VNIVERSITAT DE VALÈNCIA

Grado en Ingeniería Multimedia

Memoria del Proyecto Final

Gráficos Avanzados Y Sonido

Autor:
Javier Pascual Roca

Índice

1. Problemas y su resolución

Manijas

El primer problema aparece a la hora de diseñar las manijas del reloj Analógico. Se pedía tener una parte del modelo cuya geometría se hubiera definido explícitamente, y se decidió que sería esta.

La función encargada de dibujar estas manijas es la siguiente:

```
osg::Node* createNeedle( float w, float h, float depth, const osg::Vec4& color, float angle, double period )
```

Y se trata de una adaptación a nuestras necesidades de la función que se puede encontrar en el libro OpenSceneGraph 3 Cookbook, referenciado como [1] en la bibliografía.

En esta función, se define una serie de vértices en función de los parámetros de anchura y altura (w, h), se les asigna la normal y el color definido en la entrada (osg::Vec4& color). Mediante el elemento osg::Geometry, se hace uso de dichas normales y vértices, y se construye un osg::Geode acorde a las necesidades.

Para el movimiento se hace uso de osg::AnimationPath, se definen una serie de puntos de control y en función del período introducido por parámetro, el Callback osg::AnimationPathCallback generará la transición entre ellos.

Dígitos

El siguiente problema que aparece es la generación de los dígitos de 7 segmentos. Para su generación, se ha creado la siguiente función:

```
osg::Node* createDigit(float displacementH, float displacementV, int digit, bool big)
```

Parámetros de entrada:

- displacementH y displacementV definen la posición del dígito en los ejes.

- Digit representa el número que se quiere dibujar en pantalla

- Big es una bandera que indica si el dígito es grande, para los minutos y horas, o pequeño, para los segundos.

Mediante una sentencia Switch, se “descompone” el número introducido en el parámetro digit en la cadena de caracteres que simbolizan cada uno de los segmentos que lo conformarán. Previamente se ha establecido un orden arbitrario para estos:

```
0 Arriba, 1 Medio, 2 Abajo, 3 Arriba Derecha, 4 Abajo Derecha, 5 Arriba Izquierda, 6 Abajo Izquierda
```

Después, se recorrerán los caracteres de esta cadena, y en base a ellos se definirá la posición y la orientación de cada uno de los segmentos que se generen

Dinamismo

El último problema hallado gira en torno al cambio dinámico de los dígitos en función del tiempo del sistema.

Esto probablemente no habría supuesto un problema de haber optado por la aproximación sugerida en prácticas, generar 12 nodos dígito e ir cambiando entre unos y otros mediante distintos nodos Switch, para tener controladas las horas, los minutos y los segundos, pero debido a una cierta ignorancia, generar 12 nodos y utilizar sólo 6 en cada momento pareció una idea un tanto descabellada, y se optó por el método que sigue, pero no sin la ayuda de Manolo Pérez Aixendri, profesor de prácticas de Programación Hipermedia.

La solución adoptada comienza con la definición de 6 matrices de transformación de las cuales "colgarán" los dígitos.

```
osg::ref_ptr<osg::MatrixTransform> DigitoHora1;
osg::ref_ptr<osg::MatrixTransform> DigitoHora2;
osg::ref_ptr<osg::MatrixTransform> DigitoMinuto1;
osg::ref_ptr<osg::MatrixTransform> DigitoMinuto2;
osg::ref_ptr<osg::MatrixTransform> DigitoSegundo1;
osg::ref_ptr<osg::MatrixTransform> DigitoSegundo2;
```

En el Main, lo primero que hacemos es inicializarlas:

```
DigitoHora1 = new osg::MatrixTransform;
DigitoHora2 = new osg::MatrixTransform;
DigitoMinuto1 = new osg::MatrixTransform;
DigitoMinuto2 = new osg::MatrixTransform;
DigitoSegundo1 = new osg::MatrixTransform;
DigitoSegundo2 = new osg::MatrixTransform;
```

Una vez hecho esto, se realiza una llamada a la función createDigital, que las recibe por parámetro:

```
osg::Node* digital = createDigital(DigitoHora1, DigitoHora2, DigitoMinuto1, DigitoMinuto2, DigitoSegundo1, DigitoSegundo2);
```

