaso * 0); // Keyframe 0: origen
```

```
    kf -> setTranslate(keyframePos); // Origen: Vector3
```

```
    ... // -> ->
```

```
    AnimationState * animationState = sceneMgr -> createAnimationState("animVV");
```

```
    animationState -> setLoop(true);
```

```
    animationState -> setEnabled(true);
```

```
}
```

Nombre

Duración total  
de la animación

Camino 0

Nodo

Instante de tiempo

Posición origen: inicial + traslación

para la animación ...

# SceneNode Animation (NodeAnimationTrack)

## ❑ Ejemplo (continuación): desplazamiento vaivén arriba y abajo

// 5 keyFrames: origen (KF0), arriba (KF1), origen (KF2), abajo (KF3), origen (KF4)

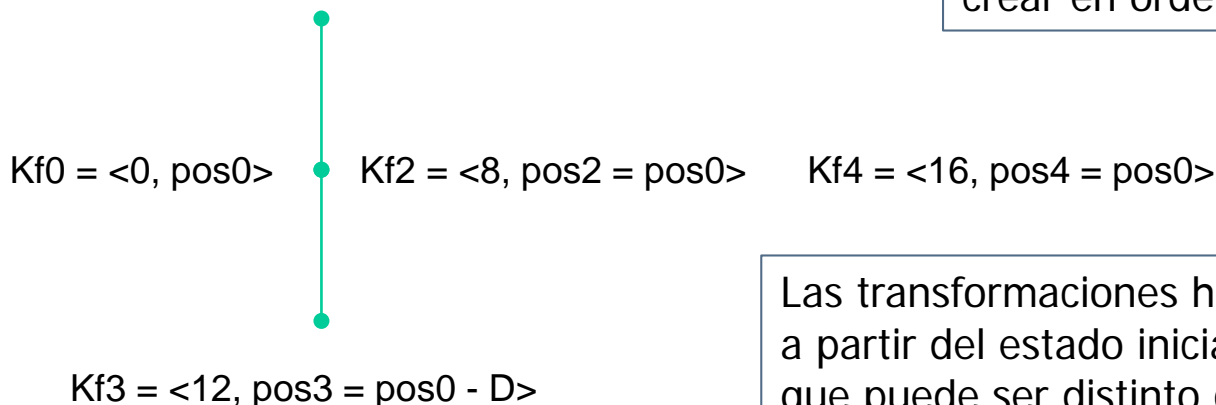
// duración total: 16 (duración = 16)

// duración entre un KF y el siguiente: 4 (durPaso = 4 = duracion / 4.)

// posición origen: pos0 = inicial + traslación

// longitud del vaivén: D (longDesplazamiento)

Kf1 = <4, pos1 = pos0 + D>



Los keyframes se deben crear en orden temporal

Las transformaciones hay que darlas a partir del estado inicial del nodo, que puede ser distinto de Kf0. El estado inicial de un nodo se puede fijar con **setInitialState** (por defecto es la identidad)



# SceneNode Animation (NodeAnimationTrack)

- ❑ Ejemplo (continuación): Hay que dar la transformación desde el estado inicial del nodo

```
// -> ->
```

```
kf = track-> createNodeKeyFrame(durPaso * 1); // Keyframe 1: arriba
```

```
keyframePos += Ogre::Vector3::UNIT_Y * longDesplazamiento;
```

```
kf-> setTranslate(keyframePos); // Arriba
```

```
// Keyframe 2: origen ....
```

```
kf = track-> createNodeKeyFrame(durPaso * 3); // Keyframe 3: abajo
```

```
keyframePos += Ogre::Vector3::NEGATIVE_UNIT_Y * longDesplazamiento;
```

```
kf-> setTranslate(keyframePos); // Abajo
```

```
kf = track-> createNodeKeyFrame(durPaso * 4); // Keyframe 4: origen
```

```
keyframePos += Ogre::Vector3::UNIT_Y * longDesplazamiento;
```

```
kf-> setTranslate(keyframePos); // Origen
```

# SceneNode Animation (NodeAnimationTrack)

- ❑ Recuerda: Para gestionar una animación hay que crear un **AnimationState**, con el método **createAnimationState**
- ❑ Recuerda añadir el tiempo transcurrido al gestor de la animación

```
void ...::frameRendered(const Ogre::FrameEvent & evt) {  
    animationState->addTime(evt.timeSinceLastFrame);  
}
```

- ❑ Para configurar el tipo de interpolación entre keyframes (por defecto **IM\_LINEAR**)

