Tutorial de marco

Resumen

Para mas información vea tutoriales básicos, creación básica de objetos y trabajo con relojes.

Las propiedades de los objetos como: posiciones, escalas y rotación son almacenadas en las estructura `orxFrame`.
Estos marcos estan juntos en la jerarquia de gráficos, lo que significa que las propiedades del marco padre afectarán las propiedades de sus hijos.

En este tutorial, tenemos 4 objetos que serán vinculados a un padre en común [1] y un quinto sin padre.
Los dos primeros hijos son creados usando las propiedades de objetos de la configuración `ChildList` mientras que los otros dos son creados y vínculados desde el código (por propositos didácticos).
El objeto padre invisible seguira el puntero del ratón. Las teclas “SHIFT IZQUIERDA” y “CONTROL IZQUIERDA” cambiarán la escala arriba y abajo del objeto padre, mientras que el click “IZQUIERDO” y “DERECHO” lo harán rotar.
Todas estas transformaciones afectaran a los 4 hijos.

Esto nos muestra una forma fácil de crear grupos de objetos complejos y transformar sus propiedades (posición, escala, rotación, vélocidad...) fácilmente.

Detalles

Como en el tutorial anterior, comenzamos cargando nuestro fichero de configuración y creando la vista.

```
orxConfig_Load("../03_Frame.ini");

orxViewport_CreateFromConfig("Viewport");
```

Luego creamos nuestro objeto padre.

```
pstParentObject = orxObject_CreateFromConfig("ParentObject");
```

Definimos en el fichero de configuración `ParentObject` para nuestro objeto padre

```
[ParentObject]
ChildList = Object3 # Object4
```

Así cuando creamos nuestro objeto padre, los dos hijos [2] son además *automagicamente* creados y vínculados.
Esto podiamos haberlo hecho en el fichero de configuración, pero para propositos de aprendizaje, crearemos y vincularemos los otros dos objetos en el código.

```
orxOBJECT *pstObject;
orxObject_CreateFromConfig("Object0");
pstObject = orxObject_CreateFromConfig("Object1");
orxObject_SetParent(pstObject, pstParentObject);
pstObject = orxObject_CreateFromConfig("Object2");
orxObject_SetParent(pstObject, pstParentObject);
```

El `Object0` es nuestro objeto estático: el único que no estará vinculado a nuestro `ParentObject` (Objeto padre).

Note que cuando creamos y vínculamos manualmente los objetos en el código, es nuestro deber borrarlos luego. En el caso del `Object3` y `Object4` serán automáticamente eliminados cuando `ParentObject` sea eliminado.

A continuación, creamos un reloj de 100 Hz y registramos nuestra función `Update`. En esta función es donde vamos a gestionar las entradas para escalar / rotar el `ParentObject` y asegurarnos de que va a seguir a nuestro cursor del ratón.

```
pstClock = orxClock_Create(orx2F(0.01f), orxCLOCK_TYPE_USER);

orxClock_Register(pstClock, Update, orxNULL, orxMODULE_ID_MAIN,
orxCLOCK_PRIORITY_NORMAL);
```

Ahora vamos a echar un vistazo a nuestra función `Update`.
En primer lugar, nos aseguramos de que podemos encontrar la posición en nuestro espacio del mundo que corresponde a nuestro cursor del ratón en el espacio de la pantalla.
A continuación, copiamos nuestra coordenada Z de `ParentObject` (es decir, mantenemos la misma profundidad que antes) sobre el mismo y finalmente la establecemos en nuestro `ParentObject`.

```
if(orxRender_GetWorldPosition(orxMouse_GetPosition(&vPosition), &vPosition))
{
  orxVECTOR vParentPosition;
  orxObject_GetWorldPosition(pstParentObject, &vParentPosition);
  vPosition.fZ = vParentPosition.fZ;
  orxObject_SetPosition(pstParentObject, &vPosition);
}
```

The only thing left to do is to apply scale and rotation according to our inputs.
In our case, we defined the following inputs in 03_Frame.ini: `RotateLeft`, `RotateRight`, `ScaleUp` and `ScaleDown`.

Let's see how we handle them. First, the rotations.

```
if(orxInput_IsActive("RotateLeft"))
{
  orxObject_SetRotation(pstParentObject,
orxObject_GetRotation(pstParentObject) + orx2F(-4.0f) * _pstClockInfo->fDT);
}
if(orxInput_IsActive("RotateRight"))
{
```

```
  orxObject_SetRotation(pstParentObject,
orxObject_GetRotation(pstParentObject) + orx2F(4.0f) * _pstClockInfo->fDT);
}
```

And now, the scales.

```
if(orxInput_IsActive("ScaleUp"))
{
  orxObject_SetScale(pstParentObject, orxVector_Mulf(&vScale,
orxObject_GetScale(pstParentObject, &vScale), orx2F(1.02f)));
}
if(orxInput_IsActive("ScaleDown"))
{
  orxObject_SetScale(pstParentObject, orxVector_Mulf(&vScale,
orxObject_GetScale(pstParentObject, &vScale), orx2F(0.98f)));
}
```

That's all!  Our `ParentObject` will be updated and all his children with it.

NB:

- *We could have used config values instead of constants for the rotation and scale values. This way, we could change them without having to recompile and even update them in real time by pressing* `BackSpace`. [3)]
- *As we use the clock's DT for the rotations, they will benefit from time consistency [4)]. Unfortunately, that won't be the case for the scales. (Which is usually something we really don't want!)*

Recursos

Código fuente: 03_Frame.c

Fichero de configuración: 03_Frame.ini

1)
un objeto vacío, sin contenido visual
2)
`Object3` y `Object4`
3)
default key in orx's launcher for config reload
4)
they won't depend on frame rate and they will be time-stretchable