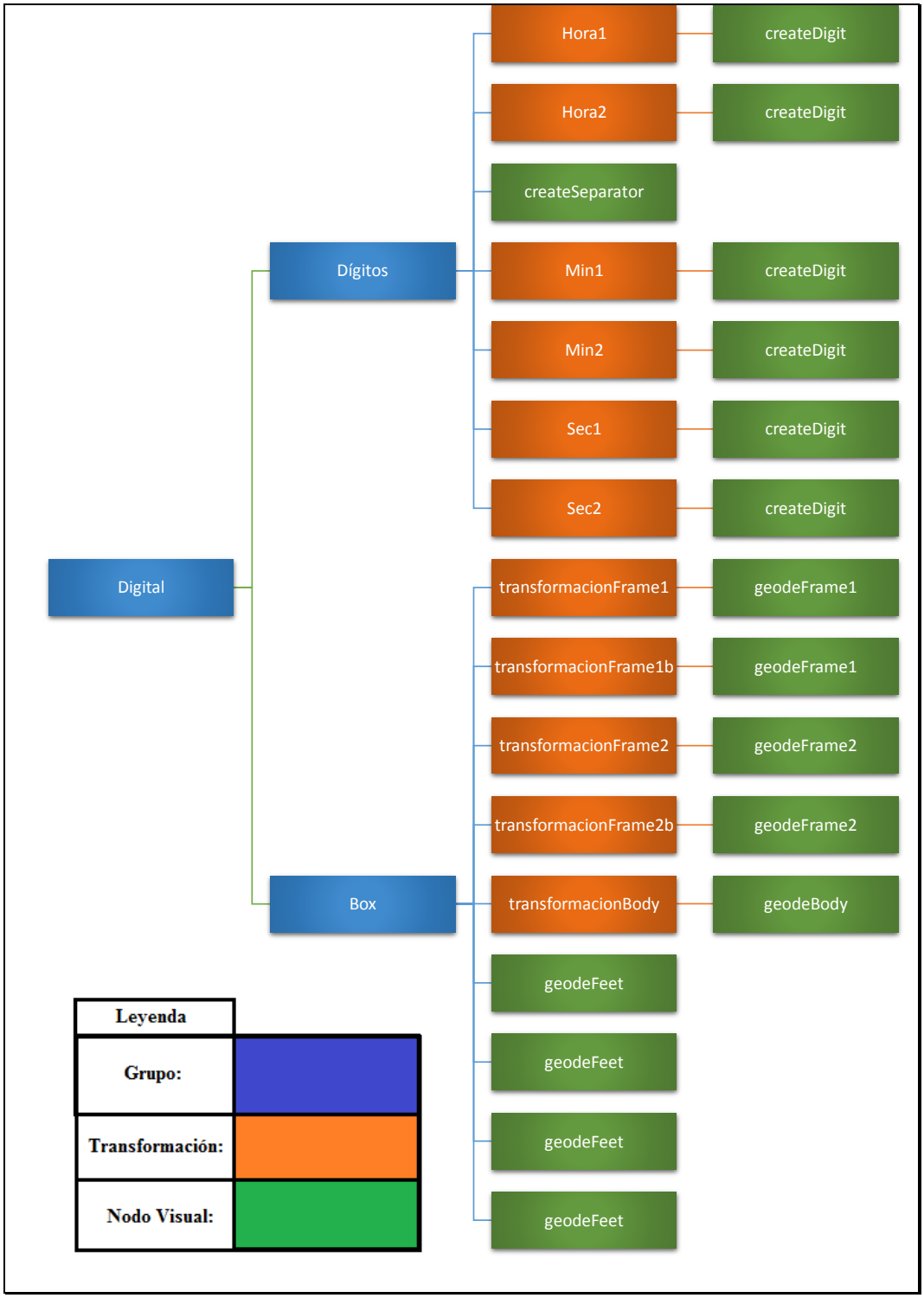
En dicha función, a cada una se le asigna su Callback correspondiente:

```
Hora1->addUpdateCallback(new UpdateTime(-2.6));
Hora2->addUpdateCallback(new UpdateTime(-1));
Min1->addUpdateCallback(new UpdateTime(1));
Min2->addUpdateCallback(new UpdateTime(2.7));
Sec1->addUpdateCallback(new UpdateTime(4.3));
Sec2->addUpdateCallback(new UpdateTime(5));
```