```
animation->setInterpolationMode(Ogre::Animation::IM_SPLINE);
```

- ❑ Para especificar el estado inicial del nodo a partir del cual se dan las transformaciones: **mNode->setInitialState()**; fija la transformación del nodo como estado inicial.

## ❑ Orientación de un objeto en 3D (se identifica con el eje Z)

Para las animaciones tenemos que usar cuaterniones para `setRotation`

## ❑ Ángulos de Euler. Toda rotación se puede establecer con los tres giros básicos (no de forma única). Problemas: Gimbal lock e interpolación

Yaw( $\beta$ )

`glRotatef( $\beta$ ,0,1,0)`

Pitch( $\alpha$ )

`glRotatef( $\alpha$ ,1,0,0)`

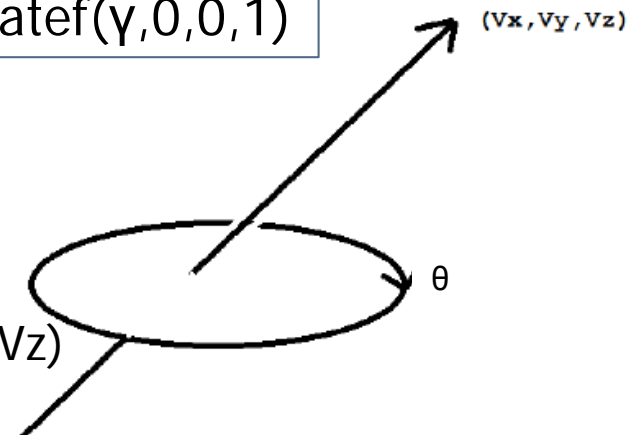
Roll( $\gamma$ )

`glRotatef( $\gamma$ ,0,0,1)`

## ❑ Rotación sobre un eje genérico:

Vector normalizado ( $V_x, V_y, V_z$ )

Ángulo de rotación ( $\theta$ )



## ❑ Matriz 3x3 de rotación. `glRotatef( $\theta, V_x, V_y, V_z$ )`

Problemas: interpolación

## ❑ Cuaterniones. La información se guarda en un `vec4`:

$$q = (w, x, y, z) = (\cos(\theta/2), V_x * \sin(\theta/2), V_y * \sin(\theta/2), V_z * \sin(\theta/2))$$

quaternion (4 valores)  $\leftrightarrow$  matriz de rotación 3x3 (9 valores)

Resuelve el problema de la interpolación de orientaciones

## ❑ Cuaterniones

❑ Eje de rotación genérico normalizado: vector  $V = (V_x, V_y, V_z)$

❑ Ángulo de rotación sobre el eje:  $\theta$

Esta información se guarda en la forma de **cuaternión unitario**:

$$(w, x, y, z) = (\cos(\theta/2), V_x * \sin(\theta/2), V_y * \sin(\theta/2), V_z * \sin(\theta/2))$$
$$= q = \cos(\theta/2) + V * \sin(\theta/2)$$

❑ Sea  $P=(x, y, z)$  un punto, el punto rotado  $\theta$  grados sobre el eje  $V$  se obtiene mediante

$$q P q^{-1} = q P q^*$$

La inversa de un cuaternión unitario es su conjugado

$$q^{-1} = q^* = \cos(\theta/2) - V * \sin(\theta/2)$$

- ❑ **Composición de rotaciones:** se corresponde con el producto de cuaterniones.

Análogo al producto de matrices de rotación: Asociativo y no conmutativo.

## ❑ Cuaterniones en OGRE

