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 rotacion son almacenadas en las estructura `orxFrame`.
Estos marcos estan juntos en la jerarquia de gráficos, lo que significa que las propiedades del marco padre afectaran las propiedades de sus hijos.

En este tutorial, tenemos 4 objetos que serán vinculados a un padre en comun [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 vinculados desde el código (para propositos didácticos).
El objeto padre invisible seguira el puntero del ratón. Las teclas “SHIFT IZQUIERDA” y “CONTROL IZQUIERDA” cambiaran 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 mostrará una forma fácil de crear grupos de objetos complejos y transformar sus propiedades (posicion, escala, rotación, vélocidad...) facilmente.

Detalles

As with the previous tutorials, we begin by loading our config file and creating a viewport.

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

orxViewport_CreateFromConfig("Viewport");
```

We then create our parent object.

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

As in the config file, for our `ParentObject` we defined:

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

Thus when we create our parent object, those two children [2] have also been *automagically* created and linked.
We could have done so for all the four children, but, for a learning purpose, we'll create the two remaining children in code and link them manually.

```
orxOBJECT *pstObject;
```

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

Here, `Object0` is our static object: the only one that won't be linked to our `ParentObject`.

Please note that when we create and link manually objects in code, it's our responsability to delete them. On the contrary, `Object3` and `Object4` will be automatically deleted when `ParentObject` will be deleted.

We then create a 100Hz clock and register our `Update` function to it. This function is where we'll manage the inputs to scale/rotate the `ParentObject` and make sure it'll follow our mouse cursor.

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

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

Let's now have a look to our `Update` function.
First, we make sure we can find the position in our world space that corresponds to our mouse cursor in the screen space.
We then copy our `ParentObject` Z coordinate (ie. we keep the same depth as before) over it and we finally set it back on our `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 vacio, sin contenido visual
2)
`Object3` and `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

From:
https://www.orx-project.org/wiki/ - **Orx Learning**

Permanent link:
https://www.orx-project.org/wiki/es/orx/tutorials/frame?rev=1252435341

Last update: **2025/09/30 17:26 (14 months ago)**