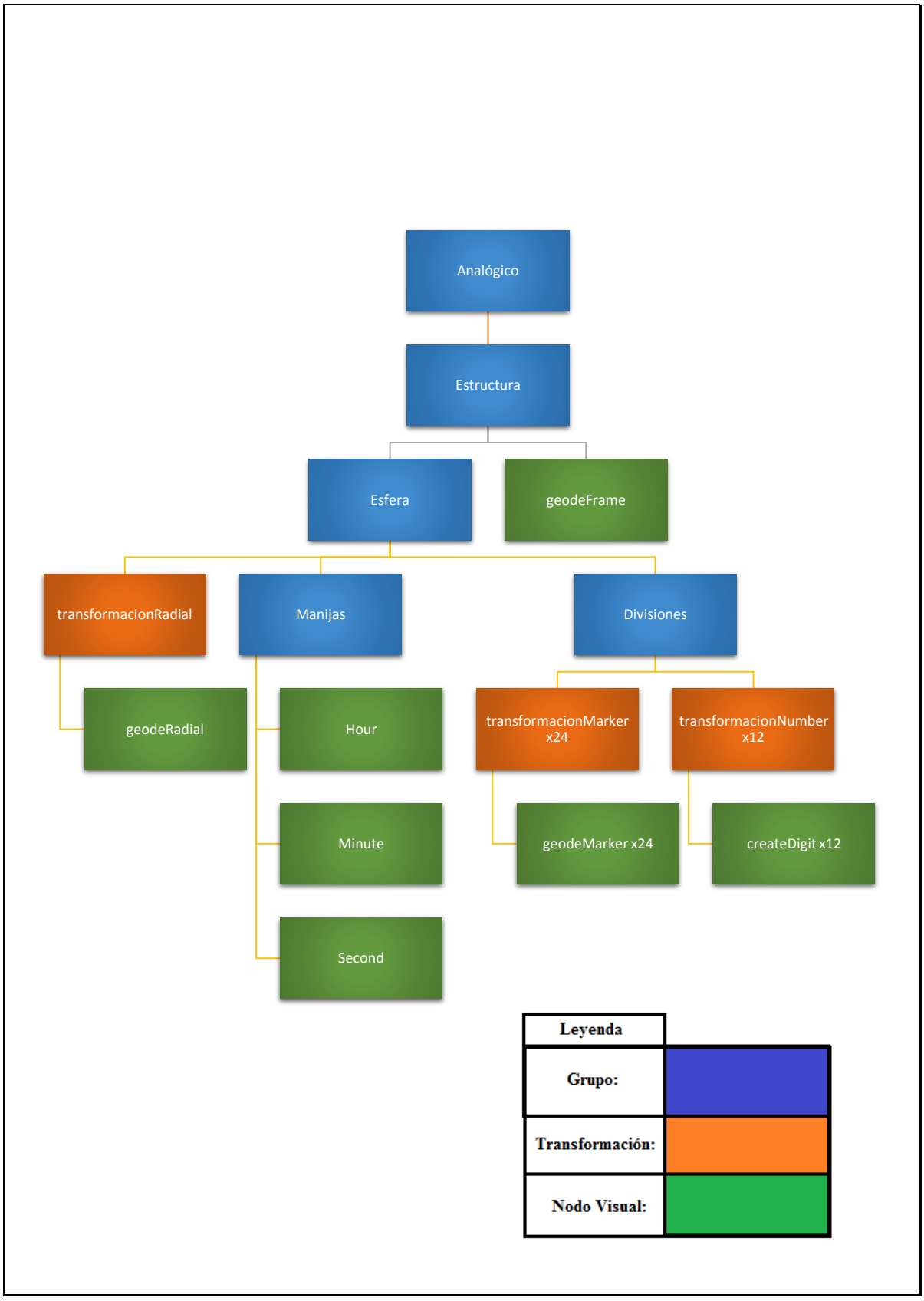
Del funcionamiento del Callback se hablará en el apartado dedicado a dicho fin, pero la idea básica consiste en la generación y destrucción dinámica de los nodos, para mantenerlos actualizados.