```
Ogre::Vector3 src(0, 1, 1);   Ogre::Vector3 dest(-1, 1, 0);  
//quaternion para rotar de src a dest (ángulo menor)  
Ogre::Quaternion quat = src.getRotationTo(dest);  
//quaternion para pich(90) en forma de ángulo y eje de rotación  
Quaternion q1 = Quaternion(Degree(90.0), Vector3(1, 0, 0));  
//R5 = sqrt(0.5) para pich(90) en forma de cuaternión unitario  
Quaternion q2 = Quaternion(R5, R5, 0, 0);  
Quaternion qp = q1 * q2; //pich(180)  
Quaternion qm = Quaternion(Matrix3);
```

- ❑ Ejemplos de cuaterniones: vector ( $V_x, V_y, V_z$ ) y ángulo ( $\theta$ )

Se guarda en un Vec4 con la siguiente información:

$$(w, x, y, z) = (\cos(\theta/2), V_x * \sin(\theta/2), V_y * \sin(\theta/2), V_z * \sin(\theta/2))$$

$$\begin{aligned}\cos(0) &= 1 \\ \sin(0) &= 0\end{aligned}$$

$$\begin{aligned}\cos(90) &= 0 \\ \sin(90) &= 1\end{aligned}$$

$$\begin{aligned}\cos(45) &= \sqrt{0.5} \\ \sin(45) &= \sqrt{0.5}\end{aligned}$$

w	x	y	z	Description
1	0	0	0	Identity quaternion, no rotation
0	1	0	0	180° turn around X axis
0	0	1	0	180° turn around Y axis
0	0	0	1	180° turn around Z axis
sqrt(0.5)	sqrt(0.5)	0	0	90° rotation around X axis
sqrt(0.5)	0	sqrt(0.5)	0	90° rotation around Y axis
sqrt(0.5)	0	0	sqrt(0.5)	90° rotation around Z axis
sqrt(0.5)	-sqrt(0.5)	0	0	-90° rotation around X axis
sqrt(0.5)	0	-sqrt(0.5)	0	-90° rotation around Y axis
sqrt(0.5)	0	0	-sqrt(0.5)	-90° rotation around Z axis

## setRotation(Quaternion)

- ❑ En Ogre podemos obtener el cuaternión necesario para rotar un vector (**src**) y llevarlo a un vector destino (**dest**) usando el método **getRotationTo()**. Por ejemplo:

```
Ogre::Vector3 src(0, 1, 1);
```

```
Ogre::Vector3 dest(-1, 1, 0);
```

```
//quaternion para rotar de src a dest (por el ángulo menor)
```

```
Ogre::Quaternion quat = src.getRotationTo(dest);
```

```
keyFrame -> setRotation(quat);
```

- ❑ Para configurar el tipo de interpolación entre keyframes (por defecto **RIM\_LINEAR**)

```
animation->setRotationInterpolationMode(Ogre::Animation::RIM_SPHERICAL);
```

- ❑ También se pueden usar cuaterniones para las rotaciones de los nodos

```
node -> rotate(quat);
```

```
node -> setOrientation(quat);
```

# SceneNode Animation (NodeAnimationTrack)

❑ Ejemplo: desplazamiento vaivén abajo y arriba, girando  $\pm 45^\circ$  en el eje Y

// 4 keyFrames: origen (Kf0), abajo (Kf1), arriba (Kf2), origen (Kf3)

// duración total: 16

// duración entre un Kf y el siguiente: no uniforme -> 0, 4, 12, 16 (durPaso = 4)

// posición y orientación iniciales: pos0, orientación0

// longitud del vaivén: D (longDesplazamiento)

Kf2 = <12, pos2 = pos0 + D = pos1 + 2D,  
orientación2 = Yaw(-45)>

Los keyframes se deben crear en orden temporal

Kf0 = <0, pos0,  
orientación0>

Kf3 = <16, pos3 = pos0,  
orientación3 = orientación0>

Kf1 = <4, pos1 = pos0 - D,  
orientación1 = Yaw(45)>

Las transformaciones hay que darlas a partir del estado inicial del nodo, que puede ser distinto de Kf0. El estado inicial de un nodo se puede fijar con **setInitialState** (por defecto es la identidad)



# SceneNode Animation (NodeAnimationTrack)

❑ Ejemplo: Hay que dar la transformación desde el estado inicial del nodo

```
Vector3 keyframePos(0.0); Vector3 src(0, 0, 1); // posición y orientación iniciales
TransformKeyFrame * kf; // 4 keyFrames: origen(0), abajo, arriba, origen(3)
kf = track -> createNodeKeyFrame(durPaso * 0); // Keyframe 0: origen
kf = track -> createNodeKeyFrame(durPaso * 1); // Keyframe 1: abajo
keyframePos += Ogre::Vector3::NEGATIVE_UNIT_Y * longDesplazamiento;
kf-> setTranslate(keyframePos); // Abajo
kf-> setRotation(src.getRotationTo(Vector3(1, 0, 1))); // Yaw(45)
kf = track-> createNodeKeyFrame(durPaso * 3); // Keyframe 2: arriba
keyframePos += Ogre::Vector3::UNIT_Y * longDesplazamiento * 2;
kf-> setTranslate(keyframePos); // Arriba
kf-> setRotation(src.getRotationTo(Vector3(-1, 0, 1))); // Yaw(-45)
kf = track-> createNodeKeyFrame(durPaso * 4); // Keyframe 3: origen
```