2. Grafo de escena

Reloj Digital:



Reloj Analógico:



3. Funciones de Callback

AnimationPathCallback

La primera función de Callback que se define en el código es AnimationPathCallback:

```
osg::ref_ptr<osg::AnimationPathCallback> apcb = new osg::AnimationPathCallback;
```

Esta función será la encargada de gestionar la animación de las manijas del reloj analógico. Mediante setAnimationPath, le asignamos los puntos de control y el período del movimiento definidos previamente en la función createNeedle, y le asignamos este Callback a la matriz de transformación de la que cuelga la manija:

```
//Transformación y movimiento
osg::ref_ptr<osg::MatrixTransform> trans = new osg::MatrixTransform;
trans->addChild( geode.get() );

osg::ref_ptr<osg::AnimationPath> clockPath = new osg::AnimationPath;
clockPath->setLoopMode( osg::AnimationPath::LOOP );

clockPath->insert( 0.0, osg::AnimationPath::ControlPoint(
osg::Vec3(0.0f, depth, 0.0f), osg::Quat(angle, osg::Y_AXIS)) );

clockPath->insert( period*0.5, osg::AnimationPath::ControlPoint(
osg::Vec3(0.0f, depth, 0.0f), osg::Quat(angle+osg::PI,osg::Y_AXIS)) );

clockPath->insert( period, osg::AnimationPath::ControlPoint(
osg::Vec3(0.0f, depth, 0.0f), osg::Quat(angle+osg::PI*2.0f, osg::Y_AXIS)) );

//Callback
osg::ref_ptr<osg::AnimationPathCallback> apcb = new osg::AnimationPathCallback;
apcb->setAnimationPath( clockPath.get() );
trans->addUpdateCallback( apcb.get() );
```

Update Time

Update Time es la clase que servirá de Callback para los dígitos del reloj Digital. A la hora de su instanciación, se le pasa como parámetro un float que servirá para discriminar de qué dígito estamos hablando en cada momento.

Tras obtener la hora del sistema, la primera tarea del Callback es la de deshacerse de todos los hijos que el nodo pueda tener:

```
SYSTEMTIME st;
GetLocalTime(&st);
float hour_time = st.wHour;
float min_time = st.wMinute;
float sec_time = st.wSecond;
osg::Group *nood=(osg::Group *)node;
nood->removeChildren(0,nood->getNumChildren());
```

A continuación se emplea el discriminador para crear el hijo correcto, en el lugar adecuado:

```
int h = (int)displacementH;
switch (h){
    case -2:
        nood->addChild(createDigit(-2.6, 0, floor(hour_time/10), true));
        break;
    case -1:
        nood->addChild(createDigit(-1, 0, hour_time-(floor(hour_time/10))*10, true));
        break;
    case 1:
        nood->addChild(createDigit(1, 0, floor(min_time/10), true));
        break;
    case 2:
        nood->addChild(createDigit(2.7, 0, min_time-(floor(min_time/10))*10, true));
        break;
    case 4:
        nood->addChild(createDigit(4.3, -.5, floor(sec_time/10), false));
        break;
    case 5:
        nood->addChild(createDigit(5, -.5, sec_time-(floor(sec_time/10))*10, false));
        break;
    default:
        break;
}
```

GUIEventHandler

ModelController, por su parte, se encarga de los eventos de teclado, y será quien informará al nodo Switch de la escena de qué elemento debe mostrar en cada momento.

En el momento en que se genere un evento de teclado, se comprobará si lo ha generado la tecla C, en cuyo caso se procederá a mostrar el siguiente nodo de la lista de hijos del nodo Switch.

```
switch(ea.getEventType())
{
case(osgGA::GUIEventAdapter::KEYDOWN):
    {
        switch(ea.getKey())
        {
            case 'c': case 'C':
                if(switcher < 4){
                    switcher++;
                    root->setValue(switcher - 1, false);
                    root->setValue(switcher, true);
                }else{
                    switcher = 0;
                    root->setValue(0, true);
                    root->setValue(4, false);
                }
                break;
             default:
                 break;
        }
    }
default:
    return false;
}
```

4. Bibliografía

[1] Wang, Rui; Qian, Xuelei "OpenSceneGraph 3 Cookbook". Ahux.narod.ru

[2] OSG Community "Changing Models Using Update Callbacks". Openscenegraph.org

[3] Wang, Rui; Qian, Xuelei "OpenSceneGraph 3.0 Beginner's Guide" Ahux.narod.ru